ULPGC
Universidad de Las Palmas de Gran Canaria

Instituto Universitario de Microelectrónica Aplicada

IUMA

# Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)


## Trabajo Fin de Máster

# Desarrollo de una plataforma *software* empotrada basada en SoC FPGA Zynq para aplicaciones de Astrofísica

**Autor:** David S. Miranda Guillén
**Tutor:** Dr. Pedro Pérez Carballo
**Fecha:** Diciembre 2022

IUMA
**t** +34 928 451 150
+34 928 451 086
**f** +34 928 451 083
**e:** iuma@iuma.ulpgc.es
**w:** www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

ULPGC Universidad de Las Palmas de Gran Canaria | Instituto Universitario de Microelectrónica Aplicada IUMA

# Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)


## Trabajo Fin de Máster

## Desarrollo de una plataforma *software* empotrada basada en SoC FPGA Zynq para aplicaciones de Astrofísica

## HOJA DE FIRMAS

| | | |
|---|---|---|
| **Alumno/a:** | David S. Miranda Guillén | Fdo.: |
| **Tutor/a:** | Dr. Pedro Pérez Carballo | Fdo.: |
| **Fecha:** | Diciembre 2022 | |

IUMA
t +34 928 451 150
+34 928 451 086
f +34 928 451 083
e: iuma@iuma.ulpgc.es
w: www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

ULPGC Universidad de Las Palmas de Gran Canaria | Instituto Universitario de Microelectrónica Aplicada IUMA

# Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)


## Trabajo Fin de Máster

## Desarrollo de una plataforma *software* empotrada basada en SoC FPGA Zynq para aplicaciones de Astrofísica

## HOJA DE EVALUACIÓN

**Calificación:** .....

**Presidente** Fdo.:

**Secretario** Fdo.:

**Vocal** Fdo.:

**Fecha:** Diciembre 2022

IUMA
t +34 928 451 150
+34 928 451 086
f +34 928 451 083
e: iuma@iuma.ulpgc.es
w: www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

# AGRADECIMIENTOS

*Como en capítulos anteriores, a mi familia por aguantar y a los que han estado.*

# RESUMEN

En este Trabajo Fin de Máster se presenta el desarrollo e implementación de un sistema *hardware/software* perteneciente a un equipo para la caracterización de la polarización del Fondo Cósmico de Microondas, y otros procesos físicos galácticos o extragalácticos que emiten en microondas en el rango de frecuencias 10-42GHz, y a grandes escalas angulares (1 grado de resolución).

Dicho sistema *hardware* está fabricado sobre un rack de 2U que actúa como continente de dos plazas ZedBoard comunicadas por su conector FMC, 4 ADC AD7768, un RTCC y una fuente de alimentación para todo el sistema. En cuanto al subsistema *software* se construye una solución AMP basada en Linux-*baremetal* para el funcionamiento del sistema. Esto permite realizar una comunicación entre alto y bajo nivel, diseñando una aplicación cliente-servidor que permita interactuar con el *hardware*.

El subsistema *software* está creada en lenguaje C++ utilizando en algunos casos un lenguaje de más bajo nivel, que permite interactuar mejor con un sistema empotrado, como es C. Se ha desarrollado el sistema, para que pueda ser utilizados por múltiples clientes, atendiendo a las características propias de este.

Para llegar a esta solución, ha sido necesario un estudio previo de las tecnologías disponibles, ya sean *software* o *hardware.* En primer lugar, escoger los sistemas operativos y lenguajes de programación adecuados que se adapten de la mejor forma al equipo disponible y, en segundo lugar, estudiar los recursos *hardware* de los que se dispone: memoria RAM, almacenamiento, periféricos, etc.

Para validar el sistema, se realizan una serie de pruebas inyectando datos de forma sintética y comprobando que las configuraciones que se realizan desde la aplicación se ven reflejadas en el comportamiento del *hardware.* Con ello se puede así comprobar que los datos capturados son los esperados y se cumplen las especificaciones del sistema planteadas.

# ABSTRACT

*This Master Thesis presents the development and implementation of a hardware/software system belonging to a equipment for the characterisation of the polarisation of the Cosmic Microwave Background, and other galactic or extragalactic physical processes that emit in microwaves in the frequency range 10-42GHz, and at large angular scales (1 degree resolution).*

*This hardware system is built on a 2U rack that acts as a container for two ZedBoard's slots connected by their FMC connector, 4 ADC AD7768, a RTCC and a power supply for the whole system. As for the software block, an AMP solution based on Linux-baremetal is built for the operation of the system. This allows communication between high and low level, designing a client-server application that allows interaction with the hardware.*

*The software system is created in C++ language, using in some cases a lower-level language that allows a better interaction with an embedded system such as C. The system has been developed so that it can be used by multiple clients, considering its own characteristics.*

*To develop this solution, it was necessary to carry out a study of the available technologies, both software and hardware. Firstly, to choose the appropriate operating systems and programming languages that best adapt to the equipment available and, secondly, to study the hardware resources available: RAM memory, storage, peripherals, etc.*

*To validate the system, a series of tests are carried out by artificially injecting data that are assimilated to reality and checking that the configurations made from the application are reflected in the behaviour of the hardware. In this way, it can be verified that the data captured are as expected and that the system specifications are met.*

# Tabla de contenidos

# Índice de figuras

# Índice de tablas

# Índice de listados

# ACRÓNIMOS

| | |
|---|---|
| FPGA | *Field-Programmable Gate Array* |
| SoC | *System On a Chip* |
| ADC | *Analog-to-digital* |
| FMC | *FPGA Mezzanine Card* |
| PL | *Programmable Logic* |
| PS | *Processing System* |
| SPI | *Serial Peripheral Interface Bus* |
| RTC | *Real-Time Clock* |
| I2C | *Inter-Integrated Circuit* |
| PMOD | *Peripheral Module* |
| GPIO | *General-Purpose Input/Output* |
| DMA | *Direct Memory Access* |
| AMP | *Asymmetric Multi-Processing* |
| SMP | *Symmetric Multi-Processing* |
| NFS | *Network File System* |
| ASIC | *Application-Specific Integrated Circuit* |
| CMOS | *Complementary Metal–Oxide–Semiconductor* |
| APU | *Accelerated Processing Unit* |
| DSP | *Digital Signal Processor* |
| ASSP | *Application Specific Standard Product* |
| AXI | *Advanced eXtensible Interface* |
| SMC | *Static Memory Controller* |
| USB | *Universal Serial Bus* |
| BSP | *Board Support Package* |
| RTOS | *Real-Time Operating System* |
| FSBL | *First Stage Boot Loader* |
| OCM | *On-Chip Memory* |
| MMU | *Memory Management Unit* |

| | |
|---|---|
| IoT | *Internet of Things* |
| LTS | *Long Time Support* |
| XSA | *Xilinx Support Archive* |
| MAC | *Media Access Control* |
| CSS | *Cascading Style Sheets* |
| DHCP | *Dynamic Host Configuration Protocol* |
| UIO | *Userspace I/O system* |
| PCI | *Peripheral Component Interconnect* |
| SCP | *Secure Copy Protocol* |
| SFTP | *Secure File Transfer Protocol* |
| SSH | *Secure Shell* |
| UART | *Universal Asynchronous Receiver-Transmitter* |

# CAPÍTULO 1. INTRODUCCIÓN

## 1.1. ANTECEDENTES

El campo de astrofísica y en general la observación de experimentos en la física requiere la captura de datos experimentales que permitan dar soporte a las teorías desarrolladas. Para ello se requieren equipos que permitan realizar la captura mediante los sensores apropiados y la conversión analógica/digital de dichos datos a velocidades de muestreo muy altas, a veces a millones de muestras por segundo. En estos casos se requieren plataformas de altas prestaciones, muchas de ellas desarrolladas sobre dispositivos FPGA (*Field-Programable Gate Array*) que sean capaces de realizar un procesamiento intensivo de los datos capturados para facilitar, primero su adaptación y transmisión y más tarde su almacenamiento en servidores de alta capacidad que sean accesibles a los investigadores. Un ejemplo de este tipo de trabajos es el que se está desarrollando en el proyecto Quijote (Q-U-I JOint TEnerife) del Instituto de Astrofísica de Canarias [1].

Los dispositivos SoC FPGA integran un conjunto de recursos para la implementación de bloques *hardware* (PL – *Processing Logic*) a medida, que puede incluir la interfaz con el dispositivo Conversor Analógico/Digital (ADC) y su posterior preprocesamiento, como por ejemplo la compresión de datos. Además, dispone de un sistema de procesamiento (PS – *Processing System*), normalmente con disponibilidad de computación multiproceso usando soluciones basadas en varios núcleos procesadores y unidades de coprocesamiento específico. El PS puede dar soporte a la ejecución de distintos *stacks* de desarrollo *software*, lo que permite la creación de una plataforma *software* compleja que da soporte al SoC.

En este trabajo se aborda la creación y diseño de una plataforma *software* que permita a un sistema electrónico basado en SoC FPGA capturar datos científicos. El equipo

se compone de un sistema de conversión ADC de 32 canales con entradas en modo diferencial y de un subsistema formado por dos ZedBoard [2] que trabajan de manera conjunta, en configuración maestra-esclava. Ambas se comunican mediante una interfaz FMC para la configuración y transferencia de datos.

Para la configuración, visualización de estado y otras operaciones del sistema se ha desarrollado una aplicación cliente-servidor que realiza estas operaciones en red mediante el uso del protocolo TCP/IP.

### 1.1.1. XILINX ZYNQ-7000 SOC FPGA

Las FPGA Xilinx son dispositivos semiconductores, que pueden ser configurados de diferentes formas por un diseñador para realizar una tarea concreta según la aplicación. Están basados en una matriz de bloques lógicos configurables (CLB) conectados a través de interconexiones programables, así como una jerarquía de interconexiones que permiten conectar los bloques entre sí. Esto crea un punto de distinción entre FPGA y los ASIC (*Application-Specific Integrated Circuit*), que se fabrican a medida para aplicaciones específicas [3]. Las FPGA son ampliamente utilizadas en el ámbito industrial para núcleos de: audio, automoción, aeroespacial, etc.

En este proyecto, se utilizan dispositivos SoC FPGA que cuentan con uno o más microprocesadores empotrados. Esto facilita la integración de diferentes tipos de sistemas operativos, solo o combinados, pudiendo crear una solución que se adecue en cada momento a la necesidad planteada. Los dispositivos Zynq-7000 [4] se basan en la arquitectura Xilinx SoC y están provistos con procesadores ARM® Cortex™ A9 de doble núcleo o un solo núcleo con numerosas funciones [5]. En la Figura 1 se puede observar que la *Application Processing Unit* (APU) está integrada junto con una matriz lógica programable Artix-7 o Kintex-7 en tecnología CMOS de 28 nm. Las FPGA tienen un importante papel en la creación de arquitecturas heterogéneas, ya que permiten la integración de aceleración *hardware* junto con funcionalidades como: DSP, CPU, señal mixta y ASSP. Además, son ampliamente utilizado en el mercado de sistemas empotrados, permitiendo iniciar un desarrollo *software* simultáneamente con el *hardware,* facilitando pruebas y simulaciones en etapas tempranas del desarrollo.

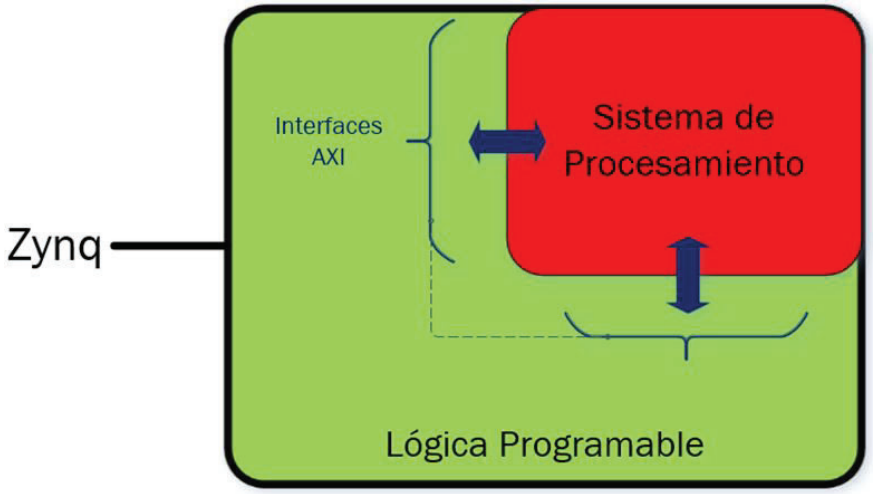



Figura 1. Diagrama de bloques de Zynq 7000 (adaptada de [2])

### 1.1.2. SISTEMAS DE ADQUISICIÓN DE DATOS

Existen numerosos casos en los que las FPGA se usan como plataforma de captura de datos en el campo científico. Por ejemplo, en el campo de la física podemos encontrar un sistema llamado Versatile Data AcQuisition (VerDAQ) [6] que es una plataforma de adquisición de datos modular y flexible diseñada para ser utilizada en una amplia gama de experimentos de física de altas energías, desde pequeños detectores de sobremesa hasta grandes experimentos en aceleradores. Esta plataforma permite una gran flexibilidad, derivada de su diseño modular, que separa la electrónica *frontend* de la de electrónica *backend*. El sistema puede ser reconfigurado tanto por *hardware* como por *firmware*, permitiendo su integración en instrumentos autónomos o integrados en amplios sistemas de adquisición de datos.

A. P. Putra, S. Fuada, Y. Aska y T. Adionose describen en [7] un sistema de captura de alta velocidad donde se puede observar un sistema SoC para la adquisición de datos digitales en comunicación de luz visible. El objetivo del sistema consiste en capturar datos en alta velocidad, separando todo el sistema en dos partes bien diferenciadas, una *hardware* y otra *software*. La parte *hardware* se establece sobre una arquitectura Zynq-700, mientras que la parte *software* hace uso de la API de Xilinx para establecer una conexión con un PC mediante Ethernet usando UDP.

En el campo de la astrofísica, como el de este proyecto, los sistemas de captura de datos basados en FPGA también son ampliamente utilizados. En [8] se describe un Sistema de adquisición de datos de alta velocidad basado en PCI que utiliza una FPGA de Xilinx. Este sistema acepta datos a través de la interfaz LVDS y permite introducir etiquetado de tiempo de los datos de forma precisa mediante relojes externos. Utilizando la interfaz host puede comunicarse para la transferencia de datos mediante DMA, además, el usuario puede asignar varios búferes en la memoria host para transferir los datos adquiridos.

En China, en el observatorio astronómico de Xinjiang y junto a la University of Chinese Academy of Sciences y otros organismos de investigación, se desarrolla un sistema acoplado a un radiotelescopio [9]. Su función consiste en la adquisición de datos de banda base digital multifunción para el muestreo, la distribución y la grabación de señales astronómicas multicanal de banda ancha. El sistema se utiliza como plataforma de verificación para probar las funciones del sistema *backend* digital del QTT (radiotelescopio QiTai) mediante la observación de púlsares. Para ello utilizan una tarjeta SNAP2 como convertidor digital de banda base para digitalizar, canalizar y empaquetar la señal recibida. Permite configurarse dinámicamente desde un solo canal hasta ocho canales con un ancho de banda máximo de 4096 MHz.

Por último, un sistema incorporado en la Estación Espacial Internacional se describe en [10], donde se presenta un *software* de adquisición de datos y control para el funcionamiento del Mini-Extreme Universe Space Observatory (EUSO), un telescopio de fluorescencia basado en el espacio para la observación de extensas lluvias de aire y fenómenos atmosféricos. La adquisición de datos, el mantenimiento y el control del subsistema se realiza utilizando un chip Xilinx Zynq XC7Z030 interconectado con un módulo de CPU PCIe/104 a través del sistema de procesamiento Zynq integrado. La parte *software* perteneciente a la interfaz de control del instrumento, se gestiona mediante un diseño C++ orientado a objetos, que puede ejecutarse tanto de forma autónoma como interactiva, según sea necesario.

## 1.2. OBJETIVOS

El trabajo tiene como objetivo el desarrollo de una plataforma *software,* formada por las librerías necesarias y el desarrollo de las aplicaciones, que permitan interactuar a un cliente con un servidor TCP/IP integrado en un sistema empotrado. El objetivo final es realizar su configuración, obtener su estado y realizar la transferencia de datos, previamente digitalizados mediante un conversor analógico/digital. Todo ello está orientado al trabajo en entornos científicos. El sistema empotrado se implementa haciendo uso de un SoC FPGA de la familia Zynq-7000 de Xilinx.

Para ello identifican los siguientes objetivos:

- Estudiar la arquitectura de la plataforma *hardware* de referencia.
- Estudiar la metodología de desarrollo de Linux Empotrado (Petalinux) y de sus componentes principales, incluida la integración de soluciones AMP.
- Definir la arquitectura de la plataforma *software* para el ámbito de aplicación.
- Desarrollar las aplicaciones cliente y servidor con criterios de optimización en recursos de memoria y consumo de potencia.
- Integrar la aplicación con el *hardware*, validando la funcionalidad de los diferentes bloques.
- Validar la plataforma *software* desarrollada
- Documentar el trabajo realizado.

## 1.3. PETICIONARIO

Actúa como solicitante del trabajo la División Sistemas Industriales y CAD (SICAD) del Instituto Universitario de Microelectrónica Aplicada (IUMA) de la Universidad de Las Palmas de Gran Canaria.

## 1.4. ESTRUCTURA DEL DOCUMENTO

Esta memoria se organiza en diez capítulos. Una vez presentado el contexto del trabajo, en el capítulo 2 se introduce la plataforma utilizada, el sistema operativo, los lenguajes de programación valorados y el flujo de desarrollo utilizado. Todo ello se complementa con la descripción del equipo utilizado presentado en el capítulo 3, conformando el marco de referencia básico para el desarrollo del trabajo

En el capítulo 4 se explica la arquitectura global del SoC, incluyendo los bloques que la constituyen y el flujo de datos diseñado. El capítulo 5 describe la arquitectura *software* del SoC, el tipo de solución heterogénea elegida de entre las distintas posibilidades existentes, en este caso una implementación de tipo AMP. En el capítulo 6 se presentan los pasos necesarios para generar el sistema operativo optimizado, en este caso Petalinux,. en función de la plataforma diseñada.

Las distintas aplicaciones desarrolladas se presentan en el capítulo 7, siendo clave para para entender el *stack* de procesamiento del sistema. En este trabajo se. desarrolla tanto una aplicación *baremetal* como otras basadas en Linux, coordinadas utilizando el entorno OpenAMP.

En el capítulo 8 se explica la gestión de los datos capturados y su almacenamiento. Finalmente, el sistema se evalúa en conjunto y se realizan pruebas de rendimiento, tal como se recoge en el capítulo 9.

El trabajo se cierra presentando las conclusiones globales del trabajo y se presentan posibles trabajos futuros que dan continuidad a la línea de desarrollo de este tipo de sistemas basados en SoC FPGA.

# CAPÍTULO 2. PLATAFORMA EMPOTRADA

## 2.1. INTRODUCCIÓN

En este capítulo se explica los dispositivos SoC Zynq-7000 mostrando de una manera general su arquitectura *hardware* a nivel de grandes bloques y sus componentes e interconexiones. Por otro lado, se presentan los principios básicos de los sistemas empotrados como referente para el desarrollo del presente trabajo.

## 2.2. BLOQUES PS Y PL

La familia de dispositivos Zynq-7000 se desarrolla a principios del año 2010 de la mano de Xilinx, combinando la lógica de una FPGA con un procesador dedicado. En este caso se basa en un procesador ARM® Cortex™ A9 de doble núcleo que pueden operar a una frecuencia de hasta 1 GHz (666 MHz para el modelo usado).

Como se muestra en la Figura 2, la arquitectura de un dispositivo SoC Zynq consta de un sistema de procesamiento (PS, *Processing System*) y la lógica programable (PL, *Programmable Logic*), conectadas a través de un conjunto de interfaces AXI (*Advanced eXtensible Interface*) de propósito general (GP) y de altas prestaciones (HP).

El bloque PS da soporte al dominio *software* del Zynq-7000, obteniendo un sistema diseñado *hardware* y *software.* A través de la herramienta Xilinx Vitis [11] se puede desarrollar el *software* para los procesadores de sus dispositivos propietarios, ya que el compilador soporta el desarrollo de aplicaciones empotradas para diferentes procesadores integrados en el ecosistema de Xilinx. Además, es capaz de generar el *lincador* a medida del sistema en función del mapa de memoria particular definido.

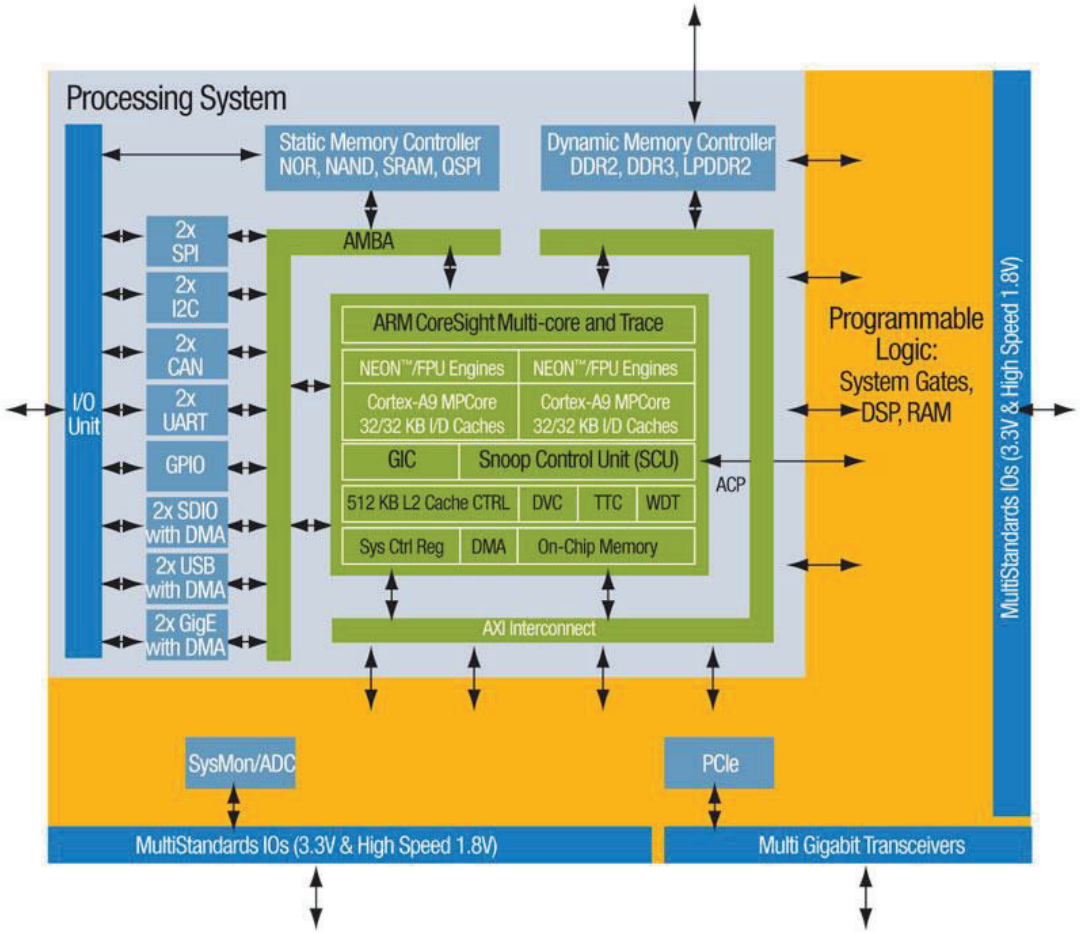



Figura 2. Modelo de arquitectura de Zynq

El bloque PS se divide en diferentes bloques, como se puede ver en la Figura 3. Entre estos bloques se encuentran:

- Unidad de Procesamiento de Aplicación (APU)
    - ARM Cortex-A9 arquitectura ARM.
    - Diferentes elementos para gestionar el sistema: puerto de coherencia del acelerador (ACP), unidad de control de *snoop* (SCU), controlador de interrupciones generales (GIC), *watchdog* y temporizadores, DMA y memoria SRAM de 256 KB.
- Interfaces de memoria: *Quad-SPI Controller,* DDR *Controller*, *Static Memory Controller (SMC)* [12].

- Múltiples periféricos de E/S: SPI (*Serial Peripheral Interface Bus*), UART (*Universal Asynchronous Receiver-Transmitter*), I2C (*Inter-Integrated Circuit*), USB (*Universal Serial Bus)*, etc.

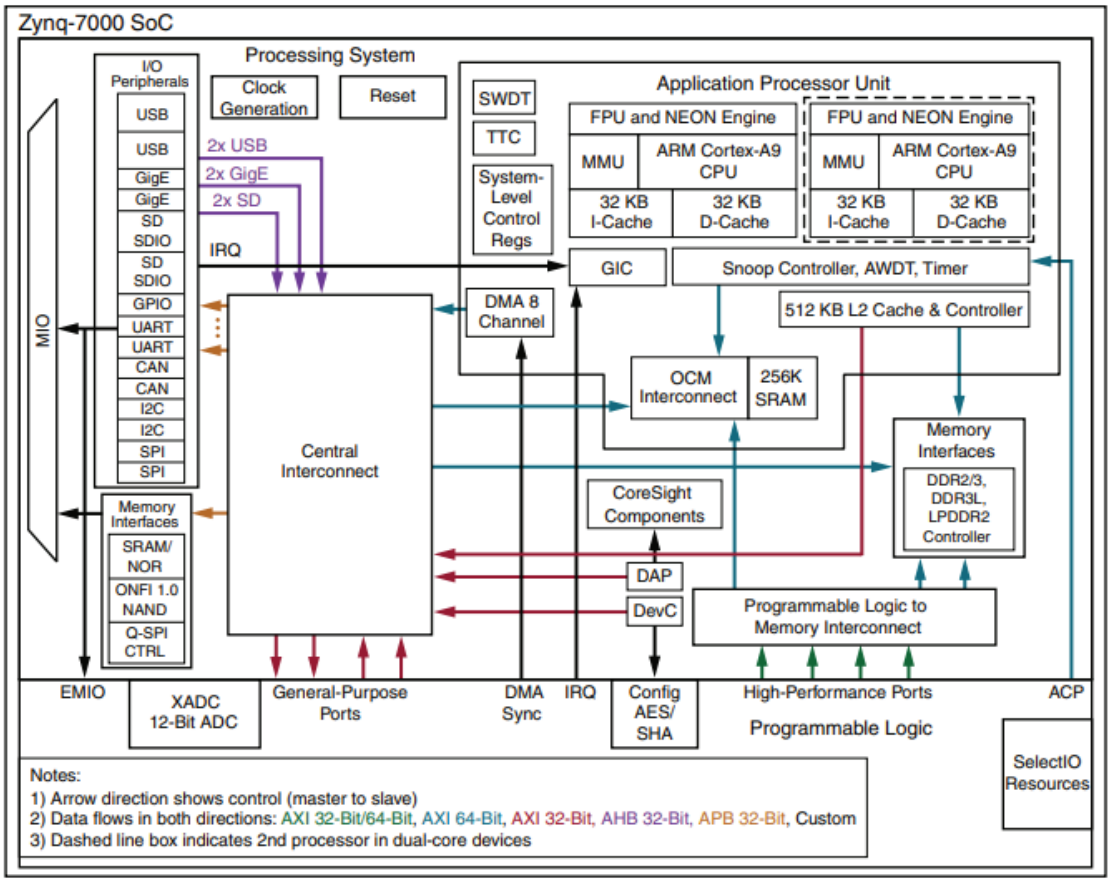



Figura 3. Detalle del bloque PS [5]

Por otro lado, el bloque PL proporciona recursos lógicos configurables para la implementación *hardware* de distintos bloques, ya sean de interfaz, de control o de procesamiento intensivo de datos. Utiliza la familia Kintex-7 o la Artix-7, dependiendo del dispositivo elegido. La Figura 4 muestra destaca algunas partes de este bloque, de donde se puede remarcar:

- Bloques de lógica configurable (CLBs). Estos componentes se encuentran en disposición de matrices bidimensionales, formando agrupaciones de elementos lógicos interconectados a otros recursos similares mediante interconexiones programables. Un CLB se compone de dos *slices* que contienen los recursos necesarios para implementar circuitos lógicos secuenciales y combinacionales.

- Bloques de entrada y salida (IOBs). Estos proporcionan una interfaz entre los '*pads*' de los dispositivos físicos utilizados para conectarse a los circuitos externos y los recursos del PL.

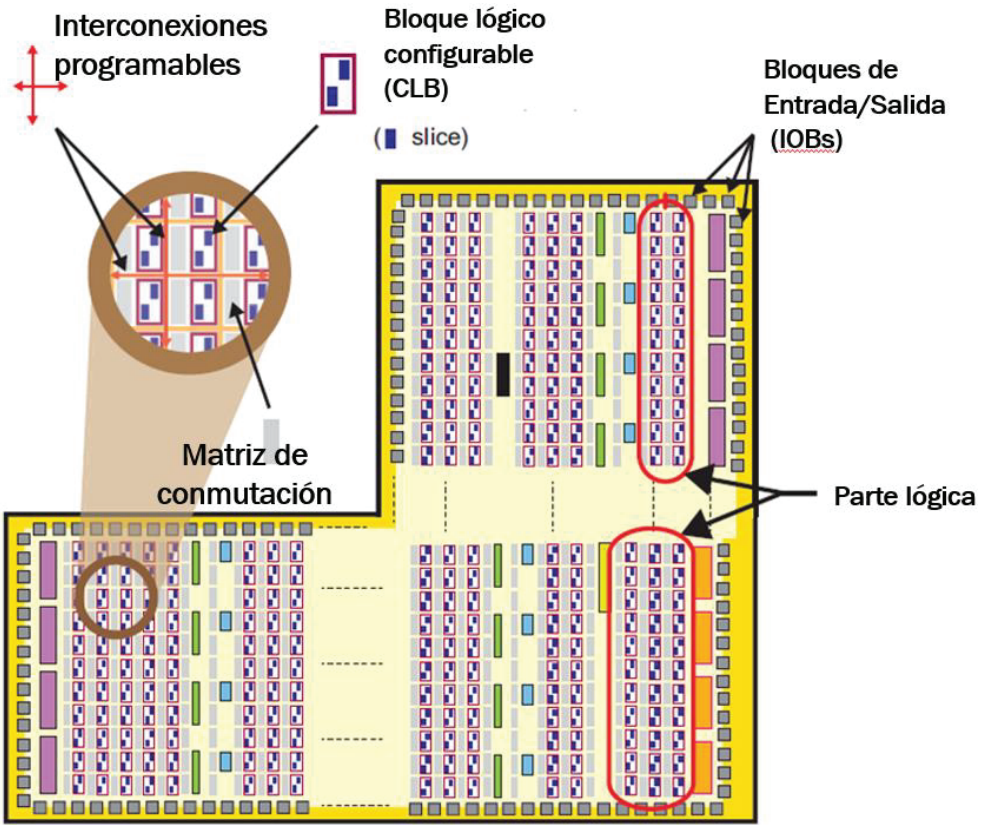



Figura 4. Bloque PL y sus partes (adaptada de [4])

## 2.3. SISTEMA EMPOTRADO

Podemos considerar la definición de un sistema empotrado desde dos puntos de vista, ya sea desde una visión *hardware* o desde una visión *software.* En este sentido, se puede considerar un sistema empotrado a un sistema *hardware* que tiene incorporado un *stack software* o también a un sistema de procesamiento específico optimizado para llevar a cabo una serie de funciones dedicadas.

Pueden formar parte de sistemas más grandes y estos solo encargarse de tareas concretas según la aplicación de destino. Las características de estos sistemas debe ser la eficiencia (Figura 5).

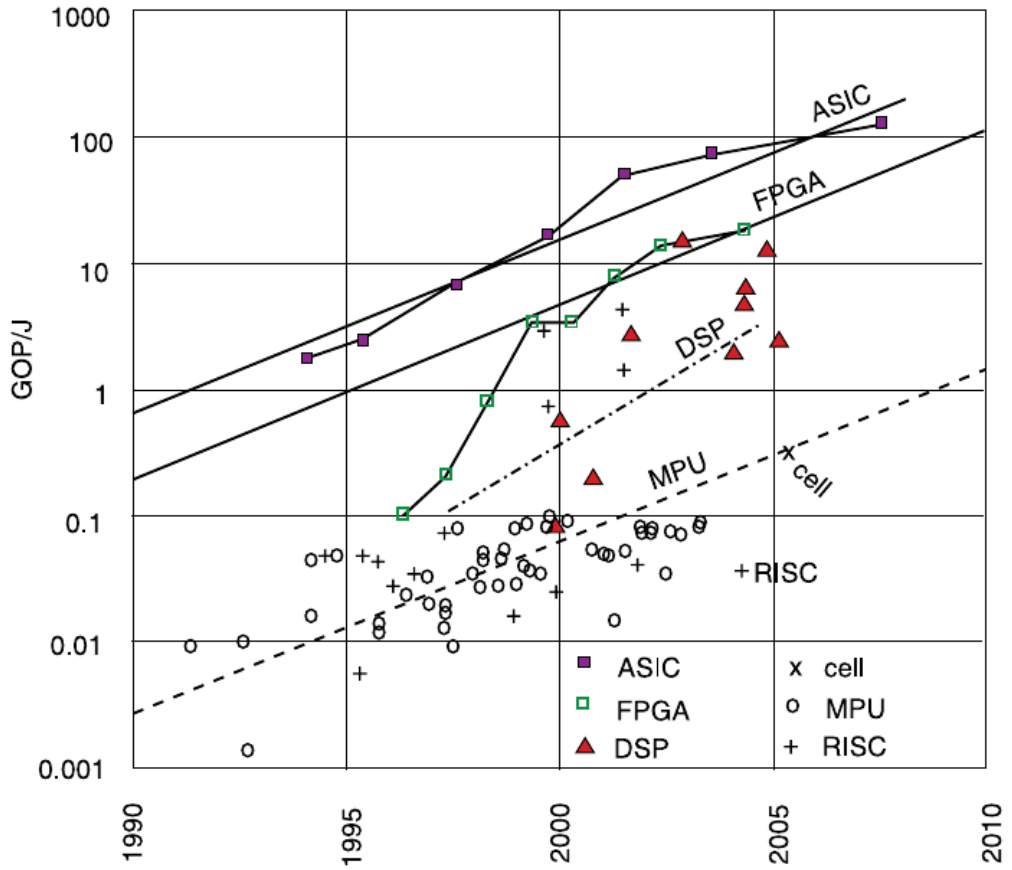



Figura 5. La eficiencia energética en función del tiempo y la tecnología [13]

- **Energía**. Es necesario mantener un equilibrio del número de operaciones por Joule, que varía dependiendo de la tecnología (Figura 5).
- **Tiempo de ejecución**. Se debe explotar al máximo la arquitectura disponible, evitando ineficiencias en el mapeado de las aplicaciones.
- **Tamaño del código.** La carga dinámica de código es una excepción en este tipo de sistemas, debido a la limitada conectividad y problemas de seguridad. Por tanto, el código que se ejecute en este sistema debe estar almacenado en memorias de rápido acceso del sistema.
- **Peso.** Los sistemas deben ser ligeros, característica importante para la elección de un sistema determinado.
- **Coste.** La competitividad en este tipo de mercados es muy elevada, por lo que los costes, tanto en *hardware* como en *software* deben reducirse al mínimo posible para implementar las funciones requeridas.

## 2.4. LENGUAJE DE PROGRAMACIÓN

La elección del lenguaje de programación a utilizar es un tema fundamental que debe ser tratado con rigurosidad. Se deben tener en cuenta aspectos como la seguridad, el soporte, la velocidad, la cercanía al *hardware*, la eficiencia y la simplicidad.

En este caso, debido a que se trata de una aplicación a bajo nivel, se busca principalmente, de entre las características anteriores, que tenga soporte para este, poco uso de memoria y la mayor eficiencia y velocidad posible. Si estas no se cumplen no será una elección viable. Por tanto, se busca una solución que unifique de la mejor forma posible todos los aspectos requeridos.

En la Figura 6 se comparan diferentes lenguajes de programación que se han valorado para la implementación del proyecto, teniendo en cuenta sus características principales. En primer lugar, se encuentra el lenguaje C [14], [15], desarrollado por Dennis Ritchie entre 1969 y 1972, es ampliamente usado para aplicaciones de bajo nivel debido a que es altamente configurable y facilita el acceso a la capa de abstracción del *hardware.* Entre sus desventajas se encuentra que existe numeroso código *legacy*, con vulnerabilidades no solucionadas, especialmente en la seguridad de acceso a memoria.

El lenguaje C++ [16], desarrollado por Bjarne Stroustrup en 1979, se crea con el propósito de extender el lenguaje C, aportando mecanismos que permiten la manipulación de objetos. Es un lenguaje multipropósito con un largo recorrido y una gran versatilidad. Entre sus desventajas, se encuentra que posee una complejidad elevada de uso, incluida la gestión de memoria.

Python [17] es un lenguaje de programación interpretado. Sus características principales son:

- La curva de aprendizaje es muy suave,
- Alta versatilidad,
- Permite implementar aplicaciones *software* de manera rápida,
- Gran soporte de la comunidad de código abierto,

- Es extremadamente lento para el uso planteado,
- Tiene un elevado consumo de memoria.

En contrapartida se encuentra Rust [18]. Se trata de un lenguaje multiparadigma creado con el propósito de sustituir a C++, proporcionando mayor velocidad y seguridad en el uso de la memoria. Sin embargo, aunque sus ventajas son extraordinarias, es un lenguaje joven y que aún requiere de recorrido para establecerse como un lenguaje estándar y con un amplio uso.

Por último, se encuentra Java [19], lenguaje multipropósito ampliamente utilizado. Es un lenguaje con el que se pueden implementar soluciones *software* con un ciclo de desarrollo relativamente corto. Sin embargo, al igual que Python, para el propósito de este proyecto, es un lenguaje lento, que consume elevados recursos de memoria y que además no tiene soporte para su uso en aplicaciones empotradas, con acceso directo al *hardware*.

| COMPARATIVA SOFTWARE | C | C++ | PYTHON | RUST | JAVA |
|---|---|---|---|---|---|
| SEGURIDAD | ✗ | ✓ | ✓ | ✓ | ~ |
| SOPORTE | ✓ | ✓ | ✓ | ~ | ✓ |
| VELOCIDAD | ✓ | ✓ | ~ | ✓ | ~ |
| CERCANÍA AL HARDWARE | ✓ | ✓ | ✗ | ✓ | ✗ |
| EFICIENCIA | ✓ | ✓ | ~ | ✓ | ~ |
| SIMPLICIDAD | ~ | ✗ | ✓ | ✗ | ✓ |
| VENTAJAS | • Altamente configurable<br>• Muy cercano al *hardware* | • Multipropósito<br>• Versátil (bajo y alto nivel) | • Desarrollo ágil<br>• Simplicidad<br>• Numerosas librerías | • Cercano al hardware<br>• Seguro<br>• Robusto | • Multipropósito<br>• Simplicidad |
| DESVENTAJAS | • Código *legacy*<br>• Vulnerabilidad<br>• *Memory leak* | • Complejidad elevada<br>• *Memory leak* | • Lento<br>• Consumo de memoria | • Joven<br>• Complejidad elevada | • Lento<br>• Consumo de memoria<br>• No hay soporte a bajo nivel |

✓ ALTO ~ MEDIO ✗ BAJO

Figura 6. Cuadro comparativo de servidores web

### 2.4.1. LENGUAJE DE PROGRAMACIÓN ELEGIDO

De los lenguajes estudiados anteriormente se eligen aquellos que sean adecuados para este proyecto, teniendo en cuenta que deben adaptarse a la plataforma empleada y respetando los criterios y métricas que permitan crear un sistema eficiente.

Viendo las necesidades del sistema, se optan por los lenguajes C y C++. En el caso de C permiten la comunicación de una manera directa con la plataforma *hardware*, además se puede cargar directamente en la memoria y ejecutarse, ya que tiene instrucciones que pueden incrementar el rendimiento del sistema.

Por otro lado, C++ se puede emplear como lenguaje de propósito general, dando una gran flexibilidad para crear aplicaciones de alto nivel.

Por tanto, teniendo en cuenta una estructura en capas en la que se organiza el desarrollo del *software* empotrado podemos concluir que se utiliza C para aquellas funcionalidades cercanas al *hardware* y C++ para dar soporte a las aplicaciones de usuario.

## 2.5. FLUJO DE DESARROLLO *SOFTWARE*

El flujo de desarrollo [20] global que se plantea para el desarrollo del *software* utilizado en este trabajo se muestra en la Figura 7.

Se parte de la toma de requisitos, donde se anotan todas las cuestiones referentes al proyecto: necesidades, funcionalidades, objetivos, etc. Para que este paso sea eficaz, se deben analizar los distintos casos posibles, ya que la mejor forma de combatir los errores en el desarrollo *software* es la prevención. Los hitos deben ser reales, es decir, que se puedan cumplir en tiempo y forma.

A continuación, se aborda la fase de diseño, donde se crea la arquitectura de la aplicación, incluyendo bloques principales, interfaces y funcionalidades ya sean físicas o de diseño.

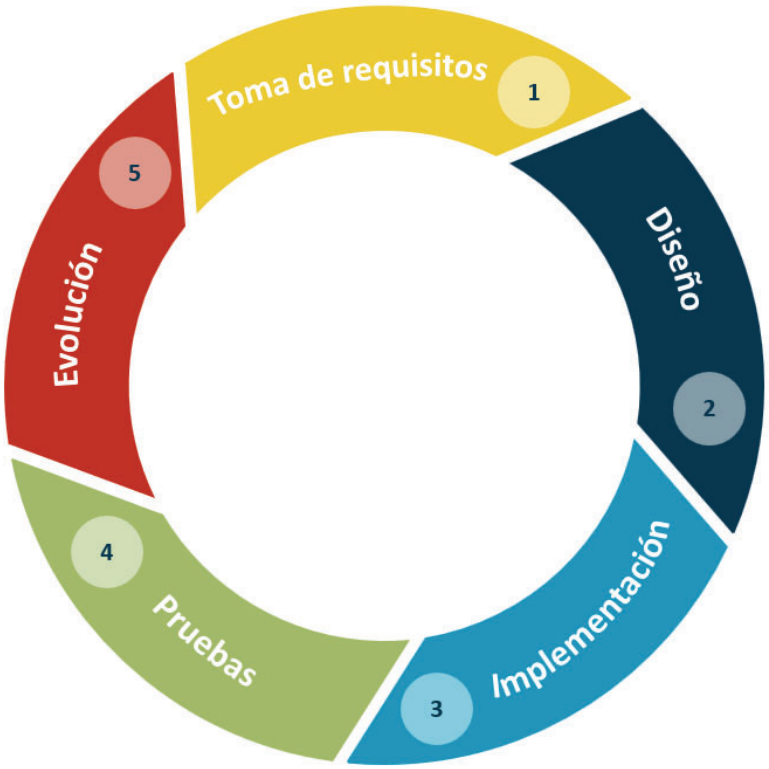



Figura 7. Flujo de desarrollo *software*

La fase siguiente representa la implementación de los pasos anteriores con el comienzo del desarrollo *software*. El desarrollo puede ser incremental, añadiendo en cada iteración funcionalidades específicas a la aplicación final. En este paso, podrán surgir problemas que deberán de ser subsanados de manera que no se desvíe el objetivo principal. De lo contrario, esto puede resultar en un funcionamiento diferente a lo esperado.

Las pruebas del código representan un paso fundamental, donde se deben realizar test ya sean funcionales [21] o unitarios [22]. Estos permiten determinar con certeza, si el código desarrollado cumple con las especificaciones exigidas e identificar y corregir todos aquellos errores presentes en el código. Abordar la corrección de los *bugs* de forma errónea o tardía puede suponer una pérdida importante de tiempo y dinero para el proyecto.

Por último, la evolución del *software* se producirá cuando se hallan alcanzados los objetivos planteados y la aplicación esté terminada. En este punto, se analiza la solución final y se estudian que nuevas funcionalidades o características podrían implementarse en

una nueva versión. Este paso inicia un nuevo flujo si se deciden implementar nuevas funcionalidades.

## 2.6. METODOLOGÍAS ÁGILES

Para el correcto desarrollo del proyecto, se han seguido los principios de las metodologías ágiles, las cuales permiten adaptar de forma rápida y efectiva la forma de trabajar a las condiciones del proyecto, alcanzando flexibilidad e inmediatez en la capacidad de amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno.

Permiten gestionar los proyectos de una forma flexible, eficaz y autónoma incrementando la productividad y reduciendo los costes. De este pensamiento nace el manifiesto ágil [23], el cual recoge 12 reglas fundamentales que deben seguirse para realizar un buen desarrollo *software*:

- La mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de *software* con valor.
- Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- Entregamos *software* funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
- Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El *software* funcionando es la medida principal de progreso.

- Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados.
- A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Dentro de estas metodologías se ha seguido el método Kanban [24], el cual consiste en un método de gestión de proyectos que permite de una forma visual establecer y procesar los flujos de trabajo y la carga de este. Se basa en tableros *Kanban* (Figura 8), donde la información se organiza en columnas representando una etapa del trabajo (trabajo pendiente, en progreso, etc.). Las tareas se representan como tarjetas individuales, que según van cambiando de estado van avanzando por las diferentes columnas. De esta forma se establece como metodología de trabajo el desarrollo incremental.

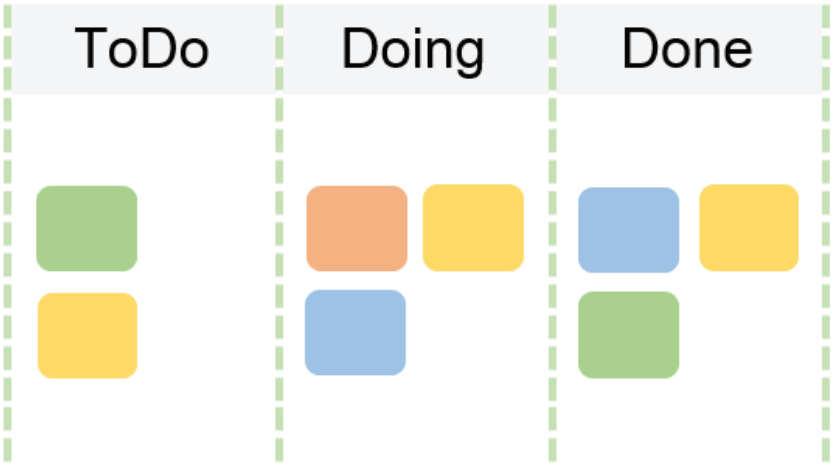



Figura 8. Tablero Kanban

## 2.7. CONTROL DE VERSIONES

El control de versiones [25] es la tarea de rastrear y gestionar los cambios que se han producido en el código fuente. Son herramientas *software* que permiten controlar el histórico del código fuente a lo largo de todo un proyecto. Este ayuda a trabajar de una forma rápida e inteligente, aumentando las implementaciones exitosas, ya que almacena todas las modificaciones del código en un tipo de base de datos especial, permitiendo ir *hacia atrás en el tiempo* cuando sea preciso.

De entre estos se ha optado por uno de los más conocidos y ampliamente utilizados Git [26], un proyecto de código abierto desarrollado originalmente por Linus Torvalds. Presenta una arquitectura distribuida, es decir, permite albergar en la copia de trabajo del código de cada desarrollador el historial completo de todos los cambios, proporcionando flexibilidad, seguridad y rendimiento.

## 2.8. DOCUMENTACIÓN

Todas las aplicaciones que se describen en este trabajo han sido documentadas con el *software* Doxygen [27] junto con un estilo CSS [28] para darle un mejor aspecto visual. Doxygen es un programa de uso estándar y gratuito usado ampliamente en la industria del *software,* que se encarga de generar la documentación de un proyecto de programación a partir de los archivos con los códigos fuentes de este. Posee una amplia versatilidad en cuanto a que no se limita a un único lenguaje de programación, sino que soporta lenguajes como C, Java y C++, entre otros.

Admite numerosas configuraciones, adaptándose a la necesidad de cada proyecto y lenguaje. Entre sus principales ventajas se encuentran:

- Generar documentación en formato HTML, la cual permite poder navegar a través de ella de una forma cómoda mediante un navegador. Crear un manual en LateX [29] o generar manuales en otros formatos conocidos como los de Microsoft Word, PDF y páginas *man* de Unix [30].

- Tiene la posibilidad de configurarse para para que extraiga la estructura del código de los archivos fuente no documentados. También puede visualizar las relaciones entre los distintos elementos mediante gráficos de dependencia de inclusión, diagramas de herencia y diagramas de colaboración, que se generan automáticamente.
- Es compatible con la mayoría de los sistemas operativos ya sean Mac OS, Linux o Microsoft, además de ser ampliamente portable.

## 2.9. CONCLUSIÓN

En este capítulo se ha estudiado de forma general los dispositivos Zynq-7000 dando a conocer sus principales bloques PS y PL y cómo se comunican entre ellos además de sus características principales. También, se ha dado una pincelada sobre lo que supone un sistema empotrado y sus métricas.

Por otro lado, se han dado a conocer algunos de los lenguajes de programación que pueden usarse para la creación de las aplicaciones que irán en nuestro sistema, analizando sus ventajas y desventajas que nos permitirá en capítulos posteriores elegir la solución más adecuada. Además, se establece el flujo con el que se desarrollará el proyecto durante sus etapas.

# CAPÍTULO 3. ARQUITECTURA HARDWARE DEL EQUIPO

## 3.1. INTRODUCCIÓN

En este capítulo se introduce la plataforma *hardware* sobre la que se desarrolla el proyecto. Se dará una breve explicación de los elementos que conforman el equipo para así poder comprender de una mejor manera el conjunto global del proyecto.

## 3.2. DESCRIPCIÓN DEL EQUIPO

El equipo que se va a usar en este proyecto está compuesto por una fuente de alimentación, dos placas ZedBoard y dos placas con ADC conectadas por FMC que permiten realizar la captura de 32 canales de señales analógicas diferenciales, todo esto en un rack de tamaño 2U. Además, dispone de diversos conectores exteriores para comunicarse con este.

En la Figura 9 se puede apreciar una vista cenital del equipo completo, remarcando las partes comentadas anteriormente.

### 3.2.1. FUENTE DE ALIMENTACIÓN

La fuente de alimentación genera 12V y dispone de varios reguladores de tensión para las diferentes tensiones de funcionamiento del sistema (1.0, 1.2, 3.3 y 5.0 V). Los reguladores actúan para mantener la potencia necesaria en la placa de prototipado según se muestra en la Figura 10.

Como se puede apreciar la placa puede funcionar hasta tensiones próximas a 6V.

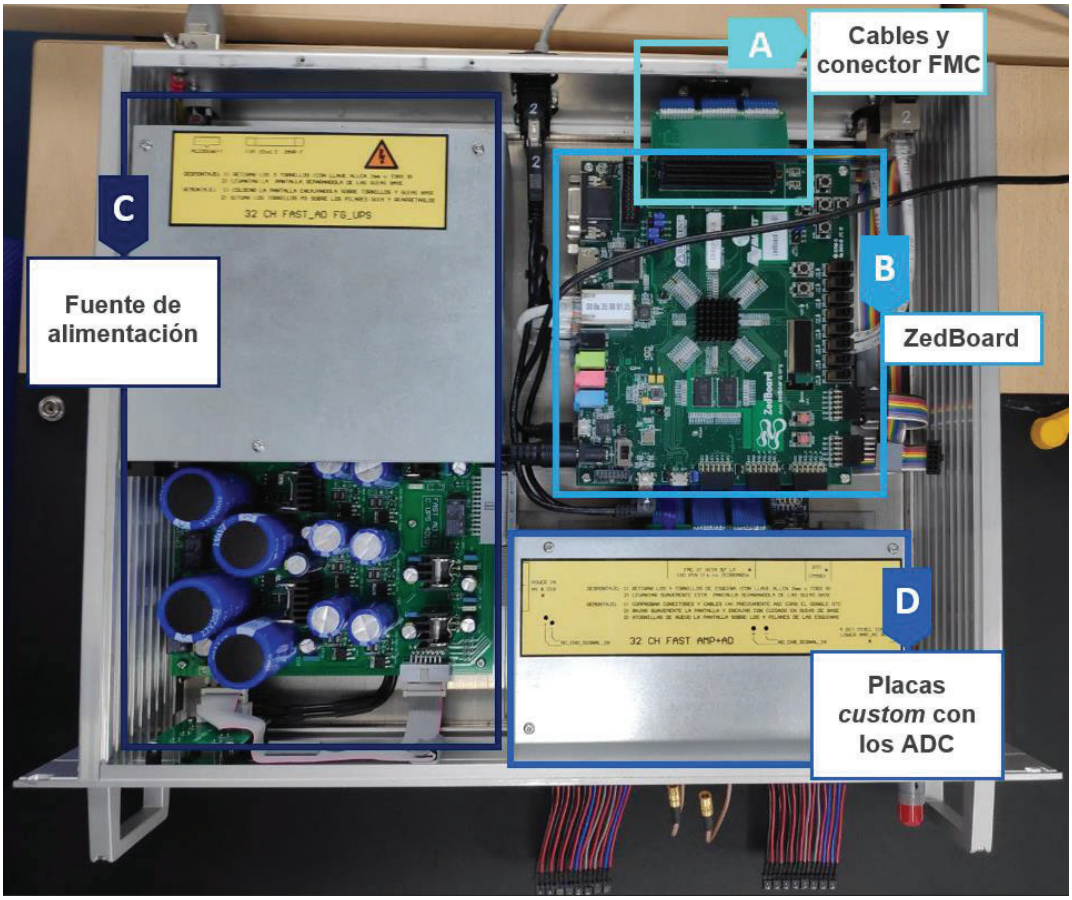



Figura 9. Equipo utilizado

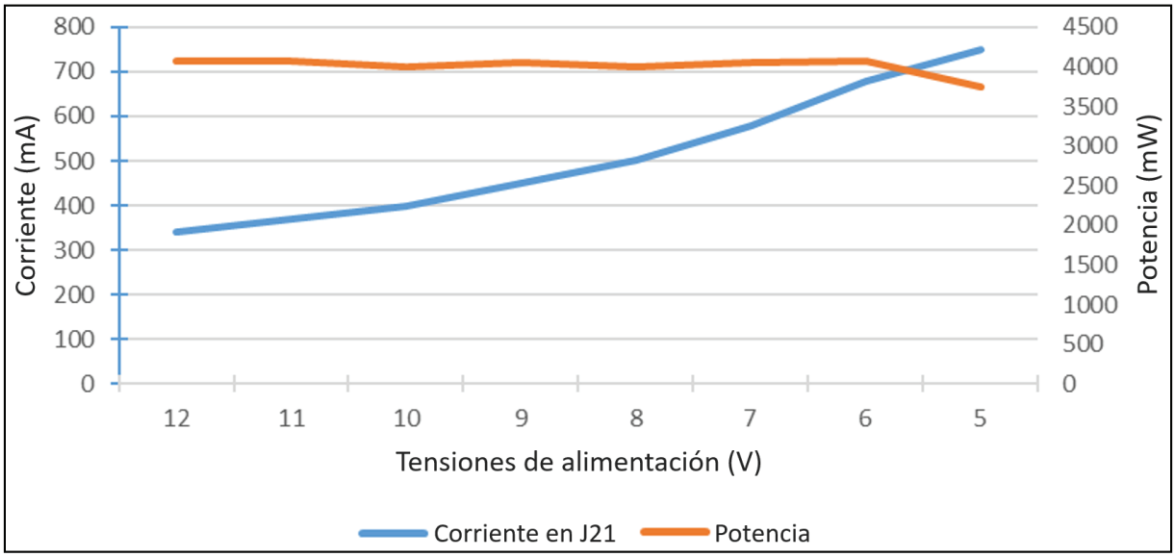



Figura 10. Variación de la corriente y potencia en función de la caída de la tensión de alimentación

### 3.2.2. CABLES Y CONECTOR FMC

Los conectores FMC LPC que comunican la placa ADC con la placa ZedBoard, permiten conectar cada una de estas a las placas que contienen los ADC con un total de 16 canales cada una, con 8 canales por conversor y una precisión de 24 bits.

Los conectores (LPC), de 40 pines, proporcionan 68 señales de datos de modo *single-ended* o 34 pares diferenciales definidos por el usuario. Estos conectores siguen el estándar FMC ANSI/VITA 57.1 el cual define una interfaz electromecánica para un sistema de tarjeta-FPGA. [31].

Los cables utilizados, de la marca Samtec [32], [33], cumplen las especificaciones antes comentadas y proporcionan aislamiento entre pares para evitar diafonías entre estos, de modo que todas las señales sean estables y resistentes a perturbaciones externas.

### 3.2.3. CONVERSOR ADC

El conversor utilizado es el AD7768 (Figura 11) el cual admite diferentes configuraciones paras las interfaces de datos, sincronización y captura de datos. Este permite muestrear 8 canales de datos a la vez, alcanzando un rango dinámico de 108 dB con un ancho de banda máximo de 110,8 kHz [34].

Además, soporta distintas opciones de configuración y funcionamiento, como la decimación, el modo de funcionamiento (rápido, medio o lento), modo de configuración (SPI o PIN), etc. que se tratarán en capítulos posteriores cuando se hable de la interfaz de configuración del sistema.

### 3.2.4. LA PLACA ZEDBOARD

Para este proyecto se han evaluado diferentes placas, entre las que se encuentran las de la familia Zynq-7000. Entre las métricas evaluadas para elegir la FPGA más adecuadas se encuentra: latencia entre procesos, escalabilidad de la plataforma, potencia de cómputo, periféricos, precio y escalabilidad.

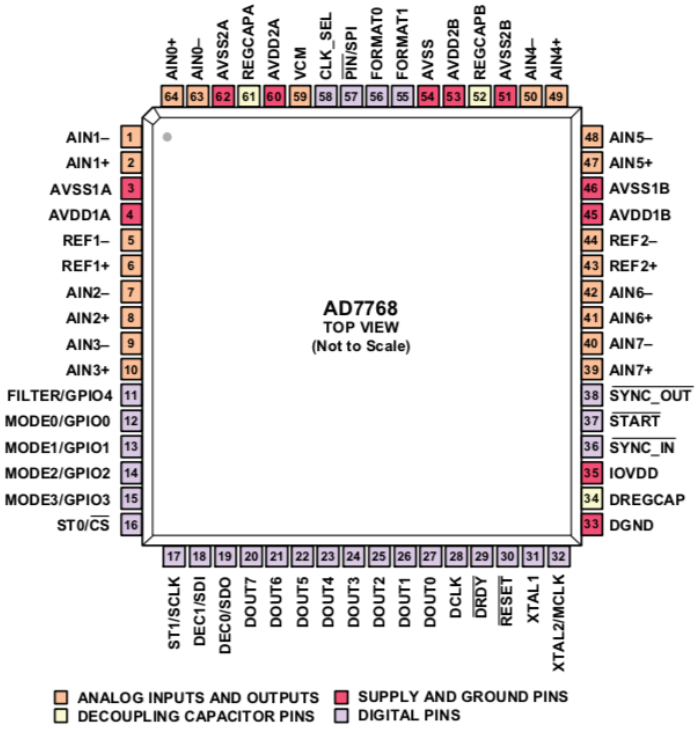



Figura 11. Disposición de pines del AD7768 [34]

Una valoración de los factores anteriores, junto con la posibilidad de integración de diferentes tipos de sistemas operativos, ha hecho que la placa seleccionada para este proyecto sea la ZedBoard. Esta cuenta con un Zynq ZC7Z020 basado en el chip Artix-7, que junto con los conectores FMC permiten realizar diferentes configuraciones y expansiones. Además, posee una memoria *flash* de 256 Mb y una RAM de 512 MB DDR3 junto con dos fuentes de reloj una a 100 MHz y otra a 33,333 MHz. Por último, dispone de un amplio número de periféricos o interfaces de comunicación como se ve en Figura 12 y Figura 13.

### 3.2.5. CONEXIONES EXTERNAS

Para poder interactuar con el sistema, dispone de una serie de conexiones externas. En primer lugar, se puede interactuar con este mediante el uso de la UART o mediante conexión Ethernet (Figura 14). En segundo lugar, dispone de una serie de conectores paralelos en la parte frontal que se utiliza para realizar mediciones y otros para la sincronización de los cuatro conversores ADC (Figura 15).

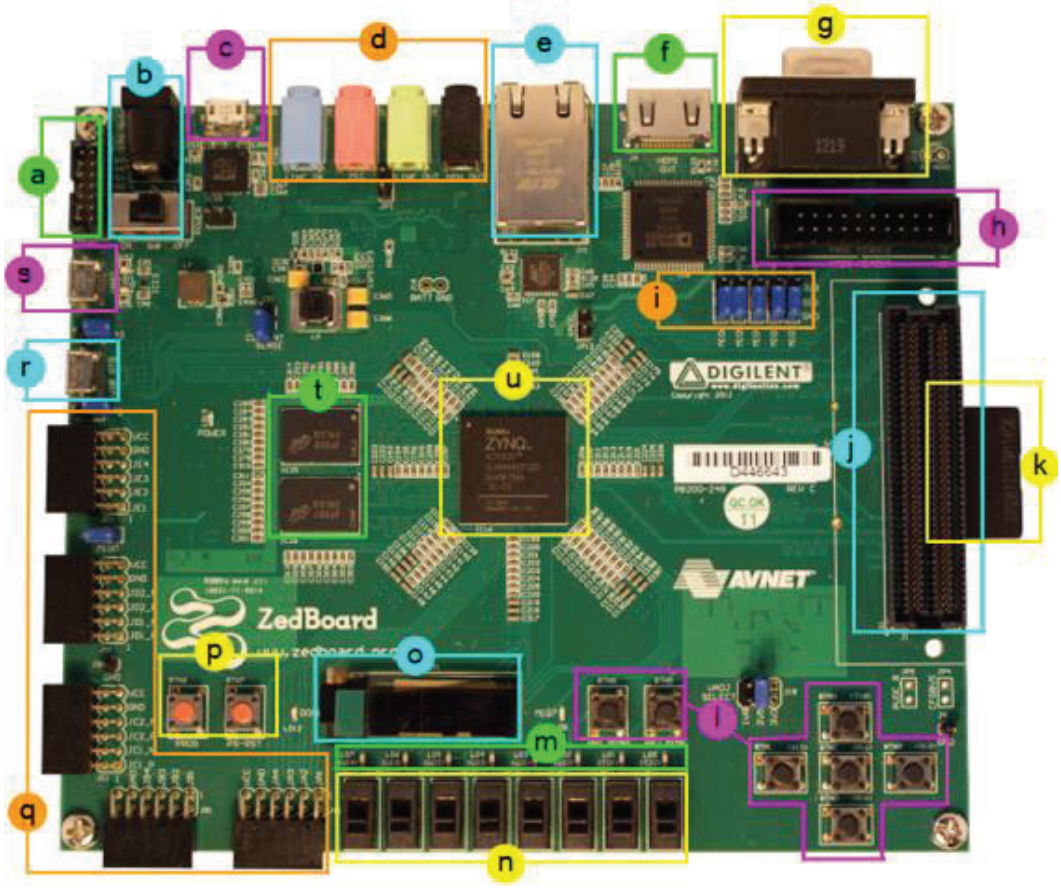



Figura 12. Partes de la ZedBoard

a Conector JTAG
b Conector de corriente e interruptor
c USB-JTAG (programación)
d Puertos de audio
e Gigabit Ethernet
f HDMI (salida)
g VGA
h Conector XADC
i Jumpers
j Conector FMC
k Ranura tarjeta SD
l Pulsadores
m LEDs
n Interruptores
o Pantalla OLED
p Botones de prog. y reinicio
q Conectores Pmod
r USB-OTG (periféricos)
s USB-UART
t Memoria DDR3
u Dispositivo Zynq

Figura 13. Etiquetas usadas en la Figura 12

Además, en la parte trasera existen dos puertos para interconectar la señal del RTC entre las dos placas. Por otro lado, en la parte frontal se dispone de una serie de diodos leds que indican el estado del equipo (fuentes en funcionamiento, si se ha producido un fallo en la alimentación, etc.), una toma de píxeles y el botón de apagado y encendido del equipo.


Figura 14. Conexiones traseras (Ethernet y UART)


Figura 15. Conexiones frontales (puertos paralelos y conectores de sincronización)

## 3.3. CONCLUSIÓN

En este capítulo se ha introducido el equipo *hardware* que se usará en este proyecto, aportando una descripción visual, así como las descripciones de los diferentes componentes. De esta manera se ha podido conocer de forma general qué elementos componen el sistema y sus características y cómo se interactúa con este mediante las conexiones existentes.

# CAPÍTULO 4. ARQUITECTURA HW/SW DEL SOC

## 4.1. INTRODUCCIÓN

En este capítulo se presenta la arquitectura desarrollada en el sistema, así como sus bloques, dando una visión general del equipo. Por otro lado, se presenta el flujo que seguirán los datos capturados por el equipo en las diferentes etapas del sistema.

## 4.2. ARQUITECTURA DEL SISTEMA

El sistema de procesamiento propuesto es el mostrado en la Figura 16. Como se puede apreciar, el sistema parte de un estado de configuración global y entra en un bucle de captura de datos, procesamiento, almacenamiento y envío de datos. Todos estos procesos se ejecutan de forma concurrente ya sea en el dominio *hardware* como en el dominio *software*. Para mantener la coherencia de los datos se establece como criterio básico la posibilidad de modificar la configuración del sistema cuando el *buffer* de salida esté vacío o los datos se vayan a desechar.

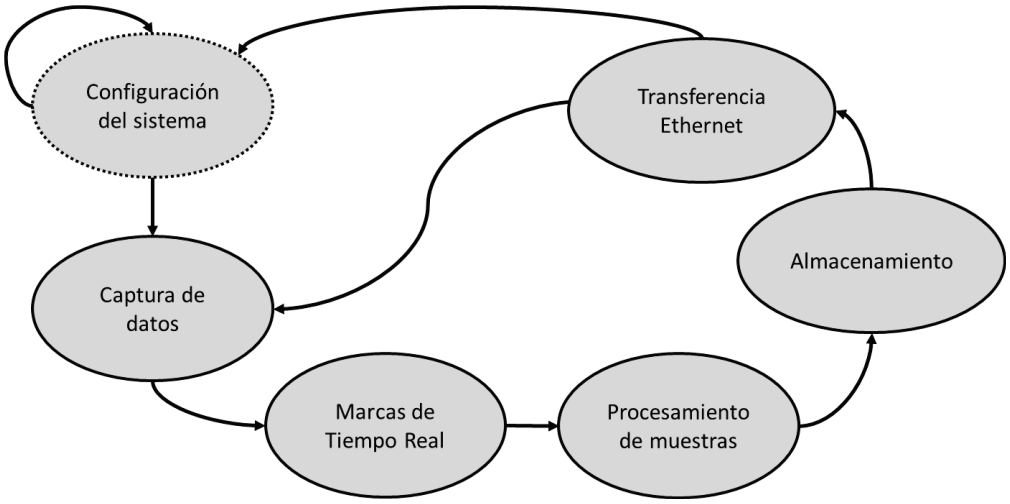



Figura 16. Principales procesos del sistema

La arquitectura del sistema SoC FPGA se muestra en la Figura 17, en la que se pueden identificar los bloques e interfaces principales del sistema. Se han mapeado los procesos indicados en la Figura 16 sobre la arquitectura de referencia, utilizando la arquitectura heterogénea del dispositivo Zynq [35]. Los procesos de configuración se han mapeado en un sistema *hardware*/*software* sobre la CPU0 y la CPU1 como bloques dedicados. Los procesos de procesamiento de datos y marcas de tiempo real se mapean en el dominio *hardware* (PL) y los procesos de almacenamiento en el dominio *software* controlado por la CPU0. Por último, los procesos de transferencia de datos se mapean en el dominio *software* sobre la CPU1. En un segundo nivel de granularidad de tareas, se incluyen los procesos de gestión de la interfaz de usuario (Tabla 1).

Tabla 1. Mapeado de tareas en los recursos de procesamiento heterogéneos

| | **Mapeado de tareas** | | |
|---|---|---|---|
| | **Dominio *Software*** | | **Dominio *Hardware*** |
| **Tareas** | **CPU0 (Linux)** | **CPU1 (*Baremetal*)** | **Lógica Programable PL** |
| **Configuración del sistema** | X | | X |
| **Captura de datos** | | X | X |
| **Procesamiento** | | X | X |
| **Marcas de Tiempo Real** | | | X |
| **Buffer de datos** | X | X | X |
| **Almacenamiento de datos** | X | | |
| **Transferencia Ethernet** | X | | |
| **Interfaz de Usuario** | X | | |

## 4.3. BLOQUES DE LA ARQUITECTURA

Como se muestra en la Figura 17, la arquitectura del sistema se organiza en dos partes principales: Sistema de procesamiento (PS) y Lógica programable (PL) FPGA. En la parte programable o PL se han incluido un conjunto de bloques que controlan la configuración de la fuente de alimentación, del sistema de adaptación y de la conversión

de los datos, así como señales auxiliares que permiten controlar ciertos dispositivos externos al equipo.

Los principales bloques del sistema son los siguientes:

- Bloque de control de ganancias
- Interfaz QSPI para el conversor ADC
- Control Modo PIN del conversor ADC
- Interfaz I2C para el acceso al RTC usando un PMOD
- Interfaz QSPI para la sincronización temporal entre placas
- GPIO para el control de fuente de alimentación y otros servicios
- Unidad de adaptación de datos, *buffer* de entrada, ordenamiento y normalización de datos
- *Timestamp d*e muestras
- Unidad hardware de procesamiento de datos parametrizable en función del número de muestras deseado.
- Arquitectura de buses AXI, integrados en el PL
- *Buffer* de salida de datos para la gestión de colas de muestras
- DMA para la transferencia de datos a memoria

Por otra parte, en el PS o sistema de procesamiento se utilizan dos núcleos ARM Cortex A9 en una configuración de multiprocesamiento asimétrico (AMP) para dar soporte al funcionamiento heterogéneo del sistema. La comunicación entre ambos subsistemas se realiza mediante mecanismos de memoria compartida y de interrupciones dedicadas, tal como se muestra en la Figura 17.

La CPU0 da soporte al sistema operativo Linux y su *stack* de procesamiento y comunicaciones. En dicha CPU se ejecutará la aplicación servidor, así como otros servicios del sistema (vigilancia de la alimentación, etc.).

La CPU1 utiliza una configuración *baremetal* para procesamiento en tiempo real que permite tener un control temporal muy definido de las tareas a realizar mediante la planificación controlada. Las funcionalidades asignadas son las siguientes:

- Configurar el sistema de adquisición, preparándolo para la captura de datos.
- Leer y actualizar la fecha del bloque de tiempo real y configurar los registros de tiempo real incluidos en el mismo bloque
- Enviar la configuración de control al bloque de adaptación de datos
- Configurar el bloque del cálculo del promedio
- Leer los datos recibidos y sus correspondientes marcas temporales desde el módulo de conversión de datos y enviarlos a la memoria compartida.

### 4.3.1. FLUJO DE DATOS

La CPU0 da soporte al usuario del sistema mediante la instalación de un sistema operativo Linux empotrado para ARM. Dispone de un *stack* TCP/IP completo que da soporte a las aplicaciones de configuración y transferencia de datos. Con objeto de configurar el sistema, el usuario accede mediante una interfaz de línea de comandos (CLI – *Command Line Interface*) en una arquitectura cliente-servidor que facilita la introducción y modificación de parámetros claves para el funcionamiento del equipo. Igualmente se dispone de un servicio de almacenamiento de datos mediante NFS (*Network File System*) [36] y de acceso al sistema en modo línea de comandos a través de una aplicación SSH.

La CPU0 leerá los datos desde la memoria compartida OCM (*On-Chip Memory*) y escribirá los ficheros necesarios en el sistema de ficheros de Linux para su almacenamiento remoto. Para ello se encarga de gestionar la creación de los ficheros correspondiente usando los *drivers* disponibles en el sistema operativo empotrado mencionado. En la Figura 18 se muestra el flujo de datos de las muestras capturadas a través del sistema de procesamiento.

Igualmente, la CPU0 se encarga de ejecutar el servidor que facilita la configuración y el control de la plataforma desde un cliente remoto. El usuario puede definir los parámetros de configuración por SPI, la ganancia, el número de muestras, etc. Los datos obtenidos por el servidor desde el cliente se envían al *hardware* mediante los correspondientes *drivers* diseñados.

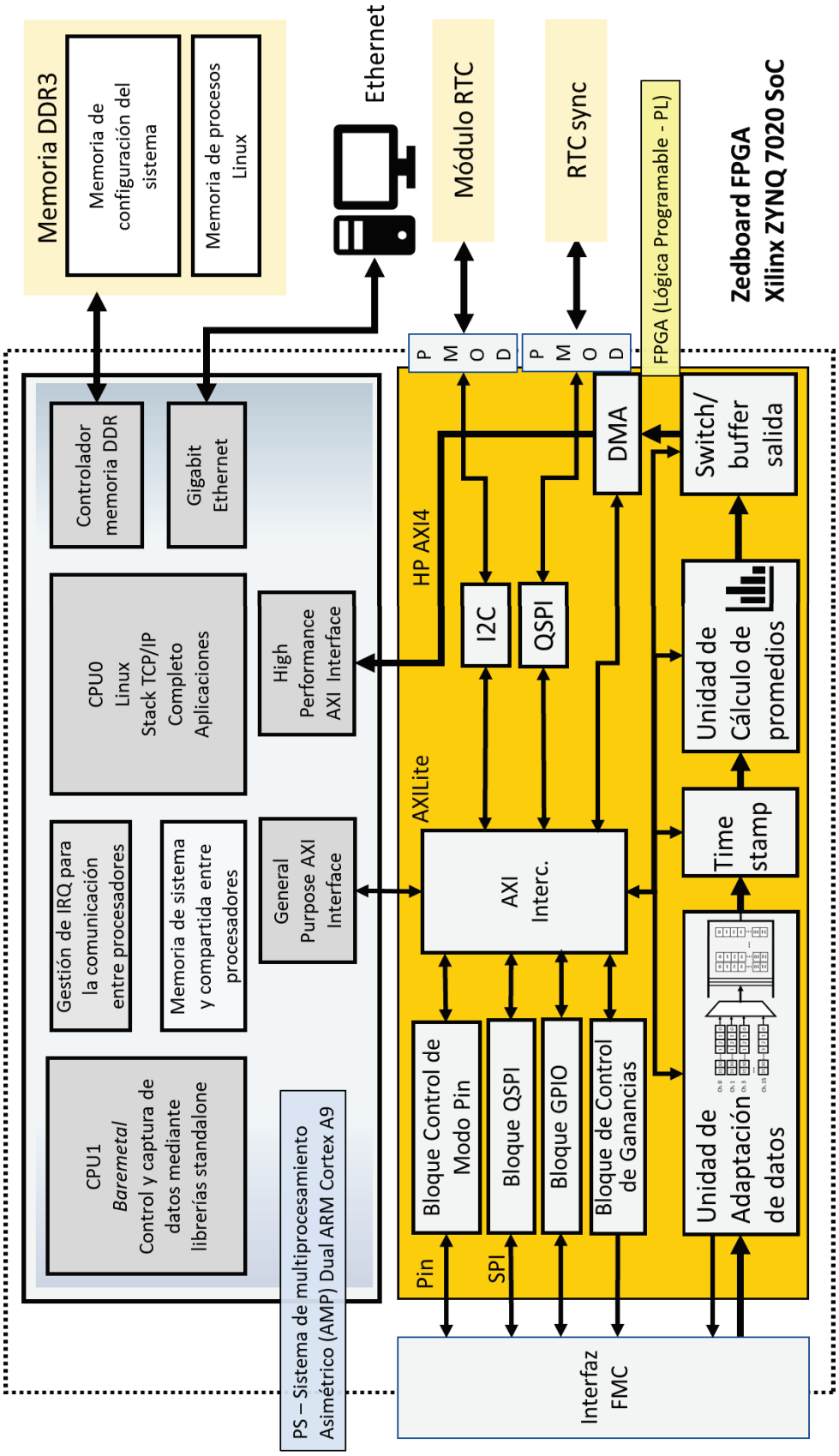



Figura 17. Diagrama general de la arquitectura del sistema SoC FPGA

Por razones de seguridad, los parámetros no podrán modificarse mediante el acceso directo al equipo, sino que los parámetros de configuración se enviarán desde el cliente al servidor que se ejecuta en el procesador CPU0. EL servidor hace chequeos de rangos de valores permitidos y del estado del sistema. Con esto se consigue un acceso controlado a los bloques del sistema. La comunicación entre el PS y el PL se realiza a través de la memoria compartida, donde los datos son transmitidos a través de un DMA (*Direct Memory Access*) [37], controlado por la CPU1. Por lo tanto, la CPU1 solo realiza escrituras a la memoria compartida, mientras que la CPU0 solo realiza lecturas a la memoria compartida. Esta comunicación queda restringida a un área específica de memoria del sistema que presenta una alta transferencia de datos (128 bits por acceso).

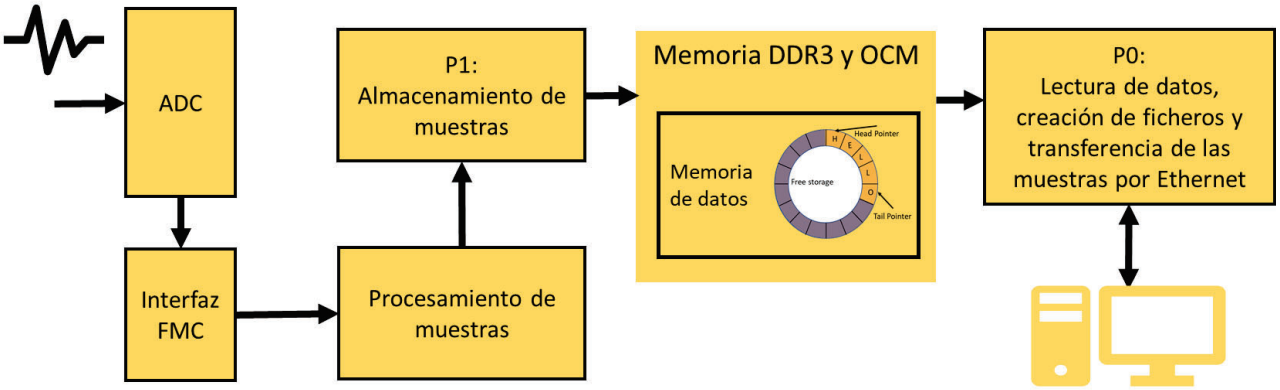



Figura 18. Flujo de datos de las muestras capturadas

## 4.4. CONCLUSIONES

En este capítulo se ha presentado la arquitectura del sistema determinando lo que cada CPU tendrá la capacidad de hacer y definiendo claramente los dominios de cada una presentando los bloques de la arquitectura. Por último, se definen las etapas por las que pasarán los datos en cada una de las CPU según el estado en el que se encuentren.

# CAPÍTULO 5. SISTEMA OPERATIVO EMPOTRADO

## 5.1. INTRODUCCIÓN

En este capítulo se presenta la arquitectura *software* desarrollada en este trabajo. Es importante conocer las características de las diferentes opciones disponibles, así como su funcionamiento para adaptar de la mejor forma posible el *software* al *hardware* del que se dispone.

## 5.2. ARQUITECTURA *SOFTWARE*

La flexibilidad de la plataforma Zynq, permite realizar mediante las diferentes herramientas de diseño, como Vitis de Xilinx, la configuración adecuada al proyecto. Sobre el diseño realizado se basará el *software* implementado, el cual se considera como una pila que se construye siguiendo las especificaciones establecidas como puede verse en Figura 19.

La capa de más bajo nivel se conoce como BSP *(Board Support Package)*. Contiene los controladores y funciones de bajo nivel para que la siguiente capa, el sistema operativo, pueda comunicarse con el *hardware.*

El uso de sistemas operativos empotrados es una práctica extendida en la industria, ya que puede reducir los tiempos de desarrollo para la llegada al mercado [38]. Debido a que existen numerosas alternativas, que se tratarán en capítulos posteriores, reduce la complejidad para migrar los sistemas a una nueva arquitectura cuando la situación lo precise o se requieran mejoras en las especificaciones.

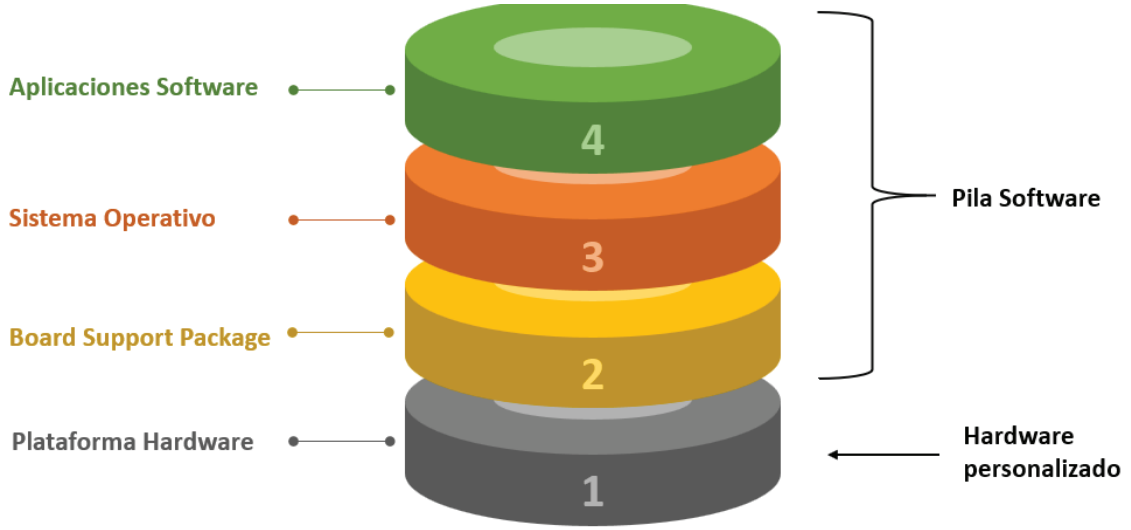



Figura 19. Capas de un diseño Zynq

Estos sistemas, además, incorporan el soporte de las empresas de origen, foros o comunidad, lo que hace que se reduzca el tiempo en la creación de nuevo código, permitiendo reducir esfuerzos de desarrollo [39].

Entre las distintas opciones disponibles para implementar el *stack software* podemos citar las siguientes: Linux, Android, freeRTOS o solución *baremetal.* Cada uno posee sus características únicas y debe estudiarse cuando es recomendable implementar uno u otro, existiendo la posibilidad en algunos casos de realizar una combinación de ambos, teniendo en cuenta también la plataforma donde se pretenden implementar y las características de esta.

En la familia Zynq, se dispone de dos procesadores ARM Cortex A9, por lo que es posible usar los dos procesadores para un mismo sistema operativo, o que cada uno tenga su propio sistema operativo. Esto se conoce como un sistema SMP (*Symmetric Multi-Processing)* y AMP (*Asymmetric Multi-Processing*), respectivamente.

Un sistema SMP utiliza ambas CPU, de la misma arquitectura, para un mismo propósito bajo un sistema operativo común. El sistema AMP utiliza varias CPU que pueden ser de arquitecturas iguales o diferentes, donde una puede ejecutar un sistema operativo, Linux por ejemplo, y la otra CPU uno totalmente diferente [40]. Por ejemplo, se puede realizar una configuración Linux-*freeRTOS.* En ambas configuraciones, SMP y AMP, la comunicación se realiza mediante una memoria compartida que permite que se coordinen

para la ejecución de tareas [41]. Se puede observar en la Figura 20 estas dos configuraciones.

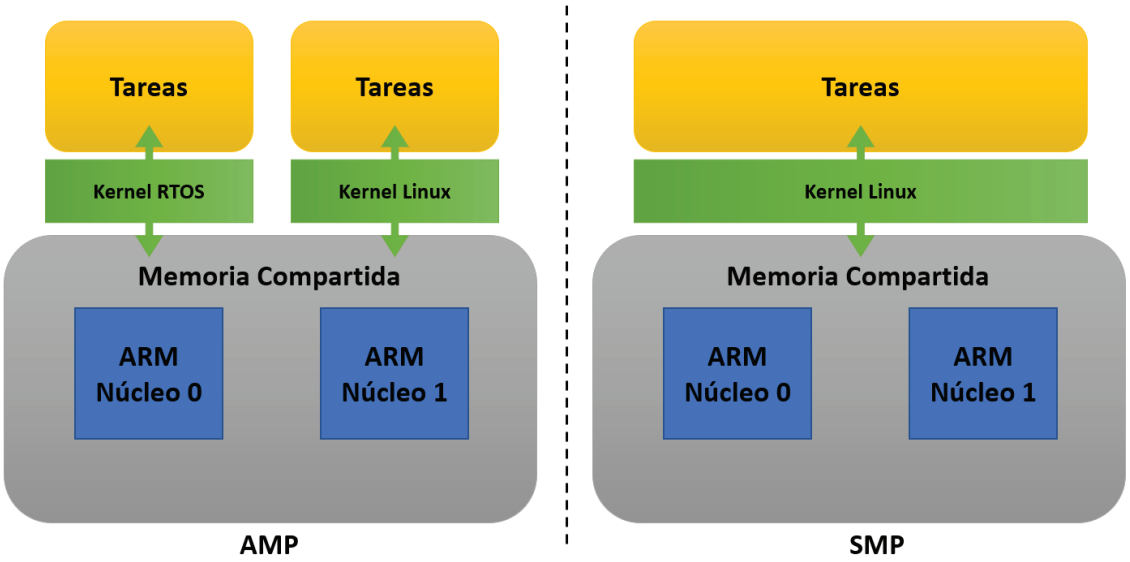



Figura 20. Estructura AMP y SMP [4]

## 5.2.1. TIPOS DE CONFIGURACIONES AMP

Como se ha indicado anteriormente, existen diferentes modos de configuraciones permitida en el modelo de multiprocesamiento asimétrico. El principal sería la diferencia entre el uso de AMP sin supervisión o con supervisión. El primer modo, utiliza un procesador para realizas las tareas de coordinación y gestión del núcleo independiente. El segundo modo, crea una zona "segura" que ofrece diferentes herramientas para la ejecución independiente de un sistema en cada núcleo.

A continuación, se presentan distintas configuraciones AMP que se han barajado como solución a adoptar para este proyecto.

### *5.2.1.1. Baremetal – Baremetal*

Esta solución consiste en ejecutar una aplicación específica en cada CPU. En esta configuración, que se puede ver en Figura 21, el FSBL (*First Stage Boot Loader*) [42] es el encargado de cargar ambas aplicaciones y ejecutar la CPU0 mientras que la CPU1 es inicializada por la aplicación de la CPU0.

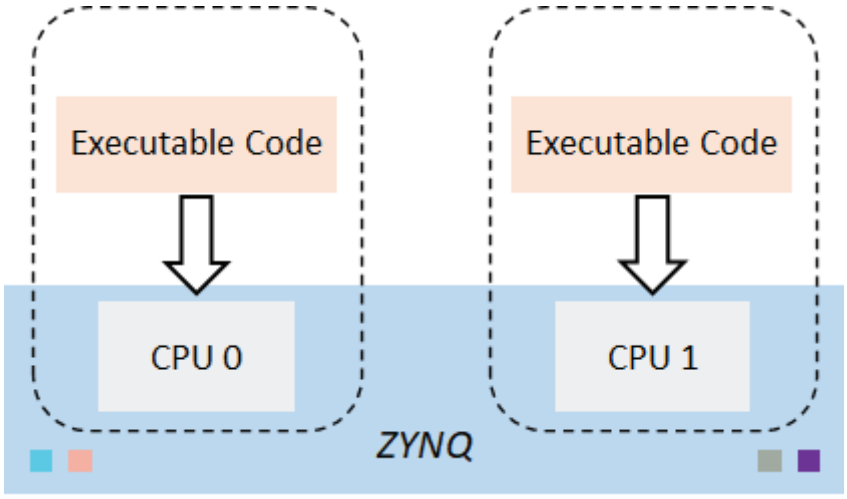



Figura 21. Configuración baremetal-baremetal, adaptada de [43]

*5.2.1.2. Baremetal – Linux*

Otra configuración, como la mostrada en la Figura 22, implementa un sistema Linux en la CPU0 y un sistema *baremetal* en la CPU1 con una aplicación específica. La aplicación CPU1 es cargada por el FSBL y Linux se encarga de iniciar su ejecución. La comunicación entre ambas se realiza mediante la OCM (*On-Chip Memory*) desactivando la caché para mejorar el determinismo.

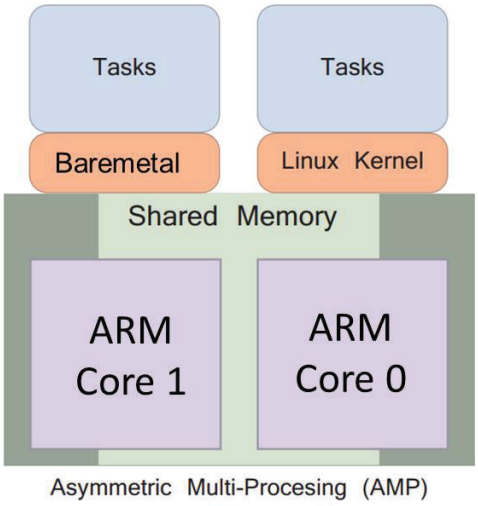



Figura 22. Sistema de Procesamiento Asimétrico

*5.2.1.3. Linux – FreeRTOS*

En esta configuración en núcleo principal CPU0, ejecuta el sistema Linux y la CPU1 usa un RTOS (*Real-Time Operating System*) [44], en este caso, FreeRTOS [45]. Esto permite a Linux acceder en modo lectura o escritura a las direcciones de memoria utilizadas por el

RTOS, como se ve en la Figura 23. También es posible el uso de otros sistemas operativos en tiempo real, además de FreeRTOS.

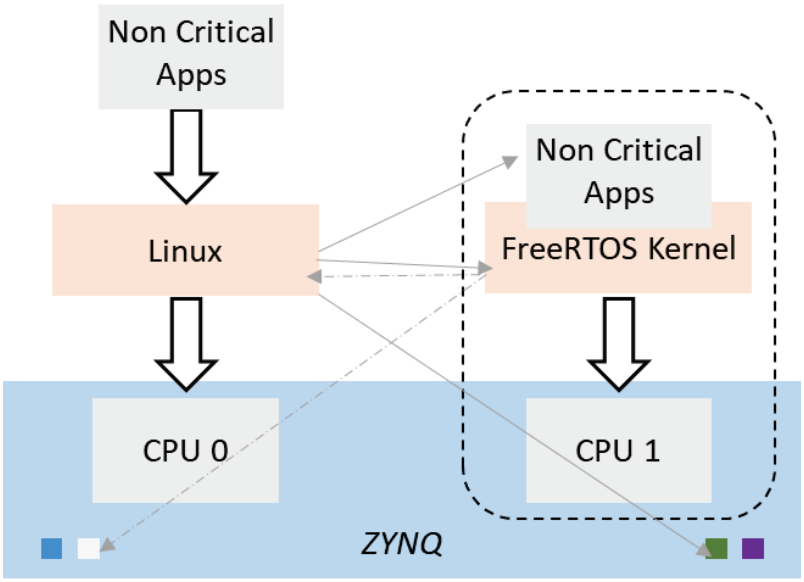



Figura 23. Configuración Linux-FreeRTOS, adaptada de [43]

*5.2.1.4. Xenomai*

Xenomai [46] es una solución de tiempo real de doble núcleo y de código abierto para Linux, que utiliza un *kernel* para aplicaciones en tiempo real y un *kernel* de Linux para aplicaciones que no sean críticas, buscando una solución a sistemas que requieran estrictos requisitos de latencia (Figura 24).

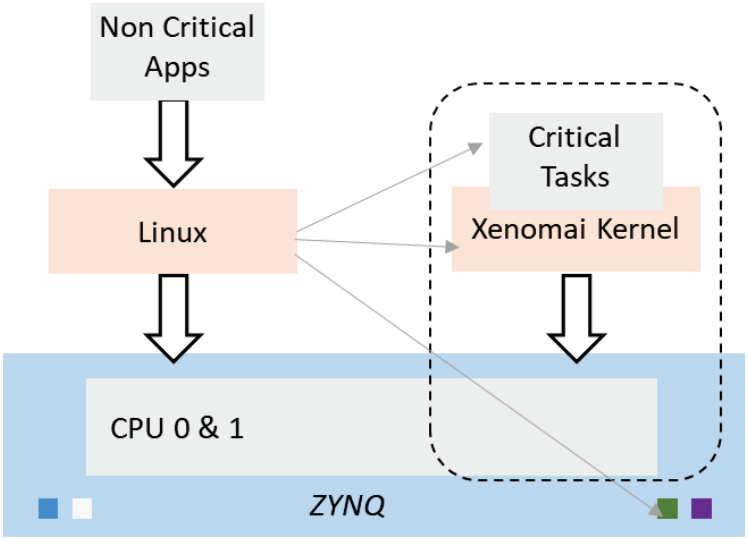



Figura 24. Configuración Xenomai, adaptada de [43]

#### *5.2.1.5. Linux – Zephyr OS*

Zephyr OS [47] es un RTOS que puede combinarse junto a Linux [48]. Este sistema de código abierto, creado por la Linux Foundation, ocupa poco espacio y está diseñado pensando en la seguridad y la protección de entornos limitados.

Se basa en un *kernel* de tamaño reducido ideado para su uso en sistemas embebidos y con recursos limitados. Está disponible para numerosas arquitecturas ampliamente usadas como ARM o Intel y otras de reciente crecimiento como RISC-V [49].

Este sistema operativo usado en sistemas IoT, *wearables*, sensores, etc. destaca por el amplio y creciente número de funciones, entre las que se encuentran: protección de memoria, compatibilidad Bluetooth de bajo consumo 5.0, definición de recursos en tiempo de compilación, desarrollo nativo en Linux, MacOS y Windows, etc.

## 5.3. CARACTERÍSTICAS DEL SISTEMA OPERATIVO

Para elegir el sistema operativo adecuado se deben conocer los costes, requisitos y vida útil del equipo donde se va a implementar la solución, encontrando un equilibrio entre los costes de la lista de materiales [50] y el coste del mantenimiento del *software.* Por norma general, los sistemas que usan Linux poseen la ventaja de poder reutilizar código abierto y encontrar con facilidad desarrolladores que puedan escribir código para la plataforma.

Por contrapartida, estos sistemas poseen requisitos *hardware* más complejos que aquellos sistemas que están basados en microcontroladores, de menor coste.

El siguiente paso que se debe estudiar son las características físicas del dispositivo *hardware* escogido. Los principales aspectos a tener en cuenta son: características de los periféricos, almacenamiento *flash,* la velocidad de reloj de la CPU y la memoria RAM disponible.

Todos estos aspectos pueden hacer que la elección del sistema elegido sea limitada, por ejemplo, para ejecutar una distribución de Linux empotrado es necesario que el procesador tenga una unidad de gestión de memoria (MMU).

En la elección también se estudia el rendimiento, dependiendo de las características que se buscan y teniendo en cuenta si se busca una respuesta en tiempo real. En este aspecto, no existe una decisión clara, pues en el ámbito de los microcontroladores se asegura que en un sistema Linux no se puede obtener un rendimiento en tiempo real. Por otro lado, en el ámbito Linux existe un parche que proporciona un comportamiento de tiempo real. En cualquier caso, cada sistema es individual y requiere un tipo de estudio único, evaluando lo determinista que debe ser el sistema para elegir la opción más adecuada que se adapte a nuestra aplicación.

La disponibilidad e integración de bibliotecas *software* puede ser un factor decisivo si se busca un desarrollo rápido o lo más a medida posible. En el ámbito *baremetal* cada desarrollo suele utilizar *software* desarrollado desde cero o integrando algunas bibliotecas de terceros, pero la oferta es limitada. Un sistema operativo de tiempo real (RTOS) contiene un conjunto de bibliotecas que la mayor parte de las veces ya están integradas en este, como herramientas de gestión de memoria, sistemas de archivos, etc. Aunque hay veces que se deben desarrollar módulos específicos para este. Por último, cuando se visualiza la oferta de Linux, estas con casi ilimitadas en cuanto a bibliotecas y opciones *software* se refiere integradas en el sistema. El usuario en este aspecto posee un sistema operativo completo, con todo lo que eso conlleva incluido los problemas de seguridad.

Por el contrario, los problemas pueden ser comunes. Por ejemplo, un problema de seguridad puede ser encontrar una brecha en una biblioteca estándar del *kernel* de Linux, lo que hará que hasta que esta no sea parcheada todos los sistemas Linux sean vulnerables, algo que en sistemas *baremetal* o RTOS es menos probable que suceda.

## 5.4. ELECCIÓN DEL SISTEMA OPERATIVO: *BAREMETAL,* RTOS O LINUX

Conociendo las características de los sistemas operativos queda plantear cuándo y dónde tiene sentido emplear cada uno de manera general, pues como se ha comentado anteriormente, cada sistema es individual y único.

### 5.4.1. BAREMETAL

Usar un sistema *baremetal* es casi una elección obligatoria cuando se usan arquitecturas de 8 o 16 bits. Esto se debe a que muchos sistemas no están adaptados a arquitecturas reducidas porque estos cuentan con un procesador y recursos limitados, y añadir un sistema operativo puede acaparar estos haciendo que el sistema sea ineficiente. También es una buena elección cuando el número de pines del microcontrolador es bajo y la memoria *flash* y SRAM disponibles es limitada, debido a que cada byte y ciclo de reloj es importante para que un sistema sea funcional.

Por último, puede emplearse cuando se necesita una aplicación "simple" que no requiere conectividad o procesamiento de alto rendimiento. Es decir, que el código no requiere de un apoyo de la infraestructura donde se necesitan conectar dispositivos a la nube, ejecutar algoritmos complejos, etc. Un sistema operativo facilita estas tareas y gestión de recursos.

### 5.4.2. RTOS

Es recomendable utilizar este sistema cuando se dispone de al menos una velocidad mínima de reloj de 40 MHz con al menos 64 kilobytes de memoria *flash* y 8 kilobytes de memoria RAM, ya que con menos recursos pueden presentarse problemas en cuanto a la demanda del RTOS, ya que para su *stack* usa entre 512-1024 bytes de SRAM. Cuando se aplica un RTOS, debe aportar valor a la aplicación y al ciclo de vida global del desarrollo de *software* de lo contrario supondrá un obstáculo.


Figura 25. FreeRTOS es uno de los RTOS más usados

Algunas de las aplicaciones donde inmediatamente se piensa en el uso de un RTOS, son por ejemplo en productos de IoT *(Internet of Things*) [51], ya que este proporciona las

herramientas y mecanismos necesarios que permiten de una manera fácil gestionar los recursos de bajo nivel y dividir la aplicación en programas semindependientes.

### 5.4.3. LINUX

El uso de Linux en sistemas empotrados se ha convertido con los años en una opción muy popular, en parte, gracias a todas las funcionalidades que conlleva (bibliotecas, funciones, herramientas, etc.). Este permite aprovechar aspectos como la multitarea e incluso hacer uso del parche de tiempo real.

Los costes de *hardware* para ejecutar Linux han ido disminuyendo con el paso de los años reduciéndose drásticamente, convirtiéndolo en una solución interesante para aplicaciones específicas. Aun así, para adoptar esta solución se debe tener en cuenta que el producto sea capaz de soportar los costes del *hardware.*

Teniendo esto presente, Linux ofrece potentes abstracciones, servicios y bibliotecas que simplifican el desarrollo de la ingeniería del producto. Ello hace que se simplifique con el uso de este, permitiendo una forma más sencilla de interactuar con el *hardware.* Además, permite usar lenguajes de programación más modernos como Python, personalizar el *kernel* si es necesario, flexibilidad, etc.

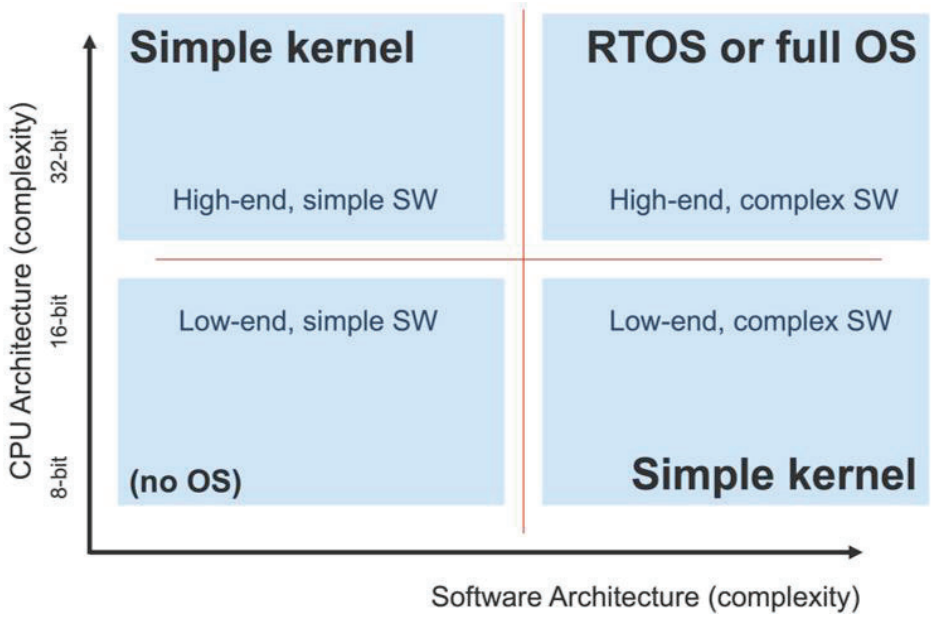



Figura 26. Dimensiones para la elección de un SO [52]

### 5.4.4. SOLUCIÓN ADOPTADA

La solución adoptada consiste en una arquitectura de multiprocesamiento heterogéneo asimétrico que permite la especialización de tareas entre los dos procesadores que componen el sistema de procesamiento (PS) del dispositivo Zynq.

De las soluciones mostradas anteriormente, se elige el uso del binomio Linux-*baremetal* (Figura 22), donde la CPU0 (Core 0) incluye un sistema operativo Linux adaptado a la arquitectura Xilinx Zynq, configurado específicamente para este proyecto, y la CPU1 (Core 1) una aplicación *baremetal*/*standalone* diseñada para la aplicación de captura de datos.

### 5.4.5. OPENAMP

Para la sincronización de los procesos que ejecutan ambos procesadores se utiliza el entorno no supervisado OpenAMP [53]. Se trata de un sistema de código abierto para la sincronización de procesos en un sistema heterogéneo que emplea un mecanismo de paso de mensajes usando recursos compartidos entre ambas CPUs. En este caso los procesadores funcionan de forma independiente, cada uno bajo su propio *stack software*, sin un *software* superior que coordine el funcionamiento. Ello requiere un ajuste detallado en cuanto al acceso a los recursos del sistema.

La arquitectura usada se muestra en la Figura 27, tal como se recoge en la documentación de Xilinx [54]. Como se puede apreciar, la sincronización entre ambos procesadores se realiza a nivel de *kernel*, mediante un sistema de paso de mensajes y la utilización de memoria compartida. Estos aspectos se configuran en el correspondiente *Device Tree*, que es necesario ajustar para cada caso y aplicación. La utilización de OpenAMP debe configurarse en el *kernel* de Linux, e incluir el correspondiente soporte de utilidades en el *rootfs* de Linux.

El funcionamiento simplificado del sistema se muestra en la Figura 28. Como se puede apreciar, la CPU0 (que en este caso denominamos maestra) arranca el sistema y mantiene en *reset* a la CPU1. En este momento la CPU0 está ejecutando Linux, con las distintas tareas de usuario y de sistema. Toda la memoria disponible puede ser utilizada por la CPU0.

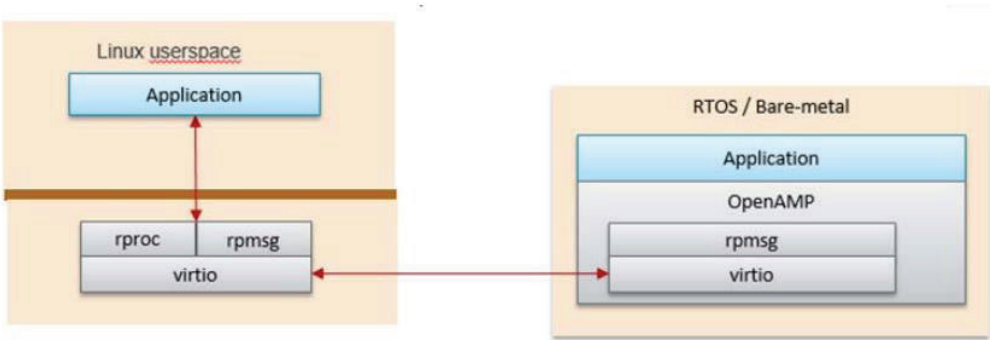



Figura 27. Arquitectura OpenAMP utilizada

Seguidamente, la CPU0 prepara la carga del *firmware* necesario usando el correspondiente módulo de *kernel*, mientras sigue manteniendo a la CPU1 en estado de *reset*. Cuando se termina de preparar el sistema remoto, se arranca la CPU1, que ya dispone de su espacio propio de memoria, y se comunican usando los mecanismos de paso de mensajes de OpenAMP. Igualmente se dispone del acceso a la memoria en chip compartida (OCM) que se utiliza para el intercambio de datos y la señalización mediante un mecanismo de sincronización de captura y de escritura en ficheros.

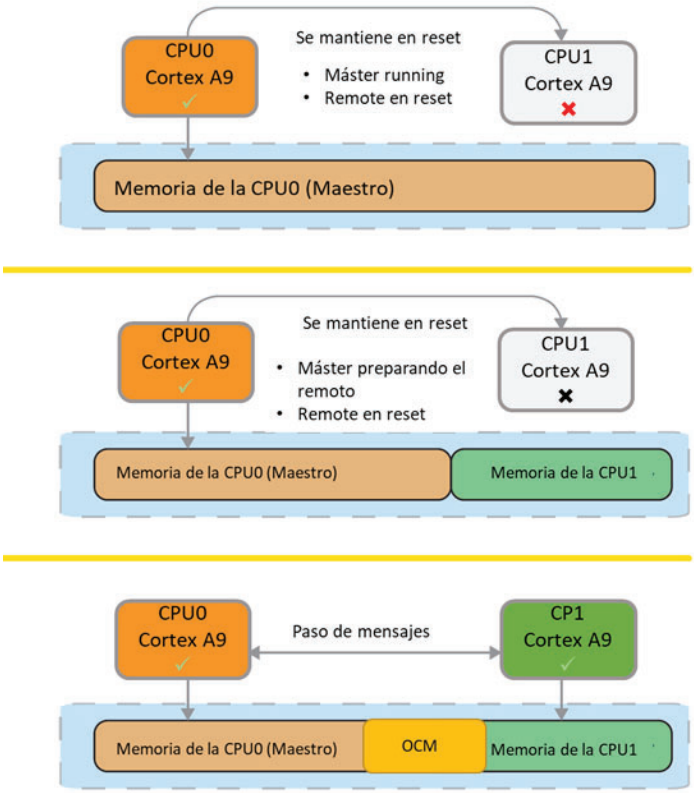



Figura 28. Funcionamiento simplificado del sistema OpenAMP no supervisado

### 5.4.6. SISTEMA CON SOPORTE *BAREMETAL (STANDALONE)*

Un sistema con soporte *baremetal*, también conocido como sistema *standalone*, dispone de un conjunto de librerías que tiene como objetivo proporcionar una API *software* que el sistema puede usar para acceder a funciones específicas del procesador. Con respecto a la plataforma Zynq específicamente, Xilinx proporciona una plataforma de sistema operativo independiente que incluye funciones para configurar cachés, configurar interrupciones y excepciones y otras funciones relacionadas con el hardware.

La plataforma independiente se encuentra directamente debajo de la capa de librerías del sistema y se utiliza siempre que una aplicación requiere acceder directamente a las funciones del sistema. El sistema *standalone* permite un control cercano sobre la ejecución de código, pero está limitado en términos de funcionalidad. De igual forma el número de tareas que lleva a cabo un sistema operativo independiente está limitado. Como ventajas hay que indicar que el diseñador tiene un control detallado del tiempo de ejecución de las tareas, lo que le permite realizar un planificador de tareas a medida. En este equipo se utiliza para la captura de datos desde el ADC, pudiendo controlar los tiempos de entrega al *buffer* de datos disponible en la memoria compartida.

La arquitectura de un sistema *standalone* se muestra en la Figura 29.

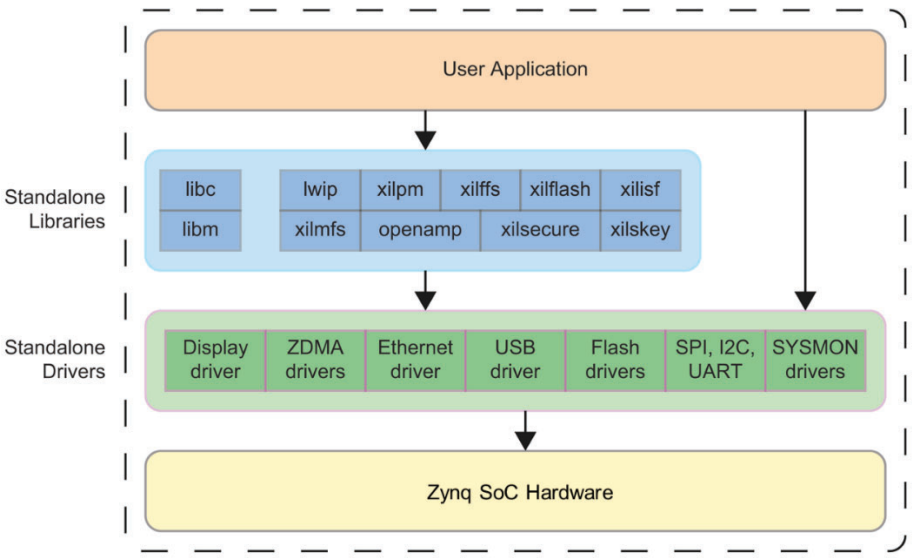



Figura 29. Arquitectura librerías *standalone*

Las funciones para acceder a los drivers disponibles se incluyen en la aplicación mediante los archivos de cabeceras (.h) en C/C++. La funcionalidad se incluye como librerías que es necesario *lincar* con la aplicación. El *linker* se define a medida en función de la arquitectura y mapa de memoria disponible en el BSP, que se obtiene al exportar el hardware diseñado.

### 5.4.7. *KERNEL* DE LINUX

De los sistemas indicados anteriormente, Linux juega un papel importante en este proyecto. Se pueden identificar tres grandes bloques:

- *Kernel*. Es el núcleo del sistema. Se encarga de mantener las abstracciones del sistema como la memoria virtual y los procesos [55].
- Bibliotecas del sistema. Recogen las funciones estándar para que las aplicaciones puedan interactuar con el *kernel*.
- Utilidades del sistema. Son programas que realizan tareas de gestión individuales y especializadas. Puede ser utilizadas una sola vez o estar ejecutándose permanentemente donde se conocen como demonios.

En la Figura 30 se puede apreciar la arquitectura de alto nivel de este.

El *kernel* puede cargar diferentes módulos bajo demanda, ejecutándose en modo privilegiado, lo que permite que puedan tener acceso a todas las capacidades del *hardware* utilizado. El soporte de módulos tiene tres componentes principales:

- La gestión de módulos. Permite que los módulos se carguen en la memoria y que puedan comunicarse con el resto del *kernel*.
- El registro de controladores. Permite a los módulos indicar al resto del *kernel* que un nuevo controlador está disponible.
- Mecanismo de resolución de conflictos. Permite que diferentes controladores de dispositivos reserven recursos de *hardware* y los protejan de un uso accidental por parte de otro controlador.

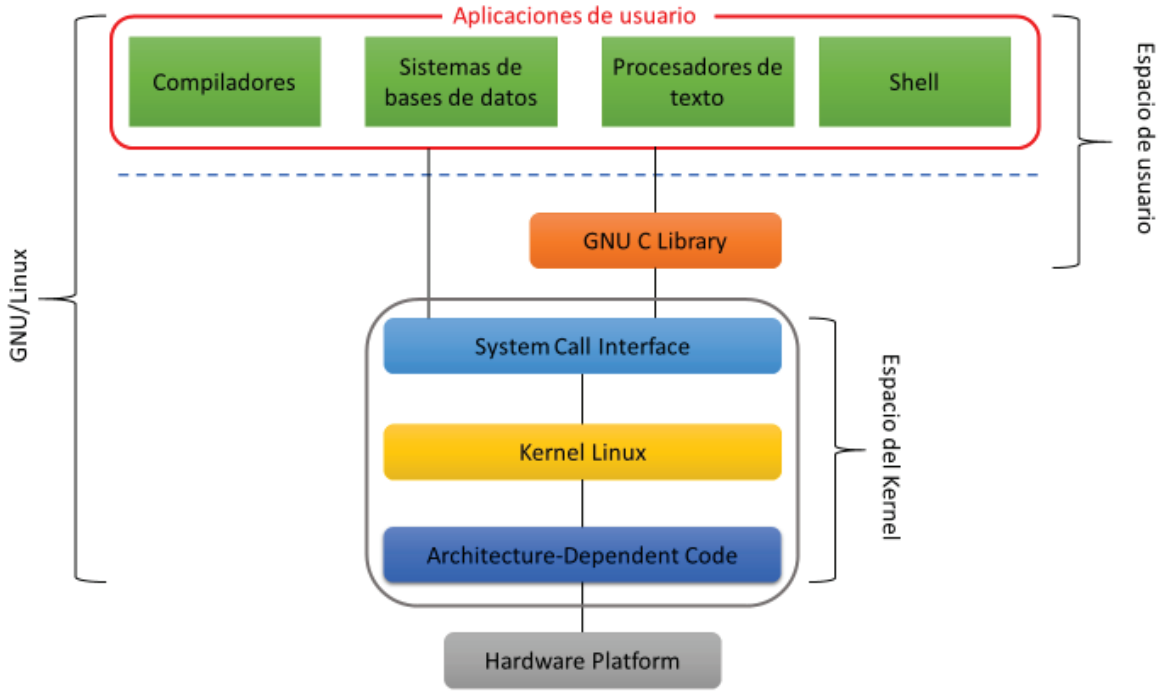



Figura 30. Modelo de arquitectura GNU/Linux de alto nivel

Por último, el *kernel* de Linux puede separarse en varias capas diferenciadas:

- Gestor de memoria. Maneja la memoria del sistema, tanto la memoria física (memoria del sistema), como la memoria virtual.
- Gestor de ficheros virtuales. Permite unificar los diferentes tipos de estructuras de almacenamiento de datos que se pueden establecer en el sistema. Además, crea una caché que permite reducir el tiempo de acceso a los ficheros del sistema y otra para los ficheros de mayor uso [56].
- BSD Socket Library. Librería que ayuda en la comunicación entre procesos TCP/IP además de proporcionar direccionamiento e información de red [57], [58].
- Gestor de procesos. Maneja los procesos del sistema, gestionando la memoria o recursos usados.
- Drivers de dispositivos. Manejan de forma abstracta los periféricos del sistema. Su comportamiento modular hace que sean de fácil instalación cuando son requeridos. No requieren estar incluidos en el *kernel* para su instalación, lo que permite que existan cientos de ellos [59].

## 5.5. CONCLUSIÓN

En este se presenta el *stack software* necesario para dar soporte a la arquitectura *software* del sistema. Se presenta las configuraciones estudiadas y se elige aquella que más se adapta al proyecto. También se ha presentado la solución OpenAMP, explicando su funcionamiento y las ventajas que aporta. Por otro lado, también se presenta brevemente el núcleo interno del sistema Linux, necesario para poder explicar la arquitectura por capas de la aplicación final.

# CAPÍTULO 6. DESARROLLO DE LA PLATAFORMA *SOFTWARE*

## 6.1. INTRODUCCIÓN

Establecida la plataforma a utilizar, el modelo *software,* los lenguajes de programación y el sistema operativo necesario para realizar la implementación del prototipo del sistema que se pretende. Se describen a continuación las diferentes opciones que se han tenido en cuenta, a la hora de desarrollar la aplicación, posteriormente, se explican las herramientas que proporciona Xilinx para su instalación en la placa y así como el flujo seguido.

## 6.2. SISTEMA OPERATIVO

En este caso, se dispone de una arquitectura de 32 bits, donde un sistema operativo completo da una mayor flexibilidad a la hora de desarrollar las aplicaciones necesarias del sistema. Como se ha indicado anteriormente, en este trabajo se trabaja con un binomio Linux-*baremetal.* Cada uno aporta un valor añadido crucial para la aplicación específica que se requiere, de modo que ambos puedan funcionar de forma cooperativa apoyándose en las librerías OpenAMP, tal como se indicó anteriormente.

Para constituir un sistema operativo basado en Linux, se deben configurar sus componentes: *bootloader, kernel,* sistema de ficheros y aplicaciones de usuarios. Entre las distribuciones Linux disponibles para la placa seleccionada se encuentran Petalinux [60], [61], Xillinux [62] y Koheron OS [63], entre otros.

Petalinux es un entorno de desarrollo creada por el fabricante Xilinx que ofrece las herramientas necesarias para crear soluciones de diseño *hardware* en Linux Embebido sobre placas de su propiedad. Está basado en Yocto [64], proyecto de colaboración de

código abierto que ayuda a crear sistemas personalizados basados en Linux, independientemente de la arquitectura del *hardware.* Además, ofrece un conjunto de herramientas que los desarrolladores pueden compartir para crear nuevas tecnologías, configuraciones y generar mejores prácticas que pueden utilizarse para crear imágenes de Linux a medida para sistemas empotrados y de IoT, o en general donde sea necesario un sistema operativo Linux personalizado. Entre las herramientas principales con las que cuenta esta distribución, se encuentra: generación BSP personalizada, configuración de Linux y desarrollo de *software*. Por tanto, Petalinux es una distribución de referencia para el desarrollo, para *hardware* basado en Xilinx.

Por otro lado, Xillinux es una distribución de la compañía Xillybus basada en Ubuntu 16.04 LTS (*Long Time Support)* para ARM, que incluye un kit de *sotfware* destinado a desarrollo facilitando la integración entre el *host* Linux (PS) y parte de la FPGA (PL). Permite ejecutar un escritorio gráfico completo, pudiendo conectar un monitor para su visualización, además de un teclado y ratón para su manejo.

Koheron OS es una distribución Linux customizada basada en Ubuntu 16.04 para los dispositivos de la familia Zynq. La ventaja de Koheron es la rapidez con la que se puede tener un sistema operativo desplegado. Además, facilitan una serie de herramientas *software* que permiten acceder al *hardware* del que disponen las placas, como los leds, interruptores, etc.

Para este trabajo, se ha optado por la distribución Petalinux, que permite construir un proyecto a medida ya sea mediante un BSP o un XSA (*Xilinx Support Archive*). Esto hace que la configuración se construya completamente a medida y se puedan introducir todos aquellos cambios que sean necesarios para conseguir la mayor eficiencia de los recursos *hardware.*

## 6.3. XILINX PETALINUX

Petalinux, sistema operativo de Xilinx, ofrece código y diversos drivers de apoyo al desarrollo [65], [66]. Es un entorno de desarrollo utilizado para construir y desplegar soluciones empotradas Linux, ya que posee: *toolchains, kernel*¸ compilador, librerías,

aplicaciones de usuario, etc. Aunque la creación y configuración del sistema, es un proceso que requiere el estudio en profundidad de la documentación.

### 6.3.1. REQUISITOS PETALINUX

Los requisitos de Petalinux son dependientes de la versión escogida [67]. En el caso de este proyecto, se usa la versión 2020.1 de Petalinux, pues Xilinx cuenta con un sistema de *rolling release*, donde continuamente lanza nuevas versiones sobre los que puede haber cambios importantes de unas a otras, por lo que la documentación y forma de trabajar son cambiante entre versiones.

Los requisitos *hardware* para instalar Petalinux en el equipo de desarrollo, son los siguientes:

- 8 GB RAM (mínimo)
- 2 GHz CPU (mínimo 8 cores)
- 100 GB de espacio libre en el disco duro.

En cuanto a los requisitos del sistema operativo, son los siguientes:

- Red Hat Enterprise Workstation/Server 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 8.1, 8.2 (64-bit)
- CentOS Workstation/Server 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 8.1, 8.2 (64-bit)
- Ubuntu Linux Workstation/Server 16.04.5, 16.04.6, 18.04.1, 18.04.2, 18.04.3, 18.04.4, 18.04.5, 20.04, 20.04.1 (64-bit)

Además, es necesario que se instalen un conjunto de librerías necesarias y que se instale como un usuario no *root*. También es necesario el uso de *bash* como s*hell* por defecto del sistema. En este proyecto se ha utilizado la distribución Ubuntu Linux 18.04.5 LTS.

### 6.3.2. CREACIÓN DEL SISTEMA OPERATIVO EMPOTRADO

Para crear un proyecto Petalinux, se sigue el flujo de trabajo presentado en la Figura 31. En primer lugar, se debe contar con el fichero que contiene la plataforma *hardware*

sobre la que se crea el proyecto (BSP o XSA). A continuación, se configuran los parámetros del sistema, el *kernel* y el sistema de ficheros raíz. Por último, se compila todo el proyecto y se generan los ficheros necesarios que irán en la tarjeta SD y que servirán para arrancar el sistema operativo.

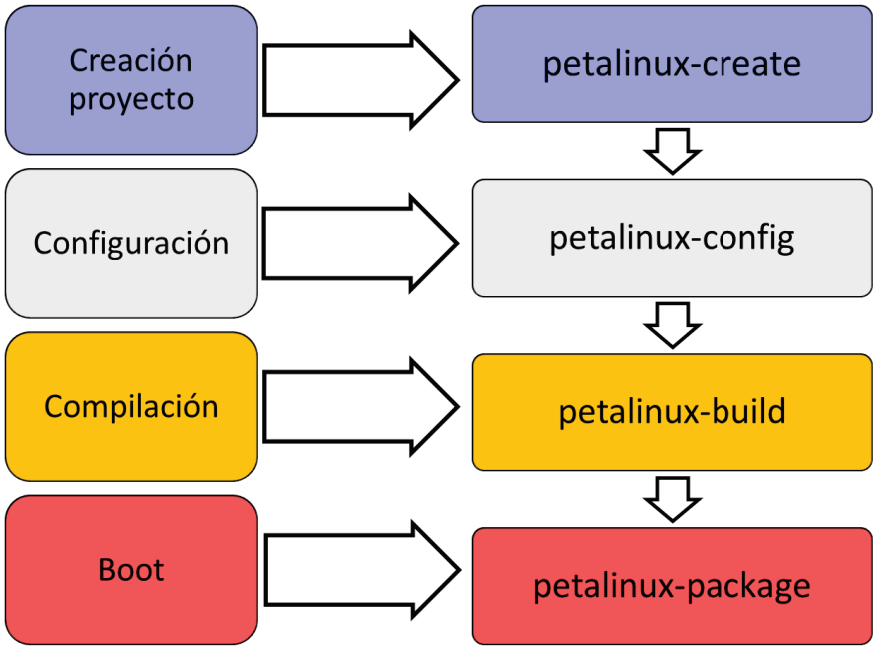



Figura 31. Flujo de trabajo Petalinux.

### 6.3.3. INSTALACIÓN DEL SISTEMA OPERATIVO

Para la implementación del sistema operativo empotrado, se han realizado las siguientes etapas con ayuda de la guía oficial:

**Paso 1:**

Se precisa el siguiente *hardware* previo a la instalación del sistema:

O1. Placa ZedBoard.

O2. Cable USB A a micro USB.

O3. Cable Ethernet.

O4. Tarjeta SD.

**Paso 2:**

Se crea el proyecto indicando la plantilla a utilizar y su nombre, en este caso se define la plantilla *zynq* y el nombre de *zedboard_os,* con el siguiente comando:

```
$ petalinux-create --type project --template zynq --name
zedboard_os
```

Con ello, se crea una carpeta con el nombre establecido que contiene un archivo de configuración del proyecto y la carpeta que contendrá todas las especificaciones.

**Paso 3:**

Para el resto de los comandos a utilizar hay que entrar en el ámbito del proyecto, es decir, ejecutar los comandos dentro de la carpeta creada en el apartado anterior. A continuación, se definirá la descripción de la arquitectura que tendrá el proyecto, para ello se puede utilizar un archivo BSP o XSA. En este proyecto se utiliza un archivo XSA, que se obtiene exportando el *hardware* diseñado desde la *suite* de diseño Vivado, como se ve a continuación:

```
$ petalinux-config --get-hw-description path_to_xsa_folder
```

Al final del proceso, se obtiene una ventana de configuración del sistema que se puede ver en Figura 32. En esta ventana se configuran aspectos como la MAC del sistema o el *hostname* del equipo.

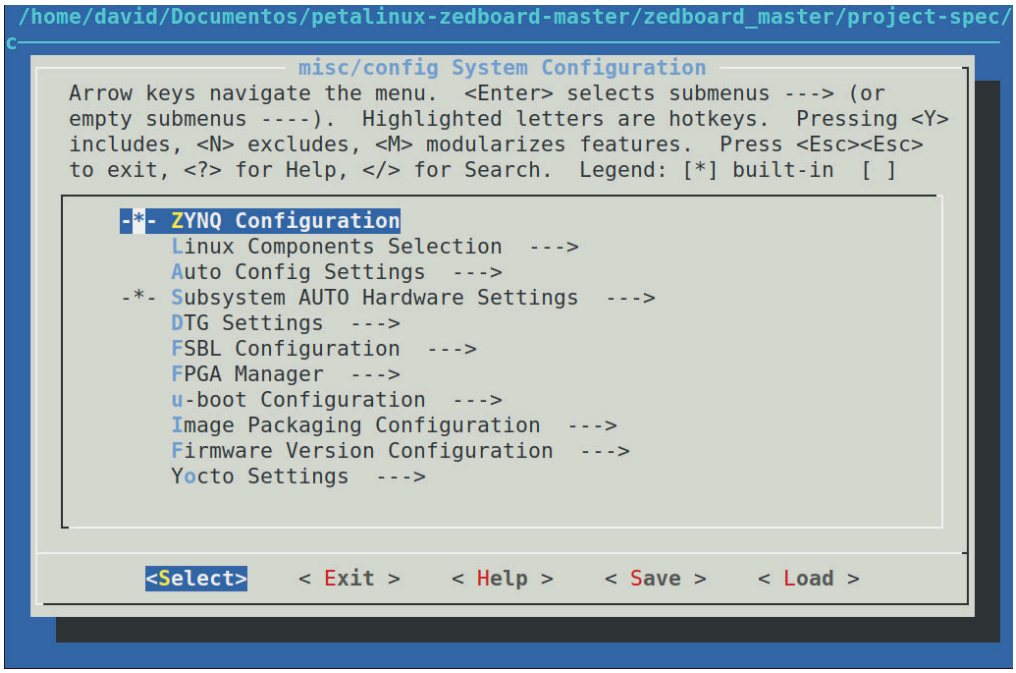



Figura 32. Ventana de configuración del sistema

**Paso 4:**

Una vez definida la configuración que soporta la plataforma *hardware* existente, se puede proceder a la configuración del *kernel* del sistema (Figura 33). Este es el elemento principal del sistema, y que actúa de capa intermedia entre el *hardware* y sus procesos. En este paso se definen todos los elementos que forman parte del *kernel* según los diferentes tipos de dispositivos que se vayan a utilizar (almacenamiento, RTC, interrupciones, GPIO, etc.). Para esta configuración se ejecuta el siguiente comando:

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

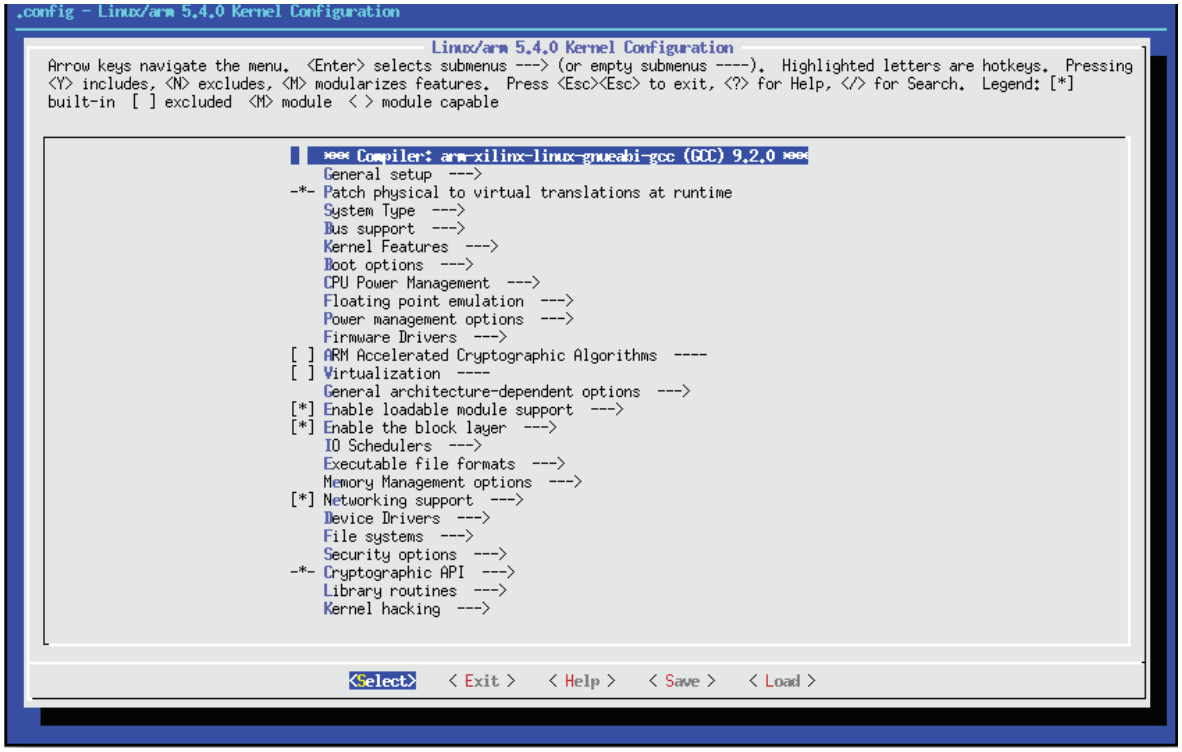



Figura 33. Ventana de configuración del kernel

**Paso 5:**

Seguidamente se configura el sistema de archivos que conforma el sistema operativo, agregando las carpetas necesarias para los directorios básicos de Linux (/, /bin, /usr, /proc, …). Contiene todos los archivos necesarios para arrancar el sistema Linux antes de montar otros sistemas de archivos. Se podrán elegir qué aplicaciones y utilidades tiene el sistema, como por ejemplo permitir montar

sistema de ficheros, utilidades para interactuar con el protocolo I2C, etc. Para configurar este apartado se introduce el comando abajo descrito, que abre la pantalla de la Figura 34.

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

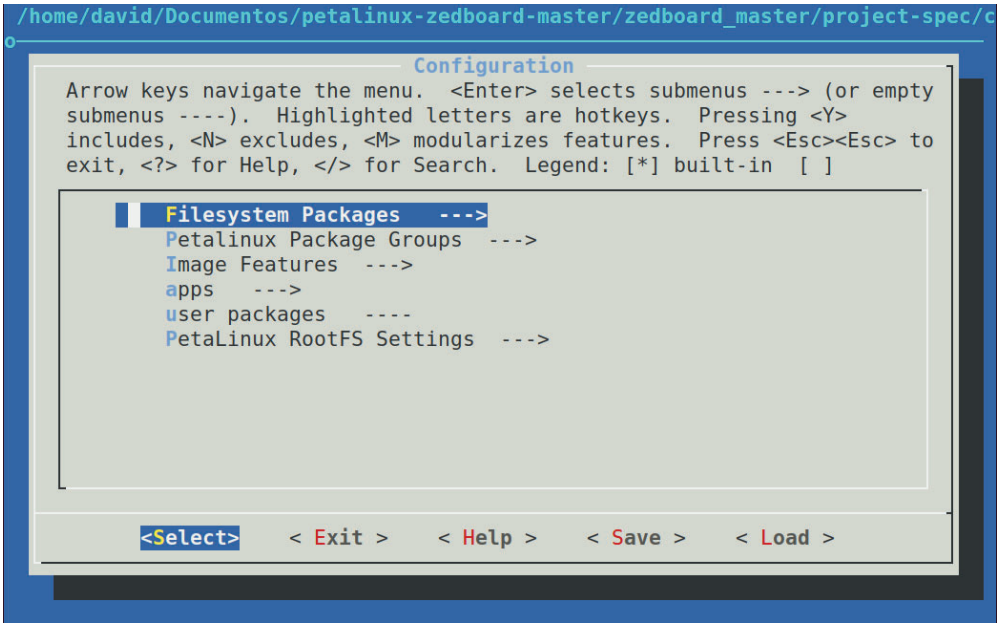



Figura 34. Ventana de configuración del rootfs

**Paso 6:**

Una vez configurado correctamente el sistema, se crean las aplicaciones objeto de este trabajo. Existen dos tipos de aplicaciones, de alto nivel escritas en C++ para realizar una serie de procesos y comunicaciones, y otra aplicación *baremetal* que permite la comunicación del *hardware* con las aplicaciones de alto nivel comentadas anteriormente.

Para la creación de las aplicaciones de alto nivel escritas en C++ se crea una plantilla del lenguaje indicado y se configura como activa en el *rootfs,* mediante el siguiente comando*:*

```
$ petalinux-create -t apps --name app_name --template c++ --enable
```

La aplicación *baremetal* también se crea haciendo uso de una plantilla. Para ello se introduce el siguiente comando:

```
$ petalinux-create -t apps --template install -n app_name --enable
```

Concluido estos procesos, se puede verificar que han quedado las aplicaciones creadas y activadas en el *rootfs* usando el comando anteriormente descrito para acceder a este y entrando al apartado *apps* como se ve en la Figura 35.

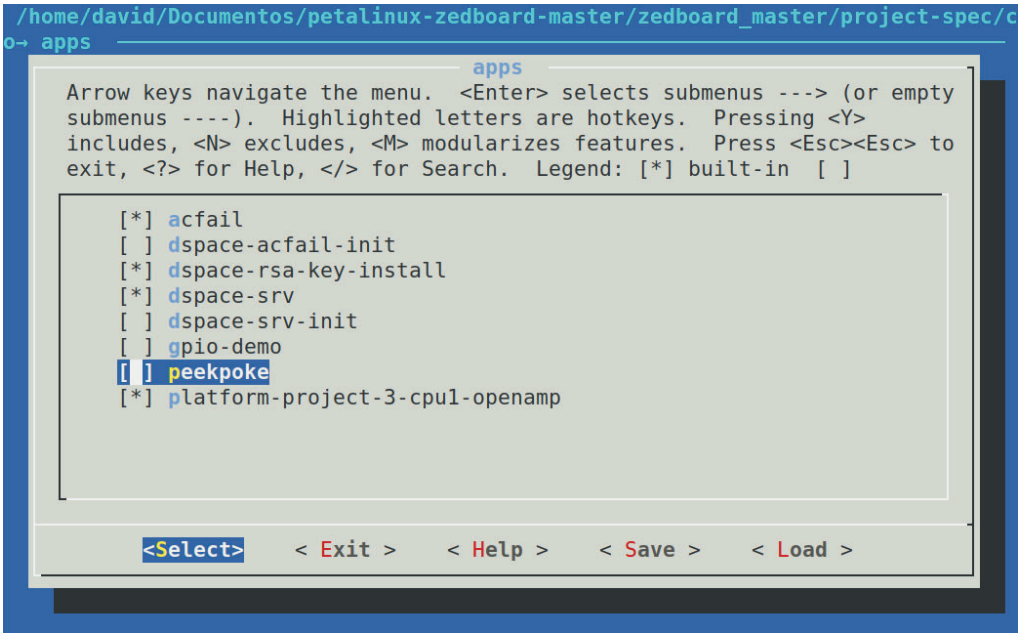



Figura 35. Aplicaciones creadas y activas en el rootfs

Las aplicaciones mostradas son las que se han creado con los comandos anteriores:

- Acfail, comprueba la alimentación del sistema
- Dspace-rsa-key-install contiene la contraseña de conexión SSH
- Dspace-srv es la aplicación que controla la configuración del sistema
- Platform-project-3-cpu1-openamp es la aplicación *baremetal*

**Paso 7:**

Una vez creadas las plantillas, pueden introducirse los ficheros que conforman cada aplicación en su correspondiente carpeta. Cabe recalcar que deben modificarse los ficheros *.bb* y *Makefile* de cada aplicación para incorporar todos los ficheros y dependencias necesarios para, en el caso que lo requiera, la compilación de la aplicación sea correcta.

Aquellas que deben compilarse son las aplicaciones de alto nivel una vez se haya realizado su configuración. Para ello se usará el siguiente comando:

```
$ petalinux-build -c app_name -f
```

Con el comando anterior especificamos el componente que queremos compilar, en este caso la aplicación desarrollada y sus correspondientes dependencias, indicando que se forzará dicha tarea.

**Paso 8:**

Una vez compiladas las aplicaciones se debe hacer la compilación del proyecto entero. Para ello se usa el comando:

```
$ petalinux-build
```

Debido a que la compilación es una de las partes que más tiempo requieren durante el desarrollo, existen técnicas para aumentar la velocidad de compilación e incluso favoreciendo la creación de otros proyectos de manera más rápida [68], como: proporcionar desde un servidor local las extensiones necesarias, versiones de Petalinux, carpetas comunes, etc.

**Paso 9:**

Terminada la compilación del proyecto se deben generar los ficheros correspondientes para el despliegue del sistema operativo en la tarjeta SD. Para ello, se realiza el empaquetamiento del proyecto Petalinux. Haciendo uso del comando:

```
$ petalinux-package --boot --fsbl
./images/linux/zynq_fsbl.elf --fpga
components/plnx_workspace/device-tree/device-
tree/platform_project.bit --u-boot --force
```

Con este comando se empaquetan todos los ficheros compilados anteriormente creando los ficheros esenciales para la tarjeta SD. Estos son:

- BOOT.bin: Es un fichero binario encargado del arranque. Responsable de cargar el *bitstream* de la FPGA, el FSBL y las utilidades *U-boot*.
- image.ub: Contiene el *kernel* y el *devicetree* de manera comprimida.

- boot.scr: Es el *script* encargado de cargar el *kernel* y otros elementos como el *devicetree.*
- rootfs.tar.gz: Contiene todos los ficheros referentes a la partición *rootfs* con la jerarquía del sistema.

**Paso 10:**

Con los archivos necesarios ya creados, se particiona la tarjeta SD de manera que exista una partición en FAT32 que contendrá los ficheros de arranque BOOT.bin, image.ub y boot.src, y, por otro lado, otra partición en ext4 que contendrá el sistema de ficheros *rootfs* como se ve en Figura 36.

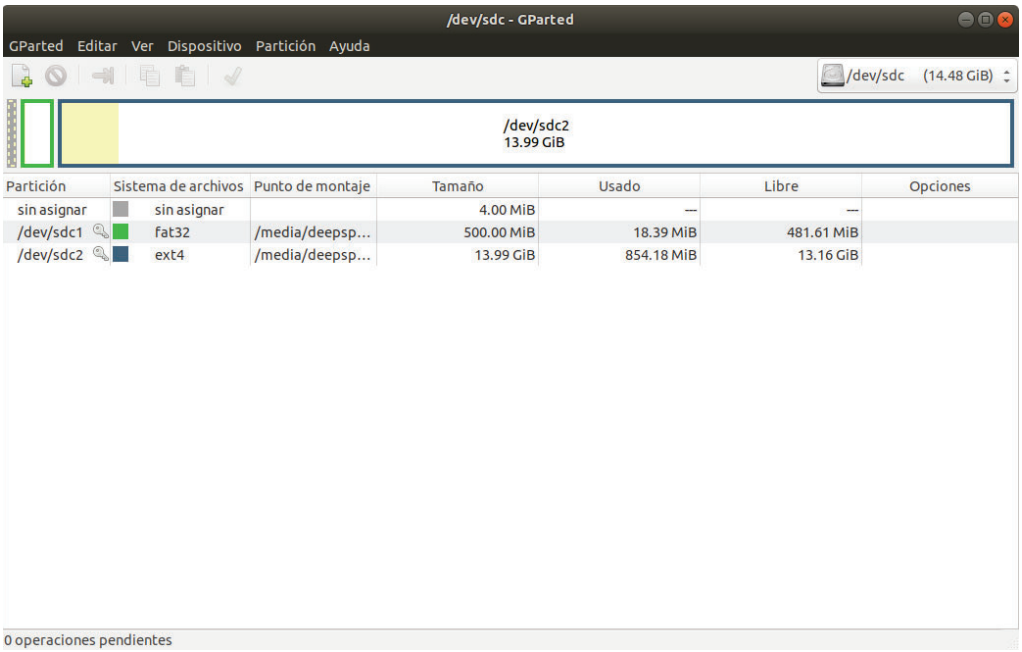



Figura 36. Particionado de la tarjeta SD

A continuación, se copian los ficheros encargados del arranque con el comando

```
$ sudo cp images/linux/file /media/usuario/fat32-partition
```

y se extrae el *rootfs* en la partición ext4 con el comando

```
$ sudo tar xvf images/linux/rootfs.tar.gz -C
/media/usuario/ext4-partition
```

**Paso 11:**

Para que la FPGA inicie correctamente y los periféricos necesarios funcionen, deben establecerse los *jumpers* de la placa es un orden específico:

1. Jumper en *JP2* para alimentar los dispositivos USB con 5V.

2. Jumper en *JP10* y *JP9* en la posición de 3V3.

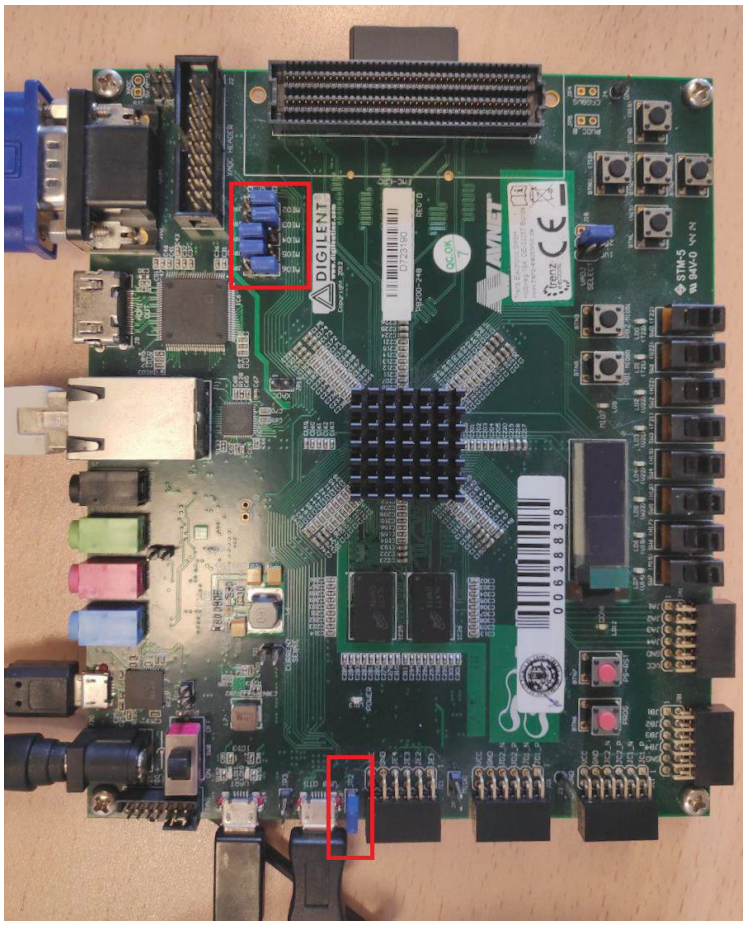



Figura 37. Configuración de jumper en la FPGA

### 6.3.4. SISTEMA DE FICHEROS

Petalinux posee su propia estructura de carpetas (Figura 38), donde cada una posee su función propia (*drivers*, paquetes, aplicaciones, etc.). Por tanto, la estructura de este proyecto se compone de la siguiente manera, donde en el directorio *recipes-bsp*, se pueden localizar las aplicaciones desarrolladas.

Otro de los ficheros importantes en cuanto a la customización del sistema embebido, es el fichero `system-user.dtsi`, donde se declaran aquellos componentes que se desean incluir en el *devicetree* y no han sido añadidos automáticamente por el generador automático a partir del fichero XSA. Para ello, existen diferentes guías [69] y documentación que se pueden consultar, ya que cada componente tiene características únicas y puede ser declarado de muchas maneras.

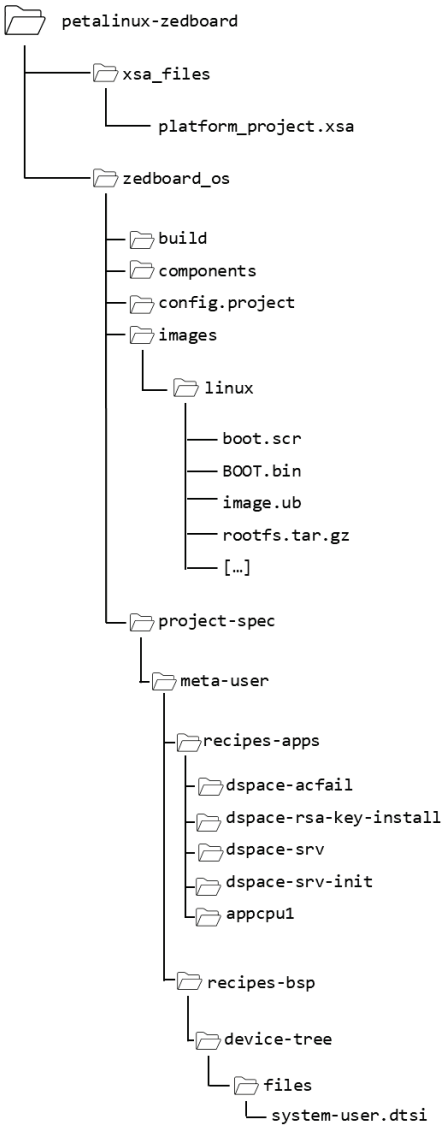



Figura 38. Jerarquía de ficheros del proyecto

### 6.3.5. CONFIGURACIÓN DEL SISTEMA

Para que todo el sistema funcione correctamente, se deben configurar los apartados del *kernel* y del *rootfs* para que incluyan todas aquellas librerías, módulos, utilidades y *drivers* que son necesarios para un correcto funcionamiento de todos los aspectos del sistema (RTC, USB, Ethernet, etc.).

A continuación, se indican las opciones de configuración incluidas en el sistema integrado en la aplicación tanto de la configuración general, del *kernel* y del *rootfs*, las cuales se configuran introduciendo los comandos `petalinux-config`, `petalinux-config -c kernel` y `petalinux-config -c rootfs` respectivamente.

**petalinux-config**
Subsystem AUTO Hardware Settings >> Ethernet Settings >> Ethernet MAC address (00:0A:35:00:01:24/25)
Firmware Version Configuration >> Host name (zedboard_[master/slave])
Firmware Version Configuration >>Product name (zedboard_[master/slave])
**petalinux-config –c kernel**
General Setup >> Control Group Support >> Memory Controller
General Setup >> Control Group Support >> Memory Controller >> Swap controller
General Setup >> Control Group Support >>Memory Controller >> Swap controller >> Swap controller enabled by default
General Setup >> Control Group Support >> IO controller
General Setup >> Control Group Support >> CPU controller
General Setup >> Control Group Support >> CPU controlller >> Group scheduling for SCHED_OTHER
General Setup >> Control Group Support >> CPU controller >> Group scheduling for SCHED_RR/FIFO
General Setup >> Control Group Support >> PIDs controller
General Setup >> Control Group Support >> Freezer controller
General Setup >> Control Group Support >> Cpuset controller
General Setup >> Control Group Support >> Cpuset controller >> Include legacy /proc/<pid>/cpuset file
General Setup >> Control Group Support >> Device controller
General Setup >> Control Group Support >> Simple CPU accounting controller
General Setup >> Control Group Support >> Perf controller
General Setup >> Control Group Support >> Debug controller
Device Drivers >> I2C support >> I2C Hardware Bus support >> Cadence I2C Controller
Device Drivers >> I2C support >> I2C Hardware Bus support >> Xilinx I2C Controller
Device Drivers >> I2C suppot >> Enable compatibility bits for old user-space
Device Drivers >> I2C suppot >> I2C device interface
Device Drivers >> I2C suppot >> I2C bus multiplexing support
Device Drivers >> Userspace I/O drivers >> Userspace I/O platform driver with generic IRQ handling
Device Drivers >> Remoteproc drivers >> <M> Support ZYNQ remoteproc
Device Drivers >> Real Time Clock >> Set system time from RTC on startup and resume
Device Drivers >> Real Time Clock >> Set the RTC time base don NTP synchronization
Device Drivers >> Real Time Clock >> RTC debug support
Device Drivers >> Real Time Clock >> RTC non volatile storage support
Device Drivers >> Real Time Clock >> /sys/class/rtc/rtcN (sysfs)
Device Drivers >> Real Time Clock >> /proc/driver/rtc (procfs for rtcN)
Device Drivers >> Real Time Clock >> /dev/rtcN (carácter devices)
Device Drivers >> Real Time Clock >> <M> TEST driver/device
Device Drivers >> Real Time Clock >> <M> Dallas/Maxin DS1307/37/38…
Device Drivers >> Real Time Clock >> Century bit support for rtc-ds1307
Device Drivers >> USB support >> USB Mass Storage support
Device Drivers >> USB support >> USB Mass Storage verbose debug
Device Drivers >> USB support >> USB Gadget Support
Device Drivers >> USB support >>  USB Gadget Support >> Debugging messages (DEVELOPMENT)
Device Drivers >> USB support >> USB Gadget Support >> Verbose debugging Messages (DEVELOPMENT
Device Drivers >> USB support >> USB Gadget Support >> <M> USB Gadget functions configurable through configfs
Device Drivers >> USB support >> USB Gadget Support >> Mass storage
Device Drivers >> USB support >> USB Gadget Support >> <M> Mass Storage Gadget
File systems >> Network File Systems >> NFS client support
File systems >> Network File Systems >> NFS client support for NFS version 2
File systems >> Network File Systems >> NFS client support for NFS version 3
File systems >> Network File Systems >> NFS client support for the NFSv3 ACL protocol extension
File systems >> Network File Systems >> NFS client support for NFS version 4
File systems >> Network File Systems >> NFS client support for NFSv4.1
File systems >> Network File Systems >> NFS client support for NFSv4.2

```
    File systems >> Network File Systems >> Root file system on NFS
petalinux-config –c rootfs
    Petalinux Package Groups >> packagegroup-petalinux-openamp
    Petalinux Package Groups >> packagegroup-petalinux-openamp-dbg
    Petalinux Package Groups >> packagegroup-petalinux-openamp-dev
    Filesystem Packages >> base >> i2c-tools >> i2c-tools
    Filesystem Packages >> base >> i2c-tools >> i2c-tools-dev
    Filesystem Packages >> base >> i2c-tools >> i2c-tools-dbg
    Filesystem Packages >> base >> i2c-tools >> i2c-tools-misc
    Filesystem Packages >> base >> usbutils >> usbutils
    Filesystem Packages >> base >> util-linux >> util-linux-hwclock
    Filesystem Packages >> console >> network >> nfs-utils >> nfs-utils
    Filesystem Packages >> console >> network >> nfs-utils >> nfs-utils-dev
    Filesystem Packages >> console >> network >> nfs-utils >> nfs-utils-client
    Filesystem Packages >> console >> network >> nfs-utils >> nfs-utils-stats
    Filesystem Packages >> console >> network >> nfs-utils >> nfs-utils-dbg
    Filesystem Packages >> libs >> boost >> boost-filesystem
    Filesystem Packages >> libs >> boost >> boost-program-options
    Filesystem Packages >> libs >> boost >> boost-iostreams
```

Como se puede observar, existe una gran lista de opciones que se deben configurar para que el sistema reconozca todos los dispositivos del sistema y no se produzcan funcionamientos erráticos.

El mapa de direcciones de los dispositivos implementados en la lógica programable (PL) en el espacio de memoria de Linux es el que se muestra en el siguiente listado. Como se puede apreciar, cada controlador diseñado está mapeado en el espacio de direcciones reservado al PL, teniendo en cuenta los detalles del hardware diseñado. Estas direcciones se encuentran en el fichero `system-user.dtsi`, que se usa para sobrescribir el *devicetree*, con objeto de modificar el comportamiento de los dispositivos deseados (añadiendo, modificando o anulando propiedades), o añadir dispositivos específicos que no estén soportados directamente, pudiendo también elegir el comportamiento de estos [70]. Xilinx proporciona información para establecer algunos elementos [71] en este *devicetree* personalizable, aunque también existe amplia lectura acerca de esto en el *kernel* de Linux [72].

```
Controlador        nombre controlador@direccion de memoria
-----------------------------------------------------------------------------------------------------------------------------
# IPs de captura de datos
  ip_axi4_stream_ad7768_if@43c50000
  ip_axi4_stream_ad7768_if@43c20000
# IPs de control de ganancias
  ip_timestamp@43c40000
  ip_timestamp@43c70000
# IPs para el cálculo del promedio
  ip_average@43c30000
  ip_average@43c60000
# IP DMA
dma@40400000
# IP Controlador de ganancias
  GainController@43c00000
# IP Control de modo PIN
  PinModeControl@43c10000
# Controladores modo SPI
axi_quad_spi@41e10000
axi_quad_spi@41e20000
# Controladores sincronización SPI
axi_quad_spi@41e30000
axi_quad_spi@41e00000
# Controladores GPIO (fuente alimentación, acfail, reset, start, …
gpio@41200000
gpio@41250000
gpio@41210000
gpio@41260000
gpio@41220000
gpio@41230000
gpio@41240000
# Controlador I2C RTC
i2c@41600000
```

### 6.3.6. ARCHIVOS EN EL ARRANQUE DEL SISTEMA

Para el correcto funcionamiento del sistema, tanto las aplicaciones como las utilidades desarrolladas deben arrancarse al inicio del sistema operativo. En la distribución Petalinux, existen varias maneras en las que conseguir este objetivo.

Una de estas consiste en la creación de una aplicación específica para añadir aquellos *scripts* que se deseen en ejecutar durante el arranque del sistema. Para ello como se indica en [73] se crea una aplicación con el siguiente comando

```
  petalinux-create -t apps --template install -n bootscript
--enable
```

Con esto, se crearán una serie de ficheros `bootscript.bb` y `bootscript` que contendrán las órdenes específicas de instalación de los scripts, rutas, etc. y el *script* con un mensaje inicial de "*Hello World*".

A continuación, se ejecutan los siguientes comandos para incluir de forma permanente la aplicación en el *rootfs.*

```
petalinux-build -c bootscript -x do_install -f
petalinux-build -c rootfs
petalinux-build
```

Con esto, la aplicación ha sido añadida correctamente al *rootfs* y, por tanto, cuando arranque el sistema se ejecutarán automáticamente los scripts creados.

Otra forma para conseguir introducir archivos al arranque del sistema en la partición *rootfs,* consiste en crear un *rootfs* externo desde el que Petalinux se nutra durante la compilación. Este método es recomendable hacerlo en la etapa final de configuración, de esta manera se evita introducir errores de configuración.

Para ello, una vez configurado el sistema y solo quedando pendiente la inclusión de *scripts* en el sistema de arranque, se compila el sistema de forma normal usando

```
petalinux-build
```

Una vez hecho esto y creados los archivos del sistema correspondiente con el comando

```
petalinux-package             --boot             --fsbl
./images/Linux/zynq_fsbl.elf                     --fpga
components/plnx_workspace/device-tree/device-
tree/$BITSTREAM_NAME --u-boot
```

en una carpeta donde se esté desarrollando el proyecto se copia el fichero rootfs.tar.gz y se extrae. Dentro de la carpeta se encuentran los directorios correspondientes del sistema Petalinux: *bin, boot, dev, home, etc.*

Una vez hecho esto se puede tener acceso por completo a todo el sistema de directorios y modificar o añadir lo que se desee. En este caso, la atención se centra en añadir *scripts* al arranque del sistema, para ello hay que centrarse en la carpeta *etc* y dentro

en los directorios *rcX.d*, que son los *scripts* o directorios de *scripts* que se ejecutan durante el arranque del sistema o al cambiar el nivel de ejecución.

En el caso de este proyecto, se ha decido usar el nivel 5 de ejecución debido a todos los componentes que contienen las aplicaciones desarrolladas y que esta se espera que se ejecute como una aplicación nativa del sistema. Por ello, los *scripts* se introducen en `etc/rc5.d/` y en `etc/init.d`. Cabe destacar que estos ficheros deben introducirse con la nomenclatura adecuada, por tanto, en el primero directorio, por ejemplo, el *script* que ejecuta el servidor se introduce como `S21dspace-serv-init.sh`. Así mismo, debe introducirse el *script* binomio en el segundo directorio, en este caso como `dspace-serv-init.sh`.

Una vez introducidos todos los *scripts* necesarios, queda configurar el *kernel*. Para ello se introduce el comando siguiente y se activa la opción que configure un *rootfs* externo como se ve en la Figura 39.

**petalinux-config –c kernel**
General Setup >> Initial RAM filesystem and RAM disk (initramfs/initrd) support >> Initramfs source file(s)

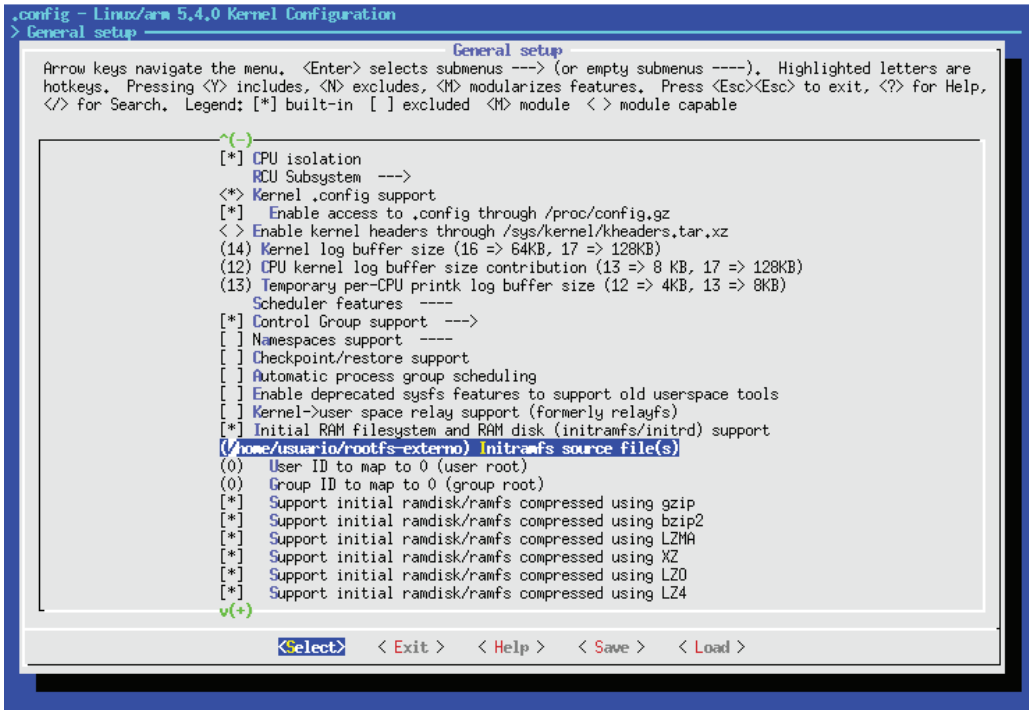



Figura 39. Configuración rootfs externo

En este último apartado, se debe introducir la ruta completa donde se encuentran los directorios del sistema extraídos anteriormente. Por ejemplo `/home/usuario/rootfs-externo`, ya que de este directorio se nutrirá Petalinux cuando se realice la compilación con el comando

```
petalinux-build
```

### 6.3.7. ARRANQUE DEL SISTEMA

El arranque del sistema precisa la conexión a la ZedBoard de una tarjeta SD, una conexión Ethernet, una conexión UART y los periféricos necesarios como RTC (Figura 40).

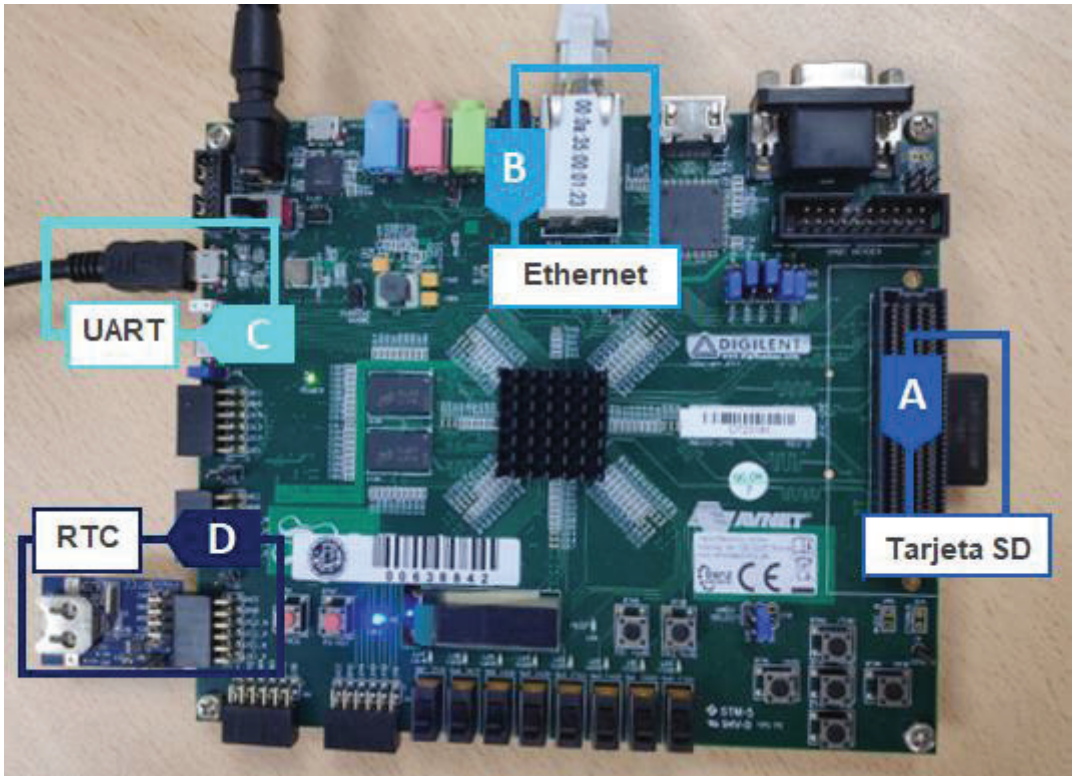



Figura 40. ZedBoard preparada para un primer arranque

Utilizando el programa minicom [74] se establece la conexión con la placa mediante el puerto UART, donde al inicio se muestra el *u-boot.* El comando *help* permite ver la ayuda del sistema como se puede ver en Figura 41 y Figura 42.


Figura 41. Arranque del sistema *u-boot* con el nombre Zynq en el *prompt*

```
Zynq> help
?         - alias for 'help'
base      - print or set address offset
bdinfo    - print Board Info structure
blkcache  - block cache diagnostics and control
boot      - boot default, i.e., run 'bootcmd'
bootd     - boot default, i.e., run 'bootcmd'
bootefi   - Boots an EFI payload from memory
bootelf   - Boot from an ELF image in memory
bootm     - boot application image from memory
bootp     - boot image via network using BOOTP/TFTP protocol
bootvx    - Boot vxWorks from an ELF image
bootz     - boot Linux zImage image from memory
clk       - CLK sub-system
cmp       - memory compare
coninfo   - print console devices and information
cp        - memory copy
crc32     - checksum calculation
dcache    - enable or disable data cache
dfu       - Device Firmware Upgrade
dhcp      - boot image via network using DHCP/TFTP protocol
dm        - Driver model low level access
echo      - echo args to console
editenv   - edit environment variable
env       - environment handling commands
erase     - erase FLASH memory
exit      - exit script
ext2load  - load binary file from a Ext2 filesystem
ext2ls    - list files in a directory (default /)
```

Figura 42. Comando *help* en *u-boot*

Entre los comandos anteriores se encuentra el comando *boot*, este es el encargado del arranque del sistema operativo cargando en memoria los datos contenidos en la tarjeta SD. Una vez introducido dicho comando, se observa en el terminal la comprobación del sistema donde se revisan: ficheros, conexiones, puertos, etc. Todo esto se realiza para garantizar su correcto funcionamiento, como se puede ver en Figura 43 y en Figura 44.

Una vez realizadas todas las comprobaciones, y si no se ha producido ningún problema, en el *prompt* de la terminal se solicita el usuario y contraseña del sistema, el cual por defecto es *root* para ambos (Figura 45).

```
   Trying 'kernel@1' kernel subimage
     Description:  Linux kernel
     Type:         Kernel Image
     Compression:  uncompressed
     Data Start:   0x100000e8
     Data Size:    4306368 Bytes = 4.1 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: 0x00200000
     Entry Point:  0x00200000
     Hash algo:    sha256
     Hash value:   d144728228d685b3b155266ccf55e352ac8215c756a7e3fd64f9c4a0d5d3d95f
   Verifying Hash Integrity ... sha256+ OK
## Loading ramdisk from FIT Image at 10000000 ...
   Using 'conf@system-top.dtb' configuration
   Verifying Hash Integrity ... OK
   Trying 'ramdisk@1' ramdisk subimage
     Description:  petalinux-image-minimal
     Type:         RAMDisk Image
     Compression:  uncompressed
     Data Start:   0x10422bb4
     Data Size:    88797783 Bytes = 84.7 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: unavailable
     Entry Point:  unavailable
     Hash algo:    sha256
     Hash value:   fa2ad03a112d97fbb611cb1a4f66c558a8f1fd2fb365647f904cfc675096a91e
   Verifying Hash Integrity ... sha256
```

Figura 43. Comprobación del *kernel* e integridad

```
IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
udhcpc: sending discover
udhcpc: sending select for 10.13.24.198
udhcpc: lease of 10.13.24.198 obtained, lease time 10800
/etc/udhcpc.d/50default: Adding DNS 193.145.138.100
/etc/udhcpc.d/50default: Adding DNS 193.145.138.200
done.
random: dbus-uuidgen: uninitialized urandom read (12 bytes read)
random: dbus-uuidgen: uninitialized urandom read (8 bytes read)
Starting system message bus: random: dbus-daemon: uninitialized urandom read (12 bytes)
dbus.
Starting haveged: haveged: listening socket at 3
haveged: haveged starting up


Starting Dropbear SSH server: dropbear.
Starting rpcbind daemon...done.
starting statd: done
Starting internet superserver: inetd.
exportfs: can't open /etc/exports for reading
NFS daemon support not enabled in kernel
Starting syslogd/klogd: done
Starting tcf-agent: OK
random: crng init done
random: 1 urandom warning(s) missed due to ratelimiting
```

Figura 44. Comprobación de la conexión Ethernet

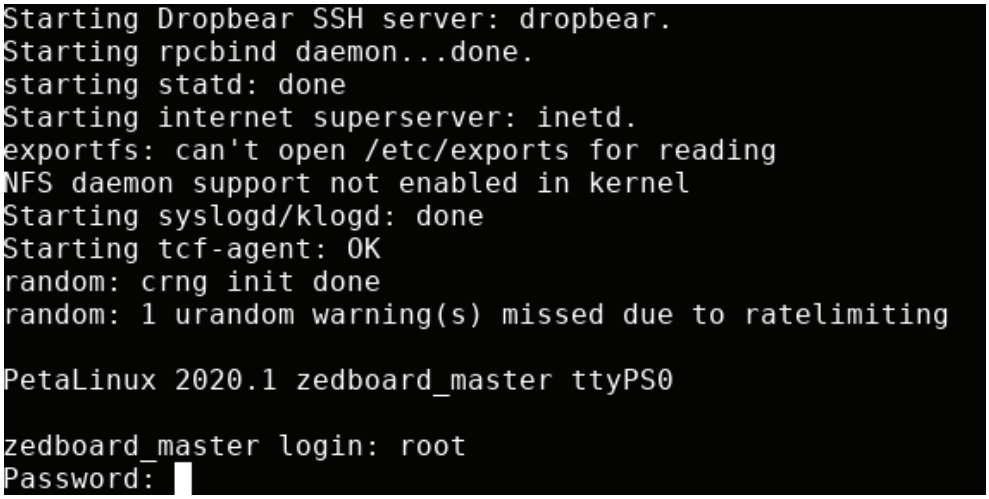



Figura 45. Petición de usuario y contraseña para el inicio de sesión

Ya introducidas las credenciales, tenemos un sistema Linux con funcionalidades básicas. Como se puede ver en Figura 46, introduciendo el comando *help* devuelve la versión de *bash* del sistema, así como otros comandos de interés.

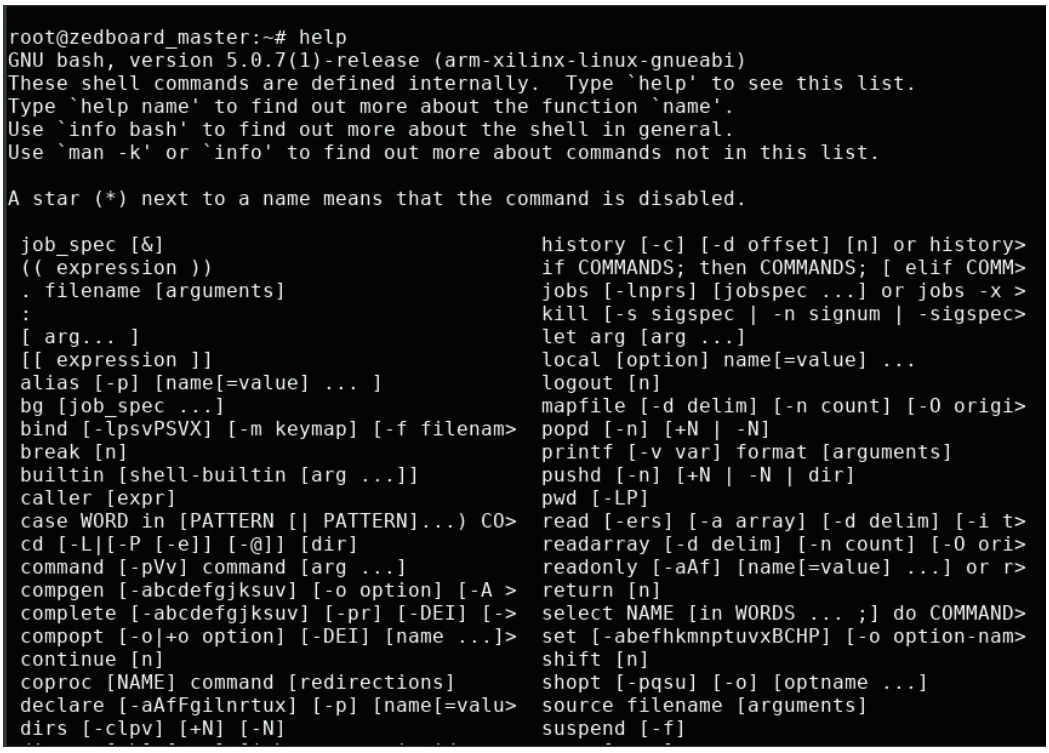



Figura 46. Comando *help* en la sesión del sistema

Si durante el arranque del sistema de produce algún error este se notifica en el terminal. Los más comunes suelen ser del tipo *kernel panic*, los cuales se pueden producir por diversos motivos: mala configuración de las direcciones de memoria, demasiados datos

para cargar en la memoria del sistema (RAM), exceso de funcionalidades activadas en el *rootfs,* etc.

### 6.3.8. OPTIMIZACIÓN DEL SISTEMA

El funcionamiento principal del sistema consiste en la captura de datos mediante los ADC. Esta es la funcionalidad fundamental y es el proceso que más recursos consume de la aplicación, recordando que estos, en una placa de este tipo son limitados. De los 512 MB de RAM de los que dispone, parte se usa para el sistema de ficheros, es decir, para el sistema operativo, por lo que solo quedan libres unos 270 MB de RAM para el funcionamiento de la placa.

Teniendo en cuenta que se trata de un sistema de captura de tiempo real, los recursos restantes se consuman rápidamente, pues las funciones utilizadas para crear los ficheros, así como para transmitirlos, hacen uso de la caché para la agilización de los procesos. Por ello, si el sistema no se optimiza correctamente, se puede producir que al inicio de la captura la caché llene toda la memoria del sistema haciendo que este deje de funcionar.

Para ello se utilizan mecanismos de control de caché. Por un lado, se puede modificar la variable `vm.vfs_cache_pressure`, dentro de nuestro sistema Petalinux, se encuentra en `/proc/sys/vm/vfs_cache_pressure`. Esta variable controla la tendencia del *kernel* a recuperar la memoria que se utiliza para el almacenamiento en caché del VFS (*Virtual File System)*, frente a la *pagechache* y la *swap.* Al aumentar este valor se incrementa la velocidad con la que se recuperan las cachés VFS. Esta variable, que por defecto está establecida en 100, tiene un rango de 0 hasta 200, donde el valor mayor implica mayor presión para recuperar las cachés. Para implementar una mayor presión, en este proyecto se establece una presión de 200 utilizando el comando

```
echo 200 > /proc/sys/vm/vfs_cache_pressure
```

Por otro lado, aunque la presión trabaja correctamente, sigue siendo insuficiente para mitigar el uso de memoria RAM tan acusado durante la captura. Por ello, durante los

procesos de captura, tanto antes como después se deben realizar una limpieza completa de la caché, para que el sistema pueda hacer uso de la mayor memoria disponible durante dicho proceso. Esto puede realizarse haciendo uso de la variable `drop_caches` la cual, dependiendo del uso requerido dispone de 3 niveles.

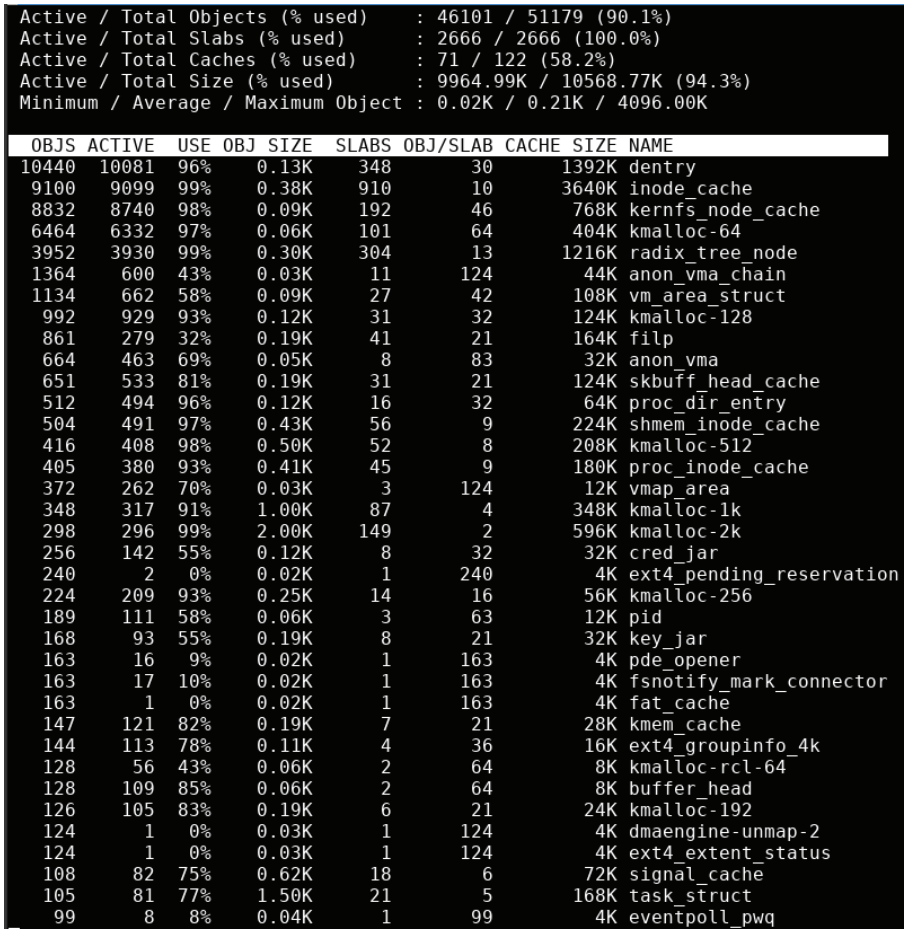
En el nivel 1 solo realiza una limpieza del *pagecache*, en el nivel 2 realiza la limpieza de los *dentries* y los *inodes*, en el nivel 3 realiza la limpieza de todos los anteriores. Para hacer un uso correcto de esta variable, antes debe usarse el comando `sync` el cual vaciará el *buffer* del sistema de archivos. En este proyecto, se usará el nivel más alto debido a la exigencia de la captura. Para ello se usan los siguientes comandos:

```
sync
echo 3 > /proc/sys/vm/drop_caches
```

Por último, este análisis de la memoria del sistema se puede realizar con la herramienta slabtop [75], incluida en el sistema Linux y permite ver en tiempo real la caché del *kernel*. Esta herramienta permite múltiples configuraciones para el estudio de la memoria, como la velocidad de actualización del uso, objetos activos, tamaño de la caché, etc. Utilizada para ver el espacio que ocupaba la caché del sistema, permitiendo analizar en qué casos era necesario liberar esta para un mejor rendimiento del sistema.

## 6.4. CONCLUSIÓN

En este capítulo se han conocido los requisitos de Petalinux para ser instalado en un sistema de tiempo real. Además, se ha mostrado cómo se personaliza el sistema operativo. Se ha mostrado el procedimiento para crear una imagen de Petalinux y del sistema de ficheros. Por último, se indica el procedimiento de arranque del sistema y cómo se optimiza para que funcione de la manera esperada.



```
Active / Total Objects (% used)    : 46101 / 51179 (90.1%)
Active / Total Slabs (% used)      : 2666 / 2666 (100.0%)
Active / Total Caches (% used)     : 71 / 122 (58.2%)
Active / Total Size (% used)       : 9964.99K / 10568.77K (94.3%)
Minimum / Average / Maximum Object : 0.02K / 0.21K / 4096.00K

  OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
10440  10081  96%    0.13K    348       30      1392K dentry
 9100   9099  99%    0.38K    910       10      3640K inode_cache
 8832   8740  98%    0.09K    192       46       768K kernfs_node_cache
 6464   6332  97%    0.06K    101       64       404K kmalloc-64
 3952   3930  99%    0.30K    304       13      1216K radix_tree_node
 1364    600  43%    0.03K     11      124        44K anon_vma_chain
 1134    662  58%    0.09K     27       42       108K vm_area_struct
  992    929  93%    0.12K     31       32       124K kmalloc-128
  861    279  32%    0.19K     41       21       164K filp
  664    463  69%    0.05K      8       83        32K anon_vma
  651    533  81%    0.19K     31       21       124K skbuff_head_cache
  512    494  96%    0.12K     16       32        64K proc_dir_entry
  504    491  97%    0.43K     56        9       224K shmem_inode_cache
  416    408  98%    0.50K     52        8       208K kmalloc-512
  405    380  93%    0.41K     45        9       180K proc_inode_cache
  372    262  70%    0.03K      3      124        12K vmap_area
  348    317  91%    1.00K     87        4       348K kmalloc-1k
  298    296  99%    2.00K    149        2       596K kmalloc-2k
  256    142  55%    0.12K      8       32        32K cred_jar
  240      2   0%    0.02K      1      240         4K ext4_pending_reservation
  224    209  93%    0.25K     14       16        56K kmalloc-256
  189    111  58%    0.06K      3       63        12K pid
  168     93  55%    0.19K      8       21        32K key_jar
  163     16   9%    0.02K      1      163         4K pde_opener
  163     17  10%    0.02K      1      163         4K fsnotify_mark_connector
  163      1   0%    0.02K      1      163         4K fat_cache
  147    121  82%    0.19K      7       21        28K kmem_cache
  144    113  78%    0.11K      4       36        16K ext4_groupinfo_4k
  128     56  43%    0.06K      2       64         8K kmalloc-rcl-64
  128    109  85%    0.06K      2       64         8K buffer_head
  126    105  83%    0.19K      6       21        24K kmalloc-192
  124      1   0%    0.03K      1      124         4K dmaengine-unmap-2
  124      1   0%    0.03K      1      124         4K ext4_extent_status
  108     82  75%    0.62K     18        6        72K signal_cache
  105     81  77%    1.50K     21        5       168K task_struct
   99      8   8%    0.04K      1       99         4K eventpoll_pwq
```

Figura 47. Herramienta slabtop [76]

# CAPÍTULO 7. APLICACIONES *SOFTWARE*

## 7.1. INTRODUCCIÓN

En este capítulo se explican las aplicaciones *software* desarrolladas en el proyecto. En primer lugar, se explica la aplicación de más bajo nivel la cual permite entender el funcionamiento del sistema entendiendo el binomio Linux-*baremetal* (*hardware-software*). A continuación, se explicarán el resto de las aplicaciones de alto nivel.

## 7.2. APLICACIÓN *BAREMETAL* 'APPCPU1'

Como ya se ha introducido anteriormente, la aplicación *baremetal,* de más bajo nivel, está desarrollada en C. Se encarga de conectar el *hardware* y el *software* del sistema, haciendo que sea posible un funcionamiento coordinado entre ambos dominios.

La aplicación *baremetal* `appcpu1` se encarga de mover los datos desde el DMA implementada en la FPGA hacia la memoria de intercambio de datos implementada en la OCM.

Presenta restricciones de tiempo real, por lo que se ha optado por separar su funcionalidad en la CPU1. La aplicación es cargada como *firmware* desde la CPU0 mediante el *framework* OpenAMP explicado con anterioridad. En la Figura 48 se muestra el diagrama de flujo utilizado por la aplicación *baremetal* y su interacción con la CPU0. Se utilizan dos *flags* ubicados en la memoria OCM (*CAPTURE* y *FILE*) que controlan el estado de ejecución del sistema.

La CPU0 indica el comienzo de la captura de datos desde el cliente, lo que hace que el servidor escriba el *flag* de captura a 1. Ello implica el borrado de *buffers* y cachés intermedios, la carga del *firmware* mediante mecanismos OpenAMP y el arranque de la

CPU1. A partir de aquí, la CPU1 verifica que el *flag* de captura realmente siga activo, inicializa la aplicación, limpia los *buffers* de escritura del *hardware* y habilita la captura. Eso implica que la interfaz de entrada de datos permite el acceso de las muestras desde el ADC hacia la plataforma de procesamiento en la FPGA (*timestamp*, promedio, interpretación de los datos, almacenamiento en *buffers* internos, validación de los datos, etc.).

Cuando se han transmitido suficientes datos para rellenar la OCM (256 muestras por canal x 16 canales => 65 Kbytes de muestras), se habilita el *flag* de escritura de fichero, lo que avisa a la CPU0 para que lea los datos desde la OCM y los escriba al correspondiente fichero. El proceso continúa mientras la captura esté activa. La captura se para desde la aplicación cliente, que comunica al servidor tal acción. Una vez parada la captura se limpian los *buffers* y la caché para continuar con el normal funcionamiento del sistema. El proceso asegura una exclusión mutua de acceso a los *flags*. De igual forma, el flujo de datos en la memoria siempre se escribe en una dirección, evitando su posible corrupción, creando un flujo de datos desde el ADC hasta el sistema de ficheros.

Los datos capturados se almacenan en el servidor con soporte para el sistema de archivos remoto NFS, creando una carpeta para cada captura, identificada por su fecha y tiempo de comienzo de captura. Los ficheros se almacenan de forma secuencial dentro de dicha carpeta.

En la aplicación diseñada se utilizan distintos *drivers* listados en la Tabla 2, siguiendo la arquitectura por capas mostrada en la Figura 29.

Tabla 2. Drivers y funciones de la aplicación Standalone

| Fichero | Funcionalidad |
|---|---|
| **sleep.h** | Funciones de control temporal |
| **stdio.h** | Funciones básicas de E/S |
| **time.h, xtime_l.h** | Funciones para la gestión de medida del tiempo |
| **xaxidma.h** | Funciones para controlar el bloque DMA |
| **xgpio.h** | Funciones para la gestión de E/S mediante GPIO |
| **xil_exception.h** | Excepciones definidas por Xilinx |

| Fichero | Funcionalidad |
|---|---|
| **xil_io.h** | Funciones básicas de E/S definidas por Xilinx |
| **xil_mmu.h** | Funciones para acceder a la MMU |
| **xil_printf.h** | Redefinición de las funciones *print* de Xilinx |
| **xil_types.h** | Tipos básicos definidos por Xilinx |
| **xparameters.h** | Parámetros de la plataforma |
| **xpseudo_asm.h** | Facilidades para escribir instrucciones en ensamblador desde C |
| **xspi.h, qspi.h** | Funciones para el acceso a los bloques SPI |
| **xstatus.h** | Estados de ejecución definidos por Xilinx |
| **xuartps_hw.h** | Acceso a la UART |

## 7.3. APLICACIÓN 'DSPACE-SERVER'

Esta aplicación se ejecuta en el sistema Linux. Es el núcleo encargado de transmitir las órdenes enviadas por el usuario desde el cliente al *hardware.*

El lenguaje elegido para desarrollar esta aplicación ha sido C++ teniendo en cuenta que se deben controlar aspectos de bajo nivel, por tanto, hará uso de algunas librerías y *drivers* escritos en C para comunicarse con los *drivers* del *hardware*.

Controla las opciones utilizadas para configurar y controlar el sistema de adquisición, incluyendo los modos de captura en modo PIN, en modo SPI, el control de ganancias, la configuración de promedios, captura de datos, distintos comandos de sincronización, estado y apagado del sistema.

A continuación, se indican las principales opciones que se detallarán más adelante en este apartado. Hay que tener en cuenta que las distintas opciones del servidor únicamente se muestran cuando se accede desde la aplicación `dspace-client` pues es este el único encargado de comunicarse con él.

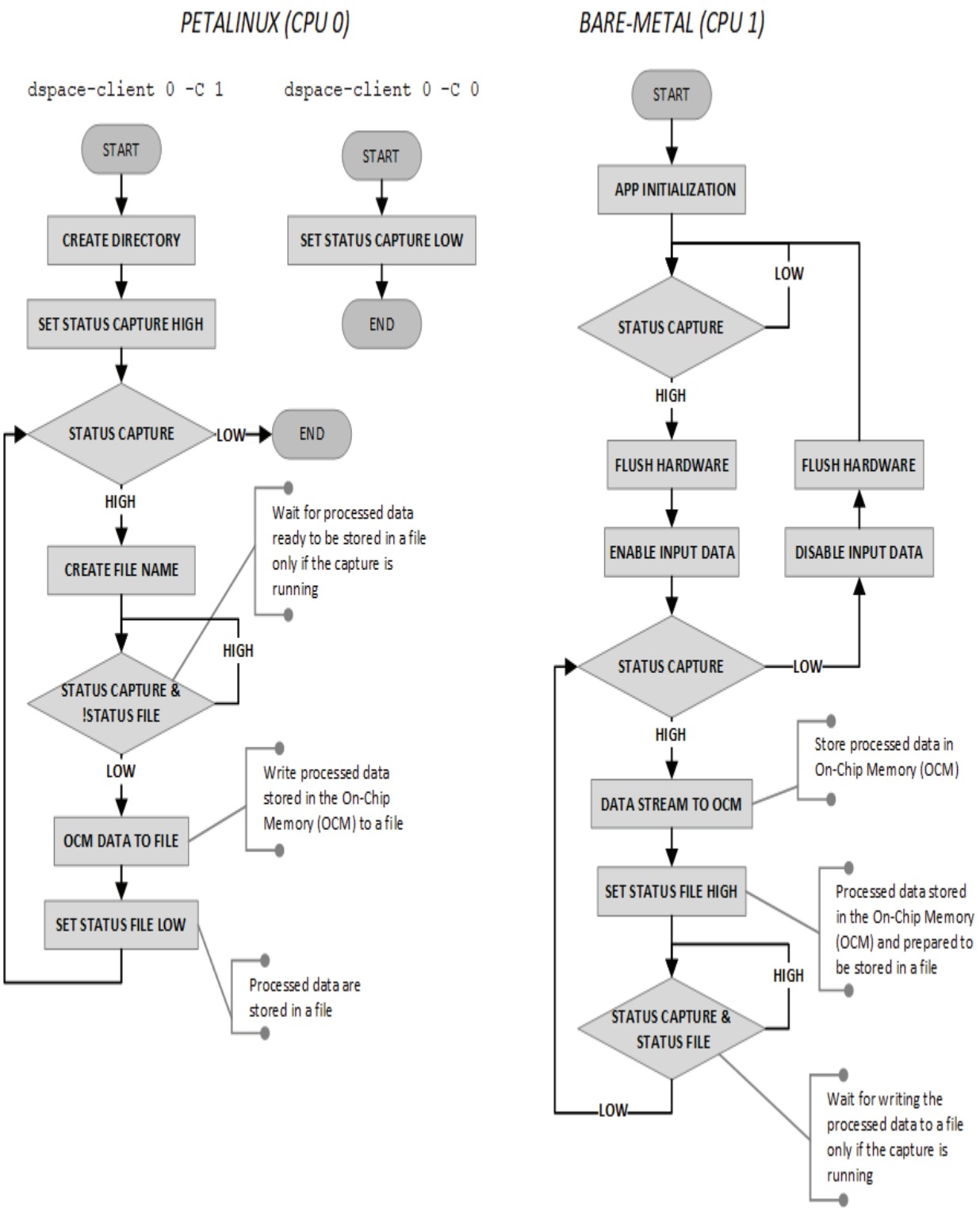



Figura 48. Aplicación *baremetal* (derecha) y su interacción con la CPU0 (izquierda)

```
dspace-client [ID_board] [IP-commands or commands]

WARNING! The number of channels changes depending on the selected board.
WARNING! Checks if the RTC time is correct.

Allowed options:

  -P [ --pin_mode ]      Set PIN mode.

...
  -S [ --spi_mode ]      Set SPI mode.
...

  -G [ --gain ] arg      Change amplifier gain.
...

  -A [ --average ] arg   Set the number of samples to be averaged.
…

  -X [ --pixel ] arg     Set pixel state.

  -C [ --capture ] arg   Start/stop data capture.

  -M [ --module ] arg    Start up or shutdown ADC module.

  -h [ --help ]          Help.


System commands:
  --sync          Synchronize system.
                  Note: this option can only be used for the MASTER board.
                  WARNING! This needs to be launched after each
                  system configuration.

  --status        Print state of the system from current state file.
                  WARNING! Each board has its own state file.

  --status-update Read state from hardware and print current state file.
                  WARNING! Each board has its own state file.

  --reboot        Reboot system.
                  WARNING! Reboot only the selected board.

  --shutdown      Shut down system.
                  WARNING! Shut down only the selected board.
```

En los apartados que siguen se explica la arquitectura de la aplicación y se dan detalles de la funcionalidad más en profundidad.

### 7.3.1. ARQUITECTURA DE LA APLICACIÓN

En la Figura 49 se muestra el diagrama de bloques de la aplicación en la que se puede apreciar las diferentes capas y su modularidad. Esta aplicación, tiene una

arquitectura cliente-servidor, donde la parte del cliente se aborda en capítulos posteriores. Como se puede apreciar, la aplicación se ha organizado en cuatro capas: *main*, comunicaciones, gestión de comandos y capa de lógica.

La capa *main* incluye la función principal del programa y se encarga de llamar a las funciones que inicializan el servidor, la configuración inicial del sistema y la configuración del ADC. Además, hace una primera lectura del sistema para crear un fichero de estado.

La capa de comunicación maneja las conexiones TCP/IP y las comunicaciones entre cliente y servidor.

La capa de gestión de comandos recibe los comandos y verifica que los comandos enviados por el cliente son correctos o pueden ser procesados.

Por último, la capa de lógica maneja toda la lógica de la aplicación, es decir, procesa los datos recibidos de los comandos, los valida y se comunica con la capa inferior para llevar a cabo la configuración de las órdenes recibidas. Esta capa se apoya en distintos *drivers* de Linux, *drivers* desarrollados durante el trabajo para esta aplicación y funciones de bajo nivel diseñadas para implementar las funciones necesarias para controlar la lógica del ADC y del sistema *hardware* implementado sobre el PL del dispositivo Zynq. Estas funciones son dependientes del dispositivo usado (AD7768) y del diseño implementado, y han sido optimizadas para su funcionamiento en este sistema.

Como puede observarse la aplicación se ha diseñado siguiendo una filosofía modular [77], [78], permitiendo que puedan seguir funcionando individualmente. Por ejemplo, podría quitarse la capa *Server* y el sistema podría reconfigurarse para funcionar de forma local como un sistema basado en comandos en línea (CLI).

#### *7.3.1.1. Dspace-srv*

Es la función principal (*main*) del programa, como ya se ha comentado, se encarga de llamar a las funciones pertinentes para la inicialización del servidor, la configuración inicial del sistema y del ADC y realizar la primera lectura para la creación del fichero de configuración `board-config.json` que recoge el estado del sistema. Este último se guarda en el directorio `/home/root/`.

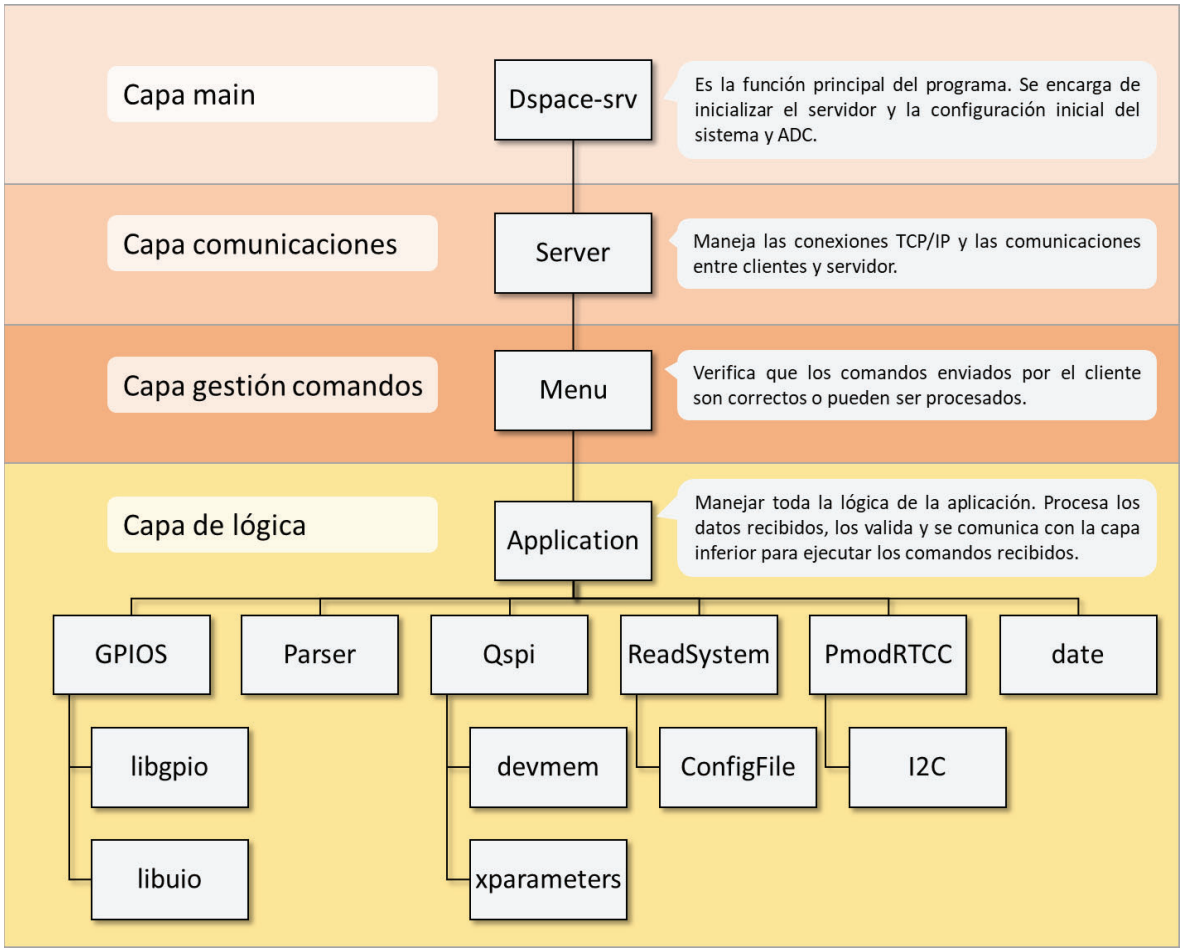



Figura 49. Arquitectura de la aplicación dpace-srv

El servidor arranca con la IP asignada al equipo (proveniente de la configuración DHCP), la MAC 00:0A:35:00:01:24/25 y en modo escucha en los puertos 8092 (tarjeta maestra) y 8093 (tarjeta esclava). Se encarga de arrancar las fuentes de alimentación en la secuencia prevista y se sincronizan los relojes de las tarjetas maestra y esclava. Este flujo puede observarse en Figura 50. En el Listado 1 se muestra parte del código que realiza esta función.

```
//System start
Server srv;
Application appSystem;
appSystem.IsMaster();
appSystem.StartADCSystem();
sleep(1);
printf("Reading system.\r\n");
appSystem.readSystem.ReadEntireSystem(JSON_CONFIG_FILE);
printf("System read.\r\n");

//Server start
srv.Server_Init(); // start server with initial parameters
```

```
while(true)
{
    srv.SendAndReceive(); // start data reception from client and the
response
}
```

Listado 1. Extracto del código de la aplicación servidor

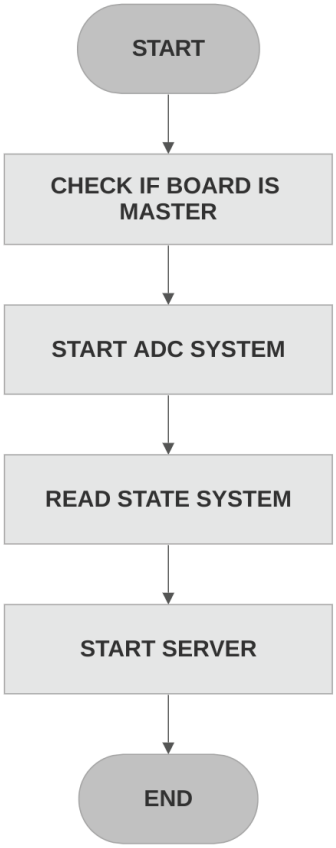



Figura 50. Secuencia de arranque del sistema

El sistema arranca con la siguiente configuración inicial:

- Fuentes arrancadas en la secuencia prevista
- Captura de datos deshabilitada
- Modo SPI deshabilitado
- Modo PIN activado
    - Filtro: Wideband
    - Modo: Fast MCLK/4
    - Canales: Todos activos
    - Decimación: x32
- Ganancia: 100 (40 dB) +/- 50mV
- Pixel: Deshabilitado
- Cálculo de promedios

Una vez el servidor haya sido iniciado correctamente se imprime en pantalla algunos datos relevantes del sistema, como la hora de ambas placas para comprobar que se han sincronizado correctamente, la IP de la placa, el *hostname*, etc. (Figura 51).


Figura 51. Información de arranque de dspace-server

*7.3.1.2. Server*

Esta clase se encarga de manejar las conexiones TCP/IP y las comunicaciones entre cliente y servidor. Para ello se hace uso de procedimientos estándares soportados en modo sockets, a partir de funciones C/C++ presentes en Linux de modo nativo [79]. Hacen uso de librerías propias del lenguaje o del propio sistema Linux, por lo tanto, no es necesario el uso de librerías externas. Soportan la posibilidad de que múltiples clientes puedan lanzar órdenes contra el servidor, de manera que se pueda generar una cola y todos sean atendidos, abriendo y cerrando los *sockets* que se consideren oportunos según el número de cliente.

Existe una amplia literatura [80], [81] y librerías para la programación de redes, de donde se han extraído los mejores mecanismos para que dicha conexión sea estable y con

la mayor reducción de fallos: comprobación de *socket,* puertos, *timeout,* etc. El mecanismo propuesto en este trabajo realiza un intercambio de información donde tanto cliente como servidor envían en primer lugar el tamaño de la trama que van a enviar y posteriormente la trama. Esto permite al destinatario crear un buffer del tamaño adecuado donde recibir toda la información, manteniendo control de la integridad de los datos transmitidos y recibidos (Listado 2).

```
//the length of the data to send
send(new_socket, &length, sizeof(length), 0);

//if the data to send is very large it will be sent several times,
//but this is an internal mechanism of the function
send(new_socket, data_to_send.data(), data_to_send.size(), 0);
```

Listado 2. Gestión de la conexión cliente/servidor mediante *sockets*

Una vez iniciado el servidor, permanece en modo escucha en los puertos 8092/8093 según se trate de que la placa actúe como maestra o como esclava respectivamente. Cuando recibe una trama desde un cliente, se encarga de actuar como pasarela de datos hacia la capa *Menu* que procederá a su validación. Según el resultado de dicha validación o el comando introducido, el cliente podrá recibir una respuesta específica desde el servidor.

#### *7.3.1.3. Menu*

Esta clase se encarga de validar si los comandos recibidos desde el cliente son correctos o pueden ser procesados, comprobar que cumplen las dependencias adecuadas, verificar las incompatibilidades entre comandos y comprobar el archivo JSON de estado del sistema. Además, verifica que el servidor no se encuentra en modo captura, su modo principal de funcionamiento, en cuyo caso solamente permite ejecutar los siguientes comandos: *help, status, status-update, reboot, shutdown y capture 0* (parar captura).

Este menú hace uso del conjunto de la biblioteca de *software* libre Boost 1.71v [82], en concreto se utiliza la librería Program Options [83] que permite la introducción de "opciones de programa", entendido como un conjunto de nombre-valor, mediante una interfaz de línea de comandos (CLI). La utilización de esta librería parte de la premisa de que es una librería ampliamente soportada, incluidas en las herramientas de Petalinux, encontrándose numerosos ejemplos [84], [85] y documentación [86].

Por otro lado, es una librería ligera, tiene una sintaxis simplificada y la complejidad de la gestión de los valores introducidos se simplifica desde el punto de vista del diseño del menú, en contraposición a desarrollar un menú desde cero. Además, posee notificaciones de errores mejorado y manejo de estos, pudiendo personalizar los mensajes de error. También, permite crear archivos de configuración o variables de entorno que proporcionan al usuario una mayor flexibilidad a la hora de usar la aplicación mediante la línea de comandos.

Dada la complejidad introducida en la aplicación por las distintas opciones soportadas por el sistema, esta librería permite la construcción de un menú robusto y fiable, pudiendo establecer numerosas funcionalidades y personalizar al máximo la experiencia de usuario.

Como se ha comentado anteriormente, este menú comprueba si la validación de los comandos introducidos es o no satisfactoria, haciendo que el cliente reciba un mensaje indicando cuando se produce un error y advirtiendo acerca de cuál ha sido. En los casos en los que la validación sea correcta, el cliente no recibirá ningún mensaje salvo en los comandos que explícitamente lo requieren como *help, status, etc.*

Los mensajes de error pueden generarse por los siguientes motivos:

- Nombre incorrecto del comando o desconocido (Figura 52)
- Falta uno o más argumentos del comando
- El comando introducido no sigue el patrón especificado
- El argumento introducido no es del tipo esperado (número, tamaño, valor, etc.) (Figura 53)
- El comando introducido no cumple las dependencias establecidas
- Los comandos introducidos contienen conflictos entre ellos

Algunas de las comprobaciones, las realiza la propia librería de manera automática y otras se hacen mediante el uso de funciones diseñadas a tal fin.

```
deepspace@vlsiws20:~/nfs$ ./dspace-client 0 --configurar
Unknown option: --configurar
Please type --help or -h for more information.
deepspace@vlsiws20:~/nfs$
```

Figura 52. Comando introducido desconocido

```
deepspace@vlsiws20:~/nfs$ ./dspace-client 0 -P -s 45
Error in command --standby from --pin_mode. Ivalidad value.
deepspace@vlsiws20:~/nfs$
```

Figura 53. Valor 45 invalido para la configuración en modo Pin

Para la comprobación de las dependencias y conflictos se usan un conjunto de funciones que se describen a continuación.

- Función `ConflictingOption`. En este caso, se comprueba que en la variable `variables_map`, que contiene todas las órdenes recibidas durante una conexión por un cliente, no se encuentran dos opciones incompatibles (Listado 3). Este caso únicamente se da entre los modos SPI y PIN, por razones de características del *hardware*.

- Función `OptionDependency`. Esta función comprueba que en la variable `variables_map` no se encuentran comandos que dependan de otro, como es el caso de la configuración de los canales activos en el modo SPI (Listado 4, Figura 54).

```
/**
 * @brief Function used to check that 'opt1' and 'opt2' are
 *        not specified at the same time.
 * @param vm The name of the file to check.
 * @param opt1 The name of the file to check.
 * @param opt2 The name of the file to check.
 * @return string with the conflicting option or empty if all is ok.
 */
std::string Menu::ConflictingOptions(
const boost::program_options::variables_map& vm,
const char* opt1, const char* opt2)
{
    if (vm.count(opt1) && !vm[opt1].defaulted() && vm.count(opt2) &&
        !vm[opt2].defaulted())
    {
        std::string op = opt1;
        return "Conflicting options '--" + op + "' and '--" +
               opt2 + "'.\n";
    }
    return "";
}
```

Listado 3. Función `ConflictingOptions`

```
/**
 * @brief Function used to check that of 'for_what' is specified,
 *        then 'required_option' is specified too.
 * @param vm The name of variable map (container for saving
 *        selected program options).
 * @param for_what The name of the dependent command.
 * @param required_option The name of the main command without
 *        which the command entered in 'for_what' cannot be used.
 * @return string with the conflicting options or empty is all is ok.
 */
std::string Menu::OptionDependency(
const boost::program_options::variables_map& vm,
const char* for_what, const char* required_option)
{

    if (vm.count(for_what) && !vm[for_what].defaulted())
    {
        if (vm.count(required_option) == 0 ||
            vm[required_option].defaulted())
        {
            std::string fw = for_what;
            return "Option '--" + fw + "' requires option '--" +
                   required_option + "'.\n" ;
        }
    }
    return "";
}
```

Listado 4. Función `OptionDependency`

```
deepspace@vlsiws20:~/nfs$ ./dspace-client 0 -s 45
Option '--standby' requires option '--pin_mode' or '--spi_mode'
deepspace@vlsiws20:~/nfs$
```

Figura 54. Dependencia de configuración del modo Pin o SPI

Un ejemplo de uso de estas dos funciones es el que se muestra a continuación, donde se comprueba que no exista conflicto entre el modo SPI y PIN, y que si en el mapa de variables se encuentra el comando `active` también debe encontrarse el comando `spi_mode` (Listado 5, Figura 55).

```
//Exclusions between commands
std::string pin_or_spi = ConflictingOptions(m_vm, "spi_mode",
"pin_mode");

// Dependecies between commands
std::string filter_status_pin = OptionDependency(m_vm, "filter",
"pin_mode");
std::string  active_status_spi  =  OptionDependency  (m_vm,  "active",
"spi_mode");
```

Listado 5. Ejemplo de utilización de las funciones de gestión de conflictos entre opciones del menú.

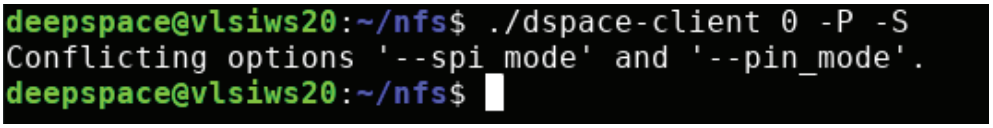



Figura 55. Conflicto entre modo Pin y SPI (`ConflictingOptions`)

Por otro lado, esta clase comprueba si existe el fichero JSON que indica el estado del sistema, ya que si no existe se crea y se procede a realizar una lectura del estado del sistema. Si por el contrario este ya existe, se procede a una lectura del fichero y posterior lectura y escritura de los datos que hayan podido ser modificados por el usuario, quedando así actualizado este fichero de cara a posibles consultas.

```
//Check if config file exists

if(!application.configFile.CheckIfConfigFileExits(JSON_CONFIG_FILE))
{
    application.readSystem.ReadEntireSystem(JSON_CONFIG_FILE);
}

// Open the config file to read it
application.readSystem.ReadConfigFile(JSON_CONFIG_FILE);
[…]
// Write the config file and close
application.readSystem.WriteConfigFile(JSON_CONFIG_FILE);
```

Listado 6. Gestión del estado del sistema

*7.3.1.4. Application*

Esta es la clase principal de la aplicación ya que maneja toda su lógica. Se encarga de procesar los argumentos de los comandos recibidos que han pasado la primera validación por la clase *Menu*, los valida nuevamente mediante métodos adicionales y se comunica con las capas inferiores para ejecutar las configuraciones solicitadas.

Algunos comandos recibidos no pueden ser validados automáticamente por la capa *Menu* porque se ha buscado separar responsabilidades entre ambas y validar algunos argumentos en la capa de la lógica debido a la modularidad indicada anteriormente. Por ello, los argumentos de tipo *string* que poseen formatos personalizados definidos por diseño, serán validados en esta capa. Por ejemplo, aquellos comandos donde los argumentos deben introducirse con signos de puntuación como son los dos puntos (:) para separar las diferentes configuraciones.

Un ejemplo de ello es el caso de la configuración en modo pin de los canales en *standby* (Listado 7). En este caso se verifica el tipo en la clase *Menu*, posteriormente en la función se separa la cadena para verificar que tiene el tamaño correcto y se comprueba si los datos introducidos son números. Si no se cumplen estas condiciones se envía un error al usuario indicando estos errores. Si se cumple, se comprueba si el número enviado se encuentra dentro de un mapa [87] previamente establecido como número-valor. Si es así se establecen en *standby* los canales deseados, pero si no se encuentra en el mapa, se envía una excepción indicando que existe un error en dicho comando.

```
/**
 * @brief Function used to set standby channels in Pin Mode.
 *        Options 1 to 10.
 * @param spiPinModeStandbyChannels is the chosen option.
 *        Is string because the same variable is reused.
 */
void Application::PinModeStandby(
std::string &spiPinModeStandbyChannels)
{
    int data_read;
    int data_write;

    std::vector<std::string> channel =
    parser.SplitStringColon(spiPinModeStandbyChannels);
    bool allAreNumbers = Parser::CheckStringNumberCommand(channel);

    try
    {
        if(!allAreNumbers)
        {
            throw std::invalid_argument("Error in command --standby
                  from --pin_mode. Not all parameters are numbers.\n");
        }

        if (standby_channels.find(stoi(channel[0])) !=
            standby_channels.end())
        {
            data_read = SpiPinMode_read(IP_PINMODECONTROL_BASEADDR +
                        PINMODECONTROL_REGISTER_OFFSET);
            data_write = (data_read & 0xFFFFFFE1) |
                          standby_channels.find(stoi(channel[0]))->
                          second;
            SpiPinMode_write((IP_PINMODECONTROL_BASEADDR +
                              PINMODECONTROL_REGISTER_OFFSET),
                              data_write);
        }
        else
        {
            throw std::invalid_argument("Error in command --standby
                                         from --pin_mode. Invalid
                                         value.\n");
        }
```

```
        readSystem.ReadPinModeStandby();

    }
    catch (std::invalid_argument& e)
    {
        message_to_send += e.what();
        return;
    }
}
```

Listado 7. Ejemplo de validación adicional de argumentos

*7.3.1.5. Parser*

Esta clase se encarga de realizar el análisis sintáctico de la estructura de los argumentos de los comandos recibidos [88] con objeto de determinar la estructura del comando. Se realiza aplicando diferentes técnicas para comprobar si los argumentos se corresponden con el tamaño, tipo, orden y forma especificada.

Estas técnicas se dividen principalmente en dos, conocidas como analizador sintáctico descendente (*Top-down parsing)* y analizador sintáctico ascendente (*Bottom-up parsing)* [89].

El analizador sintáctico descendente es el analizador que genera un análisis sintáctico para la cadena de entrada dada y genera un árbol de análisis sintáctico para esa cadena de entrada utilizando la derivación más a la izquierda (Figura 56).

Además, el analizador sintáctico descendente se clasifica en dos tipos: un analizador sintáctico de descenso recursivo, y un analizador sintáctico de descenso no recursivo.

El analizador descendente recursivo también se conoce como analizador de fuerza bruta o analizador de seguimiento. Básicamente genera el árbol de análisis sintáctico utilizando la fuerza bruta y el retroceso.

El analizador descendente no recursivo también se conoce como analizador LL(1) o analizador sin retroceso. Utiliza una tabla de análisis sintáctico para generar el árbol de análisis sintáctico en lugar de hacer retroceso.

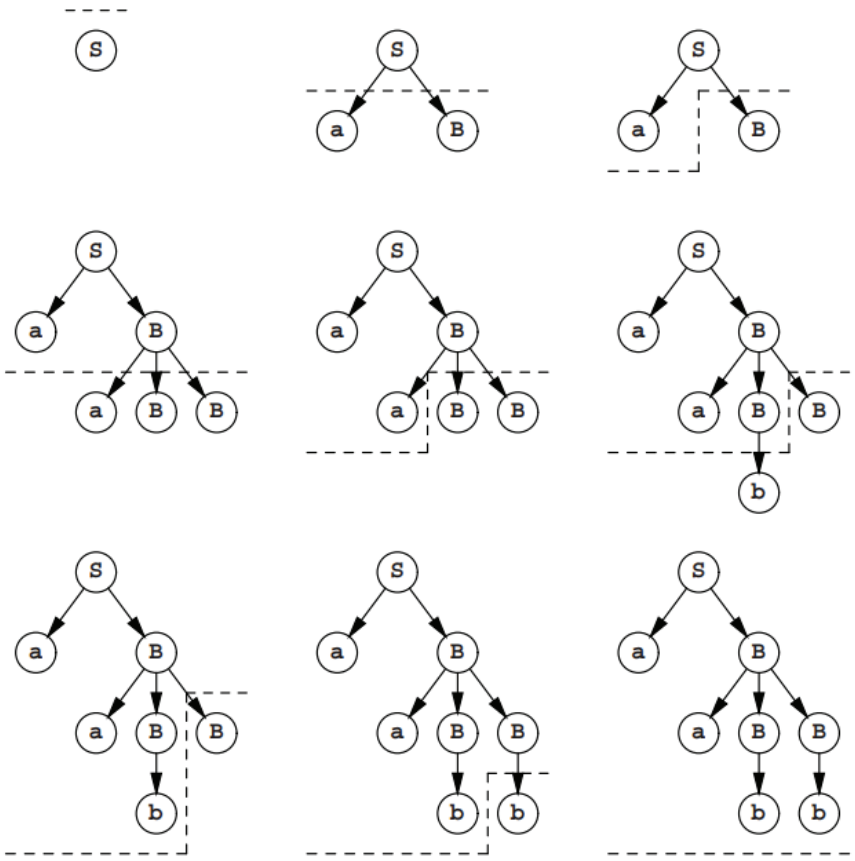



Figura 56. Ejemplo de generación de árboles mediante análisis sintáctico descendente [89]

Por otro lado, el analizador sintáctico ascendente construye el árbol de análisis sintáctico de abajo a arriba. Tiene un uso más general, siendo igual de eficaz y basándose en los principios del análisis sintáctico descendente.

Este se clasifica en dos tipos: el analizador LR y el analizador de precedencia de operadores.

El analizador LR es el analizador sintáctico ascendente que genera el árbol de análisis sintáctico para la cadena dada utilizando una gramática inequívoca.

El analizador sintáctico de precedencia de operadores solo es usado para generar el árbol de análisis sintáctico a partir de las gramáticas de operadores (+, -, * ...).

Basándonos en las técnicas anteriores se puede realizar la separación de los *string* personalizados por diseño que pueden contener diferentes valores según la configuración deseada, comprobar si un elemento es un número, realizar conversiones de datos, etc.

Por ejemplo, con el siguiente código se comprueba si un *string* es un número:

```
/**
 * @brief Check if a string is a number.
 * @param number The string to check.
 * @return True if it is a number and False if it is not.
 */
bool Parser::CheckIfIsNumber(const std::string &number)
{
    std::string::const_iterator it = number.begin();
    while (it != number.end() && isdigit(*it)) ++it;
    return !number.empty() && it == number.end();
}
```

Listado 8. Análisis de datos en la entrada de opciones

*7.3.1.6. GPIOS*

Mediante esta clase se realiza la comunicación con los bloques de entrada/salida de propósito general (GPIO) del sistema usando las clases *libgpio* [90] y *libuio* [91], creadas para utilizar fundamentalmente en tándem con el *driver* UIO (*Userspace I/O system*) [92] y los AXI GPIO de Xilinx [93], [94].

El *driver* UIO nace con la idea de evitar crear un *driver kernel* de Linux para cada dispositivo en el ámbito industrial/profesional para manejar interrupciones o acceder al espacio de memoria del dispositivo. Para esto se necesita un módulo simplificado en el *kernel* de Linux. La parte principal del *driver* se ejecuta en el espacio de usuario, simplificando el desarrollo y reduciendo el riesgo de errores graves dentro del *kernel*. De este modo cada dispositivo es accesible desde un archivo de dispositivo y varios archivos de atributos *sysfs* [95].

El archivo del dispositivo se puede encontrar en `/dev/uioX` [96], y se puede utilizar para acceder al espacio de direcciones del dispositivo mediante un mapeo virtual de los registros o las ubicaciones de la RAM con la función `mmap()` [97].

Por otro lado, también se pueden leer el estado de sus interrupciones desde el mismo fichero, haciendo uso de la función `read()` o utilizar la función `select()` en espera de una interrupción, o la función `write()` cuando se tratada de un dispositivo que tiene múltiples interrupciones.

Cada dispositivo UIO puede tener una o más regiones de memoria disponibles para el mapeado de memoria. Esto es necesario porque algunas tarjetas de E/S industriales requieren acceso a más de una región de memoria [98] en un *driver*.

Cada mapa tiene su propio directorio en *sysfs*. El primer mapa aparece como `/sys/class/uio/uioX/maps/map0/`. Los mapas posteriores crean directorios `map1/`, `map2/`, y así sucesivamente. Estos directorios sólo aparecerán si el tamaño del mapa no es nulo.

Por otro lado, cada directorio `mapX/` contiene cuatro archivos de sólo lectura que muestran los atributos de la memoria:

- *name:* un identificador para el mapa, aunque es opcional y puede estar vacío, los *drivers* pueden establecerlo para que puedan ser identificados en el espacio de usuario.
- *addr:* la dirección de la memoria que puede ser mapeada.
- *size:* el tamaño, en bytes, de la memoria apuntada por *addr.*
- *offset:* el desplazamiento, en bytes, que hay que añadir al puntero devuelto por `mmap()` para llegar a la memoria real del dispositivo. Esto es importante si la memoria del dispositivo no está alineada con la página [99], por lo que es usual añadir un *offset.*

Hay que tener presente, que este *driver* UIO no actúa como un controlador universal pues ese no es su fin. Los dispositivos que ya son manejados por otros subsistemas del *kernel,* como las redes o USB, no serán manejados de esta forma. Los dispositivos que pueden ser controlador por estos *drivers* deben cumplir una serie de requisitos:

1. Que disponga de memoria que pueda ser mapeada, pudiendo ser controlado completamente escribiendo en esa memoria.
2. El dispositivo puede generar interrupciones.
3. El dispositivo no tiene ningún subsistema del *kernel* donde encaje.

Una función de esta clase es la que se muestra en el Listado 9, donde se muestra cómo se escriben datos en el GPIO que monitoriza la fuente de alimentación de 5.4V.

```
/**
 * @brief Function used to write on the UpsPower 5.4 gpio.
 * @param dataWrite Is the data to be written to the gpio (0 or 1).
 */
void GPIOS::GpioUpsPower54(int dataWrite)
{
    GPIO vm = GPIO_init(GPIO_UPS_0_UIO, MAP_0);
    setPinMode(vm, CHANNEL_2, PIN_NUMBER_0 + 1, OUTPUT);
    digitalWrite(vm, CHANNEL_2, PIN_NUMBER_0 + 1, dataWrite);
    GPIO_Close(vm);
}
```

Listado 9. Función `GpioUpsPower54` para la gestión de la fuente de alimentación de 5.4 V.

Se observa que mediante el uso de las clases mencionadas anteriormente *libgpio* y *libuio* se procede a inicializar el *hardware* facilitándole el número del UIO declarado en el sistema y el número del mapa, el cual en este proyecto solo tienen uno el cual es el `map0`. Posteriormente, se establece el pin 0 del canal 2 como salida y a continuación se escribe en este el estado al que quiere ponerse el pin, a nivel alto (1) o bajo (0). Por último, se cierra el GPIO y se libera toda la memoria asociada a este.

#### *7.3.1.7. Qspi*

En esta clase se maneja la comunicación con la interfaz Quad-SPI [100] haciendo uso de los registros definidos en el fichero `xparameters.h` de la aplicación y mediante la clase *devmem* [101]. Recoge todas las funciones necesarias para la comunicación con esta, para la configuración de todos los bloques de la plataforma conectados a este, así como al bloque específico del conversor AD7768.

El *header-only* `xparameters.h` contiene los parámetros de la plataforma desarrollada, albergando la dirección de cada dispositivo del sistema, así como los parámetros y el mapa de memoria de cada uno. Esto permite que todas las funciones necesarias para comunicarse con el Quad-SPI pueden leer las direcciones establecidas por el diseño directamente desde este archivo, el cual cada vez que se genera una plataforma nueva y se realizan cambios en el diseño debe ser sustituido pues pueden producirse cambios en las direcciones y en general en el contenido del fichero.

Por otro lado, la clase *devmem* [102]–[104] permite que las direcciones físicas puedan ser mapeadas en direcciones virtuales mediante `mmap()` en el espacio de usuario facilitando su lectura y escritura. Esta clase está basada en el conjunto de herramientas que se encuentran en Linux para el mismo fin.

Esta es una de las clases de más bajo nivel, la cual provee acceso a la memoria física del sistema. Permite a la clase Quad-SPI, entre otras, leer y escribir las direcciones de memoria requeridas para las diferentes operaciones, así como poder interactuar con las E/S del sistema.

#### *7.3.1.8. ReadSystem*

Esta clase permite la lectura del estado del *hardware* para la creación del fichero de estado `board-config.json` usando la clase *ConfigFile*.

Esta última contiene las reglas necesarias para interpretar los datos leídos por *ReadSystem* y crear el fichero de estado. Para ello, se hace uso de la biblioteca de *software* libre Boost 1.71v usando la librería *Property Tree* [105] que proporciona una estructura de datos que almacena un árbol de valores anidados, indexado en cada nivel por una clave. Esta librería proporciona analizadores y generadores para una serie de formatos que pueden ser representados por un árbol de este tipo como son XML, INI y JSON. Estos árboles son estructuras de datos versátiles y adecuados especialmente para contener datos de configuración.

Esta, como otras librerías de la familia Boost, es una de las más utilizadas cuando se pretenden crear ficheros de configuración o estado como en este caso. Existen amplia documentación [106]–[108] en la que apoyarse para crear configuraciones que permitan guardar la información deseada con éxito. En este proyecto, se ha elegido el formato JSON [109] para almacenar el estado del sistema, manteniendo un formato jerárquico y estructurado en clave-valor.

Esta clase hace uso de la clase *devmem*, mencionada anteriormente, para leer el estado de cada uno de los registros que proporcionan la información necesaria para conocer el estado del sistema: modo de funcionamiento, canales activos, configuración de filtro, etc. Toda la información recogida de los registros se envía a la clase *ConfigFile*, donde

se tratan los datos recogidos y se crean o modifican los registros necesarios en el fichero de estado.

Un ejemplo se muestra en el Listado 10. En este caso se presenta la función usada para leer el modo de configuración en el que se encuentra el dispositivo ADC.

```
/**
 * @brief Function used to read status of Pin/Spi mode.
 */
void ReadSystem::ReadPinSpiMode()
{
    int value = SpiPinMode_read((IP_PINMODECONTROL_BASEADDR +
                PINMODECONTROL_REGISTER_OFFSET));
    int state = value & 0x1;

    configFile.PinSpiMode(state);
}
```

Listado 10. Lectura del modo de configuración (PIN/SPI) del sistema

Con ello, se consigue leer el estado del pin que controla el modo de funcionamiento, SPI o PIN, obteniendo un valor de 1 o 0 respectivamente. A continuación, el valor obtenido se envía a la clase *ConfigFile*, donde se analiza su valor y se escribe en el árbol de configuración que mantiene el estado del sistema (Listado 11).

```
/**
 * @brief Set PIN mode as active.
 */
void ConfigFile::PinSpiMode(int state)
{
    // 0 - PINMODE
    // 1 - SPIMODE
    if(state == 0)
    {
        configFilePtree.put(PINMODE dot STATUS, ENABLE);
        configFilePtree.put(SPIMODE dot STATUS, DISABLE);
    }

    if(state == 1)
    {
        configFilePtree.put(SPIMODE dot STATUS, ENABLE);
        configFilePtree.put(PINMODE dot STATUS, DISABLE);
    }
}
```

Listado 11. Tratamiento de los valores obtenidos del estado del sistema (PIN/SPI mode)

En esta función, se escribe el registro que proceda en el fichero de estado, estableciendo como *enable* o *disable* el modo que corresponda.

Para que este proceso de modificación de los registros del fichero de estado sea posible, previamente en la clase *Menu* se ha realizado una lectura del fichero mediante el método `ReadConfigFile()` de la clase *ConfigFile* (Listado 12), y al realizar todos los cambios finalmente su escritura con la función de la misma clase `WriteConfigFile()` (Listado 13).

```
/**
 * @brief Reads the CONFIG_FILE file using the Property Tree library.
 */
void ConfigFile::ReadConfigFile(std::string filename)
{
    boost::property_tree::read_json(filename, configFilePtree);
}
```

Listado 12. Lectura del fichero del estado de configuración

```
/**
 * @brief Write/Create the CONFIG_FILE file using the Property Tree
 *        library.
 */
void ConfigFile::WriteConfigFile(std::string filename)
{
    boost::property_tree::write_json(filename, configFilePtree);
}
```

Listado 13. Escritura del fichero del estado de configuración

Este es el procedimiento que se debe realizar según la documentación aportada por la librería para modificar los datos necesarios. Existen otras librerías para el mismo fin [110], sin embargo, se ha optado por esta librería debido a que en el resto del proyecto se usa la familia Boost y se puede mantener todo el proyecto bajo un mismo entorno.

#### *7.3.1.9. PmodRTCC*

Esta librería maneja la comunicación con el Pmod RTCC (*Real-Time Clock Calendar*) del sistema permitiendo la lectura y escritura de diferentes parámetros del dispositivo mediante la clase *I2C*.

La clase *I2C* facilita la lectura de los datos del RTCC a través del protocolo del mismo nombre [111]. Este protocolo utilizado en la industria facilita la comunicación con sensores digitales, ya que su arquitectura permite tener una confirmación de los datos recibidos, dentro de la misma trama, disminuyendo el número de pines necesarios. Además, permite la conexión de diferentes dispositivos al mismo bus. Por otro lado, incluye más bits en su trama de comunicación, lo que permite enviar mensajes más completos y detallados.

En la clase anterior, se encuentra una serie de funciones que son imprescindibles para que la clase *PmodRTCC* pueda funcionar correctamente. En primer lugar el manejo de la inicialización del dispositivo mediante la función `I2C_start()` (Listado 14) y su parada con la función `I2C_stop()` (Listado 15).

Por otro lado, se encuentran funciones que permiten la lectura y escritura de los registros del dispositivo. Así, se muestran a continuación cómo se inicia y se para el dispositivo con el que se pretenda interactuar.

```
/**
 * @brief Start the I2C device.
 * @param dev points to the I2C device to be started, must have
 *        filename and addr populated
 * @return - 0 if the starting procedure succeeded
 *         - negative if the starting procedure failed
 */
int I2C::I2C_start(I2C::I2cDevice *dev)
{
    int fd;
    int rc;

    //Open the given I2C bus filename.
    fd = open(dev->filename, O_RDWR);
    if (fd < 0)
    {
        rc = fd;
        return rc;
    }

    //Set the given I2C slave address.
    rc = ioctl(fd, I2C_SLAVE, dev->addr);
    if (rc < 0)
    {
        close(fd);
        return 0;
    }

    dev->fd = fd;

    return 0;
}
```

Listado 14. Arranque del dispositivo RTCC

```
/**
 * @brief Stop the I2C device.
 * @param dev points to the I2C device to be stopped
 */
void I2C::I2C_stop(I2C::I2cDevice *dev)
{
    //Close the I2C bus file descriptor.
    close(dev->fd);
}
```

Listado 15. Parada del dispositivo RTCC

Se hacen uso de funciones propias del sistema Linux como son `open()` [112] y `close()` [113] para realizar las respectivas funciones. Con estas dos funciones, puede realizarse diferentes estrategias de comunicación con el dispositivo que se desee.

Para este proyecto en concreto, se han diseñado una serie de funciones que proporcionen información útil: comprobar la batería, establecer la fecha, establecer la hora, etc. Para realizar el primer caso, que determina el estado de la batería del dispositivo, en primer lugar, se debe iniciar el dispositivo I2C con la función comentada anteriormente. En segundo lugar, se puede consultar el estado de la batería con la función mostrada en el Listado 16.

```
/** u8 RTCC_checkVbat(PmodRTCC *InstancePtr)
 *
 *   Parameters:
 *      InstancePtr - PmodRTCC device to use
 *
 *   Return Value:
 *      data - 0 or 1, the state of VBATEN
 *
 *   Description:
 *      This function returns the state of the VBATEN control bit,
 *      which determines where the RTCC draws it's power from.
 */
void PmodRTCC::RTCC_checkVbat(struct I2C::I2cDevice *dev, uint8_t reg,
uint8_t *data, int nData)
{
    i2c.I2C_readn_reg(dev, reg, data, nData);
}
```

Listado 16. Determinación del estado de la batería del dispositivo RTCC.

Según el resultado que arroje esta función, se puede determinar si el dispositivo presenta un valor aceptable del estado de la batería (1) o no (0). Una vez finalizado todos los mecanismos que se deseen, se debe hacer una llamada a la función que detiene el dispositivo para evitar bloquearlo. Todas estas funciones están basadas en el *datasheet* del dispositivo [114].

*7.3.1.10. Date*

Librería *header-only* [115] y de código abierto, disponible para C++11, C++14 y C++17, está basada en *chrono* y añade funcionalidades y parámetros extra para el manejo y obtención de marcas de tiempo.

Se utiliza para tomar la referencia necesaria para introducir las marcas de tiempo en las muestras capturadas con la mayor precisión posible, insertando marcas de tiempo con una precisión de milisegundos, lo que permite catalogar cada muestra de manera inequívoca.

## 7.3.2. OPCIONES DE USUARIO

En este apartado se explican las opciones disponibles para el usuario que se han implementado en la aplicación y que son servidas por el *Menu*. Todas estas opciones están accesibles desde la aplicación cliente como se pueden apreciar en los siguientes apartados.

### *7.3.2.1. Opciones de modo pin*

El ADC AD7768 presenta dos modos de funcionamiento: modo PIN y modo SPI, proporcionando el modo SPI un control más detallado de su funcionalidad. Para acceder al modo PIN es necesario incluir la opción `-P (--pin_mode)` en la línea de comandos.

En el modo PIN podemos poner en *standby* o en modo captura un conjunto de canales, establecer la decimación, definir el filtro a utilizar y establecer los modos de funcionamiento.

```
-P [ --pin_mode ]       Set PIN mode.
                        Available options:

                        -s [ --standby ] arg
                            Set standby channels.
                            The rest will remains 'active'.
                            Options:
                               1   -> CH0  to CH3
                               2   -> CH4  to CH7
                               3   -> CH8  to CH11
                               4   -> CH12 to CH15
                               5   -> CH0  to CH7
                               6   -> CH8  to CH15
                               7   -> CH0  to CH11
                               8   -> CH4  to CH15
                               9   -> Standby all channels
                               10  -> (Reset) All channels operative
                            Enter as: {option}

                            -d [ --decimation ] arg
                            Define decimation rates.
                            The chosen decimation rate is used
                            on all ADC channels.
                            ADC options:
                               0   -> ADC0: CH0 to CH7
                               1   -> ADC1: CH8 to CH15
```

```
                                Decimation options:
                                   32, 64, 128 or 1024
                                Enter as: {ADC}:{decimation}
                                Example:  1:64

                                -f [ --filter ] arg

                                Set filter.
                                Options:
                                   0  -> 'Wideband'
                                   1  -> 'Sinc5'
                                Enter as: {option}
                                -x [ --modex ] arg
                                Set mode (Power Mode and DCLK Frequency).
                                Power Mode:
                                   L  -> 'Low power'
                                   M  -> 'Median'
                                   F  -> 'Fast'
                                DCLK Frequency:
                                   1  -> MCLK/1
                                   2  -> MCLK/2
                                   4  -> MCLK/4
                                   8  -> MCLK/8
                                Enter as: {Power Mode}:{DCLK}
                                Example: L:2
                                Note: Only Standard Data Conversion
                                Mode available.
```

Listado 17. Opciones de configuración en Modo PIN

*7.3.2.2. Opciones del modo SPI*

En modo SPI se realiza un control completo del ADC, ya que es posible definir la decimación de cada ADC, activar o poner canales en *standby* de forma individual o en un determinado rango, definir los modos de los canales o de un rango de canales, definir la ganancia interna del ADC, definir los *offsets* de los canales de forma individual o ejecutar un ajuste automático y, por último, activar o desactivar el CRC en los canales. La numeración de los canales se ajusta a la placa usada.

```
-S [ --spi_mode ]     Set SPI mode.
                       Available options:

                        -d [ --decimation ] arg
                          Define decimation rates.
                          The chosen decimation rate is used on all
                          ADC channels.
                          ADC options:
                              0   -> ADC0: CH0 to CH7
                              1   -> ADC1: CH8 to CH15
                          Mode:
                              A or B
                          Decimation options:
```

```
        32, 64, 128, 256, 512 or 1024
    Enter as:
        {ADC}:{mode}:{decimation}
    Example:
        1:A:64

-a [ --active ] arg
    Set active channels.
    Single channel or a range between CH0 to CH15.
    Enter as: {channel}
    Example:  3
              4..11

-s [ --standby ] arg
    Select standby channels.
    Single channel or a range between CH0 to CH15.
    Enter as: {channel}
    Example:  3        Standby CH3
              4..11    Standby CH4 to CH11

-m [ --chmode ] arg
    Define channels mode.
    Single channel or a range between 0-15
    and mode A or B.
    Enter as: {channel}:{mode}
    Example: 3:A        CH3 in A mode.
             4..11:B    CH4 to CH11 in B mode.

-g [ --chgain ] arg
    Define channels ADC gains.
    Single channel or a range between 0-15.
    Enter as:
       {24 bits hexadecimal}:{channel}
       {0xABC5DE}:{0}
    Example:
      0x555555:3     Gain 0x555555 to CH3
      0x666666:4..11 Gain 0x666666 in CH4 to CH11

-o [ --choffset ] arg
    Define channels offset.
    Single channel or range between 0-15.
    Enter as:
       {sign}{offset}{unit}:{channel}
       {/-}{0.000}{/m/u}:{0}
    Example:
       -1.001m:3    Offset -1.001mV in CH3
       0.100:4..11  Offset 0.100V in CH4 to CH11

-u [ --autooffset ] arg
    Run the autotuning of the channels offset
    in both ADC's.

-c [ --crc ] arg
    Define CRC state and number of
    samples (4/16).
    CRC options:
       0   -> CRC disable.
       1   -> CRC enable.
    Samples options:
```

```
                                4   -> Every 4 samples.
                                16  -> Every 16 samples.
                             Enter as:
                                {option}:{samples}
                             Example:
                                0       CRC disable.
                                1:16    CRC enable and every 16 samples.
```

Listado 18. Opciones de configuración en Modo SPI

*7.3.2.3. Ganancias del amplificador*

Otra de las opciones de mayor uso de la aplicación es el ajuste de ganancias del amplificador. En este caso se pueden definir ganancias en grupo de dos canales o para todos los canales.

```
-G [ --gain ] arg      Select channels to change amplifier gain.
                        Available options:
                           1   -> CH0  to CH1
                           2   -> CH2  to CH3
                           3   -> CH4  to CH5
                           4   -> CH6  to CH7
                           5   -> CH8  to CH9
                           6   -> CH10  to CH11
                           7   -> CH12  to CH13
                           8   -> CH14  to CH15
                           9   -> All channels
                        Gain values:
                           1   (0 dB)  Analog Input +/- 5.0V.
                           50  (34 dB) Analog Input +/- 100mV.
                           100 (40 dB) Analog Input +/- 50mV.
                           200 (46 dB) Analog Input +/- 25mV.
                           300 (50 dB) Analog Input +/- 16.6mV.
                        Enter as:
                           {option number}:{gain}
                        Example:
                           1:100   Gain 100 in CH0 and CH1
                           7:1     Gain 1 in CH12 and CH13
```

Listado 19. Opciones de configuración de Ganancias

*7.3.2.4. Configuración de promedios*

Permite configurar el cálculo de promedios de las muestras en potencias de 2, en el rango desde 1 hasta 1024 muestras. Ello permite calcular el promedio de muestras en función del parámetro indicado.

```
-A [ --average ] arg   Set the number of samples to be averaged.
                       Available option:
                          1     -> Bypass
                          2     -> Every 2 samples
                          4     -> Every 4 samples
```

```
                              8     -> Every 8 samples
                              16    -> Every 16 samples
                              32    -> Every 32 samples
                              64    -> Every 64 samples
                              128   -> Every 128 samples
                              256   -> Every 256 samples
                              512   -> Every 512 samples
                              1024  -> Every 1024 samples
                            Enter as: {option}
```

Listado 20. Opciones de configuración de promedios.

*7.3.2.5. Acceso a control de píxeles*

Permite escribir un valor digital (1/0) cada una de las señales presentes en el conector de píxel. Admite dos parámetros: el píxel y su valor.

```
  -X [ --pixel ] arg      Set pixel state.
                          Choose pixel from 1 to 4 and state Enable(1)
                          or Disable(0).
                          Enter as:
                             {pixel}:{state}
                          Example:
                             1:0     Pixel 1 disable
                             3:1     Pixel 3 enable
```

Listado 21. Control de píxel

*7.3.2.6. Capturar datos*

Mediante esta opción se habilita (1) o deshabilita (0) la captura de muestras.

```
-C [ --capture ] arg   Start/stop data capture.
                         Options:
                            0   -> Stop/Disable.
                            1   -> Start/Enable.
                         When capture is enabled, the available commands
                         are: help, status, status-update,
                         reboot, shutdown and capture(0).
```

Listado 22. Opciones para la captura de datos

*7.3.2.7. Arranque o parada del módulo de adquisición*

Esta opción permite el arranque completo y la parada del módulo de adquisición de datos. Cuando el sistema arranca se configura con las opciones por defecto. Esta opción está únicamente disponible para las tarjetas maestras de cada equipo (identificadas como 0, 2, 4, … en el ID de la aplicación `dspace-client`).

```
  -M [ --module ] arg     Start up or shutdown module.
                          Options:
                             0   -> Shut down.
```

```
                              1   -> Start up.
                            Note: this option can only be used for the
                                  following boards: 0, 2, 4 ...
```

Listado 23. Opciones de arranque y parada del sistema de adquisición.

*7.3.2.8. Ayuda on-line*

Con esta opción, el cliente recibe la ayuda generada por el menú de opciones para saber cómo se usa la aplicación. Donde recibe todos los parámetros que se describen en este apartado (Listado 24).

```
  -h [ --help ]          Help.
```

Listado 24. Opciones de ayuda del programa

*7.3.2.9. Estado y control del sistema*

Las siguientes opciones (Listado 25) permiten conocer el estado del sistema, ya sea el registrado en el fichero de estado (Figura 57) como la lectura de los registros del sistema, la actualización de los registros de estado y su visualización. Igualmente permite el rearranque del sistema Linux o su parada completa.

De especial relevancia es la opción `--sync`. Esta opción permite enviar los pulsos de sincronización necesarios a los ADC para que puedan almacenar los cambios de su configuración. Por tanto, debe enviarse una orden desde el cliente con la opción `--sync` cada vez que se modifiquen los parámetros de los ADC para que estos surtan efecto.

```
System commands:
 --sync           Synchronize system.
                  Note: this option can only be used for the MASTER board.
                  WARNING! This needs to be launched after each
                  system configuration.

 --status         Print state of the system from current state file.
                  WARNING! Each board has its own state file.

 --status-update  Read state from hardware and print current state file.
                  WARNING! Each board has its own state file.

 --reboot         Reboot system.
                  WARNING! Reboot only the selected board.

 --shutdown       Shut down system.
                  WARNING! Shut down only the selected board.
```

Listado 25. Opciones de sistema

```
deepspace@vlsiws20:~/nfs$ ./dspace-client 0 --status
{
    "Board number": "0",
    "Capture": "Disable",
    "Pin Mode": {
        "Status": "Enable",
        "Filter": "Wideband",
        "Mode": "Fast MCLK|4",
        "Standby Channels": {
            "Channel 0": "Active",
            "Channel 1": "Active",
            "Channel 2": "Active",
            "Channel 3": "Active",
            "Channel 4": "Active",
            "Channel 5": "Active",
            "Channel 6": "Active",
            "Channel 7": "Active",
            "Channel 8": "Active",
            "Channel 9": "Active",
            "Channel 10": "Active",
            "Channel 11": "Active",
            "Channel 12": "Active",
            "Channel 13": "Active",
```

Figura 57. Estado de configuración del sistema usando el comando *status*

## 7.4. DESCRIPCIÓN DE LA APLICACIÓN 'DSPACE-ACFAIL'

La fuente de alimentación del equipo (Figura 9) dispone de una reserva de energía para proteger su funcionamiento en caso de fallo de la alimentación. Para ello se dispone de un pin de entrada digital en la ZedBoard que es vigilado por una aplicación que se ejecuta en Linux.

La estrategia de vigilancia es configurable desde la aplicación. En la implementación actual, se lee el pin que vigila el estado de la alimentación cada 20 segundos. En el caso de detectar un fallo en la alimentación, se realiza una espera de 60 segundos en previsión de que se haya producido un fallo transitorio. Si persiste el fallo en el suministro se procede al apagado del equipo, siguiendo la siguiente secuencia: apagado del modo de captura de datos, espera de 1 segundo para apagar la fuente de 7,3 V y espera de 2 segundos para el apagado de la fuente de 5,4V. Por último, se realiza el apagado del sistema operativo. Si durante la espera inicial de 60 segundos, se restaura el suministro no se realiza el proceso de apagado y se vuelve al estado de vigilancia. Este flujo puede verse en la Figura 58 y en el Listado 26 se muestra a un extracto de la aplicación que soporta este mecanismo.

```
while(true)
    {
        uint acfail_read = acfailCheck.AcfailRead();

        if(acfail_read)
        {
            printf("ACFail detected! Wait 1 min for AC power
                   recovery.\n");
            sleep(60);
            //check again acafail, maybe for AC recovery
            acfail_read = acfailCheck.AcfailRead();

            if(acfail_read)
            {
                printf("ACFail detected again! Shut down system.\n");
                // Stop capture (FLAG_CAPTURE_ADDR flag to 0)
                acfailCheck.StopCapture();
                //wait 1 s
                sleep(1);
                //shutdown PS 7.3V, wait 2 s and shutdown PS 5.4V
                acfailCheck.ModuleShutDown();
                //shutdown linux system
                sync();
                reboot(LINUX_REBOOT_CMD_POWER_OFF);
            }
            else
            {
                printf("ACFail has not been detected again.
                       The AC power is back on!.\n");
            }
        }

        sleep(20);
    }
```

Listado 26. Extracto del código que realiza la vigilancia del estado de la fuente de alimentación

La aplicación `dspace-acfail` (o simplemente `acfail`) se activa durante el proceso de arranque de la ZedBoard y permanece ejecutándose en *background*.

Para esta aplicación también se ha hecho uso del acceso a memoria mediante herramientas *devmem,* además del uso de los *drivers* UIO para poder acceder a los pines GPIO correspondientes relacionados con la fuente de alimentación del equipo. A su vez se han utilizado librerías propias de Linux [116] para controlar las opciones de sincronización, para evitar pérdidas de datos y de apagado controlado del equipo. Para esta aplicación, se ha utilizado el patrón de diseño Observer [117], el cual se basa en la suscripción para notificar eventos, modelo de cómputo utilizado en esta aplicación.

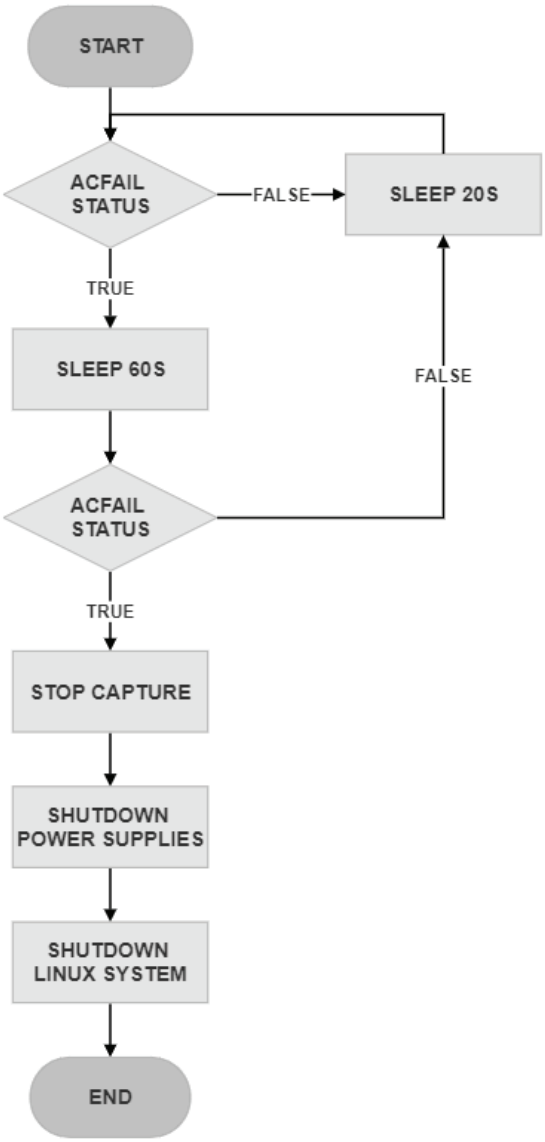



Figura 58. Diagrama de flujo de la aplicación Dspace-acfail

## 7.5. DESCRIPCIÓN DE LA APLICACIÓN 'DSPACE-CLIENTE'

La aplicación `dspace-client` es la que utiliza el usuario para interaccionar con el sistema (servidor). Se trata de una aplicación que se conecta al servidor por los puertos TCP:8092 (tarjeta máster) y TCP:8093 (tarjeta esclava). Esta aplicación está desarrollada en C++ y se comporta de forma transparente con respecto al servidor, ya que es una aplicación externa a este.

El objetivo de dicha aplicación ha sido la simplicidad, es decir, utilizar librerías propias de Linux para el uso de *sockets* [118], [119] y otros elementos o evitar en lo posible el uso de librerías con la intención de que esta aplicación pueda ser ejecutada desde cualquier distribución Linux sin importar sus características. Por ello, todo el código es perfectamente abarcable en el fichero `main` sin necesidad de crear una estructura de datos o utilizar un patrón de diseño. A su vez, no son necesarios el uso de *drivers* específicos, como los que se usan en otras aplicaciones referentes al *hardware* como las herramientas *devmem* o los *drivers* UIO, pues en este caso se realiza una conexión simple mediante TCP/IP.

Las opciones disponibles en esta aplicación son las siguientes:

- Identificador de tarjeta ZedBoard de captura: 0, 1, …
- IP de la tarjeta ZedBoard de destino, necesario si no está asignada en la dirección IP por defecto de la aplicación.

Esta aplicación está orientada a una interfaz CLI, de tal forma que es posible encadenar una secuencia de instrucciones usando un lenguaje de *scripting* (bash, Python, etc.) para automatizar tareas complejas sobre el equipo.

Las opciones disponibles se muestran con la orden `dspace-client -h` (Figura 59).

```
deepspace@vlsiws20:~/nfs$ ./dspace-client -h
Use:
./dspace-client [ID_board] [IP-commands or commands]

The boards available are:
Master  --> 0
Slave 1 --> 1
```

Figura 59. Opciones disponibles en el cliente

Si se indica la tarjeta a la que se desea conectar en la línea de comandos, entonces la opción `-h` proporcionará la ayuda del servidor utilizando los valores predefinidos en la aplicación de IP e identificador (Figura 60).

```
deepspace@vlsiws20:~/nfs$ ./dspace-client 0 -h

Usage:

./[client_name] [ID_board] [IP-commands or commands]

WARNING! The number of channels changes depending on the selected boa
rd.
WARNING! Checks if the RTC time is correct.

Allowed options:

  -P [ --pin_mode ]      Set PIN mode.
                         Available options:

                           -s [ --standby ] arg
                             Set standby channels.
                             The rest will remains 'active'.
                             Options:
                                1   -> CH0  to CH3
                                2   -> CH4  to CH7
                                3   -> CH8  to CH11
```

Figura 60. Opciones específicas para una tarjeta

Los identificadores de tarjeta resultan de la combinación de IP y puerto que están codificados en el código fuente para simplificar su utilización (Listado 27).

```
#define SERVER_IP_MASTER_ZEDBOARD_0 "192.168.1.5" /* server IP MASTER*/
#define SERVER_IP_SLAVE_ZEDBOARD_1 "192.168.1.8"  /*server IP SLAVE */
#define PORT_MASTER_ZEDBOARD_0    8092          /* server PORT MASTER */
#define PORT_SLAVE_ZEDBOARD_1     8093           /* server PORT SLAVE*/
```

Listado 27. Configuración de las tarjetas disponibles por defecto

El cliente admite proporcionar la IP en la línea de comandos, aunque si se modifican los valores por defecto no es necesario proporcionar dicha opción (Figura 61).

Tanto si se introduce una IP diferente, como si se usa la que está establecida por defecto, se realiza una comprobación de la conexión, es decir, que esta pueda ser establecida correctamente y los datos se puedan enviar. Si esta no puede llevarse a cabo, la aplicación genera un error. Las IP que se introduzcan manualmente, pasarán por un proceso de verificación para comprobar que son IP válidas y podrán ser usadas para establecer una conexión. Para ello se usa el código mostrado en el Listado 28.

```
deepspace@vlsiws20:~/nfs$ ./dspace-client 0 10.13.24.198 -h

Usage:

./[client_name] [ID_board] [IP-commands or commands]

WARNING! The number of channels changes depending on the selected boa
rd.
WARNING! Checks if the RTC time is correct.

Allowed options:

  -P [ --pin_mode ]      Set PIN mode.
                         Available options:

                           -s [ --standby ] arg
                             Set standby channels.
                             The rest will remains 'active'.
                             Options:
                                1   -> CH0  to CH3
                                2   -> CH4  to CH7
                                3   -> CH8  to CH11
```

Figura 61. Especificación de valores de conexión en el cliente

```
/**
 * @brief Check if an IP is valid.
 * @param ipAddress the IP introduced by the user.
 */
bool ValidateIpAddress(const std::string &ipAddress)
{
    struct sockaddr_in sa{};
    int result = inet_pton(AF_INET, ipAddress.c_str(), &(sa.sin_addr));
    return result != 0;
}
```

Listado 28. Verificación de IPs en el cliente

Para la programación de los *sockets* se ha hecho uso de funciones C/C++ comentadas anteriormente. Aunque se podría haber usado la librería de Boost Asio [120], se ha dado prioridad a la simplicidad como norma.

Para compilar esta aplicación, es necesario un sistema Linux preferiblemente, se usa la orden `g++ dspace-client.cpp -o dspace-client` con la versión 7.5 (o superior) del compilador GNU g++ [121].

## 7.6. DESCRIPCIÓN DE LA APLICACIÓN 'DSPACE-CONVERTER'

Se trata de una aplicación *off-line,* externa al sistema, que convierte los ficheros de datos capturados desde un formato binario a un formato legible por el usuario. Se dispone de una aplicación con funcionalidad básica que sirve como ejemplo para el desarrollo de

programas más complejos en función de las necesidades específicas del usuario. Está escrita en C++ y funciona en Linux Ubuntu.

El formato del fichero es el mostrado en la Figura 62. Se trata de líneas de 128 bits que incluye el valor del dato obtenido, la identificación de la tarjeta y del canal y el *timestamp* en que el dato ha sido capturado. El dato digitalizado se almacena en los bytes 12, 14 y 15. En el byte 13 se incluye la información del canal (4 bits menos significativos) y de la placa (4 bits más significativos). Los siguientes 8 bytes almacenan la información de tiempo real de la muestra en orden inverso y los últimos 4 bytes están reservados para información interna del equipo, sin relevancia para los datos.

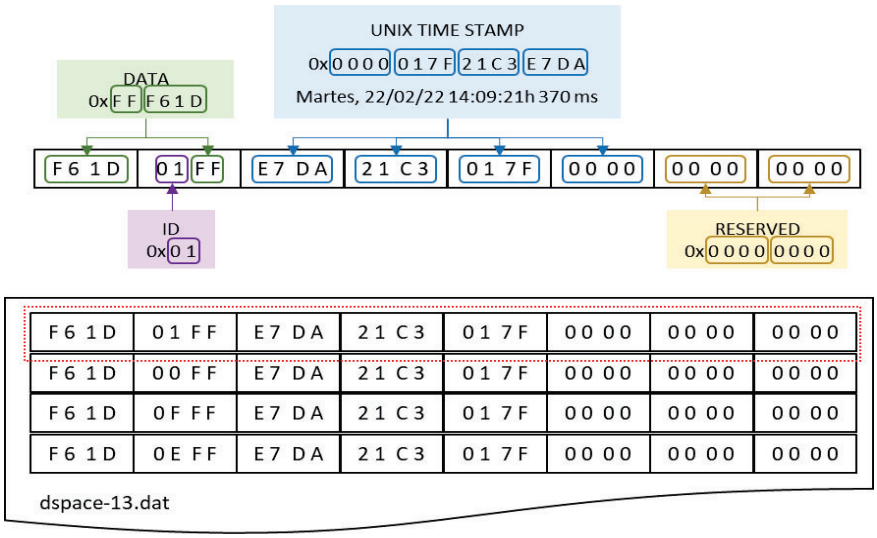



Figura 62. Formato del fichero binario para el almacenamiento de tramas

La utilización del conversor se realiza desde una interfaz CLI (Figura 63). En el caso de que no se pase argumento enviará un código de error y un ejemplo de uso (Figura 64). Un ejemplo de la salida del programa se muestra en Figura 65, que indica, de izquierda a derecha: la línea del fichero, el canal de captura, el *timestamp* con resolución de milisegundos y el valor del dato en µV

```
# dpsace-converter <fichero de datos binario a convertir>
```

Figura 63. Utilización de la orden dspace-converter

```
# dpsace-converter
Se ha producido un error.
Modo de uso: dpace-converter nombre_del_fichero_binario
Ejemplo:  dspace-converter dspace99.dat
```

Figura 64. Mensajes de error de la aplicación dspace-converter

```
...
L[255]  CH[01]  TS[Fri 2022-Feb-18 10:11:39 906]  D[2684 uV]
L[256]  CH[02]  TS[Fri 2022-Feb-18 10:11:38 906]  D[-1425 uV]
...
```

Figura 65. Ejemplo del formato del fichero de salida de dspace-converter

El código puede ser modificado para que el usuario pueda adaptar su funcionalidad y sus unidades de salida, y se puede compilar con el compilador g++ de GNU sobre Linux Ubuntu con la orden mostrada en Figura 66.

```
g++ -std=c++11 -O3 -Wall -pedantic -o dspace-converter dspace-
        converter.cpp
```

Figura 66. Orden para la compilación de dpsace-converter

## 7.7. CONCLUSIONES

En este capítulo se ha detallado las aplicaciones desarrolladas en este proyecto, así como su funcionamiento. Por otro lado, también se ha mostrado como se configura el modo de funcionamiento del sistema y como se tratan los datos capturados para su interpretación.

# CAPÍTULO 8. GESTIÓN DE LOS DATOS CAPTURADOS

## 8.1. INTRODUCCIÓN

La instalación de Linux en el sistema permite utilizar un conjunto de recursos para la transferencia de ficheros de datos hacia los servidores de almacenamiento desde el equipo de captura. Al tratarse de un equipo dedicado a investigación se ha optado por dejar este aspecto lo más abierto posible de tal forma que se adapta a la situación concreta de utilización del equipo.

Los esquemas de transferencia de datos que se proponen tratan de utilizar clientes ligeros dando mayor peso al servidor de datos, reservando la capacidad de procesamiento de la ZedBoard, el uso de recursos de memoria y ancho de banda.

## 8.2. ALMACENAMIENTO DE DATOS

La transmisión de los datos procesados por la plataforma se envía a través de la interfaz Ethernet que soporta tasas de transmisión de 1 Gb/s. Hay que tener en cuenta que este dato debe tomarse sobre la trama Ethernet para una interfaz en modo *raw*.

Para el almacenamiento de los datos capturados se ha optado por enviarlos directamente a un servidor de almacenamiento mediante el protocolo NFS. La decisión está justificada debido a que el sistema genera una cantidad importante de datos capturados, especialmente cuando se trabaja muestreando a 256 KSPS durante periodos largos. La solución basada en NFS facilita el escalado cuando se instalan nuevos equipos de captura de datos. El esquema general de la solución se muestra en la Figura 67.

Los datos se almacenan en ficheros de 64 KBytes que incluye las muestras capturadas desde los 16 canales y se organizan por carpetas que incluye la información de fecha y hora de captura. Estas carpetas se crean durante el proceso de captura. Este modelo de gestión de datos diseñado facilita el procesamiento directamente en los servidores de la red a la que está conectado el equipo, pudiendo moverlos a otros equipos mediante mecanismos conocidos, tales como rsync [122] o SCP (*Secure Copy Protocol*) [123], SFTP *(Secure File Transfer Protocol*) [124] o similar. En ningún momento es necesario interactuar con el equipo de captura de datos para realizar estas tareas, lo que supone una capa de independencia y funcionalidad, escalando cada uno de los equipos para la función que se precise. Toda esta funcionalidad recae en el servidor de ficheros, y por tanto se pueden definir las políticas que sean más apropiadas.

Dado que el sistema operativo se carga en memoria DDR, no es posible almacenar datos locales ya que el sistema de ficheros se saturará rápidamente, aspecto que debe evitarse durante el funcionamiento normal del sistema.

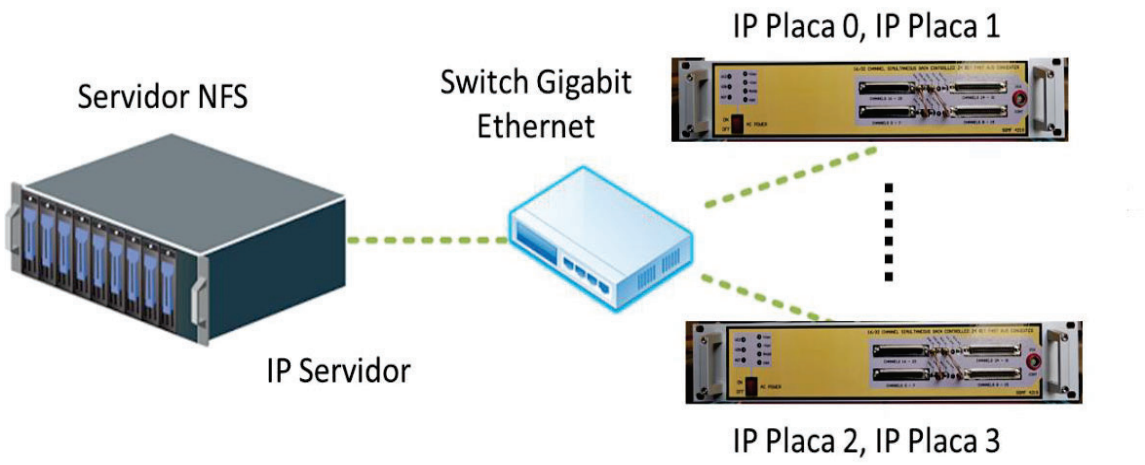



Figura 67. Arquitectura NFS para el almacenamiento de datos

Cada una de las placas deberá montar la unidad de almacenamiento exportada por el servidor NFS mediante procedimientos y comandos estándar en Linux. La optimización de las distintas opciones depende de cada instalación, siendo responsabilidad del administrador de sistemas su puesta a punto.

En cuanto a la cantidad de datos a almacenar durante la captura, en la Tabla 3 se muestra la cantidad de datos y ficheros que se crean para los distintos modos de captura (*Fast, Median, Low*), a máximo rendimiento del ADC capturando 256 KSPS, para distintos promedios de datos. Como se puede apreciar, la captura de aproximadamente una hora de datos implica el almacenamiento de 235,93 GB de muestras. Dada la capacidad de procesamiento del *hardware* diseñado, si se promedian los datos cada 64 muestras, la información almacenada se reduce significativamente, pasando a 3,69 GB para el mismo tiempo de captura.

El formato de fichero binario obtenido consta de filas de datos de 16 Bytes, incluyendo la marca de tiempo (64 bits), el dato de 24 bits y el identificador de canal y de placa de 8 bits (Figura 62). El formato está optimizado para transferencias alineadas en memoria de ancho de 128 bits, lo que permite una escritura en ráfagas de 64 bits de forma eficiente desde el DMA.

Para mostrar los datos se dispone de la aplicación `dspace-converter`, que muestra la información de las muestras a partir del fichero binario, tal como se explicó en el apartado 7.6.

Tabla 3. Cantidad de datos generados para 16 canales durante la captura de 1 hora para distintos perfiles de configuración de ADC para una decimation 0-0 (32) y una frecuencia de datos de MCLK/4.

| POWER MODE | f-MOD | OUTPUT DATA RATE (ODR) (samples/s) | SAMPLES AVERAGE | OUTPUT AVERAGED DATA (OADR) (average/s) | File Size (GB) | RATE (KBytes/s) |
|---|---|---|---|---|---|---|
| Fast | MCLK/4 | 256000 | 1 | 256000 | 235,93 | 65536 |
| Fast | MCLK/4 | 256000 | 2 | 128000 | 117,96 | 32768 |
| Fast | MCLK/4 | 256000 | 4 | 64000 | 58,98 | 16384 |
| Fast | MCLK/4 | 256000 | 8 | 32000 | 29,49 | 8192 |
| Fast | MCLK/4 | 256000 | 16 | 16000 | 14,75 | 4096 |
| Fast | MCLK/4 | 256000 | 32 | 8000 | 7,37 | 2048 |
| Fast | MCLK/4 | 256000 | 64 | 4000 | 3,69 | 1024 |

| POWER MODE | f-MOD | OUTPUT DATA RATE (ODR) (samples/s) | SAMPLES AVERAGE | OUTPUT AVERAGED DATA (OADR) (average/s) | File Size (GB) | RATE (KBytes/s) |
|---|---|---|---|---|---|---|
| Fast | MCLK/4 | 256000 | 128 | 2000 | 1,84 | 512 |
| Fast | MCLK/4 | 256000 | 256 | 1000 | 0,92 | 256 |
| Fast | MCLK/4 | 256000 | 512 | 500 | 0,46 | 128 |
| Fast | MCLK/4 | 256000 | 1024 | 250 | 0,23 | 64 |
| Median | MCLK/8 | 128000 | 1 | 128000 | 117,96 | 32768 |
| Median | MCLK/8 | 128000 | 2 | 64000 | 58,98 | 16384 |
| Median | MCLK/8 | 128000 | 4 | 32000 | 29,49 | 8192 |
| Median | MCLK/8 | 128000 | 8 | 16000 | 14,75 | 4096 |
| Median | MCLK/8 | 128000 | 16 | 8000 | 7,37 | 2048 |
| Median | MCLK/8 | 128000 | 32 | 4000 | 3,69 | 1024 |
| Median | MCLK/8 | 128000 | 64 | 2000 | 1,84 | 512 |
| Median | MCLK/8 | 128000 | 128 | 1000 | 0,92 | 256 |
| Median | MCLK/8 | 128000 | 256 | 500 | 0,46 | 128 |
| Median | MCLK/8 | 128000 | 512 | 250 | 0,23 | 64 |
| Median | MCLK/8 | 128000 | 1024 | 125 | 0,12 | 32 |
| Low | MCLK/32 | 32000 | 1 | 32000 | 29,49 | 8192 |
| Low | MCLK/32 | 32000 | 2 | 16000 | 14,75 | 4096 |
| Low | MCLK/32 | 32000 | 4 | 8000 | 7,37 | 2048 |
| Low | MCLK/32 | 32000 | 8 | 4000 | 3,69 | 1024 |
| Low | MCLK/32 | 32000 | 16 | 2000 | 1,84 | 512 |
| Low | MCLK/32 | 32000 | 32 | 1000 | 0,92 | 256 |
| Low | MCLK/32 | 32000 | 64 | 500 | 0,46 | 128 |
| Low | MCLK/32 | 32000 | 128 | 250 | 0,23 | 64 |
| Low | MCLK/32 | 32000 | 256 | 125 | 0,12 | 32 |
| Low | MCLK/32 | 32000 | 512 | 63 | 0,06 | 16 |
| Low | MCLK/32 | 32000 | 1024 | 31 | 0,03 | 8 |

## 8.2.1. ACCESO A LAS MUESTRAS CAPTURADAS

Tal como se indicó en la Figura 67, el esquema de almacenamiento y acceso a los datos capturados por el equipo está basado en la utilización del sistema de ficheros distribuidos NFS que comparte un área de datos con el equipo, que actúa de cliente NFS. La placa ZedBoard soporta la utilización de un cliente NFS de tal forma que los datos se ven de forma transparente en el servidor NFS disponible. La gestión de los datos (consolidación, análisis, almacenamiento, compresión, copia de seguridad, política de acceso, …) se realiza en el servidor o en distintos servidores y equipos con acceso a los datos desde el servidor de ficheros. El usuario deberá preparar el escenario de uso, montando el área de datos exportada desde el servidor. Esta solución aporta mayor flexibilidad y escalabilidad para el acceso a las distintas tarjetas y *subracks* de conversión de datos.

Se indican a continuación, a modo de ejemplo, la utilización de programas disponibles en Linux para la gestión de los datos.

### *8.2.1.1. Cliente NFS*

La placa ZedBoard incluye un cliente NFS con soporte a las versiones 2, 3 y 4 (incluidos 4.1 y 4.2). Para tener acceso al área de datos el servidor debe exportar el directorio donde almacenar los datos, que el cliente montará en su sistema de ficheros local.

En el siguiente ejemplo se ilustra el procedimiento a seguir:

1. IP Servidor NFS: 192.168.1.5
2. IP ZedBoard: 192.168.1.25

Suponiendo que exista un servidor NFS operativo y que los datos se guardarán en `/srv/nfs/zedboard0-data` :

En el servidor:

- Editar el fichero `/etc/exportfs` del servidor añadiendo la línea correspondiente:

```
/srv/nfs/zedboard0-data 192.168.1.25(rw,sync,no_root_squash,
        no_all_squash, crossmnt)
[Las opciones dependen de cada instalación]
```

- Ejecutar la orden `exportfs –a`

En la ZedBoard, a través de una sesión SSH, si se utiliza la versión 3 de NFS, ejecutar la orden:

```
mount -t nfs -o nolock 192.168.1.5:/srv/nfs/zedboard0-data
        /home/root/data-files
```

En el anterior ejemplo `/home/root/data-files` indica la carpeta del sistema de ficheros donde se capturan los datos en la correspondiente ZedBoard.

Esta opción puede quedarse configurada de forma permanente usando la línea correspondiente en el fichero `/etc/fstab` de la ZedBoard (configurados desde las opciones de Petalinux).

```
# <source>    <local dir>  <type>   <options>   <dump>     <pass>
192.168.1.5:/srv/nfs/zedboard0-data /home/root/data-files nfs defaults 0
        0
```

y ejecutando la orden `mount -a`.

*8.2.1.2. rsync*

La utilidad rsync permite la sincronización remota de los datos a través de una conexión SSH desde el servidor NFS a la ubicación definitiva de los datos. Una de las funcionalidades de interés es que permite una sincronización completa, incluido el borrado de los datos en el origen.

La sesión de sincronización se puede iniciar con la siguiente orden desde el servidor, en este caso copiando todos los datos disponibles. En este ejemplo se incluye la opción de borrar los datos en el origen. Las opciones dependen de la versión de Linux utilizada.

```
rsync --remove-source-files root@192.168.1.5:/home/root/data-files/* .
```

Toda la gestión de movimiento de datos se realiza desde el servidor NFS, nunca desde la placa ZedBoard.

*8.2.1.3. Secure Copy Protocol*

SCP se utiliza para la transferencia segura de archivos informáticos entre un host local y otro remoto o entre dos hosts remotos, usando el protocolo *Secure Shell* (SSH).

El programa ‘scp’ es un cliente que implementa el protocolo SCP. El cliente SCP más ampliamente usado es el programa ‘scp’ disponible en la mayoría de los sistemas, que es incorporado en la mayoría de las implementaciones de SSH.

La utilización para la transferencia de datos soporta la copia entre servidores remotos:

```
scp root@192.168.1.5:/home/root/data-files/* .
```

Toda la gestión de movimiento de datos se realiza desde el servidor NFS, nunca desde la placa ZedBoard.

## 8.3. CONCLUSIÓN

En este capítulo se ha podido ver cómo es la captura de muestras del sistema, cómo se almacenan los datos y cómo se acceden a ellos. Se presenta la configuración necesaria para que la característica principal del sistema, la captura de datos científicos se pueda realizar con éxito. Además, se dan casos de ejemplo para gestionar los datos a nivel usuario en un sistema operativo Linux.

# CAPÍTULO 9. EVALUACIÓN DEL SISTEMA

## 9.1. INTRODUCCIÓN

Establecido todo el sistema y con un flujo de trabajo definido, queda realizar la evaluación de este al completo. Comprobar que los recursos de los que se disponen se aprovechan adecuadamente y que cuando se hace un uso intensivo de estos el sistema no se degrada.

## 9.2. RECURSOS DEL EQUIPO

Como se ha comentado anteriormente, de las dos CPUs que se disponen una está destinada a una aplicación *baremetal* y otra a un sistema Linux, además, el sistema posee una serie de características *hardware* y *software,* ya mencionadas pero que se enumeran aquí a continuación nuevamente:

- *Hardware*
    - Memoria *flash* de 256 MB
    - RAM DDR3 de 512 MB DDR3
    - Dos fuentes de reloj una a 100 MHz y otra a 33,333 MHz
- *Software*
    - Aplicaciones al arranque del equipo: *'dspace-srv' y 'dspace-acfail'*

Teniendo presente esto y las optimizaciones que se le han hecho al sistema se presenta en la Figura 68 el estado de este al momento del arranque.

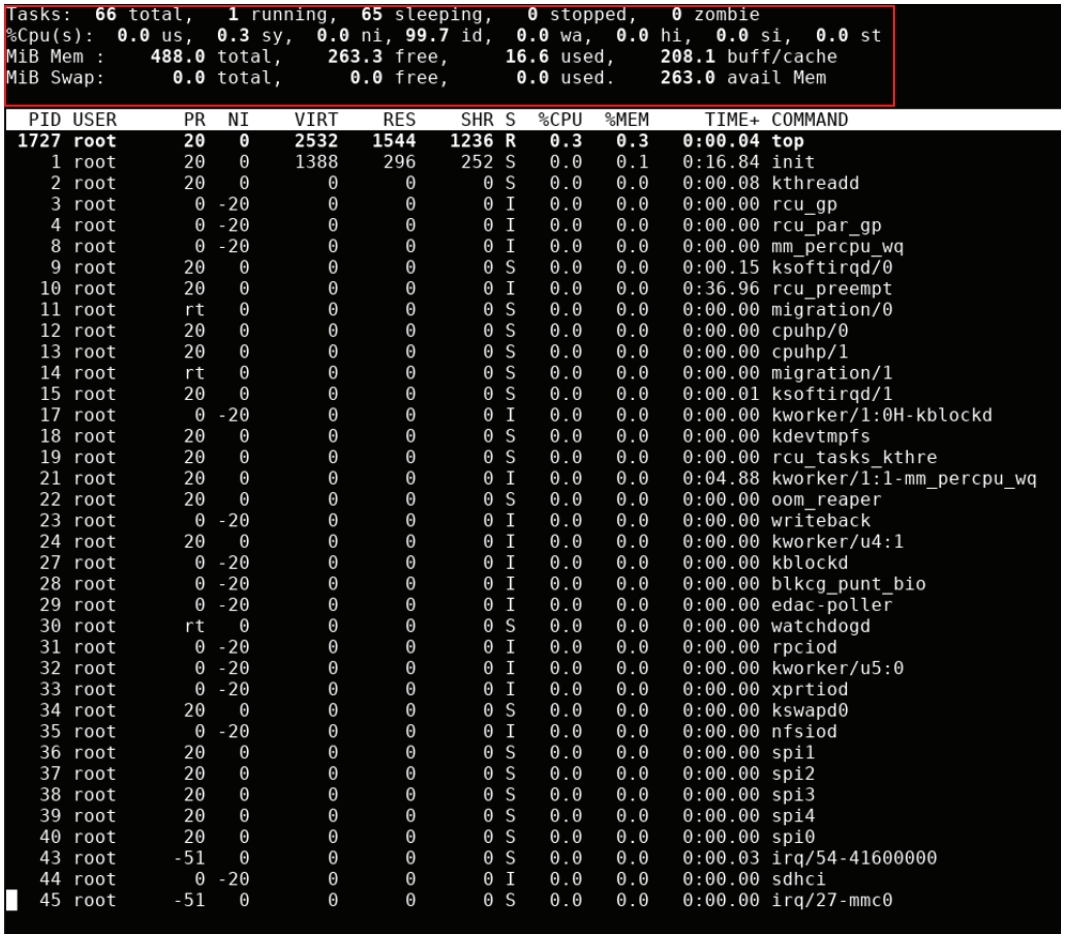



Figura 68. Equipo iniciado sin recibir órdenes

En la figura anterior se hace uso de la herramienta 'top' [125] la cual permite ver los recursos del sistema, así como los procesos en ejecución de este. Analizando la captura anterior se puede observar en la primera línea de la parte superior que en el sistema se encuentran un total de 66 tareas, de las cuales 1 está en ejecución y las otras 65 se encuentran en modo hibernación, es decir, esperando que ocurra algún evento para ejecutarse.

La siguiente línea muestra los porcentajes de uso del procesador diferenciado por el uso que se le dé. Así podemos observar la Tabla 4 que explica cada uno de los apartados.

Tabla 4. Parámetros de la herramienta 'top'

| Parámetro | Explicación |
|---|---|
| **us (*user*)** | Porcentaje de CPU que están consumiendo los procesos que se ejecutan en el espacio de usuario. |
| **sy (*system*)** | Porcentaje de CPU consumida por los procesos que corren en el espacio del sistema o *kernel*. En el espacio del *kernel* normalmente corren procesos para controlar el *hardware* de nuestro equipo, procesos para controlar la asignación de memoria, etc. |
| **ni (*nice value*)** | Muestra el porcentaje de CPU consumida por procesos de baja prioridad. |
| **id (*idle* o inactivo)** | Porcentaje de CPU que no está siendo utilizada. |
| **wa (*wait*)** | Porcentaje de tiempo que el procesador está esperando para que se completen tareas de lectura y escritura en disco, comunicación con otros dispositivos de la red, obtener información de una base de datos, etc. |
| **hi (*hardware interrupt*)** | Porcentaje de tiempo que la CPU no está procesando debido a interrupciones de *hardware*. |
| **si (*software interrupt*)** | Porcentaje de tiempo que la CPU no está procesando debido a interrupciones de *software*. |
| **st (*steal time*)** | Porcentaje de tiempo que la CPU virtual ha estado esperando a la CPU real mientras el hipervisor da servicio a otro procesador virtual. |

Por tanto, atendiendo a la tabla anterior, se puede observar que los únicos campos que tienen valor son el 'sy' con 0.3 y el 'id' con 99.9, lo cual indica que hay poco uso de la CPU por parte del *kernel* y que en un estado de reposo (*idle)* un porcentaje muy alto de la CPU no es utilizada.

En cuanto a la memoria hay disponible un total de 488 MiB de memoria de la cual 263.3 MiB está disponible, 16.6 MiB en uso y 208.1 MiB usado por el *buffer* y la caché juntos. Por otro lado, se observa que la cantidad de memoria *swap* contemplada corresponde casi totalmente con la cantidad de memoria libre. Los valores anteriores se justifican debido al modo de funcionamiento del sistema operativo, el cual es cargado en la RAM de la ZedBoard de 512 MB.

En cuanto a los procesos y su uso de RAM y carga de la CPU, se observan solamente 2 únicos procesos que hagan uso de estos recursos. Por un lado, la herramienta 'top', la cual es solo ejecutada en ocasiones específicas como esta para analizar el estado del sistema, por lo tanto, se considera despreciable en la valoración de utilización del sistema. Y, por otro lado, se encuentra el proceso 'init' con identificador 1, el cual corresponde al primer proceso o demonio que se ejecuta durante el arranque del sistema y es el encargado de inicializar, administrar y rastrear los servicios y demonios del sistema, por lo que se considera el padre de todos los procesos y si no estuviera en ejecución se produciría un *kernel panic.*

Por tanto, se puede concluir que cuando el sistema se encuentra en un estado de reposo o *idle,* los consumos de CPU y de memoria se mantienen al mínimo, quedando como único proceso activo el proceso con identificador 1 'init'.

Durante el estado de captura de datos, el uso de los recursos del equipo cambia considerablemente. Como se puede ver en la Figura 69, aparecen nuevos procesos que toman parte de la memoria y uso de la CPU.

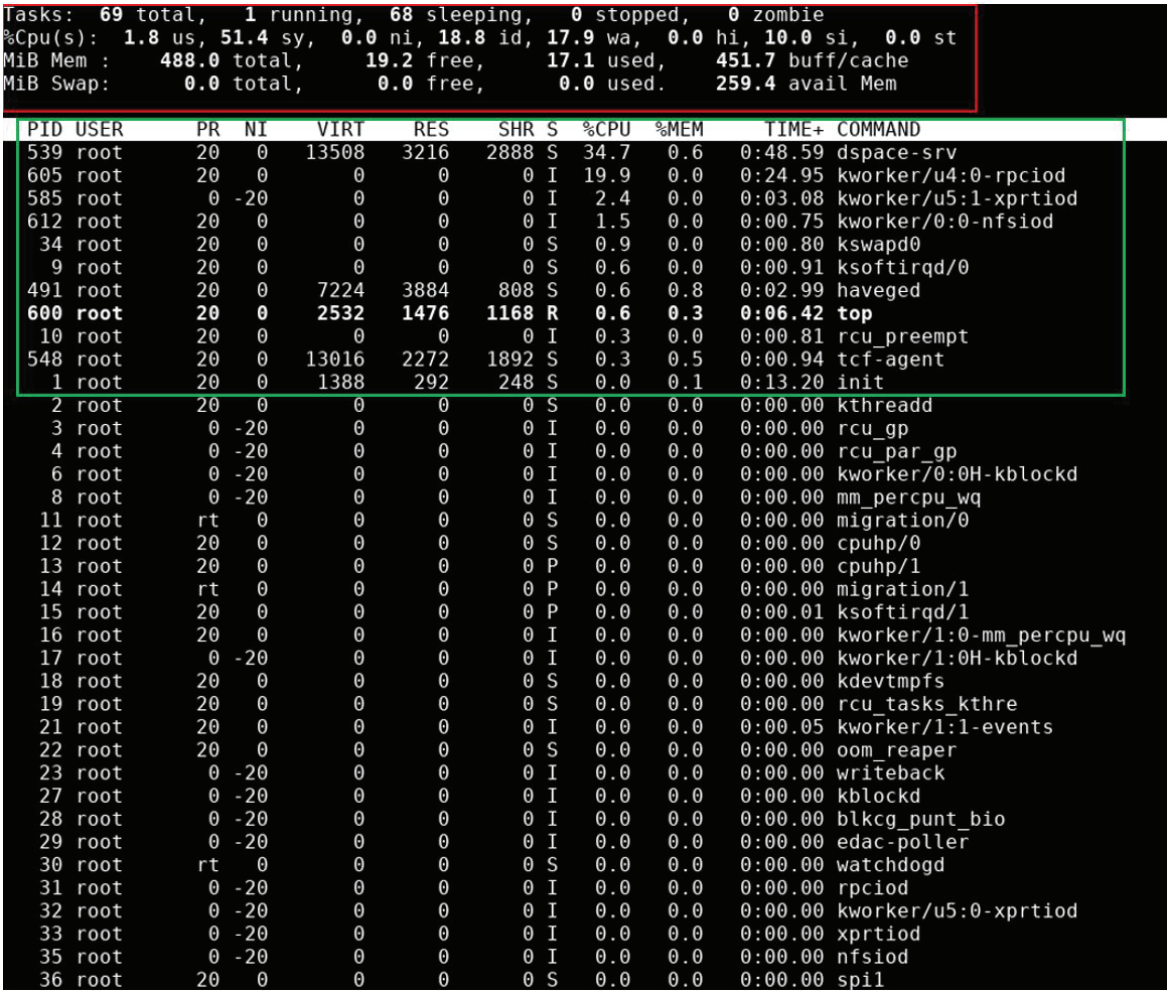



Figura 69. Consumo de recursos durante la captura de datos

La aplicación principal de este TFM aparece como la aplicación que más consumo de CPU está realizando con un 34.7% y un uso total de memoria RAM de 0.6%. Por un lado, se puede observar que la memoria virtual utilizada (VIRT), incluyendo todo el código, los datos y las bibliotecas compartidas más las páginas que han sido intercambiadas y mapeadas corresponde a un total de 13508 KiB. Además, se observar que el valor del RES, el cual corresponde a un subconjunto del espacio de direcciones virtual (VIRT) que representa la memoria física no intercambiada que se encuentra actualmente en uso es de 3216 KiB. Por último, en cuanto a la memoria, el valor del SHR que corresponde con un subconjunto de memoria residente (RES) que puede ser utilizado por otros procesos, posee un valor de 2888 KiB.

Se observa también un conjunto de procesos que en estado de reposo o *idle* no consumían recursos. Teniendo en cuenta que la aplicación se encuentra en un estado de captura, hay que recordar que estos se traspasan a un cliente mediante un protocolo NFS. Es por ello, que aparecen estos nuevos procesos, los cuales son los encargados de la gestión del protocolo, colas, tunelización, interrupciones, *timers*, manejo de la memoria, etc.

En este caso, los procesos llamados 'rpciod', 'xprtiod' y 'nfsiod', que como se observa en la imagen son aquellos que más porcentaje de CPU consumen, siendo 19.9%, 2.4% y 1.5%, son los principales encargados del manejo y funcionamiento del protocolo NFS y otras tareas del sistema. Se encargan de procesar las llamadas a procedimientos remotos (*Remote Procedure Calls*, RPC) de cualquier hilo del cliente local enviándolas a los servidores correspondiente y despachando las repuestas de vuelta a los solicitantes, procesar las colas de estos procedimientos y controlar el número máximo de procesos del *kernel* 'nfsiod' que se ejecutan en una máquina cliente NFS para atender solicitudes de E/S asíncronas a su servidor.

Cuando la captura de datos es detenida, los procesos comentados anteriormente cesan el uso de CPU y RAM, y el sistema vuelve al estado de reposo que se detallaba anteriormente en la Figura 68.

Conociendo esto, observamos las gráficas siguientes donde se recogen el uso de CPU y de RAM dependiendo si el sistema se encuentra en reposo (*idle)* o en estado de captura de datos.

En la gráfica Figura 70, se observa que cuando el sistema se encuentra en reposo la CPU usa una media del 0,3%, mientras que cuando se encuentra en modo captura se usa un 81,7%. Estos dando se pueden obtener observando la aplicación anterior 'top' introduciendo el comando

```
   top -bn2 | grep '%Cpu' | tail -1 | grep -P '(....|...)
id,' | awk '{print 100-$8 "%"}'
```

Esta diferencia tan grande se debe a que durante el modo de reposo todos los procesos se mantienen en un estado de mínimo consumo o *deep sleep*. En el momento en

el que se produce el inicio de la captura se arrancan una serie de procesos necesarios para realizar la captura de datos. Por un lado, se mantiene el hilo de la aplicación y se crea un *fork* con un hilo hijo encargado de la captura de datos. Este se encarga de recoger los datos recibidos, empaquetarlos en un fichero y guardarlo en la ruta especificada. Por otro lado, se inician todos los procesos referentes a la conexión de red y el sistema NFS.

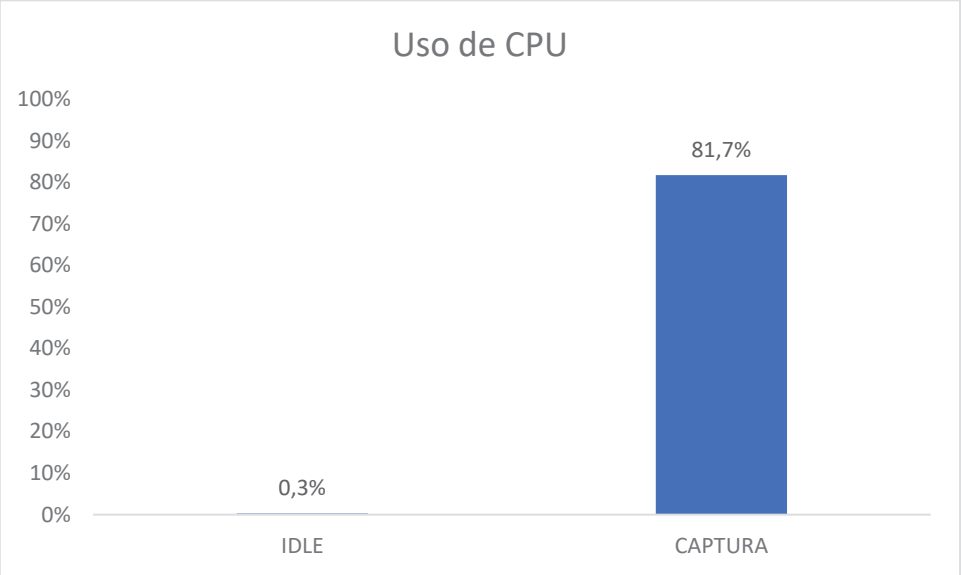



Figura 70. Uso de CPU del sistema

En cuanto al uso de la RAM se observa en la Figura 71 que durante el estado de reposo esta se encuentra en una media de uso del 54% y cuando se inicia el proceso de captura se establece en un 96,7%. Esto se mide usando el comando

```
free -t | awk 'FNR == 2 {printf("%.2f%"), $3/$2*100}'
```

Como se ha visto en capítulos anteriores, el sistema operativo introducido en la SD se carga en la memoria RAM del equipo, por ello cuando el sistema se encuentra en estado de reposo se observa que existe un 54% de memoria RAM disponible. Cuando se inicia el proceso de captura se realiza un incremento de uso de la memoria RAM, llegando al 96,7%, debido al procesamiento, creación y guardado de los ficheros de los datos capturados. Aunque el equipo posee un sistema NFS donde se guardan los ficheros, y que, por tanto, no se almacenan directamente en la tarjeta SD, las funciones utilizadas para los procesos descritos anteriormente también consumen parte de la memoria RAM en forma de caché.

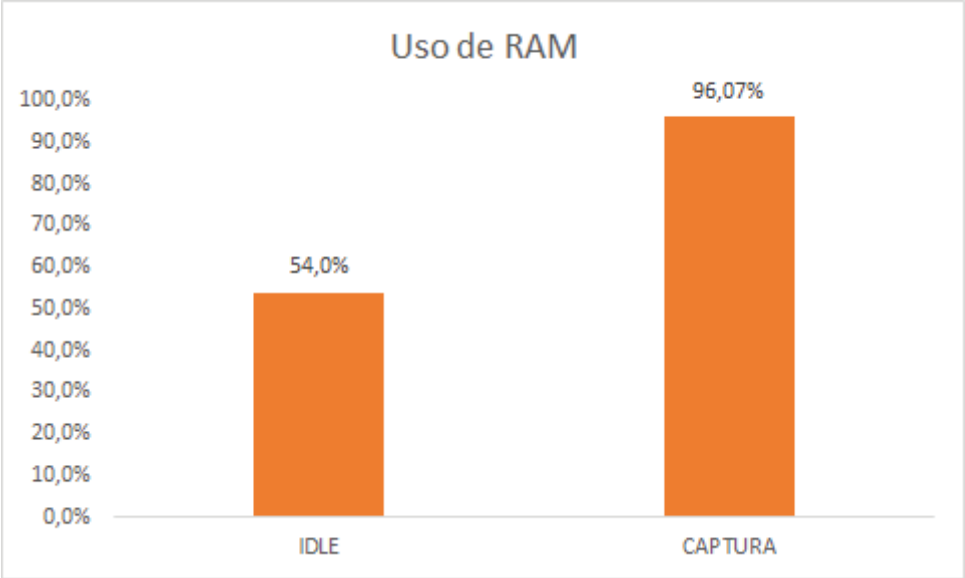



Figura 71. Uso de RAM del sistema

Este uso de la memoria en forma de caché no puede ser evitado porque está intrínsecamente relacionada con la forma en la que están implementadas las funciones de captura y procesado de los datos (`open()`, `close()`, etc.) . Esta memoria, y otra residual de otros procesos, solo puede ser liberada una vez se haya terminado el proceso de captura. Para ello, como ya se describió en capítulos anteriores de optimización del sistema, para poder obtener siempre el mayor número de recursos disponibles se hace uso del comando adecuado que permite el correcto vaciado de la memoria caché

```
echo 3 > /proc/sys/vm/drop_caches
```

## 9.3. ANÁLISIS DEL USO DE MEMORIA

Como se ha podido observar la memoria es un recurso escaso, por ello cuando se crean aplicaciones para este tipo de sistemas con lenguajes como C y C++ se debe tener en cuenta que estas cumplan con la máxima optimización posible y no posean fugas de memoria [126].

Las fugas de memoria se producen cuando un programa administra incorrectamente la asignación de memoria, no liberando aquella que ya no se necesite o creando regiones de memoria inaccesible. Esto produce que la memoria de la que dispone

el dispositivo se vaya agotando poco a poco, por lo que con el paso del tiempo se puede producir un error en el sistema por falta de memoria.

Los lenguajes C y C++ no tienen recolector de basura como si lo tienen otros lenguajes como Java [127] o C#, los cuales se encargan de liberar de la memoria automáticamente todos los objetos y regiones de memoria que ya no sean necesarias o sean inaccesibles. Por ello, cada desarrollador es responsable de crear los mecanismos necesarios para gestionar correctamente la memoria.

Una de las herramientas de *profiling* más usadas es Valgrind [128], esta permite detectar errores en la gestión de memoria e hilos. Esta herramienta está incluida en Petalinux, necesitando solo activarla en:

**petalinux-config -c rootfs**
Filesystem Packages >> misc >> packagegroup-core-tools-profile >> packagegroup-core-tools-profile
Filesystem Packages >> misc >> packagegroup-core-tools-profile >> packagegroup-core-tools-profile-dbg

El comando anterior, además de Valgrind, incluye otras herramientas de *profiling* que pueden ser útiles para otros análisis del sistema. Además, es necesario que la aplicación en el momento de la compilación añada la información de *debug*. Esto se hace añadiendo las siguientes líneas en el fichero `<plnx-proj-root>/project-spec/meta-user/recipe-apps/dspace-srv/dspace-srv.bb`:

```
DEBUG_FLAGS = "-g3 -O0"
DEBUG_BUILD = "1"
INHIBIT_PACKAGE_STRIP = "1"
INHIBIT_PACKAGE_DEBUG_SPLIT = "1"
EXTRA_IMAGE_FEATURES_append = " dbg-pkgs"
```

A continuación, se compila el proyecto nuevamente usando el comando `petalinux-build`. Con esta herramienta ya en el sistema se puede realizar el análisis de la memoria ejecutando la orden:

```
valgrind --leak-check=full --keep-debuginfo=yes --show-leak-kinds=all --track-origins=yes --verbose --log-file=valgrind-log.txt dspace-srv
```

Este comando lanza Valgrind sobre la aplicación 'dspace-srv' junto con las opciones que se especifican en la Tabla 5.

Tabla 5. Opciones Valgrind utilizadas

| Opción | Explicación |
|---|---|
| `--leak-check` | Cuando se encuentra activado, busca fugas de memoria cuando el cliente termina. Establecido como *full* o *yes,* cada fuga de memoria se mostrará de forma individual de manera detallada o se mostrará como un error según se especifique en `--show-leak-kinds` |
| `--keep-debuginfo` | Guarda los símbolos y el resto de la información de depuración del código. Esto permite que los rastros de la pila guardados incluyan información de archivo/línea para el código que ha sido *dlclose'd.* |
| `--show-leak-kinds` | Especifica los tipos de fuga que se mostrarán en una búsqueda de fuga completa. |
| `--track-origins` | Controla si Memcheck rastrea el origen de los valores no inicializados. Cuando está configurado como *yes,* Memcheck mantiene un registro de los orígenes de todos los valores no inicializados. |
| `--verbose` | Proporciona información extra sobre aspectos del programa analizado, como: los objetos compartidos cargados, las supresiones utilizadas, etc. |
| `--log-file` | Especifica que Valgrind guarde todos los mensajes al archivo especificado. |
| `app-name` | Nombre de la aplicación que se desea analizar |

Tras emplear el comando anterior como se ha definido se obtiene un fichero llamado `valgrind-log.txt` que contiene todos los mensajes que produce Valgrind durante su análisis: fugas de memoria, errores en hilos, etc.

El fichero anterior contiene los mensajes proporcionados por Memcheck [129], el cual es el encargado de detectar los errores de memoria, que facilitan conocer si se han producido errores o fugas. Estas pueden ser del tipo que se especifican en la Tabla 6.

Tabla 6. Mensajes de errores de Memcheck

| Mensaje | Explicación |
|---|---|
| **Definitely lost** | El programa tiene una fuga de memoria. |
| **Indirectly lost** | El programa está perdiendo memoria en una estructura basada en punteros. |
| **Possibly lost** | El programa tiene una fuga de memoria. Este error puede aparecer cuando algún puntero apunte a la mitad de un bloque de memoria. |
| **Still reachable** | El programa probablemente no tiene errores, aunque hay alguna memoria que no se ha liberado correctamente. Es un error común. |
| **Suppressed** | Se ha suprimido un error de fuga. Estos errores pueden ignorarse. |

Si se encuentran errores en la aplicación se presentan como en la Figura 72, donde se puede ver cuántos errores se encuentran y los tipos de fuga, indicando el número de bytes perdidos y en cuantos bloques de memoria.

```
==3944== LEAK SUMMARY:
==3944==    definitely lost: 0 bytes in 0 blocks
==3944==    indirectly lost: 0 bytes in 0 blocks
==3944==      possibly lost: 25,748 bytes in 497 blocks
==3944==    still reachable: 7,142 bytes in 186 blocks
==3944==         suppressed: 0 bytes in 0 blocks
==3944==
==3944== ERROR SUMMARY: 5 errors from 5 contexts (suppressed: 0 from 0)
```

Figura 72. Log de Valgrind indicando las fugas de memoria y errores encontrados

Analizando el fichero anterior obtenido de la ejecución de Valgrind, se puede observar que no se encuentran errores ni fugas de memoria en las aplicaciones desarrolladas. En la Figura 73 se puede ver el mensaje que aparece cuando el análisis es exitoso y no se encuentran errores ni una mala gestión de memoria.

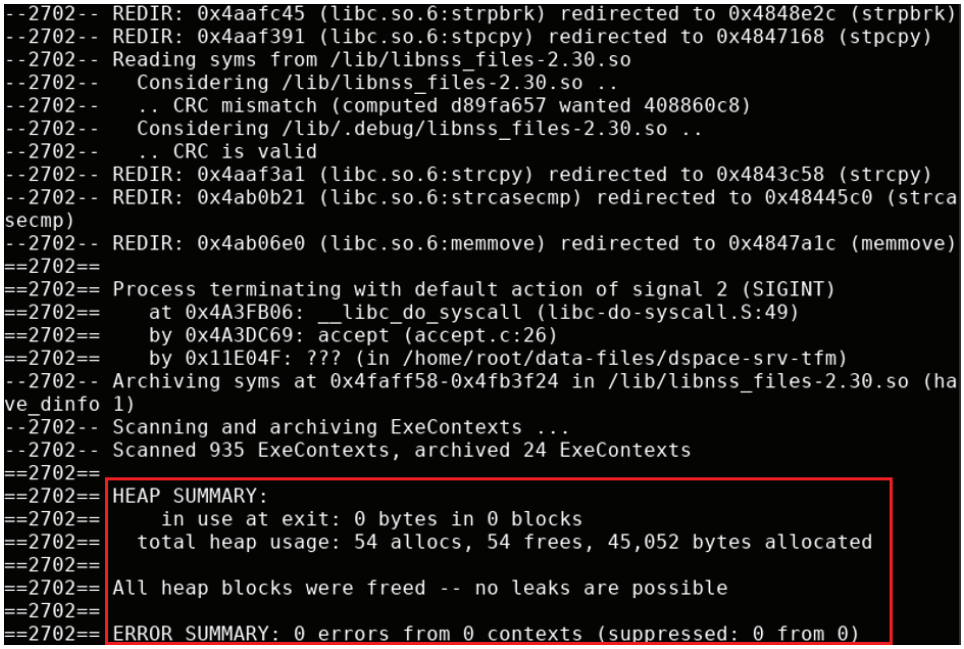



Figura 73. Log de Valgrind sin fugas de memoria ni errores encontrado

Por tanto, con este análisis se puede asegurar que todas las aplicaciones desarrolladas gestionan correctamente la memoria y no contienen errores que puedan afectar al funcionamiento del sistema.

Aunque el análisis anterior es suficiente para comprobar el buen desempeño de las aplicaciones, si se desea realizar un análisis visual, Valgrind permite crear ficheros que contienen toda la información necesaria para ser analizada por programas de terceros. Algunos de estos son Massif [130], Heaptrack [131] y Callgrind (Kcachegrind [132]).

Para crear el fichero que pueda ser analizado se usa el comando

```
valgrind --tool=<analysis_programme> myapp
```

Con esto se genera un fichero que tiene el nombre de la herramienta utilizada y un código numérico:

```
callgrind.out.XXXX

massif.out.XXXX
```

Kcachegrind es el visualizador de la herramienta Callgrind. Esta última utiliza instrumentación en tiempo de ejecución a través del *framework* Valgrind para la simulación de caché y generación de gráficos de llamadas. De esta forma, incluso las bibliotecas compartidas y los *plugins* abiertos dinámicamente pueden ser perfilados. Un ejemplo de lo

que se puede visualizar con esta herramienta, es un gráfico completo de las llamadas que se producen durante la ejecución de la aplicación analizada.

Como se puede observar en la Figura 74 y Figura 75, perteneciente a la aplicación 'dspace-srv', esta contiene cada una de las llamadas a funciones en orden de ejecución que se realizan cuando esta arranca por primera vez. La herramienta permite pinchar en cada uno de los bloques y navegar de manera más profunda en cada de unas llamadas de las funciones, permitiendo un análisis exhaustivo de cada una de las funciones de la aplicación.

Por otro lado, estas llamadas se pueden analizar en una lista (Figura 76) que proporciona el programa de forma más ordenada. Esta lista también permite navegar de forma específica en cada una de las llamadas, permitiendo ver las dependencias de cada una de ella.

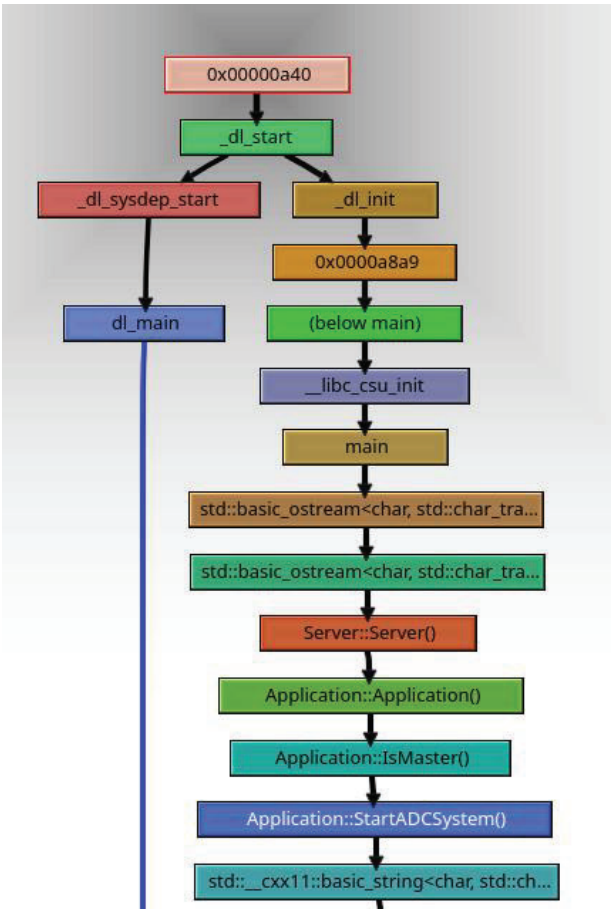



Figura 74. Gráfico con las llamadas a funciones durante el arranque de 'dspace-srv' (Parte 1)

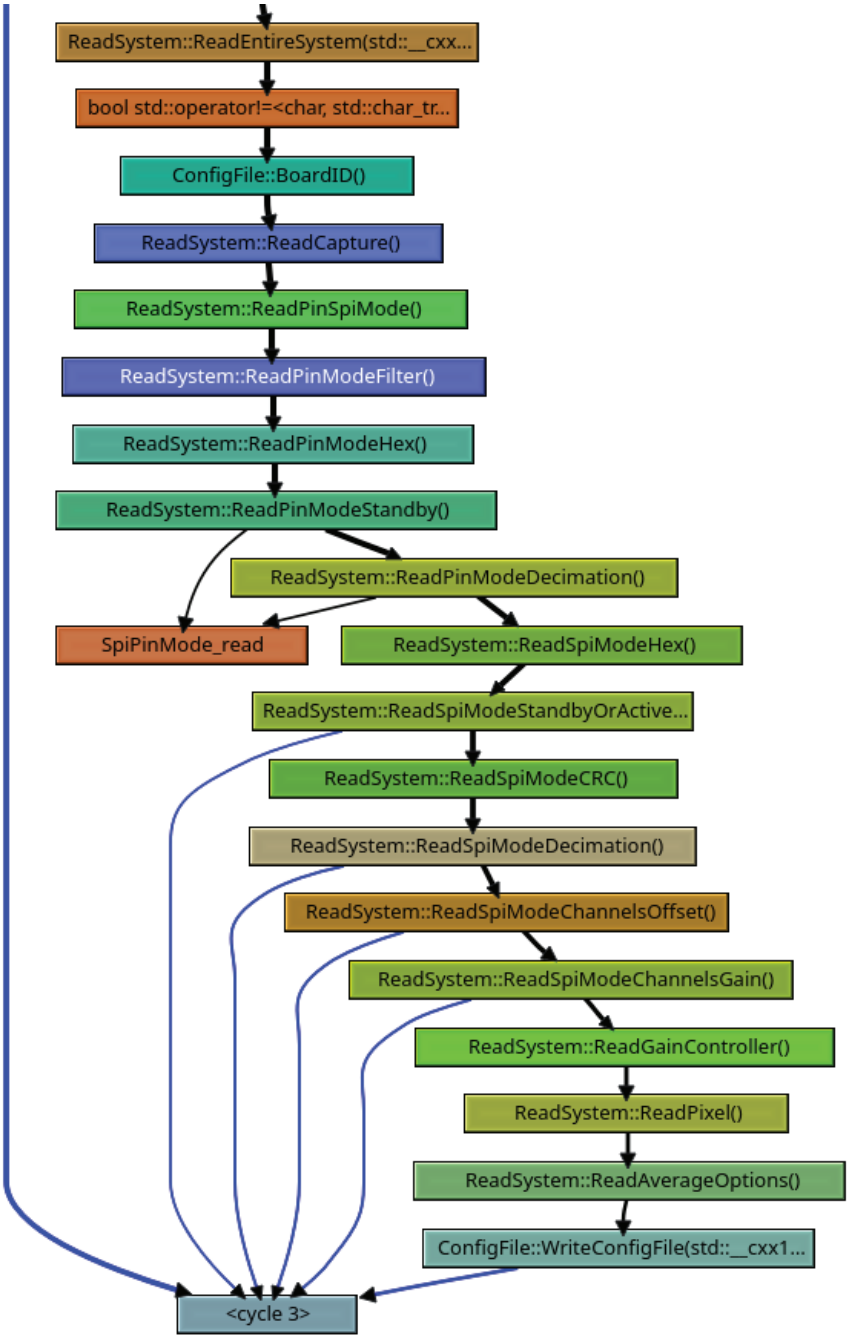



Figura 75. Gráfico con las llamadas a funciones durante el arranque de 'dspace-srv' (Parte 2)

Otra de las opciones también permite ver un mapa de las llamadas a las funciones (Figura 77), donde se puede observar el tamaño que ocupa cada una de las funciones y dónde está contenida. De nuevo, de forma gráfica se puede visualizar que paquetes de funciones realizan que función y sus dependencias.

## 9.4. ANÁLISIS DEL ANCHO DE BANDA DE RED

La conexión Ethernet del sistema es otro de los puntos clave que se deben analizar. Esto es debido a que los datos capturados por el dispositivo se transfieren mediante esta conexión y el protocolo NFS. Por tanto, se debe asegurar que la velocidad teórica de funcionamiento de la tarjeta de red que dice el fabricante (1 Gb/s) se corresponde con la velocidad real a la que se pueden transmitir datos a través de esta.

Para analizar la conexión se utiliza la herramienta 'dd' [133]. Esta herramienta sencilla, útil y fácil de usar, permite copiar ficheros desde un origen hacia un destino ya sea sobre discos o particiones.

Teniendo montada la partición NFS, la velocidad de transferencia de los archivos se mide con el comando:

```
  time  dd  if=/dev/zero  of=/home/root/data-files/dspace-
1.dat bs=4k count=16
```

El comando anterior se desglosa en la Tabla 7.

Tabla 7. Desglose del comando 'dd' usado

| Comando | Explicación |
|---|---|
| **time** | Estadísticas de tiempo |
| **dd** | Comando de copia |
| **if=FILE** | Fichero de entrada (origen) |
| **of=FILE** | Fichero de salida (destino) |
| **bs=BYTES** | Leer y escribir hasta X bytes a la vez |
| **count=N** | Copiar N bloques de entrada |

Análisis de rendimiento plano

Search: Search Query (No Grouping)

| Incl. | Self | Called | Function | Location |
|---|---|---|---|---|
| 100.00 | 0.00 | (0) | 0x00000a40 | ld-2.30.so |
| 100.00 | 0.01 | 1 | _dl_start | ld-2.30.so: rtld.c, dl-machine.h, get-dynamic-info.h, do-rel.h, dl-irel.h |
| 98.66 | 21.91 | 527 | <cycle 3> | libstdc++.so.6.0.27 |
| 64.54 | 0.00 | 1 | _dl_init | ld-2.30.so: dl-init.c |
| 63.75 | 0.00 | 1 | 0x0000a8a9 | dspace-srv-tfm: start.S |
| 63.75 | 0.00 | 1 | (below main) | libc-2.30.so: libc-start.c |
| 63.75 | 0.00 | 1 | __libc_csu_init | dspace-srv-tfm: libc-start.c, elf-init.c |
| 63.74 | 0.00 | 1 | main | dspace-srv-tfm: dspace-srv.cpp |
| 63.74 | 0.00 | 1 | std::basic_ostream<char, st... | libstdc++.so.6.0.27: dspace-srv.cpp |
| 63.60 | 0.00 | 1 | std::basic_ostream<char, st... | libstdc++.so.6.0.27: dspace-srv.cpp |
| 63.60 | 0.00 | 1 | Server::Server() | dspace-srv-tfm: Server.h, dspace-srv.cpp |
| 63.60 | 0.00 | 1 | Application::Application() | dspace-srv-tfm: Application.h, dspace-srv.cpp |
| 61.85 | 0.00 | 1 | Application::IsMaster() | dspace-srv-tfm: Application.cpp, dspace-srv.cpp |
| 61.81 | 0.00 | 1 | Application::StartADCSyste... | dspace-srv-tfm: Application.cpp, dspace-srv.cpp |
| 60.65 | 0.02 | 79 | std::__cxx11::basic_string<c... | libstdc++.so.6.0.27: stl_pair.h, dspace-srv.cpp |
| 60.52 | 0.00 | 1 | ReadSystem::ReadEntireSy... | dspace-srv-tfm: ReadSystem.cpp, dspace-srv.cpp |

Figura 76. Listado de llamadas de las funciones de ‘dspace-srv’

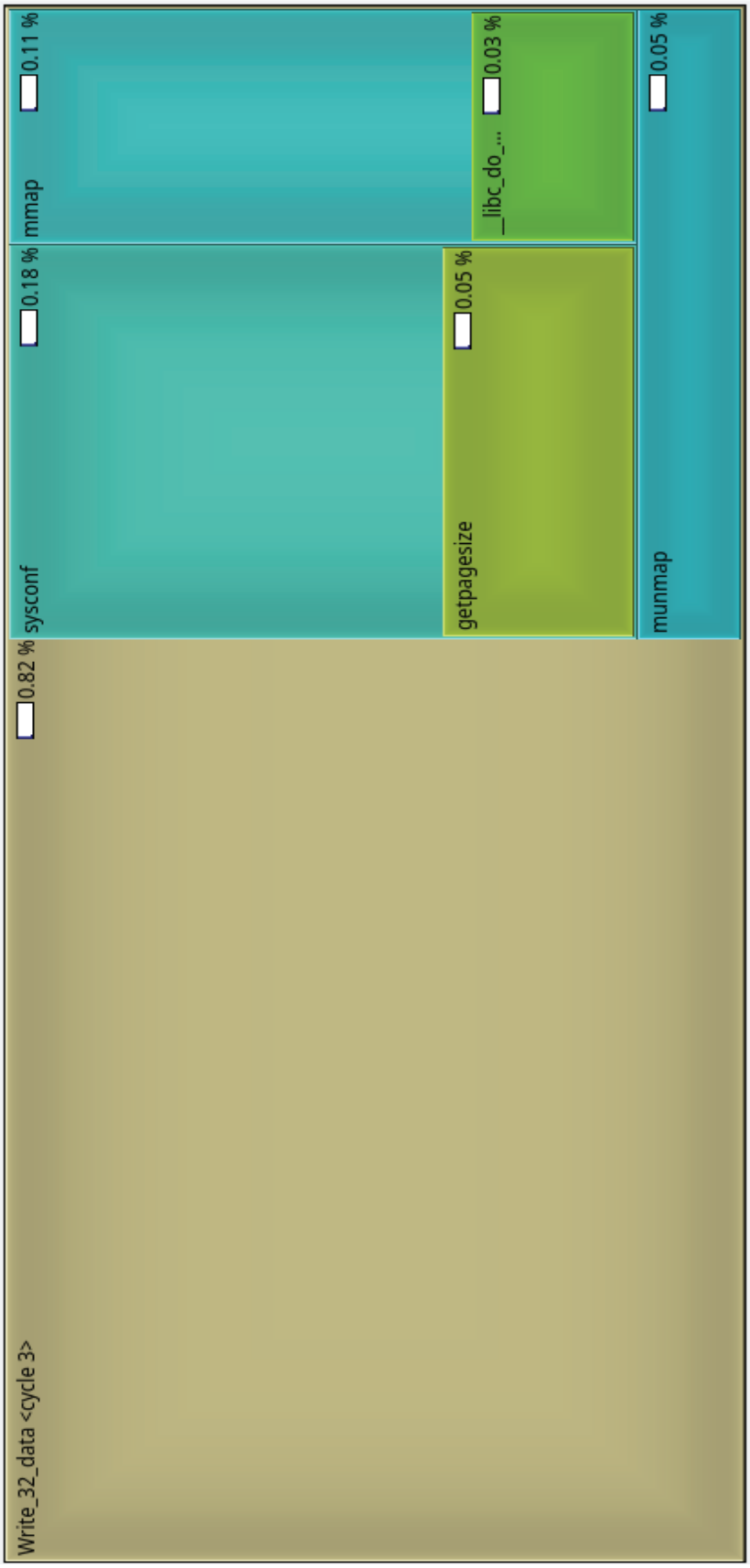



Figura 77. Mapa de llamadas de las funciones `Write_32_data`

Antes de lanzar este comando, es recomendable vaciar la caché del sistema, para que las velocidades obtenidas sean siempre las óptimas. Para ello, se utiliza el comando ya visto en otros apartados:

```
echo 3 > /proc/sys/vm/drop_caches
```

Variando el tamaño del número de bloques y los bytes que se deben leer-escribir, se determina el tamaño del fichero a enviar y se obtienen diferentes velocidades. El tamaño de este se obtiene de la multiplicación entre las opciones 'bs' y 'count'. Los tamaños utilizados para las pruebas se han realizado intercambiando los valores de 'bs' y 'count' para que resultaran en los tamaños de ficheros de 2.1GB (Figura 78), 537 MB (Figura 79), 2.1 MB (Figura 80) y 66 kB (Figura 81).

```
root@zedboard_master:~/data-files# ./dd.sh
dd.sh (4524): drop_caches: 3
131072+0 records in
131072+0 records out
2147483648 bytes (2.1 GB, 2.0 GiB) copied, 43.6679 s, 49.2 MB/s

real    0m43.687s
user    0m0.263s
sys     0m19.493s
```

Figura 78. Tamaño de fichero enviado de 2.1 GB

```
root@zedboard_master:~/data-files# ./dd.sh
dd.sh (4549): drop_caches: 3
131072+0 records in
131072+0 records out
536870912 bytes (537 MB, 512 MiB) copied, 16.0167 s, 33.5 MB/s

real    0m16.229s
user    0m0.290s
sys     0m5.492s
```

Figura 79. Tamaño de fichero enviado de 537 MB

```
root@zedboard_master:~/data-files# ./dd.sh
dd.sh (731): drop_caches: 3
512+0 records in
512+0 records out
2097152 bytes (2.1 MB, 2.0 MiB) copied, 0.140288 s, 14.9 MB/s

real    0m0.152s
user    0m0.000s
sys     0m0.029s
```

Figura 80. Tamaño de fichero enviado de 2.1 MB

```
root@zedboard_master:~/data-files# ./dd.sh
dd.sh (648): drop_caches: 3
16+0 records in
16+0 records out
65536 bytes (66 kB, 64 KiB) copied, 0.00776055 s, 8.4 MB/s

real    0m0.018s
user    0m0.000s
sys     0m0.009s
```

Figura 81. Tamaño de fichero enviado de 66 kB

Analizado las figuras, se observa que el fichero de mayor tamaño posee la mayor velocidad de transferencia con un total de 49.2 MB/s. Esta velocidad, siendo la máxima obtenida en diversas pruebas, dista de la velocidad máxima que puede ofrecer la conexión Ethernet de la placa que es de aproximadamente 125 MB/s. Estos resultados se asemejan a estudios similares donde también se obtienen velocidades bajas de transferencia mediante Ethernet. Estos problemas de bajo rendimiento se atribuyen a la velocidad del reloj de los núcleos ARM [134], [135].

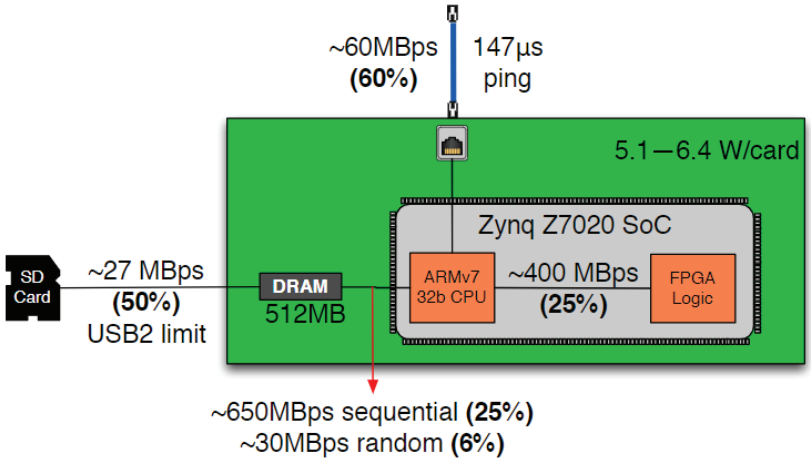



Figura 82. Velocidades de transferencia de las interfaces de la FPGA ZedBoard [134]

Por otro lado, se observa que a medida que el tamaño del fichero a transmitir disminuye, también lo hace la velocidad de transferencia. Por tanto, sería beneficioso para el sistema crear ficheros de datos tan grandes como fuera posible. Sin embargo, debido a la reducida memoria que posee la placa y que estos se crean y almacenan en ella, no es posible crear ficheros de gran tamaño sin producir fallos como *kernel panic* o saturar

rápidamente la memoria. Por ello, la creación de ficheros de gran tamaño queda descartada y se debe recurrir a ficheros de menor tamaño, lo que produce un cuello de botella en la salida de datos.

## 9.5. ANÁLISIS DEL ANCHO DE BANDA DE LA TARJETA SD

Por otro lado, aunque no se utiliza el almacenamiento de esta, también se analiza en rendimiento de la tarjeta SD. Para ello, se hace uso del comando anteriormente utilizado. Hay que tener en cuenta que en este caso no se pueden hacer pruebas con ficheros de gran tamaño, pues la memoria de la tarjeta y la RAM del equipo es limitada. Se usa el comando:

```
time dd if=/dev/zero of=/tmp/testfile.bin bs=4k count=16k
```

En este caso, las velocidades de escrituras de la tarjeta han sido bastante superiores a la velocidad obtenidas en la transferencia vía Ethernet como se puede ver en la Figura 83 y en la Figura 84.

```
root@zedboard_master:~/data-files# ./dd.sh
dd.sh (659): drop_caches: 3
16+0 records in
16+0 records out
65536 bytes (66 kB, 64 KiB) copied, 0.000583414 s, 112 MB/s

real    0m0.009s
user    0m0.006s
sys     0m0.002s
```

Figura 83. Escritura de fichero local de 66 kB

```
root@zedboard_master:~/data-files# ./dd.sh
dd.sh (662): drop_caches: 3
16384+0 records in
16384+0 records out
67108864 bytes (67 MB, 64 MiB) copied, 0.371679 s, 181 MB/s

real    0m0.380s
user    0m0.053s
sys     0m0.326s
```

Figura 84. Escritura de fichero local de 67 MB

## 9.6. CONCLUSIONES

En este capítulo se ha presentado el análisis de las aplicaciones en cuanto a la gestión de memoria del sistema. Se ha presentado que se cumple con el correcto manejo de la memoria, no dejando bloques de memoria inaccesible o punteros sin borrar. Por otro lado, se analizan los anchos de bandas posibles para distintos tamaños de fichero tanto para el estándar de conectividad Ethernet como para la comunicación con la tarjeta SD, evidenciando las dependencias de los anchos de banda en función de determinados parámetros, por lo que las especificaciones del fabricante deben interpretarse bajo un conjunto de restricciones determinadas. El diseñador debe evaluar sus restricciones antes de proceder a una implementación definitiva del SoC.

# CAPÍTULO 10. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se presentan las conclusiones extraídas tras la finalización del desarrollo e implementación de la plataforma *software*. Por último, se exponen diversas líneas de trabajos futuros sobre el proyecto desarrollado.

## 10.1. CONCLUSIONES

Durante el desarrollo de este Trabajo Fin de Máster se han estudiado y adoptado todas aquellas tecnologías que se han considerado las más adecuadas para el proyecto planteado, obteniendo un resultado satisfactorio.

En primer lugar, ha sido necesario un estudio del equipo dado por los solicitantes del proyecto. Conocer sus componentes, modo de funcionamiento, uso y objetivos planteados.

Conociendo los puntos anteriores, se plantea la metodología de trabajo, lenguajes de programación a utilizar y las posibles configuraciones *software* que permite la placa sobre la que se construye el proyecto, la ZedBoard de Xilinx.

Con los objetivos definidos, se plantea desde el principio que el sistema se basará en una configuración AMP. Esto permite el uso de las dos CPU de la placa y se consigue que las CPU trabajen de forma independiente pero coordinadas mediante el paso de mensajes. Para esta implementación se valoran diferentes soluciones como: B*aremetal*-Linux*,* Linux-FreeRTOS, Xenomai, etc.

Tras un estudio y diferentes pruebas, se escoge el binomio formado por una aplicación *baremetal* y un sistema operativo Linux. Esto se debe a que el dominio *baremetal* (CPU1) permite un acceso directo, rápido y eficaz al *hardware*, permitiendo una mayor

eficiencia a la hora de acceder a los datos. Por otro lado, el dominio Linux (CPU0) permite tener aplicaciones de alto nivel que facilitan la gestión de la plataforma de una forma cómoda para el usuario final y una mejor administración de los datos capturados.

Esta configuración permite definir con claridad qué *software* se ejecuta en cada CPU, permitiendo cumplir requisitos exigentes de tiempo real en la captura de datos y otros procesos. El dominio *baremetal* controla la captura de los datos y se comunica con la parte Linux para el tratamiento de estos. A su vez, este último gestiona los recursos del sistema de forma óptima y los maneja de una forma responsable según el estado en el cual se encuentre el sistema. Esto ligado a la inclusión de políticas de optimización, como el vaciado de la caché, permiten disponer del mayor número de recursos disponibles en cada instante.

De manera incremental y validando en el *hardware* a medida que este iba evolucionando, se han desarrollado e implementado de manera exitosa las aplicaciones deseadas para manejar la configuración del sistema y el comportamiento de este. Una aplicación principal que actúa como servidor, permite de manera intuitiva y visual configurar el comportamiento del sistema por diversos clientes, pudiendo establecer *scripts* para determinar un comportamiento específico escogido por el científico que maneje el equipo.

A su vez junto con la aplicación anterior, se ha creado una aplicación cliente con la premisa principal de la simplicidad, que permite que pueda ejecutarse en equipos de muy bajos recursos y controlar el equipo remotamente desde diferentes lugares.

Por otro lado, se ha diseñado una aplicación que controla en todo momento el estado de la alimentación actuando como protocolo de seguridad frente a un corte del suministro eléctrico. Asegurando el apagado en cascada del equipo de forma controlada y previendo la pérdida de datos o corrupción de estos cortando todos aquellos procesos relacionados con la captura de datos.

Por último, estudiando las limitaciones del sistema en cuanto a los recursos disponibles, se plantea un mecanismo que permita extraer los datos capturas mediante uno de los conectores de la placa. Para esta tarea, se realizan diferentes pruebas guardando

los datos en una memoria USB, SSD y enviando los datos mediante Ethernet, obteniendo esta última los mejores resultados. Por ello, se configura un sistema basado en NFS para salvaguardar los datos capturados.

Estos datos son capturados por la CPU1 y guardados y enviados por la CPU0, se envían a un equipo externo mediante el montaje de carpetas remotas, permitiendo la captura de datos durante un largo periodo de tiempo. Los ficheros capturados se clasifican con marcas de tiempo tanto en su nombre, como en las carpetas en las que se guardan, de esta forma se pueden distinguir los intervalos de captura y se realiza un mejor tratamiento de los datos. Este sistema permite cumplir las especificaciones objetivo en cuanto a la velocidad con la que se capturan los datos, pudiéndolos servir en el tiempo establecido. Todo esto proporcionando una manera segura de obtener a la información, almacenándola en un sistema externo al *hardware* del proyecto.

Por último, aunque es difícil realizar una comparación con otros sistemas, debido a que este equipo es una aplicación particular se presenta en Tabla 8 una tabla comparando diversos sistemas.

Tabla 8. Tabla comparativa entre proyectos

| Fichero | FPGA | Protocolo Red | Dominio |
|---|---|---|---|
| Este TFM | ZedBoard | TCP | *Baremetal* / Linux |
| [8] | Xilinx XCV300 device | N/A | *Baremetal* / Linux externo |
| [6] | Trenz TE0720 | TCP | *Baremetal* |
| [10] | Zynq XC7Z030 | TCP | *Baremetal* |
| [9] | Zynq-7000 | TCP/UDP | *Baremetal* |
| [7] | Xilinx XCKU115-FLVF1924 | TCP | *Baremetal* / Linux externo |

## 10.2. TRABAJOS FUTUROS

Durante el desarrollo y mediante la experiencia obtenida se sugieren algunos trabajos futuros:

1. Adaptar el *software* para incluir nuevas funcionalidades o adaptarse a los cambios que puedan sufrir la arquitectura. Debido a que pueden surgir cambios en la arquitectura del sistema, el *software* deberá modificarse para adaptarse a estos cambios (servidor, conversor, cliente, etc.).

2. Investigar diferentes técnicas y mecanismos para una mejor implementación de la salida de datos. Aunque el sistema NFS funciona correctamente, está cerca de alcanzar el punto crítico del cuello de botella. Por ello, se propone buscar aquellos mecanismos que permitan mejorar las funcionalidades de este como la inclusión de *jumbo frames* sobre Ethernet.

# BIBLIOGRAFÍA


[1] IAC, "QUIJOTE | Instituto de Astrofísica de Canarias • IAC." [Online]. Available: https://www.iac.es/es/proyectos/quijote. [Accessed: Jun. 09, 2022]

[2] Avnet Electronics Marketing, "ZedBoard Zynq™ Evaluation and Development Hardware User's Guide," 2012 [Online]. Available: https://digilent.com/reference/_media/zedboard:zedboard_ug.pdf. [Accessed: Mar. 07, 2022]

[3] Xilinx Inc., "Field Programmable Gate Array (FPGA)." [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. [Accessed: Jul. 22, 2019]

[4] R. Elliot Crockett, Luis, L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book*, 1st Editio. Glasgow: Strathclyde Academic Media, 2014.

[5] Xilinx Inc., "Zynq-7000 SoC First Generation Architecture," 2018. [Online]. Available: www.xilinx.com. [Accessed: Jul. 22, 2019]

[6] G. A. Caceres *et al.*, "VerDAQ: a Versatile Data AcQuisition system for high energy physics experiments," *Journal of Instrumentation*, vol. 17, no. 01, p. P01023, Jan. 2022, doi: 10.1088/1748-0221/17/01/P01023. [Online]. Available: https://iopscience.iop.org/article/10.1088/1748-0221/17/01/P01023. [Accessed: Dec. 18, 2022]

[7] A. P. Putra, S. Fuada, Y. Aska, and T. Adiono, "System-on-Chip architecture for high-speed data acquisition in visible light communication system," *2016 International Symposium on Electronics and Smart Devices, ISESD 2016*, pp. 63–67, Mar. 2017, doi: 10.1109/ISESD.2016.7886693.

[8] Y. Abhyankar, C. Sajish, P. Kulkarni, and C. R. Subrahmanya, “Design of a FPGA based data acquisition system for radio astronomy applications,” *Proceedings of the International Conference on Microelectronics, ICM*, pp. 555–557, 2004, doi: 10.1109/ICM.2004.1434723.

[9] X. Pei *et al.*, “Design of a multi-function high-speed digital baseband data acquisition system,” *Res Astron Astrophys*, vol. 21, no. 10, p. 248, Nov. 2021, doi: 10.1088/1674-4527/21/10/248. [Online]. Available: https://iopscience.iop.org/article/10.1088/1674-4527/21/10/248. [Accessed: Dec. 18, 2022]

[10] F. Capel *et al.*, “Mini-EUSO data acquisition and control software,” *J Astron Telesc Instrum Syst*, vol. 5, no. 04, p. 1, Jul. 2019, doi: 10.1117/1.JATIS.5.4.044009. [Online]. Available: http://arxiv.org/abs/1907.04938. [Accessed: Dec. 18, 2022]

[11] Xilinx, “Vitis Software Platform.” [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html. [Accessed: Jun. 08, 2022]

[12] ARM, *Cortex-A9 MPCore Technical Reference Manual*, Sexta. ARM, 2008.

[13] Peter Marwedel, *Embedded System Design*, 2nd ed. 2011.

[14] R. Dennis and K. Brian, *The C Programming Language*. 1978.

[15] ISO C, “ISO/IEC JTC1/SC22/WG14 - C.” [Online]. Available: https://www.open-std.org/jtc1/sc22/wg14/. [Accessed: Nov. 11, 2022]

[16] ISO C++, “ISO/IEC JTC1/SC22/WG21 - The C++ Standards Committee - ISOCPP.” [Online]. Available: https://www.open-std.org/jtc1/sc22/wg21/. [Accessed: Nov. 11, 2022]

[17] Python Software Foundation, “Python.org.” [Online]. Available: https://www.python.org/downloads/. [Accessed: Nov. 11, 2022]

[18] Rust Team, “Rust Programming Language.” [Online]. Available: https://www.rust-lang.org/. [Accessed: Nov. 11, 2022]

[19] Oracle, “Java | Oracle.” [Online]. Available: https://www.java.com/es/. [Accessed: Nov. 11, 2022]

[20] Richard Murch, *The Software Development Lifecycle - A Complete Guide*. 2012.

[21] Guru99, “What is Functional Testing? Types & Examples (Complete Tutorial).” [Online]. Available: https://www.guru99.com/functional-testing.html. [Accessed: Jun. 08, 2022]

[22] Smartbear, “What Is Unit Testing? | SmartBear.” [Online]. Available: https://smartbear.com/learn/automated-testing/what-is-unit-testing/. [Accessed: Jun. 08, 2022]

[23] “Manifiesto por el Desarrollo Ágil de Software.” [Online]. Available: https://agilemanifesto.org/iso/es/manifesto.html. [Accessed: Jun. 01, 2022]

[24] Kanbanize, “Qué es Kanban: Definición, Características y Ventajas.” [Online]. Available: https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban. [Accessed: Jun. 01, 2022]

[25] Atlassian, “Qué es el control de versiones | Atlassian Git Tutorial.” [Online]. Available: https://www.atlassian.com/es/git/tutorials/what-is-version-control. [Accessed: Jun. 01, 2022]

[26] Git, “Git.” [Online]. Available: https://git-scm.com/. [Accessed: Jun. 01, 2022]

[27] Doxygen, “Doxygen: Doxygen.” [Online]. Available: https://www.doxygen.nl/. [Accessed: May 25, 2022]

[28] jothepro, “Doxygen Awesome: Doxygen Awesome.” [Online]. Available: https://jothepro.github.io/doxygen-awesome-css/. [Accessed: May 25, 2022]

[29] LaTeX, “LaTeX - A document preparation system.” [Online]. Available: https://www.latex-project.org/. [Accessed: May 25, 2022]

[30] Archlinux, “man page (Español) - ArchWiki.” [Online]. Available: https://wiki.archlinux.org/title/Man_page_(Espa%C3%B1ol). [Accessed: May 25, 2022]

[31] Samtec, “VITA 57.1 Industry Standard Connectors and Systems | Samtec.” [Online]. Available: https://www.samtec.com/standards/vita/fmc/. [Accessed: Jun. 09, 2022]

[32] Samtec, “Samtec.” [Online]. Available: https://www.samtec.com/. [Accessed: Nov. 09, 2022]

[33] Samtec, “HDR-153514-XX.” [Online]. Available: https://4donline.ihs.com/images/VipMasterIC/IC/SAMI/SAMIS20113/SAMIS20113-1.pdf?hkey=6D3A4C79FDBF58556ACFDE234799DDF0. [Accessed: Nov. 09, 2022]

[34] A. Devices, “8-/4-Channel, 24-Bit, Simultaneous Sampling ADCs with Power Scaling, 110.8 kHz BW” [Online]. Available: www.analog.com. [Accessed: Jun. 09, 2022]

[35] Xilinx Inc., “ Zynq-7000 SoC Technical Reference Manual, vol. 585,” *2018*. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html. [Accessed: Nov. 11, 2022]

[36] Debian Handbook, “11.4. Servidor de archivos NFS.” [Online]. Available: https://debian-handbook.info/browse/es-ES/stable/sect.nfs-file-server.html. [Accessed: May 31, 2022]

[37] GeeksforGeeks, “Direct Access Media (DMA) Controller in Computer Architecture - GeeksforGeeks.” [Online]. Available: https://www.geeksforgeeks.org/direct-access-media-dma-controller-in-computer-architecture/. [Accessed: May 31, 2022]

[38] P. G. Smith, “ACCELERATED PRODUCT DEVELOPMENT: TECHNIQUES AND TRAPS,” in *The PDMA Handbook of New Product Development*, Second Edi., J. Wiley, Ed. 2004.

[39] M. Á. Sicilia, *Métricas del Mantenimiento de Software*. Houston, Texas, 2009.

[40] S. Mittal, “A survey of techniques for architecting and managing asymmetric multicore processors,” *ACM Comput Surv*, vol. 48, no. 3, Feb. 2016, doi: 10.1145/2856125.

[41] TechDifferences, “Difference Between Symmetric and Asymmetric Multiprocessing (with Comparison Chart) - Tech Differences,” 2016. [Online]. Available: https://techdifferences.com/difference-between-symmetric-and-asymmetric-multiprocessing.html. [Accessed: Oct. 08, 2019]

[42] Xilinx, “Zynq UltraScale+ FSBL - Xilinx Wiki - Confluence.” [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842019/Zynq+UltraScale+FSBL#ZynqUltraScale%2BFSBL-WhatisFSBL%3F. [Accessed: Jun. 01, 2022]

[43] Xilinx, “Multi-OS Support (AMP & Hypervisor) - Xilinx Wiki - Confluence.” [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841668/Multi-OS+Support+AMP+Hypervisor. [Accessed: Jun. 01, 2022]

[44] Digikey, “Sistemas operativos en tiempo real y sus aplicaciones | DigiKey.” [Online]. Available: https://www.digikey.es/es/articles/real-time-operating-systems-and-their-applications. [Accessed: May 31, 2022]

[45] FreeRTOS, “FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions.” [Online]. Available: https://www.freertos.org/. [Accessed: May 31, 2022]

[46] xenomai, “Start_Here · Wiki · xenomai / xenomai · GitLab.” [Online]. Available: https://source.denx.de/Xenomai/xenomai/-/wikis/Start_Here. [Accessed: May 31, 2022]

[47] “Zephyr Project - Zephyr Project.” [Online]. Available: https://www.zephyrproject.org/. [Accessed: Dec. 18, 2022]

[48] Zephyr OS, “Zephyr RTOS and Cortex-R5 on Zynq UltraScale+ - Zephyr Project.” [Online]. Available: https://www.zephyrproject.org/zephyr-rtos-and-cortex-r5-on-zynq-ultrascale-2/. [Accessed: Dec. 19, 2022]

[49] “RISC-V International.” [Online]. Available: https://riscv.org/. [Accessed: Dec. 18, 2022]

[50] Grupo Garatu, “BOM (Bill of materials) lista de componentes para la fabricación - Glosario Smart Factory.” [Online]. Available: https://development.grupogaratu.com/bom-bill-of-materials-componentes-fabricacion-producto/. [Accessed: Jun. 06, 2022]

[51] Oracle, “¿Qué es el Internet de las cosas (IoT)? | Oracle España.” [Online]. Available: https://www.oracle.com/es/internet-of-things/what-is-iot/. [Accessed: May 26, 2022]

[52] Colin Walls, “How to Choose an Embedded Operating System - Embedded Computing Design.” [Online]. Available: https://embeddedcomputing.com/technology/software-and-os/how-to-choose-an-embedded-operating-system. [Accessed: Jun. 06, 2022]

[53] Xilinx, “OpenAMP - Xilinx Wiki - Confluence.” [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841718/OpenAMP. [Accessed: Jun. 11, 2022]

[54] Xilinx and Inc, “Libmetal and OpenAMP User Guide.” 2020 [Online]. Available: www.xilinx.com. [Accessed: Mar. 09, 2022]

[55] G. Silberschatz, Abraham; Baer Galvin, Peter; Gagne, *Fundamentos de sistemas operativos, 7ª Edición*, Séptima Ed. Madrid: McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S. A. U., 2006.

[56] D. A. Rusling, “Chapter 9. The File system,” in *The Linux Kernel*, Version 0., Wokingham, 1999.

[57] J. Torres, “BSD sockets,” 2013. [Online]. Available: https://medium.com/jmtorres/bsd-sockets-7b50fccf71e8. [Accessed: Oct. 09, 2019]

[58] S. I. Inc., “The BSD UNIX Socket Library,” 1999. [Online]. Available: https://support.sas.com/documentation/onlinedoc/sasc/doc/lr2/lrv2ch15.htm. [Accessed: Oct. 09, 2019]

[59] G. K.-H. Jonathan Corbet, Alessandro Rubini, “An Introduction to Device Drivers,” in *Linux Device Drivers*, 3rd Editio., O’Reilly Media, 2005, p. 640.

[60] Xilinx Inc., “PetaLinux - Xilinx Wiki - Confluence,” *06-09-2019*, 2019. [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842250/PetaLinux. [Accessed: Sep. 12, 2019]

[61] Xilinx Inc., “PetaLinux Tools.” [Online]. Available: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html. [Accessed: Sep. 12, 2019]

[62] Xillybus Ltd., “Xillinux: A Linux distribution for Z-Turn Lite, Zedboard, ZyBo and MicroZed.” [Online]. Available: http://xillybus.com/xillinux. [Accessed: Sep. 12, 2019]

[63] Koheron, “Ubuntu Images for Zynq | Koheron,” 2022. [Online]. Available: https://www.koheron.com/software-development-kit/documentation/ubuntu-zynq/. [Accessed: Mar. 09, 2022]

[64] Yocto Project, “Welcome to the Yocto Project Documentation — The Yocto Project ® 4.0.1 documentation.” [Online]. Available: https://docs.yoctoproject.org/. [Accessed: May 26, 2022]

[65] Xilinx Inc, “Xilinx Github,” 2022. [Online]. Available: https://github.com/xilinx. [Accessed: Mar. 09, 2022]

[66] Xilinx Inc, “Embedded Linux,” 2022. [Online]. Available: https://support.xilinx.com/s/topic/0TO2E000000YKXXWA4/embedded-linux?language=en_US. [Accessed: Mar. 09, 2022]

[67] Xilinx Inc, “PetaLinux Tools Documentation Reference Guide,” p. 229, 2021 [Online]. Available: www.xilinx.com. [Accessed: Mar. 09, 2022]

[68] element14, “Accelerating PetaLinux BSP Build Time - Blog - FPGA - element14 Community.” [Online]. Available:

https://community.element14.com/technologies/fpga-group/b/blog/posts/accelerating-petalinux-bsp-build-time. [Accessed: May 30, 2022]

[69] Roy Messinger, “GPIO and Petalinux (Embedded Linux, Yocto based) - Hackster.io.” [Online]. Available: https://www.hackster.io/Roy_Messinger/gpio-and-petalinux-embedded-linux-yocto-based-c71773. [Accessed: May 30, 2022]

[70] Xilinx, “Customizing Device Trees in Xilinx Yocto - Xilinx Wiki - Confluence.” [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/61669922/Customizing+Device+Trees+in+Xilinx+Yocto?view=blog. [Accessed: May 26, 2022]

[71] Xilinx, “linux-xlnx/Documentation/devicetree/bindings at master · Xilinx/linux-xlnx.” [Online]. Available: https://github.com/Xilinx/linux-xlnx/tree/master/Documentation/devicetree/bindings. [Accessed: May 26, 2022]

[72] The Linux Kernel documentation, “Linux and the Devicetree — The Linux Kernel documentation.” [Online]. Available: https://www.kernel.org/doc/html/latest/devicetree/usage-model.html. [Accessed: May 26, 2022]

[73] Zach Pfeffer, “Use PetaLinux Tools to Add a Script that Will Execute at Boot.” [Online]. Available: https://www.zachpfeffer.com/single-post/execute-a-script-at-boot-using-petalinux-tools. [Accessed: Aug. 05, 2022]

[74] Linux, “minicom.” [Online]. Available: https://linux.die.net/man/1/minicom. [Accessed: Nov. 11, 2022]

[75] Linux manual page, “slabtop(1) - Linux manual page.” [Online]. Available: https://man7.org/linux/man-pages/man1/slabtop.1.html. [Accessed: Jun. 10, 2022]

[76] Kryptosolid, “Comando slabtop de Linux - Mostrar información de caché de losa del kernel.” [Online]. Available: https://conpilar.kryptonsolid.com/comando-slabtop-

de-linux-mostrar-informacion-de-cache-de-losa-del-kernel/. [Accessed: Jun. 10, 2022]

[77] Asis Rodríguez, “Software modular y reutilizable - industriaembebidahoy.com,” Dec. 14, 2018. [Online]. Available: https://www.industriaembebidahoy.com/software-modular-y-reutilizable/. [Accessed: May 26, 2022]

[78] M. Stannett, *Modular Software Design*. 1990.

[79] GeeksforGeeks, “Socket Programming in C/C++ - GeeksforGeeks.” [Online]. Available: https://www.geeksforgeeks.org/socket-programming-cc/. [Accessed: May 26, 2022]

[80] Brian Hall, “Beej’s Guide to Network Programming.” [Online]. Available: https://beej.us/guide/bgnet/html/. [Accessed: May 26, 2022]

[81] Douglas C. Schmidt and Stephen D. Huston, *C++ Network Programming, Volume 1: Mastering Complexity with ACE and Patterns*, vol. 1. 2001 [Online]. Available: https://learning.oreilly.com/library/view/c-network-programming/0201604647/. [Accessed: May 26, 2022]

[82] Boost C++ Libraries, “Boost C++ Libraries.” [Online]. Available: https://www.boost.org/. [Accessed: May 30, 2022]

[83] Vladimir Prus, “Chapter 31. Boost.Program_options - 1.71.0.” [Online]. Available: https://www.boost.org/doc/libs/1_71_0/doc/html/program_options.html. [Accessed: May 30, 2022]

[84] Evgenij Legotskoj, “Boost - Console program menu using boost::program_options.” [Online]. Available: https://evileg.com/en/post/422/. [Accessed: May 30, 2022]

[85] Baptiste Wicht, “Manage command-line options with Boost Program Options | Blog blog(‘Baptiste Wicht’);” [Online]. Available: https://baptiste-wicht.com/posts/2012/07/manage-command-line-boost-program-options.html. [Accessed: May 30, 2022]

[86] Boris Schäling, *Chapter 63. Boost.ProgramOptions*. [Online]. Available: https://theboostcpplibraries.com/boost.program_options. [Accessed: May 30, 2022]

[87] "std::map - cppreference.com." [Online]. Available: https://en.cppreference.com/w/cpp/container/map. [Accessed: May 30, 2022]

[88] Puneet Sapra, "What is Parsing?. A brief introduction to Parsing | by Puneet Sapra | The Mighty Programmer | Medium." [Online]. Available: https://medium.com/the-mighty-programmer/what-is-parsing-4012f997d265. [Accessed: May 30, 2022]

[89] G. C. J. H. J. Dick, *Parsing Techniques: A Practical Guide*. [Online]. Available: https://link.springer.com/book/10.1007/978-0-387-68954-8. [Accessed: Nov. 09, 2022]

[90] Mitchell Orsucci, "mitchellorsucci/libgpio: Recipe and source files for libgpio for Yocto builds." [Online]. Available: https://github.com/mitchellorsucci/libgpio. [Accessed: May 30, 2022]

[91] Ionel Bădișor and Mitchell Orsucci, "Digilent/libuio." [Online]. Available: https://github.com/Digilent/libuio. [Accessed: May 30, 2022]

[92] The Linux Kernel, "The Userspace I/O HOWTO — The Linux Kernel documentation." [Online]. Available: https://www.kernel.org/doc/html/v4.12/driver-api/uio-howto.html. [Accessed: May 30, 2022]

[93] Xilinx, "pg144-axi-gpio.pdf • Viewer • Documentation Portal." [Online]. Available: https://docs.xilinx.com/v/u/en-US/pg144-axi-gpio. [Accessed: May 30, 2022]

[94] Xilinx, "Working with GPIO and Embedded Linux." [Online]. Available: https://www.xilinx.com/video/soc/working-with-gpio-and-embedded-linux.html. [Accessed: May 30, 2022]

[95] Linux manual page, "sysfs(5) - Linux manual page." [Online]. Available: https://man7.org/linux/man-pages/man5/sysfs.5.html. [Accessed: May 30, 2022]

[96] Aaron Kili, “Explanation of ‘Everything is a File’ and Types of Files in Linux.” [Online]. Available: https://www.tecmint.com/explanation-of-everything-is-a-file-and-types-of-files-in-linux/. [Accessed: May 30, 2022]

[97] Linux manual page, “mmap(2) - Linux manual page.” [Online]. Available: https://man7.org/linux/man-pages/man2/mmap.2.html. [Accessed: May 30, 2022]

[98] “Accessing PCI Regions--Essential Linux Device Drivers--嵌入式linux中文站.” [Online]. Available: http://www.embeddedlinux.org.cn/essentiallinuxdevicedrivers/final/ch10lev1sec3.html. [Accessed: May 30, 2022]

[99] “Memory Alignment and Performance · Fylux.” [Online]. Available: https://fylux.github.io/2017/07/11/Memory_Alignment/. [Accessed: May 30, 2022]

[100] Balaji Gunasekaran, “Quad-SPI, Everything You Need To Know! – Embedded Inventor.” [Online]. Available: https://embeddedinventor.com/quad-spi-everything-you-need-to-know/. [Accessed: May 30, 2022]

[101] Linux manual page, “mem(4) - Linux manual page.” [Online]. Available: https://man7.org/linux/man-pages/man4/mem.4.html. [Accessed: May 30, 2022]

[102] Jan, “How to Design and Access a Memory-Mapped Device in Programmable Logic from Linaro Ubuntu Linux on Xilinx Zynq on the ZedBoard, Without Writing a Device Driver – Part One | FPGA CPU News.” [Online]. Available: http://fpga.org/2013/05/28/how-to-design-and-access-a-memory-mapped-device-part-one/. [Accessed: May 30, 2022]

[103] “Accessing physical addresses through devmem.” [Online]. Available: https://programmer.group/accessing-physical-addresses-through-devmem.html. [Accessed: May 30, 2022]

[104] Heejin Park, “Directly Access Your Physical Memory (dev/mem) - Heejin Park.” [Online]. Available: https://bakhi.github.io/devmem/. [Accessed: May 30, 2022]

[105] Boost, “Chapter 32. Boost.PropertyTree - 1.71.0.” [Online]. Available: https://www.boost.org/doc/libs/1_71_0/doc/html/property_tree.html. [Accessed: May 31, 2022]

[106] “How to use boost::property_tree to load and write JSON · Jeremy Cochoy.” [Online]. Available: http://www.cochoy.fr/boost-property-tree/. [Accessed: May 31, 2022]

[107] “Populating Boost Ptree JSON Arrays - The Unterminated String.” [Online]. Available: https://www.theunterminatedstring.com/boost-ptree-array/. [Accessed: May 31, 2022]

[108] Boris Schäling, *Chapter 25. Boost.PropertyTree*. [Online]. Available: https://theboostcpplibraries.com/boost.propertytree. [Accessed: May 31, 2022]

[109] “JSON.” [Online]. Available: https://www.json.org/json-es.html. [Accessed: May 31, 2022]

[110] Niels Lohmann, “nlohmann/json: JSON for Modern C++.” [Online]. Available: https://github.com/nlohmann/json. [Accessed: May 31, 2022]

[111] “I2C Bus.” [Online]. Available: https://www.i2c-bus.org/. [Accessed: May 31, 2022]

[112] Linux manual page, “open(2) - Linux manual page.” [Online]. Available: https://man7.org/linux/man-pages/man2/open.2.html. [Accessed: May 31, 2022]

[113] Linux manual page, “close(2) - Linux manual page.” [Online]. Available: https://man7.org/linux/man-pages/man2/close.2.html. [Accessed: May 31, 2022]

[114] Microchip, “MCP7941X” [Online]. Available: www.microchip.com. [Accessed: Jun. 11, 2022]

[115] Howard Hinnant, “date.” [Online]. Available: https://howardhinnant.github.io/date/date.html. [Accessed: May 31, 2022]

[116] Linux manual page, “reboot(2) - Linux manual page.” [Online]. Available: https://man7.org/linux/man-pages/man2/reboot.2.html. [Accessed: May 25, 2022]

[117] Refactoring, “Patrones de diseño / Design patterns.” [Online]. Available: https://refactoring.guru/es/design-patterns. [Accessed: Jun. 10, 2022]

[118] LInux manual page, “sys_socket.h(0p) - Linux manual page.” [Online]. Available: https://man7.org/linux/man-pages/man0/sys_socket.h.0p.html. [Accessed: May 25, 2022]

[119] LInux manual page, “arpa_inet.h(0p) - Linux manual page.” [Online]. Available: https://man7.org/linux/man-pages/man0/arpa_inet.h.0p.html. [Accessed: May 25, 2022]

[120] Boost, “Boost.Asio - 1.71.0.” [Online]. Available: https://www.boost.org/doc/libs/1_71_0/doc/html/boost_asio.html. [Accessed: May 26, 2022]

[121] GNU Compiler Collection, “GCC, the GNU Compiler Collection - GNU Project.” [Online]. Available: https://gcc.gnu.org/. [Accessed: May 25, 2022]

[122] Linux man page, “rsync(1) - Linux man page.” [Online]. Available: https://linux.die.net/man/1/rsync. [Accessed: May 31, 2022]

[123] Marc Wilson, “SCP - Secure Copy Protocol - What is it & Full Definition & Example Cmds!” [Online]. Available: https://www.pcwdld.com/what-is-scp. [Accessed: May 31, 2022]

[124] SSH, “SFTP protocol, clients, servers etc. Page by the original author of SFTP.” [Online]. Available: https://www.ssh.com/academy/ssh/sftp. [Accessed: May 31, 2022]

[125] Linux manual page, “top(1) - Linux manual page.” [Online]. Available: https://man7.org/linux/man-pages/man1/top.1.html. [Accessed: Aug. 08, 2022]

[126] UC3M, “6.10. Anomalías en la gestión de memoria en C.” [Online]. Available: https://www.it.uc3m.es/pbasanta/asng/course_notes/memory_leaks_es.html. [Accessed: Nov. 16, 2022]

[127] Oracle, “Java Garbage Collection Basics.” [Online]. Available: https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html. [Accessed: Nov. 16, 2022]

[128] Valgrind Developers, “Valgrind Home.” [Online]. Available: https://valgrind.org/. [Accessed: Nov. 16, 2022]

[129] Valgrind Developers, “Memcheck: a memory error detector.” [Online]. Available: https://valgrind.org/docs/manual/mc-manual.html. [Accessed: Nov. 16, 2022]

[130] Valgrind Developers, “Massif.” [Online]. Available: https://valgrind.org/docs/manual/ms-manual.html. [Accessed: Nov. 17, 2022]

[131] KDE Org, “Heaptrack.” [Online]. Available: https://apps.kde.org/es/heaptrack/. [Accessed: Nov. 17, 2022]

[132] Josef Weidendorfer, “KCachegrind.” [Online]. Available: https://kcachegrind.sourceforge.net/html/Home.html. [Accessed: Nov. 17, 2022]

[133] Linux, “dd.” [Online]. Available: https://man7.org/linux/man-pages/man1/dd.1.html. [Accessed: Nov. 18, 2022]

[134] P. Moorthy and N. Kapre, “A Case for Embedded FPGA-based SoCs in Energy-Efficient Acceleration of Graph Problems.”

[135] P. Moorthy and N. Kapre, “Zedwulf: Power-Performance Tradeoffs of a 32-node Zynq SoC cluster.”