ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA (E.T.S.)

Máster Universitario en Ingeniería Informática

Sistemas Neuromórficos
Investigación sobre sistemas FPGA

Autor
Adrián Gil Andrés

Curso: 2021-2022 - Convocatoria: Septiembre

Tutor UNED: Dr. Agustín Carlos Caminero Herráez
Tutor UNED: Dr. Antonio Robles Gómez
Tutor Externo: Dr. Javier Sedano Franco

Burgos
28 de septiembre de 2022

GIL ANDRES ADRIAN - 72892509N

Firmado digitalmente por:GIL ANDRES ADRIAN - 72892509N
Fecha: 2022-09-28 18:21:50 +0200

Resumen

Se ha realizado un estudio sobre diferentes sistemas y herramientas existentes, tanto de software como de hardware, que se pueden utilizar con las FPGAs para cubrir las necesidades de aceleración requeridos para la automatización de sistemas inteligentes mediante el uso de modelos de redes neuronales.

Sobre este estudio, se ha realizado una fase de desarrollo que se ha dividido en 3 lineas diferentes. Por un lado se han utilizado FPGAs de la serie ZUS+ (*Xilinx Ultrascale*), específicamente las Ultra96-V2, para la creación de un sistema FPGA as a Service. Por otro lado se ha estudiado el uso de Vivado y HLS para la integración de más redes sobre la FPGA y por último se ha trabajado con placas aceleradoras de Xilinx para servidor cómo son la VCK5000 y la Alveo U50 Acceleration Card.

El sistema FPGA as a Service se ha implementado mediante una placa de desarrollo Ultrascale U96-V2 para la inferencia de los modelos y un PC industrial *Beckhoff* sobre el que se despliegan los controladores, almacenamiento y gestión del sistema completo. Este sistema permite realizar un mantenimiento predictivo en base a modelos de forma totalmente automatizada incluyendo el *concept drift* y permitiendo que el modelo se reentrene automáticamente en caso de que las predicciones comiencen a arrojar una tasa de fallos superior.

En lo referente a al uso de Vivado, se ha utilizado HLS y C para desarrollar los kernel para la placa ZUS+ Ultra96-V2 sobre la que se ha conseguido ejecutar una red consistente en distintas capas densas o con una capa densa y una capa LSTM con unos resultados eficientes y un uso de recursos muy optimizado.

Por último se ha realizado la investigación sobre los sistemas FPGA para servidor de Xilinx Inc. entre los que se incluyen las VCK5000 Versal y las Alveo U50. Estas tarjetas aceleradoras se han instalado sobre distintos servidores para comprobar el funcionamiento y la compatibilidad de las mismas ya que inicialmente la especificación de requisitos del fabricante no era correcta. En el momento de compra de la VCK5000, se trataba de una tarjeta en fase de desarrollo, por lo que nos hicimos con una versión ES (*Engineering Sample*) y se encontraban ciertos errores en la documentación inicial, la cual ha ido mejorando hasta el momento de la finalización.

Con esto se ha logrado integrar un sistema FPGA as a Service completo que puede ser desplegado para realizar un mantenimiento predictivo, se ha avanzado en la implementación de redes LSTM sobre estos sistemas FPGA, específicamente y hasta ahora, en las UltraScale+ y se ha procedido a la investigación sobre el uso de las FPGA para centros de datos con la instalación de las tarjetas Alveo U50 y VCK5000 versal sobre servidores de producción.

Palabras clave: FPGA, LSTM, Xilinx, redes neuronales, FaaS, Versal, Alveo, UltraScale

Abstract

A study has been done about the different existing tools and sistems, including software and hardware ones, that may be used with FPGAs technologies to cover the acceleration needs for the automatization of intelligent systems when using neural network models.

For this study, we have followed a development phase divided in 3 lines. First of all, we have used ZUS+ (*Xilinx Ultrascale*) FPGAs, in this case we used the AVNET Xilinx Ultra96-V2, for creating a FaaS (FPGA as a Service). Secondly, we have studied about Vivado SDK and HLS for integrating more neural networks in the FPGA and finally we worked with Xilinx server-side acceleration cards like Versal VCK5000 and Alveo U50.

The FPGA as a Service system has been implemented over a development UltraScale 96-V2 board for inference and an Industrial*Beckhoff* PC in which we deploy docker containers with controllers, storage and full system management. This FaaS System is responsible of work with predictive maintenance based on trained models fully automatically including *concept drift* allowing model to be automatically retrained when predictions are not correct and failure rate is above a specific threshold.

Talking about Vivado, we used HLS and C languages to develope ZUS+ Ultra96-V2 kernel to be able to run a neural network containing different dense layers or with a dense and a LSTM layer with great results and an optimized resource management.

Last but not least , we make a research about Xilinx Inx. FPGA server-side systems, specifically regarding ACAP VCK5000 Versal and Alveo U50 acceleration card. This cards have been installed in different servers to test compatibility between systems as first specification was not correctly made. When we bought VCK5000 Versal card, it was still in development status so we got an ES (*Engineering Sample*) version so there where still some documentation errors.

Thus, we have integrated a fully functional FPGA as a Service system that may be deployed to automatically make a predictive maintenance, we have make advance in LSTM neural networks integration over FPGA systems, specifically for now in UltraScale+ devices, and make some investigation with FPGA use in data centers by installing Alveo U50 and VCK5000 Versal cards in production servers.

Palabras clave: FPGA, LSTM, Xilinx, neural networks, FaaS, Versal, Alveo, UltraScale

Índice general

Índice de Figuras

Índice de tablas

Índice de Códigos

Abreviaturas

Abreviatura	Significado
ACAP	Adaptive Compute Acceleration Platform
ACID	Atomicidad, Consistencia, Aislamiento y Durabilidad (Atomicity, Consistency, Isolation and Durability)
ANN	Redes neuronales artificiales (*Artificial Neural Networks*)
BLAS	Basic Linear Algebra Subroutines
CNN	Redes neuronales convolucionales (*Convolutional Neural Networks*)
CPU	Unidad central de procesamiento (*Central Processing Unit*)
CWI	Centro de matemáticas e informática (*Centrum Wiskunde & Informatica*)
DDR	*Double Data Rate*
DPU	Unidad de Procesamiento Profundo (*Deep Processing Unit*)
ES	Muestra de ingeniería (*Engineering Sample*)
FaaS	FPGA as a Service
FPGA	Field Programmable Gate Array
IA	Inteligencia Articifial
GHz	Gigahercios
GPU	Unidad de Procesamiento Gráfico (*Graphics Processing Unit*)
HLS	*High-Level Synthesis* (Lenguaje de programación)
IaaS	Infrastructure as a Service
ITCL	Instituto Tecnológico de Castilla y León
LSTM	(*Long Sort Term Memory*)
LTS	Soporte de largo plazo (*Long Term Support*)
MHz	Megahercios
ML	Aprendizaje automático (*Machine Learning*)
MNIST	Modified NIST (Modified National Institute of Standards and Technology)
módulos IP	módulos de propiedad intelectual (*Intellectual Property modules*)
NIST	National Institute of Standards and Technology
NN	Redes Neuronales (*Neural Networks*)
NoC	Red en un chip (*Network on Chip*)
OS	Sistema Operativo (*Operating System*)
PC	Ordenador Personal (*Personal Computer*)
PL	Lógica Programable (*Programmable Logic*)
PS	Systema de Procesamiento (*Processing System*)
QoS	Calidad de Servicio (*Quality of Service*)
RAM	Memoria de acceso aleatorio (*Random Access Memmory*)

Abreviatura	Significado
RDBMS	Sistema de administración de bases de datos relacionales (*Relational DataBase Management System*)
RDP	Protocolo de escritorio remoto (*Remote Desktop Protocol*)
REST	Transferencia de estado representacional (*Representational State Transfer*)
RHEL	Red Hat Enterprise Linux
RNN	Recurrent Neural Networks
RTL	*Register Transfer Level* (Lenguaje de programación)
SaaS	Software as a Service
SDK	Kit de desarrollo de software (*Software Development Kit*)
SO	Sistema Operativo
SoC	Sistema en chip (*System on a Chip*)
SNN	Red Neuronal de Impulsos (*Spiking Neural Network*)
UUID	Identificador único universal (*Universally Unique IDentifier*)
VA	Visión Artificial
VPU	Unidad de Procesamiento de Visión (*Visual Processing Unit*)
Win NT	Windows New Technology
XRT	Xilinx RunTime library
ZUS+	Zynq UltraScale+

Tabla 1: **Tabla de Abreviaturas.**

Capítulo 1

Introducción

1.1 Motivación del trabajo

El presente trabajo de fin de máster viene motivado por una necesidad del ITCL (Instituto Tecnológico de Castilla y León) de avanzar en la investigación sobre los sistemas neuromórficos y el despliegue de redes neuronales (NN, del inglés *Neural Networks*) [Schmidhuber, 2015] sobre FPGA (Field Programmable Gate Array).

Por este motivo, el proyecto es un trabajo de fin de máster realizado en empresa que se ha desarrollado en las instalaciones del Instituto Tecnológico de Castilla y León (ITCL), por consiguiente el contenido completo de los códigos desarrollados no puede ser aportado en la memoria del proyecto por motivos de confidencialidad.

Las redes neuronales tienen una gran importancia, la cual aumenta a gran velocidad y muestran cada vez un uso más frecuente en distintas ramas como la robótica y la informática, pero también en otras cómo las finanzas o la medicina. Las Redes Neuronales son una serie de algoritmos desarrollados para tratar de emular el funcionamiento del cerebro humano, permitiendo que los sistemas sean capaces de aprender por si mismos. Estas características, las dotan de gran importancia en los sectores de inteligencia artificial (IA) y específicamente en el de la visión artificial (VA).

ITCL tiene equipo especializado en el desarrollo, entrenamiento y optimización de redes neuronales para la generación de modelos estadísticos. Entre los diferentes tipos de ANN (Artificial Neural Networks)[Ozcalici and Bumin, 2022, p. 4] existentes se incluyen las CNN (Convolutional Neural Networks)[Chen et al., 2020, p. 4], las RNN (Recurrent Neural Networks)[Mou et al., 2017, p. 3640] y el caso específico de estas últimas que son las LSTM (Long Short Term Memmory)[Mou et al., 2017, p. 3641] . Las primeras sí que se pueden ejecutar sobre la FPGA mientras que no tienen soporte actualmente para las segundas.

Las FPGA son dispositivos programables con bloques de memoria y lógicos cuyas conexiones y funcionalidades pueden ser reprogramadas. Este tipo de dispositivos se implementan normalmente cerca de una CPU y se denominan SoC (System on a Chip).

Las características generales de estos dispositivos es el alto rendimiento, el bajo consumo de energía, pequeño tamaño y baja latencia, por lo que estos dispositivos se engloban

dentro del paradigma*Edge Computing* [Caiazza et al., 2022].

Un ejemplo de estos dispositivos es la Ultra96-V2 de Xilinx Inc.[Xilinx, 2019] que es un sistema completo mientras que tenemos las VCK5000 Versal[VCK,] y las Alveo U50[U50,] que son expansiones para utilizar sobre un servidor.

Se ha detectado la utilidad de poder gestionar un sistema automático para la ejecución de inferencias sobre una FPGA, optimizando los modelos y creando resultados en tiempo real y su utilidad pudiendo implementar los diferentes tipos de redes neuronales descritos.

1.2 Objetivo

El principal objetivo del trabajo será la investigación de las diferentes herramientas FPGA existentes y las tecnologías hardware y software disponibles para avanzar con la implementación de estos dispositivos, tanto a nivel embebido con las Ultra96-V2 como en sistemas a mayor escala como las Alveo y las Versal. Este objetivo se ha separado en los 3 apartados sobre los que se va a ir hablando a lo largo del proyecto y que serán lineas de investigación independientes.

La parte más práctica del objetivo será la referente al sistema de FPGA as a Service en la que se realizará la investigación e implementación de los distintos servicios tanto a nivel software cómo hardware para lograr un sistema independiente que permita un mantenimiento predictivo sobre una linea de producción.

Sobre las Ultra96-V2, se tratará de conseguir un sistema que se pueda entregar como servicio y de completar un desarrollo piloto completamente funcional que permita ejecutar inferencias de forma totalmente automatizada sobre estos dispositivos a raíz de datos que se reciban en tiempo real con un sistema de gestión externa.

En otra linea de investigación se va a tratar de conseguir implementar mediante el framework de Xilinx y con Vivado y C el generar redes recursivas y LSTM ,para que se puedan ejecutar sobre los distintos aceleradores FPGA. Empezando por las placas de desarrollo de menor escala Ultra96-V2 y otras placas de la familia Ultrascale, y posteriormente realizando estos procesos para que sean compatibles con las familias de mayor escala para servidor Versal y Alveo.

En lo referente a los dispositivos de servidor, se investigará sobre las compatibilidades con los diferentes sistemas y componentes disponibles y se realizará una instalación completa sobre estos equipos que permitan integrar las herramientas de desarrollo y pruebas para poder avanzar con la implementación de las distintas redes comentadas y lograr una compatibilidad con las RNN obteniendo un máximo rendimiento en la ejecución de los modelos, partiendo de una placa en estado de desarrollo.

1.3 Alcance

El alcance quedará sujeto a los avances que se vayan obteniendo sobre la investigación y la disponibilidad de los diferentes requisitos en las distintas líneas de investigación que se van a seguir.

Por una parte se conseguirá desplegar un sistema completo que contenga los servicios de comunicación, almacenamiento y ejecución de los algoritmos junto con una API que permita la interacción con los distintos elementos.

Para lograr esta independencia de sistemas sin incurrir en problemas en el uso de los datos, en muchos casos de carácter confidencial, y las vulnerabilidades de los sistemas compartidos, se decide utilizar sistemas virtualizados en función de las necesidades y las opciones de las que se dispone y las cuales se estudiarán a lo largo del desarrollo del proyecto.

Con la implementación de las redes recursivas y LSTM se pretende, al menos, lograr la creación de una red que contenga una capa LSTM y se pueda ejecutar sobre las placas de desarrollo, que por defecto no funcionan correctamente, con este tipo de redes y es necesario transformarlas a menor nivel con Vivado SDK realizando una elaboración propia para cada dispositivo específicamente.

Sobre las placas de servidor, se van a establecer los requisitos necesarios para la compatibilidad con el servidor y el despliegue de una plataforma completa, en la que se podrán ejecutar las diferentes redes preparadas mediante algún sistema que las recoja (como una librería) y se pueda utilizar de forma sencilla.

1.4 Estructura de la memoria

La memoria se ha estructurado en diferentes capítulos y secciones tal y como se describe a continuación. Esto se ha realizado para disponer de una división comprensible de los diferentes pasos realizados y, ya que se basa en 3 subproyectos hasta cierto punto independientes, las secciones se dividen para explicar de forma independiente cada uno de estos subproyectos.

El Capítulo 2 contiene un breve estudio sobre otros trabajos existentes que se pueden relacionar en cierta medida con lo estudiado en la siguiente memoria.

El primero de los puntos sobre el desarrollo, Capítulo 3 , trata sobre el hardware utilizado para los diferentes proyectos. En esto se ha diferenciado entre el hardware base, que se trata de los servidores utilizados, y en hardware específico, que contiene las diferentes tarjetas, adaptadores o sistemas independientes que forman parte de los distintos servicios o ayudan a su conexión. En este punto no se ha separado entre los diferentes subproyectos ya que se explica en los apartados posteriores lo utilizado en cada uno de ellos.

El siguiente punto, el Capítulo 4 se enfoca de forma similar, pero en el apartado software y la investigación realizada al respecto. Este apartado se ha dividido en 3 secciones

que contienen los diferentes sistemas operativos estudiados con relevancia para el trabajo y algunas consideraciones, las distintas tecnologías de virtualización que tienen un valor muy importante especialmente en el subproyecto de la FPGA as a Service, pero también utilizado en alguno de los otros con menor importancia y el software de servicios dónde se habla de otro tipo de software con una complejidad considerablemente menor o de uso más genérico que también se ha utilizado durante el desarrollo del proyecto. En este punto tampoco separamos entre subproyectos, ya que se ha utilizado para varios de ellos y se especifica en los apartados posteriores.

El Capítulo 5 Implementación Realizada es el primero que se divide en 3 secciones que se corresponden con los 3 subproyectos mecionados (FPGA as a Service, implementación de LSTM sobre FPGA e instalación de VCK5000 y Alveo U50) y se detallan los avances prácticos llevados a cabo en la fase de desarrollo. Para cada uno de los subproyectos se explican su estado, su modo de funcionamiento o estado de la investigación en función de los requisitos de cada uno de ellos. Este es el punto principal de la memoria ya que contiene la forma de despliegue y los ejemplos y códigos realizados.

A continuación, se ha añadido el Capítulo 6 con las pruebas realizadas en cada uno de los subproyectos y los resultados obtenidos. En lo referente a la instalación de VCK5000 y Alveo U50, como no ha sido posible finalizar todo el despliegue debido a los problemas con el hardware y la disponibilidad del mismo, se han realizado pruebas sobre un sistema externo con el que pudimos hacer ciertos avances en la investigación y una serie de pruebas.

En el Capítulo 7 se ha detallado la Planificación temporal y presupuesto dónde se incluyen sus distintas líneas de trabajo y las semanas en las cuales se debe realizar cada una de ellas. Además, en la parte de presupuesto se muestran de forma separada los gastos materiales y los personales.

El Capítulo 8 de Conclusiones habla sobre los diferentes logros alcanzados tras la ejecución de este proyecto en el momento actual, ya que sigue en desarrollo, ciertas limitaciones encontradas en las diferentes líneas del mismo, problemas encontrados que se basan principalmente en las demoras en el suministro de hardware y un subapartado de trabajos futuros en el que se indican los próximos pasos a seguir y lo que se quiere lograr en el alcance global de este proyecto. Los apartados de este capítulo se dividen también en los diferentes subproyectos comentados.

Por último se encuentran los apéndices y anexos. Los apéndices son de elaboración propia mientras que los anexos contienen información obtenida de terceros y contienen principalmente las hojas de características de los servidores utilizados y sus componentes importantes como son los diferentes procesadores y placas base utilizadas.

Los apéndices, por otro lado, y siendo de elaboración propia, incluyen diferentes fotos y capturas de pantalla tanto de estos servidores como de las tarjetas y elementos utilizados. Además, en el último de estos apéndices se incluyen unos apuntes realizados durante la investigación sobre la instalación de las tarjetas de Xilinx Inc. para centros de datos y contiene ciertas observaciones y comandos que pueden ser útiles en la instalación y administrador de estos servidores y servicios.

Capítulo 2

Trabajo Relacionado

Se ha realizado un análisis sobre otros trabajos realizados anteriormente que se puedan relacionar con el tema estudiado en la presente memoria.

En el trabajo [Orellana and Ricardo, 2017] se realiza un estudio sobre el uso de estas tecnologías de FPGA y evaluan su implementación como un sistema SaaS (*Software as a Service*) o IaaS (*Infrastructure as a Service)*. En este caso se realiza un estudio más teórico de los sistemas en cuanto al hardware de la FPGA y se centra en su gestión de recursos para el entorno cloud [Qian et al., 2009]. Además realiza estudios sobre la estimación de consumos y tiempos y la usabilidad de estos sistemas en entornos virtualizados de computación. Sin embargo este enfoque es totalmente diferente del realizado en esta memoria que se centra en un punto más práctico, orientando el estudio a la implementación práctica de diferentes sistemas sobre hardware específico incluyendo un sistema que hemos denominado FaaS (*FPGA as a Service*) en el que incluímos el hardware para montar un sistema embebido para el uso de la FPGA. En el último caso de nuestra memoria referente a las tarjetas de servidores, y para trabajos futuros, sí que será necesario hacer un enfoque de este estilo para convertir nuestro servidor con estas tarjetas en una especie de sistema IaaS que se pueda ofrecer a terceros.

Por otro lado, en [Proaño Orellana et al., 2016] se habla sobre el uso de las FPGA de una forma que se orienta completamente a las estrategias de planificación de tareas y del uso de recursos de los sistemas para su virtualización. Del mismo modo que en el trabajo anterior, esto nos será más util en los trabajos futuros una vez recibamos el servidor de producción y podamos realizar la instalación y la configuración del sistema y las tarjetas aceleradoras y desplegadas las librerías y el sistema de interacción con las mismas para poder gestionar los recursos de forma eficiente al utilizar estos sistemas tanto de forma interna como tras su orientación a ofrecer a terceros la capacidad de cómputo disponible.

Una vez analizados algunos de los trabajos realizados hasta el momento, vamos a proceder a estudiar el hardware que hemos utilizado en los tres subproyectos que hemos realizado. En este momento no vamos a detallar en cuál de esos proyectos se ha utilizado cada uno de estos elementos hardware pero vamos a analizar tanto los elementos de uso más genérico como servidores y equipos completos como el hardware más específico cómo las tarjetas aceleradoras.

Capítulo 3

Hardware utilizado

3.1 Hardware base

A lo largo del desarrollo del proyecto, se han utilizado distintos elementos hardware como servidores para hacer funcionar los distintos elementos específicos.

En lo relativo al hardware base podemos distinguir 2 apartados diferentes:

- **Beckhoff:** Equipo utilizado para la virtualización de los servicios incluidos en el sistema FaaS y la conexión de la FPGA Ultrascale.
- **Servidores:** Equipos utilizados para la VCK5000 y las U50.

3.1.1 Beckhoff

El dispositivo Beckhoff se ha utilizado para la conexión de la FPGA Ultra96 y la instalación de alguno de los servicios utilizados en FPGA as a Service.

Específicamente hemos utilizado el modelo C6930-0060 que es un PC industrial compacto para montaje en armarios de conexiones, fabricado en aluminio y con un amplio número de interfaces de entrada/salida [bec,]. Se muestra una captura de las características desde el panel de Windows en la Figura 3.1 .

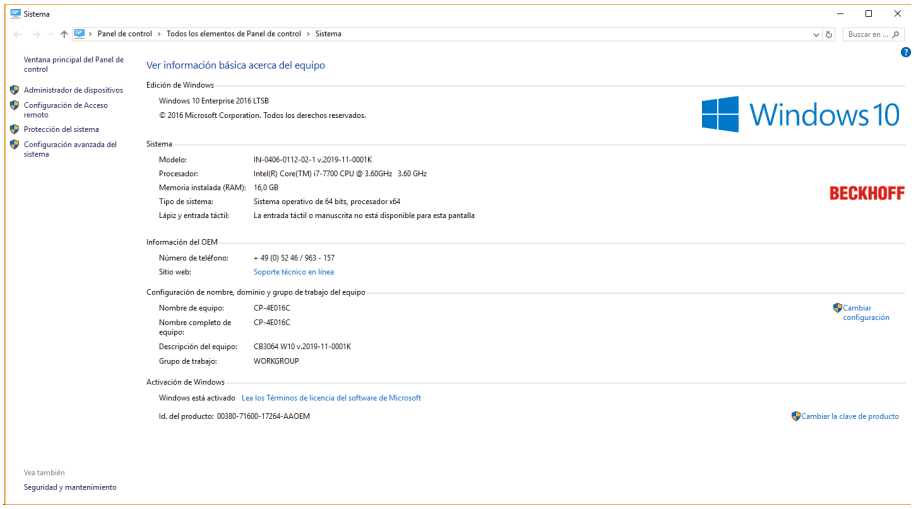


Figura 3.1: **Captura de las características del equipo Beckhoff C6930.**

Este dispositivo tiene una placa base desarrollada por Beckhoff (modelo CB3064-0002 [cb3,]) en la que se integran un procesador Intel i7-7700 con 16Gb de memoria RAM, además dispone de 2 slots de conexión PCIe 3.0 e incluye otras conexiones cómo son 4 USB 3.0, un puerto de conexión de pantalla DVI, un puerto RS232 y 4 puertos de red RJ45. Estas interfaces de conexión se pueden ver en Figura 3.2

Figura 3.2: **Captura frontal de las interfaces de conexión del Beckhoff.**

3.1.2 Servidores

Para la conexión de las aceleradoras orientadas a centros de datos (Versal y Alveo) se utilizará un único servidor sobre el que realizar la instalación. En nuestro caso hemos

necesitado utilizar distintos equipos debido a incompatibilidades que íbamos encontrando con las tarjetas en los diferentes componentes de los que disponíamos.

3.1.2.1 Servidor 1

La primera prueba se realizó sobre un servidor del que se disponía en la empresa con las siguientes características. La Figura 3.3 muestra una captura con los datos de la placa base de este servidor.

- **Placa base:** Gigabyte H270-HD3-CF [Gig, a]

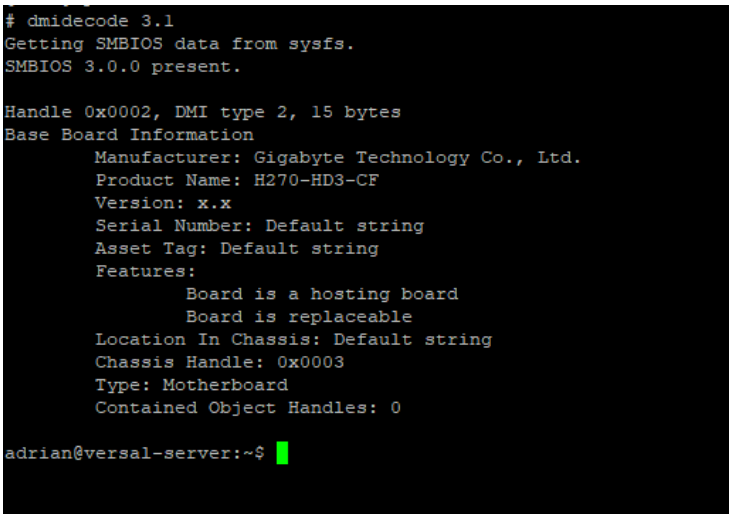


Figura 3.3: **Detalles de la placa base del servidor 1 obtenidas mediante *dmidecode -t 1***

  - **Chipset:** Intel® H270 Express Chipset
  - **Memoria:** 4 x DDR4 DIMM dual channel ECC sockets
  - **Red:** Intel® GbE LAN chip (10/100/1000 Mbit)
  - **Expansión:**
    - 1x PCI Express 3.0 x16 slot
    - 2x PCI Express 3.0 x4 slots (tamaño x16)
    - 1x PCI Express 3.0 x1 slot
    - 1x PCI slot

- **Procesador:** Intel i7-7700K @ 4.20GHz. Se muestra en la Figura 3.4

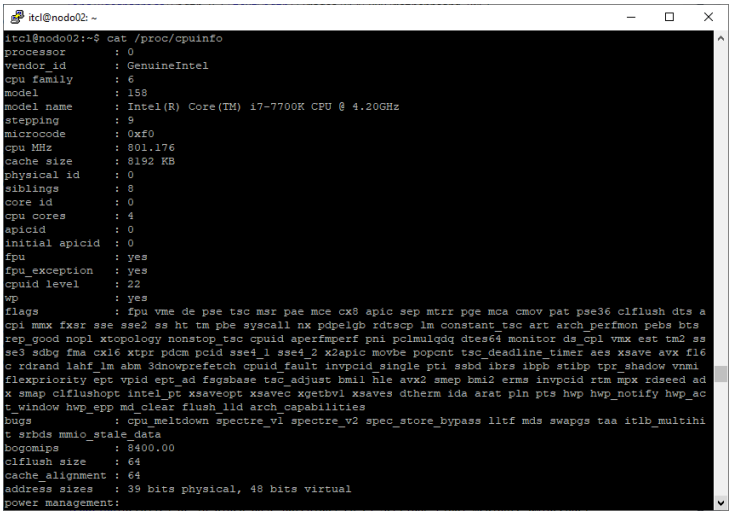


Figura 3.4: **Detalles del procesador utilizado en el Servidor 1 mediante el comando *cpuinfo***

3.1.2.2 Servidor 2

Para el segundo servidor se utilizó otro equipo de los que estaban siendo utilizados para *Deep Learning* con GPUs y diferentes componentes. La Figura 3.5 muestra una captura con los datos de la placa base de este servidor.

- **Placa base:** Gigabyte GA-Z170XP-SLI [Gig, b]

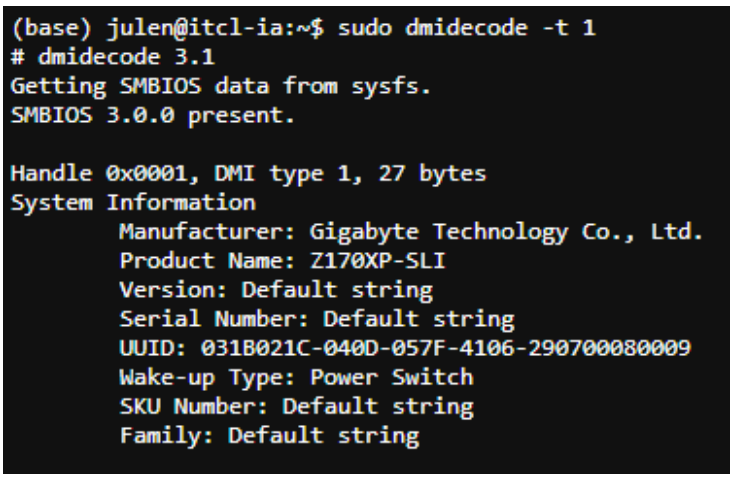


Figura 3.5: **Detalles de la placa base del servidor 2 obtenidas mediante *dmidecode -t 1***

  - **Chipset:** Intel® Z170 Express Chipset
  - **Memoria:** 4 x DDR4 DIMM dual channel ECC sockets
  - **Red:** Intel® GbE LAN chip (10/100/1000 Mbit)
  - **Expansión:**
    * 1x PCI Express 3.0 x16 slot

  - 1x PCI Express 3.0 x8 slot (tamaño x16)
  - 1x PCI Express 3.0 x4 slot (tamaño x16)
  - 2x PCI Express 3.0 x1 slots
  - 2x PCI slots

- **Procesador:** Intel i7-7700K @ 4.20GHz Se muestra en la Figura 3.6

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz
stepping        : 9
microcode       : 0xea
cpu MHz         : 1299.835
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 0
cpu cores       : 4
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx p
dpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopo
logy nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx e
st tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt
 tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpu
id_fault invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriori
ty ept vpid ept_ad fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid
 rtm mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves
 dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp md_clear flu
sh_l1d
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf
 mds swapgs taa itlb_multihit srbds
bogomips        : 8400.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
```

Figura 3.6: **Detalles del procesador utilizado en el Servidor 2 mediante el comando *cpuinfo***

3.1.2.3 Servidor Final

- **Plataforma:** Chassis Server RackMount 2U ESC4000A-E11 [Azk,]

Figura 3.7: **Detalles de la placa base obtenidas en el Servidor Final mediante dmidecode**

  - **Socket:** SP3 (LGA 4094) AMD EPYC™ 7003 Series
  - **Memoria:** DDR4 3200 RDIMM, DDR4 3200 LRDIMM, DDR4 3200 LRDIMM 3DS. Hasta 2048Gb
  - **Red:** 1 x Dual Port Intel I350-AM2 Gigabit LAN controller + 1 x Mgmt LAN
  - **Expansión:**
    * 8x PCI Express 4 x16 slots

- **Procesador:** AMD EPYC 7313P 3GHz 128MB Cache 16 Cores

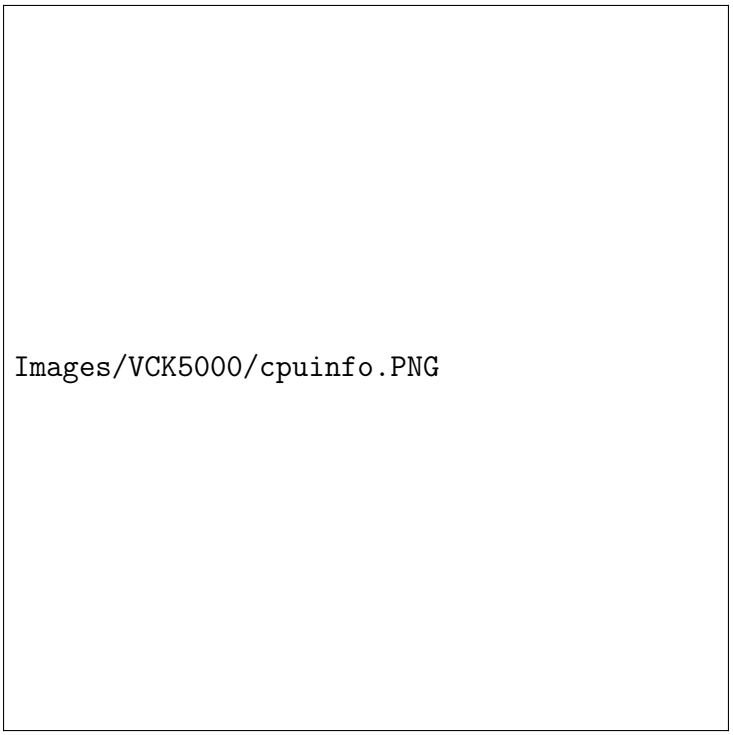


Figura 3.8: **Detalles del procesador utilizado en el Servidor Final mediante el comando *cpuinfo***

3.2 Elementos específicos

Además del hardware base mencionado anteriormente, se han utilizado ciertos elementos hardware tanto de aplicación específica cómo de uso general.

3.2.1 AVNET Xilinx Ultra96-V2

La placa Ultra96-V2 es un SoC basado en la familia de Xilinx Zynq Ultrascale+ (ZUS+) MPSoc. La placa se muestra como Avnet Xilinx Ultra96-V2 y podemos ver una imagen de esta placa como Figura 3.10

La familia Zynq Ultrascale+ es un dispositivo que consiste en 2 áreas principales:

- Sistema de procesamiento (PS)
- Lógica programable (PL)

En la Figura 3.9 se muestra un diagrama de los componentes de la familia ZUS+ extraída de [Xilinx, 2020c, p. 201].

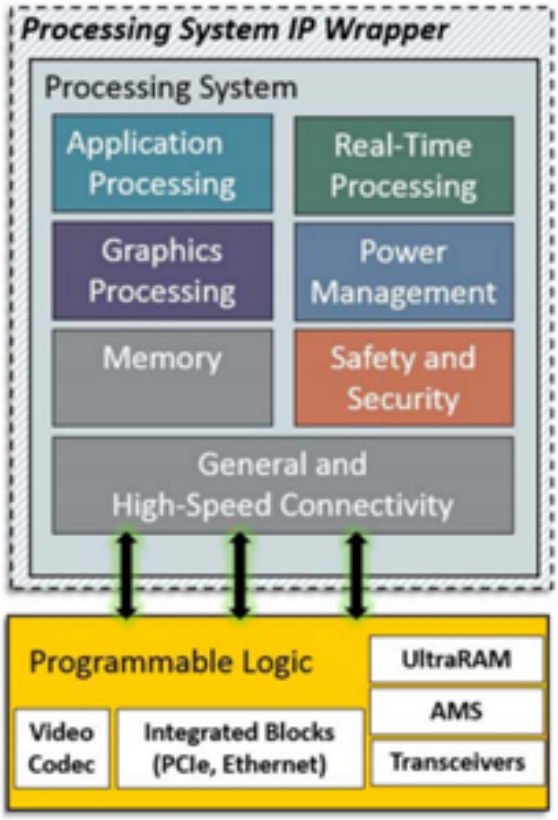


Figura 3.9: **Diagrama de bloques de ZUS+, extraído de [Xilinx, 2020c, p. 201]**

El área PS contiene la CPU y las conexiones con dispositivos periféricos. Contiene un microprocesador ARM Cortex A53 con arquitecturas de 32 o 64 bits consistente en 4 núcleos de hasta 1.5GHz que pueden trabajar bien de forma independiente o de forma conjunta como un clúster y se pueden programar a través de un sistema operativo (OS) Linux o bien programándolas directamente con C/C++.

La unidad de procesamiento en tiempo real es un ARM Cortex de 2 núcleos de hasta 600MHz con arquitectura de 32 bits. Se puede controlar a través de un sistema RT-Linux o a través de C/C++ cómo en el caso de la CPU.

Cómo sistemas de memoria, se utilizan memorias DDR4 y flash.

El área PL tiene la tecnología FPGA. Dispone de BlockRAMs y UltraRAMs para la lectura y la escritura de los datos a alta velocidad. Las ZUS+ tienen miles de estos bloques de memoria, lo que les permite el procesamiento masivo de datos en paralelo. Este área (PL) se puede programar utilizando lenguajes de programación como *RTL* o *Verilog* diseñando el circuito digital de forma muy compleja a bajo nivel. Otra forma para programarla es utilizar C/C++ a través de HLS que traduce el código desarrollado en C a diseños en bajo nivel RTL. Estos algoritmos se llaman módulos IP (Propiedad Intelectual).

Sobre las FPGA se puede implementar prácticamente cualquier algoritmo utilizando C y C++ mediante HLS, especialmente los modelos de ML (Aprendizaje automático). Debido al reciente crecimiento en el uso de estos modelos, Xilinx Inc. ha desarrollado

módulos de IP específicos y de alto rendimiento para modelos de ML, la DPU (Unidad de Procesamiento Profundo). [Xilinx, 2020b, p. 338]. LA DPU es un conjunto de instrucciones especializadas que permiten la implementación eficiente de muchas CNN, con un flujo de trabajo relativamente sencillo.

Xilinx suministra el entorno de trabajo para implementar modelos ML de alto nivel con TensorFlow, PyTorch o Caffe directamente en la DPU a través de Vitis-AI. A pesar de que esta DPU es muy eficiente para CNNs, no soporta por el momento otro tipo de redes como las LSTM (Long Short Term Memory), que son un tipo de RNN o las SNN (Spiking Neural Networks). Para resolver esta limitación, es necesario trabajar directamente en HLS, para ello hay distintas iniciativas como HLS4ML que simplifican la generación del HLS al traducir TensorFlow y PyTorch directamente a HLS como hace Vitis-AI.

A pesar de que existen otras soluciones para este tipo de *edge computing* basados en GPU y en VPU cómo *Jetson Nano* de *NVIDIA* basado en GPU o el MX8 de *Intel* basado en VPU, se basan en la aceleración de los modelos de IA en lugar de basarse directamente en la programación lógica. Por lo tanto, para un problema de ML los algoritmos de preprocesado y postprocesado se pueden suministrar junto con el modelo en una FPGA para entender la salida del mismo reduciendo el tiempo de procesado mientras que en el caso de GPU y VPU no se pueden implementar cerca del acelerador y se requiere de una CPU adicional para realizarlo.

La tecnología FPGA aporta una solución completa para los problemas de ML ya que los modelos se pueden implementar junto con el resto de algoritmos.

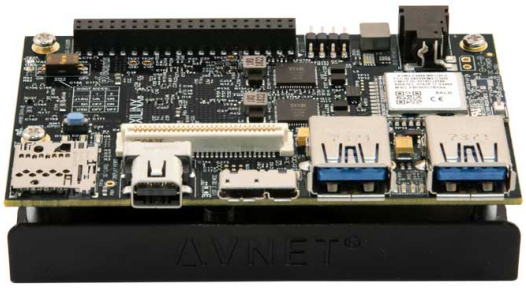


Figura 3.10: **Detalle de la placa Ultra96-V2 en la web de Avnet**

3.2.2 Xilinx VCK5000 Versal

La placa VCK5000 pertenece a la familia de arquitectura ACAP (Adaptive Compute Acceleration Platform) y es parte de *AI Core Series*. Las capturas de esta placa se muestran en el apéndice VCK5000 Versal.

La familia ACAP es un sistema de *Network on Chip* (NoC) programable tanto a nivel de software como a nivel de hardware que disponen de una plataforma de computación multinúcleo [ACA,].

En la Figura 3.11 se muestra el diagrama funcional de bloques de los componentes de las Versal ACAP extraida de [Xilinx, 2020a, p. 4]

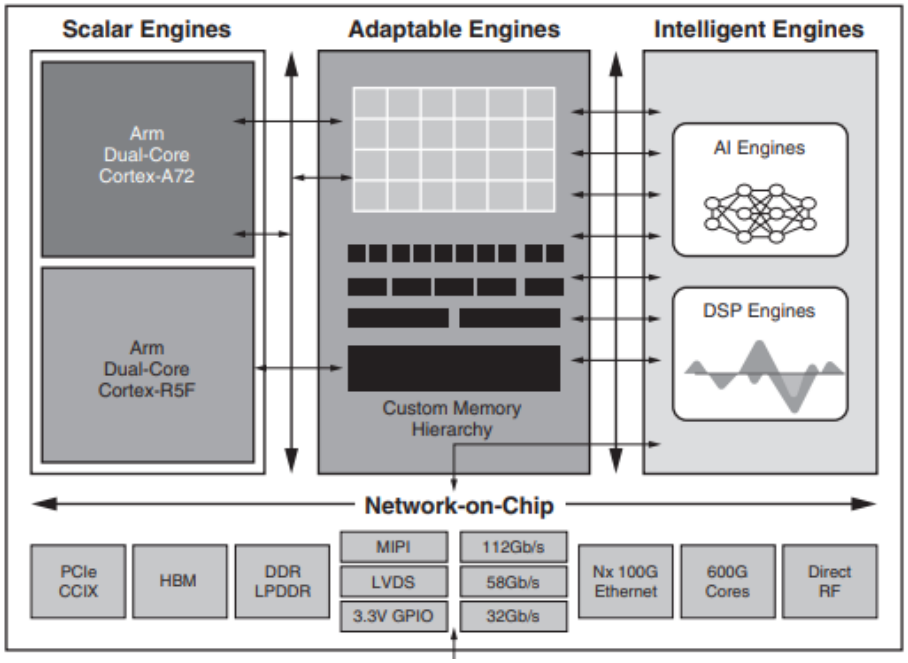


Figura 3.11: **Diagrama de bloques de Versal ACAP, extraído de [Xilinx, 2020a, p. 4]**

Esta tarjeta, la VCK5000, tiene un dispositivo VC1902 cuyo diagrama interno es el mostrado en la Figura 3.11 y el diagrama propio de la tarjeta con sus componentes se muestra en la Figura 3.12 obtenido de la guía de usuario confidencial de Xilinx.

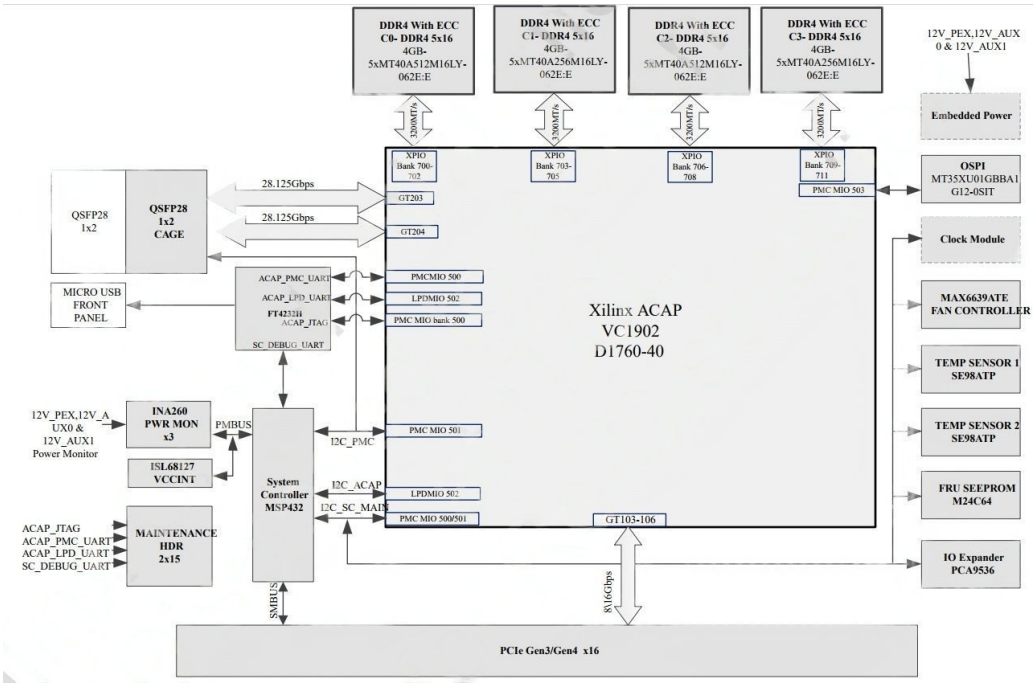


Figura 3.12: **Diagrama de bloques de Versal VCK5000**

Las características de esta tarjeta se muestran en la Figura 3.13 y de entre ellas destacamos que a pesar de que indica que tiene interfaz PCIe Gen3x16, a nosotros nos ha dado problemas en todas las placas que hemos probado con estas características, por

lo que finalmente se deja su instalación a un equipo con PCI Gen4x16 para asegurar su funcionamiento. No sabemos si es un error en las especificaciones o algún otro tipo de compatibilidad.

En cuanto a conectividad dispone de 2 interfaces de fibra QSFP28 (100GbE) y un puerto microUSB interno que sirve como interfaz para cargar modificaciones y actualizaciones de firmware mediante Vivado SDK y soporta picos de hasta 145 TOPs para enteros de 8 bits. La Figura 3.13 muestra las características principales de esta tarjeta.

Card Specifications	**VCK5000**	
Device	**VC1902**	
Compute	**Active**	**Passive***
INT8 TOPs (peak)	145	145
Dimensions		
Height	Full	Full
Length	Full	3/4
Width	Dual Slot	Dual Slot
Memory		
Off-chip Memory Capacity	16 GB	16 GB
Off-chip Total Bandwidth	102.4 GB/s	102.4 GB/s
Internal SRAM Capacity	23.9 MB	23.9 MB
Internal SRAM Total Bandwidth	23.5 TB/s	23.5 TB/s
Interfaces		
PCI Express	Gen3 x 16 / Gen4 x 8	Gen3 x 16 / Gen4 x 8
Network Interfaces	2x QSFP28 (100GbE)	2x QSFP28 (100GbE)
Logic Resources		
Look-up Tables (LUTs)	899,840	899,840
Power and Thermal		
Maximum Total Power	225W	225W
Thermal Cooling	Active	Passive

Figura 3.13: **Especificaciones técnicas de la VCK5000 Versal**

3.2.3 Alveo U50 Accelerator Card

Se trata de una tarjeta aceleradora para centros de datos instalable en cualquier servidor. Es una tarjeta de arquitectura UltraScale+, en la Figura 3.14 se muestra el diagrama funcional de bloques de los componentes de las Versal ACAP extraida de [Xilinx, 2020a, p. 4]

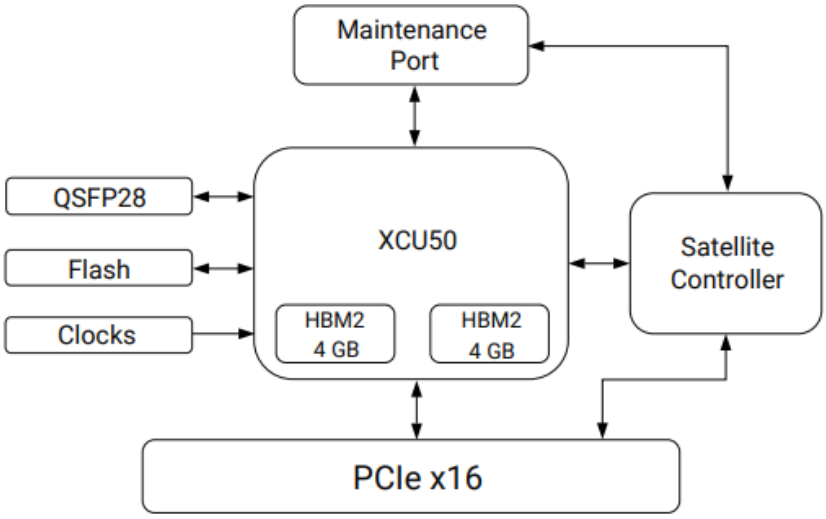


Figura 3.14: **Diagrama de bloques de la tarjeta Alveo U50**

Tiene una potencia de 75W, de pequeño tamaño, y cuenta con una interfaz de red de 100 Gbps y disipación de calor pasiva y sus características técnicas se muestran como Figura 3.15 . Es compatible con el framework Vitis AI y programable mediante Vivado SDK.

Specification	U50 Production[1]	U50 LV Production[1]
Product SKU	A-U50-P00G-PQ-G	A-U50-P00G-LV-G
Total electrical card load[2]	75W	75W
Thermal cooling solution	Passive	Passive
Weight	300g – 325g	300g – 325g
Form factor	Half height, half length	Half height, half length
Network interface	1x QSFP28 (100 GbE)	1x QSFP28 (100 GbE)
Network clock precision	IEEE 1588	IEEE 1588
PCIe interface[3, 4]	Gen3 x16, Gen4 x8, CCIX	Gen3 x16[5,6]
HBM2 total capacity	8 GB	8 GB
HBM2 bandwidth	316 GB/s[7]	316 GB/s[7]
Look-up tables (LUTs)	872K	872K
Registers	1,743K	1,743K
DSP slices	5,952	5,952
Max. Dist. RAM	24.6 Mb	24.6 Mb
36 Kb block RAM	1344 (47.3 Mb)	1344 (47.3 Mb)
288 Kb UltraRAM	640 (180.0 Mb)	640 (180.0 Mb)
GTY transceivers	20	20
V_{CCINT} supported	V_{NOM} (0.85V)	V_{LOW} (0.72V)
Vitis™ Development Enviroment	Yes	Yes
Vitis platform	Gen3 x16 XDMA, Gen3 x4 XDMA[8]	Gen3 x4 XDMA[9]
Vivado Design Suite	Yes	Yes
Target workloads	Fintech, video, database, and computational storage	Machine learning (ML) inference

Figura 3.15: **Especificaciones técnicas de la Alveo U50**

3.2.4 tp-link UE300

Adaptador de USB 3.0 a gigabit ehternet 100/1000 utilizado para la conexión de la FPGA AVNET Xilinx Ultra96-V2 mediante red con el Beckhoff para permitir la conecti-

vidad entre los servicios.

Compatible con los diferentes sistemas operativos y de diseño compacto [tp-,] que facilita su integración en el armario de conexiones. Podemos observar este dispositivo en la Figura 3.16 y una imagen de como lo tenemos conectado en un sistema en producción en la Figura 3.17

Figura 3.16: **Captura de la web de tp-link del dispositivo utilizado.**

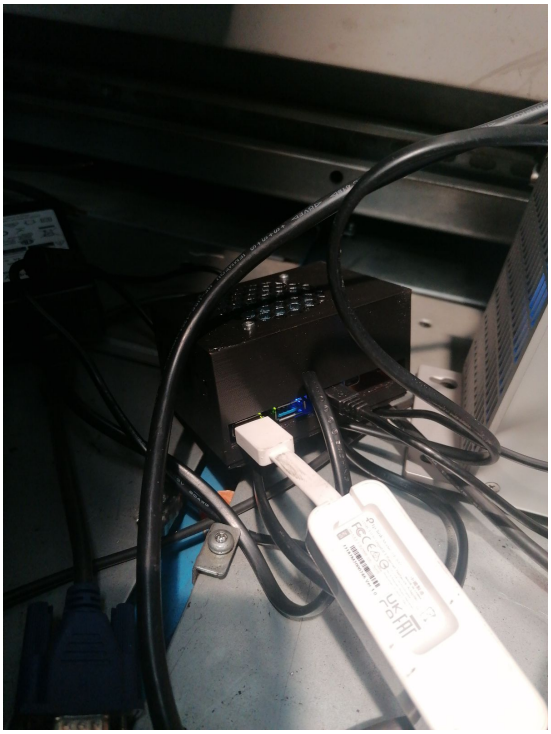


Figura 3.17: **Captura del adaptador *tp-link UE300* conectado con el sistema.**

Hasta aquí tenemos estudiados y descritos los diferentes elementos hardware que vamos a utilizar durante el desarrollo de este proyecto, procedemos a realizar un estudio de investigación sobre diferentes tecnologías software que nos pueden ser útiles para las diferentes secciones del proyecto. Se evalúan distintos sistemas operativos que se van a utilizar, sistemas de virtualización que servirán inicialmente para el sistema FaaS y resto de software auxiliar entre el que destaca la investigación sobre bases de datos relacionales y NoSQL a pesar de que estas últimas no se llegarán a utilizar.

Capítulo 4

Software Estudiado

Para el desarrollo de este proyecto hemos utilizado una variedad de elementos software, tanto a nivel de sistema operativo cómo de software específico o de servicios.

4.1 Sistemas operativos

Hemos utilizado sistemas operativos basados en Windows para el Beckhoff y sistemas Linux para los servidores, máquinas virtuales y las ZUS+ sobre los que se instalan las herramientas de Xilinx.

4.1.1 Windows

Utilizados en los dispositivos Beckhoff, realizando pruebas con el SO Win7 y utilizado finalmente para el despliegue Win10 ya que es la última versión del SO publicado por la marca.

4.1.1.1 Windows 7

Windows 7 es la versión inicial entregada con los sistemas industriales de Beckhoff Automation GmbH, los nuevos sistemas se encuentran disponibles con Windows 10 y son los que finalmente se han utilizado en el despliegue real del producto.

4.1.1.2 Windows 10

Se trata de la versión del SO de Microsoft basado en el núcleo de la arquitectura Win NT lanzado en 2015 y la última versión en la actualidad para los sistemas Beckhoff [Bec,]. Se ha utilizado finalmente un sistema operativo Windows 10 Enterprise 2016 LTSB, incluida en el dispositivo Beckhoff.

4.1.2 GNU/Linux

Los sistemas operativos GNU/Linux son sistemas operativos de software libre y código abierto de tipo UNIX que se basan en el kernel *OpenSource* Linux. Se utilizarán como SO para los servidores y en las placas ZUS+ como base para el despliegue del resto de software y servicios utilizados. En este caso se han utilizado:

- **Ubuntu:** Sistema operativo Linux basado en Debian [Deb,] y desarrollado por Cannonical Ltd. Es la principal distribución de Linux y la utiliza cerca del 50 % de los usuarios.
  - **Ubuntu 18.04 LTS**
  - **Ubuntu 20.04 LTS**
  - **Ubuntu 22.04 LTS**
- **CentOS:** (*Community ENTerprise Operating System*) Se trata de un fork del SO RHEL (*Red Hat Enterprise Linux*) a partir del código fuente del mismo en su versión 7 que tendrá soporte hasta 2024 ya que desde la versión 5, CentOS, ofrece soporte durante 10 años. A partir de 2020, estando patrocinado por Red Hat, pasó a ser discontinuado y se terminó su soporte liberando CentOS Stream cómo versión de desarrollo *Rolling Release*.
  - **CentOS 7**
- **Petalinux:** Sistema de Xilinx Inc. que utilizaremos para las ZUS+ Ultra96v2.

4.1.2.1 CentOS 7

Uno de los sistemas operativos más utilizados para servidores a nivel empresarial. Se han realizado pruebas pero finalmente no se ha utilizado debido al cambio en la política, la discontinuación y finalización de soporte del sistema operativo ya que el nuevo CentOS Stream no ofrece el tipo de sistemas estables que necesitamos para los servidores si no que se convierte en un entorno de desarrollo para el *mainstream* que será RHEL.

4.1.2.2 Ubuntu 18.04

Versión del SO de Cannonical Ltd. publicada en abril de 2018 bajo el nombre *Bionic Beaver* en la versión LTS (*Long Term Support*) con soporte para 5 años, hasta abril de 2023. Es la versión más estable de la que disponemos en servidores por su compatibilidad con librerías y el resto de software utilizado y se basa en el *kernel* de Linux versión 4.15

4.1.2.3 Ubuntu 20.04

Cannonical Ltd publicó la versión 20.04 de su sistema operativo en abril en abril de 2022 en la versión LTS bajo el nombre *Focal Fossa* con soporte para 5 años, hasta abril de 2025. Venía sobre la versión de *kernel* 5.4 con soporte ampliado para nuevo hardware.

4.1.2.4 Ubuntu 22.04

Es la última versión desplegada del del SO de Cannonical Ltd. en abril de 2022 y con soporte hasta 2027 en su versión LTS. Esta versión nos presentaba mucha incompatibilidad con librerías antiguas por lo que no lo hemos utilizado ya que nos interesa buscar una estabilidad con las versiones existentes.

4.1.2.5 Petalinux 2022.1

No se trata de un sistema operativo como tal, si no que es una herramienta que permite compilar e implementar distintos tipos de soluciones basadas en Linux, específicamente en la versión de Linux embebido, para las herramientas de Xilinx. Esta herramienta está diseñada para la optimización de estos sistemas pudiendo generar diferentes distribuciones con las dependencias requeridas para la aplicación final una vez generado el kernel del sistema.

Permite desarrollar las distintas soluciones basadas en el kernel de Linux, integrando desde el arranque del sistema como las aplicaciones y las herramientas internas generando un sistema software específico para el sistema hardware a utilizar.

4.2 Tecnologías de virtualización

Las tecnologías de virtualización nos permiten simular sistemas informáticos independientes con diferentes servicios en un mismo servidor físico. De esta forma, se permite optimizar el rendimiento y el uso de recursos de una manera más económica ya que se posibilita administrar de forma más conveniente los sistemas con grandes capacidades de recursos.

Existen diferentes métodos para la virtualización de los sistemas informáticos y nos basamos en 2 tipos que nos ofrecen enfoques diferentes para realizar esta virtualización. Cada uno de estos tipos los utilizaremos en distintos puntos del proyecto.

- **Virtualización de hardware**
- **Virtualización de contenedores**

A continuación en la Figura 4.1 se muestra un gráfico comparando las distintas estructuras mencionados. A la izquierda se representa la emulación de hardware mientras que en la imagen derecha se muestra un ejemplo de virtualización de contenedores (en este caso Docker).

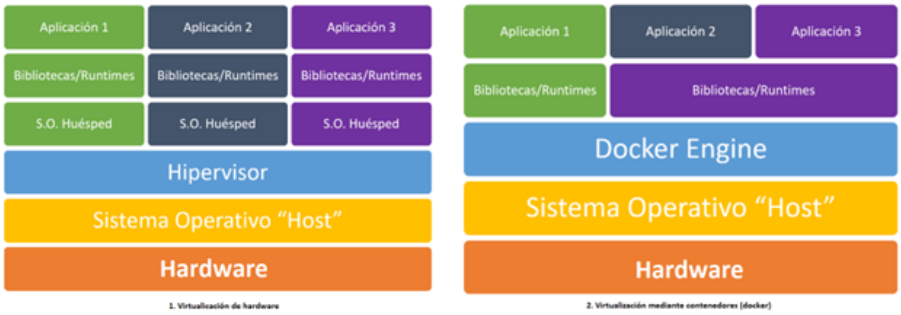


Figura 4.1: **Virtualización de hardware vs virtualización de contenedores.**

En la virtualización de contenedores, la capa de **librerías/runtime** se comparte entre las diferentes máquinas que utilizan el mismo sistema operativo y que poseen librerías en común y únicamente se duplica la parte correspondiente a la capa de aplicación exclusivamente.

Por otro lado, la virtualización mediante emulación de hardware nos permite disponer de unos sistemas completamente independientes con su **propio hardware virtual**, conteniendo su propio sistema operativo y conjunto de librerías exclusivas, las cuales se duplicarán entre distintos sistemas virtuales incluso si utilizan alguna librería en común, permitiendo disponer de un sistema aislado completamente tanto de los elementos del anfitrión cómo de los elementos del resto de sistemas virtualizados.

4.2.1 Virtualización de hardware

Es el enfoque más clásico de virtualización y lo utilizan los sistemas como las máquinas virtuales. Consiste en la emulación completa de los componentes hardware del servidor de forma que cada una de las máquinas virtualizadas dispone de su propia asignación de recursos hardware como son la CPU, la memoria, los discos (duros, CD,...) o las interfaces de red.

Existen 2 tipos diferentes de *hipervisores*, que son los sistemas encargados de gestionar la virtualización:

- Los **hipervisores de tipo1** o ***bare metal*** son los sistemas que se instalan directamente en el hardware de la máquina de forma idéntica a la instalación de un sistema operativo. De esta forma, estos hipervisores tienen acceso completamente directo a los distintos recursos de la máquina y ofrecen una mejor eficiencia y mayor seguridad ya que cada SP virtualizado se encuentra aislado, evitando que un ataque dirigido a alguna parte del sistema pueda extenderse a otros sistemas virtualizados que se ejecuten sobre la misma máquina.
- Los **hipervisores de tipo 2** o ***hosted hypervisor*** son aquellos sistemas de virtualización que se instalan **sobre** un sistema operativo. Este tipo de hipervisores necesitan hacer llamadas al propio sistema operativo anfitrión para realizar la reserva y asignación de recursos para las distintas máquinas virtuales, lo cual aumenta la latencia respecto al caso anterior. Además, este sistema de hipervisión tiene el problema de que cualquier vulnerabilidad del sistema operativo anfitrión puede

comprometer al resto de máquinas virtualizadas. Por este motivo, estos sistemas de virtualización se suelen utilizar en sistemas dónde se va a realizar un uso ocacional ya que tienen un menor coste y una mayor facilidad de instalación y configuración de los sistemas.

A continuación se muestra en la Figura 4.2 un diagrama sobre la diferencia entre ambos tipos de hipervisores.

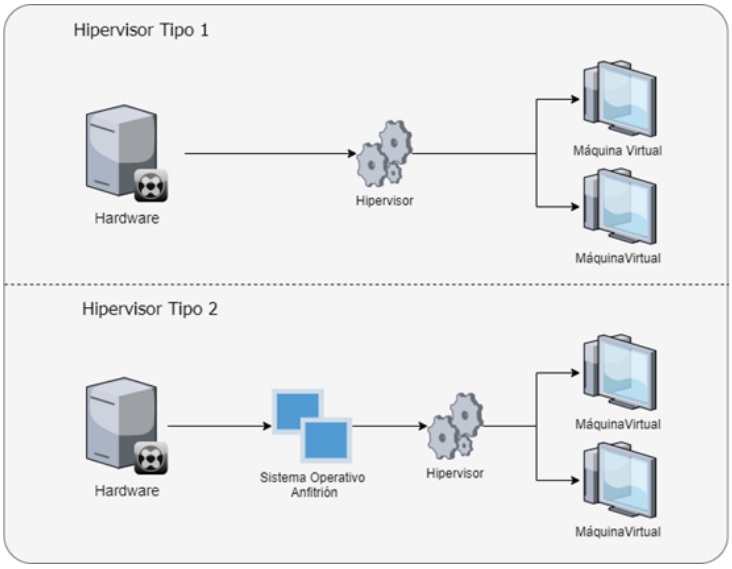


Figura 4.2: **Distintos tipos de hipervisores.**

Respecto a las tecnologías de virtualización de hardware hemos estudiado y utilizado distintos sistemas:

4.2.1.1 VMWare

Se trata de un sistema de virtualización clásico de emulación de hardware, desarrollado y propiedad de VMWare Inc.

Tiene una oferta de diferentes productos tanto gratuitos como comerciales para la emulación de sistemas, entre ellos destacamos:

- **VMWare Sphere:** Sistema hipervisor **bare metal** (tipo 1) que funciona cómo SO dedicado a la virtualización. Proporciona una menor pérdida de rendimiento que utilizando VMWare cómo software sobre otro sistema operativo.
- **VMWare Workstation/Player/Fusion:** Es el sistema hipervisor **de escritorio** (tipo 2) que permite ejecutar tanto máquinas virtuales como contenedores e incluso clústeres de kubernetes. La versión **workstation** es de **pago** mientras que la versión **player** es gratuita a cambio de tener ciertas limitaciones. Ambas versiones están disponibles para ejecutarse sobre sistemas *Windows* y sobre sistemas *Linux* mientras que la versión para *MacOS* es la versión **Fusion**.

4.2.1.2 VirtualBox

Virtualbox es un hipervisor de **tipo 2** desarrollado inicialmente por *Innotek* aunque actualmente es propiedad de *Oracle Corp.*

Está disponible en versiones para sistemas operativos *Windows*, *MacOS*, *Linux* y *OpenBSD* y permite emular sistemas tanto de 32 bits como de 64 bits.

Es un sistema de virtualización de **distribución gratuita** compatible con la administración remota de máquinas virtuales sin necesidad de ningún tipo de licencia para su uso comercial aunque presenta un rendimiento algo inferior a otros hipervisores de tipo 2.

4.2.1.3 Hyper-V

Se trata de un sistema desarrollado por Microsoft Corp. para la virtualización de sistemas operativos de **64 bits** y que dispone de dos modos de uso:

- **Hyper-V server:** Se trata de un hipervisor de **tipo 1** gratuito que se instala como si fuera un sistema operativo. No dispone de interfaz gráfica, por lo que se deben realizar todas las operaciones a través de la linea de comandos. Permite su administración de forma directa o a través de la red.
- **Hyper-V:** Es un hipervisor de **tipo 2** que funciona de forma similar a los anteriores. Sólo está disponible para sistemas operativos **Windows Server** y para las versiones profesionales de escritorio de **Windows**.

A continuación se muestra en la Figura 4.3 una captura de la arquitectura de este software.

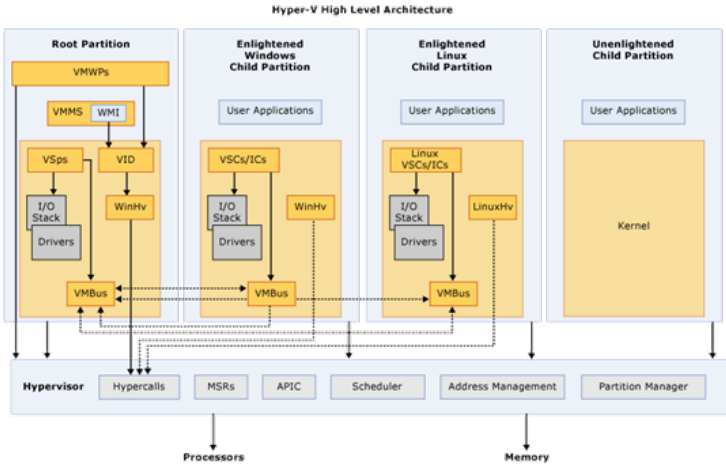


Figura 4.3: **Arquitectura de Hyper-V [hyp,]**

A continuación en la Tabla 4.0 podemos ver una tabla resumen con la comparativa entre los distintos sistemas de virtualización de hardware que se han comentado anteriormente.

Name	Max. host cores/CPUs	Max host mem	Max host disk size	Max guest VM running	Max vCPU per VM	Max mem per VM	Max SCSI+IDE disks per VM	Max disk size per guest
VMWare Player 15.0	No limit	No limit	No limit	No limit	16	4 GB (32bit) 64GB (64 bit)	?	8 TB
VMWare vSphere Hypervisor (ESXi 7.0)	16 NUMA 896 vCPU	24TB	Depending on fs	1024	768	24 TB	4 IDE 256 SCSI 120 SATA 60 NVMe	62 TB
VirtualBox	No limit	No limit	No limit	No limit	32	No limit	4 IDE no limit SATA no limit SCSI no limit SAS	GUI 2TB no limit cmd
Hyper-V server 2016	512 cores 320CPUs	24 TB	No limit	1024	240	12 TB	4 IDE 256 SCSI	64 TB

Tabla 4.0: **Comparación entre sistemas de virtualización. [vir,]**

4.2.2 Virtualización de contenedores

La virtualización de contenedores consiste en el uso de paquetes de software ligero que permiten realizar una virtualización a nivel de **sistema operativo** y que permite que se ejecuten como aplicaciones del sistema en lugar de realizar una emulación de componentes hardware.

De este modo, se pueden ejecutar varios contenedores (**máquinas virtuales**) aislados sobre el*kernel* del sistema operativo anfitrión sin generar una sobrecarga cómo sucedía en el caso de la virtualización de hardware ya que en este caso no es necesario correr un sistema operativo completo en cada una de las instancias.

4.2.2.1 Docker

Se trata del sistema de virtualización de contenedores más conocido. Es un software de **código abierto** que se encuentra disponible para gran cantidad de plataformas y de sistemas operativos, incluyendo *MacOS*, *Windows* y *Linux* (para el que fue inicialmente desarrollado y en el que más funcionalidades y compatibilidades encontramos) y las distintas plataformas tanto x86_64 como arm.

Funciona mediante una arquitectura cliente-servidor en la cual el cliente de Docker realiza peticiones a un *daemon* que realiza las operaciones sobre los diferentes contenedores. El cliente y el servidor se pueden encontrar en la misma máquina o en equipos diferentes ya que permite una comunicación y administración de forma remota.

Un contenedor es un entorno aislado del resto de contenedores, lo que ofrece una seguridad por defecto y de forma similar a la virtualización de hardware, permite ejecutar múltiples contenedores en la misma máquina física. Cada contenedor contiene todo lo necesario para ejecutar la aplicación desplegada y para realizar las conexiones de red, incluyendo la opción de conectarlos mediante **redes virtuales privadas** entre los diferentes contenedores.

Internamente Docker funciona como una interfaz y una serie de herramientas que utiliza como **containerd** que es el elemento generador de contenedores. Por ello, se pueden crear estos contenedores utilizando directamente containerd u otras herramientas que lo faciliten.

La arquitectura interna de funcionamiento de docker se muestra en la Figura 4.4 extraída de la web de docker [doc,]

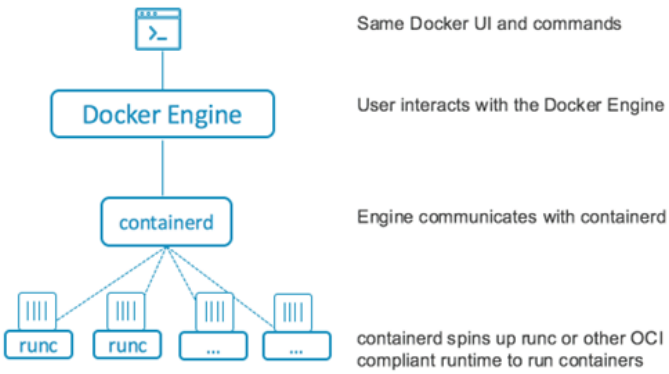


Figura 4.4: **Arquitectura interna de docker. [doc,]**

Además de Docker, se han valorado y estudiado otras herramientas de contenerización que se detallan a continuación, aunque finalmente no han sido utilizadas.

4.2.2.2 Apptainer (anteriormente Singularity)

Es un gestor de contenerización orientado a sistemas en **desarrollo** ya que utiliza entornos virtuales para generar contenedores no persistentes en base a imágenes. Su uso es similar al de *venv* en python.

No ofrece opciones de persistencia en sus contenedores, por lo que no está aconsejado su uso en sistemas de producción a no ser que sea un caso de uso muy específico ya que el ciclo de vida es el de la imagen y todos los datos desaparecen en cuanto se para el contenedor.

Se trata de un sistema *daemonles* orientado a evadir los puntos críticos que tiene el controlador de Docker facilitando la paralelización en los entornos, algo que es bastante más complicado e ineficiente en entornos de servidor como Docker.

Se originó para dar soporte al desarrollo y para la **programación científica** y emplea un *chroot* del directorio y la carga de nuevas librerías en el entorno virtual para simular el sistema aislado.

4.2.2.3 Podman

Se trata de otro gestor similar al anterior. También funciona de modo *daemonless* y **no** dispone de un proceso controlador de todo el sistema. Además no requiere permisos elevados para su ejecución.

Es un gestor de *pods* totalmente compatible con los comandos de docker. Mantiene los mismos comandos a pesar de que no dispone de las herramientas extra como *docker-swarm* o *docker-compose* ni de su orquestador de contenedores.

4.2.2.4 Docker-compose

También es necesario hablar de los orquestadores de contenedores, vamos a comentar **docker-compose** que es el que hemos utilizado ya que es una herramienta integrada de Docker. Vemos en la Figura 4.5 cómo se sitúa esta herramienta por encima de docker.

Es una herramienta desarrollada para definir y compartir aplicaciones que requieren el uso de más de un contenedor de forma simultanea y conectada.

Se apoya en el uso de ficheros **YAML** con la especificación de los distintos servicios (contenedores) que se crearán así como sus características, variables de entorno, redes de conexión,... Con esto y junto con los ficheros Dockerfile de creación de las imágenes, genera el conjunto de las acciones para la generación de una aplicación completa.

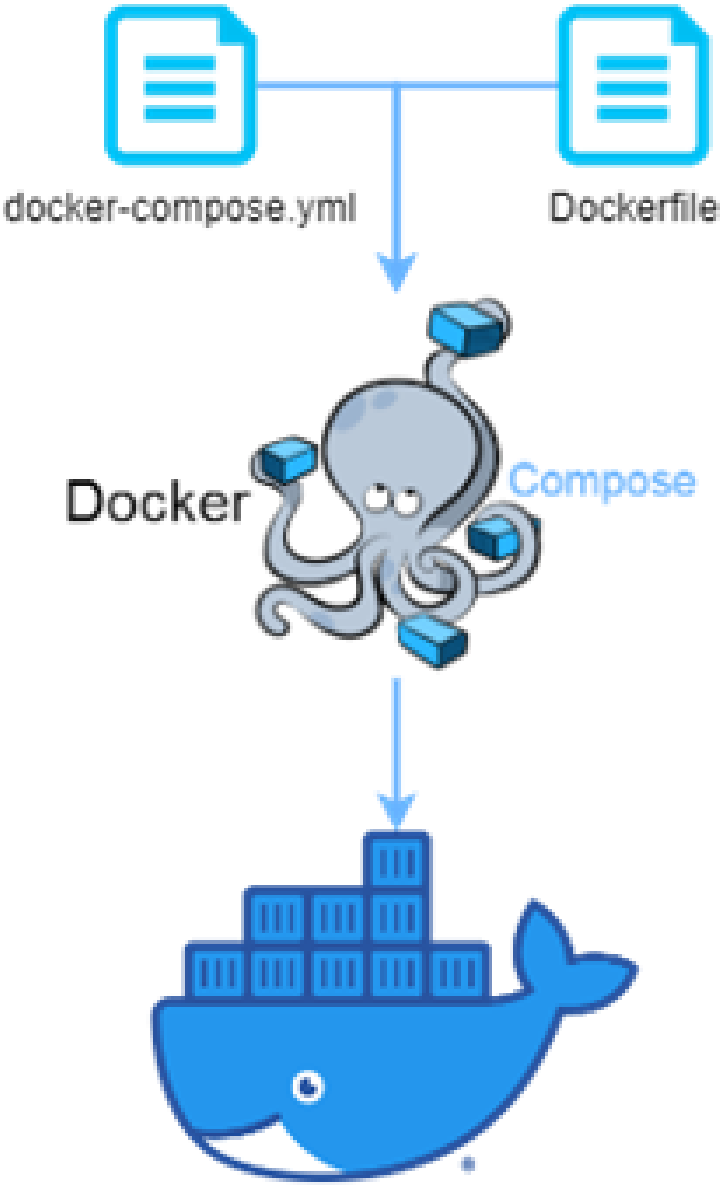


Figura 4.5: **Docker y docker-compose. [doc,]**

Los diferentes entornos se aíslan utilizando el nombre de la aplicación para funcionar

de forma independiente, sin interacción indeseada entre ellos a no ser que se haya indicado de forma explícita. Esta indicación se puede realizar mediante la comunicación utilizando puertos TCP.

Docker-compose almacena en caché la configuración utilizada para crear cada uno de los contenedores, por lo que únicamente recreará aquellos cuya configuración haya sido modificada. De esta forma acelera los procesos de inicialización de los entornos al dejar intactos aquellos contenedores que no han recibido ninguna modificación.

4.3 Software de servicios

4.3.1 Python

Lenguaje de programación de alto nivel que fue desarrollado entre finales de la década de los 80 y principios de los 90 y cuyo código fuente fue publicado por primera vez en 1991 por *Guido van Rossum* en el CWI de la Universidad de Amsterdam, en los Países Bajos.[pyt,]

Es un lenguaje **multiparadigma** que soporta la programación orientada a objetos, la programación imperativa y la programación funcional.

Python se basa en un tipado dinámico, siento un lenguaje fuertemente tipado ya que permite que una variable cambie el tipo de datos que contiene a lo largo del código pero que necesita realizar una conversión explícita de tipos a la hora de realizar ciertas operaciones que pertenecen a un tipo de datos concreto.

Además, permite el desarrollo de nuevos módulos o funciones tanto en el propio Python como con código escrito en C/C++ y al ser un lenguaje interpretado, dispone de una consola en modo interactivo en la cual se puede ir ejecutando código y es especialmente útil durante el desarrollo.

El lenguaje se diseñó en base a una serie de principios llamados la filosofía de python y que se incluyen dentro del código y se muestran al ejecutar '*import this*' y se muestra en la Figura 4.6 .

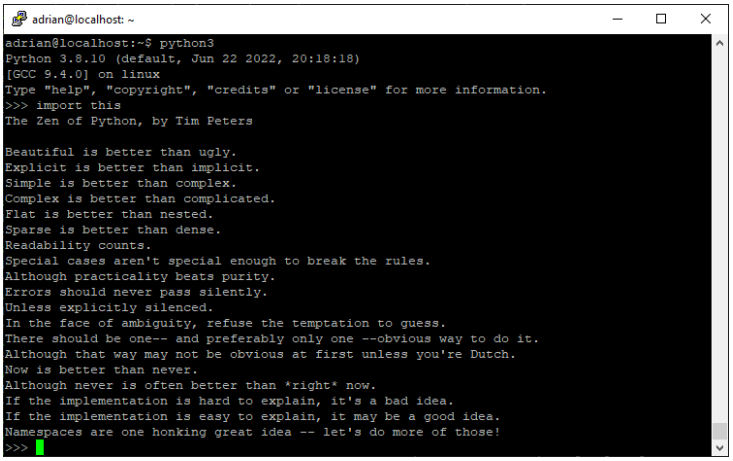


Figura 4.6: **Filosofía de Python mostrada desde el código.**

Por ello se considera un lenguaje de muy fácil comprensión y lectura con un formato muy ordenado y que utiliza la indentación obligatoria para organizar los bloques funcionales en contraposición con otros lenguajes como C o Java en los que los bloques se integran entre llaves ($\{\}$) y la indentación, aunque ayuda a entender el código al leerlo, no es obligatoria.

Además del código y los paquetes básicos inclídos, Python soporta también la importación (e instalación) de diversos paquetes diseñados para funcionalidades específicas y se pueden descargar de forma sencilla desde el repositorio de software [PyP,] Pypi y realizar su instalación mediante *pip*.

La versión de python más utilizada ha sido posiblemente Python 2.7, o cualquiera de las versiones menores dentro de la 2.x aunque desde 2020 se encuentra obsoleta a favor de la versión 3.0, dentro de la cual la última versión estable es la 3.10.5 que tiene cambios bastante importantes respecto a la versión 2 y por lo que no permite la retrocompatibilidad entre versiones.

4.3.2 Bases de datos relacionales

Las bases de datos SQL se basan en el modelo relacional y son las más extendidas actualmente. Se basan en un RDBMS (o sistema gestor de bases de datos) y utilizan el lenguaje de consulta sql (lenguaje de consulta estructurada) para la administración y la recuperación de la información en este tipo de bases de datos.

Este tipo de datos consta de tablas totalmente estructuradas, con un esquema definido y relaciones entre las tablas mediante la utilización de un conjunto de claves primarias, que identifican a la propia tabla al ser únicas para cada registro, y de claves foráneas que permiten la identificación del registro de una tabla (mediante la utilización de su clave primaria) desde los registros de otra tabla.

4.3.2.1 PostgreSQL

PostgreSQL es una implementación de una base de datos relacional. Se trata de un RDBMS de código abierto orientado a objetos que cumple con todas las características de una base de datos relacional incluyendo las transacciones **ACID**:

- **Atomicidad:** Las transacciones en estas bases de datos deben de ser completas. Es decir que cuando una transacción requiere de una sucesión de pasos para llevarla a cabo, o se realizan todos o no se realizará ninguno.
- **Consistencia:** Implica que cualquier transacción que se realice en la base de datos, llevará a la misma de un estado válido a otro estado válido.
- **Aislamiento:** Asegura que la realización de distintas transacciones sobre la misma información, son totalmente independientes y no interfieren entre ellas. Por lo tanto, varias operaciones sobre la misma información no pueden generar error ni afectarse mutuamente.
- **Durabilidad:** Esta característica garantiza que cualquier transacción realizada sobre la base de datos será persistente y no desaparecerá, aunque se de algún fallo en el sistema.

Ofrece una alta concurrencia y permite accesos múltiples a una misma tabla de forma simultanea sin bloqueos siempre y cuándo únicamente uno de esos accesos sea de escritura. Lleva muchos años (se lanzó en 1996) y dispone de una gran comunidad debida a su amplia utilización.

Dispone de una gran cantidad de tipos de datos integrados y de funciones para la representación de texto de tamaño fijo o variable (incluso tamaño 'ilimitado'), para la representación númerica con diferente representación y precisión y otros tipos de datos complejos como los arrays, las direcciones o los UUID. Existen, incluso, ciertas librerías para la utilización de figuras geométricas y funciones para el tratamiento de estos datos de forma eficaz.

Por último, nos permite la generación y el uso de triggers para desencadenar ciertos procedimientos almacenados, llamados funciones, mediante diferentes lenguajes de programación entre los que se encuentran PL/PgSQL y otros basados en lenguajes de programación como **PL/Python**.

4.3.3 Bases de datos NoSQL

En cuanto a las bases de datos NoSQL, se trata de un modelo que difiere considerablemente del basado en RDBMS en el que básicamente no se utiliza el lenguaje sql debido a que no se requiere una estructura definida como tablas y no utilizan secuencias del tipo JOIN ni se basan en ACID (características tradicionales de las bases de datos relacionales).

A pesar de esto, estas bases de datos sí que pueden consistir en tablas con una mayor o menor estructuración y utilizar lenguajes 'tipo SQL'. Sin embargo, se suelen utilizar diferentes modelos de almacenamiento de los objetos y según esos modelos es cómo se suele diferenciar y categorizar a estos sistemas de bases de datos.

- **Orientadas a documentos:** Se trata de bases de datos que organizan los datos en medios medianamente estructurados en forma similar a un documento. Normalmente se utilizan codificaciones estándar como YAML, JSON (o BSON, que es un tipo de JSON binario) o XML. De esta categoría, vamos a hablar de **MongoDB**.
- **Orientadas a Columnas:** Son bases de datos que se organizan en tablas, de forma similar a las bases de datos SQL, pero agrupando las celdas mediante columnas en lugar de hacerlo mediante filas cómo se realiza en las bases de datos tradicionales. Permiten la utilización de lenguajes similares a sql y resultan óptimas para consultas de tipo map-reduce. En este caso resaltamos **Cassandra**.
- Además, existen otros modelos, entre los que se encuentran las bases de datos **orientadas a grafos** o las bases de datos **clave/valor**.

Es imprescindible identificar el tipo de dato que se va a adquirir para realizar una selección de la BD óptima para cada problema. A continuación, se ha realizado una breve descripción de las opciones de las bases de datos NoSQL. Según el Teorema CAP que se muestra en la Figura 4.7 :

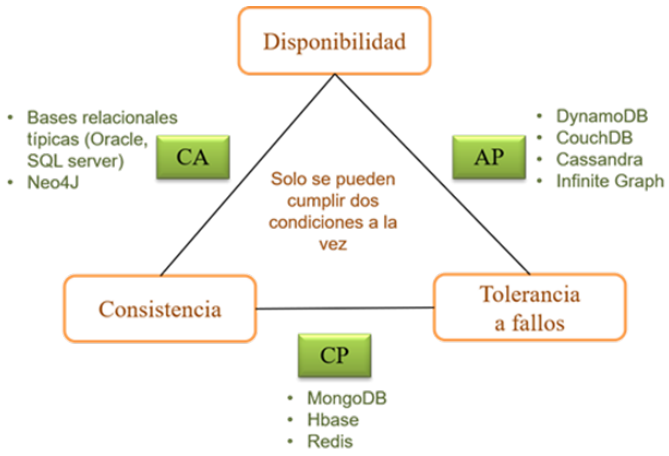


Figura 4.7: **Teorema CAP.**

A la hora de seleccionar una base de datos ha de tenerse en cuenta el teorema CAP o teorema Brewer [Alférez Zamora, 2016], este dice que en sistemas distribuidos es imposible garantizar a la vez: consistencia, disponibilidad y tolerancia a particiones (Consistency-Availability-Partition Tolerance). Veamos qué son estas características:

- **Consistencia:** al realizar una consulta o inserción siempre se tiene que recibir la misma información, con independencia del nodo o servidor que procese la petición. Un cambio en el sistema se debe efectuar si y sólo si se efectúa en todos los nodos de forma que estos datos nunca difieran.

- **Disponibilidad:** todos los clientes puedan leer y escribir, aunque se haya caído uno de los nodos. Todas las consultas deben obtener una respuesta, aunque sea inconsistente.
- **Tolerancia a particiones:** a veces traducido como tolerancia a fallos. Los sistemas distribuidos pueden estar divididos en particiones (generalmente de forma geográfica). El sistema tiene que seguir funcionando, aunque existan fallos o caídas parciales .

Para ser escalables y distribuidas se siguen distintos métodos, por lo que no todas cumplen los mismos puntos del teorema CAP, esto según la imagen Figura 4.7 se resumen en las siguientes opciones:

- **AP:** garantizan disponibilidad y tolerancia a particiones, pero no la consistencia, al menos de forma total. Algunas de ellas consiguen una consistencia parcial a través de la replicación y la verificación.
- **CP:** garantizan consistencia y tolerancia a particiones. Para lograr la consistencia y replicar los datos a través de los nodos, sacrifican la disponibilidad.
- **CA:** garantizan consistencia y disponibilidad, pero tienen problemas con la tolerancia a particiones. Este problema lo suelen gestionar replicando los datos.

Algunos de estos sistemas NoSQL pueden configurarse para cambiar su comportamiento.

Por ejemplo, **MongoDB** es **CP** por defecto. Pero también podemos configurar el nivel de consistencia, eligiendo el número de nodos a los que se replicarán los datos o elegir si se pueden leer datos de los nodos secundarios (en MongoDB solo hay un servidor principal, que es el único que acepta inserciones o modificaciones) sacrificando consistencia a favor de la disponibilidad.

Por tanto, además de pensar en el tipo de base de datos NoSQL que se mejor se adapta a nuestro modelo de datos, también tendremos que pensar en su funcionamiento. Así podremos conseguir que nuestra aplicación funcione de la mejor manera posible. Disponemos de distintas categorías:

- **Orientadas a documentos:** Son aquellas que gestionan datos semi estructurados. Es decir, documentos. Estos datos son almacenados en algún formato estándar como XML, JSON o BSON. Para hacernos una idea un documento suele ser algo parecido a:

Código 4.1: **Ejemplo de documento de mongoDB.**

```
1 {
2   "Name": "ITCL Burgos",
3   "Type": "Tech. Cent.",
4   "Categories": [
5     {
```

```
6         "Title": "Desarrollo",
7         "Articles": 89
8       },
9       {
10        "Title": "Formacion",
11        "Articles": 45
12      }
13    ]
14  }
```

Son las bases de datos NoSQL más versátiles. Se pueden utilizar en gran cantidad de proyectos, incluyendo muchos que tradicionalmente funcionarían sobre bases de datos relacionales. En esta categoría encontramos:

  - **MongoDB:** Probablemente la base de datos NoSQL más famosa del momento. Con más de 150 millones de dólares en financiación es una da las *startups* más prometedoras. Algunas compañías que actualmente utilizan MongoDB son **Foursquare** o **eBay**.
  - **CouchDB:** Es la base de datos orientada a documentos de Apache. Una de sus interesantes características es que los datos son accesibles a través de unqa API REST.

- **Orientadas a columnas:** Este tipo de bases de datos están pensadas para realizar consultas y agregaciones sobre grandes cantidades de datos. Funcionan de forma parecida a las bases de datos relacionales, pero almacenando columnas de datos en lugar de registros. En esta categoría encontramos:

  - **Cassandra:** Incluida en esta sección, aunque en realidad sigue un modelo híbrido entre orientada a columnas y clave-valor. Es utilizada por **Facebook** y **Twitter** (aunque dejaron de usarla para almacenar tweets).
  - **HBase:** Escrita en Java y mantenida por el Proyecto Hadoop de Apache, se utiliza para procesar grandes cantidades de datos.

- **clave-valor:** Estas son las más sencillas de entender. Simplemente guardan tuplas que contienen una clave y su valor. Cuando se quiere recuperar un dato, simplemente se busca por su clave y se recupera el valor. En esta categoría encontramos:

  - **DynamoDB:** Desarrollada por Amazon, es una opción de almacenaje que podemos usar desde los Amazon Web Services.

- **Orientadas a grafos:** Basadas en la teoría de grafos utilizan nodos y aristas para representar los datos almacenados. Son muy útiles para guardar información en modelos con muchas relaciones, como redes y conexiones sociales. En esta categoría encontramos:

  - **Infinite Graph:** Escrita en Java y C++ por la compañía Objectivity. Tiene dos modelos de licenciamiento: uno gratuito y otro de pago.
  - **Neo4j:** Base de datos de código abierto, escrita en Java por la compañía Neo Technology.

Después del estudio, se ha decidido utilizar PostgreSQL que nos permite utilizar los códigos ya existentes para la creación de las nuevas bases de datos y, principalmente, porque aporta la posibilidad de crear los procedimientos almacenados a través de *triggers* configurados en la propia base de datos.

Al poder utilizar estos procedimientos almacenados programados en Python a través de PL/Python.

4.3.4 crond

Crond es un *daemon* de Linux que hace las veces de un programador de tareas, permitiendo la ejecución automática de ciertos comandos o scripts cuándo se cumplen ciertas condiciones temporales o de reinicio de los equipos.

Este *'demonio'* viene incluído con la mayoría de las distribuciones de Linux para el control de tareas recurrentes [Keller, 1999] y busca en los directorios `/var/spool/cron/crontabs` and `/etc/cron.d` y en el fichero `/etc/crontab` buscando todas las órdenes para la ejecución. El demonio entonces se ejecuta cada minuto para ejecutar todas las instrucciones que correspondan y se *'duerme'* hasta el comienzo del siguiente minuto tras lanzar la ejecución de todas ellas. También se pueden establecer ficheros *'crontab'* en rutas personalizadas.

Para configurar este demonio, simplemente debemos modificar estos ficheros añadiendo las instrucciones deseadas.Para ello en vez de abrir el fichero para su edición, existen los comandos `crontab -e` y `crontab -l` para editar y consultar nuestro fichero personal de cron respectivamente evitando posibles errores que corrompan los ficheros.

```
1  # Edit this file to introduce tasks to be run by cron.
2  #
3  # Each task to run has to be defined through a single line
4  # indicating with different fields when the task will be run
5  # and what command to run for the task
6  #
7  # To define the time you can provide concrete values for
8  # minute (m), hour (h), day of month (dom), month (mon),
9  # and day of week (dow) or use '*' in these fields (for 'any').\#
10 # Notice that tasks will be started based on the cron's system
11 # daemon's notion of time and timezones.
12 #
13 # Output of the crontab jobs (including errors) is sent through
14 # email to the user the crontab file belongs to (unless redirected).
15 #
16 # For example, you can run a backup of all your user accounts
17 # at 5 a.m every week with:
18 # 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
19 #
20 # For more information see the manual pages of crontab(5) and cron(8)
21 #
22 MAILTO=""
23 # m h  dom mon dow   command
24 0 0 * * * rm ~/df.txt
25 5 * * * * df>>~/df.txt
26 0 * * * * nodetool flush
27 5 * * * * /user/bigdata/ejecutarAlgoritmos.sh
28 @reboot /home/bigdata/alertaReinicio.py
```

Código 4.2: **Ejemplo de crontab.**

El Código 4.3.4 muestra un ejemplo de un fichero crontab en el que todos los días a las 00:00 borra el fichero ~/df.txt y cada hora al minuto 05 ejecuta el comando df redirigiendo la salida al fichero mencionado anteriormente. Además cada hora ejecutará el comando *'nodetool flush'* a la hora en punto y el comando */user/bigdata/ejecutarAlgoritmos.sh'* a los 5 minutos. Por último tenemos una instrucción que ejecutará */home/bigdata/alerta-Reinicio.py* cada vez que se reinicie la máquina.

4.3.5 mosquitto (MQTT)

MQTT es un protocolo de paso de mensajes muy utilizado que se basa en el patrón del publicador-suscriptor (*Publisher-subscriber*) que se basa en la conexión de los clientes a un servidor intermedio (llamado *bróker*) que se encarga de distribuir los mensajes a los suscriptores.

Se trata de un protocolo muy sencillo y ligero, que permite una comunicación de alta velocidad a bajo coste para realizar conexiones de máquina a máquina (M2M) en base a TCP/IP, manteniendo la conexión abierta para todas las transmisiones en lugar de abrir y cerrar conexión en cada envío, lo que agiliza las comunicaciones.

MQTT funciona manteniendo un registro de clientes que se conectan al servidor, en el cual se dispone de temas llamados *topics* con una estructura jerárquica. Un publicador envía mensajes al bróker con un determinado topic y el servidor entonces se los envía a cada uno de los clientes que estén suscritos a ese topic.

Los topic, al ser jerárquicos permiten suscribirse a 1 o varios temas que cuelguen de la misma raíz para recibir todos los mensajes o sólo parte de ellos, que se envíen a un topic hijo.

Es un sistema fácilmente escalable y con pocas limitaciones, ya que basta con conectar un nuevo publicador o suscriptor al bróker para ello. Además tiene desacoplamiento total de los clientes ya que al no enviar los mensajes directamente entre ellos, no importa quién sea el que envía los mensajes o el que los recibe. La gestión de esta tarea es competencia del bróker y los clientes se encargan de enviar o recibir los mensajes independientemente.

Este sistema de comunicación permite añadir nuevos publicadores o suscriptores según sean las necesidades en cada momento en caso de que se necesiten enviar datos de nuevos dispositivos o procesarlos de diferentes modos utilizando distintos suscriptores.

Se destaca la ligereza y sencillez de estos mensajes, que requieren de un ancho de banda mínimo y un bajo consumo de energía, especialmente útil para utilizar mediante redes inalámbricas y dispositivos alimentados con batería.

MQTT cuenta con un mecanismo de Calidad de servicio (QoS) en 3 niveles:

- **Nivel 0:** El mensaje se envía una vez, por lo que no se garantiza su entrega en caso

de fallo.

- **Nivel 1:** El mensaje se envía y se espera una respuesta de entrega. En caso de fallo, se garantiza su entrega aunque puede dar lugar a recepciones repetidas del mismo mensaje.
- **Nivel 2:** Se garantiza que la entrega del mensaje se va a realizar una única vez, lo que supone un envío mayor de mensajes para garantizarlo.

Se puede elegir el QoS tanto al publicar como al recibir, sin embargo si el publicador o el suscriptor utilizan un QoS inferior al otro, será este el que se aplique en la práctica. Los QoS superiores, conllevan una mayor carga del sistema debido a la necesidad de de una mayor comunicación para garantizar la entrega de los mensajes. En caso de sistemas deterministas, un QoS de nivel 1 podría ser optimo ya que se garantiza la entrega del mensaje al menos una vez y, aunque se reciba más veces el mismo mensaje, no habría problema ya que el resultado sería el mismo.

Además, MQTT permite la utilización de sistemas de seguridad para el envío de los mensajes. En primer lugar permite la utilización de usuarios y contraseñas para permitir la conexión, así como la autenticación mediante certificado y la encriptación mediante SSL/TLS en transporte, si bien es verdad que el uso de estos sistemas implican una sobrecarga en el sistema debido a que se deben intercambiar un mayor número de mensajes.

Se han realizado una serie de pruebas y de configuraciones en este estudio para probar los distintos modos de funcionamiento de seguridad y también se han realizado accesos no autenticados de diferente modo para ver la estabilidad del sistema y los errores que arroja en caso de conexiones inválidas.

- **Sistema sin cifrado TLS con usuario y contraseña**
  - **conexión válida:** La siguiente prueba se ha realizado estableciendo la conexión entre el dispositivo y el bróker con el usuario y la contraseña correctos, al ser todos los parámetros correctos, se establece la conexión. Este ejemplo lo podemos ver en la Figura 4.8

```
root@mqtt:/# tail -n 0 -f /var/log/mosquitto/mosquitto.log
1661420740: New connection from 95.127.8.234:45109 on port 1883.
1661420740: New client connected from 95.127.8.234:45109 as clienttest (p1, c1, k60, u'itcl').
```

Figura 4.8: **Prueba de conexión sin seguridad con usuario y contraseña correctos.**

  - **Usuario o contraseña incorrectos:** La siguiente prueba se ha realizado estableciendo la conexión entre el dispositivo y el bróker con el usuario y la contraseña incorrectos. Al no ser los correctos, se niega la conexión. Se muestra el ejemplo de este caso como Figura 4.9

```
root@mqtt:/# tail -n 0 -f /var/log/mosquitto/mosquitto.log
1661420597: New connection from 37.10.235.117:46054 on port 1883.
1661420597: Client clienttest disconnected, not authorised.
```

Figura 4.9: **Prueba de conexión sin seguridad con usuario y contraseña incorrectos.**

- **Sin aportar credenciales:** La siguiente prueba se ha realizado estableciendo la conexión entre el dispositivo y el bróker con el usuario y la contraseña incorrectos. Al no enviar ni el usuario ni la contraseña, no se permite establecer la conexión. Vemos el mensaje recibido como Figura 4.10

```
root@mqtt:/# tail -n 0 -f /var/log/mosquitto/mosquitto.log
1661420965: New connection from 95.127.181.252:45528 on port 1883.
1661420965: Client clienttest disconnected, not authorised.
```

Figura 4.10: **Prueba de conexión sin seguridad sin enviar el usuario y contraseña.**

- **Intentando conexión con SSL/TLS:** La siguiente prueba se ha realizado estableciendo una conexión segura TLS entre el dispositivo y el bróker. Al no coincidir el tipo de conexión que se está intentando establecer, el bróker indica que los paquetes enviados están mal formados y niega la conexión como se muestra en la Figura 4.11

```
root@mqtt:/# tail -n 0 -f /var/log/mosquitto/mosquitto.log
1661420965: New connection from 95.127.181.252:45528 on port 1883.
1661420965: Client clienttest disconnected, not authorised.
```

Figura 4.11: **Prueba de conexión sin seguridad intentando establecer conexión segura.**

- **Sistema con cifrado TLS sin usuario y contraseña**
  - **Conexión válida:** La siguiente prueba se ha realizado estableciendo una conexión segura ssl entre el dispositivo y el bróker con los tres certificados correctos. Esta prueba se muestra en la siguiente Figura 4.12

```
root@mqtt:/# tail -n 0 -f /var/log/mosquitto/mosquitto.log
1661419649: New connection from 95.124.178.115:13643 on port 8883.
1661419649: Client clienttest already connected, closing old connection.
1661419649: New client connected from 95.124.178.115:13643 as clienttest (p1, c1, k60).
```

Figura 4.12: **Prueba de conexión con seguridad enviando los certificados correctos.**

- **Intentando conexión con certificados incorrectos:** La siguiente prueba se ha realizado estableciendo una conexión segura ssl entre el dispositivo y el bróker con algún de los certificados incorrectos. El mensaje de respuesta es el de la Figura 4.13

```
root@mqtt:/# tail -n 0 -f /var/log/mosquitto/mosquitto.log
1661419890: Client clienttest has exceeded timeout, disconnecting.
```

Figura 4.13: **Prueba de conexión con seguridad enviando los certificados incorrectos.**

– **Intentando conexión sin certificado:** La siguiente prueba se ha realizado estableciendo una conexión insegura y sin enviar los certificados. Al no aportar certificado, el serivor nos responde como en la Figura 4.14

```
root@mqtt:/# tail -n 0 -f /var/log/mosquitto/mosquitto.log
1661420149: New connection from 95.127.5.242:26149 on port 8883.
1661420149: OpenSSL Error[0]: error:1408F10B:SSL routines:ssl3_get_record:wrong version number
1661420149: Client <unknown> disconnected: Protocol error.
```

Figura 4.14: **Prueba de conexión con seguridad intentando establecer una conexión no segura.**

– **Intentando conexión con certificado y credenciales:** La siguiente prueba se ha realizado estableciendo una conexión segura, con los certificados y añadiendo el usuario y la contraseña. Dado que cuando se verifica una conexión segura no se tiene en cuenta ni se comprueba el usuario y la contraseña, se establece la conexión. En la Figura 4.15 se ve como sí que se completa la conexión.

```
root@mqtt:/# tail -n 0 -f /var/log/mosquitto/mosquitto.log
1661420319: New connection from 80.29.200.215:44309 on port 8883.
1661420319: New client connected from 80.29.200.215:44309 as clienttest (p1, c1, k60, u'itcl').
```

Figura 4.15: **Prueba de conexión con seguridad enviando el usuario y la contraseña.**

Finalmente, ya que se ejecuta todo en local y no nos importa tanto la seguridad en ese punto ya que no hay ningún tipo de comunicación con el exterior al ser un sistema independiente aislado, hemos utilizado un servicio de mosquitto sin cifrado para reducir la carga de datos en los mensajes y reducir la complejidad en las comunicaciones además de la latencia que aumenta al utilizar tanto el cifrado como el QoS. Por lo tanto, el servidor se ha configurado sin certificados pero con usuario y contraseña ya que nos es suficiente para el objetivo.

Hasta el momento se ha estudiado una colección de software y se ha decidido lo que se utilizará para realizar la implementación. En cuanto a bases de datos nos quedamos con PostgreSQL por la utilidad que nos aportan los procedimientos almacenador frente a NoSQL. Una vez completada esta fase, se procede a realizar la implementación de los 3 proyectos tal y como se describe en el siguiente punto de la memoria.

Capítulo 5

Implementación Realizada

5.1 FPGA as a Service

Se ha dessarrollado un sistema que permite la automatización del proceso de inferencia a través de un sistema FPGA AVNET Xilinx Ultra96-V2 con un modelo de desarrollo propio mediante el uso de diferentes servicios que se detallan a continuación. Además de la FPGA, se ha utilizado un sistema Beckhoff C6930-0060 conectado a través de Ethernet y en el que se han desplegado tanto una base de datos PostgreSQL y dos máquinas virtuales que completan el servicio.

En la Figura 5.1 se muestran las 2 máquinas virtuales utilizadas y en la Figura 5.2 se muestra una captura del pgAdmin en la que se encuentran los 2 servidores conectados:

- **localdb:** Base de datos local, montada sobre windows en el Beckhoff. Aquí se insertan los datos recibidos desde el sistema existente y contiene todas las tablas que se utilizan en el resto de sistemas de monitorización externos al alcance de este proyecto. Sobre esta base de datos se crearán los procedimientos almacenados que se encargarán de transferir los datos a la otra base de datos y a la FPGA. Esto se muestra en la Figura 5.2
- **keeper:** Base de datos desplegada a través de contenerización (Docker) sobre la máquina virtual llamada *controller-machine*, cuyo contenido se muestra en la Figura 5.3 . Aquí se registran datos de configuración, que mantiene la versión y el modelo utilizado, la tabla de medidas '*measurements*' que contiene los datos registrados en la base de datos local y 2 tablas de alertas, una generada y recibida en la base de datos local y otra ('*ai_alerts*') en la que se guardan los datos de alerta generados por la red neuronal ejecutada en la FPGA.

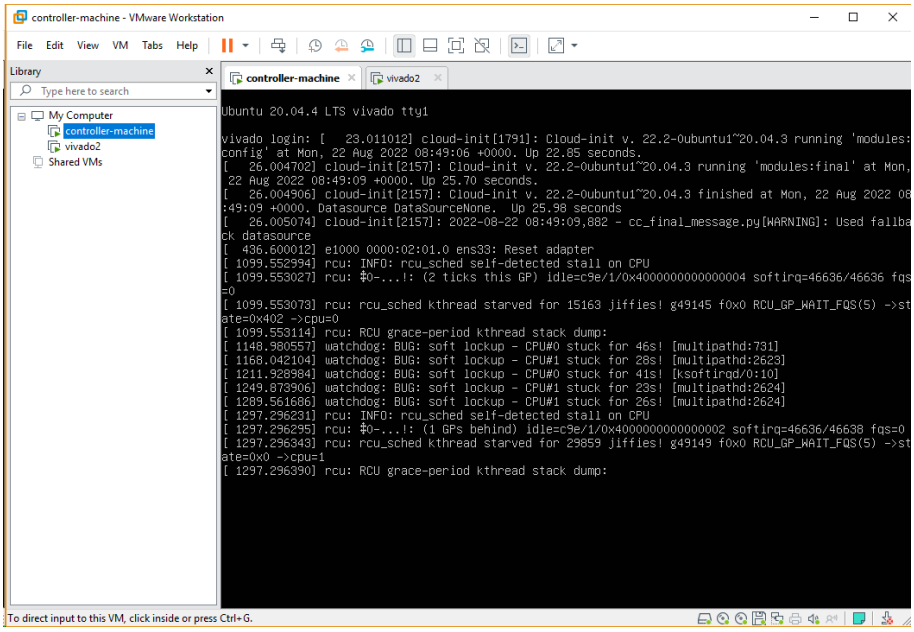


Figura 5.1: **Servicios virtualizados en VMWare.**

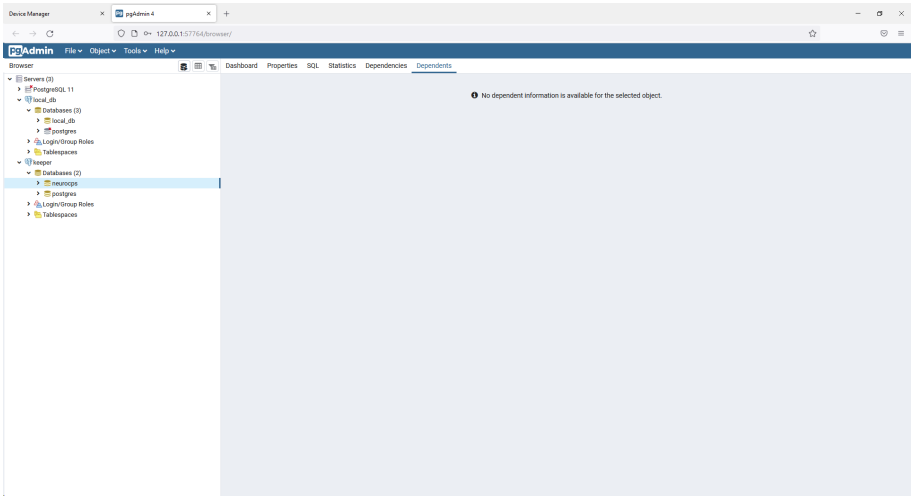


Figura 5.2: **Visión de las bases de datos de Postgres a través de PGAdmin.**

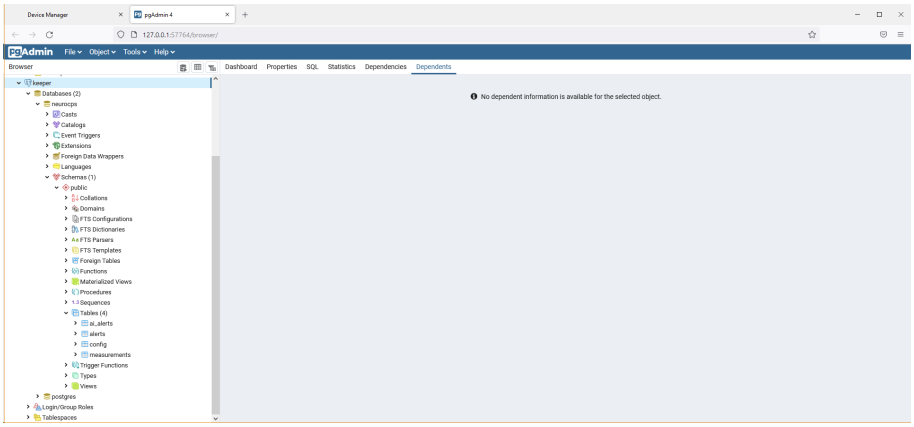


Figura 5.3: **Visión de la base de datos *keeper* de Postgres a través de PGAdmin.**

5.1.1 Sistema físico

El sistema físico se compone del sistema Beckhoff conectado a otro sistema Windows que está a su vez conectado a la red de la empresa y que se utilizará como puente para permitir el acceso mediante RDP (*Remote Desktop Protocol*) bloqueando el resto de tráfico de red, la FPGA ZUS+ Ultra96-V2 se conecta tanto por USB (interfaz microUSB) como por conexión de red ethernet RJ45 con el dispositivo Beckhoff para realizar la comunicación IP y las conexiones ssh para configuración y monitorización del sistema. Esta conexión de red física entre la FPGA y el beckhoff está puenteada con la red virtual VMWare de forma que haya conexión directa entre ésta y las máquinas virtuales de Vivado y el controlador dockerizado.

Ya que la FPGA no dispone de conexión ethernet, se ha utilizado un adaptador USB-ethernet *tp-link UE300* que conecta el puerto USB 3.0 para poder realizar la conexión con el Beckhoff.

En la siguiente Figura 5.4 se ve un esuqema de cómo se realiza la conexión entre estos sistemas.

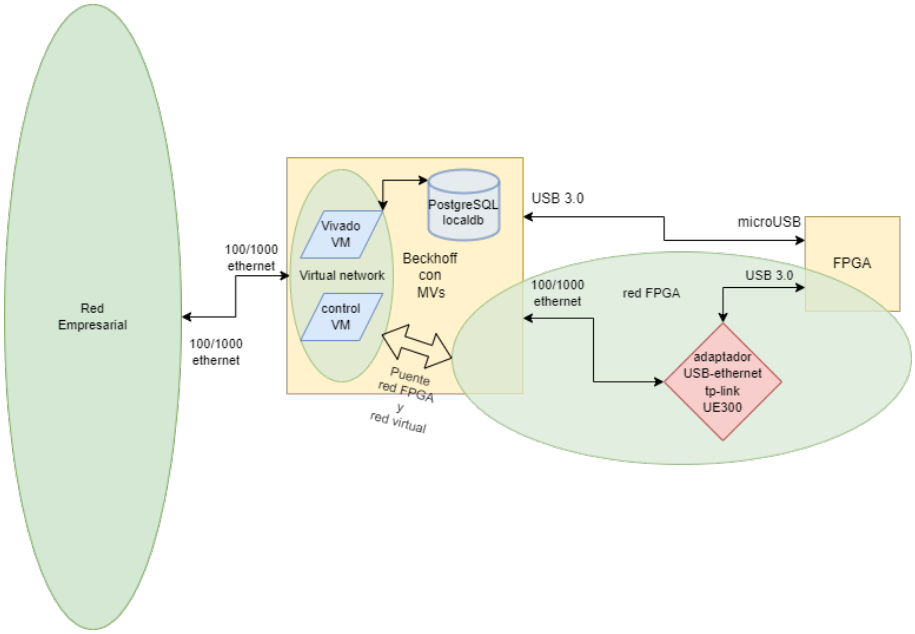


Figura 5.4: **Esquema de los sistemas físicos y virtuales y la interconexión entre ellos.**

5.1.2 Sistemas virtualizados

De entre las tecnologías de virtualización de hardware estudiadas hemos decidido utilizar el sistema de VMWare Workstation en la versión 15.0.0 ya que nos parece el sistema que más se ajusta a nuestras necesidades y que ofrece un mejor rendimiento que otras plataformas como VirtualBox.

Se han generado 2 sistemas virtualizados, ambos con Sección 4.1.2.3 que contienen tanto el sistema de controladores de la red, dockerizado, como el paquete de software *Vivado*, de Xilinx Inc.

5.1.2.1 Controlador dockerizado

La máquina virtual llamada *controller-machine* contiene un sistema de Docker. Los contenedores arrancados se muestran en la Figura 5.5 y el contenido del docker-compose.yaml sería el de la Figura 5.6 . Se dispone de una imagen de esta máquina virtual para realizar la importación sobre VMWare Workstation con los códigos y la configuración realizada.

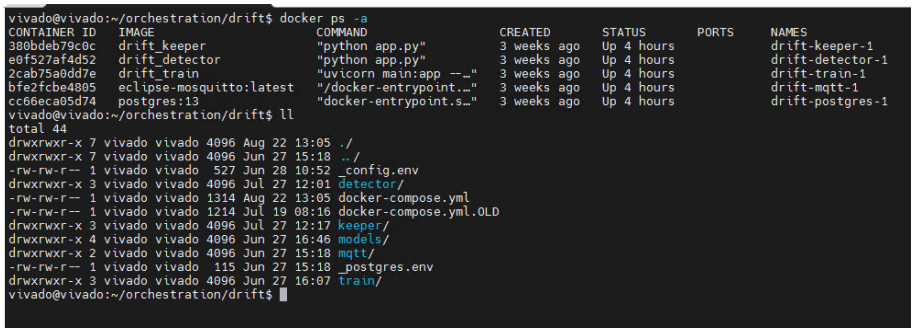


Figura 5.5: **Contenedores desplegados y contenidos del directorio de trabajo.**

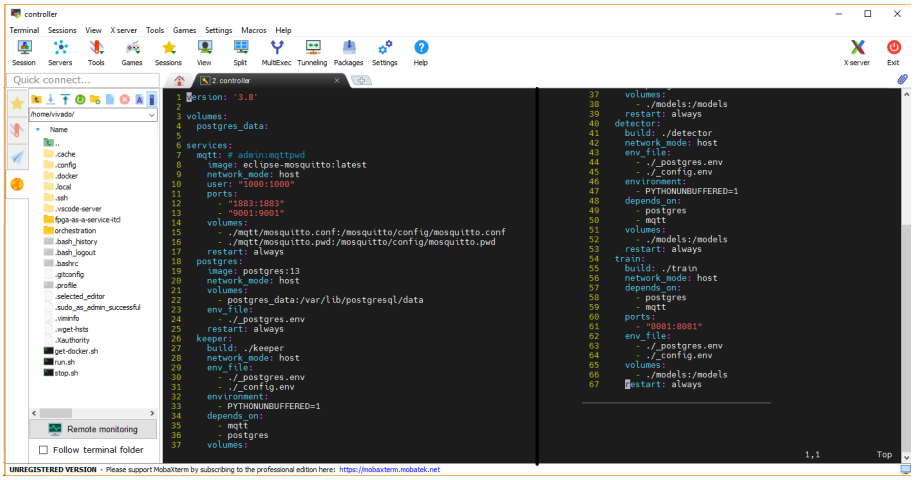


Figura 5.6: **Contenido del *docker-compose.yaml* de despliegue de los servicios del controlador.**

5.1.2.2 Vivado SDK

Por otro lado disponemos de la máquina virtual llamada *vivado2*. En ella se han instalado una serie de librerías dependencias para la instalación del SDK de Xilinx Vivado. Esta ejecución se demora bastante tiempo ya que la descarga del sdk es bastante pesada (*21.4Gb*). Por ello disponemos directamente de las imágenes de las máquinas virtuales para importar directamente sobre el VMWare Workstation con la configuración realizada para el despliegue del servicio.

En el directorio finn disponemos del servicio de API de uvicorn. Disponemos de un fichero de requisitos (*requirements.txt*) que se detalla a continuación como Código 5.1.2.2 .

```
1 fastapi
2 uvicorn[standard]
3 finn-base
4 tensorflow
5 toposort
6 jinja2
7 onnx
8 pandas
```

Código 5.1: **Contenido del fichero *requirements.txt* de uvicorn.**

```
vivado@vivado:~/orchestration/finn$ ll
total 40
drwxrwxr-x 6 vivado vivado 4096 Aug 23 12:22 ./
drwxrwxr-x 7 vivado vivado 4096 Jun 27 16:00 ../
drwxrwxr-x 3 vivado vivado 4096 Jun 27 16:00 lstm/
-rw-rw-r-- 1 vivado vivado 7869 Jun 27 16:00 main.py
drwxrwxr-x 3 vivado vivado 4096 Jun 27 16:00 models/
-rw------- 1 vivado vivado 2203 Jul 18 07:26 nohup.out
drwxrwxr-x 2 vivado vivado 4096 Jun 27 16:20 __pycache__/
-rw-rw-r-- 1 vivado vivado   75 Jun 27 16:22 requirements.txt
drwxrwxr-x 2 vivado vivado 4096 Jun 27 16:00 templates/
vivado@vivado:~/orchestration/finn$
```

Figura 5.7: **Contenido del directorio de la API de Uvicorn.**

Además de este fichero de configuración, se encuentra el fichero *main.py* junto con el resto de directorios auxiliares tan y como se observa en la Figura 5.7 . Las funciones definidas en el fichero se enumeran, aunque no se muestra su implementación debido a la confidencialidad, a continuación en el Código 5.1.2.2 .

```
from lstm.src.lstmtohlsparams import TFToHLSParams
import datetime
import pathlib
from jinja2 import Environment, FileSystemLoader
import tensorflow as tf
import subprocess
import hashlib
import shutil
from fastapi.responses import HTMLResponse
from fastapi import FastAPI, File, UploadFile
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'


# FastAPI
app = FastAPI()


############################
#   PROJECT PARAMETERS   #
############################

ROOT_DIR = str(pathlib.Path().resolve())
env = Environment(loader=FileSystemLoader('./templates'))

HLS_FILE = ROOT_DIR + '/lstm/hls.tcl'
RTL_FILE = ROOT_DIR + '/lstm/rtl.tcl'

\# Vivado HLS TCL variables
\# sourcery skip: use-fstring-for-concatenation
HLS_PROJECT_NAME = 'hls_proj'
TOP_FUNCTION = 'topLevel_LSTM'
TARGET_DEVICE = 'xczu3eg-sbva484-1-i'
TARGET_BOARD = 'avnet.com:ultra96v2:part0:1.2'

HLS_PROJECT_DIR = 'lstm_hls_proj'
VIVADO_HLS_DIR = ROOT_DIR + '/lstm/' + HLS_PROJECT_DIR
VIVADO_HLS_PROJ = VIVADO_HLS_DIR + '/' + HLS_PROJECT_NAME

HLS_LIB = ROOT_DIR + '/lstm/src/library/hls'
HOST_LIB = ROOT_DIR + '/lstm/src/library/host'
PLAIN_INCL = ROOT_DIR + '/lstm/src/network/plain'
```

```
43 PARAMS_INCL = ROOT_DIR + '/lstm/src/network/plain/parameters'
44
45 DESIGN_FILE = ROOT_DIR + '/lstm/src/network/plain/top.cpp'
46 TESTBENCH_FILE = ROOT_DIR + '/lstm/src/network/main_tb.cpp'
47
48 \# Vivado TCL variables
49 RTL_PROJECT_NAME = 'lstm_rtl_proj'
50 RTL_PROJECT_PATH = ROOT_DIR + '/lstm/' + RTL_PROJECT_NAME
51 RTL_PROJECT_LANG = 'VHDL'
52 IP_PATH = VIVADO_HLS_PROJ + '/sol1/impl/ip'
53
54
55 def generate_tcl(hls_template: str, rtl_template: str):
56     """Generate HLS and RTL TCL files from templates using Jinja2"""
57
58
59 def generate_params(model_path: str):
60     """Generate parameters for the LSTM model"""
61
62
63 @app.get("/")
64 def read_root():
65     """Show basic parameters for debugging purposes"""
66
67
68 @app.post("/generate-overlay/{checksum}/")
69 def generate_overlay(checksum: str, file: UploadFile = File(...)):
70     """Generate HLS Overlay to be loaded into PYNQ"""
71
72
73
```

Código 5.2: **Código del fichero *main.py* de la API de uvicorn para Vivado.**

5.1.2.3 Configuración del arranque automático de los servicios virtualizados

Estos sistemas se ejecutan al arrancar las máquinas de forma automática, de forma que cuándo se reinicie el servidor Beckhoff en el que se encuentran las máquinas, se vuelvan a ejecutar todos los servicios. Para ello se han realizado varias tareas:

1. Arranque automático de las máquinas virtuales.

   Para que las máquinas virtuales en VMWare se inicien automáticamente al iniciar el sistema operativo anfitrión, debemos crear un script para realizar dicho auto-arranque. Para ello ejecutamos el comando *shell:startup* para que se nos abra la carpeta donde se ubican los scripts que se ejecutarán al iniciar el sistema operativo y creamos en esta ubicación un fichero llamado '*vmautostart.cmd*' con el siguiente contenido:

```
"C:\Program Files (x86)\VMware\VMware Workstation\vmrun.exe" start "C:\
    Users\Administrator\Documents\Virtual Machines\vivado\vivado.vmx"
"C:\Program Files (x86)\VMware\VMware Workstation\vmrun.exe" start "C:\
    Users\Administrator\Documents\Virtual Machines\vivado2\vivado2.vmx"
```

Código 5.3: **Código de autoarranque de las máquinas virtuales.**

   Una vez guardado este fichero, a partir del siguiente reinicio se ejecutará el script automáticamente y se arrancarán las MV inmediatamente tras el inicio del sistema.

2. Arranque automático de los contenedores dockerizados.

   Para el arranque de los contenedores, una vez iniciada la máquina virtual hemos utilizado crond utilizando la instrucción *@reboot* que permite que se ejecute una orden cuándo se inicie la máquina, para realizar esta configuración en el crontab debemos abrir el editor del fichero de configuración de cron con el comando *shell:startup* y añadir el siguiente contenido:

```
1 @reboot bash /home/vivado/run.sh
```

***Código 5.4:* Crontab del *controller-machine*.**

   Una vez configurado el crond, pasamos a editar el fichero /home/vivado/run.sh para añadir el script de arranque:

```
1 fecha=`date`
2 echo Se ha ejecutado el script de autoarranque en la fecha: $fecha >> /home/
      vivado/run.log
3 echo "------------------" >> /home/vivado/run.log
4 /home/vivado/.local/bin/uvicorn main:app --host 0.0.0.0 --port 8082 --app-
      dir /home/vivado/orchestration/finn/
```

***Código 5.5:* script *run.sh* para el arranque de los contenedores.**

   Una vez realizado esto, tendremos configurado el sistema para que cada vez que se reinicie, lo cuál se realizará al inicial el sistema operativo del servidor por lo configurado en el punto anterior, ejecutará el script */home/vivado/run.sh* para lanzar los contenedores del controlador.

3. Arranque automático de la API en el Servidor de Vivado.

   De forma similar a lo realizado en el punto anterior, sobre la MV de Vivado haremos una configuración similar configurando el crontab con el mismo contenido ya que cada imagen incluye el fichero run.sh dentro la carpeta principal del usuario *vivado*.

   El contenido del fichero run.sh es la que varía en contenido y el código en este caso se muestra como Código 6.1.2 .

```
1 fecha=`date`
2 echo Se ha ejecutado el script de autoarranque en la fecha: $fecha >> /home/
      vivado/run.log
3 echo "------------------" >> /home/vivado/run.log
4 docker compose -f /home/vivado/orchestration/drift/docker-compose.yml up -d
```

***Código 5.6:* script *run.sh* para el arranque de los contenedores.**

5.1.3 Edge FPGA

En la ZUS+ Ultra96-v2 tenemos una aplicación, la Figura 5.8 muestra el contenido de su directorio, desarrollada en python que permitirá escuchar en un *topic* de mosquitto (**MQTT**) para ejecutar la inferencia sobre los datos cada vez que se reciben los paquetes

correspondientes. Esta aplicación dispone de un fichero de requisitos (*requirements.txt*) que se detalla a continuación como Código 5.1.3 .

```
paho-mqtt
sqlalchemy
```

Código 5.7: **Contenido del fichero *requirements.txt* de la FPGA.**

```
xilinx@pynq:~/orchestration/edge$ ll
total 44
drwxrwxr-x 4 xilinx xilinx  4096 Aug 22 21:27 ./
drwxrwxr-x 7 xilinx xilinx  4096 Jun 27 16:12 ../
drwxrwxr-x 3 xilinx xilinx  4096 Jul 23 10:05 common/
-rw-rw-r-- 1 xilinx xilinx   372 Jul 11 01:16 .env
-rw-rw-r-- 1 xilinx xilinx 17525 Aug 19 18:58 main.py
drwxrwxr-x 6 xilinx xilinx  4096 Jun 27 17:15 models/
-rw------- 1 root   root       0 Jul 15 12:56 nohup.out
-rw-r--r-- 1 root   root       0 Jul 15 12:58 output.txt
-rw-rw-r-- 1 xilinx xilinx    36 Jun 27 16:12 requirements.txt
xilinx@pynq:~/orchestration/edge$
```

Figura 5.8: **Contenido del directorio de la aplicación en python de la FPGA.**

El contenido del fichero *main.py* que contiene las funciones utilizadas para establecer las conexiones y realizar los procesos importantes de ejecución de inferencia sobre los modelos se muestra, sin detallar su implementación, en el Código 5.1.3

```
import os
import math
import json
import numpy as np
import paho.mqtt.client as mqtt
from pathlib import Path
from common.mqtt import on_connect, on_log, on_publish, on_subscribe
from sqlalchemy.orm import sessionmaker
from sqlalchemy import inspect
from common.models import get_engine, Config
from dotenv import load_dotenv
from pynq import Overlay
from pynq import DefaultIP
from pynq import allocate
from pynq import Clocks
import time
import numpy as np


load_dotenv()


# Postgres Settings
"""Check if postgreSQL (in controller-machine) is up and running or wait till
    for it to be up"""
session = None
for _ in range(5):
    try:
        Session = sessionmaker(bind=get_engine())
        session = Session()
        print("SUCCESSFULY CONNECTED to server")
        break
    except:
        print("POSTGRES DISCONNECTED, trying again")
        time.sleep(5)

```

```
if session == None:
    exit()



# Inference Settings
ROOT_PATH = Path.cwd()
last_model = session.query(Config).filter(
    Config.variable == 'last_edge_model').first()
MODEL_NAME = last_model.value
OVERLAY_PATH = f'{str(ROOT_PATH)}/models/{MODEL_NAME}/{MODEL_NAME}.bit'

# MQTT Settings
MQTT_HOST = os.getenv('CONTROLLER_HOST')
MQTT_PORT = int(os.getenv('MQTT_PORT'))
MQTT_USER = os.getenv('MQTT_USER')
MQTT_PWD = os.getenv('MQTT_PASSWORD')

# SQL Settings
SQL_HOST = os.getenv('HOST')
SQL_PORT = int(os.getenv('POSTGRES_PORT'))
SQL_USER = os.getenv('POSTGRES_USER')
SQL_PWD = os.getenv('POSTGRES_PASSWORD')
SQL_DB = os.getenv('POSTGRES_DB')


TELEMETRY_TOPIC = os.getenv('TELEMETRY_TOPIC')
MODEL_CHANGE_TOPIC = os.getenv('CHANGE_MODEL_TOPIC')
INFERENCE_TOPIC = os.getenv('INFERENCE_TOPIC')

# LSTM Accelerator Settings
IP_NAME = 'topLevel_LSTM'

DATAWIDTH = 32
NUM_IN_COLS = 180
NUM_OUT_COLS = 1
NUM_LAYERS = 1
NUM_FEATURES = 4
INPUT_WIDTH = 16
NUM_NEURONS = 1
OUTPUT_WIDTH = 16
BATCH = 2000

ADDRESS_MAP = {
    'STATUS': 0x00,
    'CURR_LYR': 0x10,
    'NUM_COLS': 0x18,
    'NUM_BYTES': 0x20,
    'IN_BUFF': 0x28,
    'OUT_BUFF': 0x34
}

BIT_START = (1 << 0)
BIT_DONE = (1 << 1)
BIT_IDLE = (1 << 2)

BUFF_DATAWIDTH = DATAWIDTH
BUFF_DATAWIDTH_BYTES = BUFF_DATAWIDTH // 8
NP_DTYPE = f'u{str(BUFF_DATAWIDTH_BYTES)}'
IN_BUFF_RAT = math.ceil(NUM_FEATURES*INPUT_WIDTH/BUFF_DATAWIDTH)
OUT_BUFF_RAT = math.ceil(NUM_NEURONS*OUTPUT_WIDTH/BUFF_DATAWIDTH)
NUM_IN_BYTES = NUM_IN_COLS * IN_BUFF_RAT * BUFF_DATAWIDTH_BYTES
out_type = f'u{str(OUTPUT_WIDTH // 8)}'


def quantize(quant_input, bit_width):
    """ Quantization function to signed fixed point """

def dequantize(quant_input, bit_width):
    """ Dequantization function from signed fixed point """
```

```
106
107 def pack_col(column):
108     """ Takes a whole input column of size NUM_FEATURES
109
110 class LSTMLayer(DefaultIP):
111     """" Python driver for the LSTM IP Core """
112
113     bindto = [Overlay(OVERLAY_PATH).ip_dict[IP_NAME]['type']]
114
115     def __init__(self, description):
116         super().__init__(description)
117         self._exe_time = None
118         self._raw_output = np.zeros((NUM_OUT_COLS, NUM_NEURONS))
119         self._in_buffer = allocate(
120             shape=(NUM_IN_COLS*IN_BUFF_RAT,), dtype=NP_DTYPE)
121         self._out_buffer = allocate(
122             shape=(NUM_OUT_COLS*OUT_BUFF_RAT,), dtype=NP_DTYPE)
123         self.write(ADDRESS_MAP['CURR_LYR'], 0)
124         self.write(ADDRESS_MAP['NUM_COLS'], NUM_IN_COLS)
125         self.write(ADDRESS_MAP['NUM_BYTES'], NUM_IN_BYTES)
126         self.write(ADDRESS_MAP['IN_BUFF'], self._in_buffer.physical_address)
127         self.write(ADDRESS_MAP['OUT_BUFF'], self._out_buffer.physical_address)
128
129     @property
130     def runtime(self):
131         return self._exe_time
132
133     @property
134     def raw_output(self):
135         return self._raw_output.astype(out_type)
136
137     def store_input(self, data):
138         """ Takes an input array of shape (NUM_IN_COLS, NUM_FEATURES) and stores
139
140     def recover_output(self):
141         """ Takes the accelerator output buffer, which is packed in
    BUFF_DATAWIDTH sized
142
143     def exec_accel(self):
144
145     def run_model(self, v_input, num_layer=0):
146
147
148 def NormalizeData(data: np.ndarray) -> np.ndarray:
149     """Apply a Min-Max normalization to the data
150
151 def on_message(client, obj, msg):
152     """Define on_message actions depending on topic which message is posted"""
153
154 if __name__ == '__main__':
155     # Pynq Bitstream Loading
156     overlay = Overlay(OVERLAY_PATH)
157
158     # MQTT Client
159     client = mqtt.Client(clean_session=True)
160
161     client.on_message = on_message
162     client.on_connect = on_connect
163     client.on_publish = on_publish
164     client.on_subscribe = on_subscribe
165     client.on_log = on_log
166
167     client.username_pw_set("admin", "mqttpwd")
168     # client.will_set(topic=MASTER_TOPIC, payload="Disconnected",
169     #                 qos=0, retain=False)
170     client.connect(host=MQTT_HOST, port=MQTT_PORT, keepalive=60)
171
172     client.subscribe(MODEL_CHANGE_TOPIC, qos=0)
173     client.subscribe(TELEMETRY_TOPIC, qos=0)
174
```

```
175     client.loop_forever()
176
177
```

Código 5.8: **Código del fichero *main.py* de la aplicación de Python en la FPGA**

5.1.3.1 Arranque automático de la FPGA

La FPGA está puenteada de forma que al conectarla a la alimentación, se encienda automáticamente sin necesidad de activarla manualmente.

Así, cuándo se conectan los dispositivos, se iniciarán automáticamente y arrancarán los servicios. En los sistemas virtualizados tal y como se ha comentado anteriormente y en la FPGA mediante la siguiente orden en el crontab:

```
1 @reboot sudo -i bash -c 'source /usr/local/share/pynq-venv/bin/activate && cd /
      home/xilinx/orchestration/edge && python main.py >> /home/xilinx/output.txt'
```

Código 5.9: **Crontab de la FPGA.**

Este crontab ejecutará la activación de un entorno virtual y la ejecución del Código 5.1.3 mencionado anteriormente, el cual comenzará, como se muestra en la función *__main__* del código anterior, la suscripción al broker de mosquitto en los distintos *topics* que hemos definido en las variables de entorno.

Al comienzo del código, comprobamos si existe conexión con el servidor *controller-machine* ya que necesitamos que esté funcionando para poder establecer las conexiones con mosquitto y PostgreSQL mediante el intento de conexión con este último. En caso de que falle la conexión, quedará temporalmente a la espera reintentando dicha conexión periódicamente hasta que sea satisfactoria o finalice el *timeout* y se de por fallida la conexión, en cuyo caso habría que reiniciar el dispositivo.

5.2 Implementación de LSTM en Ultra96

El principal objetivo del proyecto es la implementación de una biblioteca para la implementación de redes neuronales LSTM como kernels de aceleración, mediante la descripción de hardware digital con síntesis de alto nivel (HLS). De igual forma es necesario contar con un framework que facilite la utilización de esta biblioteca en servidores con arquitectura basada en placas aceleradoras Alveo y Versal ACAP.

Antes de llevar estos sistemas a los servidores de producción, se ha realizado un proceso de diseño e implementación de redes neuronales sencillas utilizando HLS para la concepción de un flujo de diseño. Para ello se han utilizado redes neuronales compuestas por capas tipo Fully Connected o Densas. Los algoritmos de inferencia han sido optimizados para obtener un mayor rendimiento utilizando el proceso de cuantización presente en los frameworks de ONNX y TensorFlow Lite. Para la validación de estas implementaciones

se utilizado el flujo acelerado de Vitis para crear un sistema embebido compuesto por el sistema de procesado y un kernel de aceleración que describe el algoritmo de inferencia de este tipo de redes. Este sistema se ha probado en una placa de desarrollo AVNET Xilinx Ultra96-V2.

En la primera aproximación, se ha utilizado la base de datos *MNIST* [mni,] que se suele utilizar para las primeras pruebas de entrenamiento de diferentes sistemas de procesamiento de imágenes. Esta base de datos contiene 60.000 datos para entrenamiento y 10.000 de inferencia de imágenes de tamaño fijo de dígitos escritos a mano, cuyo ejemplo se muestra como Figura 5.9

Figura 5.9: **Ejemplo de imágenes de MNIST.**

Esta primera aproximación de flujo de diseño para la implementación de redes neuronales en hardware reconfigurable, es la base para la implementación de redes LSTM. Comenzado por la varinte no cuantizada del algoritmo de inferencia.

5.3 Instalación de VCK5000 y Alveo U50

5.3.1 Configuración del hardware

Para este proyecto necesitábamos un servidor, según las especificaciones mostradas en Xilinx VCK5000 Versal, que fuera compatible con PCIe Gen3 x16 y por ello utilizamos unos equipos disponibles en un primer momento, sin embargo finalmente daba problemas de compatibilidad y observamos que necesitábamos realizarlo sobre un servidor con PCIe Gen4 x16 por lo que se ha compró el Servidor Final, el cual por problemas de suministro en los componentes no ha llegado a tiempo para poder realizar una instalación completa por lo que se ha quedado este apartado en la fase de investigación sin poder realizar una implementación completa ya que llegado cierto punto en la instalación del software no podíamos avanzar por los problemas de comunicación con las tarjetas.

Para la configuración del hardware en los diferentes servidores ha sido necesario liberar la ranura PCIe Gen3 x16 de la que disponen las placas, ocupada por las RTX en las pruebas realizadas sobre los servidores disponibles y conectar en su lugar la ACAP VCK5000 Versal.

En el primer servidor no tuvimos ningún inconveniente para su instalación y únicamente necesitamos conectar la tarjeta en la ranura correspondiente y la alimentación de 6+8 pines PCIe mientras que en el segundo de los servidores fue necesario retirar el metal exterior de la placa ya que chocaba con el chasis del servidor y realizar la sujección de otro modo al estar doblada parte de la torre del equipo por una mala instalación de la tarjeta gráfica conectada anteriormente.

Para el Subsubsección 3.1.2.3 se estudió la opción de instalarla en modo de refrigeración activa y en modo de refrigeración pasiva, quitando los ventiladores tal y como se indican en en Apéndice A en la guía de hardware de Xilinx [Inc., 2021].

5.3.2 Instalación del software

Una vez preparada a nivel hardware la Versal VCK5000, se procede a realizar la instalación de software del servidor.

Se ha realizado la instalación del sistema operativo Subsubsección 4.1.2.2 ya que con la versión 20.04 no realizaba correctamente la detección de alguna de las interfaces de red. Posteriormente se actualizará a la versión Subsubsección 4.1.2.3 o Subsubsección 4.1.2.4 cuándo este problema esté solucionado.

A continuación, una vez instalado el sistema operativo e instalados y actualizados todos los paquetes para una instalación básica, procedemos a comprobar que la tarjeta está bien conectada y Linux la reconoce. Para ello ejecutamos el comando *lspci -evd 10ee:* y comprobamos la salida mostrada en la Figura 5.10 y observamos que sí que nos detecta la tarjeta pero observamos que no detecta los NUMA node en las etiquetas como sí que lo observamos en la Figura 5.11 qu se ha sacado de un servidor funcional.

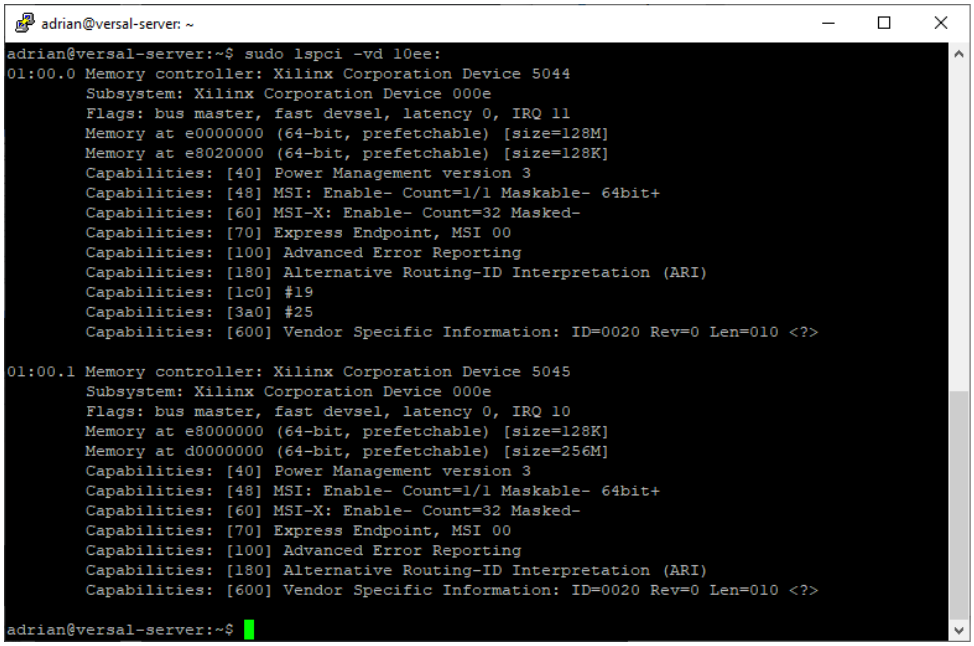


Figura 5.10: **Salida del comando *lspci -evd 10ee:* en nuestras workstation.**

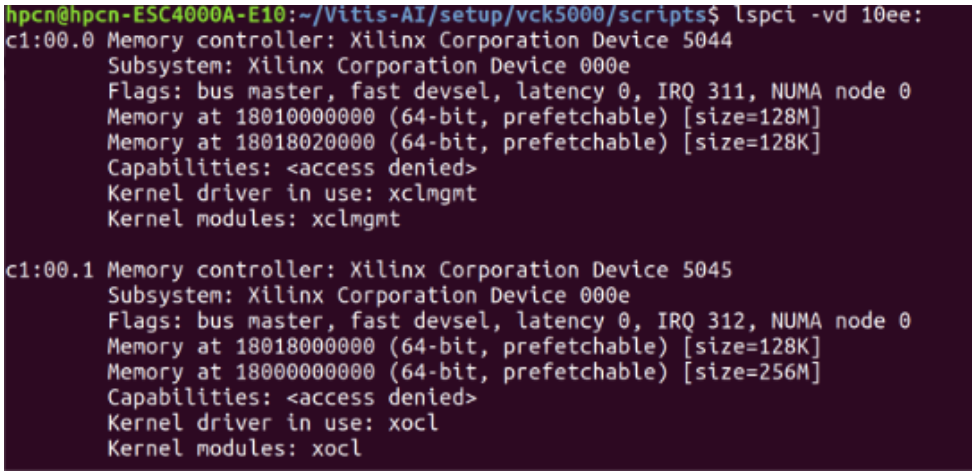


Figura 5.11: **Salida del comando *lspci -evd 10ee:* funcionando.**

Una vez comprobada la conexión, y a pesar de que no detecte cono NUMA node ya que no le dimos mayor importancia en un inicio, se ha procedido con la instalación del software siguiendo los pasos que incluye AMD Xilinx Inc. en su zona privada de usuarios como guía de primeros pasos para la Versal VCK5000.

En primer lugar, nos indica que descarguemos Vitis AI y al ser una tarjeta ES debemos bajar la versión 1.4.1 ya que no funciona con la última versión, lo cual se puede obtener desde el repositorio público en github de Xilinx con el comando *git clone –branch 1.4.1 –recurse-submodules https://github.com/Xilinx/Vitis-AI* y en la el directorio setup/vck5000 se encuentra el fichero *README.md* en el cual se indican los pasos correctos para su instalación, para ello hemos generado el siguiente script Código 5.3.2 con la clonación del proyecto de git y el resto de software requerido y la instalación de todo ello sobre el servidor.

```
1  git clone --branch 1.4.1 --recurse-submodules https://github.com/Xilinx/Vitis-AI
2  cd Vitis-AI/
3  cd setup/vck5000/
4  source ./install.sh
5  cd ~/Vitis-AI
6  wget -O xilinx-vck5000-es1-gen3x16-platform-2-1_all.deb.tar.gz https://www.xilinx.
       com/bin/public/openDownload?filename=xilinx-vck5000-es1-gen3x16-platform-2-1
       _all.deb.tar.gz
7  wget -O xilinx-vck5000-es1-gen3x16-2-202020-1-dev_1-3123623_all.deb https://www.
       xilinx.com/member/forms/download/eula-xef.html?filename=xilinx-vck5000-es1-
       gen3x16-2-202020-1-dev_1-3123623_all.deb
8  mkdir development
9  mv xilinx-vck5000-es1-gen3x16-2-202020-1-dev_1-3123623_all.deb development/
10 mkdir deployment
11 tar -xzvf xilinx-vck5000-es1-gen3x16-platform-2-1_all.deb.tar.gz -C ./deployment/
12 cd deployment
13 sudo apt install ./xilinx-sc-fw-vck5000_4.4.6-2.e1f5e26_all.deb
14 sudo apt install ./xilinx-vck5000-es1-gen3x16-validate_2-3123623_all.deb
15 sudo apt install ./xilinx-vck5000-es1-gen3x16-base_2-3123623_all.deb
16 cd ~/Vitis-AI
```

Código 5.10: **Instalación de Vitis AI en sistemas ubuntu.**

Una vez instalado todo lo anterior ya estaría instalado lo necesario para el funcionamiento de la versal VCK5000 en el servidor y podemos comprobar los datos de la tarjeta con el comando sudo */opt/xilinx/xrt/bin/xbmgmt flash -scan* para ver la versión del firmware de la placa, en caso de no estar actualizado se actualizaría con *sudo /opt/xilinx/xrt/bin/xbmgmt flash -update* aunque nuestro sistema no es compatible con la tarjeta por lo que nos muestra el mensaje '*No card is found!*' que se muestra en la figura Figura 5.12 mientras que el mensaje que deberíamos observar se muestra en la Figura 5.13 que hemos obtenido de un servidor que reconoce correctamente la tarjeta.

```
[adrian@versal-server development]$ sudo /opt/xilinx/xrt/bin/xbmgmt flash --scan
[sudo] password for adrian:
--------------------------------------------------------------------------
Deprecation Warning:
    The given legacy sub-command and/or option has been deprecated
    to be obsoleted in the next release.

    Further information regarding the legacy deprecated sub-commands
    and options along with their mappings to the next generation
    sub-commands and options can be found on the Xilinx Runtime (XRT)
    documentation page:

    https://xilinx.github.io/XRT/master/html/xbtools_map.html

    Please update your scripts and tools to use the next generation
    sub-commands and options.
--------------------------------------------------------------------------
No card is found!
[adrian@versal-server development]$
```

Figura 5.12: **Error *no card found* con el comando *xbmgmt flash -scan***

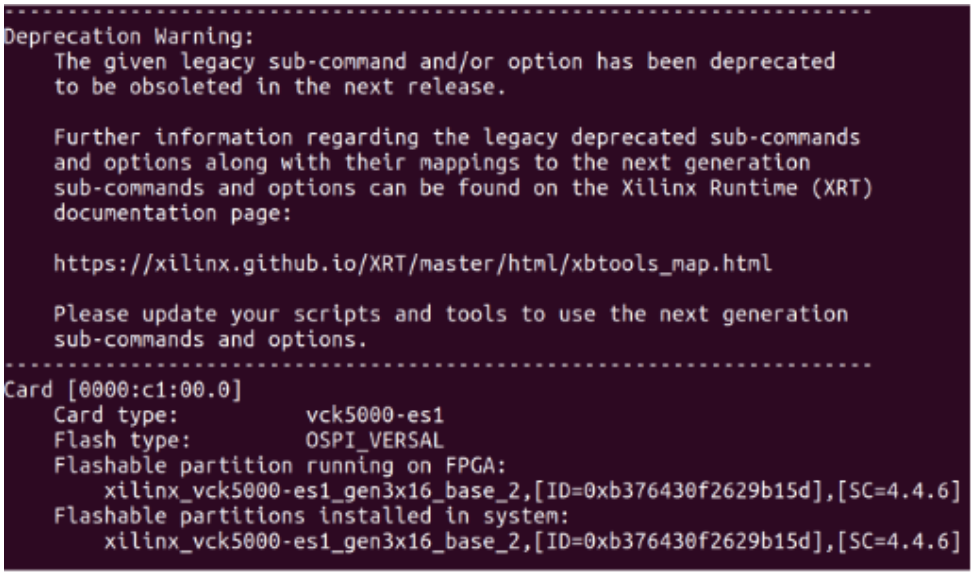


Figura 5.13: **Salida correcta al ejecutar *xbmgmt flash –scan***

Los pasos mencionados anteriormente los hemos seguido en los servidores Servidor 1 y Servidor 2 que no eran compatibles, por lo que no hemos podido continuar a mayores ya que no ha sido posible conseguir el Servidor Final a tiempo por los problemas de suministro.

A pesar de no poder continuar con esto, y avanzando hasta que se reciba el servidor final sobre el que instalar las cosas, hemos continuado instalando docker para desplegar los servicios de Vitis-AI y poder probar el funcionamiento de la aceleradora. Para ello necesitamos ejecutar los siguientes comandos:

```
1 sudo apt update
2 sudo apt install docker-io
3 sudo apt install docker-compose
```

Código 5.11: **Instalación de *docker* y *docker-compose* en sistemas ubuntu.**

A continuación, modificamos el fichero *docker-run.sh* para añadir a la variable **docker_run_params** la siguiente línea

```
1 -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix -v $HOME/.Xauthority:/tmp/.
    Xauthority \
```

Código 5.12: **Permitir X11 en el servidor de la VCK5000 para mostrar imágenes.**

Posteriormente, después de arrancar el docker con el script *docker-run.sh* ejecutamos los siguientes comandos:

```
1 cp /tmp/.Xauthority ~/
```

```
2 sudo chown vitis-ai-user:vitis-ai-group ~/.Xauthority
```

Código 5.13: **Habilitar X11.**

Una vez tenemos una implementación completa de los sistemas, teniendo uno de ellos finalizado y. puesto en producción y los otros 2 en diferentes fases del proceso debido a los problemas encontrados en cada uno de ellos, hemos procedido a realizar una serie de pruebas sobre estos proyectos para evaluar los avances logrados durante el desarrollo.

Capítulo 6

Pruebas Realizadas

A continuación se detallan las diferentes pruebas realizadas sobre los sistemas implementados que se han detallado en el apartado anterior.

6.1 Pruebas realizadas sobre FPGA as a Service

6.1.1 Simulación de ejecución automática

Para realizar esta prueba, se han generado una serie de datos basándose en datos antiguos de los que se disponía para poder simular un envío de datos en tiempo real idéntico al que se realizaría con el sistema en producción pero sin interferir con el mismo.

Se ha generado un dataset en formato *csv* con los datos obtenidos desde la base de datos con aproximadamente 24000 registros, los cuales se tratan en bloques de 4 debido al formato de datos que se trata de 4 registros por segundo para el mismo modelo. Para realizar esta inserción en la base de datos, desde donde se enviarán por mqtt a través del trigger, se ha realizado el CODIGO que lee el csv, compone las sentencias de inserción y se mandan a la base de datos PostgreSQL. El código no se muestra completamente implementado por privacidad.

```
1  import pandas as pd
2  import psycopg2
3  import time
4
5  HOST_IP=''
6  HOST_PORT=''
7  DATABASE=''
8  PG_USER=''
9  PG_PASSWD=''
10
11 df = pd.read_csv ('C:/ITCL/FaaS/dataset_testeo.csv',sep=';',)
12 columnas = df.columns
13 conn = psycopg2.connect(host=HOST_IP, port=HOST_PORT, database=DATABASE, user=
       PG_USER, password=PG_PASSWD)
14 cur = conn.cursor()
15
16 for index, row in df.iterrows():
17 if index % 4 == 0:
18     time.sleep(1)
19 #print(df[[' id ',' auditdatetime ']].iloc[index])
```

```
20 consulta = #Construir consulta INSERT INTO(...) VALUES(...)
21 print(consulta)
22 cur.execute(consulta)
23 conn.commit()
24
25 cur.close()
26 conn.close()
27
28
```

Código 6.1: **Código del script para la inserción de los sql.**

Durante la ejecución de este código, al que hemos metido un tiempo de espera de 1 segundo en cada envío de forma que imite a la realidad, obtenemos los logs en directo en cada uno de los servicios que se están ejecutando.

La Figura 6.1 muestra la salida de log obtenida en el contenedor de mosquitto (MQTT) en la máquina *controller-machine*. Este log nos muestra el registro de las conexiones que se van estableciendo con el servidor de mosquitto. Estas conexiones se corresponden con el dispositivo Beckhoff, son las conexiones que se hacen desde el procedimiento almacenado del servidor PostgreSQL que se conecta para enviar los datos hacia el controlador.

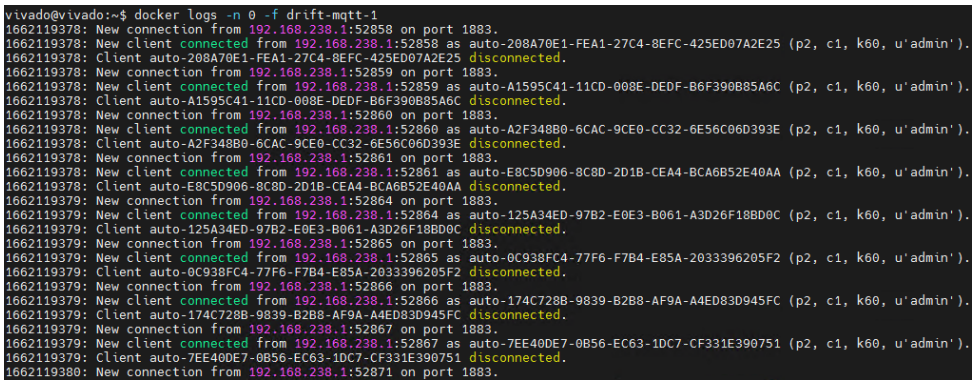


Figura 6.1: **Registro del contenedor de mosquitto durante la simulación de ejecución automática.**

La Figura 6.2 muestra los registros del servicio llamado *keeper*. En esta figura podemos observar los mensajes que le llegan como suscriptor del topic *''beckhoff telemetry''* que son los correspondientes a los envíos por parte del PostgreSQL mostrados en la Figura 6.1 y otros mensajes en el topic *''ultra96v2 inference*', los cuales aparecen cada 4 mensajes de los anteriores. Esto es porque cada 4 mensajes (1 por cada uno de los diferentes sets de datos que se envían cada segundo) es cuando la FPGA realiza la inferencia y devuelve el resultado a través de este segund topic y es recogida por el keeper para guardarlos en la base de datos. Los datos recibidos en *''beckhoff telemetry''* los registra en el PostgreSQL contenerizado en la tabla ***measurements*** mientras que los del topic *''ultra96v2 inference*' que corresponden a los datos de inferencia para la generación de alarmas los guarda en la tabla ***ai_ alerts***. El contenido de ambas tablas lo veremos más adelante.

```
vivado@vivado:~$ docker logs -n 0 -f drift-keeper-1
Received PUBLISH (d0, q0, r0, m0), 'beckhoff/telemetry', ...  (2986 bytes)
Received PUBLISH (d0, q0, r0, m0), 'beckhoff/telemetry', ...  (3043 bytes)
Received PUBLISH (d0, q0, r0, m0), 'beckhoff/telemetry', ...  (2927 bytes)
Received PUBLISH (d0, q0, r0, m0), 'beckhoff/telemetry', ...  (2925 bytes)
Received PUBLISH (d0, q0, r0, m0), 'ultra96v2/inference', ...  (59 bytes)
Received PUBLISH (d0, q0, r0, m0), 'beckhoff/telemetry', ...  (3043 bytes)
Received PUBLISH (d0, q0, r0, m0), 'beckhoff/telemetry', ...  (3040 bytes)
Received PUBLISH (d0, q0, r0, m0), 'beckhoff/telemetry', ...  (3049 bytes)
Received PUBLISH (d0, q0, r0, m0), 'beckhoff/telemetry', ...  (3003 bytes)
Received PUBLISH (d0, q0, r0, m0), 'ultra96v2/inference', ...  (59 bytes)
Received PUBLISH (d0, q0, r0, m0), 'beckhoff/telemetry', ...  (3037 bytes)
Received PUBLISH (d0, q0, r0, m0), 'beckhoff/telemetry', ...  (3048 bytes)
Received PUBLISH (d0, q0, r0, m0), 'beckhoff/telemetry', ...  (3045 bytes)
Received PUBLISH (d0, q0, r0, m0), 'beckhoff/telemetry', ...  (3012 bytes)
Received PUBLISH (d0, q0, r0, m0), 'ultra96v2/inference', ...  (59 bytes)
```

Figura 6.2: **Registro del contenedor con el código python del '*keeper*' durante la simulación de ejecución automática.**

El último punto de las pruebas referente a los sistemas virtualizados es el correspondiente al *detector*. Este servicio se encarga de escuchar los mensajes que llegarían en caso de que las predicciones realizadas por la inferencia en la FPGA causaran falsos positivos o falsos negativos. En caso de que se reciban estos mensajes de forma no esporádica, se realizará un reentrenamiento del modelo, por lo tanto en la Figura 6.3 observamos los mensajes del *keepalive* de mosquitto y una serie de mensajes periódicos que indican que el modelo no se está reentrenando ya que las alertas generadas por la FPGA son correctas.

```
vivado@vivado:~$ docker logs -n 0 -f drift-detector-1
> The model is currently not retraining. Number of threads: 2
> The model is currently not retraining. Number of threads: 2
Sending PINGREQ
Received PINGRESP
> The model is currently not retraining. Number of threads: 2
> The model is currently not retraining. Number of threads: 2
> The model is currently not retraining. Number of threads: 2
> The model is currently not retraining. Number of threads: 2
Sending PINGREQ
Received PINGRESP
> The model is currently not retraining. Number of threads: 2
> The model is currently not retraining. Number of threads: 2
> The model is currently not retraining. Number of threads: 2
> The model is currently not retraining. Number of threads: 2
Sending PINGREQ
Received PINGRESP
```

Figura 6.3: **Registro del contenedor con el código python del '*detector*' durante la simulación de ejecución automática.**

Con esto habríamos acabado de analizar los sistemas virtualizados y lo último en lo referente a los logs sería ya en el sistema físico de la FPGA. En la FPGA AVNET Xilinx Ultra96-V2 existe una aplicación en Python que escucha mosquitto para realizar la inferencia. En la Figura 6.4 se muestran los logs de esta aplicación. En esta figura podemos observar como se reciben mensajes publicados en mosquitto con los datos recibidos del trigger y que se muestran por pantalla, a continuación indica que se está realizando la inferencia y se envían al topic correspondiente para que sean recogidos por el *keeper* e insertados en la base de datos.

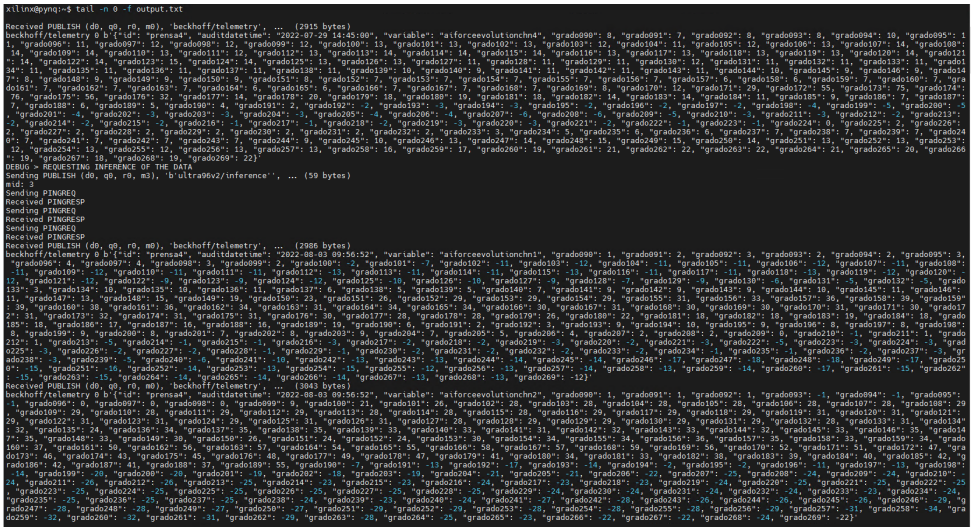


Figura 6.4: **Registro del la ejecución en la FPGA durante la simulación de ejecución automática.**

Todo lo mostrado anteriormente queda reflejado en el PostgreSQL dockerizado tal y como se ve en en la Figura 6.5 en la que se muestran los datos recibidos (la imagen ha sido editada por privacidad en las columnas 'id' y 'variable' y en la Figura 6.6 en la que se muestran las alertas generadas por la inferencia sobre la FPGA en la que todas las predicciones han sido negativas (*false*) ya que no se generaron alertas en esos datos, por lo que son correctas.

	id [PK] text	auditdatetime [PK] timestamp without time zone	variable [PK] text	grado090 integer	grado091 integer	grado092 integer	grado093 integer	grado094 integer	grado095 integer	grado096 integer
1	prueba	2022-08-03 09:56:52	var1	1	1	1	2	3	3	4
2	prueba	2022-08-03 09:56:52	var2	2	2	2	1	1	2	3
3	prueba	2022-08-03 09:56:52	var3	1	1	1	-1	-1	-1	0
4	prueba	2022-08-03 09:56:52	var4	1	2	3	2	2	3	4
5	prueba	2022-08-03 09:48:35	var1	0	0	0	-1	-2	-3	-4
6	prueba	2022-08-03 09:48:35	var2	0	0	-1	-1	0	1	0
7	prueba	2022-08-03 09:48:35	var3	0	-1	0	0	0	0	0
8	prueba	2022-08-03 09:48:35	var4	0	0	2	4	5	5	5
9	prueba	2022-08-03 09:48:30	var1	0	0	1	0	-1	-1	-2
10	prueba	2022-08-03 09:48:30	var2	1	1	1	0	1	2	1
11	prueba	2022-08-03 09:48:30	var3	1	1	1	1	0	0	-1
12	prueba	2022-08-03 09:48:30	var4	1	3	3	3	3	4	4
13	prueba	2022-08-03 09:48:27	var1	1	1	3	3	2	0	0
14	prueba	2022-08-03 09:48:27	var2	1	1	1	1	1	1	1
15	prueba	2022-08-03 09:48:27	var3	1	1	1	1	-1	-2	-2
16	prueba	2022-08-03 09:48:27	var4	1	3	4	4	4	5	4
17	prueba	2022-08-03 09:48:23	var1	0	0	0	0	0	-1	-2
18	prueba	2022-08-03 09:48:23	var2	0	0	0	0	1	2	2
19	prueba	2022-08-03 09:48:23	var3	0	0	0	1	2	2	2
20	prueba	2022-08-03 09:48:23	var4	0	2	4	5	5	5	6
21	prueba	2022-08-03 09:48:20	var1	0	0	1	2	2	2	1
22	prueba	2022-08-03 09:48:20	var2	0	1	1	0	0	1	2
23	prueba	2022-08-03 09:48:20	var3	0	0	0	0	0	-1	-5

Figura 6.5: **Captura de la tabla de medidas recibidas tras la simulación de ejecución automática.**

	auditdatetime [PK] timestamp without time zone	prediction boolean
1	2022-08-03 09:56:52	false
2	2022-08-03 09:48:35	false
3	2022-08-03 09:48:30	false
4	2022-08-03 09:48:27	false
5	2022-08-03 09:48:23	false
6	2022-08-03 09:48:20	false
7	2022-08-03 09:48:18	false
8	2022-08-03 09:48:14	false
9	2022-08-03 09:48:11	false
10	2022-08-03 09:48:08	false
11	2022-08-03 09:48:05	false
12	2022-08-03 09:48:01	false
13	2022-08-03 09:47:58	false
14	2022-08-03 09:47:55	false
15	2022-08-03 09:47:48	false
16	2022-08-03 09:47:41	false
17	2022-08-03 09:47:34	false
18	2022-08-03 09:47:19	false
19	2022-08-03 09:47:09	false
20	2022-08-03 09:46:59	false
21	2022-08-03 09:46:56	false
22	2022-08-03 09:46:53	false
23	2022-08-03 09:46:50	false
24	2022-08-03 09:46:48	false
25	2022-08-03 09:46:44	false

Figura 6.6: **Captura de la tabla de alertas inferidas tras la simulación de ejecución automática.**

6.1.2 Pruebas de ejecución de vivado

Para realizar estas pruebas, que consisten en la generación de un modelo realizando la optimización, compilación y cuantización del mismo para enviarlo a la fpga como modelo a utilizar para las siguientes inferencias. Para ello, desde el directorio `/home/vivado/fpga-asaserviceitcl/finn` ejecutamos el siguiente comando *bash /home/vivado/orchestration/-finn/lstm/generate.sh lstm_hls_proj* que es la equivalencia a lo que ejecutaría desde la REST API desplegada.

```
1  #!/bin/bash
2
3  source /tools/Xilinx/Vivado/2019.1/settings64.sh
4
5  cd ./lstm/$1
6  vivado_hls -f ../hls.tcl
7  vivado -mode tcl -source ../rtl.tcl
```

***Código 6.2:* script *generate.sh* para lanzar el sdk de vivado y generar el proyecto HLS.**

A continuación se muestran ciertas capturas de la salida tras la ejecución, no se muestra completa ya que es excesivamente larga y son salidas repetitivas para los diferentes módulos con el mismo contenido. Se comentan los puntos más interesantes de la ejecución.

En la Figura 6.7 se muestra la ejecución del script de generación del proyecto de Vivado y la ejecución de los comandos que contiene.

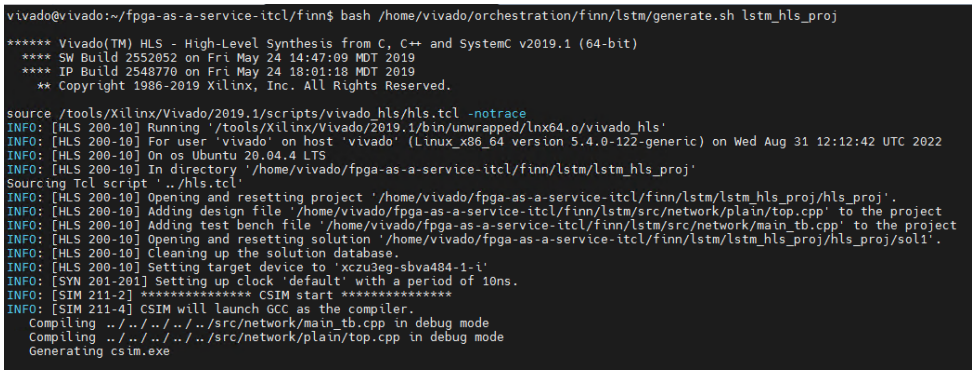


***Figura 6.7:* Captura del lanzamiento del script y llamadas internas.**

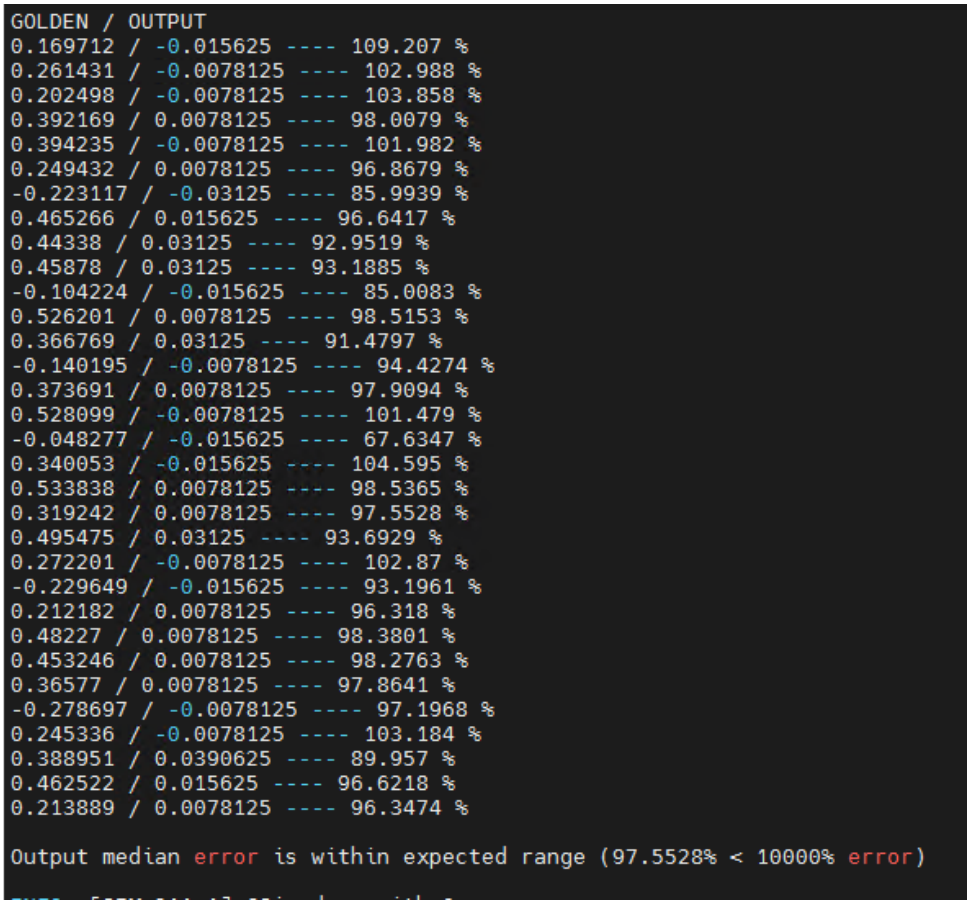


Figura 6.8: **Captura de la secuencia de ejecución de la generación.**

En la Figura 6.8 y en la Figura 6.9 se muestra como el script continúa su ejecución y va procediendo a realizar las transformaciones y operaciones de síntesis correspondientes mientras que en la Figura 6.10 vemos el redimensionado y particionado de los diferentes arrays.

Figura 6.9: **Captura del comienzo de las operaciones de síntesis y transformaciones.**

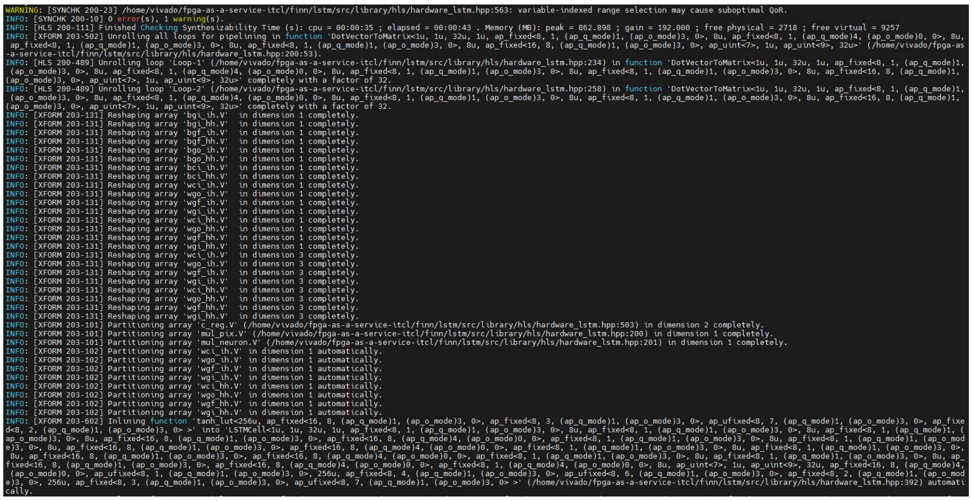


Figura 6.10: **Captura del comienzo del redimensionado y particionado de los arrays.**

6.2 Pruebas realizadas sobre Implementación de LSTM en Ultra96

Con las implementaciones realizadas respecto a la integración de las redes LSTM sobre nuestras ZUS+ Ultra96-V2. Se han realizado pruebas sobre la implementación en Vitis HLS y se han hecho las pruebas mediante el flujo acelerado de Vitis. Para esto se ha utilizado la red que consta de:

- **Capas LSTM:** 1
  - Número de celdas: 30
  - Número de características: 4
  - Tamaño de la ventana de inferencia: 180
- **Capas densas:** 1
  - Número de neuronas: 1

A continuación se muestran los resultados que brinda la herramienta Vitis HLS™. Estos son una **estimación** de los resultados reales que se obtienen mediante la integración del kernel de aceleración al flujo de Vitis™.

6.2.1 Resultados de Vitis HLS™

6.2.1.1 Co-simulación

La Co-simulación C/RTL es una etapa fundamental en el diseño hardware utilizando síntesis de alto nivel (HLS). Este proceso permite verificar que el comportamiento funcional del kernel a nivel de transferencia de registros (RTL) es igual que el descrito en

C/C++. Por lo general si el diseño pasa correctamente el banco de pruebas (*testbench*) utilizado en esta etapa, el kernel implementado en la FPGA lleva a cabo correctamente las funciones para las cuales fue diseñado.

En la Figura 6.11 se observa que la primera aproximación de red neuronal LSTM con HLS funcionalmente es correcta. Para ello se han utilizado las 5 primeras ventanas de datos, comparando la salida de la capa densa del kernel de aceleración con los resultados obtenidos utilizando TensorFlow.

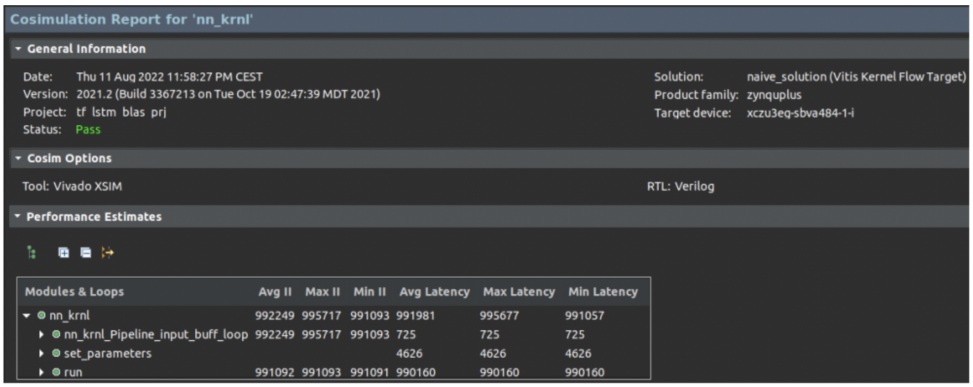


Figura 6.11: **Co-simulación.**

6.2.1.2 Rendimiento

El rendimiento del kernel se puede determinar por el tiempo que demora la red neuronal en realizar la inferencia sobre una ventana de datos, es decir procesar 180x4 datos. Como se puede observar en la Figura 6.12 se estima que el sistema tenga una latencia máxima de 4.812ms para una frecuencia de trabajo de 200MHz.

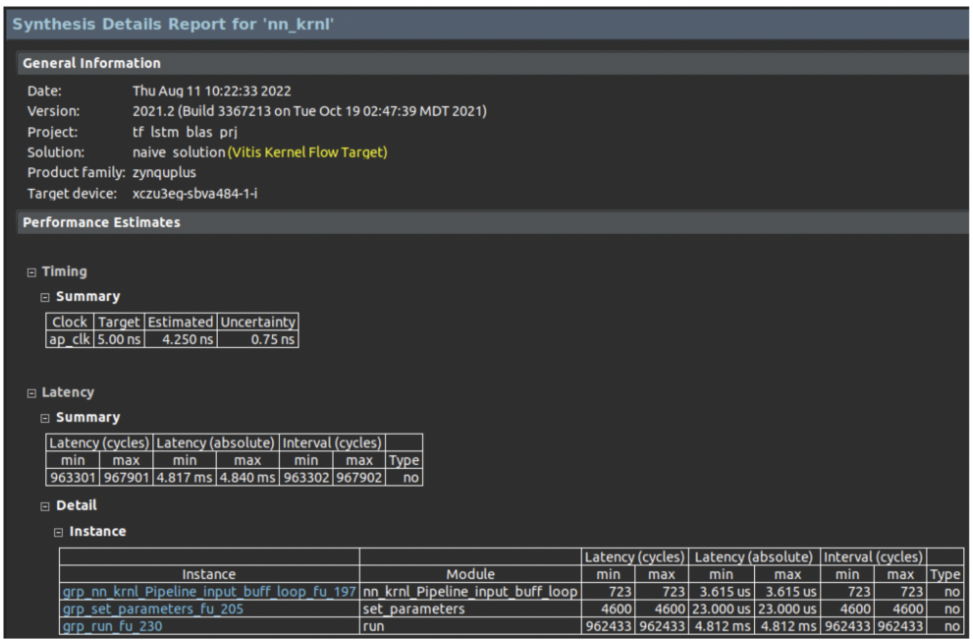


Figura 6.12: **Estimación del rendimiento del kernel de aceleración.**

6.2.1.3 Recursos

Para este rendimiento se obtiene un consumo de recursos relativamente bajo, con 42 % de consumo de LUTs como elemento más crítico. Mientras que el resto de componentes no superan el 20 %. Los detalles se pueden encontrar en la Figura 6.13 .

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	58	69	22518	28303	0
Memory	12	-	192	45	0
Multiplexer	-	-	-	1317	-
Register	-	-	333	-	-
Total	70	69	23043	29669	0
Available	432	360	141120	70560	0
Utilization (%)	16	19	16	42	0

Figura 6.13: **Estimación del consumo de recursos del kernel de aceleración.**

6.2.2 Resultados de flujo acelerado de Vitis™

A continuación se presentan los resultados obtenidos con la integración del kernel de aceleración al flujo acelerado de Vitis™. En la Figura 6.14 se muestra un diagrama del sistema completo con la integración del kernel de aceleración.

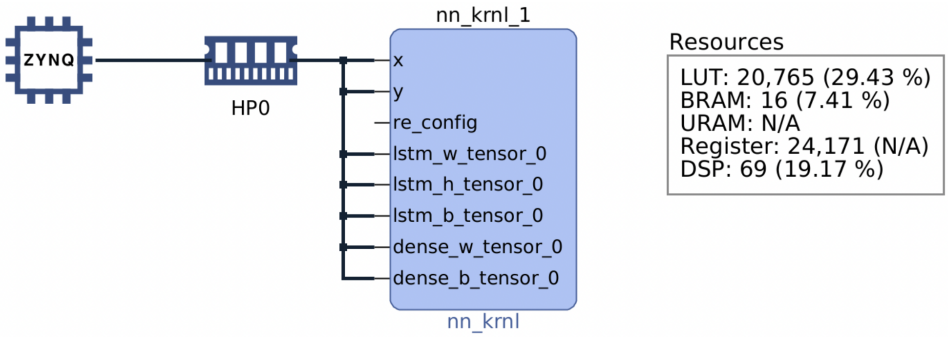
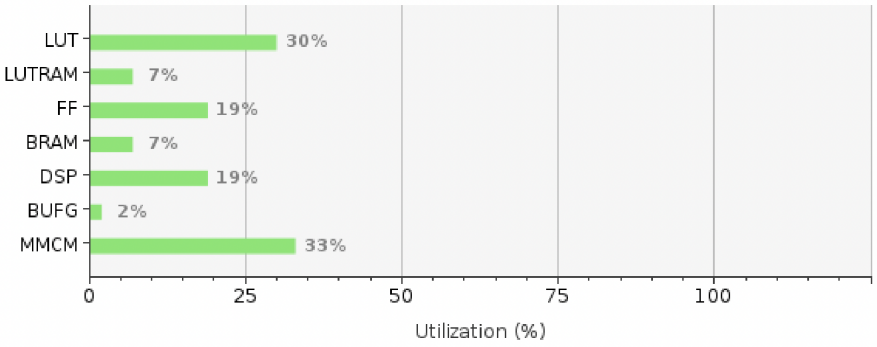


Figura 6.14: **Diagrama del sistema.**

6.2.2.1 Recursos

En esta etapa se obtienen resultados reales. En el caso del consumo de recursos el uso de las LUTs sigue siendo crítico, aunque menor al estimado, para un valor de 34.73 % para el kernel de aceleración (Figura 6.16) y un 37 % para el sistema completo (Figura 6.15). Manteniéndose el resto de elementos por debajo del 20 %.



Resource	Utilization	Available	Utilization %
LUT	20953	70560	29.70
LUTRAM	2115	28800	7.34
FF	26130	141120	18.52
BRAM	16	216	7.41
DSP	69	360	19.17
BUFG	3	196	1.53
MMCM	1	3	33.33

Figura 6.15: **Consumo de recursos del sistema completo.**

Kernel Route Utilization

Name	LUT	LUTAsMem	REG	BRAM	DSP
Platform	2.64%	0.44%	1.57%	0.00%	0.00%
User Budget	100.00%	100.00%	100.00%	100.00%	100.00%
Used Resources	27.79%	6.94%	17.22%	7.41%	19.17%
Unused Resources	72.21%	93.06%	82.78%	92.59%	80.83%
nn_krnl (1)	27.79%	6.94%	17.22%	7.41%	19.17%
nn_krnl_1	27.79%	6.94%	17.22%	7.41%	19.17%

Figura 6.16: **Consumo de recursos del kernel de aceleración.**

6.2.2.2 Rendimiento

El rendimiento del kernel de aceleración integrado en el sistema final generado con el flujo acelerado de Vitis™se observa en la Figura 6.17 y Figura 6.18 Para validar estos resultados se realiza una comparación entre la salida de la capa densa del kernel de aceleración y la salida de la capa densa del modelo diseñado en Python. Una muestra del correcto funcionamiento se observa en la Figura 6.17 , donde se analizan las 3 primeras ventanas de datos. Como resultado se obtiene que los valores iguales.

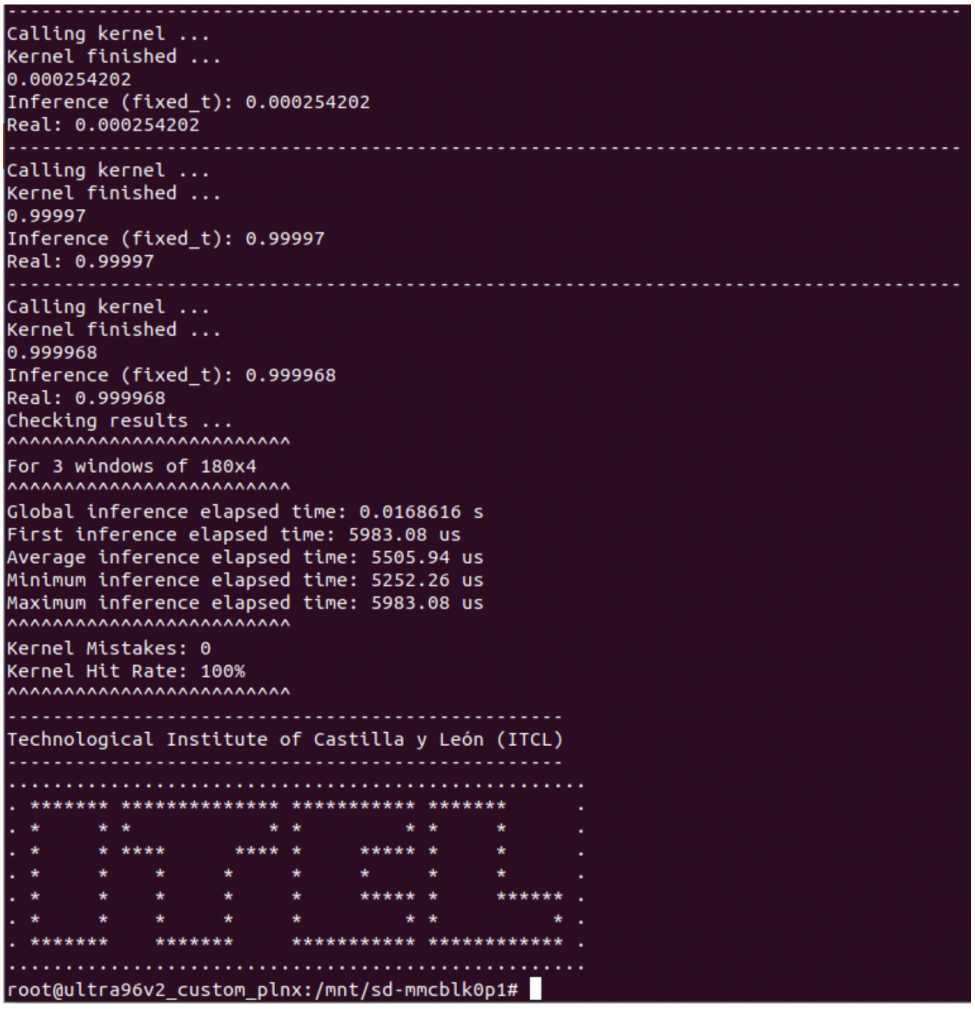


Figura 6.17: **Rendimiento del kernel de aceleración integrado en el flujo acelerado de Vitis. Inferencia sobre 3 ventanas.**

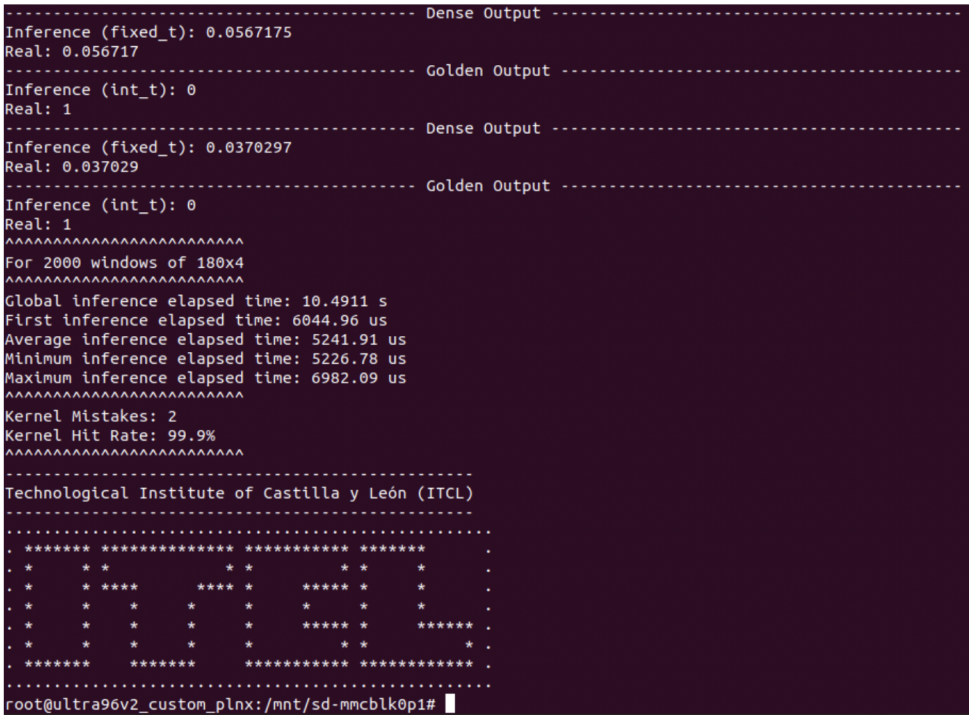


Figura 6.18: **Rendimiento del kernel de aceleración integrado en el flujo acelerado de Vitis. Inferencia sobre 2000 ventanas.**

Con estas pruebas conseguimos un resultado del orden de 5ms para un consumo de recursos bastante bajo. Para mejorar el rendimiento se puede utilizar el paralelismo brindado por la biblioteca *BLAS* (Basic Linear Algebra Subroutines) de Vitis™, pero para ello es necesario que las dimensiones de las matrices y vectores presentes en el algoritmo de inferencia sean potencia de 2. Sin embargo, esta mejora de rendimiento implica un aumento del consumo de recursos.

6.3 Pruebas realizadas sobre Instalación de VCK5000 y Alveo U50

Para realizar estas pruebas hemos recurrido a un sistema externo ya que no se ha podido realizar la instalación propia al no disponer todavía del servidor sobre el cual instalarlo porque se ha demorado sin fecha de entrega el suministro de este servidor.

En primer lugar se han descargado los ejemplos de Vitis-AI disponibles en la documentación de Xilinx Inc.

```
wget -O vitis_ai_runtime_r1.4.0_image_video.tar.gz https://www.xilinx.com/bin/public/openDownload?filename=vitis_ai_runtime_r1.4.0_image_video.tar.gz
tar -xzvf vitis_ai_runtime_r1.4.0_image_video.tar.gz -C ~/Vitis-AI/demo/VART
cd ~/Vitis-AI/
./docker_run.sh xilinx/vitis-ai-cpu:1.4.1.978
#### Aquí tendremos que pulsar varias veces <\textit{enter}> y pulsar <\textit{y}> la última vez para aceptar la licencia y esperar a la descarga de las imágenes de docker la primera vez, lo que lleva bastante tiempo

source /workspace/setup/vck5000/setup.sh

cd /workspace/demo/VART/resnet50
./build.sh

#Descargamos el modelo resnet50 para las pruebas
wget https://www.xilinx.com/bin/public/openDownload?filename=resnet50-vck5000-DPUCVDX8H-r1.4.1.tar.gz -O resnet50-vmk.tar.gz
sudo mkdir -p /usr/share/vitis_ai_library/models
tar -xzvf resnet50-vmk.tar.gz
sudo cp resnet50 /usr/share/vitis_ai_library/models -r
cd /workspace/demo/Vitis-AI-Library/samples/classification
source ./build.sh

cd ../../apps/seg_and_pose_detect/
bash -x build.sh

#Por último ejecutamos la clasificación
./test_video_classification resnet50/ ../../apps/seg_and_pose_detect/pose_960_540.avi
```

Código 6.3: **Descarga y ejecución de las pruebas sobre VCK5000 en servidor.**

Durante la ejecución, podemos utilizar los comandos del Apéndice I - Apuntes sobre XRT y VCK5000 Apuntes sobre XRT y VCK5000 para verificar el funcionamiento de la tarjeta. La figura Figura 6.19 muestra una captura de pantalla del momento de ejecución de estas pruebas. En la ventana derecha se observa la ejecución del comando */opt/xilinx/xrt/bin/xbutil examine -report thermal* como ejemplo de los que podemos ejecutar

mientras que en la otra ventana se muestra la ejecución de las pruebas. La imagen que se está analizando se muestra en la ventana flotante.

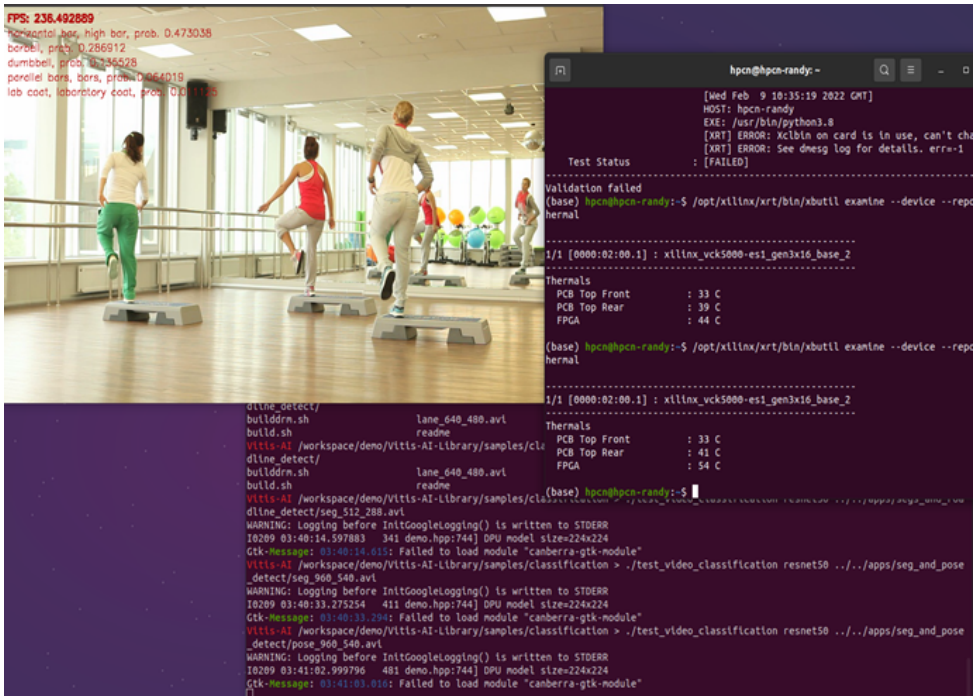


Figura 6.19: **Captura del durante la ejecución de las pruebas sobre la FPGA en el servidor externo.**

Realizadas las pruebas sobre los trabajos realizados procedemos a probar la correcta implementación de los diferentes desarrollos. En el caso de Sección 6.1 hemos comprobado que la inferencia funciona de forma correcta al recibir datos en tiempo real, con Sección 6.2 LOQUESEALOESCRIBIMOSAQUI y con Sección 6.3 hemos comprobado como funcionarían las pruebas incluidas en el sistema de Vitis-AI a pesar de que no hemos podido realizar esto en nuestras instalaciones al no haber recibido el servidor sobre el cual deben funcionar estas tarjetas.

Capítulo 7

Planificación temporal y presupuesto

7.1 Planificación temporal del proyecto

La planificación inicial del proyecto se basaba en una planificación a 7 meses orientada al estudio y creación de la arquitectura de servidor para el despliegue de las herramientas LSTM. Durante el ciclo de vida del proyecto, tras la imposibilidad de adquirir ciertos de los componentes, especialmente en el suministro de los servidores profesionales para el montaje de estas placas aceleradoras, se decidió hacer un cambio en el enfoque del proyecto para poder avanzar con los elementos disponibles además de ir haciendo pruebas con las herramientas de Vitis AI de Xilinx.

Por ello, se hizo una planificación final con un mayor alcance y una duración de 12 meses, tal y como se muestra en la Figura 7.1 , dividido entre las fases de investigación, desarrollo y pruebas y con una fase de documentación que se extiende a lo largo del proyecto. Cada una de estas fases se dividen en distintas tareas y en todas ellas se considera una reunión quincenal para la puesta en común de los avances realizados.

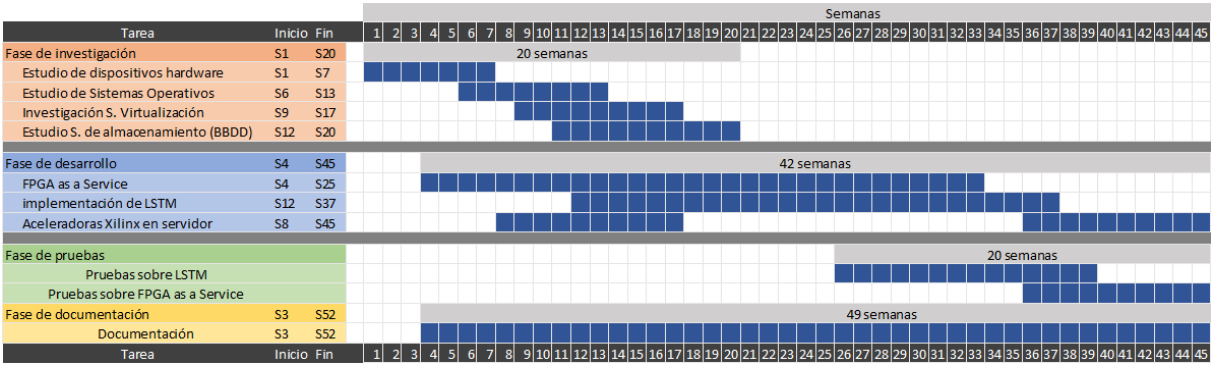


Figura 7.1: **Planificación del proyecto.**

En esta planificación, la parte referente a las aceleradoras Xilinx, se ha dividido en 2 partes debido a que, con la dificultad de suministro de los servidores, se planifica el estudio y las pruebas sobre las herramientas de Xilinx de forma inicial, procediendo con el despliegue utilizando las propias tarjetas en una fase posterior al disponer de los diferentes servidores para su montaje.

7.2 Presupuesto

A continuación se muestran las diferentes tablas que contienen los costes asociados a los recursos utilizados a lo largo del estudio.

En primer lugar se muestra la Tabla 7.1 con el registro de los recursos materiales requeridos, su coste unitario y los costes totales.

Recurso	Concepto	Cantidad	C. unitario [€/u]	C. total [€]
VCK5000 Versal	Tarjeta expansión ACAP Xilinx	1	2963,46 €	2693,46 €
Alveo U50	Tarjeta Aceleradora Xilinx	2	3623,95 €	7247,90 €
Ultra96-V2	Placa Avnet Xilinx FPGA	4	245,23 €	980,92 €
microSD	microSD cards for Ultra96	4	6,95 €	27,78 €
Servidor Asus	Servidor enrackable 1U	1	6205,23 €	6205,23 €
Beckhoff	PC Industrial C6930-0060	1	1304,63 €	1304,63 €
ethercat cp	PC ethercat for beckhoff	1	118,87 €	118,87 €
SSD	Disco duro SSD Kingston A400 1TB	1	137,00 €	137,00 €
RAM	Memoria RAM DDR4 2x16GB	1	185,00 €	185,00 €
TPLink-EU300	USTB to ETH adapter	1	19,99 €	19,99 €
TOTAL				18920,78 €

Tabla 7.1: **Costes de los recursos materiales.**

Por otra parte, se detallan en la Tabla 7.2 los costes asociados a los recursos humanos. En este caso se ha dispuesto de un jefe de proyecto con un coste de 60 € por hora de trabajo, 2 programadores con un coste de 30 € la hora y de 2 investigadores con un coste de 50 €/h. El proyecto tiene una duración de 12 meses durante los cuales se han realizado un total de 640h de trabajo repartidas de la forma que se indica en dicha tabla.

Categoría	Nº personas	Nº horas	C. unitario [€/u]	C. total [€]
Jefe de proyecto	1	40	60,00 €	2400,00 €
Investigador	2	150	50,00 €	7500,00 €
Programador	2	150	30,00 €	4500,00 €
TOTAL				14400,00 €

Tabla 7.2: **Costes de los recursos humanos.**

La labor del jefe de proyecto será la de organizar y coordinar las diferentes lineas de trabajo y de encargarse de realizar las diferentes reuniones de seguimiento, los investigadores tratarán de estudiar las diferentes soluciones diferentes en el mercado junto con la realización de un estado del arte de los conocimientos existentes y el análisis de los componentes hardware y posibilidades software con las que trabajar. Por último, la función de los programadores será la de desarrollar las distintas arquitecturas, servicios y códigos que se utilicen a lo largo del proyecto, tanto de forma auxiliar como de forma práctica.

El presupuesto final del proyecto se describe en la Tabla 7.3 y es de 33320,78 €. Todos los precios que se indican en este apartado incluyen el 21 % de IVA.

Tipo de coste	C. total [€]
Coste de recursos materiales	18920,78 €
Coste de recursos humanos	14400,00 €
TOTAL	33320,78 €

Tabla 7.3: **Presupuesto final.**

Capítulo 8

Conclusiones

A partir de las tecnologías investigadas en el TFM se puede demostrar que la familia ACAP permite integrar las redes LSTM, ya que como indican es la plataforma ideal para integrar tanto CNN como RNN, para la ejecución a través de la nube. Sin embargo, utilizando Vitis HLS es posible integrar las redes mediante compilación directa tal y como se han realizado en la Sección 5.2 sobre la Ultra96-V2.

En este trabajo se han estudiado 3 tecnologías diferentes de Xilinx:

1. Xilinx UltraScale embedded
2. Xilinx Alveo
3. Xilinx Versal ACAP

En lo referente a las UltraScale embebidas, son perfectas para el *edge computing* y están preparadas para la inferencia sobre redes CNN mientras que para la ejecución de redes RNN y LSTM se requiere de realizar una implementación en HLS y C para poder realizar una compilación directa para estos sistemas ya que Xilinx no lo ofrece por defecto.

La familia Alveo y Versal están orientadas a centros de datos, por lo que se utilizan para poder realizar su despliegue en servidores como sistemas cloud de procesamiento de datos permitiendo una integración con periféricos.

Las Alveo U50, son sistemas UltraScale orientadas a sistemas cloud frente a la orientación edge de las Ultra96-V2 soportando redes de tipo CNN. La VCK5000 Versal comparte la orientación cloud de las Alveo pero añade la compatibilidad con redes RNN que no contemplan las otras familias mencionadas.

8.1 Logros alcanzados

Durante el desarrollo del proyecto, se han seguido varias vías de investigación diferentes, obteniendo unos resultados completamente distintos en cada una de estas vías.

8.1.1 FPGA as a Service

En el apartado de la FPGA as a Service se han conseguido varios logros. En primer lugar, se ha logrado generar una API, con una serie de códigos y configuraciones, a través de la cuál comunicarse con el SDK de Vivado para realizar la cuantización y la compilación de un modelo para la FPGA correspondiente reduciendo la complejidad computacional y mejorando el rendimiento.

En segundo lugar, se ha logrado realizar una automatización en el tratamiento de los datos que se reciben por el sistema. Mediante la creación de los trigger, hemos conseguido que al recibir los datos en la base de datos se transfieran automáticamente a través de protocolos de paso de mensajes para que se puedan utilizar en la FPGA y realizar la inferencia sobre los datos, cuyos resultados se guardarán a continuación en la base de datos establecida para ello.

Por último, se ha conseguido generar un sistema totalmente autónomo mediante las configuraciones de arranque automático y el puenteado de los pines de la FPGA que permiten que al iniciar la línea de producción se arranquen automáticamente tanto los sistemas físicos del Beckhoff y la FPGA como los sistemas virtualizados sobre el Beckhoff y los servicios integrados en cada uno de estos sistemas estableciendo una conexión automática entre ellos para empezar a funcionar sin retardos ni problemas permitiendo una iniciación sin pérdida de datos y realizando la inferencia de alertas desde el comienzo.

8.1.2 Implementación de LSTM en Ultra96

En el apartado referente a la generación de redes LSTM para los dispositivos FPGA, hemos logrado optimizar una serie de redes neuronales concretas para la Ultra96-V2.

Entre estas redes desplegadas se encuentran las redes neuronales convolucionales, algunas de las cuales están disponibles directamente a través de la herramienta de Xilinx Inc., las cuales no suponen una gran dificultad de implementación. Además, se ha trabajado con redes consistentes en diferentes capas densas en la primera implementación.

Posteriormente, se logró desarrollar una red neuronal en la que se ha logrado incorporar a una red con una capa densa una capa LTSM para 4 características y una ventana de inferencia de 180 sobre la capa densa con una neurona.

8.1.3 Instalación de VCK5000 y Alveo U50

En este sentido hemos logrado realizar una instalación de los sistemas con los requisitos y dependencias para su utilización con este tipo de placas de servidor.

Se han instalado las herramientas Vitis AI de Xilinx Inc. junto con el XRT y el sistema de notebooks para disponer de un entorno de programación vía aplicación web aunque no ha sido posible la integración final de estos sistemas junto con las tarjetas debido a la incompatibilidad con nuestro equipos y la demora en la entrega del servidor final sobre el que se podría completar este apartado.

A nivel hardware se ha realizado la adaptación de la VCK5000 para su integración en el modo de disipación activa, que es el modo en el que se entrega de serie con ambos ventiladores de turbina, como en el modo de disipación pasiva, realizando la sustracción de estos ventiladores y el acortamiento de la placa para su integración en sistemas de menor espacio disponible y que dispongan de una disipación de calor externa suficiente.

8.2 Limitaciones

8.2.1 Limitaciones del sistema FaaS

La principal limitación del sistema de FPGA as a Service con la Ultra96-V2 es que actualmente solo funciona para uno de los modos de funcionamiento del sistema externo del cliente ya que es para el que ha sido generado el modelo y entrenada la red a pesar de que ya existen definidos los métodos de cambio de modelo para poder adaptarlo a los diferentes modos de funcionamiento una vez definidos los nuevos modelos.

En caso de que el cambio a un nuevo modo de funcionamiento se pueda realizar en base a un reentrenamiento del modelo, no sería demasiado complejo a pesar de que la generación del RTL es demasiado larga y se puede demorar sobre 1h por lo que en caso de detectar un fallo en la inferencia, llevaría bastante tiempo volver a entrenar el modelo. En caso de que no sirva el volver a entrenar el modelo y sea necesario generar uno nuevo, el proceso no es automático y habría que detener la ejecución, generar un nuevo modelo y volver a poner en marcha el sistema completo.

Otra de las limitaciones nos refiere a mosquitto ya que dispone de un tamaño máximo del *payload* (tamaño máximo del mensaje) de 256Mb, por lo que si hubiera que enviar una cantidad de datos excesivamente grande tendríamos un problema ya que habría que buscar otros medios de comunicación o tratar de realizar el envío a través de varios mensajes por lo que habría que hacer una amplia modificación en los códigos de procesamiento para adaptarse a esta situación.

8.2.2 Limitaciones de implementación de LSTM en FPGAs

La principal limitación de la implementación de redes LSTM en FPGA es el consumo de recursos que supone este tipo de algoritmos. A diferencia de las redes neuronales formadas solamente por capas densas, el algoritmo de inferencia de una capa LSTM es mucho más complejo, presentando un mayor número de operaciones e iteraciones de las mismas. Por lo que para la implementación de redes neuronales más complejas, es decir, con más capas LSTM, es necesario contar con FPGAs de mayores prestaciones.

Este sistema está limitado, en el momento de esta documentación, a una única capa LSTM en la red neuronal o al uso de diferentes capas densas debido a la dificultad de este tipo de implementaciones y que se encuentra todavía en fase de investigación para poder llegar a implementar un tipo de redes más complejas.

8.2.3 Limitaciones del sistema de aceleración en servidor con VCK5000 y Alveo U50

Por el momento no hemos sido capaces de detectar limitaciones en este sistema ya que al no disponer del servidor para hcerlo funcionar completamente en nuestro entorno no hemos tenido la posibilidad de investigar a fondo estas condiciones.

8.3 Problemas encontrados

8.3.1 Problemas referentes al sistema FPGA as a Service

El principal problema referente a este sistema, al igual que para el resto de apartados, ha sido el problema de stock y el retraso continuado en el suministro de los diferentes elementos necesarios, los cuales se han retrasado varios meses en algunos casos y han tenido las fases de desarrollo bastante estancadas por lo que ha sido difícil continuar con ciertos puntos y ha reducido considerablemente el alcance de las distintas líneas de investigación.

Por otro lado, un problema en el suministro eléctrico ha tenido parados algunos de los sistemas desplegados por lo que las pruebas se han tenido que retrasar hasta que ese problema ha sido solucionado y no se ha podido dar por finalizada esta implementación hasta un tiempo bastante posterior al previsto. A pesar de esto, finalmente se ha solucionado y se ha podido verificar el funcionamiento del sistema.

8.3.2 Problemas referentes a las LSTM sobre FPGA

El principal problema para la implementación de redes neuronales LSTM en FPGA es la falta de soporte por parte de bibliotecas como FINN y hls4ml para la implementación con HLS. De igual forma, las actuales versiones del módulo IP DPU de Xilinx tampoco soporta este tipo de redes neuronales.

8.3.3 Problemas referentes a la placa VCK5000

Durante la realización de las implementaciones prácticas de este proyecto nos hemos encontrado diferentes problemas, especialmente en lo relativo a la instalación de la Xilinx VCK5000 Versal.

En primer lugar, hemos tenido grandes problemas de suministro de los componentes necesarios, tanto con la FPGA en sí que tardó varios meses cómo en el suministro del Servidor Final que no se ha recibido hasta finales de agosto, motivo por el cual se hicieron las pruebas con los equipos que teníamos disponibles en la empresa con otros proyectos como servidores. Para conseguir las Alveo U50 Accelerator Card también tuvimos que

esperar bastantes meses e hicimos la investigación sobre la VCK5000 para ir avanzando con la familiarización con el despliegue de las herramientas de Xilinx.

Para la instalación de la VCK5000 Versal, en las especificaciones iniciales de Vitis AI para la tarjeta VCK5000 mostrada en la Figura 8.1 nos indicaba que requería una placa base con una conexión libre de PCIe 3.0 y utilizamos las placas bases mencionadas en la Subsección 3.1.2 que disponían de dicho slot, en el cual se encontraban conectadas las GPU RTX3090 que se ven en las imágenes del Apéndice A - Servidor 1 y del Apéndice B - Servidor 2, sin embargo al instalar la FPGA y proceder con la instalación de las herramientas de Xilinx nos encontramos con una serie de errores en los que el sistema no encontraba la tarjeta o se quedaba bloqueado al terminar la instalación del *XRT*, se muestran las capturas de estos errores en el Apéndice G - Errores servidor con VCK5000.

Component		Requirement
FPGA	Alveo	U50, U50LV, U200, U250, U280 cards
	Zynq UltraScale+ MPSoc	ZCU102 and ZCU104 Boards
	Versal	VCK190 and VCK5000 boards
	Kria	KV260
Motherboard		PCI Express 3.0-compliant x16 with one or dual slot
System Power Supply		225W
Operating System	Ubuntu	18.04, 20.04
	CentOS	7.8, 7.9, 8.1, 8.2
	RHEL	8.3, 8.4
CPU		Intel i3/i5/i7/i9/Xeon 64-bit CPU
		AMD EPYC 7F52 64-bit CPU
GPU (Optional to accelerate quantization)		NVIDIA GPU supports CUDA 11.0 or higher, like NVIDIA P100, V100, A100
CUDA Driver (Optional to accelerate quantization)		Driver compatible to CUDA version, NVIDIA-450 or higher for CUDA 11.0
Docker Version		19.03 or higher

Figura 8.1: **Diagrama de bloques de Versal VCK5000**

Como podemos observar en la Figura 8.1 , estos requisitos también aplican para las tarjetas Alveo U50. En este caso cómo llegaron bastante más tarde que el resto del hardware específico utilizado, no se llegaron a probar con esas placas ya que ya habíamos comprobado que eran incompatibles con la Versal y necesitábamos compatibilidad completa con todas ellas.

8.4 Trabajos futuros

8.4.1 Trabajos futuros con FPGA as a Service

El principal trabajo a futuro planteado para este sistema es hacer posible la ejecución de este modelo con con el resto de modos de funcionamiento de la máquina, comenzando por los más sencillos en los cuales el modelo utilizado es válido con la única necesidad

de volver a entrenarlo y pasando a una automatización en el sistema de generación de nuevos modelos para poder adaptarlo a aquellos modos en los que sea necesario un modelo diferente para realizar la inferencia.

8.4.2 Trabajos futuros en el desarrollo de LSTM

Los futuros trabajos en el desarrollo de redes neuronales LSTM con Vitis para los dipositivos FPGA pasan por la implementación de redes que contengan distintas capas LSTM a mayores de lo conseguido hasta el momento que es una red con una capa LSTM y una capa densa.

Por otro lado, se trabajará en el desarrollo de un kernel que permita generar una serie de redes neuronales optimizadas para desplegar sobre los servidores (VCK5000 Versal y Alveo U50) y que permitan generar una biblioteca de redes LSTM para su uso en los servidores con las diferentes placas aceleradoras.

8.4.3 Trabajos futuros con las placas de servidor

Este es el punto menos avanzados ya que no hemos podido disponer del servidor final a tiempo, por lo tanto nos queda mucho trabajo a futuro en este punto.

Inicialmente, se instalará el servidor con las tarjetas aceleradoras mencionadas, realizando las adaptaciones que se deban realizar a nivel de hardware e instalando el software y las dependencias probadas siguiendo el manual desarrollado para ello sobre el sistema compatible.

Además, como se ha mencionado en Subsección 8.4.2 , se trabajará de forma conjunta en las 2 lineas de investigación para lograr el desarrollo de redes diseñadas con HLS para que funcionen sobre estas aceleradoras y poder completar una librería que se pueda dejar accesible a través del servidor.

Apéndice A

Servidor 1

Se añaden las capturas del servidor 1. Las especificaciones se muestran en los anexos Anexo 1 - Características placa GA-H270-HD3 y Anexo 3 - Características Procesador Intel i7 7700K.

Figura A.1: **Captura frontal de la *workstation* utilizada**

Figura A.2: **Captura interior de la *workstation* utilizada**

Figura A.3: Captura lateral de la *workstation* utilizada

Figura A.4: Captura del interior de la *workstation* utilizada

Apéndice B

Servidor 2

Se añaden las capturas del servidor 2. Las especificaciones se muestran en los anexos Anexo 2 - Características placa Z170XP-SLI y Anexo 3 - Características Procesador Intel i7 7700K.

Figura B.1: **Captura frontal de la *workstation* utilizada**

Figura B.2: **Captura interior de la *workstation* utilizada**

Figura B.3: **Captura lateral de la *workstation* utilizada**

Figura B.4: **Captura del interior de la *workstation* utilizada**

Apéndice C

Servidor Final

Se añaden las capturas del servidor enrackable utilizado finalmente para el montaje de las placas VCK5000 Versal y Alveo U50. Las especificaciones se muestran en los anexos Anexo 4 - Datasheet Servidor Final y Anexo 5 - Características Procesador AMD Epyc 7313P.

Figura C.1: **Captura frontal de la *workstation* utilizada**

Figura C.2: **Captura interior de la *workstation* utilizada**

Figura C.3: **Captura lateral de la *workstation* utilizada**

Figura C.4: **Captura del interior de la *workstation* utilizada**

Apéndice D

VCK5000 Versal

Se añaden las capturas de la placa de desarrollo VCK5000 Versal de Xilinx.

Figura D.1: **Vista lateral de la Xilinx VCK5000 Versal**

Figura D.2: **Vista superior de la Xilinx VCK5000 Versal**

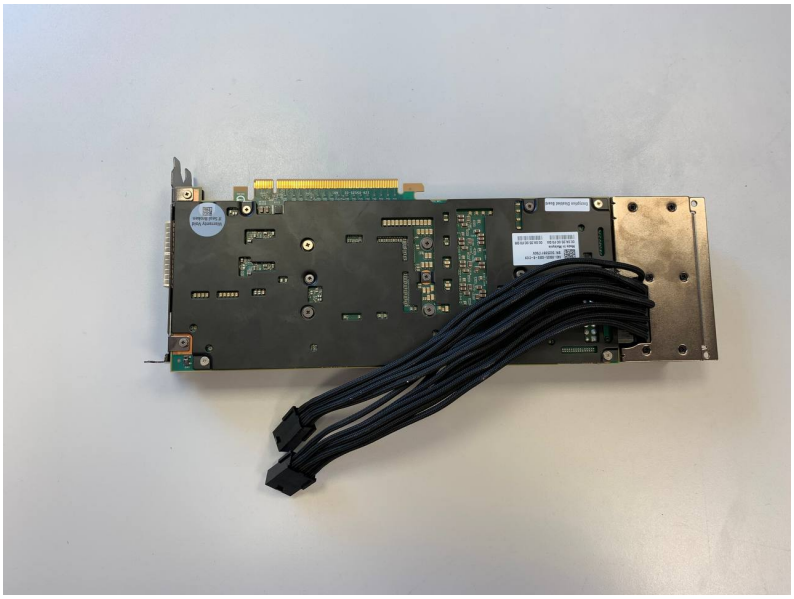


Figura D.3: **Vista inferior de la Xilinx VCK5000 Versal**

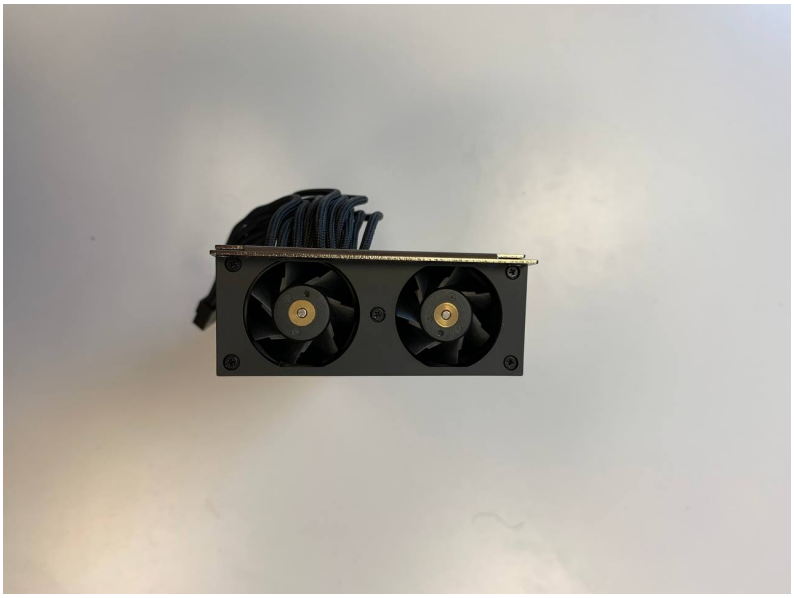


Figura D.4: **Vista de la parte interior (ventiladores turbina) de la Xilinx VCK5000 Versal**

Figura D.5: **Vista de la parte exterior (conexiones 2x QSFP28 (100GbE)) de la Xilinx VCK5000 Versal**

Apéndice E

Alveo U50

Se añaden las capturas de la placa Alveo U50 Acceleration Card de Xilinx.

Figura E.1: **Vista superior de la Alveo U50 Acceleration Card**

Figura E.2: **Vista inferior de la Alveo U50 Acceleration Card**

Figura E.3: **Vista de la parte exterior (conexión QSFP28 (100GbE)) de la Alveo U50 Acceleration Card**

Apéndice F

Avnet Xilinx Ultra96-V2

Se añaden las capturas de la placa ZUS+ Ultra96-v2 de Xilinx.

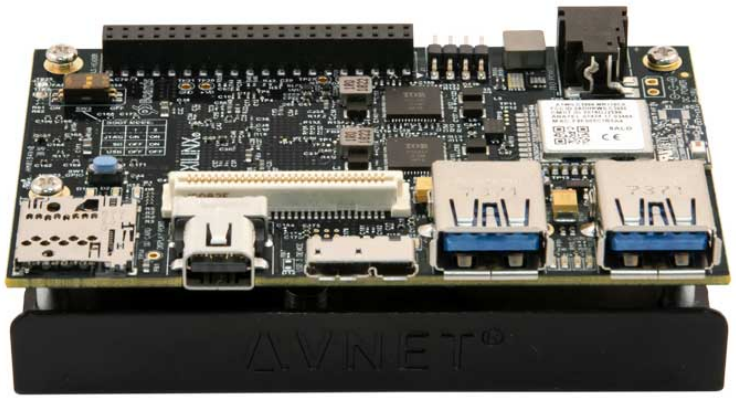


Figura F.1: **Detalle de la placa Ultra96-V2 en la web de Avnet**

Figura F.2: **Vista superior de la caja fabricada para la ZUS+ Ultra96-V2**

Figura F.3: **Vista de las conexiones sobre la caja de la ZUS+ Ultra96-V2**

Apéndice G

Errores servidor con VCK5000

Se adjuntas las imágenes relativas a los errores encontrados durante la instalación de la placa VCK5000 Versal en los servidores.

```
[adrian@versal-server development]$ sudo lspci -vd 10ee:
01:00.0 Memory controller: Xilinx Corporation Device 5044
        Subsystem: Xilinx Corporation Device 000e
        Flags: bus master, fast devsel, latency 0, IRQ 11
        Memory at e0000000 (64-bit, prefetchable) [size=128M]
        Memory at e8020000 (64-bit, prefetchable) [size=128K]
        Capabilities: [40] Power Management version 3
        Capabilities: [48] MSI: Enable- Count=1/1 Maskable- 64bit+
        Capabilities: [60] MSI-X: Enable- Count=32 Masked-
        Capabilities: [70] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [180] Alternative Routing-ID Interpretation (ARI)
        Capabilities: [1c0] #19
        Capabilities: [3a0] #25
        Capabilities: [600] Vendor Specific Information: ID=0020 Rev=0 Len=010 <?>

01:00.1 Memory controller: Xilinx Corporation Device 5045
        Subsystem: Xilinx Corporation Device 000e
        Flags: bus master, fast devsel, latency 0, IRQ 10
        Memory at e8000000 (64-bit, prefetchable) [size=128K]
        Memory at d0000000 (64-bit, prefetchable) [size=256M]
        Capabilities: [40] Power Management version 3
        Capabilities: [48] MSI: Enable- Count=1/1 Maskable- 64bit+
        Capabilities: [60] MSI-X: Enable- Count=32 Masked-
        Capabilities: [70] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [180] Alternative Routing-ID Interpretation (ARI)
        Capabilities: [600] Vendor Specific Information: ID=0020 Rev=0 Len=010 <?>
```

Figura G.1: **Reconocimiento de la placa conectada con *lspci***

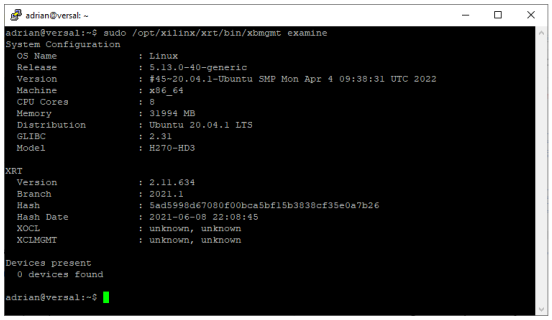


Figura G.2: Error *0 devices found* con el comando *xbmgmt examine*

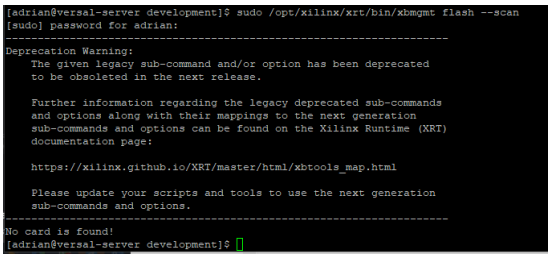


Figura G.3: Error *no card found* con el comando *xbmgmt flash –scan*

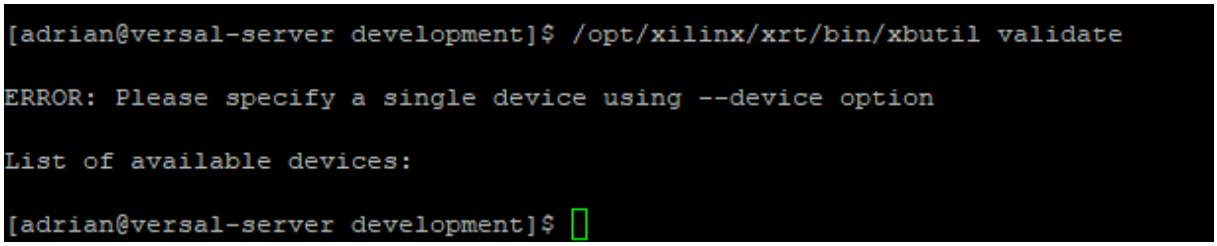


Figura G.4: Ningún dispositivo detectado al ejecutar el comando *xbutil validate*

Apéndice H

Beckhoff C6930-0060

Se añaden las capturas del Beckhoff C6930-0060 utilizado en el despliegue del producto. Las especificaciones del procesador se muestran en el anexo Anexo 6 - Características Procesador Intel i7 7700.

Figura H.1: **Captura frontal del Beckhoff C930-0060 con la etiqueta de características**

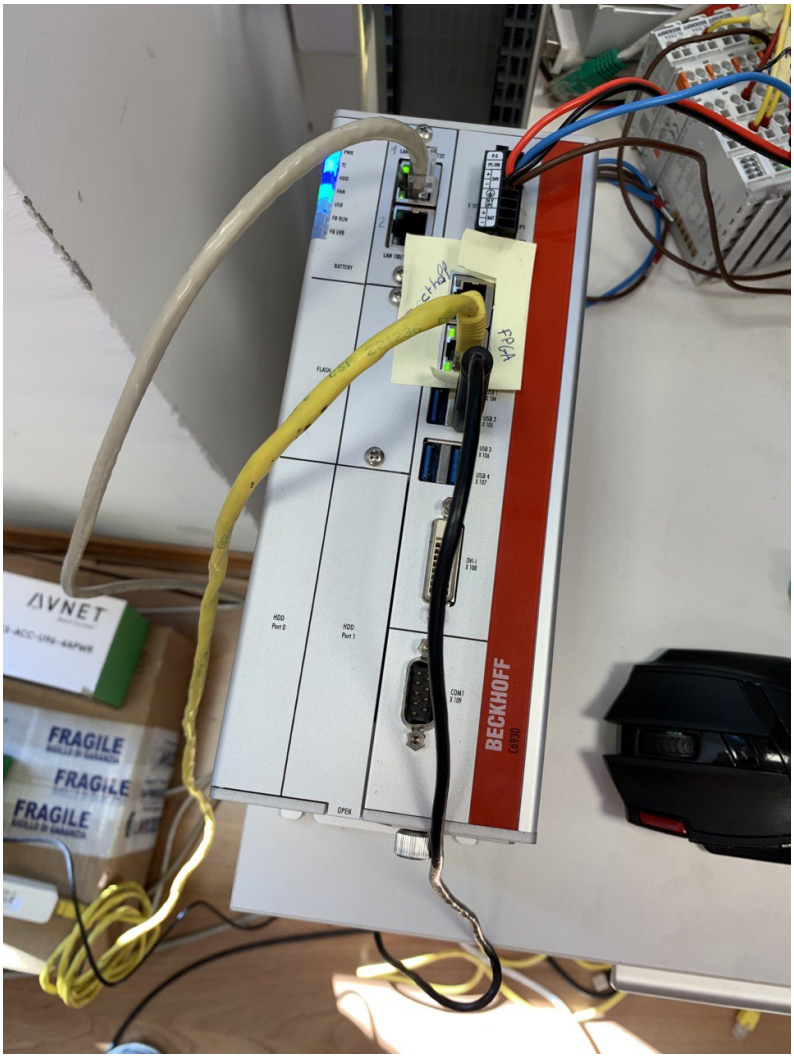


Figura H.2: **Captura superior del Beckhoff C930-0060 mostrando todas las conexiones disponibles en el equipo**

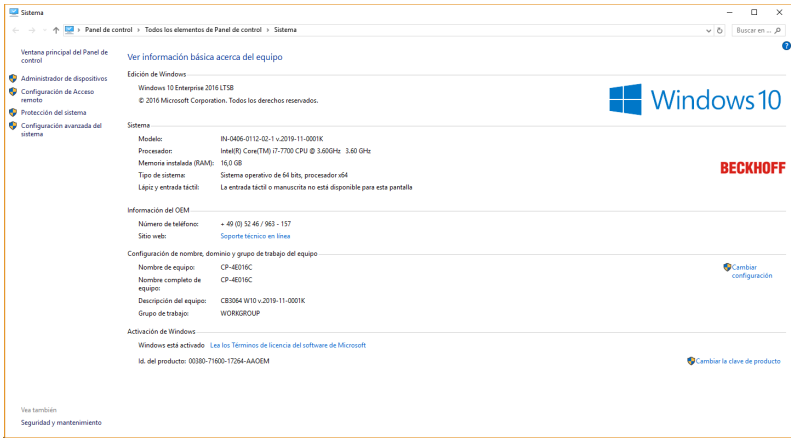


Figura H.3: **Captura de pantalla de las propiedades del sistema en el Beckhoff**

A continuación se añaden las capturas realizadas del programa CPU-Z en las cuales se muestran las distintas características del sistema en detalle.

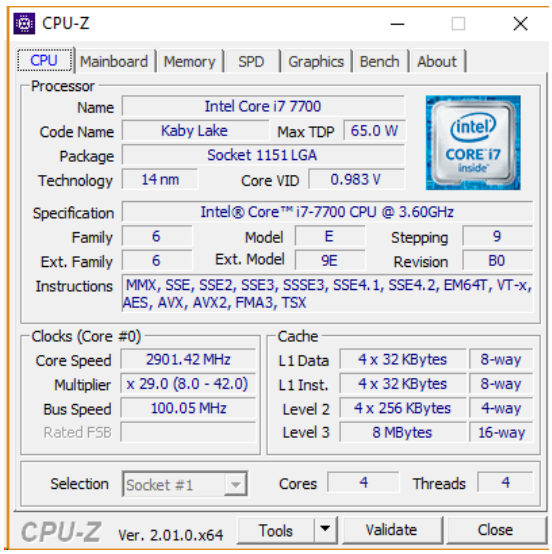


Figura H.4: Captura de la página CPU del programa CPU-Z

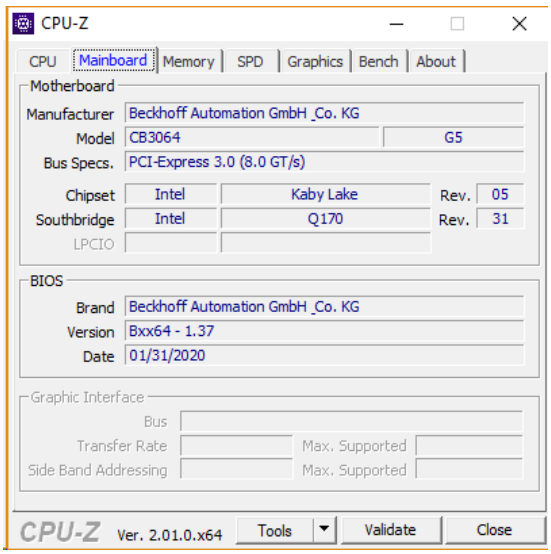


Figura H.5: Captura de la página de placa base del programa CPU-Z

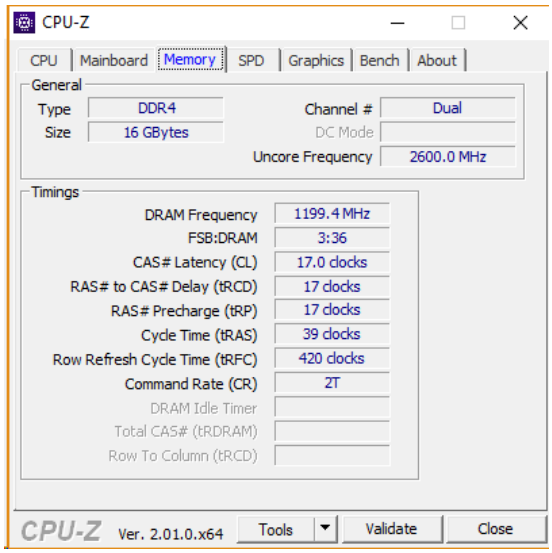


Figura H.6: Captura de la página de memoria del programa CPU-Z

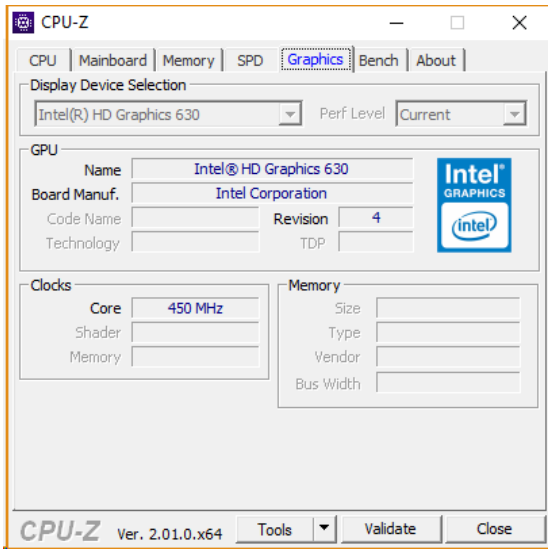


Figura H.7: Captura de la página de gráficos del programa CPU-Z

Apéndice I

Apuntes sobre XRT y VCK5000

Los comandos han cambiado de versiones anteriores, los nuevos comandos se encuentran en el githb de xilinx en: https://xilinx.github.io/XRT/master/html/xbtools_map.htm

Para actualizar el firmware de la placa hay que ejecutar los siguientes comandos. El primero corresponde con la versión anterior y el segundo es el adaptado a la nueva versión de comandos. A continuación, se muestra la Figura I.1 con el uso de este segundo comando.

Tras actualizar es necesario hacer un ***reboot*** a la máquina.

```
1 sudo /opt/xilinx/xrt/bin/xbmgmt flash --update
2 sudo /opt/xilinx/xrt/bin/xbmgmt program --device 0000:31:00.0 --base
```

Código I.1: **Actualización de firmware de VCK5000.**

Figura I.1: **Uso del comando cbmgmt para actualizar el firmware. Ya estaba actualizado.**

Para chequear la versión del sistema se puede utilizar el siguiente comando cuya salida se muestra como Figura I.2

```
1 sudo /opt/xilinx/xrt/bin/xbmgmt examine
```

Código I.2: **Examinar versión firmware VCK5000.**

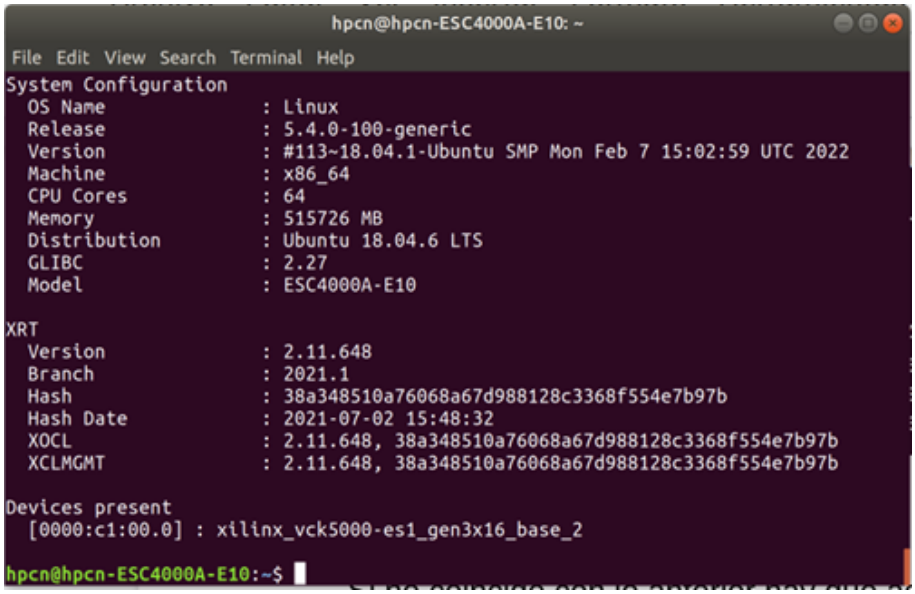


Figura I.2: **xmbgmt examine para ver la versión del firmware.**

Los siguientes comandos se pueden ejecutar para validar la instalación en la Figura I.3 se muestra el resultado correcto y en la Figura I.4 se muestra la salida si no se ha reiniciado la máquina tras la actualización.

```
1 /opt/xilinx/xrt/bin/xbutil validate
2 /opt/xilinx/xrt/bin/xbutil validate --device 0000:C1:00.1
```

Código I.3: **Actualización de firmware de VCK5000.**

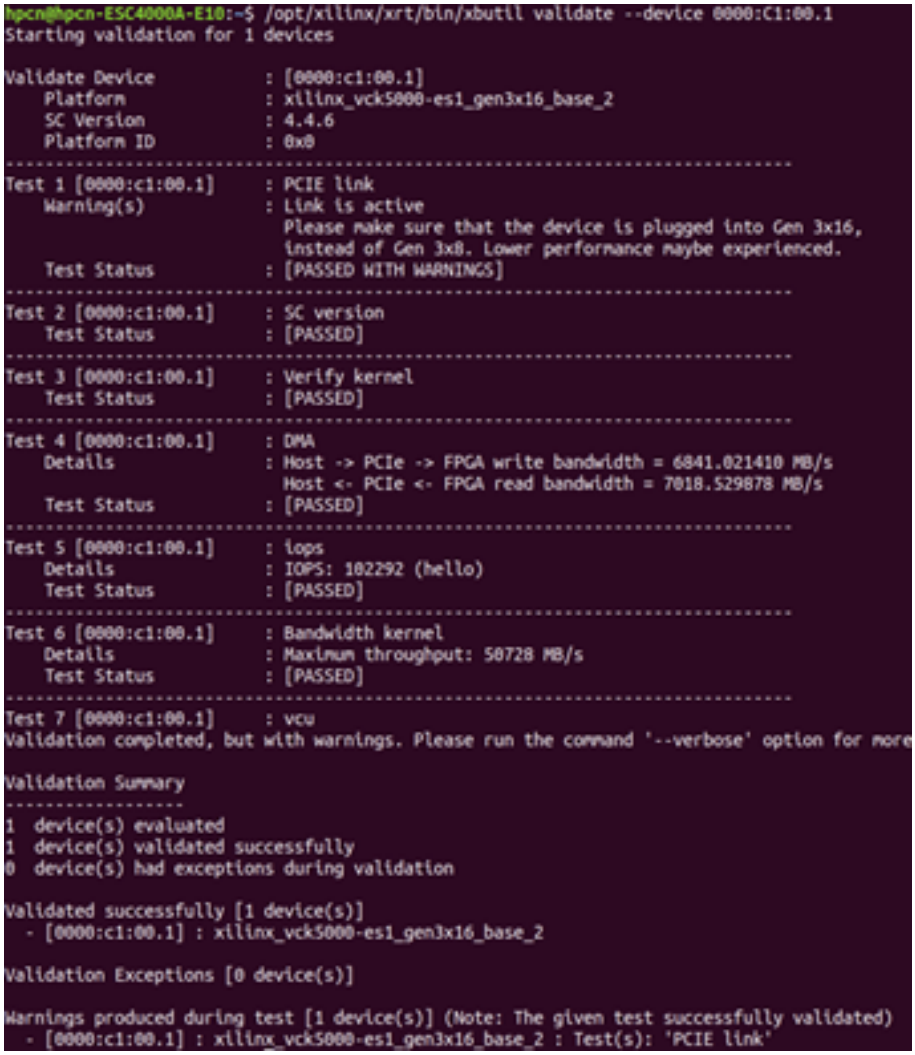


Figura I.3: **Validación correcta del firmware.**

SI NO se hace el reboot da algo por el estilo:

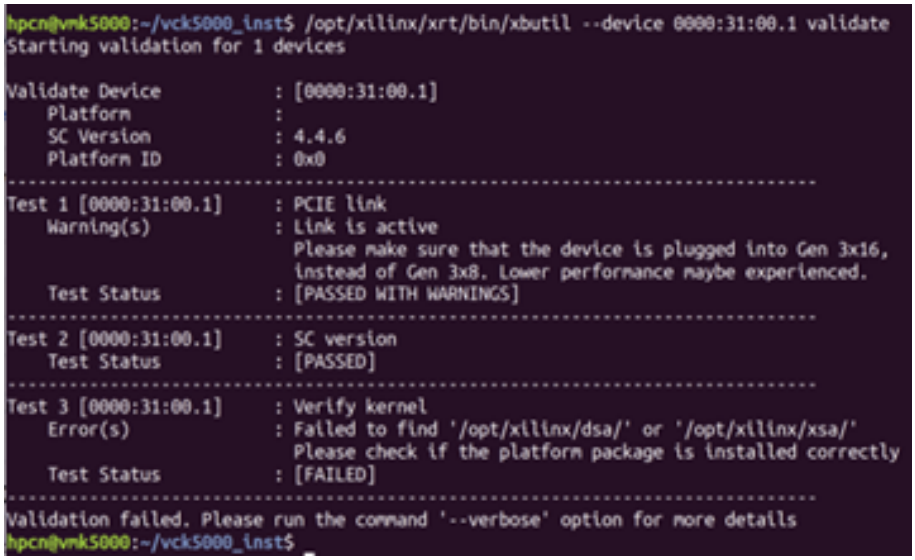


Figura I.4: **Validación del firmware sin reiniciar el sistema.**

Este comando se puede ejecutar también en modo *verbose* y nos daría una salida similar a la siguiente de la FGG

```
1 /opt/xilinx/xrt/bin/xbutil validate --device 0000:C1:00.1 --verbose
```

Código I.4: **Validación en modo *verbose***

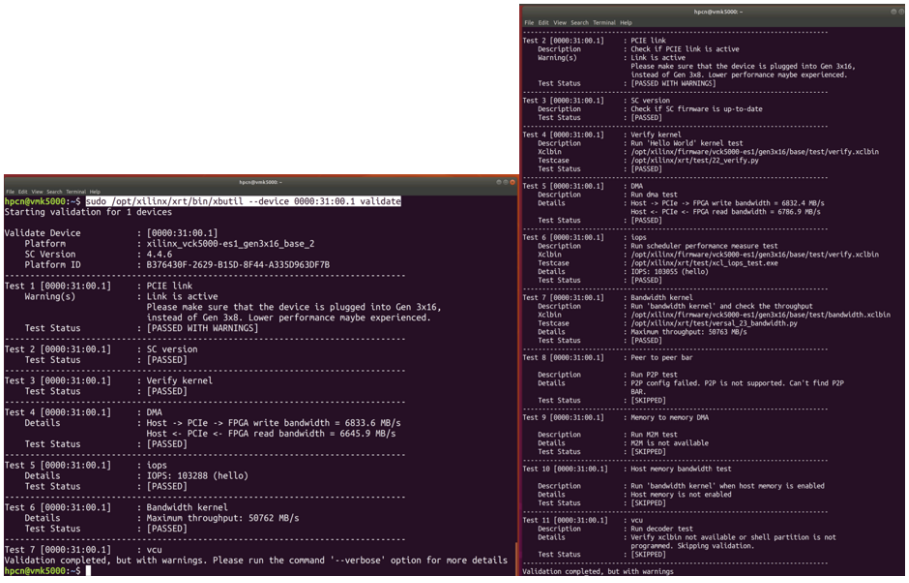


Figura I.5: **Validación correcta del firmware.**

Otros comandos interesantes para obtener datos sobre la instalación:

```
1 /opt/xilinx/xrt/bin/xbutil examine --device
2 /opt/xilinx/xrt/bin/xbutil examine --device 0000:81:00.1
3 /opt/xilinx/xrt/bin/xbutil examine -r all                    # -r == --report
```

```
/opt/xilinx/xrt/bin/xbutil examine --help
/opt/xilinx/xrt/bin/xbutil examine -r thermal          # -r == --report
/opt/xilinx/xrt/bin/xbutil examine -report electrical # -r == --report
/opt/xilinx/xrt/bin/xbutil --device 0000:02:00.1 validate --run DMA
```

Código I.5: **Otros comandos de interés.**

Anexo 1

Características placa GA-H270-HD3

GA-H270-HD3

CPU	1. Support for 7th and 6th generation Intel® Core™ i7 processors/Intel® Core™ i5 processors/Intel® Core™ i3 processors/Intel® Pentium® processors/Intel® Celeron® processors in the LGA1151 package 2. L3 cache varies with CPU (Please refer "CPU Support List" for more information.)
Chipset	1. Intel® H270 Express Chipset
Memory	1. 4 x DDR4 DIMM sockets supporting up to 64 GB of system memory * Due to a Windows 32-bit operating system limitation, when more than 4 GB of physical memory is installed, the actual memory size displayed will be less than the size of the physical memory installed. 2. Dual channel memory architecture 3. Support for DDR4 2400* / 2133 MHz memory modules 4. Support for ECC Un-buffered DIMM 1Rx8/2Rx8 memory modules (operate in non-ECC mode) 5. Support for non-ECC Un-buffered DIMM 1Rx8/2Rx8/1Rx16 memory modules 6. Support for Extreme Memory Profile (XMP) memory modules * To support 2400 MHz or XMP memory, you must install a 7th generation processor. (Please refer "Memory Support List" for more information.)
Onboard Graphics	Integrated Graphics Processor-Intel® HD Graphics support:

	1. 1 x D-Sub port, supporting a maximum resolution of 1920x1200@60 Hz 2. 1 x DVI-D port, supporting a maximum resolution of 1920x1200@60 Hz * The DVI-D port does not support D-Sub connection by adapter. 3. 1 x HDMI port, supporting a maximum resolution of 4096x2160@24 Hz * Support for HDMI 1.4 version. Support for up to 3 displays at the same time Maximum shared memory of 1 GB
Audio	1. Realtek® ALC887 codec 2. High Definition Audio 3. 2/4/5.1/7.1-channel 4. Support for S/PDIF Out
LAN	1. Intel® GbE LAN chip (10/100/1000 Mbit)
Expansion Slots	1. 1 x PCI Express x16 slot, running at x16 (PCIEX16) * For optimum performance, if only one PCI Express graphics card is to be installed, be sure to install it in the PCIEX16 slot. 2. 2 x PCI Express x16 slot, running at x4 (PCIEX4_1, PCIEX4_2) 3. 2 x PCI Express x1 slots (All of the PCI Express slots conform to PCI Express 3.0 standard.) 4. 1 x PCI slot
Multi-Graphics Technology	1. Support for AMD Quad-GPU CrossFireX™ and 2-Way AMD CrossFire™ technologies (PCIEX16 and PCIEX4_1)

Storage Interface	Chipset: 1. 1 x M.2 connector (Socket 3, M key, type 2242/2260/2280/22110 SATA and PCIe x4/x2 SSD support) 2. 1 x SATA Express connector 3. 6 x SATA 6Gb/s connectors 4. Support for RAID 0, RAID 1, RAID 5, and RAID 10 * Refer to "1-7 Internal Connectors," for the installation notices for the M.2 and SATA connectors.
USB	Chipset: 1. 8 x USB 3.1 Gen 1 ports (4 ports on the back panel, 4 ports available through the internal USB headers) 2. 6 x USB 2.0/1.1 ports (2 ports on the back panel, 4 ports available through the internal USB headers)
Internal I/O Connectors	1. 1 x 24-pin ATX main power connector 2. 1 x 8-pin ATX 12V power connector 3. 1 x M.2 Socket 3 connector 4. 1 x SATA Express connector 5. 6 x SATA 6Gb/s connectors 6. 1 x CPU fan header 7. 3 x system fan headers 8. 1 x front panel header 9. 1 x front panel audio header 10. 1 x S/PDIF Out header 11. 2 x USB 3.1 Gen 1 headers 12. 2 x USB 2.0/1.1 headers 13. 1 x Trusted Platform Module (TPM) header 14. 1 x serial port header 15. 1 x parallel port header 16. 1 x Clear CMOS jumper
Back Panel	

Connectors	1. 1 x PS/2 keyboard/mouse port 2. 1 x D-Sub port 3. 1 x DVI-D port 4. 1 x HDMI port 5. 4 x USB 3.1 Gen 1 ports 6. 2 x USB 2.0/1.1 ports 7. 1 x RJ-45 port 8. 6 x audio jacks (Center/Subwoofer Speaker Out, Rear Speaker Out, Side Speaker Out, Line In, Line Out, Mic In)
I/O Controller	1. iTE® I/O Controller Chip
H/W Monitoring	1. Voltage detection 2. Temperature detection 3. Fan speed detection 4. Overheating warning 5. Fan fail warning 6. Fan speed control * Whether the fan speed control function is supported will depend on the cooler you install.
BIOS	1. 2 x 64 Mbit flash 2. Use of licensed AMI UEFI BIOS 3. Support for DualBIOS™ 4. PnP 1.0a, DMI 2.7, WfM 2.0, SM BIOS 2.7, ACPI 5.0
Unique Features	1. Support for APP Center * Available applications in APP Center may vary by motherboard model. Supported functions of each application may also vary depending on motherboard specifications. 3D OSD @BIOS Ambient LED

	AutoGreen BIOS Setup Color Temperature Cloud Station EasyTune Easy RAID Fast Boot Game Boost ON/OFF Charge Platform Power Management Smart Backup Smart Keyboard Smart TimeLock System Information Viewer USB Blocker V-Tuner 2. Support for 3TB+ Unlock 3. Support for Q-Flash 4. Support for Xpress Install
Bundled Software	1. Norton® Internet Security (OEM version) 2. Intel® Optane™ Memory Ready 3. cFosSpeed
Operating System	1. Windows® 10 64-bit (for 7th Generation Intel® Processors) 2. Windows® 10 64-bit / Windows® 8.1 64-bit / Windows® 7 32-bit / 64-bits (for 6th Generation Intel® Processors) * Please download the "Windows USB Installation Tool" from GIGABYTE's website and install it before installing Windows 7.
Form Factor	1. ATX Form Factor; 30.5cm x 22.5cm
Remark	1. Due to different Linux support condition provided

by chipset vendors, please download Linux driver from chipset vendors' website or 3rd party website.

2. Most hardware/software vendors may no longer offer drivers to support Win9X/ME/2000/XP. If drivers are available from the vendors, we will update them on the GIGABYTE website.

Anexo 2

Características placa Z170XP-SLI

GA-Z170XP-SLI

Procesador	1. Support for 7th/6th Generation Intel® Core™ i7 processors/Intel® Core™ i5 processors/ Intel® Core™ i3 processors/Intel® Pentium® processors/ Intel® Celeron® processors in the LGA1151 package * For 7th Generation Intel® Core™ processors support need to update the latest BIOS. 2. L3 cache varies with CPU (Please refer "CPU Support List" for more information.)
Chipset	1. Intel® Z170 Express Chipset
Memoria	1. 4 x DDR4 DIMM sockets supporting up to 64 GB of system memory * Due to a Windows 32-bit operating system limitation, when more than 4 GB of physical memory is installed, the actual memory size displayed will be less than the size of the physical memory installed. 2. Dual channel memory architecture 3. Support for DDR4 3466(O.C.) /3400(O.C.) /3333(O.C.) /3300(O.C.) /3200(O.C.) /3000(O.C.) /2800(O.C.) /2666(O.C.) /2400(O.C.) /2133 MHz memory modules 4. Support for ECC UDIMM 1Rx8/2Rx8 memory modules (operate in non-ECC mode) 5. Support for non-ECC UDIMM 1Rx8/2Rx8/1Rx16 memory modules 6. Support for Extreme Memory Profile (XMP) memory modules (Please refer "Memory Support List" for more information.)

Gráficos Integrados	Integrated Graphics Processor- Intel® HD Graphics support: 1. 1 x D-Sub port, supporting a maximum resolution of 1920x1200@60 Hz 2. 1 x DVI-D port, supporting a maximum resolution of 1920x1200@60 Hz * The DVI-D port does not support D-Sub connection by adapter. 3. 1 x HDMI port, supporting a maximum resolution of 4096x2160@24 Hz * Support for HDMI 1.4 version. 4. Support for up to 3 displays at the same time 5. Maximum shared memory of 1024 MB
Audio	1. Realtek® ALC1150 codec 2. High Definition Audio 3. 2/4/5.1/7.1-channel 4. Support for S/PDIF Out
LAN	1. Intel® GbE LAN chip (10/100/1000 Mbit)
Puertos de Expansión	1. 1 x PCI Express x16 slot, running at x16 (PCIEX16) * For optimum performance, if only one PCI Express graphics card is to be installed, be sure to install it in the PCIEX16 slot. 2. 1 x PCI Express x16 slot, running at x8 (PCIEX8) * The PCIEX8 slot shares bandwidth with the PCIEX16 slot. When the PCIEX8 slot is populated, the PCIEX16 slot will operate at up to x8 mode. 3. 1 x PCI Express x16 slot, running at x4 (PCIEX4) * The PCIEX4 slot shares bandwidth with the PCIEX1_2 slot. When the PCIEX1_2

	slot is populated, the PCIEX4 slot will operate at up to x1 mode. 4. 2 x PCI Express x1 slots (All of the PCI Express slots conform to PCI Express 3.0 standard.) 5. 2 x PCI slots
Tecnología Multi-Gráficos	1. Support for NVIDIA® Quad-GPU SLI™ and 2-Way NVIDIA® SLI™ technologies 2. Support for AMD Quad-GPU CrossFireX™ and 3-Way/2-Way AMD CrossFire™ technologies
Interfaz de Almacenamiento	Chipset: 1. 1 x M.2 Socket 3 connector (Socket 3, M key, type 2242/2260/2280 SATA & PCIe x4/x2/x1 SSD support) 2. 3 x SATA Express connectors 3. 6 x SATA 6Gb/s connectors 4. Support for RAID 0, RAID 1, RAID 5, and RAID 10 * Refer to "1-8 Internal Connectors," for the supported configurations with the M.2, SATA Express, and SATA connectors.
USB	Chipset: 1. 7 x USB 3.0/2.0 ports (3 ports on the back panel, 4 ports available through the internal USB headers) 2. 6 x USB 2.0/1.1 ports (2 ports on the back panel, 4 ports available through the internal USB headers) Chipset+ASMedia® ASM1142 chip: 1. 1 x USB Type-C™ port on the back panel, with USB 3.1 support

	2. 1 x USB 3.1 Type-A port (red) on the back panel
Conectores Internos de E/S	1. 1 x 24-pin ATX main power connector 2. 1 x 8-pin ATX 12V power connector 3. 1 x M.2 Socket 3 connector 4. 3 x SATA Express connectors 5. 6 x SATA 6Gb/s connectors 6. 1 x CPU fan header 7. 1 x water cooling fan header (CPU_OPT) 8. 3 x system fan headers 9. 1 x front panel header 10. 1 x front panel audio header 11. 1 x S/PDIF Out header 12. 2 x USB 3.0/2.0 headers 13. 2 x USB 2.0/1.1 headers 14. 1 x Trusted Platform Module (TPM) header 15. 1 x Thunderbolt™ add-in card connector 16. 1 x serial port header 17. 1 x parallel port header 18. 1 x Clear CMOS jumper
Conectores del Panel Trasero	1. 1 x PS/2 keyboard/mouse port 2. 1 x D-Sub port 3. 1 x DVI-D port 4. 1 x HDMI port 5. 1 x USB Type-C™ port, with USB 3.1 support 6. 1 x USB 3.1 Type-A port (red) 7. 3 x USB 3.0/2.0 ports 8. 2 x USB 2.0/1.1 ports 9. 1 x RJ-45 port 10. 6 x audio jacks (Center/Subwoofer Speaker Out, Rear Speaker Out, Side Speaker Out, Line In, Line Out, Mic In)
Controlador de E/S	1. iTE® I/O Controller Chip

Monitorización H/W	1. System voltage detection 2. CPU/System/Chipset temperature detection 3. CPU/CPU OPT/System fan speed detection 4. CPU/System/Chipset overheating warning 5. CPU/CPU OPT/System fan fail warning 6. CPU/CPU OPT/System fan speed control * Whether the fan speed control function is supported will depend on the cooler you install.
BIOS	1. 2 x 64 Mbit flash 2. Use of licensed AMI UEFI BIOS 3. Support for DualBIOS™ 4. PnP 1.0a, DMI 2.7, WfM 2.0, SM BIOS 2.7, ACPI 5.0
Características Únicas	1. Support for APP Center * Available applications in APP Center may vary by motherboard model. Supported functions of each application may also vary depending on motherboard specifications. 3D OSD @BIOS Ambient LED AutoGreen Cloud Station EasyTune Easy RAID Fast Boot Smart TimeLock Smart Keyboard Smart Backup System Information Viewer USB Blocker 2. Support for Q-Flash 3. Support for Smart Switch 4. Support for Xpress Install

Software Incluido	1. Norton® Internet Security (OEM version) 2. Intel® Smart Response Technology 3. cFosSpeed
Sistema Operativo	1. Windows® 10 64-bit (for 7th Generation Intel® Processors) 2. Windows® 10 64-bit / Windows® 8.1 64-bit / Windows® 7 32-bit / 64-bits (for 6th Generation Intel® Processors) * Please download the "Windows USB Installation Tool" from GIGABYTE's website and install it before installing Windows 7.
Diseño	1. ATX Form Factor; 30.5cm x 22.5cm
Box Contents	1. GA-Z170XP-SLI motherboard 2. Four SATA cables 3. Motherboard driver disk 4. I/O Shield 5. User's Manual 6. One 2-Way SLI bridge connector 7. Quick Installation Guide 8. One G Connector
Otras Descripciones	1. Due to different Linux support condition provided by chipset vendors, please download Linux driver from chipset vendors' website or 3rd party website. 2. Most hardware/software vendors may no longer offer drivers to support Win9X/ME/2000/XP. If drivers are available from the vendors, we will update them on the GIGABYTE website.

* Todos los materiales aquí provistos son solo para referencia. GIGABYTE se reserva el derecho de modificar o revisar el contenido a cualquier hora sin aviso alguno.
* Todas las marcas y logos son propiedad de sus propios dueños.
* Debido a la arquitectura estándar de la PC, cierta cantidad de memoria es reservada para uso del sistema y actualmente la memoria disponible es menor a la mencionada.

Anexo 3

Características Procesador Intel i7 7700K

Procesador Intel® Core™ i7-7700K

caché de 8 M, hasta 4,50 GHz

Especificaciones

Especificaciones de exportación ↓

Esencial

Colección de productos	7th Generation Intel® Core™ i7 Processors
Nombre de código	Products formerly Kaby Lake
Segmento vertical	Desktop
Número de procesador	i7-7700K
Estado	Discontinued
Fecha de lanzamiento	Q1'17
Litografía	14 nm
Elementos incluidos	Please note: The boxed product does not include a fan or heat sink

Especificaciones de la CPU

Cantidad de núcleos	4

Cantidad de subprocesos	8
Frecuencia turbo máxima	4.50 GHz
Frecuencia de la Tecnología Intel® Turbo Boost 2.0‡	4.50 GHz
Frecuencia básica del procesador	4.20 GHz
Caché	8 MB Intel® Smart Cache
Velocidad del bus	8 GT/s
Cantidad de enlaces QPI	0
TDP	91 W

Información adicional

Opciones integradas disponibles	No
Hoja de datos	Ver ahora

Especificaciones de memoria

Tamaño de memoria máximo (depende del tipo de memoria)	64 GB
Tipos de memoria	DDR4-2133/2400, DDR3L-1333/1600 @ 1.35V
Cantidad máxima de canales de memoria	2
Compatible con memoria ECC ‡	No

Gráficos de procesador

Gráficos del procesador ‡	Gráficos HD Intel® 630
Frecuencia de base de gráficos	350 MHz
Frecuencia dinámica máxima de gráficos	1.15 GHz
Memoria máxima de video de gráficos	64 GB
Compatibilidad con 4K	Yes, at 60Hz
Resolución máxima (HDMI)‡	4096x2304@24Hz
Resolución máxima (DP)‡	4096x2304@60Hz
Resolución máxima (eDP - panel plano integrado)‡	4096x2304@60Hz
Compatibilidad con DirectX*	12
Compatibilidad con OpenGL*	4.5
Intel® Quick Sync Video	Yes
Tecnología Intel® InTru™ 3D	Yes
Tecnología Intel® Clear Video HD	Yes
tecnología Intel® de video nítido	Yes
Cantidad de pantallas admitidas ‡	3
ID de dispositivo	0x5912

Opciones de expansión

Escalabilidad	1S Only
Revisión de PCI Express	3.0
Configuraciones de PCI Express ‡	Up to 1x16, 2x8, 1x8+2x4
Cantidad máxima de líneas PCI Express	16

Especificaciones del paquete

Zócalos compatibles	FCLGA1151
Máxima configuración de CPU	1
Especificación de solución térmica	PCG 2015D (130W)
$T_{JUNCTION}$	100°C
Tamaño de paquete	37.5mm x 37.5mm

Tecnologías avanzadas

Compatible con la memoria Intel® Optane™ ‡	Yes
Tecnología Intel® Turbo Boost ‡	2.0
Tecnología Intel® Hyper-Threading ‡	Yes
Intel® Transactional Synchronization Extensions – New Instructions	Sí
Intel® 64 ‡	Yes
Conjunto de instrucciones	64-bit
Extensiones de conjunto de instrucciones	Intel® SSE4.1, Intel® SSE4.2, Intel® AVX2
Estados de inactividad	Yes
Tecnología Intel SpeedStep® mejorada	Yes
Tecnologías de monitoreo térmico	Yes
Tecnología de protección de la identidad Intel® ‡	Yes

Seguridad y fiabilidad

Idoneidad para la plataforma Intel® vPro™ ‡	No
Nuevas instrucciones de AES Intel®	Yes
Secure Key	Yes
Intel® Software Guard Extensions (Intel® SGX)	Yes with Intel® ME
Extensiones de protección de la memoria Intel®	Yes
Intel® OS Guard	Yes
Bit de desactivación de ejecución ‡	Yes
Intel® Boot Guard	Yes
Programa Intel® de imagen estable para plataformas (SIPP)	No
Tecnología de virtualización Intel® (VT-x) ‡	Sí
Tecnología de virtualización Intel® para E/S dirigida (VT-d) ‡	Yes
Intel® VT-x con tablas de páginas extendidas (EPT) ‡	Yes

Anexo 4

Datasheet Servidor Final

ESC4000A-E11

1-Socket 2U Accelerator Server with 4 GPUs supported

AMD

Form Factor / 2U

CPU Number / 1 280W TDP

Memory Number / 8

PCI EXPRESS 4.0

OCP 3.0

ASUS ESC4000A-E11
1-socket server satisfies most of your workload needs, helping you reduce cooling expenses and licenses.

Feature

- Total 11 x PCI-E 4.0 Expansion Slots in 2U
- 4*PCI-E 4.0 x16 link supported for dual-slot GPUs/ Full length cards or 8*PCI-E 4.0 x16 link with PLX SKU Board for single-slot GPUs
- 8 x 3.5”/2.5” Hot-swap HDD bays (up to 4 x NVMe Supported)
- OCP 3.0 supported

Target market

- Streaming Media
- Cloud Computing
- Virtualized & VDI Application
- Enterprise & HPC Application

3rd Gen AMD EPYC™ processors with AMD 3D V-Cache™ technology

ASUS ESC4000A-E11 is built on the 3rd Gen AMD EPYC™ processors with AMD 3D V-Cache™ technology with double core density compared to previous generation in a single socket to increase server utilization.

Flexible Design and Performance

- Supports up to 4 x PCI-E Gen 4.0 dual-slot GPUs/ Full length cards or 8 x PCI-E Gen 4 single-slot GPUs and 3 x PCI-E Gen4 slots for increased acceleration.
- OCP3.0 Mezzanine slot option for added networking flexibility.
- Capacity for up to 8 x 3.5“ / 2.5’’ hot-swap storage. 4 x drive bays can be configured to support NVMe drives.

Comprehensive IT infrastructure management solution

ASMB10-iKVM and ASUS Control Center (ACC*)
ASUS ESC4000A-E11 features an embedded iKVM module and is bundled with ASUS Control Center to provide comprehensive out-of-band and in-band management.

1. 4 x PCI-E Gen 4.0 x16 slot for GPU/ Full length cards cards
2. 4 x PCI-E Gen 4.0 x16 slot for GPU / Full length cards cards
3. 2 x PCI-E Gen 4.0 x16 for butterfly riser cards
4. 1 x OCP 3.0 Gen 4.0 x16 Mezzanine slot
5. 1 x PCI-E Gen 4.0 x8 slot

ESC4000A-E11

SPECIFICATION

Processor		3rd Gen AMD EPYC™ processors with AMD 3D V-Cache™ technology (up to TDP 280W) AMD EPYC™ 7003/7002 Series Processor Family (up to TDP 280W)
Memory	**Total Slots**	8 (8-channel per CPU, 8 DIMM per CPU)
	Capacity	Up to 2TB
	Memory Type	DDR4 3200 RDIMM *Refer to ASUS server AVL for the latest update
	Memory Size	128GB, 64GB, 32GB, 16GB *Please refer to www.asus.com for latest memory AVL update
Expansion Slots	**Total PCI/PCI-X/PCI-E/PIKE Slots**	11
	Slot Type	Full-length/Full-height: 4* PCI-E 4.0 x16 link for dual-slot GPU/ Full length cards or 8*PCI-E 4.0 at x16 link with PLX SKU Board for single-slot GPU Cards Half-length/Low-profile: 2 * PCI-E 4.0 x16 for butterfly riser card 1 * PCI-E 4.0 x8 for internal HBA/RAID card or OCP3.0 slot
Storage Bays		8 x 2.5" or 3.5" Hot-swap Storage Device Bays (up to 4 x NVMe Supported)
Networking	**LAN**	2 x 1Gb/s LAN ports (Intel® I350-AM2) 1 x dedicated management port
Graphic	**VGA**	AST2600 64MB
Security		TPM-SPI module (optional) PFR module (optional)
Front I/O Ports		4 x USB3.2 Gen1 ports
Rear I/O Ports		2 x USB3.2 Gen1 ports 2 x Gigabit LAN ports (RJ45) 1 x Management port (RJ45) 1 x VGA port
Switch/LED		Front : 1 x Power Switch/LED 1 x Location Switch/LED 1 x HDD LED 1 x Message LED 1 x Q-Code/Port 80 LED 2 x LAN LED Rear : 1 x Power switch/LED 1 x Location LED 1 x Message LED 1 x HDD Access LED
OS Support		Windows® Server 2019, RedHat® , SuSE®, Ubuntu, Vmware, *Please find the latest OS support from https://www.asus.com/event/Server/OS_support_list/OS.html
Management Solution	**Software**	ASUS Control Center (Classic)
	Out of Band Remote Management	On-Board ASMB10-iKVM for KVM-over-IP
Dimension		800mm x 440mm x 88mm (2U) 31.50" x 17.22" x 3.46"
Net Weight Kg (CPU, DRAM & HDD not included)		34 kg
Gross Weight Kg (CPU, DRAM & HDD not included, Packing include)		44 kg
Power Supply (following different configuration by region)		1+1 Redundant 1600W 80 PLUS Platinum Power Supply 1+1 Redundant 2200W 80 PLUS Platinum Power Supply
Environment		Operation temperature: 10℃ ~ 35℃ Non operation temperature: -40℃ ~ 70℃ Non operation humidity: 20% ~ 90% (Non condensing)

Anexo 5

Características Procesador AMD Epyc 7313P

AMD EPYC™ 7313P

General Specifications

Platform:	Server
Product Family:	AMD EPYC™
Product Line:	AMD EPYC™ 7003 Series
# of CPU Cores:	16
# of Threads:	32
Max. Boost Clock ⓘ:	Up to 3.7GHz
Base Clock:	3.0GHz
L3 Cache:	128MB
Default TDP:	155W
AMD Configurable TDP (cTDP):	155-180W
CPU Socket:	SP3
Socket Count:	1P
Launch Date:	3/15/2021

Connectivity

PCI Express® Version:	PCIe 4.0 x128
System Memory Type:	DDR4
Memory Channels:	8

System Memory Specification:	Up to 3200MHz
Per Socket Mem BW:	204.8 GB/s

Graphics Capabilities

Integrated Graphics:	No

Product IDs

Product ID Boxed:	100-100000339
Product ID Tray:	100-000000339

Key Features

Supported Technologies:	AMD Infinity Guard AMD Infinity Architecture
Workload Affinity:	EDA Media streaming Value per core CAE/CFD/FEA

Anexo 6

Características Procesador Intel i7 7700

Procesador Intel® Core™ i7-7700

caché de 8M, hasta 4,20 GHz

Especificaciones

Esencial

Conjunto de productos	7th Generation Intel® Core™ i7 Processors
Nombre de código	Products formerly Kaby Lake
Segmento vertical	Desktop
Número de procesador	i7-7700
Estado	Launched
Fecha de lanzamiento	Q1'17
Litografía	14 nm
Condiciones de uso	PC/Client/Tablet
Precio recomendado para clientes	$303.00

Especificaciones de la CPU

Cantidad de núcleos	4
Cantidad de subprocesos	8
Frecuencia turbo máxima	4,20 GHz
Frecuencia de la Tecnología Intel® Turbo Boost 2.0‡	4.20 GHz
Frecuencia básica del procesador	3,60 GHz
Caché	8 MB Intel® Smart Cache
Velocidad del bus	8 GT/s
Cantidad de enlaces QPI	0

TDP	65 W

Información adicional

Opciones integradas disponibles	Yes
Hoja de datos	Ver ahora

Especificaciones de memoria

Tamaño de memoria máximo (depende del tipo de memoria)	64 GB
Tipos de memoria	DDR4-2133/2400, DDR3L-1333/1600 @ 1.35V
Cantidad máxima de canales de memoria	2
Compatible con memoria ECC ‡	No

Gráficos de procesador

Gráficos del procesador ‡	Gráficos HD Intel® 630
Frecuencia de base de gráficos	350 MHz
Frecuencia dinámica máxima de gráficos	1.15 GHz
Memoria máxima de video de gráficos	64 GB
Compatibilidad con 4K	Yes, at 60Hz
Resolución máxima (HDMI)‡	4096x2304@24Hz
Resolución máxima (DP)‡	4096x2304@60Hz
Resolución máxima (eDP - panel plano integrado)‡	4096x2304@60Hz
Compatibilidad con DirectX*	12
Compatibilidad con OpenGL*	4.5
Intel® Quick Sync Video	Yes
Tecnología Intel® InTru™ 3D	Yes

Tecnología Intel® Clear Video HD	Yes
tecnología Intel® de video nítido	Yes
Cantidad de pantallas admitidas ‡	3
ID de dispositivo	0x5912

Opciones de expansión

Escalabilidad	1S Only
Revisión de PCI Express	3,0
Configuraciones de PCI Express ‡	Up to 1x16, 2x8, 1x8+2x4
Cantidad máxima de líneas PCI Express	16

Especificaciones del paquete

Zócalos compatibles	FCLGA1151
Máxima configuración de CPU	1
Especificación de solución térmica	PCG 2015C (65W)
$T_{JUNCTION}$	100°C
Tamaño de paquete	37.5mm x 37.5mm

Tecnologías avanzadas

Compatible con la memoria Intel® Optane™ ‡	Yes
Tecnología Intel® Turbo Boost ‡	2,0
Tecnología Intel® Hyper-Threading ‡	Yes
Intel® Transactional Synchronization Extensions – New Instructions	Sí
Intel® 64 ‡	Yes
Conjunto de instrucciones	64-bit
Extensiones de conjunto de instrucciones	Intel® SSE4.1, Intel® SSE4.2, Intel® AVX2

Estados de inactividad	Yes
Tecnología Intel SpeedStep® mejorada	Yes
Tecnologías de monitoreo térmico	Yes
Tecnología de protección de la identidad Intel® ‡	Yes

Seguridad y fiabilidad

Idoneidad para la plataforma Intel® vPro™ ‡	Sí
Nuevas instrucciones de AES Intel®	Yes
Secure Key	Yes
Intel® Software Guard Extensions (Intel® SGX)	Yes with Intel® ME
Extensiones de protección de la memoria Intel®	Yes
Intel® OS Guard	Yes
Tecnología Intel® Trusted Execution ‡	Yes
Bit de desactivación de ejecución ‡	Yes
Intel® Boot Guard	Yes
Programa Intel® de imagen estable para plataformas (SIPP)	Sí
Tecnología de virtualización Intel® (VT-x) ‡	Sí
Tecnología de virtualización Intel® para E/S dirigida (VT-d) ‡	Yes
Intel® VT-x con tablas de páginas extendidas (EPT) ‡	Yes

Pedido y cumplimiento

Productos compatibles

Controladores y software

Bibliografía

[ACA,] Adaptive compute acceleration platform. https://www.xilinx.com/products/silicon-devices/acap/versal.html. Accessed: 2022-08-23.

[U50,] Alveo U50 Accelerator Card. https://www.xilinx.com/products/boards-and-kits/alveo/u50.html. Accessed: 2022-08-08.

[hyp,] Arquitectura de Hyper-V | MicrosoftDocs. https://docs.microsoft.com/es-es/virtualization/hyper-v-on-windows/reference/hyper-v-architecture, note = Accessed: 2022-05-08.

[U96,] Avnet ultra96-V2 Board. https://www.avnet.com/wps/portal/us/products/new-product-introductions/npi/aes-ultra96-v2/. Accessed: 2022-07-08.

[Azk,] Azken. https://www.azken.com/. Accessed: 2022-08-08.

[Bec,] Beckhoff Automation.

[bec,] C6930-0060 | control cabinet industrial pc | beckhoff españa. https://www.beckhoff.com/es-es/products/ipc/pcs/c69xx-compact-industrial-pcs/c6930-0060.html. Accessed: 2022-07-12.

[cb3,] Cb3064-xxxx. https://download.beckhoff.com/download/document/ipc/embedded-pc/industrial-motherboards/cb3064en.pdf. Accessed: 2022-07-13.

[vir,] Comparison of platform virtualization software - Wikipedia. https://en.wikipedia.org/wiki/Comparison_of_platform_virtualization_software. Accessed: 2022-05-09.

[Deb,] Debian - The Universal Operating System. https://www.debian.org/index.es.html. Accessed: 2022-05-05.

[doc,] Docker 1.11: The first runtime built on containerd and based on oci technology. https://www.docker.com/blog/docker-engine-1-11-runc/. Accessed: 2022-05-12.

[Gig, a] Ga-h270-hd3 (rev. 1.0) specification | motherboard - gigabyte global. https://www.gigabyte.com/Motherboard/GA-H270-HD3-rev-10/. Accessed: 2022-02-08.

[Gig, b] GA-Z170XP-SLI (rev. 1.0) Overview | Motherboard - GIGABYTE Global. https://www.gigabyte.com/Motherboard/GA-Z170XP-SLI-rev-10. Accessed: 2022-04-12.

[pyt,] History and license | python documentation. https://docs.python.org/3/license.html. Accessed: 2022-05-15.

[mni,] Mnist handwritten digit database. http://yann.lecun.com/exdb/mnist/. Accessed: 2022-08-04.

[PyP,] Pypi · the python package index. https://pypi.org/. Accessed: 2022-06-20.

[tp-,] Ue300 | adaptador de red usb 3.0 a ethernet gigabit | tp-link españa. https://www.tp-link.com/es/home-networking/computer-accessory/ue300/v3/. Accessed: 2022-07-15.

[VCK,] VCK5000 Versal Development Card. https://www.xilinx.com/products/boards-and-kits/vck5000.html. Accessed: 2022-07-19.

[Alférez Zamora, 2016] Alférez Zamora, A. J. (2016). Diseño y despliegue de servicios de alta disponibilidad en la nube usando herramientas de código abierto.

[Caiazza et al., 2022] Caiazza, C., Giordano, S., Luconi, V., and Vecchio, A. (2022). Edge computing vs centralized cloud: Impact of communication latency on the energy consumption of lte terminal nodes. *Computer Communications*, 194:213–225.

[Chen et al., 2020] Chen, C., Hua, Z., Zhang, R., Liu, G., and Wen, W. (2020). Automated arrhythmia classification based on a combination network of cnn and lstm. *Biomedical Signal Processing and Control*, 57:101819.

[Inc., 2021] Inc., X. (2021). *VCK5000 Data Center Acceleration Development Kit Hardware Installation Guide*.

[Keller, 1999] Keller, M. S. (1999). Take command: cron: Job scheduler. *Linux Journal*, 1999(65es):15–es.

[Mou et al., 2017] Mou, L., Ghamisi, P., and Zhu, X. X. (2017). Deep recurrent neural networks for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(7):3639–3655.

[Orellana and Ricardo, 2017] Orellana, P. and Ricardo, J. (2017). Proposals for efficient management of fpgas within cloud computing environments. Accepted: 2017-09-07T08:36:41Z Publisher: Universidad de Castilla-La Mancha.

[Ozcalici and Bumin, 2022] Ozcalici, M. and Bumin, M. (2022). Optimizing filter rule parameters with genetic algorithm and stock selection with artificial neural networks for an improved trading: The case of borsa istanbul. *Expert Systems with Applications*, 208:118120.

[Proaño Orellana et al., 2016] Proaño Orellana, J., Caminero, B., Carrión, C., Tomas, L., Kostentinos Tesfatsion, S., and Tordsson, J. (2016). Fpga-aware scheduling strategies at hypervisor level in cloud environments. *Scientific Programming*.

[Qian et al., 2009] Qian, L., Luo, Z., Du, Y., and Guo, L. (2009). Cloud Computing: An Overview. In Jaatun, M. G., Zhao, G., and Rong, C., editors, *Cloud Computing*, pages 626–631, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

[Xilinx, 2019] Xilinx (2019). *Zynq UltraScale+ Device Technical Reference Manual, User Guide 1085.*

[Xilinx, 2020a] Xilinx (2020a). *Versal: The First Adaptive Compute Acceleration Platform (ACAP).*

[Xilinx, 2020b] Xilinx (2020b). *Zynq DPU c3.2, Product Guide.*

[Xilinx, 2020c] Xilinx (2020c). *Zynq UltraScale+ MPSoC Processing System v3.3, LogiCORE IP Product Guide.*