Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN SISTEMAS ELECTRÓNICOS AVANZADOS

TRABAJO FIN DE MÁSTER

DISEÑO E IMPLEMENTACIÓN DE NUEVAS FUNCIONALIDADES EN EMULADOR DE CANAL RF SOBRE DISPOSITIVO CON MULTIPROCESAMIENTO ASIMÉTRICO Y LÓGICA PROGRAMABLE

Estudiante: Ros Marauri, Gorka

Director: Bidarte Peraita, Unai

Departamento: Tecnología Electrónica

Curso: 2022-2023

Fecha: Bilbao, 2 de marzo de 2023

Resumen Laburpena Abstract

En la industria 4.0, se quiere fomentar el uso de tecnologías inalámbricas. Sin embargo, debido a la naturaleza de la propagación inalámbrica, aún es difícil cumplir con los requisitos de fiabilidad, latencia y determinismo que ofrecen las comunicaciones cableadas. Para evaluar el funcionamiento de las comunicaciones inalámbricas, es necesario realizar pruebas de campo, lo que supone un aumento del coste total. Otra opción es hacer uso de un emulador de canal, ahorrando el coste. Este Trabajo de Fin de Máster se centra en el diseño e implementación de nuevas funcionalidades en un emulador de canal RF sobre una tarjeta de desarrollo con multiprocesamiento asimétrico y lógica programable. Se utiliza una tarjeta de la familia Zynq Ultrascale+ con un procesador Cortex A53 con un Linux para tareas generales y un Cortex R5 para tareas con requisitos de tiempo, haciendo uso del framework openAMP e implementando el emulador de canal en la FPGA de la tarjeta. Se realiza un análisis de las tecnologías relacionadas al emulador, al igual que el estudio del propio emulador. Se detalla el desarrollo de las nuevas funcionalidades desde el diseño hasta la evaluación mediante diferentes pruebas.

Palabras Clave: emulador de canal, Zynq Ultrascale+, openAMP.

4.0 industrian, haririk gabeko teknologien erabilera sustatu nahi da. Hala ere, haririk gabeko hedapenaren izaera dela eta, oraindik zaila da komunikazio kableatuek eskaintzen dituzten fidagarritasun, latentzia eta determinismo baldintzak betetzea. Haririk gabeko komunikazioen funtzionamendua ebaluatzeko, kanpo-probak egin behar dira, eta horrek kostu osoa handitzea dakar. Beste aukera bat kanal-emuladore bat erabiltzea da, kostua aurreztuz. Master Amaierako Lan hau multiprozesamendu asimetriko eta logiko programagarria duen garapen-txartel baten gainean RF kanaleko emuladore batean funtzionalitate berriak diseinatzean eta inplementatzean oinarritzen da. Zynq Ultrascale+ familiako txartel bat erabiltzen da, Cortex A53 prozesadore batekin, Linux batekin, zeregin orokorretarako, eta Cortex R5 txartel batekin, denbora-eskakizunak dituzten zereginetarako, openAMP framework-a erabiliz eta txartelaren FPGAn kanal-emuladorea inplementatuz. Emuladorearekin lotutako teknologien azterketa egiten da, baita emuladorearen beraren analisia ere. Funtzionalitate berrien garapena zehazten da, diseinutik hasi eta proba desberdinen bidezko ebaluaziora arte.

Gako-hitzak: kanal-emuladorea, Zynq Ultrascale+, openAMP.

In Industry 4.0, the use of wireless technologies is to be encouraged. However, due to the nature of wireless propagation, it is still difficult to meet the reliability, latency and determinism requirements offered by wired communications. To evaluate the performance of wireless communications, field testing is required, which increases the overall cost. Another option is to make use of a channel emulator, saving the cost. This Master's Thesis focuses on the design and implementation of new functionalities in an RF channel emulator on a development card with asymmetric multiprocessing and

programmable logic. A Zynq Ultrascale+ family board with a Cortex A53 processor is used with a Linux for general tasks and a Cortex R5 for time-sensitive tasks, making use of the openAMP framework and implementing the channel emulator on the board's FPGA. An analysis of the technologies related to the emulator is carried out, as well as the study of the emulator itself. The development of the new functionalities is detailed from the design to the evaluation by means of different tests.

Keywords: channel emulator, Zynq Ultrascale+, openAMP.

Índice

Lista de figuras

Lista de tablas

Lista de acrónimos

ADC Analog Digital Converter

AMP Asymmetric Multiprocessing

API Application Programming Interface

APU Application Processing Unit

ASIC Application-Specific Integrated Circuit

ASSP Application Specific Standard Product

AXI Advanced eXtensible Interface

CCI Cache Coherent Interconnect

CLB Configurable Logic Blocks

CPU Central Processing Unit

DAP Debug Access Port

DDR Double Data Rate

DMA Direct Memory Access

ECC Error-Correcting Code

ETC Ericsson Texture Compression

FF Flip-FLop

FPGA Field Programmable Gate Array

FPU Floating Point Unit

GIC General Interrupt Controller

GPU Graphic Processing Unit

IF Intermediate Frequency

IP Intellectual Property

IWSN Industrial Wireless Sensor Network

LCM Life Cycle Management

LUT Look-up Table

MMU Memory Management Unit

MPSoC Multi-Processor System on Chip

MPU Memory Protection Unit

OCM On-chip Memory

openAMP Open Asymmetric Multiprocessing

PL Programmable Logic

PS Processing System

RAM Random Access Memory

RF Radio Frequency

RGBA Red Green Blue Alpha

ROM Read Only Memory

RPU Real-time Processing Unit

RTOS Real Time Operating System

SIMD Single Instruction, Multiple Data

SMP Symmetric Multiprocessing

SO Sistema Operativo

SoC System On-chip

SPI Serial Peripheral Interface

TCM Tightly-Coupled Memory

VFP Virtual Floating Point

1. Introducción

Actualmente, gracias a los nuevos dispositivos inteligentes y procesos de automatización, se está dando una nueva revolución en el sector industrial, la industria 4.0. Esta cuarta revolución esta marcada por la aparición de nuevas tecnologías, así como *Internet of Things, Inteligencia Artificial, Big Data, etc* .

Dentro de este cambio, se espera que las comunicaciones inalámbricas desempeñen un papel importante. Se espera que muchas de las redes de comunicaciones industriales cableadas sean sustituidas por redes inalámbricas, ya que estas últimas ofrecen varias ventajas: mayor escalabilidad, mejor eficiencia, menor coste de despliegue, etc.

Un ejemplo de la importancia de las comunicaciones inalámbricas en la industria son las Redes de Sensores Inalámbricas Industriales (Industrial Wireless Sensor Network). Gracias a los avances en la tecnología IWSN, la realización de sistemas de automatización industrial embebidos de bajo coste se ha convertido en algo factible. En estos sistemas, unos pequeños nodos de sensores son instalados en cada equipo industrial para monitorizar los parámetros críticos que afectan a la eficiencia y fiabilidad para control predictivo de cada uno. Esta información es transmitida desde los equipos a un nodo que procesa esa información y analiza los datos de cada sensor.

Para poder desarrollar este tipo de tecnologías, y que estas sustituyan a las tecnologías basadas en comunicaciones cableadas, es totalmente necesario que cumplan una serie de requisitos en relación con cada caso de uso. Ejemplos de diferentes tipos de aplicaciones son de tiempo real, de transmisión alta de datos o de seguridad funcional. Estos requisitos tratan sobre tiempos de latencia, Quality of Service, transmisión de datos, redundancia de paquetes, seguridad, detección de errores, etc [14].

Para cumplir todos estos requerimientos, se debe tener en cuenta que el medio de propagación inalámbrico, también conocido como canal de comunicaciones inalámbrico, es diferente al cableado y ofrece una serie de desventajas en relación con la comunicación. En un entorno industrial, la señal de radiofrecuencia propagada por el canal se refleja, difracta y dispersa en los objetos circundantes, y llega al receptor como una superposición de múltiples copias atenuadas, retrasadas y desplazadas en fase y/o frecuencia de la señal original [15]. Esta superposición de múltiples copias de la señal transmitida es un fenómeno conocido como multipath, y es un problema desde el punto de vista de las comunicaciones. Este multipath da lugar a desvanecimientos y fluctuaciones en la intensidad de la señal recibida que afectan gravemente a la velocidad y fiabilidad de la comunicación. Además, este no es el único efecto por el que las señales inalámbricas se ven alteradas, el ruido y las interferencias son efectos a tener en cuenta con igual importancia.

Debido a las razones anteriores, en aplicaciones donde se quiere implementar una comunicación inalámbrica, es necesario validar los sistemas desarrollados sobre entornos

reales, con el objetivo de que se cumplan los requisitos anteriormente mencionados para cada tipo de aplicación. Estas validaciones normalmente suelen ser costosas y conllevan un largo tiempo. Para evitar esto, se puede realizar una validación del sistema inalámbrico en un entorno aproximado al real mediante un emulador de canal.

Los emuladores de canales de radio sustituyen el canal de radio del mundo real entre un transmisor de radio y un receptor, proporcionando una representación difuminada de una señal transmitida a la entrada del receptor. Los emuladores de canales de radio permiten crear modelos matemáticos que representan el medio físico de transmisión de señales de radio. Estos emuladores generan una distorsión digital en las señales transmitidas. Esta distorsión está basada en los efectos que sufre una señal por el medio de propagación inalámbrico, es decir, por los efectos explicados al inicio del apartado. De esta manera, se pueden realizar medidas aproximadas acerca de un sistema de comunicaciones, y validar su funcionamiento.

En este trabajo se van a realizar mejoras y añadir nuevas funcionalidades a la tarjeta de procesamiento y control de un emulador de canal ya implementado. El dispositivo principal es la tarjeta ZCU102 de la familia Zynq Ultrascale+ de Xilinx.

1.1. Contextualización

Este Trabajo Fin de Máster (TFM) se lleva a cabo durante una cooperación educativa con la empresa Ikerlan, S. Coop [16]. Concretamente, en el área de Hardware y Comunicaciones, dentro del equipo de Sistemas de Comunicaciones.

El equipo de Sistemas de Comunicaciones de Ikerlan ha estado investigando durante años cómo integrar comunicaciones inalámbricas en la industria. A pesar de los avances en esta área, todavía es un desafío garantizar la fiabilidad requerida en entornos industriales debido a la naturaleza de la propagación inalámbrica.

Dentro de las tecnologías de comunicaciones inalámbricas, en el equipo se ha investigado en especial sobre la integración de la tecnología 5G en la industria 4.0[17], así como la integración de la tecnología WiFi en aplicaciones industriales con requisitos de alta fiabilidad y baja latencia [18].

Para investigar sobre estas tecnologías, el grupo cuenta con un emulador de canal RF propio desarrollado en Ikerlan. El emulador se utiliza principalmente para validar las tecnologías Wireless que se desarrollan en el grupo. Antes de realizar una validación de campo, el sistema es verificado mediante el emulador, lo que permite detectar fallos y problemas en etapas anteriores y disminuir el tiempo de desarrollo y el coste[19] . La primera versión del emulador fue desarrollada en 2013. Desde entonces, se ha seguido desarrollando mediante estudiantes en prácticas, proyectos TFG y TFM.

1.2. Descripción del proyecto

Este trabajo, continua el desarrollo de la tarjeta de procesamiento y control de un emulador de canal ya en funcionamiento. La tarjeta sobre la que se va a trabajar es la ZCU102 de Xilinx. Esta tarjeta tiene como elemento principal un MPSoC de Xilinx, concretamente la Zynq Ultrascale+. Este MPSoC es un sistema de multiprocesamiento asimétrico que cuenta con dos tipos de procesadores, cada uno con varios cores: Cortex-

A53 (para tareas generales) y Cortex R5 (para tareas con requisitos de tiempo real). El emulador implementado es un emulador RF de hasta 6 canales, con 10 paths por canal, capaz de cubrir las bandas frecuenciales entre 400MHz y 6GHz, gracias al Front-End RF usado. Por tanto, el emulador de canal se puede separar en dos grandes bloques: por un lado, el MPSoC (ZCU102), donde se implanta la aplicación del emulador, y por otra parte el Front-End. El esquema general del emulador se puede observar en la figura 1.

El sistema implementado en la Zynq Ultrascale+, consta de dos sistemas procesadores (A53 y R5) y la parte FPGA. Para la comunicación entre procesadores, se emplea openAMP (Open Asymmetric Multi-Processing), un framework que proporciona los componentes de software necesarios para permitir el desarrollo de aplicaciones de software para sistemas de muli-procesamiento asimétrico. El sistema consta de lo siguiente:

- Una aplicación Linux ejecutándose en uno de los cores Cortex-A53, en la que se ejecuta la interfaz de control del emulador de canal, con la que configurar los parámetros de los canales y los parámetros de los Front-End.
- La aplicación que genera los coeficientes necesarios para la emulación de canal sobre un core Cortex R5. Esta aplicación recibe los mensajes de configuración del Cortex-A53.
- En la parte lógica del MPSoC se ha implementado el emulador de canal. Recibe del Cortex R5 los coeficientes necesarios para la emulación mediante el bus AXI.

El Front-End RF está formado por cuatro microcontroladores EFM32-Gecko. Estos microcontroladores están controlados por el Cortex R5 mediante comunicación SPI. Cada microcontrolador configura los parámetros de los siguientes 4 elementos programables:

- Una atenuador variable, con la cual poder realizar la emulación de un perfil de atenuaciones o dejar un valor de atenuación equivalente a un pathloss especifico.
- Un amplificador de IF (Intermediate Frecuency), con el que optimizar el nivel de potencia de entrada a los ADC del MPSoC.
- Dos osciladores, con los cuales realizar el Downconversion o Upconversion de la señal a la frecuencia IF de adquisición de los ADC.

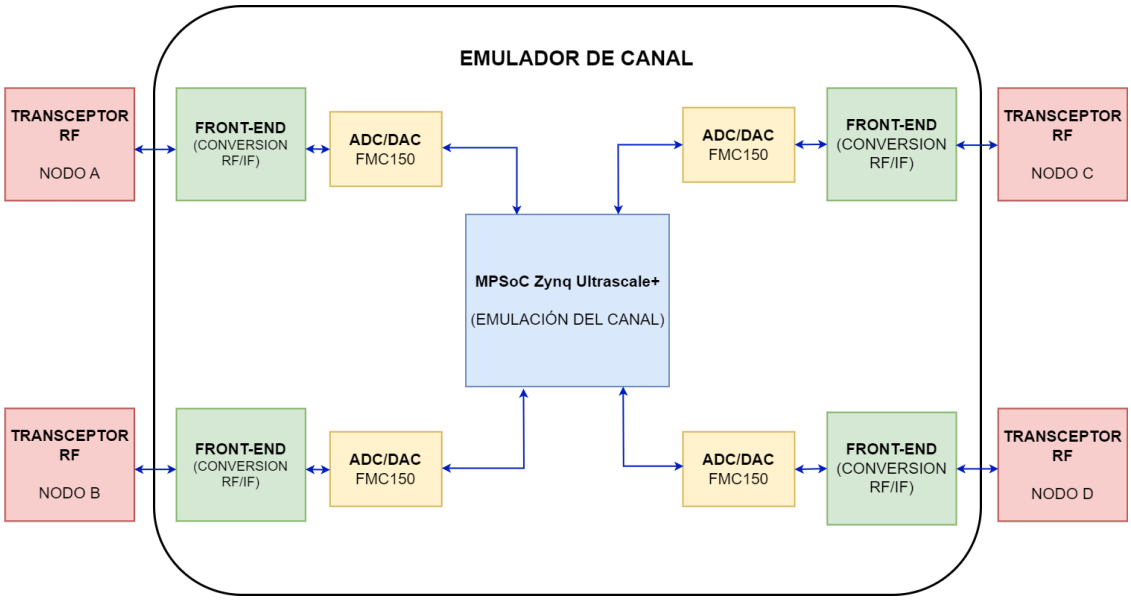


Figura 1: Esquema general del emulador de canal.

1.3. Objetivos y alcance

El objetivo principal de este proyecto es **diseñar e implementar nuevas funcionalidades a la tarjeta de procesado y control de un emulador de canal**. Al diseñar e implementar las nuevas funcionalidades, este proyecto servirá para considerar el emulador de canal como una herramienta de validación en los futuros desarrollos inalámbricos dentro del grupo de investigación de Ikerlan. Los objetivos de este proyecto se definen a continuación:

- **Analizar el estado del arte:** Estudiar las características de las comunicaciones inalámbricas y los emuladores de canal, así como el funcionamiento del MPSoC Zynq Ultrascale+ y el uso de multiprocesamiento en sistemas embebidos.
- **Analizar el emulador de Ikerlan:** Estudiar el trabajo previo del emulador, analizar sus partes y su funcionamiento. Observar el código de las aplicaciones de los procesadores Cortex A53 y R5, así como el diseño del emulador implementado en la FPGA.
- **Desarrollar las nuevas funcionalidades del emulador:** Desarrollar nuevas funcionalidades al emulador de canal con el objetivo de mejorar el funcionamiento. Se pretenden añadir dos nuevas funcionalidades al emulador:
  - **Lectura e interpretación de un fichero binario Matlab de parámetros de canal para la reproducción offline:** Realizar cambios para habilitar la carga de un archivo con parámetros predefinidos, sin necesidad de añadir los parámetros a mano mediante la interfaz de usuario. El principal trabajo será crear un archivo que contenga los parámetros, y este sea leído por parte del procesador Cortex-A53. A su vez, el A53 enviará los datos del fichero al procesador Cortex-R5, para que este los aplique en la FPGA. Esta tarea incluye programación en C, comunicación entre procesadores mediante openAMP y uso de Linux embebido (Petalinux).

- **Estimador de potencia para ajuste automático de la ganancia:** En el sistema actual, la ganancia de los amplificadores del Front-End es introducida manualmente. Se pretende añadir un estimador de potencia en la parte de la lógica programable, con el objetivo de que el ajuste de la ganancia de los Front-End sea automático. El trabajo se centrará en el desarrollo del estimador mediante System Generator, donde se generará una IP que será añadida al diseño de la FPGA mediante Vivado. El Cortex-R5 será el encargado de leer los valores del estimador y de actuar sobre los elementos del Front-End (amplificadores y atenuadores).

2. Estado del arte

2.1. Zynq Ultrascale+

El sistema desarrollado en este trabajo se basa en una tarjeta de procesamiento MPSoC+FPGA de la familia Zynq Ultrascale+ de Xilinx. Esta tarjeta de desarrollo cuenta con dos procesadores; un procesador de cuatro núcleos para aplicaciones de propósito general de alto rendimiento y otro procesador de dos núcleos para aplicaciones con requisitos de tiempo real. Ambos procesadores están basados en la arquitectura ARM. Además, el dispositivo cuenta con sistema de lógica programable. Gracias a esta combinación, estos dispositivos ofrecen la flexibilidad y escalabilidad de una FPGA, al tiempo que ofrece el rendimiento, la potencia y la facilidad de uso que suelen asociarse a los ASIC (Application-Specific Integrated Circuit) y ASSP (Application Specific Standard Product).

La arquitectura de la Zynq Ultrascale+ permite alcanzar niveles de rendimiento del sistema muy altos y eficientes, con un procesamiento inteligente. Además, Los dispositivos basados en la arquitectura UltraScale+ abordan un amplio espectro de requisitos de sistemas de gran ancho de banda y alto aprovechamiento mediante el uso de técnicas innovadoras como el enrutamiento de señales de nueva generación, la sincronización de reloj similar a la de los ASIC, las tecnologías de SoC multiprocesador y las nuevas funciones de reducción de potencia [2].

En este trabajo, se ha utilizado la tarjeta de procesamiento ZCU102[10]. A continuación, se detallan las partes más importantes de la tarjeta diferenciando los dos bloques principales: PS y PL. Además, en la figura 2 se puede observar un diagrama general de los componentes de la arquitectura de la familia Ultrascale+.

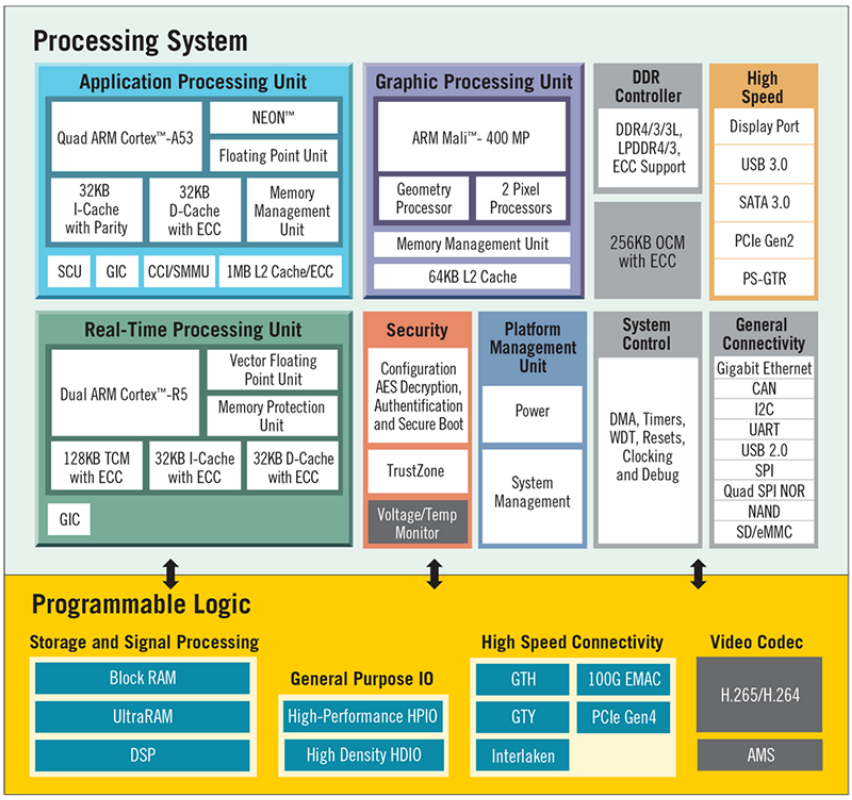


Figura 2: Diagrama general de los componentes de la arquitectura de la Zynq Ultrascale+[1].

2.1.1. Processing System (PS)

PS actúa como un SoC autónomo y es capaz de arrancar y soportar todas las características del sistema de procesamiento. Este sistema de procesamiento se puede dividir en tres grandes subsistemas: la APU, RPU y GPU. Además de estos grandes subsistemas dentro de PS se encuentran otras funciones de procesamiento como la memoria, los controladores periféricos y las entradas/salidas (E/S).

2.1.1.1. Application Processing Unit (APU)

La APU, Application Processing Unit, está formada por cuatro procesadores ARM Cortex A53 MPCore con dos memorias caché nivel L1 independientes para cada núcleo, y otra memoria cache nivel L2 compartida entre todos los núcleos. Los procesadores A53 son los procesadores de arquitectura ARM-v8 de mayor eficiencia energética, con la capacidad de soportar código de 32 y 64 bits. Además, utiliza una canalización en orden de 8 etapas de alta eficiencia equilibrada con técnicas avanzadas de obtención y acceso a datos para mejorar el rendimiento[2]. En la figura 3 se puede observar un diagrama de la arquitectura de la APU.

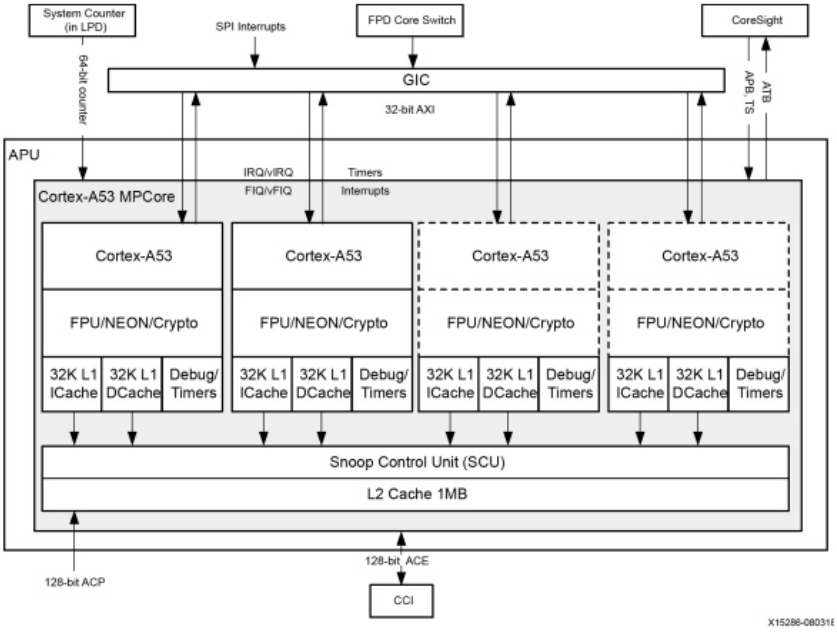


Figura 3: Diagrama de bloques de la APU[2].

Cada procesador Cortex-A53 incluye las siguientes características:

- Estados de ejecución AArch32 y AArch64 para arquitecturas de 32 y 64 bits.
- Todos los niveles de excepción (EL0, EL1, EL2 y EL3) en cada estado de ejecución.
- Conjunto de instrucciones de arquitectura ARM v8-A que incluye SIMD (Single Instruction Multiple Data) avanzado, extensiones de coma flotante (VFPv4) y extensiones para aplicaciones de criptografía.
- Cachés L1 independientes de 32 KB para instrucciones y datos.
- Unidad de gestión de memoria (MMU) de dos etapas (hipervisor y huésped).
- La CPU incluye un pipeline de 8 etapas con doble emisión simétrica de la mayoría de las instrucciones.
- 1 MB de memoria caché nivel L2.
- Accelerator coherency port (ACP).
- Interfaz maestra de extensión de coherencia AXI (ACE) de 128 bits hacia CCI(Cache Coherent Interconnect).
- Arquitectura Arm v8.
- Endianess configurable.
- Soporta virtualización de hardware que permite a múltiples entornos de software y sus aplicaciones acceder simultáneamente a las capacidades del sistema.
- Criptografía acelerada por hardware: rendimiento entre 3 y 10 veces superior al cifrado por software.
- El gran alcance de las direcciones físicas permite al procesador acceder a más de 4 GB de memoria física.

- La tecnología TrustZone garantiza una implementación fiable de las aplicaciones de seguridad.

2.1.1.2. Real-time Processing Unit (RPU)

La RPU, Real-Time Processing Unit, consiste en dos procesadores ARM Cortex-R5F para procesamiento en tiempo-real. EL procesador Cortex-R5F implementa la arquitectura ARM v7-R de 32 bits, e incluye una unidad de coma flotante ARM VFPv3.

En el procesador Cortex-R5F, la latencia de las interrupciones se mantiene baja, interrumpiendo y reiniciando las múltiples instrucciones de carga/almacenamiento. Esto se consigue con un puerto de periféricos dedicado que proporciona acceso de baja latencia al controlador de interrupciones y con puertos de memoria estrechamente acoplados para obtener accesos de baja latencia y deterministas a la RAM local. Habitualmente, el procesador Cortex-R5F se utiliza para muchas aplicaciones críticas de seguridad.[2]. En la figura 4 se puede observar el diagrama de bloques de la RPU.

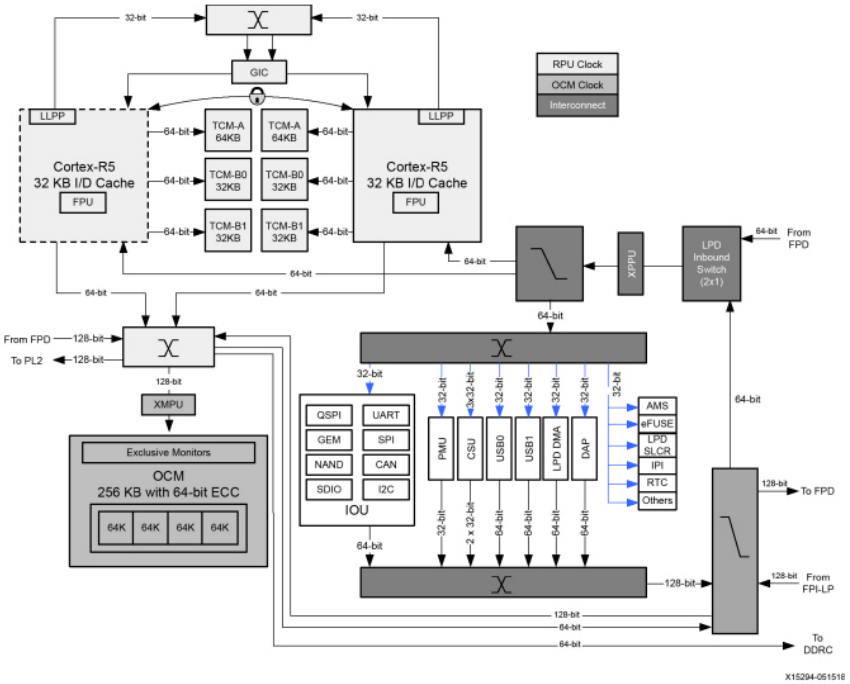


Figura 4: Diagrama de bloques de la RPU[2].

Cada procesador Cortex-R5 incluye las siguientes características:

- Conjunto de instrucciones de 32 bits con arquitectura Arm v7-R.
- FPU de precisión simple o doble con instrucciones VFPv3.
- Unidad de protección de memoria (MPU) de arquitectura Arm v7-R.
- Interfaz AXI3 maestra de 64 bits para acceder a la memoria y a los periféricos compartidos.
- Interfaz AXI3 esclava de 64 bits para acceso a DMA a las memorias TCM (Tightly-Coupled Memory).

- Bancos de memoria TCM separados de 128 KB con protección de errores ECC para cada TCM.
- Cachés L1 de instrucciones y datos de 32 KB con protección ECC.
- Configuración de forma independiente o redundante de ejecución de los procesadores.
- Interfaz APB de depuración a un puerto de acceso de depuración (DAP) CoreSight .
- Baja latencia de interrupción e interrupciones rápidas no enmascarables.
- Unidad de supervisión del rendimiento.
- Manejo de excepciones y protección de memoria.
- Detección y corrección de errores ECC en memorias de nivel L1.
- Built-in-self-test (BIST) para detectar fallos aleatorios en el hardware. Probablemente, causados por fallos permanentes.
- Watchdog para detectar fallos tanto sistemáticos como aleatorios que causan errores en el flujo del programa.

2.1.1.3. Graphic Processing Unit (GPU)

La GPU es un subsistema de gráficos 2D y 3D basado en el acelerador de hardware Arm® Mali -400 MP2. Tiene como componentes principales un procesador geométrico y dos procesadores de píxeles, además de un controlador de memoria cache compartida de nivel L2 de 64 KB. Cada procesador incorpora una unidad de manejo de la memoria (MMU). En la figura 5 se puede observar el diagrama de bloques de la GPU.

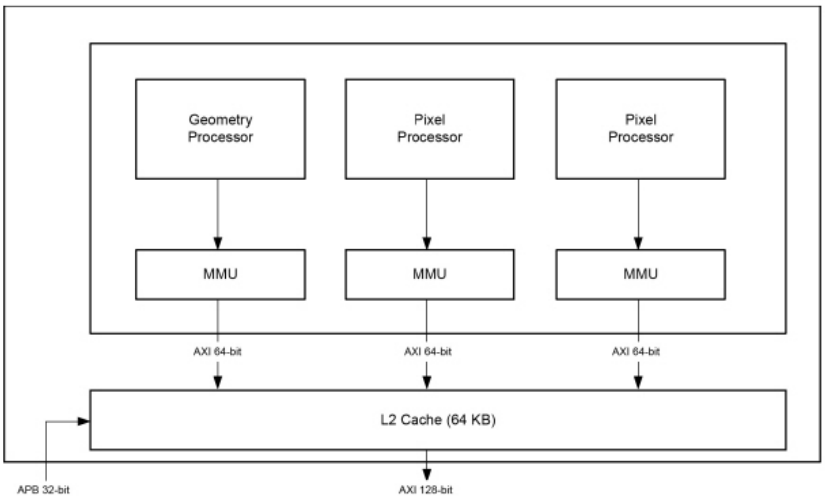


Figura 5: Diagrama de bloques de la GPU[2].

La GPU Mali 400 MP2 incluye las siguientes características:

- Es compatible con las APIs graficas Open GL ES 1.1 y 2.0 y Open VG 1.1.
- SIMD: aritmética de coma flotante de 32 bits y ejecución simultánea de 4 instrucciones de 32 bits.

- Unidad DMA del Vertex Loader.
- Alta tolerancia a la latencia de datos.
- Antialiasing 4x y 16x avanzado.
- Tamaño de texturas de hasta 4096 x 4096 píxeles.
- Compresión de texturas Ericsson (ETC) para reducir el ancho de banda de la memoria.
- Caché local para reducir el ancho de banda de la memoria.
- Diferentes formatos de texturas:
    - RGBA 8888, 565, 1556
    - Mono 8 y Mono 16
    - Formato YUV

2.1.2. Programmable Logic (PL)

La lógica programable (PL) es la segunda gran parte del sistema que compone la Zynq Ultrascale+. La FPGA que componen el subsistema está formada por lógica programable de propósito general, que está compuesta de "slices" y de CLBs (Configurable Logic Blocks), así como bloques de Entrada/Salida[20].

Estos son los principales componentes de la PL:

- **Configurable Logic Block (CLB):** los CLB son pequeñas agrupaciones regulares de elementos lógicos dispuestos en una matriz bidimensional en PL y conectados a otros recursos similares mediante interconexiones programables. Cada CLB está situado junto a una matriz de conmutación y contiene dos "slices"lógicas.
- **Slice:** Subunidad dentro del CLB que contiene recursos para implementar circuitos lógicos combinatorios y secuenciales. Las slices Zynq están compuestas por 4 LUTs, 8 flip-flops y otros elementos lógicos.
- **Look-up Table (LUT):** recurso flexible capaz de implementar:
    - Una función lógica de hasta seis entradas.
    - Una pequeña memoria de sólo lectura (ROM).
    - Una pequeña memoria de acceso aleatorio (RAM).
    - Un registro de desplazamiento.

  Las LUT pueden combinarse entre sí para formar funciones lógicas, memorias o registros de desplazamiento más grandes, según sea necesario.
- **Flip-flop (FF):** Elemento de circuito secuencial que implementa un registro de 1 bit, con función de reset. Uno de los FF puede utilizarse opcionalmente para implementar un "latch".
- **Matriz de conmutación:** junto a cada CLB se sitúa una matriz de conmutación que proporciona una función de enrutamiento flexible para realizar conexiones entre elementos dentro de un CLB y/o desde un CLB a otros recursos del PL.

- **Carry Logic:** los circuitos aritméticos requieren la propagación de señales intermedias entre cortes adyacentes, lo que se consigue mediante la "carry logic". Se comprende de una cadena de rutas y multiplexores para enlazar las slices en una columna vertical.
- **Bloques de entrada/salida (I/OB):** los I/OB son recursos que sirven de interfaz entre los recursos lógicos del PL y los "pads"del dispositivo físico utilizados para conectar circuitos externos. Cada I/OB puede manejar una señal de entrada o salida de 1 bit. Los I/OB suelen estar situados alrededor del perímetro del dispositivo.

En la figura 6 se pueden observar algunos de los componentes mencionados anteriormente. Además de la estructura general, hay varios componentes de propósito especial:

- Block RAMs (BRAM) para requisitos de memoria.
- DSP48E1 slices para operaciones aritméticas de alta velocidad.
- Transreceptores GTH y GTY para señales de alta velocidad: PCIe, 100 Gigabit Ethernet, USB3.0, etc.

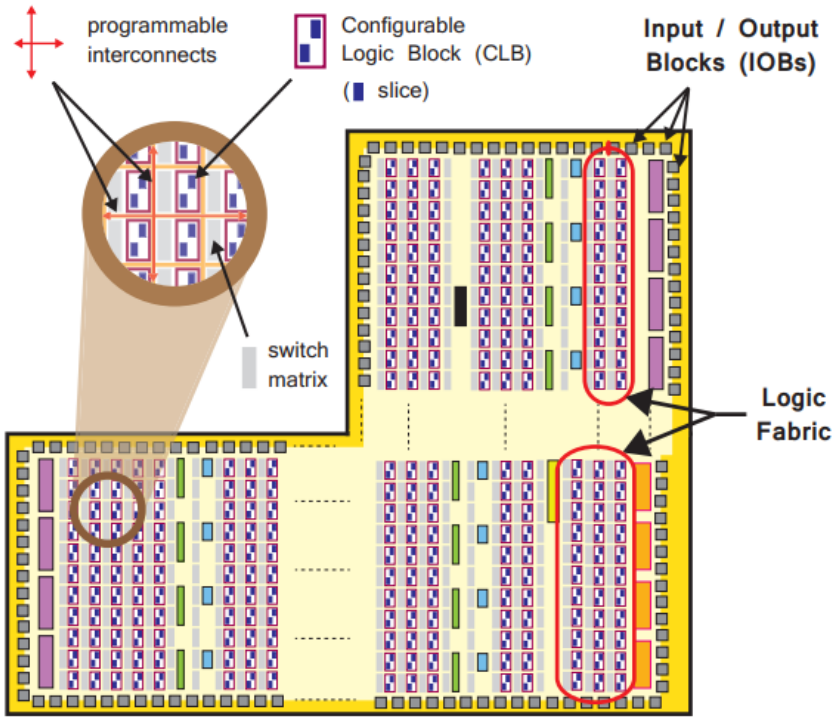


Figura 6: Diagrama de la PL y sus componentes.

2.2. Multiprocesamiento en Sistemas Embebidos

Desde hace unos años, los sistemas embebidos incorporan más de un núcleo en sus procesadores. No solamente en ordenadores de uso general, sino en especial, en los sistemas situados en entornos industriales donde es necesario no únicamente procesadores de bajo consumo y alto rendimiento, sino también la capacidad de aglutinar varias aplicaciones en un único chip, SoC (System on Chip); sistema de control, procesado de señal, comunicaciones, gráficos, visualización de imagen, etc.

Para poder conseguir este tipo de sistemas SoC, es necesario integrar varios tipos diferentes de sistemas operativos para hacer funcionar distintos dispositivos de hardware. Por ejemplo, Linux embebido admite varias tecnologías de comunicación y visualización de gráficos e imágenes. Por otro lado, el control de sistemas de tiempo real y el procesamiento de señales se realizan mediante un sistema operativo en tiempo real integrado.

Para conseguir esto, es necesario admitir varios sistemas operativos y aplicaciones que se ejecuten en paralelo en distintos núcleos del procesador, y lograr el aislamiento de particiones, la cooperación síncrona y la comunicación entre sistemas operativos, con el objetivo de maximizar el rendimiento de los procesadores multinúcleo[21]. Para conseguir este multiprocesamiento, es necesario tener en cuenta la complejidad que supone que distintos entornos procesadores accedan al mismo PL. Para evitar la generación de problemas, se deben tener en cuenta cuestiones como la sincronización de los relojes de los procesadores, el acceso a las regiones de memoria compartida, la compatibilidad de la interfaces, etc.

Los procesadores multinúcleo pueden ser de dos tipos dependiendo de su diseño: homogéneos o heterogéneos. En los sistemas multinúcleo homogéneos, todos los procesadores tienen la misma arquitectura, mientras que, en los heterogéneos, existen diferencias en la arquitectura de los procesadores.

2.2.1. Multiprocesamiento Simétrico y Asimétrico

Desde el punto de vista del software, existen dos tipos de arquitecturas de procesadores multinúcleo: Multiprocesamiento Simétrico (SMP) y Multiprocesamiento Asimétrico (AMP).

- **SMP:** En un sistema de arquitectura SMP todos los procesadores comparten la misma arquitectura y un diseño homogéneo. Además, todas las CPU comparten la memoria del sistema y los recursos periféricos, y el sistema operativo es responsable de la cooperación entre procesadores y de la coherencia de la estructura de datos. El sistema operativo puede programar dinámicamente cualquier proceso en cualquier núcleo, permitiendo la plena utilización de todos los núcleos. Esto proporciona mayor escalabilidad y paralelismo que un sistema AMP, junto con una gestión de recursos compartidos más sencilla[21].
- **AMP:** En un sistema de arquitectura AMP los procesadores pueden compartir o no compartir la misma arquitectura, es decir, pueden tener un diseño homogéneo o heterogéneo. Cada núcleo es gestionado por un sistema operativo distinto, o por una copia separada del mismo SO. Normalmente, cada proceso de software está bloqueado a un único núcleo (por ejemplo, el proceso A sólo se ejecuta en el núcleo 1, el proceso B sólo se ejecuta en el núcleo 2, etc.). Mayormente, un núcleo potente es elegido el núcleo principal del procesador, y el resto de los núcleos del procesador son utilizados como núcleos auxiliares. Estos sistemas proporcionan un entorno de ejecución similar al de los sistemas uniprocesador, lo que permite una migración sencilla del código heredado. También permite a los desarrolladores gestionar cada núcleo de forma independiente[22].

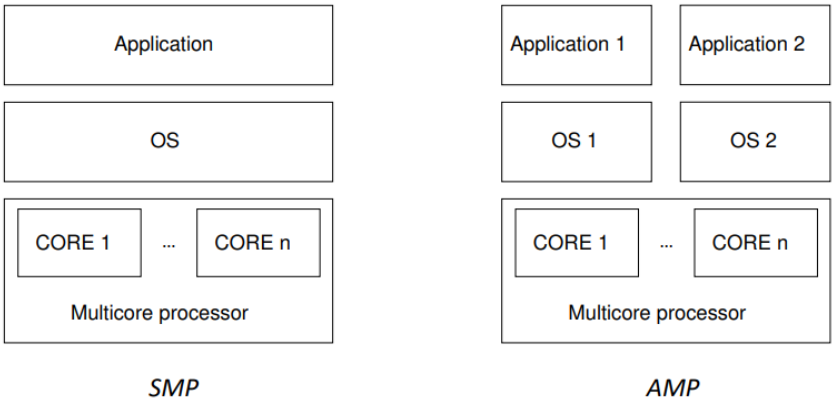


Figura 7: Sistema de multiprocesamiento simétrico y asimétrico.

2.2.2. OpenAMP

OpenAMP es un framework creado por la Multicore Association (MCA) que ofrece los componentes de software requeridos para desarrollar aplicaciones software para sistemas AMP. Ofrece funciones de gestión del ciclo de vida (LCM) y comunicación entre procesadores para gestionar recursos informáticos remotos. También proporciona una biblioteca independiente que puede utilizarse con entornos de software RTOS (Real-Time Operating System) y bare-metal (sin sistema operativo)[3].

OpenAMP utiliza Libmetal como capa de abstracción para acceder a dispositivos, manejar interrupciones y memoria compartida. Libmetal se utiliza porque proporciona una interfaz uniforme para acceder a dispositivos y memoria. OpenAMP utiliza libmetal para acceder a IPI (Interrupción entre Procesadores) y memoria compartida. OpenAMP aprovecha los estándares para la gestión de la memoria compartida, la gestión del ciclo de vida y la comunicación[23].

La librería OpenAMP proporciona la implementación de **RPMsg**, **virtIO** (Módulo de Virtualización), y **remoteproc**, que están implementados en el kernel de Linux. La biblioteca OpenAMP proporciona la implementación de estos componentes para los siguientes entornos: Baremetal, FreeRTOS (Real-Time Operating System), y espacio de usuario de Linux. A continuación, se detallan los componentes uno a uno:

- **virtIO:** La biblioteca OpenAMP implementa el estándar virtIO para la gestión de memoria compartida. El virtIO es un estándar de virtualización para drivers de dispositivos de red y disco donde sólo el driver en el dispositivo invitado sabe que se está ejecutando en un entorno virtual, con el hipervisor.
- **remoteproc:** Remoteproc proporciona capacidad para la gestión del ciclo de vida (LCM) de los procesadores remotos. La API remoteproc que utiliza la librería OpenAMP es compatible con la infraestructura presente en el Kernel Linux 3.18 y posteriores. El remoteproc utiliza la información publicada a través de la tabla de recursos del firmware del procesador remoto para asignar recursos del sistema y crear dispositivos virtIO. El remoteproc puede utilizarse para cargar firmware arbitrario; no está limitado a firmware OpenAMP.
- **RPMsg:** Esta API permite las comunicaciones entre procesos (IPC) entre el software que se ejecuta en núcleos independientes en un sistema AMP. También es compa-

tible con la infraestructura de bus RPMsg presente en el Kernel Linux versión 3.18 y posteriores.

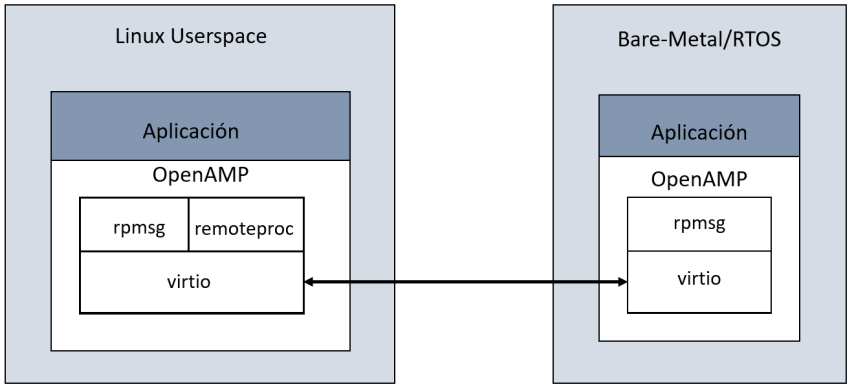


Figura 8: Implementación de OpenAMP en Linux y Bare-Metal/RTOS.[3]

OpenAMP se basa en tres tecnologías clave para poder funcionar:

- **Recursos compartidos dual-core:** Los principales recursos compartidos del dual-core incluyen 512 KB de caché L2, 256 KB de memoria en chip (OCM), memoria DDR, controlador de interrupciones GIC, reloj, etc.
- **Método de arranque dual-core:** Durante el proceso de arranque dual core, el procesador esclavo arranca bajo el control del procesador maestro, como se puede observar en la figura 9. El módulo Remoteproc de OpenAMP proporciona una interfaz de software para gestionar el ciclo de vida del procesador remoto para las aplicaciones de software que se ejecutan en el procesador principal.

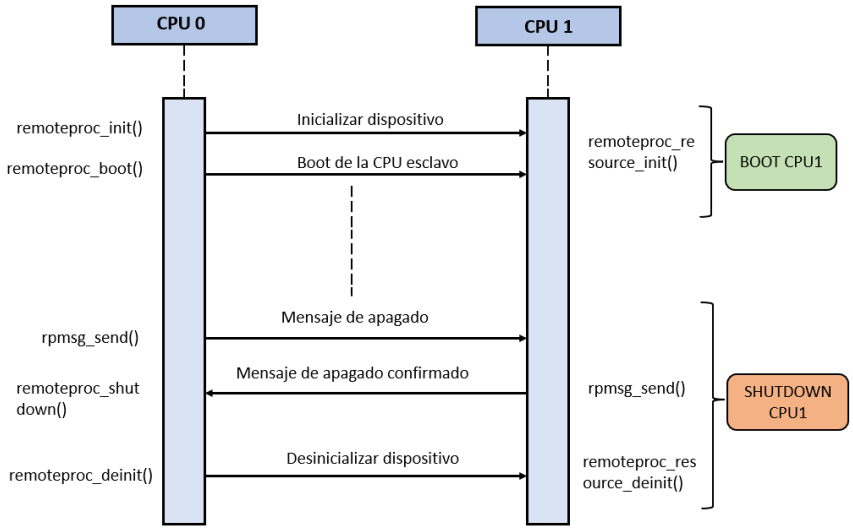


Figura 9: Secuencia de arranque dual-core mediante OpenAMP[4].

- **Método de comunicación dual-core:** La memoria compartida se utiliza para la comunicación en la arquitectura AMP. Los dos núcleos tienen derecho a leer y escribir en la misma zona de memoria[24]. Un diagrama del método de comunicación entre los núcleos se muestra en la figura 10.

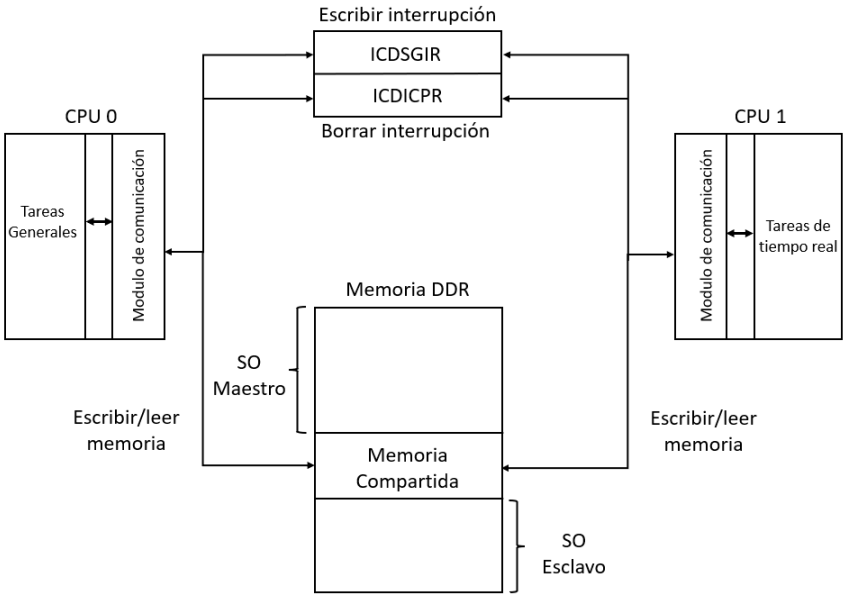


Figura 10: Método de comunicación dual-core mediante OpenAMP[4].

2.3. Emuladores de canal

Un emulador de canal de RF (radiofrecuencia) es un dispositivo o sistema que emula los efectos de un entorno de RF real en un sistema de comunicación inalámbrica. En este proyecto, se ha trabajado sobre un emulador de canal añadiendo nuevas funcionalidades. Por ello, es de vital importancia conocer este tipo de sistemas, los efectos que emulan, su funcionamiento y sus principales características.

En la figura 11 se presenta un diagrama general de un emulador de canal. El primer paso es convertir la frecuencia de operación de la señal de entrada RF a una frecuencia intermedia (IF) mediante el Front-End RF. A continuación, el modelo de canal en banda base compleja distorsiona la señal según el canal programado. Finalmente, la señal digital se convierte en analógica y se hace una conversión de IF a RF mediante el Front-End de salida RF.

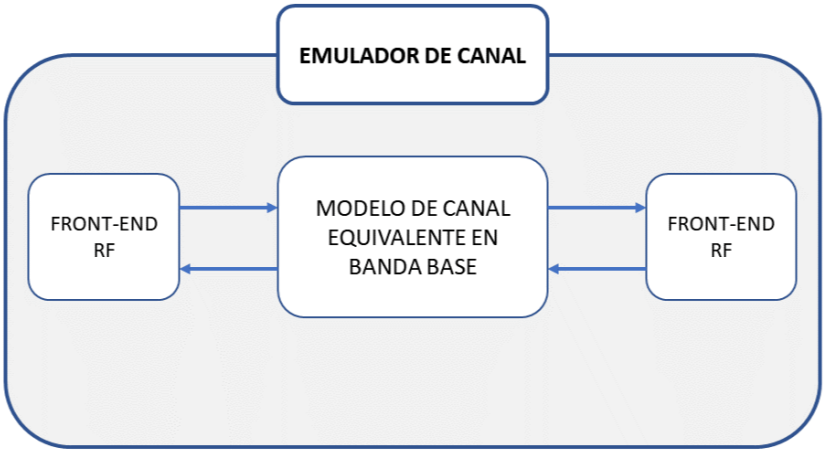


Figura 11: Esquema de un emulador de canal general.

En lo que al modelo de canal se refiere, principalmente, hay dos formas de modelar un canal inalámbrico de comunicaciones:

- La primera forma más común de modelar el canal es utilizando **modelos estocásticos** basados en mediciones. Los parámetros de estos modelos se describen mediante distribuciones estadísticas como los desvanecimientos de Rayleigh y Rice, por el PDP (Power Delay Profile), el tiempo coherente del canal y el ancho de banda[25]. Estos modelos se suelen basar en el análisis CIR (Channel Impulse Response) del canal. Este método se implementa usando filtros FIR (Finite Impulse Response).

- La segunda forma más común de modelar un canal de comunicaciones es basándose en las propiedades físicas del canal, por ejemplo, las propiedades electromagnéticas y geométricas. Estos modelos son también conocidos como **modelos deterministas**. Esta segunda forma, requiere de una mayor complejidad, puesto que, para emular la respuesta del canal se utilizan las ecuaciones de Maxwell. Dada esta complejidad, y puesto que para realizar el modelado del canal se requiere una gran cantidad de datos reales y un largo tiempo operativo para describir un entorno de propagación específico, la universalidad de los modelos deterministas es limitada[26].

2.3.1. Comunicaciones inalámbricas

La comunicación inalámbrica se refiere a la transferencia de información o datos a distancia sin el uso de hilos o cables. En su lugar, la comunicación inalámbrica utiliza ondas electromagnéticas, como ondas de radio para transmitir y recibir datos. Ejemplos de esta tecnología son las redes WiFi, redes móviles (3G,4G y 5G), Bluetooth, etc.

Los emuladores de canales de radiofrecuencia emulan diversos efectos reales que pueden afectar al rendimiento de los sistemas de comunicación inalámbrica. Estos efectos son una de las consecuencias que sufren las señales por el medio de propagación. Además, estos efectos se acentúan en los entornos industriales. Estos entornos pueden incluir una amplia gama de factores como la presencia de metal y otras superficies reflectantes, interferencias de otros sistemas inalámbricos y condiciones adversas como temperaturas extremas, vibraciones y polvo. Estos factores pueden tener un impacto significativo en el rendimiento de los sistemas de comunicación inalámbrica, provocando problemas como el desvanecimiento de la señal, la atenuación y las interferencias. Es por ello, que, en este apartado se explican los principales efectos que afectan a la calidad de la señal en una comunicación inalámbrica.

2.3.1.1. Multipath

El efecto multipath se refiere al fenómeno en el que una señal inalámbrica toma múltiples caminos mientras viaja desde el transmisor al receptor. Esto puede deberse a la reflexión, refracción o difracción de la señal por obstáculos del entorno [27], esto se observa en la figura 12. Los múltiples caminos dan lugar a la superposición de las señales en el receptor. Esta superposición de las señales puede generar dos tipos de interferencias:

- **Interferencia constructiva:** El multipath constructivo se produce cuando los múltiples trayectos de una señal inalámbrica llegan al receptor en fase unos con otros. Estas similitudes en fase entre las señales interfieren constructivamente en el receptor, aumentando la intensidad de la señal. Esto puede ocurrir cuando la señal

recorre distancias o caminos parecidos para llegar al receptor, o cuando la señal es reflejada o refractada por superficies similares con propiedades similares.

- **Interferencia destructiva:** El multipath destructivo se produce cuando los múltiples trayectos de una señal inalámbrica llegan al receptor desfasados entre sí. Esta diferencia de fase entre las señales interfiere destructivamente en el receptor, reduciendo la intensidad de la señal. Esto puede ocurrir cuando la señal recorre diferentes distancias para llegar al receptor, o cuando la señal es reflejada o refractada por diferentes superficies con diferentes propiedades.

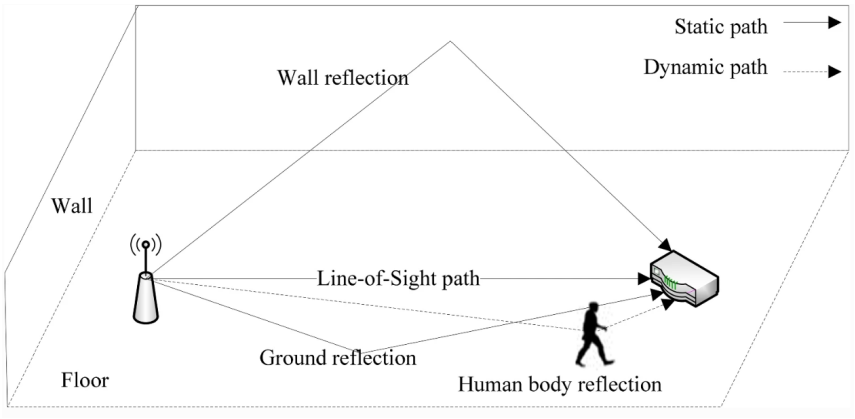


Figura 12: Efecto multipath[5].

2.3.1.2. Pathloss

El pathloss, también conocida como pérdida de trayectoria, es la disminución de la densidad de potencia (atenuación) de una onda electromagnética a medida que se propaga por el espacio. En los sistemas de comunicación inalámbricos, el pathloss se refiere a la reducción de la intensidad de la señal que se produce cuando ésta viaja del transmisor al receptor[28]. La pérdida de trayectoria se ve afectada principalmente por la distancia entre el transmisor y el receptor. Esta disminución de potencia viene dada por la ecuación de Friis:

$$Aten(dB) = 20 * log_{10}(\frac{2 * pi * d * f}{c}) \tag{2.1}$$

donde d es la distancia entre las antenas (metros), f la frecuencia de la señal enviada (Hz) y c es la velocidad de la luz.

Para conseguir atenuaciones significativas, se necesitan señales con frecuencias altas o que la distancia entre emisor/receptor sea grande. En la figura 13 se puede observar la atenuación por PathLoss para diferentes bandas ISM (Industrial Scientific and Medical Band). Las bandas de radio ISM son porciones del espectro radioeléctrico reservadas internacionalmente para fines industriales, científicos y médicos.

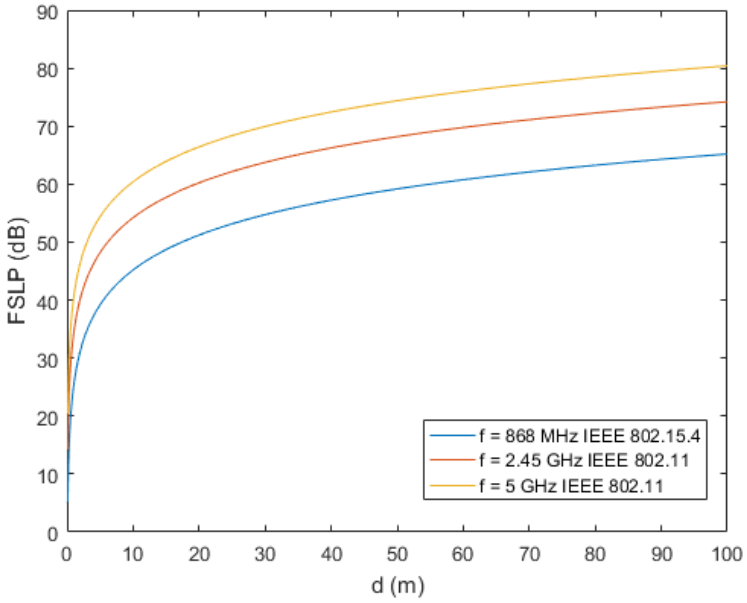


Figura 13: Path Loss para diferentes bandas ISM.

2.3.1.3. Efecto Doppler

El efecto Doppler, también conocido como desplazamiento Doppler, es el cambio de frecuencia o longitud de onda de una señal causado por el movimiento relativo de la fuente de la señal y el observador[29]. En las comunicaciones inalámbricas, el efecto Doppler puede causar desvanecimiento, distorsión y otras alteraciones de la señal, dependiendo de la velocidad relativa del transmisor, el receptor y los dispersores del entorno. Por ello, el efecto Doppler es un factor clave en el diseño y funcionamiento de los sistemas de comunicación inalámbrica, sobre todo en los sistemas móviles y de alta velocidad.

Sin embargo, cuando se habla de modelar un canal, es más interesante hablar sobre la distribución Doppler del canal. La distribución Doppler de un canal se refiere a la distribución estadística del desplazamiento Doppler, que es el cambio en la frecuencia de una onda causado por el movimiento relativo del transmisor, el receptor y los dispersores en el entorno.

Existen varios tipos de distribuciones Doppler que se utilizan habitualmente para modelar canales inalámbricos, como las distribuciones de Gauss, Rayleigh, Rician y Nakagami. Estas distribuciones tienen diferentes parámetros que dependen de las características del canal, como la velocidad de los dispersores, la longitud de onda de la señal y la densidad de los dispersores en el entorno.

2.3.1.4. Ruido

Una forma en que se puede alterar la naturaleza original de una señal es la introducción en el medio de propagación de señales extrañas provocando en la señal original alteraciones en amplitud y frecuencia, conociéndose este fenómeno como ruido. Las causas por las que la señal sufre ruido pueden ser varias, dependiendo de su origen: El ruido en una señal se refiere a cualquier energía eléctrica o electromagnética que pueda interferir con la señal original o deseada. El ruido puede proceder de diversas fuentes, como

el ruido térmico, el ruido atmosférico, las interferencias provocadas por el hombre y los componentes electrónicos de un sistema.

Existen distintos tipos de ruido, en función de su origen:

- El **ruido térmico**, está causado por el movimiento aleatorio de los electrones en una resistencia u otro componente eléctrico. Es un tipo de ruido fundamental que está presente en todos los sistemas electrónicos.
- **Ruido de disparo**, está causado por la llegada aleatoria de electrones a una unión semiconductora, es importante en dispositivos electrónicos como foto-diodos, transistores y amplificadores electrónicos.
- **Ruido impulsivo**, está causado por factores externos como rayos, interferencias electromagnéticas, interferencias de radiofrecuencia y otros tipos de interferencias.
- **Ruido de intermodulación**, está causado por las no linealidades de los componentes electrónicos, como los amplificadores.

El ruido puede tener un efecto perjudicial en el rendimiento de un sistema de comunicación. En la figura 14 se puede observar el efecto que produce este fenómeno. Por eso es importante diseñar sistemas de comunicación que puedan minimizar o eliminar el ruido y aumentar la SNR (Signal to Noise Ratio). Su forma matemática queda definida como:

$$SNR(dB) = 10 * log_{10}(\frac{P_{signal}}{P_{noise}}) \tag{2.2}$$

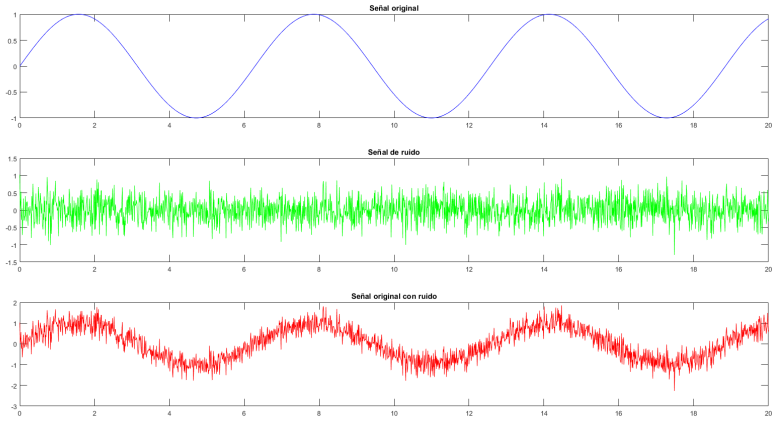


Figura 14: Efecto del ruido en una señal.

2.3.2. Emulador de canal de Ikerlan

En este apartado se van a explicar las principales características del emulador de canal que se ha utilizado en este proyecto. Como se ha explicado en la introducción, se dispone de un emulador de canal multipunto en banda base basado en la tarjeta de procesamiento Zynq Ultrascale +, con un ancho de banda de 100 MHz y un rango que

cubre las frecuencias desde 400 MHz hasta 6 GHz de la banda ISM. En la figura 1, se puede observar un diagrama general del emulador de canal.

El emulador de canal está conectado a cuatro transceptores formando cuatro nodos: A, B, C y D. Estos nodos forman entre ellos 6 enlaces inalámbricos (A-B, A-C, A-D, ...). El emulador cuenta en cada nodo, con un Front-End RF para la conversión de frecuencia tanto descendente como ascendente, conversión de RF a IF (concretamente 61.44 MHz) y viceversa. Además de una etapa conversora analógico-digital o digital-analógica para discretizar la señal y realizar la emulación de canal en el MPSoC o reconvertir la señal analógica tras la emulación.

El proceso de emulación de un canal sería el siguiente: El emulador de canal recibe datos a través de los transceptores RF. El Front-End RF de entrada, realiza la conversión de RF/IF para reducir espurios y mejorar la calidad de filtrado. La señal es desplazada en frecuencia en dos pasos; primero a 645 MHz y después a 61.5 MHz. La señal convertida es entonces discretizada por el ADC y enviada a la Zynq UltraScale+. Dentro del MPSoC, la señal se mueve a banda base y se le aplica el modelo de canal predefinido. Para finalizar, se vuelve a convertir la señal de banda base a IF, se recupera la señal analógica mediante el DAC, y se desplaza en frecuencia la señal de nuevo a RF mediante el Front-End de salida.

2.3.2.1. Front-End RF

Cada uno de los nodos dispone de un Front-End RF para la Downconversion y Upconversion de la señal. Cada uno de estos Front-End dispone de cuatro elementos programables:

- Dos osciladores locales (EV-ADF4355)[13] para convertir la señal RF a frecuencia intermedia en dos etapas; primero, a 645 MHz y después a 61.44 MHz.
- Un atenuador variable (RUDAT 6000-110)[30] para realizar la emulación de un perfil de atenuación.
- Un amplificador de IF (ADL5202)[31] con el objetivo de optimizar el nivel de potencia de la señal de entrada del ADC.

Estos elementos se controlan mediante una comunicación USB desde un microcontrolador EFM32 [12], que actúa como hub USB para los cuatro elementos programables. Este microcontrolador recibe los valores mediante una comunicación SPI con el MPSoC.

Cada Front-End cuenta con un detector de potencia. La función del detector de potencia es distinguir si existe señal de entrada. En caso de que exista, el puerto RF se pondrá en modo "recepción" (la señal entra al Front-End) y la señal de entrada se redirige al ADC tras la downconversion. Por el contrario, si el detector de potencia no detecta señal se supone que la señal está saliendo por defecto, proveniente del DAC. A continuación, se observa un diagrama del Front-End.

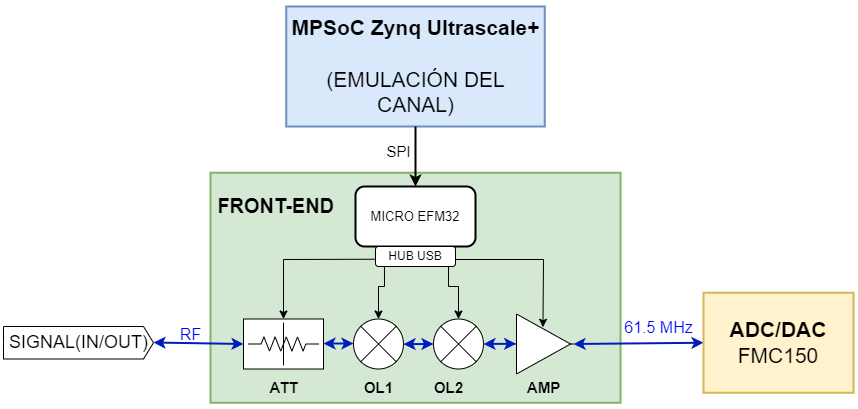


Figura 15: Diagrama del Front-End del emulador de canal.

2.3.2.2. Tarjeta de procesamiento Zynq Ultrascale+ MPSoC

La tarjeta de procesamiento es la parte fundamental del diseño del emulador. Dentro de esta se pueden dividir dos grandes secciones: el sistema de procesamiento (PS) y la unidad de lógica programable (FPGA). Como se ha detallado en el apartado referente a la tarjeta Zynq Ultrascale+, esta cuenta en su PS con dos unidades de procesamiento. Por un lado, un ARM Cortex A53 con cuatro cores, y por otro lado, una unidad de procesamiento de tiempo real ARM Cortex R5 dual-core. A continuación, se detalla el funcionamiento de cada uno de estos cores, así como de la parte lógica del MPSoC.

2.3.2.2.1. Cortex A53 - Interfaz de Usuario

Esta unidad de procesamiento contiene una aplicación Linux, donde se ejecuta la interfaz de usuario del emulador de canal. Esta interfaz se encarga de adquirir los datos introducidos por el usuario, y enviar los mensajes de configuración al procesador de tiempo real. Esta interfaz permite configurar el modelo y las características del canal a emular, así como los parámetros para la configuración de los elementos del Front-End.

Para el desarrollo de la aplicación, se ha utilizado la herramienta Petalinux [32]. El kit de desarrollo de software (SDK) PetaLinux es una herramienta de desarrollo de Xilinx que contiene todo lo necesario para construir, desarrollar, probar y desplegar sistemas Linux embebidos. La aplicación de la interfaz de usuario está desarrollada mediante la herramienta QT Creator[9]. En la figura 16 se observa la pantalla principal de la interfaz de usuario.

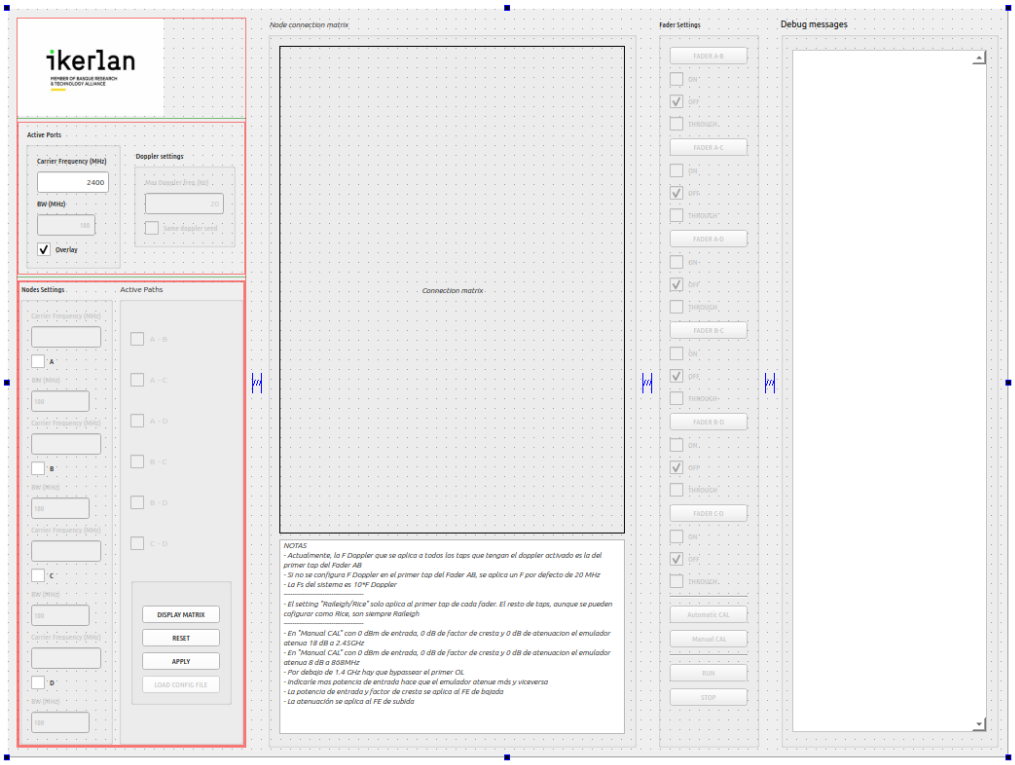


Figura 16: Pantalla principal de la interfaz de usuario del emulador.

Como se observa, se pueden configurar varios valores relacionados al canal que se pretende emular, como la frecuencia Doppler, la frecuencia de la señal portadora, etc. El funcionamiento de la interfaz se explicará detalladamente en los apartados relacionados al desarrollo de las nuevas funcionalidades.

Por otro lado, para enviar los datos de los parámetros desde la interfaz al procesador de tiempo real se utiliza openAMP. Como se ha explicado anteriormente, openAMP ofrece la comunicación entre procesadores para la gestión de recursos, gestión memoria compartida, manejo de interrupciones, etc. En este caso, el procesador A53 actúa como maestro de la comunicación y cuando la aplicación se ejecuta, arranca el procesador remoto, en este caso, el procesador de tiempo real R5. Después, se procede al envío de las estructuras con información acerca del canal.

2.3.2.2.2. Cortex R5-Procesador de tiempo real

En el procesador de tiempo real Cortex R5 se ejecuta la aplicación de control de bajo nivel del emulador de canal. Esta aplicación recibe los parámetros de configuración en estructuras con información sobre el canal enviadas desde la aplicación de la interfaz mediante el uso de openAMP. Tras la recepción de estos datos, el R5 genera los coeficientes necesarios para la emulación del canal. Estos coeficientes, con información relacionada al modelo del canal, se emiten a la parte lógica del MPSoC mediante el bus AXI y registros. Para los coeficientes relacionados con información sobre los elementos del Front-End, el R5 genera los coeficientes necesarios y los envía mediante SPI.

2.3.2.2.3. FPGA-Lógica Programable

Finalmente, la implementación del emulador de canal se encuentra en la lógica programable del MPSoC. Este elemento realiza la emulación de los modelos de canal y se comunica con la unidad de procesamiento Cortex-R5 a través del bus AXI. A continuación, se muestra un pequeño diagrama de bloques de los apartados de la PL.

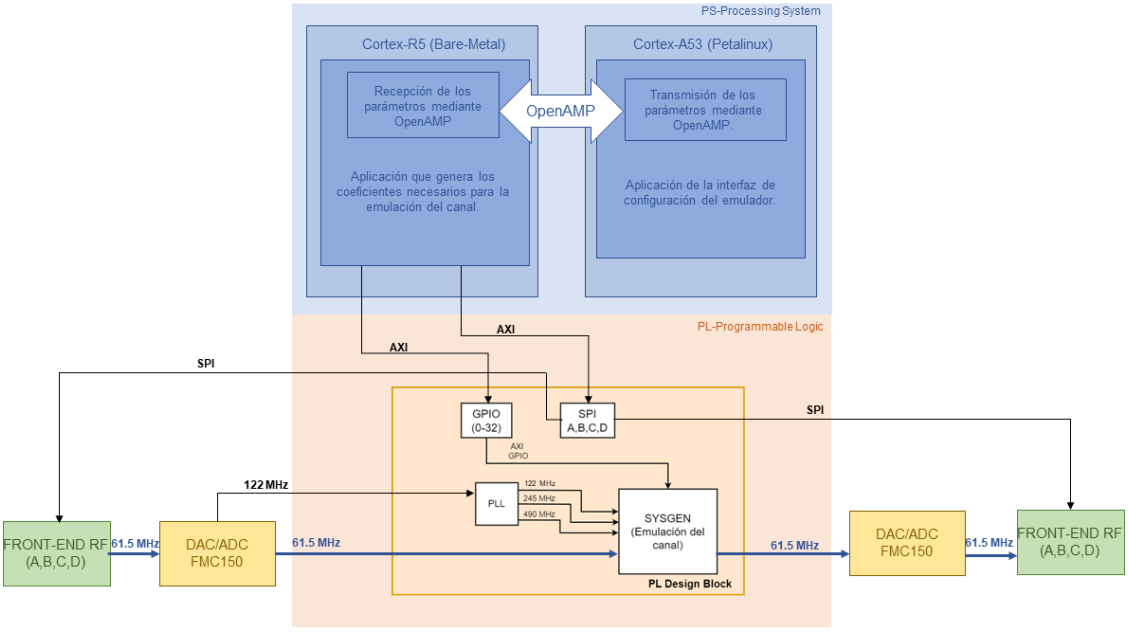


Figura 17: Diagrama de bloques de la parte de la Zynq Ultrascale+.

Como se observa en la figura 17, existen varias partes dentro de PL. El bloque Sysgen hace referencia al diseño del emulador desarrollado mediante la herramienta System Generator. Se ha generado una IP, y se ha añadido al diseño en Vivado. En este bloque la señal pasa de IF a banda base, y se realiza la emulación del canal teniendo en cuenta los valores introducidos desde el R5. Come se ve en la figura, los coeficientes se transmiten mediante la interfaz AXI GPIO.

2.3.2.3. Emuladores de canal comerciales

Al margen del emulador de canal creado en Ikerlan, existen varias opciones comerciales de emuladores que cumplen funciones similares. Los emuladores de canal comerciales son dispositivos costosos, especialmente cuando tienen una configuración de alto rendimiento con un ancho de banda amplio, varios puertos I/O y un rango de frecuencias amplio. Los emuladores comerciales mas conocidos son de la marca Spirent y Keysight. En la tabla 1, se muestra una comparativa entre los modelos comerciales Spirent Vertex[33], Keysight Propsim FS8[34] y el emulador de Ikerlan.

Características	Spirent Vertex	PROPSIM FS8	Emulador Ikerlan
RF in	Up to 16	Up to 8	Up to 4
Número de canales	Up to 256	Up to 32	Up to 6
Trayectorias máximas	6144(256x24)	1536(32x48)	60(6x10)
Frecuencias entrada	30MHz to 5925MHz	350MHz to 6000MHz	400MHz to 6000MHz
Max. Ancho de Banda	100 MHz	40 MHz	100 MHz
Delay	0 to 4000us	0 to 3000us	0 to 8us
Step Delay	0.1ns	-	8ns
Max. Doppler Shift	Up to 12000Hz	-	Up to 1000 Hz

Tabla 1: Comparativa de emuladores comerciales y emulador de Ikerlan.

3. Desarrollo

El emulador de canal es un prototipo en funcionamiento que es capaz de emular hasta 6 canales de comunicaciones con hasta 10 trayectorias por canal. Además, es capaz de capturar señales en el rango de 400 a 6000 MHz, con un ancho de banda de 100 MHz.

Como se explica en el apartado 1.3, el objetivo principal de este trabajo es añadir nuevas funcionalidades al emulador de canal RF basado en la tarjeta de procesamiento ZCU102, combinando las unidades de procesamiento (APU y RPU), así como la lógica programable.

En este apartado, se van a explicar todos los pasos que se han seguido con el objetivo de añadir estas funcionalidades. La primera, es la capacidad de habilitar una opción en la interfaz gráfica, de lectura e interpretación de un fichero con parámetros acerca de los coeficientes del canal. De esta manera, se pretende que no haya la necesidad de ajustar o añadir los parámetros del canal manualmente mediante la interfaz, sino que, cargando un fichero, el procesador Linux lea los coeficientes y los transmita mediante la comunicación openAMP al procesador de tiempo real, para que este los aplique en la emulación del canal mediante AXI.

La segunda, es el diseño de un estimador de potencia para realizar el ajuste de la ganancia del amplificador del Front-End. Hasta ahora, la ganancia del amplificador se añade manualmente en la interfaz gráfica. El objetivo de diseñar el estimador es que se actualice el valor de ganancia del amplificador dinámicamente una vez calculada la potencia de entrada de la señal, para adaptar la potencia de entrada del ADC con el objetivo de aprovechar todo su rango dinámico. El objetivo es diseñar este estimador mediante la herramienta System Generator y añadirla al diseño actual de Vivado.

Por lo tanto, teniendo dos tareas principales que completar, este apartado se dividirá en dos grandes secciones. En cada sección, se explicará en que consiste con mas exactitud cada tarea, los pasos que se han seguido en el diseño e implementación de la funcionalidad, así como los resultados o conclusiones que se pueden sacar.

3.1. Carga del fichero con parámetros del canal

Como se ha expuesto anteriormente, existe la necesidad de añadir una opción en la interfaz gráfica del emulador que sea capaz de leer un fichero con parámetros y los envíe al procesador de tiempo real R5 para que este los aplique adecuadamente en la emulación del canal. El diseño actual del emulador no cuenta con esa capacidad. Pero, tras la configuración de canal desde la interfaz, el emulador es capaz de enviar la información sobre los parámetros de emulación de un procesador a otro, y a su vez, el procesador de tiempo real una vez ha recibido e interpretado esta información, es

capaz de generar los coeficientes necesarios para la emulación del canal. Los parámetros hacen referencia a los valores de las distintas opciones de configuración que se envían encapsuladas en estructuras desde la interfaz, mientras que los coeficientes, son los valores que genera el procesador R5 para la emulación del canal en base a los parámetros de las estructuras recibidas.

Cuando se enciende el emulador, el procesador Linux A53 es el primero en iniciarse. Después, el procesador A53 carga la aplicación del procesador de tiempo real R5 y lo arranca gracias a OpenAMP. Una vez que la aplicación de la interfaz gráfica se inicia en el procesador A53, el procesador R5 queda en espera de recibir los mensajes de la interfaz. A través de la interfaz, se pueden configurar los parámetros de emulación, que se envían al procesador R5 en estructuras. Cuando este las recibe, las procesa y gracias a los parámetros es capaz de generar los coeficientes necesarios para la emulación del canal: Ajustar los elementos del Front-End mediante SPI o aplicar los coeficientes del canal mediante AXI al emulador en PL. Para aplicar los coeficientes durante la emulación, el Cortex-R5 hace uso de un flag de interrupciones que se activa mediante un timer. La frecuencia de este timer es 10 veces superior a la frecuencia Doppler definida en la interfaz. Este valor 10 veces más grande se hace para mantener una década de margen y evitar problemas en la emulación.

Para poder implementar la nueva funcionalidad, el primer paso ha sido tratar de entender cómo funcionan y se comunican entre ellos los dos entornos procesadores para conseguir la emulación del canal. Y una vez se ha entendido el funcionamiento, construir o añadir la nueva funcionalidad al actual diseño. Es por lo que, en el siguiente apartado, se van a explicar en detalle todas las tecnologías y elementos que son partícipes de la emulación.

3.1.1. Implementación Linux en MPSoC

La implementación del entorno Linux en el MPSoC se realiza mediante la herramienta Petalinux. El flujo de diseño de Petalinux parte siempre de un fichero de descripción de hardware *(Hardware Description File)*, que se importa a través de un diseño de Vivado. En este caso, el diseño de Vivado incluye la implementación del emulador. Una vez se realiza la síntesis y la implementación del diseño, se crea el fichero de descripción de hardware *".hdf"* a partir del cual se crean las imágenes para la integración de Linux sobre la plataforma.

Para acceder al conjunto de herramientas de Petalinux, es necesario una distribución Linux para poder ejecutar los diferentes comandos. En este caso, se ha hecho uso del programa VirtualBox con una distribución Ubuntu para la ejecución del programa.

El primer paso es crear un proyecto Petalinux mediante el comando *"petalinux-create"*, eligiendo la plantilla de hardware correspondiente.

```
~$ petalinux-create --type project --template zynqmp -name <project_name>
```

Después, se cambia al directorio donde se encuentra el proyecto y se utiliza el comando *petalinux-config* para personalizar el proyecto Petalinux. En la primera ejecución del comando se añade el path al directorio donde se encuentra el archivo con la descripción de hardware. Además de importar el hardware, este comando lanza el menú de configuración del sistema. Mediante este menú que se observa en la figura 18 se configuran/modifican las características relacionadas con el hardware y el sistema en general.

```
$ cd .\project_name
$ petalinux-config --get-hw-description=
```

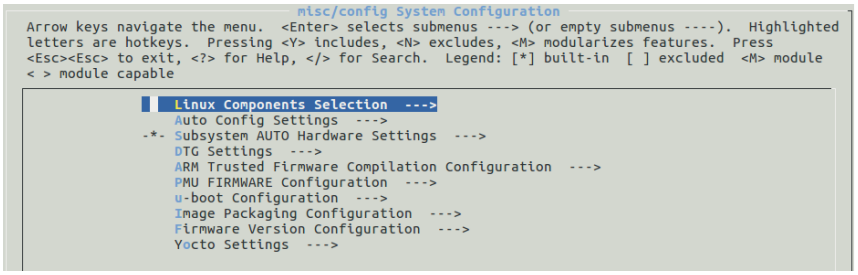


Figura 18: Menú de configuración de Petalinux.

Dentro de este menú, se deben realizar algunos cambios, los primeros son para que la revisión de la tarjeta de evaluación concuerde con la tarjeta usada. El último es necesario para que el sistema de ficheros se monte en una tarjeta SD.

```
-> DTG Settings -> MACHINE_NAME -> zcu102-rev1.0
-> u-boot Configuration -> u-boot config target -> …102_rev1_0_defconfig
-> Image Packaging Configuration -> Root FS type -> SD card
```

Después, usando el comando *"petalinux-create"*, Petalinux proporciona la capacidad de añadir aplicaciones creadas al sistema desarrollado.

```
$ petalinux-create -t apps --template install -name <app-name> --enable
```

Este comando crea una nueva carpeta en la que se encuentra un fichero *.bb* y una carpeta llamada *"files"*. El fichero *".bb"*que se genera, es la receta de la aplicación del emulador. Las recetas contienen instrucciones sobre cómo configurar, compilar y desplegar un determinado software. En la carpeta *"files"* se copia el archivo *.elf*, este archivo es un ejecutable generado tras la compilación de la aplicación del procesador R5 mediante la herramienta Xilinx SDK.

Posteriormente añadiendo el siguiente comando se abre el menú de configuración del *Root File System*.

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

En este menú se deben seleccionar los paquetes necesarios para el sistema de archivos. En este caso, para poder establecer la comunicación entre los cores A53 y R5 mediante openAMP, es necesario activar los siguientes paquetes:

```
Filesystem Packages --->
    libs --->
         libmetal ->
            [*] libmetal
         openamp ->
            [*] openamp
    misc --->
```

```
        sysfsutils --->
            [*] libsysfs
```

Además, en el apartado *apps* del menú de configuración del *Root File System* se seleccionan las aplicaciones remotas que se quieren integrar en el sistema de archivos. Estas aplicaciones se ejecutan en el procesador remoto, es decir, en el Cortex R5. Sin embargo, en la opción *"user packages"* se seleccionan las aplicaciones que se ejecutan en el host, en este caso la aplicación de la interfaz para configurar el emulador.

Finalmente, como último paso de personalización de la imagen de Linux, se deben activar algunas opciones en el núcleo del sistema operativo con el siguiente comando.

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

En un principio no hay que hacer ningún cambio, ya que al importar el hardware el kernel se autoconfigura correctamente. De todos modos, conviene asegurarse que el kernel está configurado para trabajar con OpenAMP.

```
[*] Enable loadable module support --->
    Device Drivers --->
        Generic Driver Options --->
             <*>Userspace firmware loading support
        Remoteproc drivers --->
            <*>ZynqMP_r5 remoteproc
```

Posteriormente, se debe modificar el *Device Tree* generado por defecto con el fichero *system_user.dtsi* (generado sin contenido de forma automática por Petalinux) para incluir información relativa a *Remoteproc* y la ubicación de los descriptores de buffer y la memoria compartida entre núcleos. Además de deshabilitar la *UART1* para que no se pueda acceder a ella desde el A53 y solo el procesador remoto la utilice, así como definir los dispositivos SPI de la PS. Las definiciones son las siguientes:

```
/include/ "system-conf.dtsi"
/ {
     aliases {
        spi0 = &qspi;
        spi1 = &spi0;
        spi2 = &spi1;
        spi3 = &axi_quad_spi_0;
        spi4 = &axi_quad_spi_1;
     };


    reserved-memory {
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;
        rproc_0_reserved: rproc@3ed000000 {
            no-map;
            reg = <0x0 0x3ed00000 0x0 0x1000000>;
        };
    };

    power-domains {
```

```
        pd_r5_0: pd_r5_0 {
            #power-domain-cells = <0x0>;
            pd-id = <0x7>;
        };
        pd_tcm_0_a: pd_tcm_0_a {
            #power-domain-cells = <0x0>;
            pd-id = <0xf>;
        };
        pd_tcm_0_b: pd_tcm_0_b {
            #power-domain-cells = <0x0>;
            pd-id = <0x10>;
        };
    };

    amba {
        r5_0_tcm_a: tcm@ffe00000 {
            compatible = "mmio-sram";
            reg = <0x0 0xFFE00000 0x0 0x10000>;
            pd-handle = <&pd_tcm_0_a>;
        };
        r5_0_tcm_b: tcm@ffe20000 {
            compatible = "mmio-sram";
            reg = <0x0 0xFFE20000 0x0 0x10000>;
            pd-handle = <&pd_tcm_0_b>;
        };
        elf_ddr_0: ddr@3ed00000 {
            compatible = "mmio-sram";
            reg = <0x0 0x3ed00000 0x0 0xF0000>;
        };
        test_r50: zynqmp_r5_rproc@0 {
                        compatible = "xlnx,zynqmp-r5-remoteproc-1.0";
                        reg = <0x0 0xff9a0100 0x0 0x100>,
            <0x0 0xff340000 0x0 0x100>,
            <0x0 0xff9a0000 0x0 0x100>;
                        reg-names = "rpu_base", "ipi", "rpu_glbl_base";
                        dma-ranges;
                        core_conf = "split0";
                        srams = <&r5_0_tcm_a &r5_0_tcm_b &elf_ddr_0>;
                        pd-handle = <&pd_r5_0>;
                        interrupt-parent = <&gic>;
                        interrupts = <0 29 4>;
        } ;
    };

};

&uart1
{
 status = "disabled";
};

&spi0 {
  is-decoded-cs = <0>;
  num-cs = <1>;
  status = "okay";
  spidev@0x00 {
    compatible = "spidev";
    spi-max-frequency = <5000000>;
```

```
    reg = <0>;
  };
};

&spi1 {
  is-decoded-cs = <0>;
  num-cs = <1>;
  status = "okay";
  spidev@0x00 {
    compatible = "spidev";
    spi-max-frequency = <5000000>;
    reg = <0>;
  };
};
```

Finalmente, mediante el siguiente comando se realiza una compilación completa del núcleo y el sistema de archivos de Linux:

```
$ petalinux-build
```

Por último, se accede a la carpeta *"\images\linux"* dentro del directorio del proyecto y se generan los ficheros *BOOT.bin* e *image.ub* que se deben incluir en la partición de la tarjeta SD para ejecutar el sistema operativo en el MPSoC.

```
$ petalinux-package --boot --fsbl zynqmp_fsbl.elf --u-boot --force
```

En la figura 19 se puede observar un diagrama del flujo del diseño de Petalinux.

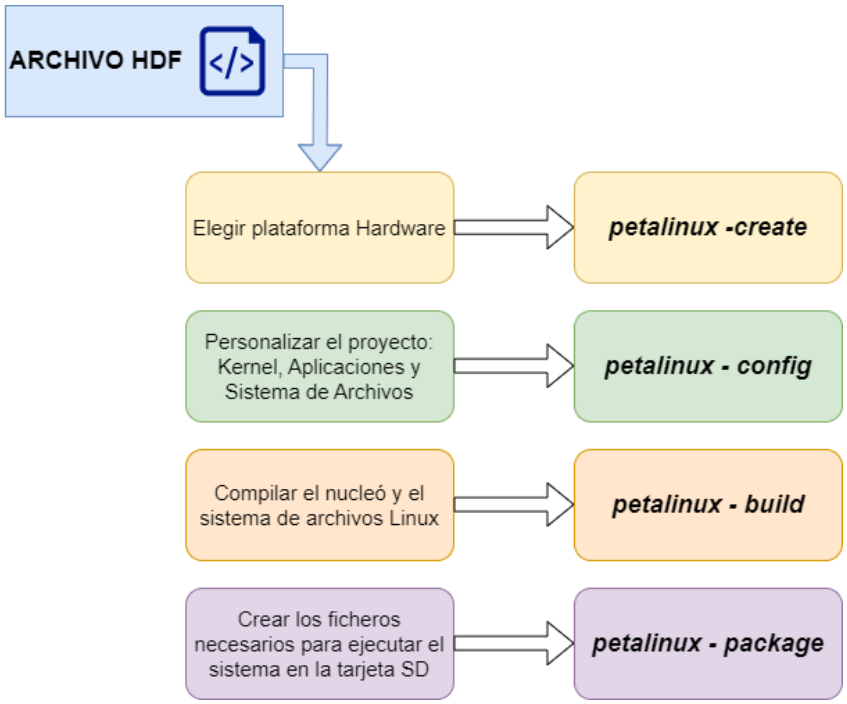


Figura 19: Etapas de diseño de Petalinux.

3.1.2. Aplicación Bare-metal en Cortex R5

Para poder aplicar los valores de los parámetros provenientes del procesador A53, el procesador de tiempo real debe ejecutar una aplicación que sea capaz de establecer una comunicación openAMP, así como generar los coeficientes necesarios para el canal. Para crear la aplicación del emulador de canal, se utiliza la plantilla base *"openAMP echo test"*, de la librería openAMP de *Xilinx SDK*.

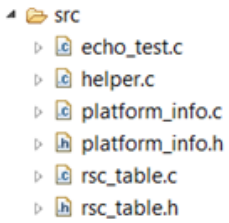


Figura 20: Archivos de la plantilla openAMP echo test.

Como se observa en la figura 20, la plantilla base *echo_test* contiene varios ficheros esenciales para habilitar la comunicación openAMP en el procesador R5:

- **Información de la Plataforma** (*platform_info.c/.h*): Estos ficheros contienen valores hard-codeados específicos de la plataforma para obtener la información necesaria para OpenAMP.
  - ***#define IPI_IRQ_VECT_ID:*** Vector de Interrupción de Inter-Procesador (IPI, Inter-Processor Interrupt) del agente IPI utilizado para la comunicación entre procesadores.
  - ***#define IPI_BASE_ADDR:*** Dirección base del agente IPI utilizado para la comunicación entre procesadores.
  - ***#define RPMSG_CHAN_NAME:*** Nombre utilizado para identificar un canal de comunicación entre dos procesadores.
  - ***#define IPI_CHN_BITMASK:*** La máscara de bit del IPI para el procesador remoto. Esta máscara identifica con qué procesador remoto comunicarse.
- **Tabla de Recursos**(*rsc_table.c/.h*): La tabla de recursos contiene entradas que especifican los recursos de memoria y *virtIO*. El dispositivo *virtIO* contiene características del dispositivo, y la información de las direcciones, tamaños y alineamientos de los *vring*. Mediante la memoria que se especifica en el recurso *RSC_RPROC_MEM*, el kernel de Linux remoteproc asigna la memoria compartida para los *vring*s y los buffers de *RPMsg*. Este recurso tiene que estar dentro del rango de la memoria reservada definida en *DTS(Device Tree Specification)* para el procesador remoto, pero sin que se superponga en las direcciones de memoria asignadas a la aplicación en el linker script.
- **Helper**(*helper.c/h*): Contienen APIs específicas de la plataforma que permiten a la aplicación remota conectarse con el hardware. Incluyen funciones para inicializar y controlar el *GIC(Generic Interrupt Controller)*.
- **Código de aplicación**(*echo_test.c*): Contiene el código de la aplicación. En este caso, el procesador remoto manda de vuelta al maestro Linux los mensajes de *RPMsg* que recibe.

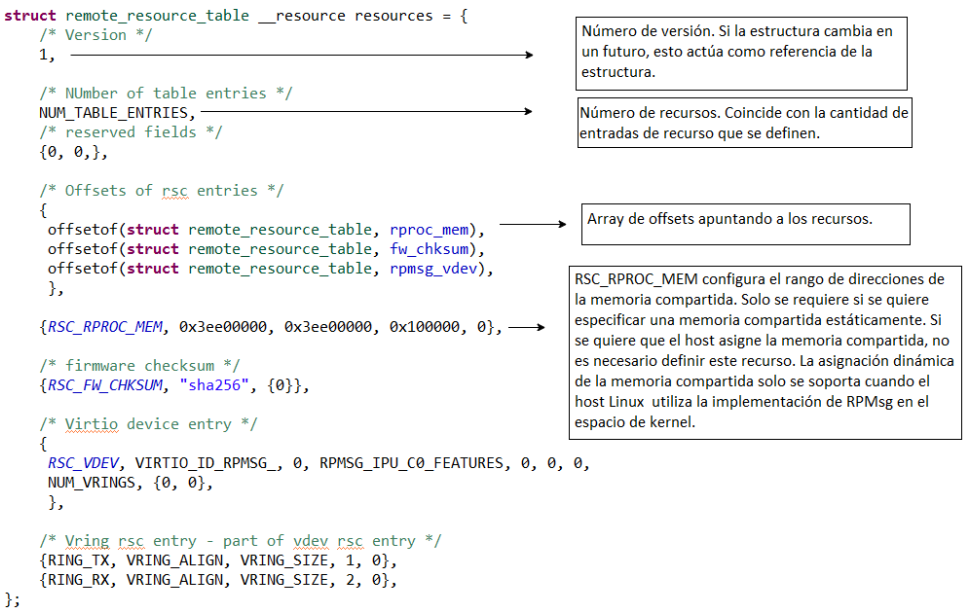


Figura 21: Tabla de Recursos del firmware.

En lo que al flujo de aplicación se refiere, a continuación, se detallan los pasos concretos que deben seguirse para establecer una comunicación. En este caso para realizar una aplicación simple de *echo*, es decir, un aplicación que lea los mensajes y los devuelva sin modificarlos ni realizar ninguna operación.

1. El primer paso es crear una tabla de recursos del firmware como la que se observa en la figura 21. La tabla de recursos define las entradas necesarias del firmware para la aplicación OpenAMP. Es una lista de los recursos del sistema requeridos por el *remote_proc*.

2. Definir las funciones de callback de *RPMsg* para la creación del canal, la supresión y la recepción de mensaje, como se observa en la figura22.

```
/*-----------------------------------------------------------------------------*
 *  RPMSG callbacks setup by remoteproc_resource_init()
 *-----------------------------------------------------------------------------*/
static void rpmsg_read_cb(struct rpmsg_channel *rp_chnl, void *data, int len,
              void *priv, unsigned long src)
{
    (void)priv;
    (void)src;

    /* On reception of a shutdown we signal the application to terminate */
    if ((*(unsigned int *)data) == SHUTDOWN_MSG) {
        evt_chnl_deleted = 1;
        return;
    }

    /* Send data back to master */
    if (rpmsg_send(rp_chnl, data, len) < 0) {
        LPERROR("rpmsg_send failed\n");
    }
}

static void rpmsg_channel_created(struct rpmsg_channel *rp_chnl)
{
    rp_ept = rpmsg_create_ept(rp_chnl, rpmsg_read_cb, RPMSG_NULL,
                  RPMSG_ADDR_ANY);
}

static void rpmsg_channel_deleted(struct rpmsg_channel *rp_chnl)
{
    (void)rp_chnl;

    rpmsg_destroy_ept(rp_ept);
    rp_ept = NULL;
    evt_chnl_deleted = 1;
}
```

Figura 22: Funciones de callback de *RPMsg*.

3. A continuación, se debe crear una instancia *hil_proc*. La instancia *hil_proc* representa un procesador remoto y encapsula la memoria compartida y la información de las notificaciones que se requieren para la comunicación entre procesadores. Es imprescindible crear la instancia de *hil_proc* para inicializar el framework de *RPMsg*.

4. Para poder inicializar la comunicación es necesario llamar a la función *remoteproc_resource_init()* para configurar el framework de *RPMsg*. *remoteproc_resource_init()* inicializa el framework *RPMsg* en una aplicación OpenAMP, utilizando como argumentos la tabla de recursos del firmware y las estructuras de *hil_proc* junto con las funciones de callback de *RPMsg*.

5. Para saber si se ha recibido algún mensaje nuevo es necesario llamar a la función *hil_poll()*. Esta función realiza un polling en el procesador remoto.

6. Para finalizar, mediante *rpmsg_send()* se envía un mensaje al procesador maestro. En este caso, la misma información que se ha recibido.

En resumen, después de inicializar el framework *remoteproc_resource_init()*, el flujo de una aplicación OpenAMP consiste en un canal *RPMsg* actuando como comunicación entre el procesador maestro y el remoto vía *rpmsg_send()* y las funciones de callback.

Una vez se ha comprendido el uso de openAMP, se detalla el funcionamiento de la aplicación y cómo se generan los coeficientes en el procesador Cortex R5.

El procesador Cortex R5 recibe desde la interfaz gráfica diferentes tipos de estructuras con información relacionada al canal y al Front-end. Las estructuras que intercambian los procesadores son las siguientes:

- **Canal:** Esta estructura recoge la información sobre la activación del canal, los coeficientes de Rice y la atenuación del canal.

```
typedef struct Canal {
  bool_t b_canal_enable; /**< Canal habilitado*/
  float32_t f32_rice_C1; /**< Coeficiente de Rice C1*/
  float32_t f32_rice_C2; /**< Coeficiente de Rice C2*/
  int8_t i8_atenuacion; /**< Atenuacion del canal*/

  //Variables para guardar el estado en caso de que se pase el fader de
    ON a OFF/THROUGH
  float32_t f32_rice_C1_old; /**< Coeficiente de Rice C1*/
  float32_t f32_rice_C2_old; /**< Coeficiente de Rice C2*/
  int8_t i8_atenuacion_old; /**< Atenuacion del canal*/
} Canal;
```

- **Path:** Esta estructura recoge la información sobre el path activado, los desvanecimientos, el retardo y la ganancia.

```
typedef struct Path {
  bool_t b_path_enable; /**< Path habilitado*/
  bool_t b_desvanecimientos; /**< Desvanecimientos habilitados*/
  int32_t i32_delay; /**< Delay del path*/
  float32_t f32_gain_real; /**< Ganancia real del path*/
  float32_t f32_gain_imag; /**< Ganancia imaginaria del path*/

  //Variables para guardar el estado en caso de que se pase el fader de
    ON a OFF/THROUGH
  bool_t b_path_enable_old; /**< Path habilitado*/
  bool_t b_desvanecimientos_old; /**< Desvanecimientos habilitados*/
  int32_t i32_delay_old; /**< Delay del path*/
  float32_t f32_gain_real_old; /**< Ganancia real del path*/
  float32_t f32_gain_imag_old; /**< Ganancia imaginaria del path*/

} Path;
```

- **Timer:** Esta estructura recoge la información sobre el contador del timer, para después obtener la frecuencia deseada.

```
typedef struct Timer {
  float32_t f32_freq; /**< Frecuencia a la que se actualizará el valor
    para emular el efecto doppler/desvanecimientos*/
  uint32_t ui32_irq_micro; /**< valor con el que obtener obtener la
    frecuencia de interrupciones para actualizacion de coeficientes*/
} Timer;
```

- **Puertos Activos:** Esta estructura recoge la información sobre los puertos activos del emulador, es decir, si los canales están habilitados o no.

```
struct activePorts
{
  bool_t activeChannel[MAX_CHANNELS]; // true canal activo, false canal
    no activo
```

```
  float carrierFreq [MAX_PORTS]; // en MHz
  float signalBW [MAX_PORTS]; //en MHZ
  int CRC;
};
```

- **Estado del fader:** Esta estructura recoge la información sobre el fader de cada canal. El fader es un elemento que actúa sobre la señal de la siguiente manera: si esta en *ON*, permite a la señal pasa por el modelo de canal, en *OFF* no permite a la señal pasar por el modelo de canal, y en *THROUGH* la señal pasa por el path sin sufrir ningún modelado.

```
struct faderState
{
  int ID;
  int state[MAX_CHANNELS]; // on = 1, off = 0, through = -1
  int same_doppler_seed;
  int CRC;
};
```

- **Calibración Manual:** Mediante esta estructura se recoge la información acerca de la calibración de cada Front-end.

```
struct manualCal
{
  float powerInput [MAX_PORTS]; // en dBm
  float crestFactor [MAX_PORTS]; //en dB
  float att [MAX_PORTS]; //en dB
  int dummy; // para que tenga tamaño diferente
  int CRC;
};
```

- **Large Scale:** Esta estructura recoge información relacionada al pathloss y el shadowing de cada canal.

```
struct largeScale
{
  float pathLoss [MAX_CHANNELS]; // en dB
  float shadowing [MAX_CHANNELS]; //en dB
  int CRC;
};
```

- **Tap Info:** Esta estructura recoge información relacionada a los parámetros de cada path de cada canal *(A-B, A-C, B-C,...)* como la ganancia o el tipo de espectro Doppler.

```
struct tapInfo
{
      int ID;
      int indexTap;
      int indexChannel;
        float delay; // en ns
        float gain; // en dB
        int dopplerSpectrum; // Jakes = 1, bell = 2, Gausian = 3...
        float maxFDoppler; // en Hz
        int envelopeDistr; // Rayleigh = 1, Rice = 2...
      float kFactor;
      int CRC;
```

```
13 };
```

Cada vez que el procesador de tiempo real recibe un flujo de datos nuevo, compara su tamaño al tamaño de las estructuras anteriormente mencionadas. De esta manera, el procesador sabe a qué estructura corresponden los datos que ha recibido.

Después de cada comparación, el procesador ejecuta las operaciones necesarias relacionadas a cada estructura. Por ejemplo, en el caso de recibir una estructura *activePorts*, actualizará el valor de los canales que quieren ser activados, qué frecuencia portadora tendrán las señales de cada puerto, así como su ancho de banda. Una vez se han realizado todas las operaciones, el procesador reenvía la estructura mediante *rpmsg_send()* como señal de *ACK(acknowledged)*. En la figura 23, se observa el flujo de recepción de mensajes del procesador de tiempo real Cortex-R5.

Además de la aplicación, la plantilla *echo_test* genera un Linker Script donde se controla la posición en memoria de las diferentes secciones del archivo ejecutable. Como se observa en la figura 24, estas direcciones y tamaños en memoria que se definen en el Linker Script deben coincidir con las asignaciones de memoria que se realizan en el device-tree. Por defecto se asigna un tamaño de memoria de 0x00010000 posiciones a cada sección, para que la aplicación funcione es necesario aumentar el tamaño de memoria DDR asignado por defecto a la aplicación.

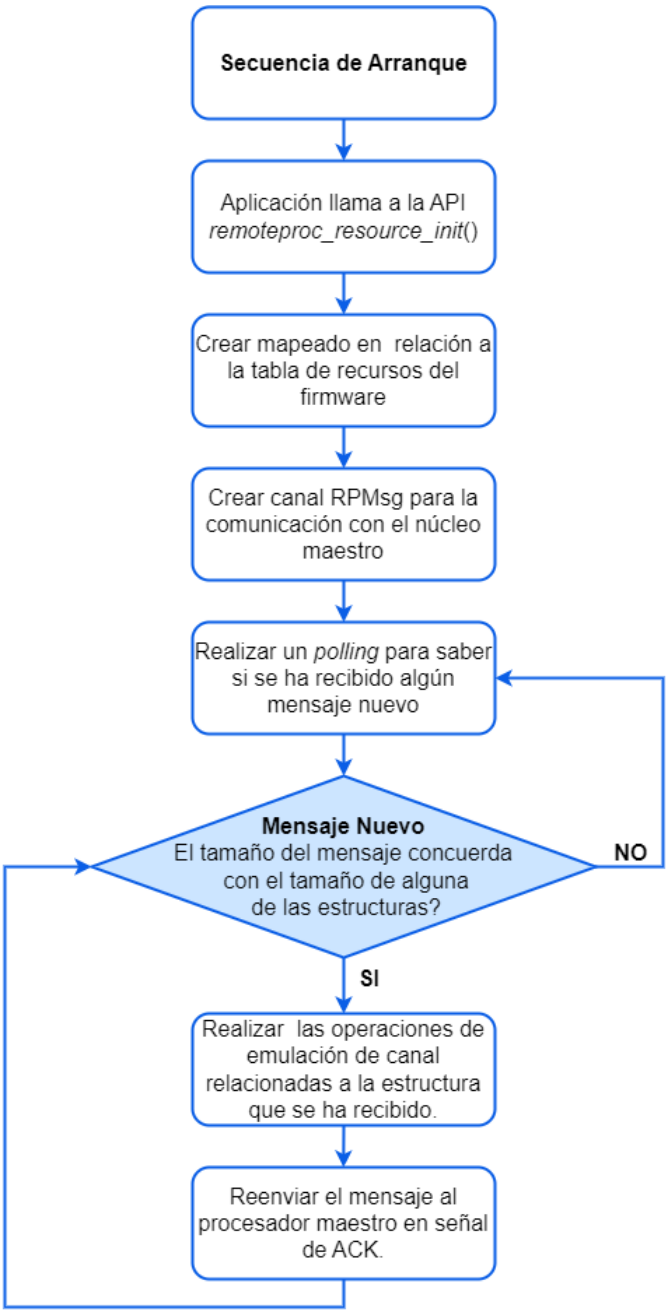


Figura 23: Flujo de ejecución de la recepción de mensajes del procesador Cortex R5.

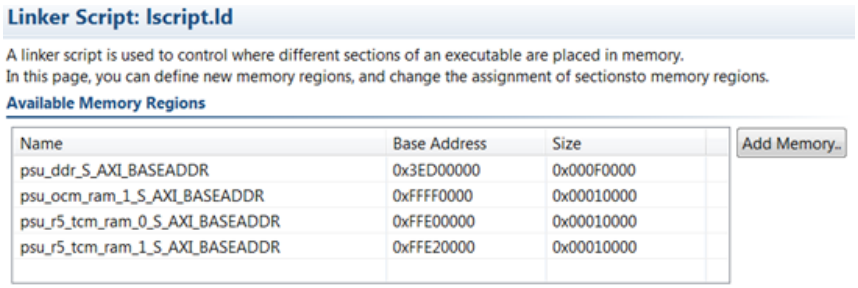


Figura 24: Linker Script de la aplicación.

3.1.3. Aplicación Linux en Cortex A53

Una vez se ha entendido el funcionamiento del procesador remoto R5, en este apartado se va a detallar el funcionamiento de la aplicación Linux en el procesador A53. Esta aplicación corresponde a la interfaz gráfica del emulador, que se encarga de recolectar los parámetros para la emulación del canal y enviarlos al procesador de tiempo real.

Esta aplicación se ejecuta sobre una distribución Linux, concretamente, el Linux que se construye en el apartado 3.1.1 con la herramienta Petalinux. La aplicación ha sido desarrollada mediante la herramienta *Qt Creator*. Para poder utilizar el software *Qt Creator* ha sido necesario utilizar un maquina virtual con Ubuntu.

Para poder entender mejor cómo funciona la aplicación y qué parámetros se configuran, a continuación, se detallan las partes más importantes de la interfaz gráfica:

- ***Pantalla Principal:*** La pantalla principal de la interfaz se muestra en la figura 16. Como se observa en la figura, en la pantalla principal se pueden encontrar varias opciones de configuración, así como unas notas acerca de la emulación y un apartado para los mensajes de Debug del procesador A53.

  En el margen izquierdo de la pantalla principal se encuentran las opciones de configuración relacionadas a los canales y los Front-End. Se configuran la frecuencia portadora de la señal y el ancho de banda. Además, se eligen los puertos (A, B, C o D) que van a ser utilizados y los canales (A-B, A-C, A-D, B-C, ...) que se van a utilizar. Una vez se configuran estos parámetros mediante la opción *APPLY* se envían las estructuras correspondientes a estos parámetros al procesador R5. Concretamente, la estructura *ActivePorts* que se explica en el apartado 3.1.2.

  En la parte inferior central se encuentran unas notas con información acerca de la emulación del canal. Por ejemplo, se detalla que la frecuencia de muestreo del sistema es 10 veces la definida en la frecuencia Doppler o que en el modo de calibración manual con 0dBm de entrada, 0 dBm de factor de cresta y 0dBm de atenuación el emulador atenúa 18 dBm por defecto a una frecuencia de 2.45 GHz.

  En el margen central-derecho de la pantalla se muestra un apartado relacionado con el estado del fader de cada canal. En este apartado, si se clica sobre cada opción de fader (Fader A-B, Fader A-C, etc) se abre una pantalla donde podemos configurar los parámetros de los paths de cada canal. Cuando se configuran los parámetros y se manda la información al procesador R5 al clicar en *RUN*, se da comienzo a la emulación del canal. El fader de cada canal puede estar en 3 estados distintos. Cuando esta en *ON*, el emulador realiza la emulación del modelo de canal elegido, en *OFF* el emulador no permite la emulación del canal y en la opción *THROUGH* la señal pasa por el emulador sin ser alterado por ningún modelo de canal.

  Para finalizar, en el margen derecho de la pantalla se muestra una consola de debug del procesador A53. En esta consola se muestra la información acerca de la transmisión y recepción de las estructuras mediante openAMP entre los procesadores maestro y remoto.

- ***Pantalla de los fader:*** Esta pantalla se muestra cada vez que se quieren configurar los parámetros de los paths de cada canal al clicar sobre el botón de los faders *(Fader A-B, Fader A-C, etc)*. En este menú se configuran parámetros de cada path como la ganancia, el delay, la distribución Doppler, etc. Se pueden añadir hasta 10 paths por cada canal. Una vez se configura el fader se clica en OK y en la pantalla principal se

clica en *RUN*. Los parámetros que se han definido en esta pantalla con información de cada canal se envían en la estructura *tapInfo* al procesador R5.

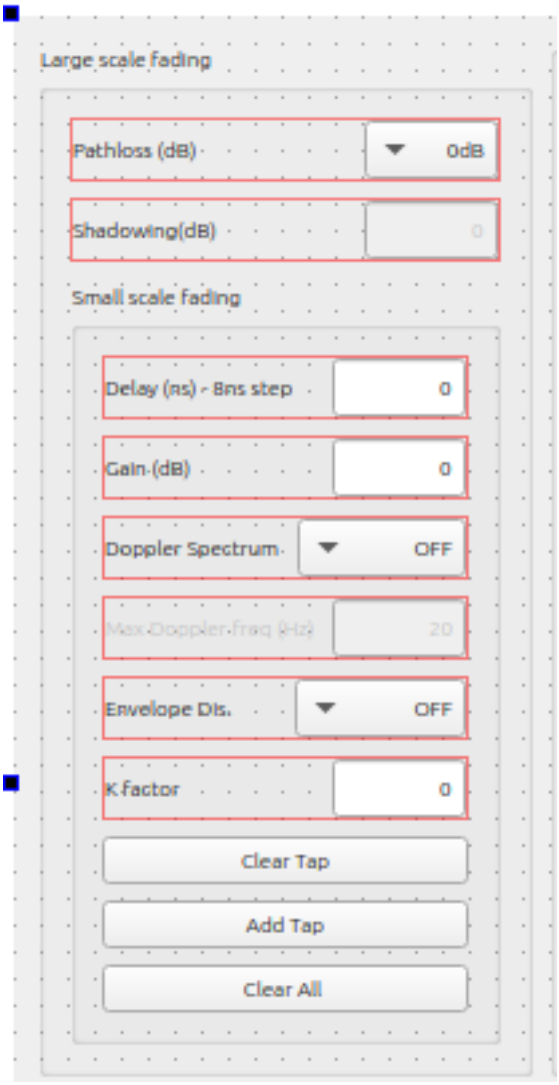


Figura 25: Pantalla de configuración de los paths de cada canal.

- ***Pantalla de Calibración:*** Esta pantalla se muestra al clicar sobre la opción *MANUAL CAL* de la pantalla principal. Como se muestra en la figura 26, este menú nos permite configurar varios parámetros acerca de los elementos de cada Front-End: la potencia de entrada, el factor de cresta y la atenuación. Cuando se ejecuta la opción *RUN* de la pantalla principal, los valores recogidos en esta pantalla se envían dentro de la estructura *manualCal* descrita en el apartado 3.1.2.

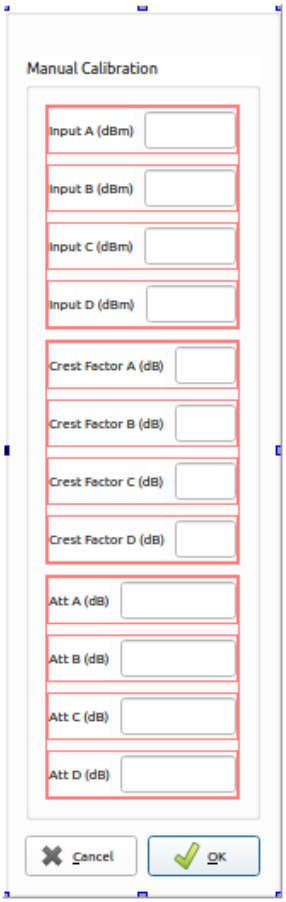


Figura 26: Pantalla de configuración de la calibración de los elementos del Front-End.

3.1.4. Desarrollo de la funcionalidad

Una vez se ha comprendido el funcionamiento de las aplicaciones de los dos entornos procesadores del sistema, en este apartado de van a detallar los cambios que se han realizado en el actual diseño para poder realizar la transmisión de unos parámetros recogidos desde un fichero por parte de la interfaz gráfica al procesador de tiempo real. Este último leerá estos valores y los aplicará a la emulación del canal. Se explicarán todos los pasos seguidos desde el inicio del proyecto hasta la implementación de la nueva funcionalidad.

El objetivo principal de añadir esta funcionalidad es la posibilidad de emular un canal de comunicaciones a una frecuencia de Doppler de 300 Hz durante 1 minuto aproximadamente, sin que los valores de los coeficientes se repitan en la emulación del canal durante ese tiempo. Para ello, se debe tener en cuenta que la frecuencia de muestreo del procesador de tiempo real para la emulación del canal, es decir, la frecuencia a la que se actualizan los valores de los coeficientes que actúan sobre la emulación, viene dada por la siguiente expresión:

$$F_s = 10 * F_{doppler} \tag{3.1}$$

Por lo que, a esa frecuencia Doppler de 300 Hz, los valores de los coeficientes se deben actualizar 3000 veces por segundo. Teniendo en cuenta que el tiempo total de emulación

que se pretende conseguir es aproximado al minuto, se obtiene el numero de valores que debería contener el fichero para poder realizar la emulación:

$$N = t_s * F_{doppler} = 60s * 3000s^{-1} = 180000 \tag{3.2}$$

En resumen, se debe enviar un fichero que contenga el valor de 180000 coeficientes para poder emular un canal de comunicaciones con una frecuencia Doppler de 300 Hz durante 1 minuto aproximadamente.

3.1.4.1. Primer paso: Reproducir un array en el procesador R5

Los primeros pasos para poder desarrollar la funcionalidad han sido las de familiarizarse con los entornos de programación y diseño del sistema. Para poder hacer uso de las herramientas*Xilinx SDK* y Petalinux correctamente, se ha seguido el tutorial de diseño de Xilinx acerca de la tarjeta Zynq Ultrascale+[35]. De esta manera, se ha entendido el proceso de diseño de una aplicación para la Zynq Ultrascale+.

Primero, se describe la lógica programable que se va a utilizar mediante la herramienta Vivado. Después, se exporta el hardware de Vivado y mediante*Xilinx SDK* se crea la aplicación de software para el sistema procesador, eligiendo el tipo de plataforma del sistema operativo: bare-metal, freeRTOS o Linux. Si se pretende generar una aplicación para Linux, se hace uso de Petalinux como se ha descrito en el apartado 3.1.1. Finalmente, se generan los ficheros necesarios para la ejecución y se elige cual será el modo de arranque del sistema: JTAG, SD o QSPI.

Una vez se ha entendido el flujo de diseño, se debe tener en cuenta que en este caso no se realiza ningún cambio en la parte lógica programable del diseño, por lo que se trabaja directamente sobre los entornos procesadores A53 y R5. Para la generación de la aplicación Linux de la interfaz gráfica se utiliza el programa *Qt Creator*, mientras que para el diseño del R5, se utiliza*Xilinx SDK*. Para poder implementar las funciones de lectura, transmisión y escritura del fichero y realizar la comunicación del diseño, lo primero ha sido entender el funcionamiento de los procesadores y analizar el código en C de las aplicaciones.

Para comprender cómo funciona el código del procesador R5, el primer paso ha sido encontrar la parte del código donde se generan los valores de coeficientes y se escriben para su transmisión mediante AXI GPIO. Se ha observado que la generación de los coeficientes se produce al momento de comparar si los paths de cada canal están activados o no. Si el path está activado, en base al tipo de espectro Doppler se generan unos coeficientes u otros para la emulación, esto se observa en la figura 27.El emulador trabaja con valores en formato I/Q, por lo que es necesario que el fichero que se pretenda reproducir posea coeficientes en ambos formatos.

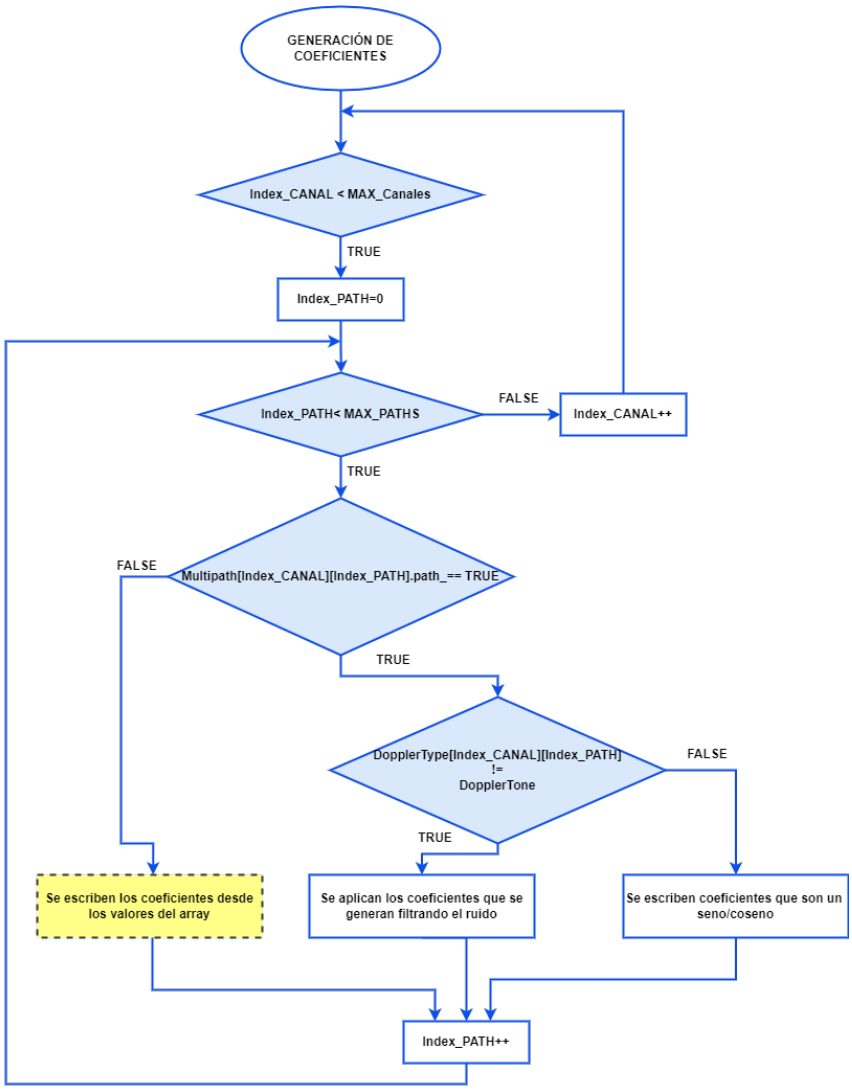


Figura 27: Diagrama del flujo de ejecución de la escritura de los coeficientes de emulación.

Como primera prueba, se han definido dos arreglos; uno correspondiente a los valores reales y el otro a los imaginarios. Estos arreglos se definen en el código de aplicación del R5, y se rellenan con valores predefinidos. En este caso, para realizar la prueba el valor de los elementos de los dos arreglos es el mismo:

```
for(int  i=0;i<ARRAY_SIZE;i++){
     array_gorka_r[i]=aux;
     array_gorka_i[i]=aux;
     if(aux==7){
       aux=0;
     }else{
       aux=aux+1;
     }
}
```

Como se observa en el código, los arreglos se inicializan siguiendo una secuencia de suma desde 0 a 7 sucesivamente hasta que se termine el tamaño del array. Una vez inicializado los valores, se añaden las siguientes líneas a la parte de código donde se realiza la escritura en los coeficientes GPIO. En este caso se ha seleccionado de manera temporal, que se realice la emulación del canal con los valores del array cuando los paths estén

deshabilitados, es decir, que cuando no se configure el path de un canal mediante la interfaz el valor del coeficiente que se escribe en el procesador R5 sea el del array. Esto se observa en el recuadro amarillo del flujo de proceso de la figura 27.

```
x=array_gorka_r[aux];
y=array_gorka_i[aux];
aux =aux+1;
if (aux==(ARRAY_SIZE-1)){
  aux=0;
}

i16_coef_real = x * 4096;
i16_coef_imag =y * 4096;

ui16_coef_real = (uint16_t) i16_coef_real;
ui16_coef_imag = (uint16_t) i16_coef_imag;
ui32_variable_c2 = ui16_coef_real + 0x10000 * ui16_coef_imag;

escribir_coef(ui32_variable_c2, channel_index, tap_index);
```

En la primera parte del código, se leen los valores del array a partir del índice "aux", después se realiza la actualización del valor de este índice y se comprueba si se ha llegado al final del tamaño de los arreglos, si es así, el valor del índice se reinicia a 0. Después se aumenta el valor de los coeficientes multiplicándolos por el 4096. Esto se hace para ajustar el rango dinámico del ADC y mejorar la visualización de la forma de onda de los valores del array en la salida del emulador. Por último, se realiza una conversión del tipo de dato de los coeficientes para la escritura de estos mediante GPIO. Para finalizar se llama a la función *escribir_coef* que realiza la escritura del coeficiente teniendo en cuenta el índice del canal y el path.

Una vez se ha desarrollado el código, es necesario comprobar su funcionamiento poniendo el emulador en marcha. El método de arranque elegido para el arranque del emulador es una memoria SD. La memoria incluye dos particiones: BOOT y *ROOTFS*. La primara corresponde a la partición de arranque que contiene los archivo BOOT.bin e image.ub. La partición *ROOTFS* hace referencia al sistema de archivos del Linux creado para el procesador Cortex-A53. En ella se encuentran los ejecutables para las aplicaciones de los procesadores A53 y R5.

A continuación, se van a mostrar los pasos seguidos para el arranque del emulador.

1. El primer paso es preparar la memoria SD con todos los archivos ejecutables para la emulación. Para ello, se compilan las aplicaciones de ambos procesadores para obtener los ficheros ejecutables *".elf"*. El archivo correspondiente al procesador R5 se copia dentro de la partición *ROOTFS* en el directorio *"\lib\firmware"*. El archivo de la aplicación del A53, sin embargo, se copia dentro del directorio *"\usr\bin\gorka"*. Si se han realizado cambios en el diseño hardware o en algún aspecto de Petalinux, se vuelven a generar los archivos BOOT.bin e *image.ub* después de compilar el proyecto mediante el comando *petalinux-package*, como se explica en el apartado 3.1.1. Estos archivos se copian en la partición BOOT de la SD.

2. El segundo paso es encender la tarjeta y comprobar que el Linux del procesador A53 arranca correctamente. Para ello, se hace uso de un programa que permita acceder mediante puerto serie a la tarjeta, en este caso se ha utilizado el software *Tera-Term*. Mediante la consola accedemos al Linux introduciendo el usuario y contraseña, en este caso *root/root*. Dentro del sistema de archivos, nos movemos hasta el directo-

rio *"\etc\init.d"*. Dentro de este directorio se encuentra el archivo *ini_emulador.sh*, el cual es el encargado de inicializar la interfaz gráfica y establecer la comunicación openAMP entre ambos procesadores. Su contenido es el siguiente:

```
export QT_QPA_PLATFORM_PLUGIN_PATH=/usr/lib/platforms
export QT_QPA_FONTDIR=/usr/lib/fonts
export QT_PLUGIN_PATH=/usr/lib/plugins

export QT_QPA_GENERIC_PLUGINS=evdevmouse, libinput
export QT_QPA_EVDEV_MOUSE_PARAMETERS=/dev/input/event1
export QT_QPA_FB_HIDECURSOR=0

export QT_PLUGIN_PATH=/usr/lib/plugins

touch /etc/machine-id
echo abcdabcdabcdabcdabcdabcdabcdabcd > /etc/machine-id

modprobe rpmsg_user_dev_driver

cd /lib/firmware/
echo Emulador_R5.elf > /sys/class/remoteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state

/usr/bin/gorka/EMULV2_GORKA -platform linuxfb -plugin libinput
-plugin evdevmouse -plugin evdevkeyboard &
```

El script comienza estableciendo varias variables de entorno relacionadas con la configuración de Qt. Las variables de entorno establecidas incluyen la ubicación de plugins, directorios de fuentes y otros parámetros relacionados con la interfaz de usuario.

Luego, el script crea un archivo de identificación de máquina y carga un controlador de dispositivo. A continuación, cambia el directorio de trabajo y se indica al componente *Remoteproc* el programa *.ELF* que debe cargar en memoria e iniciar su ejecución en uno de los procesadores de tiempo real. Posteriormente, el comando *"echo start"* inicia la ejecución del firmware. Por último, se carga la aplicación gráfica llamada *EMULV2_GORKA* y se espera que se ejecute.

3. Una vez se ha lanzado la interfaz gráfica, solo queda configurarla para realizar las pruebas correspondientes. En este caso, se elige una frecuencia de 2.45 GHz con un ancho de banda de 100 MHz, se eligen los puertos A y B y se activa el canal A-B. Después se elige la frecuencia Doppler, 300 Hz y se clica en *APPLY*. En este momento, se carga una barra de progreso mientras se configuran los elementos de los Front-End A y B. Una vez terminado, se configura el fader del canal elegido. Se pueden añadir hasta 10 paths por canal, con varias configuraciones distintas. En este caso, para realizar la prueba del array, se ha elegido que los coeficientes del array se apliquen cuando el path de los canales esté deshabilitado, por lo que después de añadir un solo path, el mínimo necesario para poder realizar la emulación, se clica en *RUN* para la ejecución del canal.

Finalmente, una vez se ha configurado el emulador, es necesario comprobar que la configuración del generador de señales que entra al emulador esté dentro del ancho de banda que cubre el emulador 2.4-2.5 GHz y que la potencia de entrada de la señal sea suficiente como para que el Front-end detecte que existe una señal de entrada. Por último, se visualiza la señal de salida del emulador mediante un osciloscopio. En la figura

28, se muestra una captura del osciloscopio de la señal de salida del emulador, después de ser alterada por los coeficientes del array predefinido en la aplicación del Cortex R5.

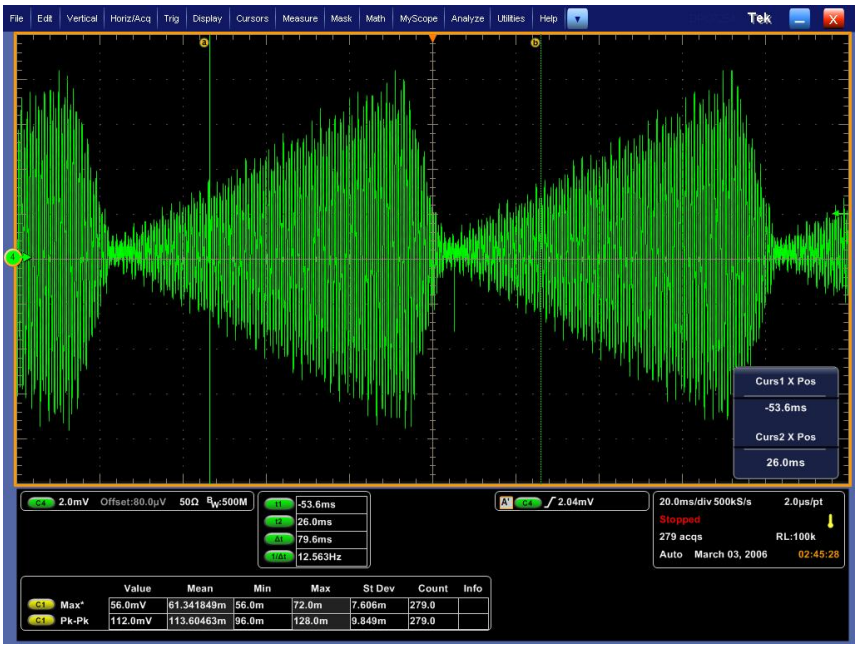


Figura 28: Captura de la señal de salida del canal A-B del emulador con osciloscopio.

3.1.4.2. Segundo paso: Leer un archivo desde el A53

Para poder enviar la información de los parámetros del fichero, es necesario que la aplicación del emulador sea capaz de leer esta información. Teniendo en cuenta que el emulador trabaja con coeficientes en formato I/Q, el archivo contiene dos líneas de valores con los parámetros de la emulación del canal, la primera con los valores reales y la segunda con los imaginarios. El objetivo es que la aplicación del emulador lea las líneas de este archivo y vuelque la información en dos arreglos, uno para el formato real y el otro para el formato imaginario.

Para conseguir que la aplicación lea el fichero, se ha utilizado la herramienta *Qt Creator*, que fue utilizada para el diseño de la interfaz gráfica. Mediante el siguiente diseño, se consigue volcar los valores de los ficheros a dos arreglos:

```
constexpr int ARRAY_SIZE = ;

#include <QFile>
#include <QDebug>
#include <QString>
#include <QCoreApplication>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    float array_R [ARRAY_SIZE];
    float array _I[ARRAY_SIZE];
    QFile f("/home/emulador/Desktop/file_prueba.txt");
    bool ok;

    if (!f.exists()) {
        qCritical() << "File does not exist";
```

```
19          return 1;
20      }
21
22      if (!f.open(QFile::ReadOnly)) {
23          qCritical() << "Unable to open file";
24          return 2;
25      }
26
27      QTextStream stream(&f);
28
29      int i = 0;
30      while (i < ARRAY_SIZE && !stream.atEnd()) {
31          const auto line = stream.readLine();
32          const auto values = line.trimmed().split(" ");
33          for (const auto& value : values) {
34              const auto number = value.toFloat(&ok);
35              if (ok) {
36                  array_R[i++] = number;
37              }
38          }
39      }
40
41      i = 0;
42      while (i < ARRAY_SIZE && !stream.atEnd()) {
43          const auto line = stream.readLine();
44          const auto values = line.trimmed().split(" ");
45          for (const auto& value : values) {
46              const auto number = value.toFloat(&ok);
47              if (ok) {
48                  array_I[i++] = number;
49              }
50          }
51      }
52
53      for (int i = 0; i < ARRAY_SIZE; i++) {
54          printf("VALOR ARRAY_R [%d]: %f\n", i, array_R [i]);
55      }
56      for (int i = 0; i < ARRAY_SIZE; i++) {
57          printf("VALOR ARRAY_I [%d]: %f\n", i, array_I[i]);
58      }
59
60      f.close();
61
62      return a.exec();
63  }
```

El programa comienza por definir la constante *ARRAY_SIZE*, este valor hace referencia al numero de parámetros que tiene el fichero, seguido de la inclusión de las librerías necesarias.Para la primera prueba se utilizaron ficheros con diferentes tamaños para comprobar que en todos los casos los valores del fichero se leían correctamente. Después, se crea una instancia de *QCoreApplication* para manejar los argumentos de entrada.

El programa primero verifica si el archivo existe y puede ser abierto. Si el archivo no existe o no puede ser abierto, el programa muestra un mensaje de error y termina la ejecución. Si el archivo puede ser abierto, el programa crea un objeto *QTextStream* a partir del objeto *QFile* para poder leer el contenido del archivo.

El archivo se lee línea por línea y se separa cada línea en elementos individuales usando el método *split()* de la clase *QString*. Cada elemento se convierte en un número flotante y se almacena en los vectores *array_R* y *array_I*. Finalmente, se utiliza el comando

"*printf*" para mostrar los valores de los arreglos en consola. En la figura 29, se muestra una captura de la consola de Qt mostrando el valor de los elementos del array.

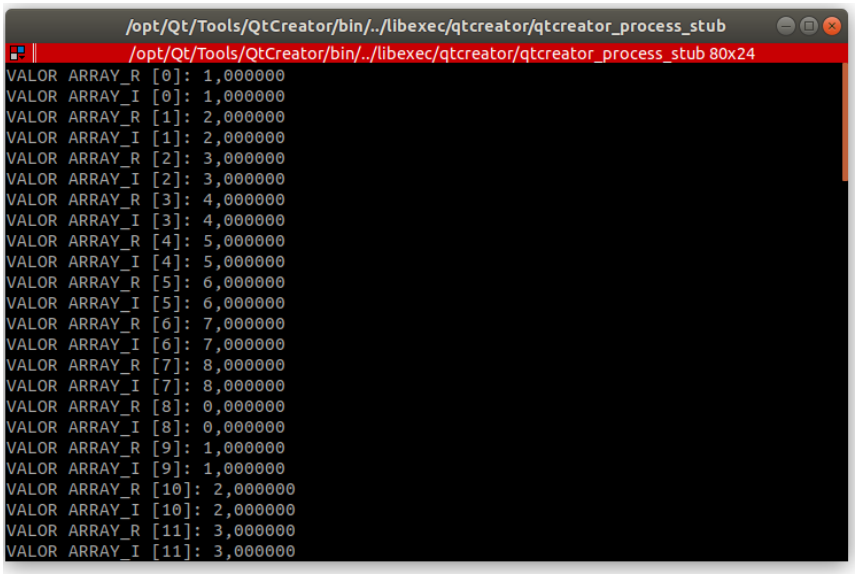


Figura 29: Captura de la consola de Qt mostrando los valores de los arreglos I/Q.

Teniendo en cuenta que los valores del fichero siguen la misma secuencia que la inicialización de los array representada en el apartado 3.1.4.1, y observando la figura, se demuestra que los valores se leen correctamente.

3.1.4.3. Tercer paso: Establecer la comunicación entre los procesadores

Hasta ahora, se ha conseguido reproducir un array con valores predefinidos en el procesador R5 y realizar la lectura de un archivo con parámetros para la emulación en el procesador A53. No obstante, la parte primordial del desarrollo de la nueva funcionalidad es que los valores del fichero se transmitan de un procesador a otro mediante la comunicación con el framework openAMP.

Para conseguirlo, es necesario adaptar el código actual de la aplicación de la interfaz, así como el del procesador R5. La interfaz gráfica debe añadir una opción de carga del fichero para habilitar la lectura del fichero y la transmisión de los parámetros encapsulados en una estructura. A su vez, se debe actualizar la función de recepción de mensajes del procesador R5, para que pueda leer los mensajes relacionados a la transmisión del fichero.

Para poder efectuar la emulación de los parámetros del fichero en el procesador R5, es necesario saber en qué parte del código añadir esta opción. Como se ha descrito en el apartado 3.1.4.1, los coeficientes se escriben dependiendo del tipo de espectro Doppler. Por lo que, para realizar la ejecución de los valores del fichero, se generará un nuevo tipo de opción de espectro Doppler que hará referencia a la transmisión y recepción del fichero.

El primer paso es la implementación de este nuevo tipo de espectro Doppler en la interfaz gráfica. Para ello se añade una opción llamada "File" al menú desplegable del espectro Doppler dentro de la pantalla del Fader, como se observa en la figura 30. Esta información indica al R5 que debe aplicar los parámetros que recibe en los arrays de

la estructura relacionada a la transmisión del fichero. En los casos anteriores, cuando se envía un tipo de espectro "Jakes"(tipo de distribución que se suele dar en canales en los que los nodos que se están comunicando están en movimiento), por ejemplo, el propio R5 es quien genera los coeficientes para ese tipo de espectro. En este caso, los coeficientes a ser aplicados se envían en la estructura *file_info* que se muestra más adelante.

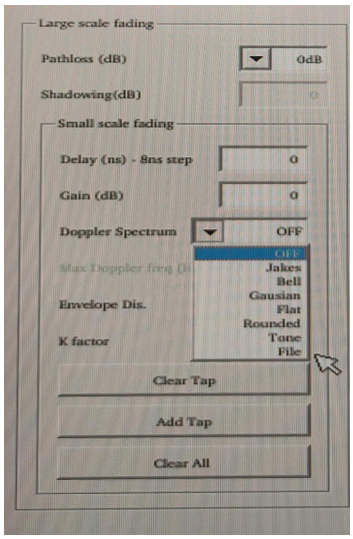


Figura 30: Menú de configuración del tipo de espectro Doppler con la opción "File".

Una vez se ha añadido la opción de *"File"*, se debe añadir el código que posibilite el envío de los arreglos reales e imaginarios con los parámetros del fichero. Para ello, se deben encapsular en una estructura, para que se envíen como si fuera un paquete y el procesador remoto, comprobando el tamaño del mensaje, sea capaz de interpretar el mensaje como si fuese una estructura relacionada con la transmisión del fichero. Para realizar esta transmisión se debe adaptar el fichero *"communicationthread.cpp"* del proyecto de *Qt Creator*. Este fichero es el encargado de la transmisión y recepción de los mensajes en el procesador A53. Este fichero se encarga de enviar las estructuras de configuración de la interfaz y comprobar que el procesador remoto confirme su recepción. Antes de explicar el algoritmo que se ha utilizado para posibilitar la comunicación conviene tener en cuenta algunos aspectos.

Los recursos de memoria compartida para habilitar la comunicación entre los procesadores son limitados, por lo que, para conseguir enviar un archivo con 180000 valores en formato I/Q es necesario que la información sea transmitida por partes. Si no, cuando se intenta transmitir un mensaje demasiado grande la comunicación falla y el sistema deja de responder. Para evitar esto, se ha creado un algoritmo que envíe la información del fichero en tramas de longitud fija, de esta manera se posibilita que la aplicación de recepción de mensajes del R5 comprenda que se trata de una trama correspondiente a la reproducción del fichero y que la información de los parámetros del fichero se transmitan correctamente. A continuación, se muestra la estructura de mensaje que se envía para poder transmitir la información del fichero:

```
struct file_info
{
    int File_id;
    float array_R[ARRAY_SIZE];
    float array_I[ARRAY_SIZE];
    int seq;
```

```
};
```

La estructura contiene dos arreglos correspondiente al formato I/Q con una longitud fija denominada *ARRAY_SIZE*, que como máximo debe tener un valor de 60. Puesto que, si se supera este valor el tamaño de la trama que se envía colapsa los recursos de memoria compartida entre ambos procesadores y la comunicación se termina. Además, se observan dos variables más, *File_id* y *seq*. El primero de los dos hace referencia al tipo de trama que se está enviando:

- ***File_id* = 1:** En este caso, la trama corresponde a los primeros valores del fichero, es decir, la trama que tiene el valor de *File_id* = 1, es la primera trama que se envía.
- ***File_id* = 0:** En este caso, la trama corresponde a los valores intermedios del fichero, es decir, la trama que se envía está entre la primera y la última.
- ***File_id* = -1:** En este caso, la trama corresponde a los últimos valores del fichero, es decir, la trama que tiene el valor de *File_id* = -1, es la última trama que se envía.

El segundo hace referencia al numero de trama que se ha enviado. Gracias a estos dos valores, la aplicación de recepción de paquetes del R5 puede comprobar si la trama que ha recibido corresponde al orden correcto y de esta manera, saber si alguna de las tramas se ha perdido. A continuación, se muestra el algoritmo de transmisión y recepción de las estructuras de ficheros del procesador A53:

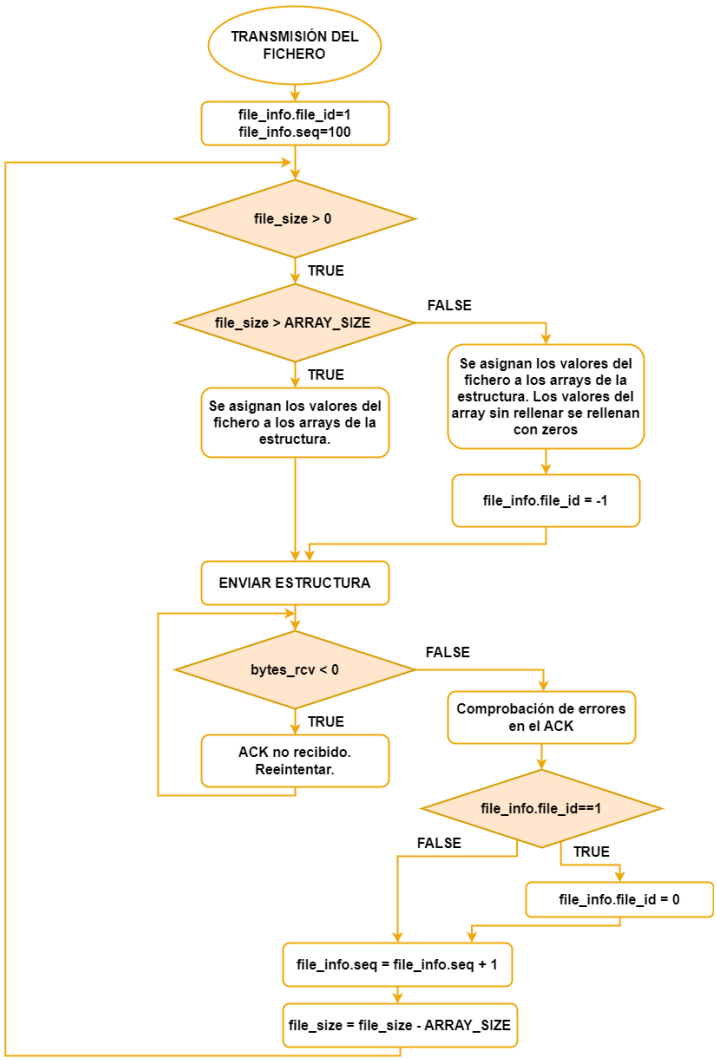


Figura 31: Diagrama del algoritmo de transmisión y recepción de estructuras *"File_info"* en el A53.

Una vez que se ha leído el fichero siguiendo el algoritmo descrito en la sección 3.1.4.2, se procede a la transmisión del mismo. Como se observa en la figura 31, se crea una instancia de la estructura *file_info*.

Primero, se determina el tamaño total del archivo y se almacena en la variable *"file_size"*. A continuación, se establecen algunos valores iniciales para la estructura *file_info*, como el identificador del tipo de trama, que se inicializa con el valor de 1 y el número de secuencia, que se inicializa con el valor de 100.

Luego, se comienza un bucle que se ejecuta mientras la variable *"file_size"* sea mayor que cero. Dentro del bucle, se realiza una verificación para determinar si el tamaño de la variable *"file_size"*es mayor o igual al tamaño definido para los arrays de la estructura, valor definido por la variable "*ARRAY_SIZE*". Si es así, se copian los valores del fichero en la estructura *file_info*. En caso de que el valor de *"file_size"* sea menor, significa que los valores por enviar del fichero no ocupan todos los espacios asignados a los arrays de la estructura. Estos espacios restantes se rellenan con zeros y se actualiza el valor del identificador de tipo de trama a "*-1*", puesto que, se ha llegado al final del fichero, y esta es la ultima trama que se debe enviar.

Después de enviar cada estructura *file_info*, el código espera recibir una confirmación

de que la estructura se recibió correctamente a través de la conexión. Si la confirmación no se recibe, el código reenvía la estructura un número limitado de veces. El código también comprueba que la estructura recibida de la confirmación coincida con la estructura original enviada, y si no se recibió una confirmación o si no coincide, el código emite mensajes de error.

Posteriormente, se añade la actualización del identificador del tipo de trama después de enviar la primera de estas, es decir, la primera vez que se envía la trama, el valor del identificador está a "1", y una vez se envía esta trama, se actualiza su valor a 0 y todas las demás tramas se envían con este valor, a excepción de la última trama, que como ya se ha explicado, recibe el valor de "*-1*". Finalmente, se actualiza el número de secuencia, sumando uno a su propio valor, y se resta la longitud de los arreglos de la estructura a la variable *file_size*. Esta variable actúa como contador descendente de los valores que se deben enviar. Cuando su valor es menor o igual a "0", significa que no quedan valores del fichero por transmitir, por lo que la transmisión del fichero se da por finalizada. Una vez que se han enviado todas las estructuras *file_info*, el código emite un mensaje indicando que la transmisión ha finalizado.

En resumen, la interfaz gráfica añade la opción de configurar el tipo de espectro Doppler como *"File"*. Cuando se elige esta opción en el fader, se envían las estructuras necesarias para la emulación así como las tramas relacionadas con los coeficientes del fichero mediante el algoritmo que se ha descrito. Para poder emular los parámetros, es necesario añadir un algoritmo de recepción en el procesador R5 que recibe las tramas relacionadas al fichero. En la siguiente figura se muestra el algoritmo de recepción de los mensajes relacionados a la transmisión de los coeficientes del fichero.

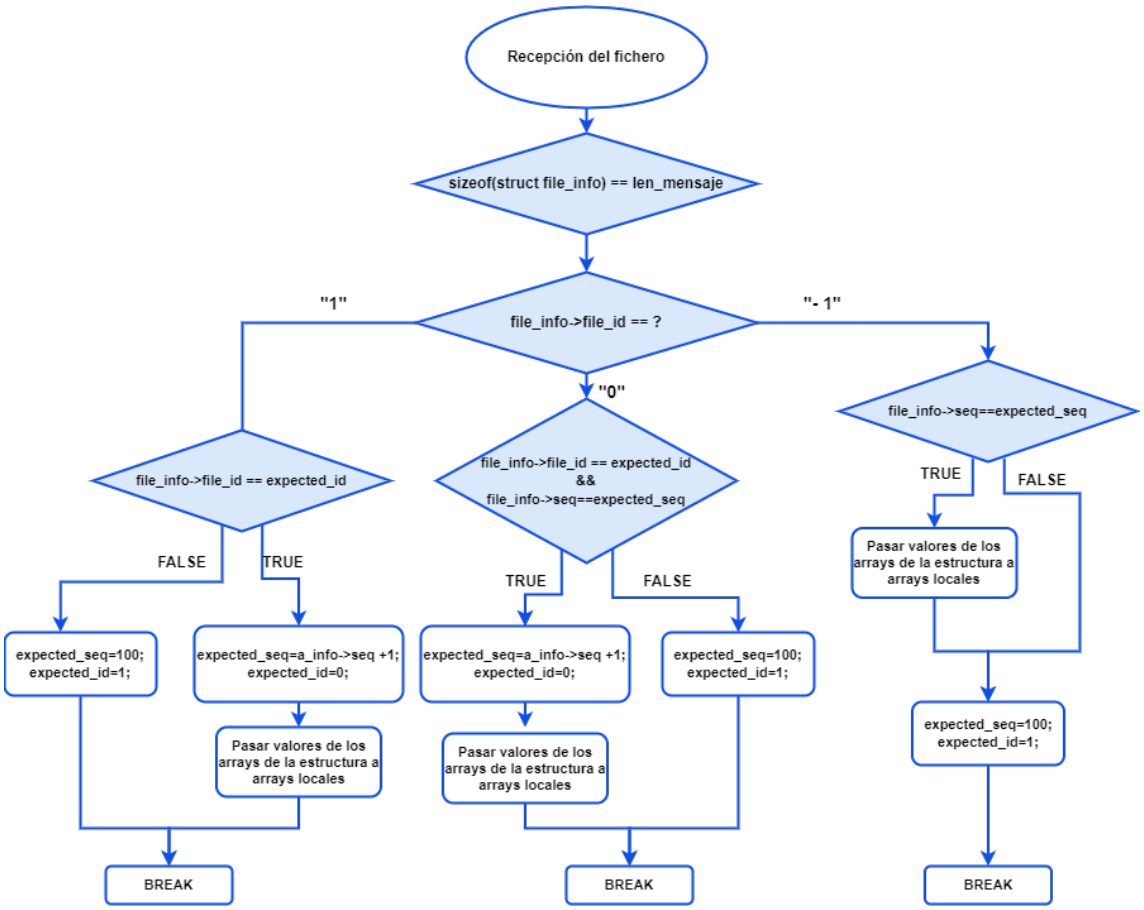


Figura 32: Diagrama del algoritmo de recepción de estructuras *"File_info"* en el R5.

El algoritmo comprueba si el mensaje que se ha recibido mediante la conexión openAMP es del tipo relacionado a la transmisión del fichero comparando el tamaño del mensaje recibido con el de la estructura del tipo *file_info*. En caso de confirmarlo, se vuelca la información del mensaje a una estructura tipo *file_info* y se comprueba el valor del *file_id*. Por cada tipo de trama recibida, se comprueba que su valor concuerde con el valor del tipo de trama que se espera. Por ejemplo, si después de recibir una trama del tipo *file_id*=1, que corresponde a la primera trama de la transmisión, se recibe otra del mismo tipo, algo ha fallado en la comunicación, aunque el número de secuencia sí corresponda con el esperado. Es por ello por lo que se comprueban el tipo de identificador y el número de secuencia de cada trama, con el valor de número de secuencia y tipo de trama esperado. Estos valores se actualizan cada vez que se recibe una nueva trama.

Cada vez que se recibe una trama y concuerda su identificador del tipo de trama y el número de secuencia con los valores esperados, los valores de los arrays de la estructura se pasan a los arrays real e imaginario locales, los cuales tienen el tamaño total del fichero. Gracias a una variable que se utiliza como índice, se va recorriendo los arrays y se añaden los valores recogidos en cada trama hasta que se recibe la última.

Cuando se recibe la ultima trama, *file_id=-1*, los valores esperados del identificador del tipo de trama y el numero se secuencia se inicializan a los valores por defecto, preparándose así para la recepción de otro fichero.

Después, cuando la transmisión del fichero ha finalizado, se da inicio a la emulación de los coeficientes. Como se muestra en el apartado 3.1.4.1, dependiendo del tipo de espectro Doppler se ejecutan unos coeficientes u otros. Anteriormente, se ha añadido una nueva opción de tipo Doppler a la interfaz llamada *"File"* por lo que, se debe actualizar el código del procesador R5 para realizar las operaciones correspondientes con los coeficientes del fichero cuando el path esté configurado con un espectro Doppler del tipo *"File"*. A continuación, se muestra el código que se ha añadido para la recepción de este tipo de mensajes:

```
if(doppler_type[channel_index][tap_index] == DopplerFile){ //Si es un doppler
    tipo file, aplicar valor de array

  x=array_gorka_r[aux];
  y=array_gorka_i[aux];
  aux =aux+1;

  if (aux==(SIZE_FILE-1)){
    aux=0;
  }

  i16_coef_real = x * 4096;
  i16_coef_imag =y * 4096;

  _coef_real = (uint16_t) i16_coef_real;
  _coef_imag = (uint16_t) i16_coef_imag;

  ui32_variable_c2 = ui16_coef_real + 0x10000 * ui16_coef_imag;
  escribir_coef(ui32_variable_c2, channel_index, tap_index);

}
```

Siguiendo los mismo pasos que se han detallado el apartado 3.1.4.1, se aplican los valores de los arrays a los coeficientes de la emulación. Cuando se llega al final de los valores del array, se reinicia el valor del índice a 0, para poder continuar con la emulación.

Gracias a los cambios que se han realizado en este apartado, se ha conseguido transmitir un fichero con parámetros de los coeficientes del canal mediante la comunicación openAMP. En el apartado 4, se detallan los resultados que se han conseguido acerca de la funcionalidad.

3.2. Estimador de potencia para ganancia automática

La segunda funcionalidad que se pretende desarrollar consiste en diseñar e implementar un estimador de potencia con el objetivo de que el ajuste de la ganancia de los elementos del Front-End se realice automáticamente. En el diseño actual, la configuración de los elementos del Front-End se realiza manualmente mediante la interfaz, como se muestra en la figura 26.

Por lo tanto, cuando se configuran estos valores del emulador dependiendo de la señal de entrada que se va a utilizar, el emulador de canal funciona correctamente. Pero si la potencia de la señal de entrada no está controlada, el emulador puede llegar a atenuar la señal demasiado o incluso aumentar tanto su potencia a un punto donde puede saturar la entrada del ADC.

Ante esta problemática, el objetivo de esta nueva funcionalidad es crear un estimador de potencia que calcule la potencia de la señal de entrada y dinámicamente configure los elementos del Front-End. Lo primero de todo, es crear el diseño del estimador de potencia mediante System-Generator e incluirlo en el diseño actual del emulador. Después, se actualiza la IP correspondiente al diseño de System-Generator en Vivado y se establecen los valores de la potencia de entrada como GPIOs para poder leer los valores desde el procesador R5. Por último, el procesador R5 leerá estos valores de potencia y actualizará los valores de los elementos del Front-End.

3.2.1. Diseño del estimador en System Generator

El primer paso ha sido realizar el diseño del estimador de potencia con la herramienta System Generator. Xilinx System Generator es un complemento de MATLAB Simulink que permite el desarrollo de diseños FPGA a nivel de arquitectura mediante la programación de bloques. Los usuarios pueden validar los diseños mediante simulación en Simulink y el diseño puede empaquetarse en una IP e importarse fácilmente a un proyecto Vivado.

Para realizar el diseño del estimador se debe tener en cuenta que el emulador trabaja con señales en formato I/Q. Debido a esto, el estimador debe leer los valores de la señal en este formato y realizar el cálculo de la potencia en base a ellos. La potencia instantánea de una señal se define como la magnitud al cuadrado de la señal en un instante de tiempo dado. En el formato I/Q, la magnitud se define como la raíz cuadrada de la suma de las magnitudes al cuadrado de las señales I y Q en cada instante de tiempo. Por lo tanto, la ecuación de la potencia instantánea se define de la siguiente manera:

$$P(t) = I(t)^2 + Q(t)^2 \tag{3.3}$$

Es importante tener en cuenta que la potencia instantánea puede variar con el tiempo, ya que depende de la señal en cada instante de tiempo. Para obtener la potencia total de la señal, se debe realizar la media de un numero de muestras durante un tiempo

de medición. Por lo que la potencia total de la señal se puede definir de la siguiente manera:

$$P = \sum_{n=0}^{N} P(n) \cdot \frac{1}{N+1} \tag{3.4}$$

De esta manera se puede conseguir el valor de la potencia de entrada gracias a estos cálculos. Teniendo en cuenta estas dos fórmulas, se ha realizado el diseño del estimador de potencia. A continuación, se va a explicar el diseño del estimador; en qué parte del diseño se ha colocado, las diferentes partes que componen el bloque, así como las señales de entrada y salida del bloque. En la figura 33 se muestra el bloque del estimador.

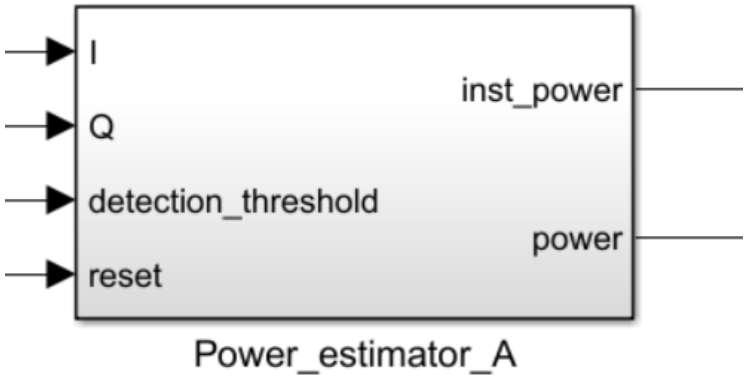


Figura 33: Bloque del estimador de potencia de System Generator.

Por un lado, el estimador recibe cuatro señales de entrada, las dos primeras son el valor en formato I/Q de la señal de entrada. Estas dos señales llegan al estimador desde un bloque que ha pasado la señal de entrada de frecuencia intermedia a banda base y ha generado las señales de I/Q. La señal *"detection_threshold"* hace referencia al nivel mínimo de potencia que debe tener la señal de entrada. Si la potencia de la señal entrante es inferior al valor de *"detection_threshold"*, no se tiene en cuenta, y se entiende que no existe señal entrante. La señal *reset"* se utiliza para resetear el valor de la potencia media calculada por el estimador.

Por otro lado, el bloque del estimador tiene dos señales de salida. La primera señal "*inst_power"* es la señal de la potencia instantánea de la señal, mientras que, la señal "*power*", es la potencia media de la señal calculada después de medir un numero de muestras. Para entender mejor como se generan estas dos señales de salida, a continuación, se muestra el diseño del bloque:

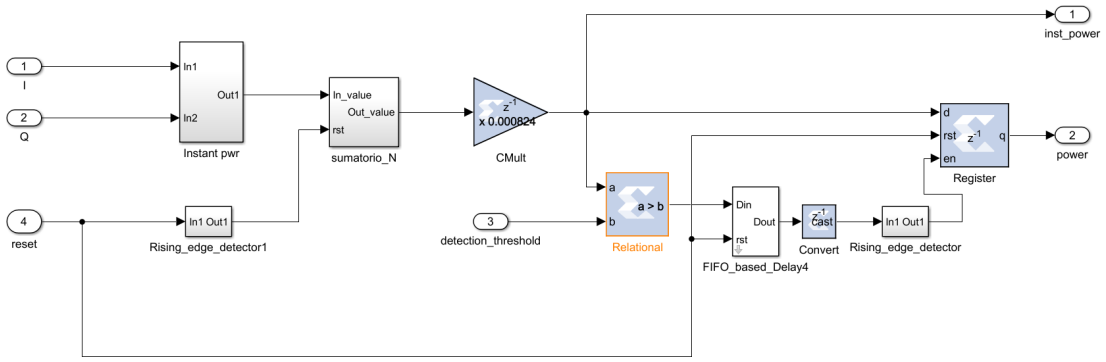


Figura 34: Diseño del interior del estimador de potencia de System Generator.

Como se muestra en la figura 34, el primer paso es calcular la potencia instantánea de la señal de entrada. Para ello, se introducen los valores I/Q en el bloque llamado "*Instant pwr*". Este bloque aplica la ecuación 3.3, primero multiplica al cuadrado los valores de las señales I/Q y después las suma, obteniendo así la potencia instantánea de la señal. Después la señal generada entra al bloque *"sumatorio_N"*, dentro de este bloque se encuentra un sumatorio que recoge "N" valores de potencia instantánea y los suma. Cuando ya ha recogido N muestras, cada vez que llegue un nuevo valor de potencia instantánea, se resta el primer valor guardado para hacer la media, y de esta manera se va actualizando el valor. El bloque cuenta también con la entrada de la señal de reset, cuando se activa se resetea el valor de la suma. Del bloque sale el valor de la suma de N muestras y pasa por un bloque que divide el valor de la suma de las muestras por el numero de muestras "N". De esta manera se consigue el valor medio de la potencia. Este es el valor que se le da a la potencia instantánea de la señal de salida.

Por otra parte, este valor se compara con la señal "*detection_threshold*", en caso de que supere el valor del threshold, se envía un valor de "1" a una FIFO con un retardo de *N* muestras. Cuando el valor de la potencia instantánea ha superado el threshold después de *N* comprobaciones, se activa el registro y el valor de la potencia instantánea se convierte en el valor de la potencia final del estimador. Esto se realiza para evitar que la medición de la potencia sea incorrecta. Ya que, la potencia instantánea no se estabiliza hasta que se han medido por lo menos N muestras de la señal y además, se debe tener en cuenta que el valor de la potencia medida sea superior al valor fijado por el threshold. Una vez se ha estimado el valor medio de la potencia, es necesario recibir una señal de reset para reiniciar la medición.

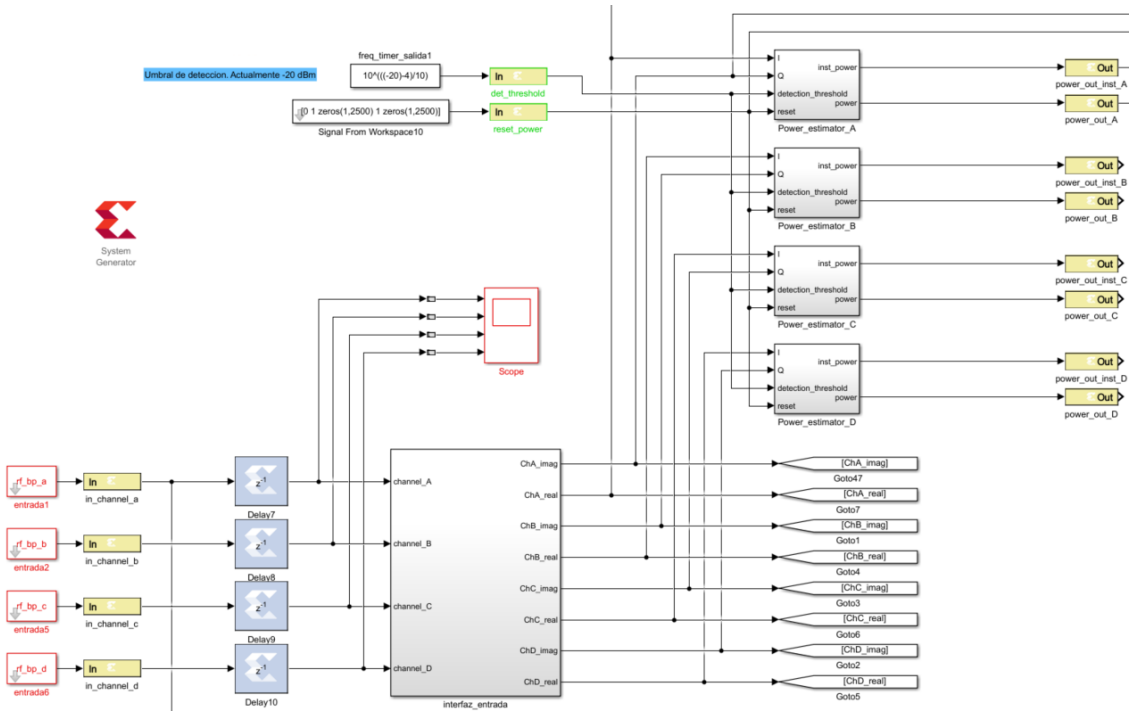


Figura 35: Captura de las señales de salida de la potencia de los puertos A y B del diseño de Sytem Generator.

Una vez se ha entendido cómo se realiza el cálculo de los valores de la potencia instantánea y de la potencia total, como existen cuatro puertos diferentes (A, B, C y D), se añade un estimador a cada uno de ellos. Como se muestra en la figura 35, cada estimador se añade después del bloque conocido como "*interfaz_entrada*", este bloque recibe las señales de los cuatro canales en IF y los pasa a banda base. Además, convierte las señales entrantes a formato I/Q, para poder enviarlas a los bloques donde se realiza la emulación del canal. Los estimadores reciben estos valores en I/Q y realizan los cálculos explicados anteriormente. Las señales de salida de los estimadores se configuran como señales de salida de la IP. Para finalizar, solo queda generar el IP del diseño de System Generator para incluirla en el proyecto de Vivado. Al generar el IP para su uso en Vivado, System Generator permite seleccionar el lenguaje de descripción hardware, la tarjeta de procesamiento, etc.

3.2.2. Añadir la IP generada al proyecto de Vivado

Después de generar la IP con System Generator, es esencial actualizar la IP en el proyecto de Vivado y establecer las señales de salida del estimador de potencia como AXI GPIOs, para que puedan ser leídas por el procesador R5. El proyecto de Vivado consta de múltiples archivos y la arquitectura de las distintas partes del proyecto se puede observar en la siguiente figura.

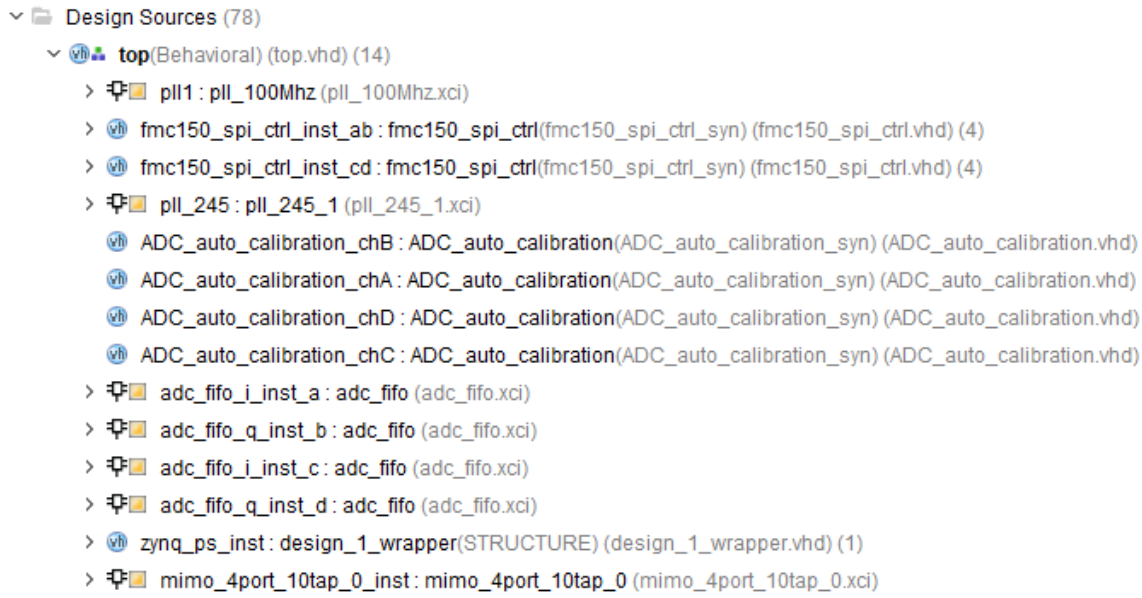


Figura 36: Arquitectura de los archivos del proyecto en Vivado.

El proyecto consta de diferentes partes como se observa en la figura 36; las IPs correspondientes a las PLLs que generan los relojes de 100 y 245 MHz del sistema, las IP de las FIFOs correspondientes a los ADC de cada canal y el "*wrapper vhdl*" correspondiente al subsistema PS donde se encuentran los bloques AXI GPIO para la aplicación de los coeficientes de emulación y el envío de los paquetes del Front-End mediante SPI. Por último, la IP correspondiente al diseño de System Generator donde se implementa el emulador de canal con el estimador de potencia recién añadido.

El primer paso es añadir la nueva IP generada al proyecto, para ello se clica en la pestaña IP Catalog. Dentro de esta pestaña se debe añadir el path al repositorio donde se encuentra la IP que se ha generado, y de esta manera añadir la IP al proyecto, como se muestra en la figura 37.

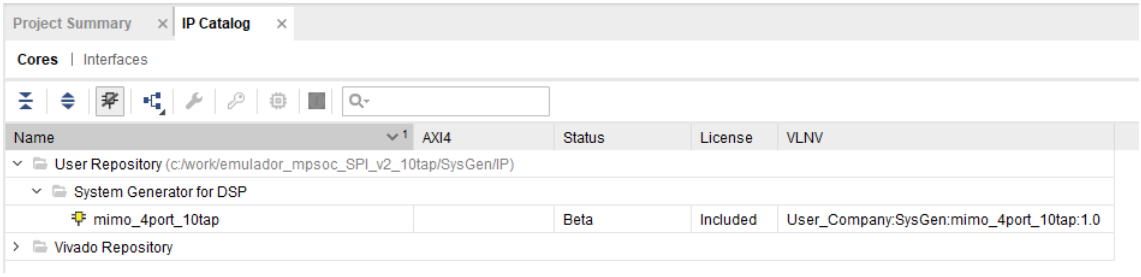


Figura 37: Captura del Ip Catalog de Vivado.

Para poder tratar las señales de potencia de los estimadores como AXI GPIOs, es necesario instanciar las IPs correspondientes en el diseño de bloques relacionado a la estructura del Processing System del proyecto, esta estructura se muestra en la figura 36 como "*zynq_ps_inst*". Para realizar esto, dentro del diseño de bloques de la estructura se añaden las IPs de AXI GPIO, como se muestra en la figura 38, para añadir las señales relacionados al estimador de potencia. Estos AXI GPIO se configuran como señales de entrada al diseño, menos las señales de *reset* y *threshold* que se configuran como señales de salida, puesto que son señales que se generan desde el Cortex R5. Después se añaden las señales del estimador como puertos al diseño de bloques y se conectan a los bloques

AXI GPIO. Para finalizar con el diseño, se configuran las conexiones de los bloques GPIO, se valida el diseño y se genera el *"wrapper VHDL"* para poder añadirlo al diseño *top* del proyecto.

Figura 38: IP correspondiente al AXI GPIO de la señal de potencia del canal A.

En este caso, se añade la IP en VHDL al código *top* del proyecto, se instancia y se realiza las conexiones pertinentes, además de las nuevas conexiones relacionadas al estimador de potencia entre la IP generada en System Generator y la estructura relacionada al PS de la Zynq Ultrascale+. La información compartida entre PS y el IP de System Generator se muestra en el diagrama de la figura 39.

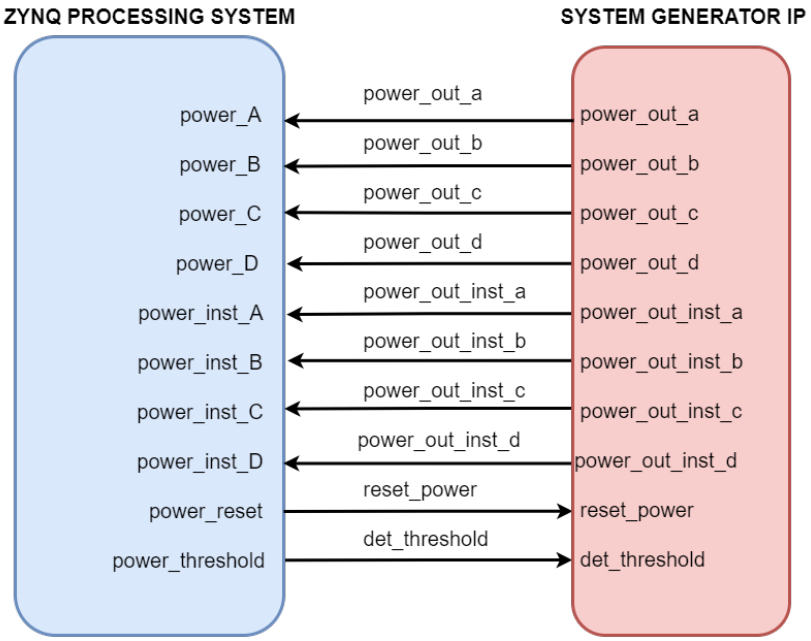
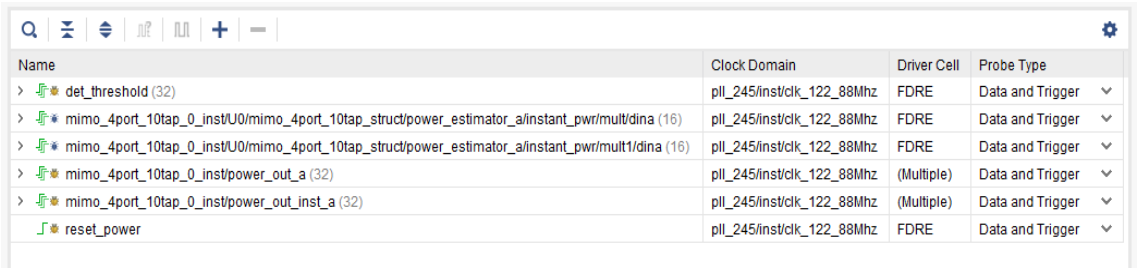


Figura 39: Diagrama de los datos compartidos entre el sistema procesador R5 y la IP de System Generator relacionados al estimador.

Para poder realizar las pruebas de funcionamiento del estimador de potencia, se ha utilizado la herramienta ILA (Integrated Logic Analyzer) que ofrece Vivado. El ILA es un núcleo de analizador lógico personalizable que puede utilizarse para monitorizar cualquier señal interna del diseño. Para incluirlo al diseño, se sintetiza el diseño actual y se abre la pestaña de *"Synthesized Design"* y se clica sobre *"Set Up Debug"*. Se despliega una pestaña como la que se observa en la figura 40, donde se pueden añadir las señales que se quieren capturar del diseño, y su correspondiente reloj. Una vez se han añadido las señales que se quieren debuggear, se clica sobre *APPLY*. De esta manera, se pueden

comprobar los valores de las señales capturadas mediante el ILA cuando se ejecuta el emulador.



Name	Clock Domain	Driver Cell	Probe Type
det_threshold (32)	pll_245/inst/clk_122_88Mhz	FDRE	Data and Trigger
mimo_4port_10tap_0_inst/U0/mimo_4port_10tap_struct/power_estimator_a/instant_pwr/mult/dina (16)	pll_245/inst/clk_122_88Mhz	FDRE	Data and Trigger
mimo_4port_10tap_0_inst/U0/mimo_4port_10tap_struct/power_estimator_a/instant_pwr/mult1/dina (16)	pll_245/inst/clk_122_88Mhz	FDRE	Data and Trigger
mimo_4port_10tap_0_inst/power_out_a (32)	pll_245/inst/clk_122_88Mhz	(Multiple)	Data and Trigger
mimo_4port_10tap_0_inst/power_out_inst_a (32)	pll_245/inst/clk_122_88Mhz	(Multiple)	Data and Trigger
reset_power	pll_245/inst/clk_122_88Mhz	FDRE	Data and Trigger

Figura 40: Señales elegidas para su captura mediante el analizador lógico.

Para finalizar, se implementa el diseño y se genera el bitstream. Por último, se exporta el hardware del diseño junto con el bitstream.

3.2.3. Preparar los archivos de la tarjeta SD

Como se ha modificado el diseño hardware del sistema, es necesario volver a generar los archivos *"BOOT.bin" e "image.ub"*. Para poder hacerlo, se utiliza el fichero de descripción de hardware que se ha exportado desde Vivado y se utiliza el siguiente comando en el proyecto del emulador en Petalinux:

```
$ petalinux-package --boot --fsbl "zynqmp_fsbl.elf" --fpga "top.bit"
   --u-boot --force
```

El comando que se utiliza es el mismo que el mostrado en el apartado 3.1.1, solo que esta vez, se añade el fichero bitstream puesto que se ha modificado el hardware. Tras esto, se añade el *–force* para poder ejecutar el comando y sobrescribir los ficheros "BOOT.bin" e "image.ub".

Para finalizar, se copian los archivos a la tarjeta SD, dentro de la partición conocida como BOOT.

4. Resultados

En este apartado se muestran los resultados experimentales obtenidos durante la realización del proyecto, donde se han utilizado los algoritmos y las funcionalidades desarrolladas en el apartado 3 para su obtención. Se han realizado diferentes pruebas para certificar el funcionamiento de las dos nuevas funcionalidades que se han añadido en el proyecto.

4.1. Resultados de la transmisión de un fichero con parámetros del canal

Para comprobar que la transmisión del fichero con los parámetros del canal se realiza correctamente, es necesario comprobar que se aplican los valores transmitidos en el fichero. Para ello, haciendo uso de MATLAB, se generan unas señales con una forma de onda conocida (triangular, cuadrada y sinusoidal) y se extraen sus valores I/Q para escribirlas en un fichero. Este fichero será leído por el procesador A53 y se enviará al procesador R5. Este ultimo, recogerá los valores del fichero y los aplicará mediante AXI al emulador de canal implementado en la FPGA. La señal entrante al emulador será un tono a 2.45 GHz, por lo que al ser multiplicado por los valores de los ficheros y debido a que la frecuencia de la aplicación de los valores es mucho menor, por la salida del emulador se debería visualizar la propia forma de onda de la señal del fichero.

4.1.1. Forma de onda cuadrada

En la primera prueba, se ha generado una señal cuadrada en MATLAB y se han extraído sus valores al fichero "files_prueba.txt". El valor mínimo de la señal es 0 y el valor máximo 6. A continuación se muestra el script que genera los parámetros del fichero en MATLAB:

```
% Frecuencia de la señal cuadrada
freq = 500;

% Amplitud de la señal cuadrada
amplitude = 6;

% Ancho de pulso de la señal cuadrada
duty_cycle = 0.5;

% Generar valores de tiempo
t = linspace(0, 1, 6400);
```

```
% Generar señal cuadrada en formato I/Q
period = 1/freq;
sq_signal = amplitude/2 * (square(2*pi*freq*t + pi*duty_cycle,
   duty_cycle*100) + 1i*square(2*pi*freq*t, (1-duty_cycle)*100))
    + amplitude/2;

% Obtener valores reales e imaginarios de la señal
real_part = real(sq_signal);
imag_part = imag(sq_signal);

% Abrir el archivo de texto para escritura
fileID = fopen('file_prueba.txt', 'w');

% Escribir los valores reales en la primera fila del archivo de
   texto
fprintf(fileID, '%f ', real_part);
fprintf(fileID, '\n');

% Escribir los valores imaginarios en la segunda fila del
   archivo de texto
fprintf(fileID, '%f ', real_part);

% Cerrar el archivo de texto
fclose(fileID);
```

En este caso se ha enviado un fichero con 6400 valores debido a que es un tamaño donde la transmisión del fichero entre ambos procesadores se realiza rápidamente. Los valores se escriben en el formato en el que el procesador A53 los lee, como se muestra en el apartado 3.1.4.2. Una vez se ha creado el fichero se copia en el sistema de archivos del Linux del A53, para que la aplicación pueda leer los valores. Se arranca el emulador y se configura para la transmisión del fichero. Cuando se transmiten todos los valores del fichero mediante openAMP al procesador R5, esté aplica los valores mediante AXI en el emulador de canal implementado en PL. A continuación, en la figura 41, se muestra la captura del osciloscopio de la señal de salida del emulador.

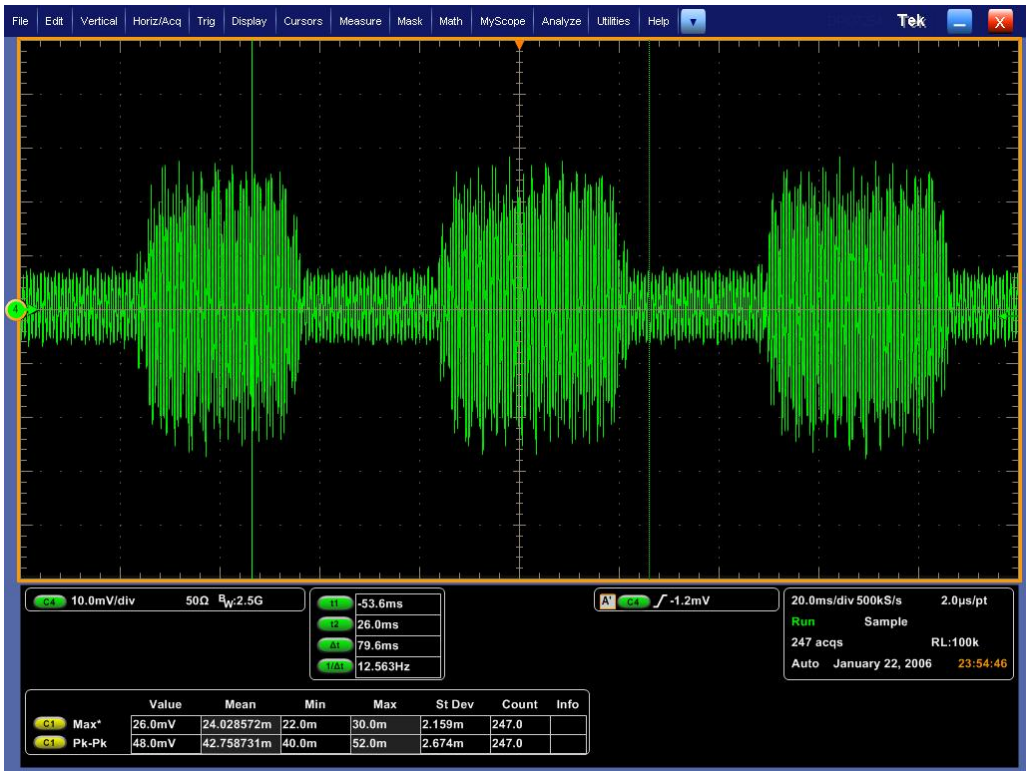


Figura 41: Señal de salida del emulador capturada en el osciloscopio con el fichero con parámetros de una señal cuadrada.

Como se observa en la figura, la señal resultante de la aplicación de los parámetros del fichero con el tono introducido da lugar a una señal con forma de onda cuadrada.

4.1.2. Forma de onda triangular

Al igual que en el apartado anterior, se genera una señal mediante MATLAB con una forma de diente de sierra y se escriben los valores I/Q en el fichero "files_pruebas". A continuación se muestra el script utilizado para generar los parámetros del fichero para una señal de diente de sierra:

```
% Frecuencia de la señal de diente de sierra
freq = 500;

% Fase de la señal de diente de sierra
phase = pi/4;

% Amplitud de la señal de diente de sierra
amplitude = 3.5; % para que el rango sea 0 a 7

% Generar valores de tiempo
t = linspace(0, 1, 6400);

% Generar señal de diente de sierra en formato I/Q con amplitud
   3.5 y rango entre 0 y 7
iq_signal = amplitude * sawtooth(2*pi*freq*t + phase, 0.5) + 1i*
   amplitude*0.5 + 3.5;
```

```
% Obtener valores reales e imaginarios de la señal
real_part = real(iq_signal);
imag_part = imag(iq_signal);

% Abrir el archivo de texto para escritura
fileID = fopen('file_prueba.txt', 'w');

% Escribir los valores reales en la primera fila del archivo de
    texto
fprintf(fileID, '%f ', real_part);
fprintf(fileID, '\n');

% Escribir los valores imaginarios en la segunda fila del
    archivo de texto
fprintf(fileID, '%f ', imag_part);

% Cerrar el archivo de texto
fclose(fileID);
```

Como en el caso anterior, se utiliza un fichero con 6400 valores. Los valores de la forma de onda van de 0 a 7. En la siguiente figura se muestra la captura del osciloscopio de la señal de salida del emulador para la emulación de este fichero.

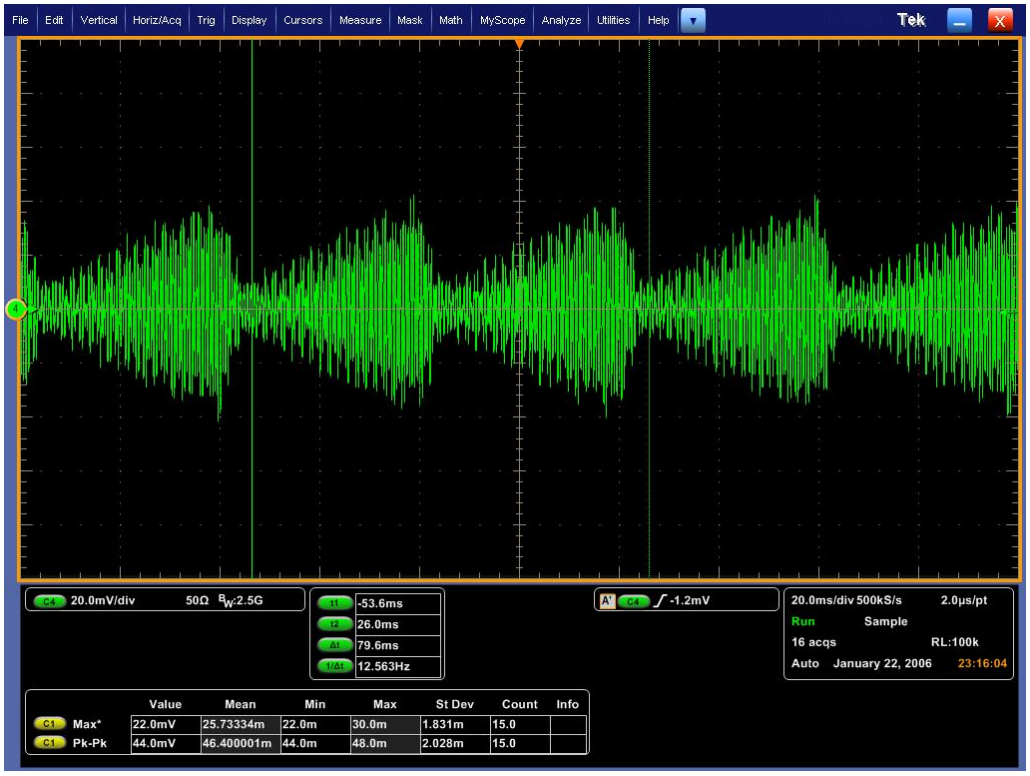


Figura 42: Señal de salida del emulador capturada en el osciloscopio con el fichero con parámetros de una señal de diente de sierra.

Como se observa en la figura 42, la señal resultante de la aplicación de los parámetros del fichero con el tono introducido da lugar a una señal con forma de onda triangular.

4.1.3. Forma de onda sinusoidal

En esta última prueba, se ha generado una señal con una forma de onda sinusoidal y se escriben los valores de la señal I/Q en el fichero "files_pruebas". A continuación se muestra el script utilizado para generar los parámetros del fichero para una señal sinusoidal:

```
% Frecuencia de la señal senoidal
freq = 500;

% Fase de la señal senoidal
phase = pi/4;

% Amplitud de la señal senoidal
amplitude = 3;

% Generar valores de tiempo
t = linspace(0, 1, 6400);

% Generar señal senoidal en formato I/Q con amplitud 3 y rango
   entre 0 y 6
iq_signal = amplitude * sin(2*pi*freq*t + phase) + 1i*amplitude*
   cos(2*pi*freq*t + phase) + 3;

% Obtener valores reales e imaginarios de la señal
real_part = real(iq_signal);
imag_part = imag(iq_signal);

% Abrir el archivo de texto para escritura
fileID = fopen('file_prueba.txt', 'w');

% Escribir los valores reales en la primera fila del archivo de
   texto
fprintf(fileID, '%f ', real_part);
fprintf(fileID, '\n');

% Escribir los valores imaginarios en la segunda fila del
   archivo de texto
fprintf(fileID, '%f ', imag_part);

% Cerrar el archivo de texto
fclose(fileID);
```

En esta prueba también se utiliza un fichero con 6400 valores. La forma de onda sinusoidal tiene valores entre 0 y 6. En la siguiente figura se muestra la captura del osciloscopio de la señal de salida del emulador para la emulación de este fichero.

Figura 43: Señal de salida del emulador capturada en el osciloscopio con el fichero con parámetros de una señal sinusoidal.

Como se observa en la figura 43, la señal resultante de la aplicación de los parámetros del fichero con el tono introducido da lugar a una señal con forma de onda sinusoidal.

Después de las tres pruebas, se puede certificar que los parámetros se transmiten de forma correcta entre los procesadores Cortex A53 y R5. Y que este último, aplica los valores correctamente en el emulador de canal implementado en la FPGA.

4.2. Resultados del estimador de potencia

4.2.1. Simulaciones en Simulink

En primer lugar, se ha evaluado el funcionamiento del estimador de potencia mediante simulaciones en Simulink. Para ello, se ha creado un script de MATLAB que genere una señal de entrada al emulador controlando los parámetros de la frecuencia y la potencia de la señal, junto con la relación señal-ruido. La señal generada se modula a la frecuencia intermedia utilizada en el diseño (61.5 MHz). El código del script utilizado para la simulación se muestra a continuación:

```
Fs = 245.76e6;

f=20e6;
power_dbm = -25; %max 4 dBm -> 1 Vpp %min -80 dBm (sensibilidad
   entrada) -40 dBm (sensibilidad estimador potencia ->
   threshold -20)
```

```
to=(0:(Fs/1000)-1)/Fs;

power_W= (10^(power_dbm/10))*1e-3;
Vpico=sqrt(power_W*50)*sqrt(2);
Vpico_adc=Vpico*2; %El ADC tiene una ganancia de 6 dB en tensión

dur = 1;
x = chirp(to, f, dur, f, 'quadratic'); % Señal chirp lineal
snr = 60; % Nivel de relación señal-ruido deseado
y = awgn(x, snr); % Señal real con ruido gaussiano

signal1=Vpico_adc.*y;
signal=ssbmod(signal1,Fs/4,Fs,0,'upper');

rf_bp_a = signal;        % Generar señal
rf_bp_b = signal;
```

A raíz de los valores de la potencia de la señal de entrada y su frecuencia, así como su relación SNR, se genera la señal de entrada de los canales A y B. Para poder comprobar el funcionamiento del estimador se comprueban mediante un Scope las señales de salida del estimador *power* y *power_inst*. En la figuras 44 y 45, se muestran las señales de salida del estimador en Watts y en dBm, para una señal de entrada de 0 dBm y 20 MHz.

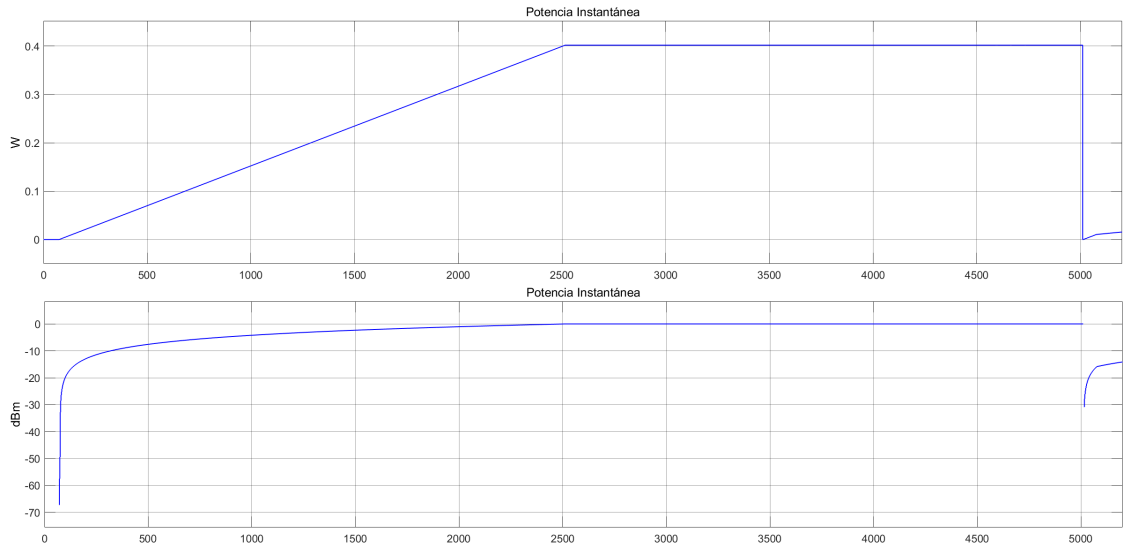


Figura 44: Potencia instantánea del estimador en Watts y dBm en Simulink con una señal de entrada de 0 dBm y 20 MHz

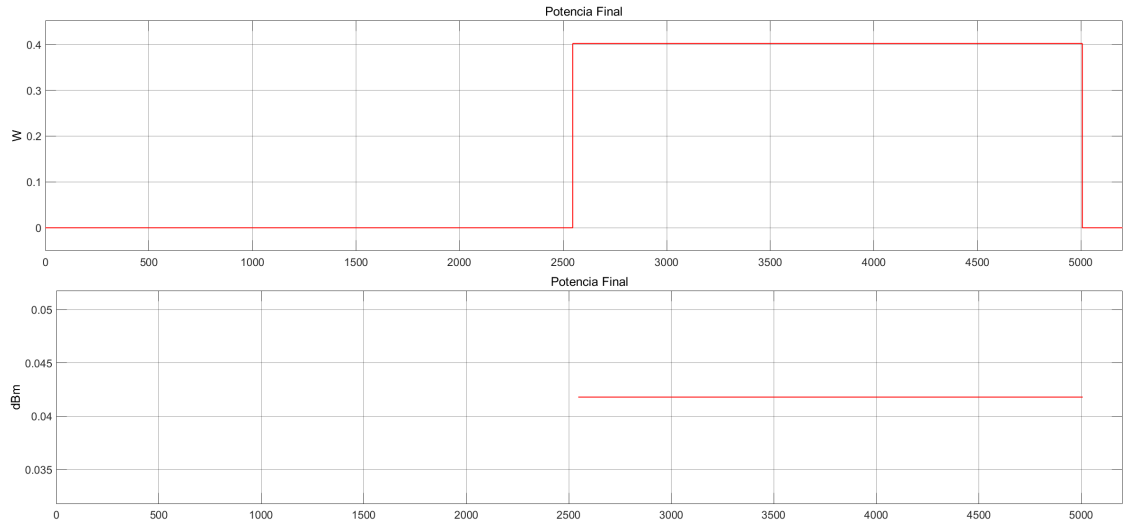


Figura 45: Potencia final del estimador en Watts y dBm en Simulink con una señal de entrada de 0dBm y 20 MHz.

Como se observa en las gráficas la potencia instantánea necesita un margen para estabilizarse, concretamente el número de muestras elegido en System Generator, que en este caso es $N = 1220$. Este valor se elige para cubrir las señales a partir de 1 kHz.

Además cuando la potencia instantánea supera el umbral establecido de potencia mínima, fijado por la señal de *"threshold"*, se activa la cuenta de muestras para establecer la potencia final de la señal, después de un margen de $N = 1220$ muestras, al igual que para la potencia instantánea. Tras esperar el numero de muestras definido, se iguala el valor de la potencia instantánea a la potencia final del estimador, ya que se considera que el valor actual corresponde a una media de valores que ha sido calculada después de detectar una señal por encima del umbral de 20 dBm. Una vez se ha calculado el valor de la señal, el valor de la potencia total no se actualiza hasta recibir la señal de *reset*, para reiniciar el proceso de estimación de nuevo.

Para comparar el funcionamiento del estimador con otras potencias, a continuación, se muestran dos figura de las gráficas para una señal de entrada de -10 dBm y 20 MHz.

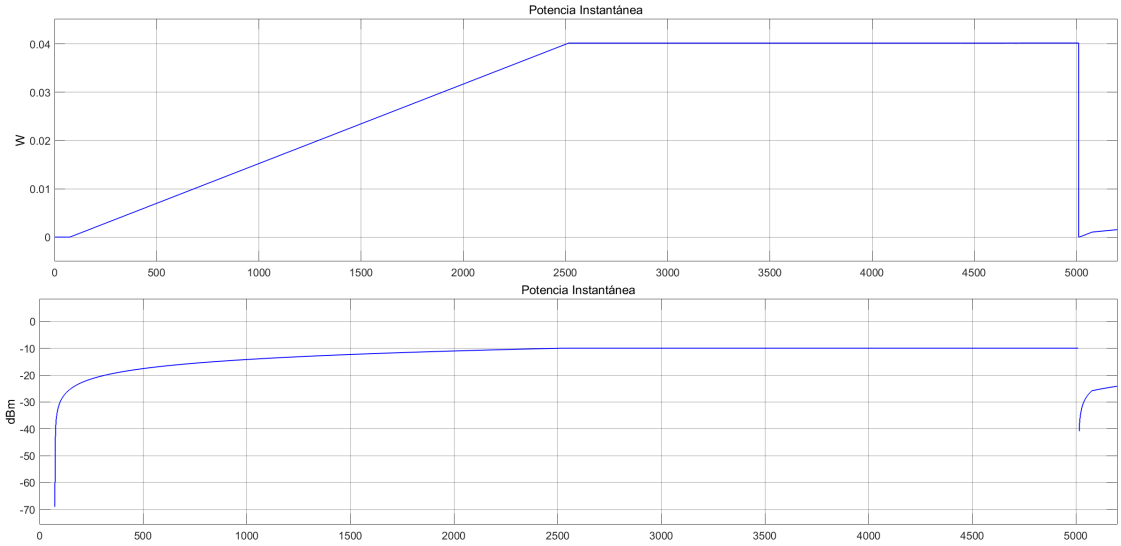


Figura 46: Potencia instantánea del estimador en Watts y dBm en Simulink con una señal de entrada de -10 dBm y 20 MHz

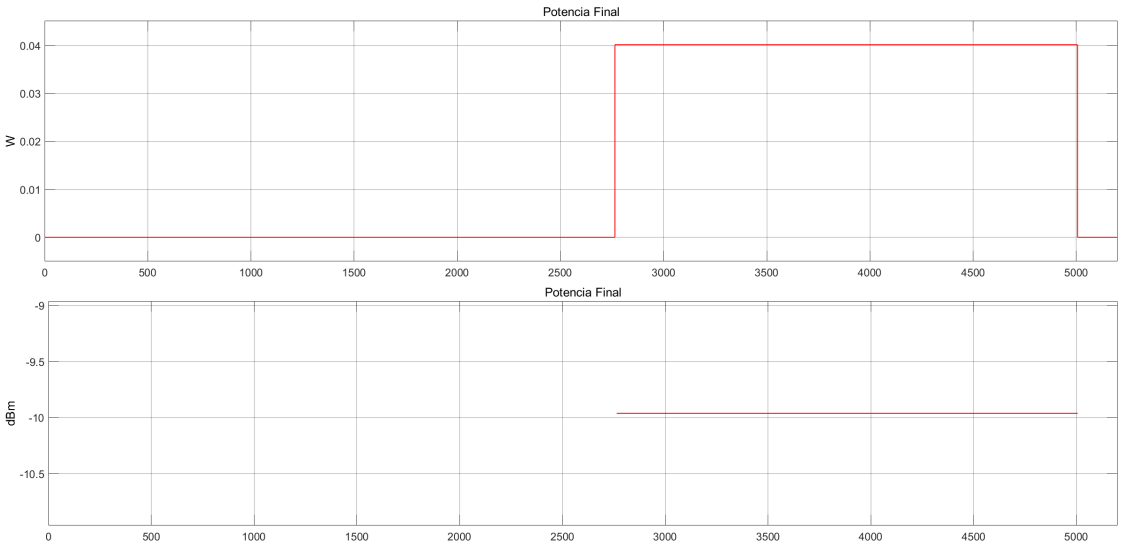


Figura 47: Potencia final del estimador en Watts y dBm en Simulink con una señal de entrada de -10dBm y 20 MHz.

Si se comparan las figuras de la señal de 0dBm y -10 dBm, se observa cómo para la señal de -10 dBm de potencia, el calculo de la potencia final (señal *power*) se retrasa más que para el valor de 0dBm. Esto ocurre debido a que para una señal de menor potencia, de -10 dBm, le cuesta más tiempo superar el umbral de potencia mínima al valor de potencia instantánea media como para tener en cuenta la señal. Por lo tanto, durante ese tiempo no se activa la cuenta de muestras para establecer la señal de potencia final.

En ambos casos, las medidas realizadas de la potencia de ambas señales son correctas, por lo que se da por bueno el diseño del estimador.

4.2.2. Capturas del Analizador Lógico

El segundo paso para confirmar el correcto funcionamiento del estimador ha sido capturar las señales del estimador mediante el analizador lógico. El analizador lógico utilizado está integrado dentro del propio diseño de Vivado, como se muestra en el apartado 3.2.2.

Para poder capturar las señales se arranca el emulador y se configura un canal, con un path y los parámetros que se deseen (no es relevante puesto que no afecta al cálculo del estimador). Una vez se ha configurado el canal, se clica en *RUN* para comenzar la emulación.

Después, se abre el Hardware Manager en Vivado y se conecta a la tarjeta de procesamiento mediante JTAG clicando en *Auto-Connect*. Una vez conectada a la tarjeta, se muestra en pantalla el monitor donde se añaden las señales capturadas que se quieren visualizar.

En la figura 48, se muestra una captura del analizador lógico para una señal de entrada 1 MHz modulada a 2.45 GHz, con una potencia de 0dBm.

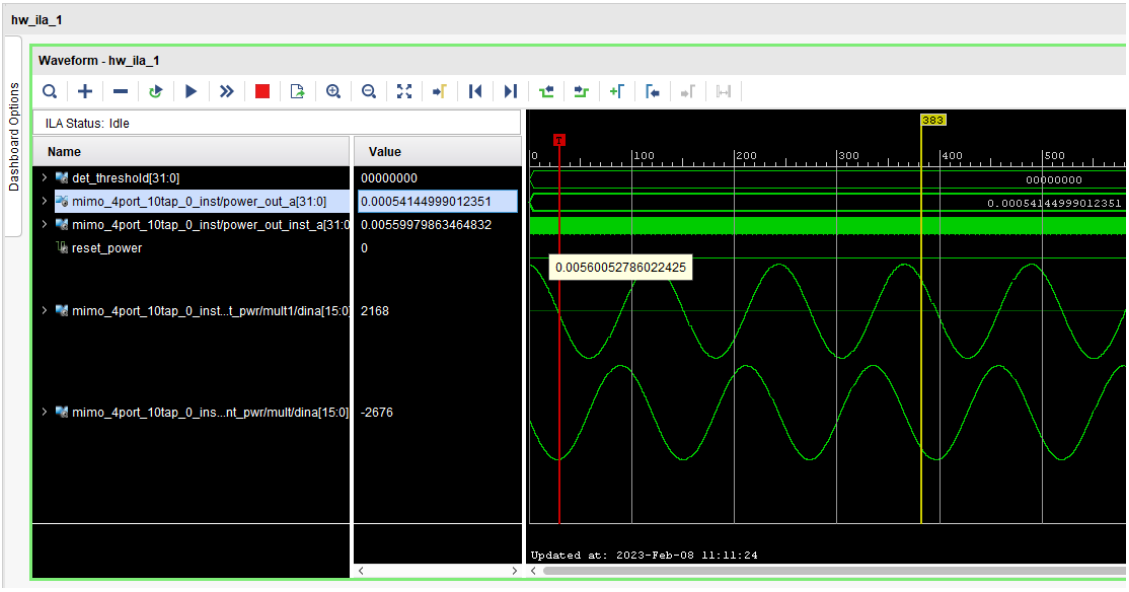


Figura 48: Captura de las señales del analizador lógico para una señal de 1MHz y una potencia de 0 dBm.

Antes de analizar las señales obtenidas mediante el analizador lógico, hay que destacar un par de cosas. Por un lado, como se observa en la figura 48, la señal de potencia final no corresponde con un valor correcto. Como también se observa en la figura, el *threshold* elegido tiene un valor de 0, puesto que todavía no se ha implementado el código en la aplicación del Cortex R5 que fije su valor. Por lo que la señal de potencia tiene en cuenta muestras donde el valor de la señal de potencia todavía no ha pasado un nivel aceptable de umbral. Además, no se ha implementado el código que genera la señal de *reset* para los estimadores en estas pruebas, por lo que, una vez establecido el valor, no se vuelve a actualizar.

Por otro lado, la cadena del emulador añade una importante perdida de potencia debido al recorrido de la señal desde el generador de señales hasta la FPGA. Esta perdida de potencia se calcula entorno a los 15 dBm para una señal modulada a 2.45 GHz.

A raíz de estos factores, se tiene en cuenta el valor de la potencia instantánea para valorar el funcionamiento del estimador. El valor se considera estable una vez se ha esperado el tiempo suficiente para superar el margen donde el valor de la señal no es estable.

Observando la figura, se puede extraer el valor de la potencia instantánea de 0.0056W, que convertidos a dBm resultan en una potencia de -18.518 dBm. Si se tiene en cuenta la etapa de perdida de potencia del emulador, se considera que el valor obtenido de la medición es aceptable y que el estimador funciona correctamente.

4.2.3. Comparación entre las simulaciones y las capturas del analizador lógico

En los dos anteriores apartados, se ha mostrado cómo se han obtenido los resultados de los valores de la potencia tanto en las simulaciones de Simulink como en el analizador lógico integrado. En este apartado, se va a realizar una comparativa de todas las mediciones que se han realizado para señales de entrada con diferentes frecuencias y potencias. Se van a comparar los valores calculados entre la simulación en Simulink y los valores capturados por el analizador lógico.

En la tabla 2, se muestra la comparativa de una señal con una frecuencia de 1 MHz, con valores de potencia de 10 dBm a -10dBm, con un salto de 5dBm entre medición.

Potencia Entrada	ILA	ILA	SYSGEN	SYSGEN	Diferencia (dBm)
dBm	WATT	dBm	WATT	dBm	dBm
10 dBm	0,054300	-8,652	0,1271	-4,959	3,693
5dBm	0,018900	-13,235	0,0402	-9,958	3,278
0 dBm	0,005770	-18,388	0,0127	-14,962	3,426
-5dBm	0,001820	-23,399	0,00399	-19,990	3,409
-10 dBm	0,000554	-28,565	0,001243	-25,055	3,510

Tabla 2: Medidas de las simulaciones y el analizador lógico para una señal de entrada de 1MHz.

En la tabla se puede observar que los valores obtenidos entre la simulación y el analizador lógico difieren en torno a los 3.5 dBm en todos los casos. Esta diferencia de potencia se puede atribuir a las pérdidas de potencia del camino que recorre la señal que no se hayan tenido en cuenta, por ejemplo, las pérdidas introducidas por los cables. A pesar de la diferencia, los resultados obtenidos se pueden dar como válidos.

En la tabla 3, se muestra la comparativa de una señal con una frecuencia de 10 MHz, con valores de potencia de 10 dBm a -10dBm.

Potencia Entrada	ILA	ILA	SYSGEN	SYSGEN	Diferencia (dBm)
dBm	WATT	dBm	WATT	dBm	dBm
10 dBm	0,041990	-9,769	0,1271	-4,959	4,810
5dBm	0,013680	-14,639	0,04017	-9,961	4,678
0 dBm	0,004300	-19,665	0,01268	-14,969	4,697
-5dBm	0,001350	-24,697	0,003995	-19,985	4,712
-10 dBm	0,000417	-29,799	0,001242	-25,059	4,740

Tabla 3: Medidas de las simulaciones y el analizador lógico para una señal de entrada de 10MHz.

En este segundo caso se observa como la diferencia entre la potencia calculada en la simulación y el analizador ha aumentado en un 1dBm respecto a las señales mostradas en la anterior tabla.

En la tabla 4, se muestra la comparativa de una señal con una frecuencia de 20 MHz, con valores de potencia de 10 dBm a -10dBm.

Potencia Entrada	ILA	ILA	SYSGEN	SYSGEN	Diferencia (dBm)
dBm	WATT	dBm	WATT	dBm	dBm
10 dBm	0,035775	-10,464	0,12709	-4,959	5,505
5dBm	0,011538	-15,379	0,040175	-9,960	5,418
0 dBm	0,003659	-20,366	0,012682	-14,968	5,398
-5dBm	0,001134	-25,454	0,003993	-19,987	5,467
-10 dBm	0,000349	-30,572	0,0012425	-25,057	5,515

Tabla 4: Medidas de las simulaciones y el analizador lógico para una señal de entrada de 20MHz.

Al igual que en la anterior tabla, al subir la frecuencia de la señal a 20 MHz, la diferencia entre el valor obtenido de la simulación y del analizador se aumenta en 1 respecto a la anterior tabla, llegando a una diferencia de 5,5 dBm.

En la siguiente tabla 5, se muestra la comparativa de una señal con una frecuencia de 30 MHz, con valores de potencia de 10 dBm a -10dBm.

Potencia Entrada	ILA	ILA	SYSGEN	SYSGEN	Diferencia (dBm)
dBm	WATT	dBm	WATT	dBm	dBm
10 dBm	0,031560	-11,009	0,12708	-4,959	6,049
5dBm	0,010121	-15,948	0,04016	-9,962	5,986
0 dBm	0,003198	-20,952	0,01268	-14,969	5,983
-5dBm	0,001003	-25,988	0,003991	-19,989	5,999
-10 dBm	0,000306	-31,143	0,0012422	-25,058	6,085

Tabla 5: Medidas de las simulaciones y el analizador lógico para una señal de entrada de 30MHz.

En la siguiente tabla 6, se muestra la comparativa de una señal con una frecuencia de 40 MHz, con valores de potencia de 10 dBm a -10dBm.

Potencia Entrada	ILA	ILA	SYSGEN	SYSGEN	Diferencia (dBm)
dBm	WATT	dBm	WATT	dBm	dBm
10 dBm	0,026847	-11,711	0,1269	-4,965	6,746
5dBm	0,008512	-16,700	0,04009	-9,970	6,730
0 dBm	0,002674	-21,729	0,01266	-14,976	6,753
-5dBm	0,000834	-26,788	0,003989	-19,991	6,797
-10 dBm	0,000254	-31,949	0,001241	-25,062	6,886

Tabla 6: Medidas de las simulaciones y el analizador lógico para una señal de entrada de 40MHz.

En la siguiente tabla 7, se muestra la comparativa de una señal con una frecuencia de 50 MHz, con valores de potencia de 10 dBm a -10dBm.

Potencia Entrada dBm	ILA WATT	ILA dBm	SYSGEN WATT	SYSGEN dBm	Diferencia (dBm) dBm
10 dBm	0,021148	-12,747	0,1268	-4,969	7,778
5dBm	0,006663	-17,764	0,04007	-9,972	7,792
0 dBm	0,002091	-22,797	0,01265	-14,979	7,818
-5dBm	0,000647	-27,890	0,003983	-19,998	7,892
-10 dBm	0,000193	-33,138	0,00124	-25,066	8,073

Tabla 7: Medidas de las simulaciones y el analizador lógico para una señal de entrada de 50MHz.

Comparando los valores de cada tabla se observa que la diferencia de potencia entre la simulación y el analizado lógico aumenta en un 1dBm cada vez que se incrementa en 10MHZ la frecuencia de la señal de entrada. Los valores que hacen que la diferencia se incremente son los del ILA, que se obtienen valores de potencia cada vez más bajos, mientras que, los valores de las simulaciones se mantienen muy parecidas entre si a pesar de la diferencia de frecuencias entre unas tablas y otras.

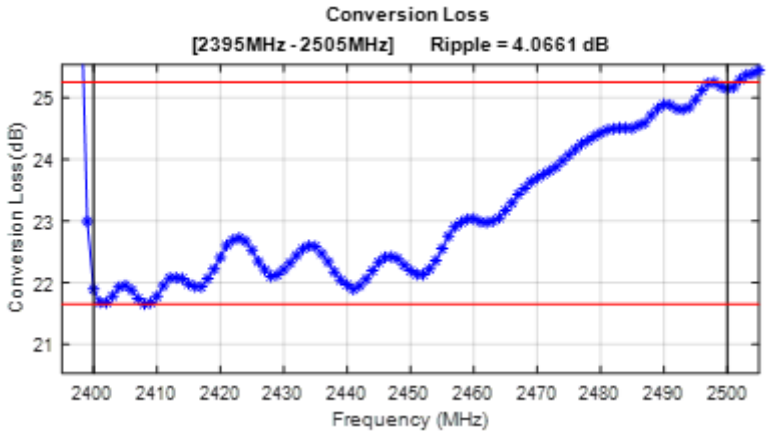


Figura 49: Caracterización de las pérdidas de conversión del Front-End para el rango de frecuencias de 2.4 a 2.5 GHz.

Esta disminución del nivel de potencia junto con el aumento de la frecuencia se puede deber a las pérdidas que introduce el Front-End en la Downconversion de la señal. La señales que se han introducido son tonos a *2.45 GHz + f_{signal} (1MHz, 10MHz, 20 MHz,...)*. En la figura 49, se muestra una gráfica de unas pruebas que se hicieron sobre la caracterización del Front-End en un trabajo propio dentro del grupo de investigación, donde se muestra las pérdidas de potencia de la conversión en dB, respecto a la frecuencia de la señal introducida. Se puede observar cómo a partir de los 2450 MHz, la señal se atenúa entorno a 1 dB más por cada incremento de 10 MHz en la señal de entrada.

Debido a esto, es comprensible que los resultados que se han observado para diferentes frecuencias en las capturas de analizador lógico difieran entre si. Y como es lógico, el incremento entre la diferencia de pérdidas en relación a la frecuencia observada en el analizador y la observada en la gráfica sobre la caracterización del Front-End son muy parecidas. Por lo que se entiende que en este caso, las diferencias entre las mediciones están originadas en el Front-End.

5. Metodología

En este apartado se explica cómo se ha organizado el proyecto en diferentes tareas y fases. Además se explica cual ha sido el equipamiento y las herramientas utilizadas para el desarrollo del proyecto.

5.1. Descripción de tareas y fases

En la primera fase del proyecto se han definido los objetivos de este y se ha realizado el plan de trabajo para alcanzar esos objetivos.

- **Fase 1: Definición del proyecto**
  - **Tarea 1.1**: Definición de objetivos.
  - **Tarea 1.2**: Definición de especificaciones
  - **Tarea 1.3**: Definición de plan de trabajo.

En la segunda y tercera fases del proyecto se ha realizado el análisis del estado del arte de las tecnologías relacionadas al proyecto y se ha estudiado el trabajo previo del emulador de canal de Ikerlan, para entender su arquitectura y funcionamiento.

- **Fase 2: Análisis del estado del arte**
  - **Tarea 2.1**: Estudio de la familia MPSoC Zynq Ultrascale+.
  - **Tarea 2.2**: Estudio del multiprocesamiento en sistemas embebidos.
  - **Tarea 2.3**: Estudio de los emuladores de canal y comunicaciones inalámbricas.
- **Fase 3: Análisis del emulador de Ikerlan**
  - **Tarea 3.1**: Análisis de los entornos procesadores Cortex A53 y R5.
  - **Tarea 3.2**: Análisis de los códigos de las aplicaciones de los procesadores Cortex A53 y R5.
  - **Tarea 3.3**: Análisis del emulador de canal implementado en la FPGA y su diseño en System Generator.

La fase 4 del Proyecto corresponde al desarrollo de este, la fase más larga e importante del proyecto. Se detallan los diferentes pasos que se han seguido para conseguir completar las nuevas funcionalidades del emulador.

- **Fase 4: Desarrollo de las funcionalidades**
    - **Tarea 4.1**: Reproducción de un array de manera local en el procesador R5.
    - **Tarea 4.2**: Lectura de un fichero con parámetros en formato I/Q en el procesador A53.
    - **Tarea 4.3**: Transmisión de los parámetros del fichero desde el procesador A53 al procesador R5.
    - **Tarea 4.4**: Diseño del estimador de potencia en System Generator.
    - **Tarea 4.5**: Implementación del estimador en el diseño del emulador de canal en Vivado.
    - **Tarea 4.6**: Lectura de los valores de potencia del estimador y calibración automática del Front-End mediante el Cortex R5.

La siguiente fase del proyecto corresponde al análisis de los resultados obtenidos de las pruebas de validación de la transmisión del fichero con parámetros del canal, así como de las simulaciones y capturas del analizador lógico del estimador de potencia.

- **Fase 5: Análisis de los resultados**
    - **Tarea 5.1**: Medición de las señales de salida transmitiendo ficheros con parámetros diferentes.
    - **Tarea 5.2**: Simulación de las mediciones del estimador de potencia.
    - **Tarea 5.3**: Medición de la señales del estimador de potencia mediante el analizador lógico integrado.
    - **Tarea 5.4**: Comparación de los valores de las señales de simulación y las capturadas mediante el analizado lógico.

En la última fase del proyecto se ha realizado un análisis del trabajo realizado hasta el final del proyecto y se han definido las posibles mejoras que se pueden añadir al sistema en un futuro.

- **Fase 6: Finalización del proyecto**
    - **Tarea 5.1**: Conclusiones.
    - **Tarea 5.2**: Líneas futuras.

5.2. Herramientas de Desarrollo

Para el desarrollo del proyecto, han sido necesarias las siguientes herramientas de desarrollo.

5.2.1. Vivado

Vivado es un conjunto de herramientas de diseño de sistemas electrónicos desarrollada por Xilinx. Esta herramienta se utiliza para diseñar, implementar y depurar sistemas digitales basados en lógica programable.

Figura 50: Logo de Vivado[6].

En este trabajo se ha utilizado para el diseño del emulador de canal implementado en la lógica programable de la Zynq Ultrascale+. Además, mediante el *Hardware Manager* que proporciona, se han visualizado las señales capturadas mediante el analizador lógico integrado.

5.2.2. Xilinx Software Development Kit

Xilinx SDK es un entorno de desarrollo integrado en Vivado. Xilinx SDK proporciona un conjunto completo de herramientas para el desarrollo de software, incluyendo un editor de código, un depurador, un compilador de C/C++, un analizador de rendimiento y una amplia variedad de bibliotecas de software. También permite la integración con otras herramientas de terceros y ofrece soporte para múltiples sistemas operativos, incluyendo Linux, FreeRTOS y otros sistemas operativos en tiempo real.

5.2.3. MATLAB-Simulink

MATLAB-Simulink es una plataforma de software desarrollada por MathWorks que se utiliza para modelar, simular y analizar sistemas dinámicos y procesamiento de señales. MATLAB es un lenguaje de programación numérica utilizado para análisis de datos, visualización y cálculo matemático, mientras que Simulink es una herramienta de diseño de sistemas basados en modelos que permite a los usuarios crear y simular modelos de sistemas en tiempo real.

Figura 51: Logo de MathWorks[7].

En este proyecto se ha utilizado esta herramienta para la creación de ficheros con parámetros del emulador de canal.

5.2.4. System-Generator

Xilinx System Generator es una herramienta que utiliza Simulink para generar algoritmos de procesamiento digital de señal de manera rápida y cómoda. Es una herramienta de diseño de sistemas digitales útil para los diseñadores que trabajan con FPGAs y SoCs de Xilinx, ya que permite la validación de diseños antes de implementarlos en hardware.

Figura 52: Logo de System Generator [8].

La herramienta cuenta con una amplia variedad de bloques en librerías destinados a diferentes funciones, como procesamiento de señales, lo que resulta muy útil para el diseño de sistemas digitales. Estos bloques permiten la generación de modelos destinados a realizar la parte lógica de cada proyecto, los cuales pueden ser simulados en Simulink. Esto facilita la validación y prueba previa de los resultados esperados, lo que ayuda a garantizar el correcto funcionamiento del sistema digital.

En este trabajo se utiliza para el diseño del estimador de potencia y el emulador de canal.

5.2.5. Qt Creator

Qt Creator es un entorno de desarrollo integrado (IDE) para programación en C++ y QML, utilizado principalmente para desarrollar aplicaciones con la biblioteca de herramientas de desarrollo de software Qt. Esta herramienta proporciona un editor de código completo, depurador, herramientas de compilación y herramientas de diseño visual para ayudar a los desarrolladores a crear aplicaciones de alta calidad y rendimiento.

Figura 53: Logo de Qt Creator[9].

En este proyecto se ha utilizado para realizar la aplicación de la interfaz gráfica del emulador en el procesador Cortex A53.

5.3. Equipamiento

A continuación se explican de forma breve las características técnicas de los diferentes equipos utilizados a lo largo del proyecto.

5.3.1. Tarjeta de evaluación ZCU102

La ZCU102 es una placa de evaluación de propósito general para la creación rápida de prototipos basada en el Zynq UltraScale+ MPSoC. Las interfaces de memoria DDR4

SODIMM de alta velocidad y de componentes, los puertos de expansión FMC, los transceptores serie multigigabit por segundo, una variedad de interfaces periféricas y la lógica FPGA para diseños personalizados por el usuario proporcionan una plataforma de prototipado flexible[10].

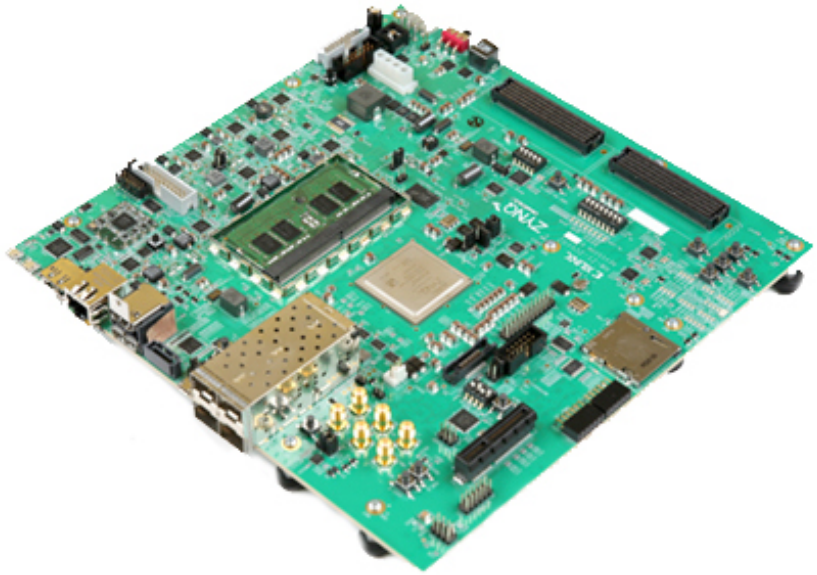


Figura 54: Vista superior de la tarjeta de evaluación ZCU102[10].

A continuación se presentan las características más importantes de la tarjeta:

- Procesador Cortex R5 dual-core y procesador Cortex A53 quad-core
- Conector USB-JTAG incorporado para programación
- Memoria flash Dual Quad-SPI
- Arranque desde tarjeta SD
- Memoria
    - PS 4GB DDR4 SODIMM de 64 bits
    - Memoria DDR4 PL de 512MB
    - EEPROM de 8KB
    - Dual flash Quad SPI de 64MB
    - Ranura para tarjeta SD
- Control y E/S
    - 6x Botones direccionales (5x PL, 1x PS)
    - Interruptores

    - PMBUS y controlador de sistema MSP430 para alimentación, relojes y conmutación de bus I2C
    - USB2/3 (MIO ULPI y 1 GTR)
- Comunicación y Redes
    - Puente UART a USB
    - Conector RJ45 Ethernet
    - SATA
    - Puerto PCIe
- Pantalla
    - Entrada y salida de video HDMI
    - 9x LEDs de usuario GPIO (8x PL, 1x PS)
- Reloj
    - Relojes programables
    - Relojes de sistema, de usuario, de atenuación de jitter
- Alimentación
    - Adaptador de 12V

5.3.2. FMC150

La tarjeta FMC150, la cual se observa en la figura 55, está equipada con dos canales de entrada ADC de 14 bits y dos canales de salida DAC de 16 bits que pueden ser sincronizados por un reloj tanto interno como externo. Cuenta con un Conector LPC (Low Pin Count) para facilitar el manejo de la tarjeta. Su diseño está basado en el convertidor analógico-digital ADS62P49 de doble canal de 14 bits a 250 Msps, el convertidor analógico-digital DAC3283 de doble canal de 16 bits a 800 Msps, y el circuito integrado CDCE72010, todos de Texas Instrument[11].

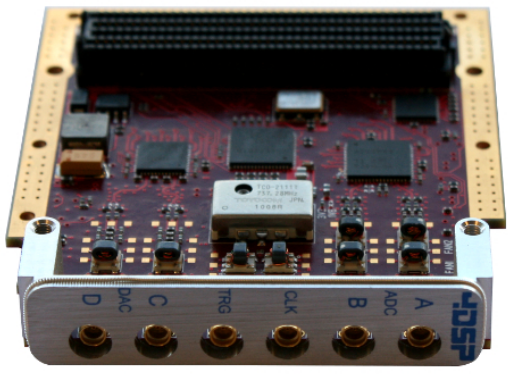


Figura 55: Tarjeta FMC150[11].

La tarjeta FMC150 se utiliza para realizar la conversión analógico-digital y digital-analógica de las señales de entrada y salida del emulador.

5.3.3. EFM32 Giant Gecko

El EFM32GG es un microcontrolador con 1MB flash y 128KB RAM, lo que lo hace ideal para aplicaciones sensibles a la energía con altos requisitos de memoria y conectividad. El Giant Gecko incorpora un depurador SEGGER J-Link y un avanzado sistema de monitorización de energía, que permite programar, depurar y realizar perfiles de corriente en tiempo real de su aplicación sin necesidad de utilizar herramientas externas[12]. A continuación, se muestran las características principales del EFM32 Giant Gecko:

- Plataforma de CPU ARM Cortex-M3
- 32 MHz
- Hasta 128 Flash
- Hasta 16 kB RAM
- 180 A/MHz en modo activo
- Periféricos autónomos en reposo.
- DMA
- USART, I2C y SPI
- Hasta 90 pines de E/S de propósito general
- Fuente de alimentación única de -1,98 V a 3,8 V

En la figura 56, se muestra el microcontrolador EFM32 Giant-Gecko. En este proyecto se utiliza como hub USB para los cuatro elementos programables del Front-End, recibiendo los mensajes de configuración desde el procesador Cortex R5 mediante SPI.

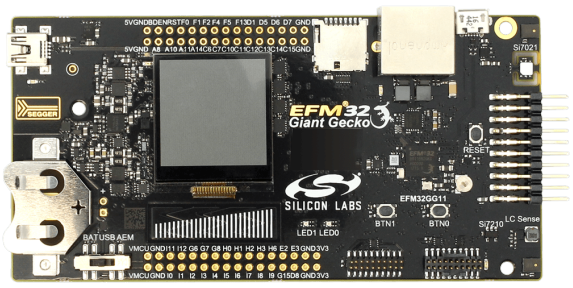


Figura 56: Vista superior del microcontrolador EFM32 Giant Gecko[12].

5.3.4. Atenuador RUDAT-6000-110

El RUDAT-6000-110 es un atenuador programable mono-canal de propósito general adecuado para una amplia gama de aplicaciones de control de nivel de señal de 1 MHz a 6 GHz. El atenuador proporciona una atenuación de 0 a 110 dB en pasos de 0,25 dB. Su diseño único mantiene un cambio de atenuación lineal por dB, incluso en los ajustes de atenuación más altos. El atenuador puede controlarse mediante USB o RS232[30].

En este proyecto, este atenuador es uno de los elementos programables del Front-End, se programa desde el microcontrolador EFM32 Giant Gecko.

5.3.5. EV-ADF4355

El EV-ADF4355 es un sintetizador de microondas de banda ancha con VCO integrado. Permite la implementación de sintetizadores de frecuencia de bucle bloqueado en fase (PLL) fraccional o entero cuando se utiliza con un filtro de bucle externo y una frecuencia de referencia externa. Una serie de divisores de frecuencia en la salida permite el funcionamiento desde 51,5625 MHz hasta 6600 MHz[13]. En la figura 57, se observa la tarjeta EV-ADF4355.

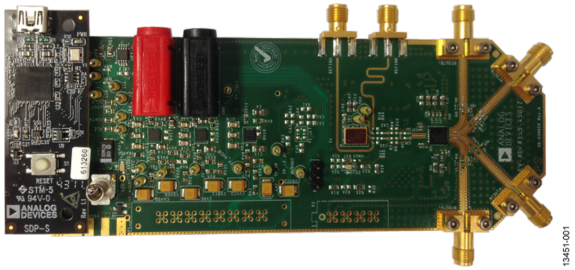


Figura 57: Tarjeta EV-ADF4355[13].

En este proyecto, el EV-ADF4355 es uno de los elementos programables del Front-End. Se utiliza para realizar la Downconversion y la Upconversion de las señales de entrada y salida del emulador.

5.3.6. Amplificador ADL5202

El ADL5202 es un amplificador de amplio ancho de banda, ganancia variable y control digital que proporciona un control preciso de la ganancia y un bajo factor de ruido. El excelente rendimiento de distorsión y el gran ancho de banda de señal hacen del ADL5202 un excelente dispositivo de control de ganancia para una gran variedad de aplicaciones de receptor. El ADL5202 también incorpora una opción de modo de baja potencia que reduce la corriente de alimentación[31]. A continuación se muestran algunas de sus características:

- Doble VGA independiente, controlada digitalmente
- Ganancia de -11,5 dB a +20 dB
- Step size de 0,5 dB
- Entrada y salida diferenciales de 150 ohm
- Figura de ruido de 7,5 dB a ganancia máxima
- Múltiples opciones de interfaz de control
    - Interfaz de control paralelo de 6 bits (con latch)
    - Interfaz periférica serie (SPI)
- Amplio rango dinámico de entrada
- Opción de modo de bajo consumo
- Funcionamiento con alimentación de 5 V

En este proyecto, el amplificador ADL5202 es uno de los elementos programables del Front-End. Se utiliza para optimizar el nivel de potencia de las señales de entrada a los ADC,así como de aumentar la potencia de la señal de salida del emulador en la Upconversion a RF.

5.4. Diagrama de Gantt

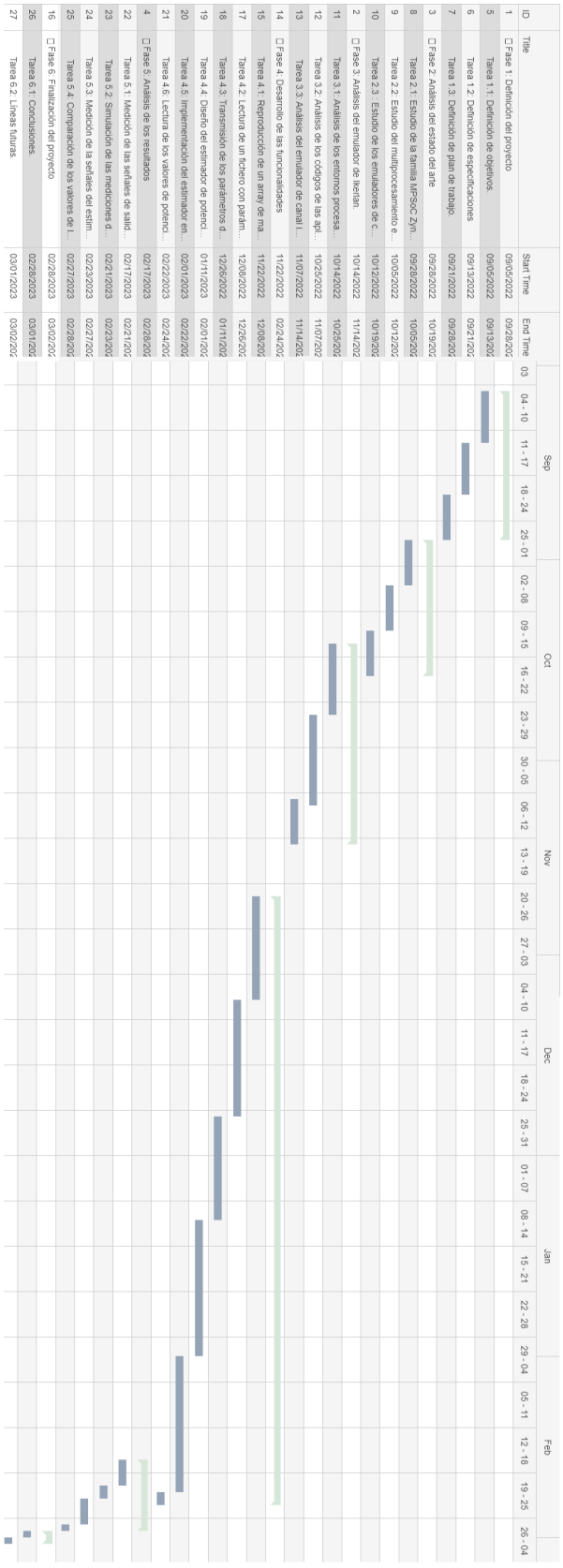


Figura 58: Diagrama de Gantt del proyecto.

6. Descripción del presupuesto

6.1. Horas internas

Los recursos empleados para este proyecto han sido un ingeniero junior, un ingeniero senior para dar soporte en diferentes fases del proyecto y el propio director del proyecto.

En la tabla 8, se detalla el coste horario de los participantes del proyecto.

Trabajador	Coste Horario (€/h)	Horas (h)	Coste (€)
Ingeniero Junior	20	600	12000
Ingeniero Senior	40	40	1600
Director del Proyecto	40	20	800
		SUBTOTAL	**14.400 €**

Tabla 8: Tasa horaria y distribución de horas de los recursos humanos.

El subtotal correspondiente a horas internas es de 14.400 €, teniendo en cuenta las tasas y el tiempo dedicado por cada una de las personas involucradas.

6.2. Amortizaciones

Para el cálculo de las amortizaciones se consideran los equipos informáticos utilizados en el desarrollo y las pruebas, así como las tarjetas de hardware utilizadas para desarrollar el emulador. Además, se incluye el software utilizado, ya que es propietario y su uso requiere el pago de licencias correspondientes.

Inversión	Coste (€)	Vida útil (meses)	Uso (meses)	Cantidad (uds)	Amortización (€)
Ordenador	1000	48	6	1	125,0 €
Licencias	6000	12	6	1	3.000,0 €
Tarjeta ZCU102	2500	48	6	1	312,5 €
Front-End	400	48	6	4	200,0 €
ADC FMC150	200	48	6	4	100,0 €
Osciloscopio	3000	120	6	1	150,0 €
Gen. Señal	800	96	6	1	50,0 €
				SUBTOTAL	**3.937,5 €**

Tabla 9: Coste de las amortizaciones.

El coste de las amortizaciones asciende a 3937,5€.

6.3. Gastos

A continuación en la tabla 10, se reflejan los gastos generados por el uso de recursos exclusivos para este proyecto.

Gastos	Coste (€)
Cableado	60 €
Tarjetas SD	30 €
Memoria USB	20 €
Material de oficina	50 €
SUBTOTAL	**160 €**

Tabla 10: Gastos del proyecto.

6.4. Presupuesto Total

La tabla 11 muestra el presupuesto total, el cual se obtiene al sumar los subtotales de las horas internas (tabla 8), amortizaciones (tabla 9) y gastos (tabla 10).

Concepto	Coste (€)
Horas internas	14.400,00 €
Amortizaciones/Inversiones	3.937,50 €
Gastos	160 €
TOTAL	**18.497,5 €**

Tabla 11: Coste total del proyecto.

7. Conclusiones

En este proyecto se han conseguido implementar dos nuevas funcionalidades a un emulador de canal RF basado en una tarjeta de procesamiento de la familia Zynq Ultrascale+. Para ello, se han cumplido los objetivos fijados en el apartado 1.3.

Primeramente, se ha realizado el análisis del estado del arte de varias de las tecnologías relacionadas con el proyecto. Se ha estudiado la arquitectura de los MPSoC de la familia Zynq Ultrscale+ de Xilinx. Además, se han estudiado las técnicas de multiprocesamiento en sistemas embebidos, haciendo hincapié en el framework de multiprocesamiento asimétrico openAMP. Finalmente, se han estudiado los efectos que sufren las señales inalámbricas al ser propagadas por el medio y cómo los emuladores de canal imitan estos fenómenos.

Segundo, se ha analizado el comportamiento del emulador de canal de Ikerlan para entender su funcionamiento. Se ha estudiado el funcionamiento de las diferentes partes del emulador, como el Front-End, los ADC y el MPSoC. Se ha analizado la arquitectura, así como las comunicaciones entre las diferentes partes del MPSoC; la interfaz gráfica, el procesador de tiempo-real y la lógica programable. Además, se han estudiado los códigos de las aplicaciones de los procesadores para poder desarrollar las nuevas funcionalidades.

Tercero, se ha desarrollado una funcionalidad que permite transmitir un archivo con coeficientes para la emulación del canal desde la interfaz gráfica al procesador Cortex R5 y aplicarlos sobre el emulador de canal implementado en la lógica programable del MPSoC. Para ello, ha sido necesario añadir una opción que habilitase la carga del fichero en la interfaz gráfica, que se ejecuta sobre un Linux en el procesador A53 y permite configurar los parámetros del canal. Se ha modificando el código de la aplicación para que también transmitiese los valores del archivo mediante openAMP al procesador R5. En este último, se ha modificado el código para poder realizar la recepción de los valores del archivo, así como la aplicación de los coeficientes recibidos del fichero en el emulador. Se ha conseguido transmitir ficheros con diferentes tamaños, pero ha sido necesario aumentar los tamaños de memoria compartida dedicada a ambos procesadores para agilizar la duración de la transmisión y también la memoria definida para el procesador R5, para poder compilar los coeficientes de ficheros de mayor tamaño.

Cuarto, se ha desarrollado un estimador de potencia para la señal de entrada con el objetivo de configurar los amplificadores del Front-End dinámicamente. Para conseguirlo, ha sido necesario estudiar el diseño y el funcionamiento del emulador de canal implementado en la lógica programable. Después, se ha diseñado el estimador de potencia en System Generator, se ha añadido al propio diseño del emulador de System Generator y se ha comprobado su funcionamiento mediante simulaciones en Simulink-MATLAB. Tras esto, se ha generado una IP a partir del nuevo diseño y se ha incluido en el proyecto de Vivado para incluir el estimador de potencia al diseño actual del emulador. Además, se ha añadido un analizador lógico embebido en el MPSoC para capturar las señales del

estimador y realizar las pruebas correspondientes para comprobar su correcto funcionamiento. Después de haber generado el nuevo bitstream, ha sido necesario actualizar el contenido de los archivos "*BOOT.BIN* e "*image.ub*"de la tarjeta SD. Para ello, se han regenerado estos ficheros mediante Petalinux, incluyendo la nueva descripción del hardware del sistema. Para finalizar, se ha realizado la emulación de un canal con el estimador implementado y los valores obtenidos por parte del analizador lógico se han considerado como correctos. Además, se ha realizado una comparativa entre los valores obtenidos en la simulación y en el analizador lógico.

En resumen, mediante este proyecto se han estudiado las tecnologías relacionadas con los sistemas embebidos MPSoC y los emuladores de canal. Se ha conseguido implementar una nueva funcionalidad que permite enviar transmitir un archivo con parámetros predefinidos del canal y aplicarlos sobre el emulador y se ha diseñado e implementado un estimador de potencia para poder medir la potencia de las señales de entrada.

En un futuro, haciendo uso del estimador podría implementarse un código en el procesador R5 que leyese los valores de la potencia y que automáticamente actuara sobre los amplificadores del Front-End, ajustando el valor de la ganancia dinámicamente y adaptando la potencia de la señal de entrada a los ADCs, para poder utilizar todo su rango dinámico.

En relación con el diseño del emulador, solamente el 32 % de las DSPs de la Zynq Ultrascale+ son utilizadas en el diseño, por lo que en un futuro se podrían aumentar el número de paths por canal e incluso añadir más canales al sistema. Aunque, eso conllevaría añadir un Front-End junto con otro ADC por cada nuevo canal.

8. Referencias

[1] "Página Web de la plataforma Zynq Ultrascale+ MPSoC." https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html.

[2] Xilinx, "Zynq UltraScale+ Device Technical Reference Manual (UG1085)," 2022.

[3] S. Alonso, J. Lazaro, J. Jimenez, L. Muguira, and U. Bidarte, "Evaluating the openamp framework in real-time embedded soc platforms," in *2021 XXXVI Conference on Design of Circuits and Integrated Systems (DCIS)*, pp. 1–6, 2021.

[4] Y. Sun, E. Li, G. Yang, Z. Liang, and R. Guo, "Design of a dual-core processor based controller with rtos-gpos dual operating system," in *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 1859–1864, 2019.

[5] P. Duan, C. Li, B. Zhang, and E. Wang, "Wipd: A robust framework for phase difference-based activity recognition," *Mobile Networks and Applications*, pp. 1–12, 2022.

[6] "Vivado Design Tools." https://www.xilinx.com/products/design-tools/vivado.html#editions.

[7] "Mathworks Brand Guidelines." https://es.mathworks.com/brand.html.

[8] "Vivado - System Generator for DSP." https://www.xilinx.com/support/documentation-navigation/design-hubs/2020-1/dh0014-vivado-system-generator-hub.html.

[9] "QT Creator Cross-platform IDE for Software Development." https://www.qt.io/product/development-tools.

[10] Xilinx, "ZCU102 Board User Guide (UG1182)," 2022.

[11] "ADC FMC150." https://www.abaco.com/products/fmc150-fpga-mezzanine-card.

[12] "EFM32 Giant Gecko Datasheet." https://www.silabs.com/documents/public/data-sheets/efm32gg-datasheet.pdf.

[13] "EV-ADF4355 Frecuency Synthesizer Datasheet." https://www.analog.com/media/en/technical-documentation/user-guides/EV-ADF4355-3SD1Z_UG-873.pdf.

[14] B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee, and B. Yin, "Smart factory of industry 4.0: Key technologies, application case, and challenges," *IEEE Access*, vol. 6, pp. 6505–6519, 2018.

[15] W. U. Bajwa, J. Haupt, A. M. Sayeed, and R. Nowak, "Compressed channel sensing: A new approach to estimating sparse multipath channels," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 1058–1076, 2010.

[16] "Página Web de Ikerlan S. Coop." https://www.ikerlan.es/.

[17] A. Larrañaga, M. C. Lucas-Estañ, I. Martinez, I. Val, and J. Gozalvez, "Analysis of 5g-tsn integration to support industry 4.0," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, pp. 1111–1114, 2020.

[18] G. Lacalle, I. Val, O. Seijo, M. Mendicute, D. Cavalcanti, and J. Perez-Ramirez, "Analysis of latency and reliability improvement with multi-link operation over 802.11," in *2021 IEEE 19th International Conference on Industrial Informatics (INDIN)*, pp. 1–7, 2021.

[19] . Seijo, I. Val, and J. A. López-Fernández, "Portable full channel sounder for industrial wireless applications with mobility by using sub-nanosecond wireless time synchronization," *IEEE Access*, vol. 8, pp. 175576–175588, 2020.

[20] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq book: embedded processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 all programmable SoC*. Strathclyde Academic Media, 2014.

[21] N. Bin, L. Dejian, Y. LiXin, B. Zhihua, H. Longlong, Z. Guang, and L. Meng, "Asymmetric software architecture design of high performance control chip applied in industrial control field," in *2021 4th International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE)*, pp. 916–920, 2021.

[22] NiuBin, L. Dejian, Y. LiXin, B. Zhihua, H. Longlong, Z. Guang, and L. Meng, "Asymmetric hardware and software integration design based on multi-core processor," in *2021 International Conference on Communications, Information System and Computer Engineering (CISCE)*, pp. 109–113, 2021.

[23] Xilinx, "Libmetal and OpenAMP User Guide (UG1186)," 2022.

[24] Y. Sun, E. Li, G. Yang, Z. Liang, and R. Guo, "Design of a dual-core processor based controller with rtos-gpos dual operating system," in *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 1859–1864, 2019.

[25] P.-F. Cui, J. A. Zhang, W.-J. Lu, Y. J. Guo, and H. Zhu, "Statistical sparse channel modeling for measured and simulated wireless temporal channels," *IEEE Transactions on Wireless Communications*, vol. 18, no. 12, pp. 5868–5881, 2019.

[26] R. Jia, Y. Li, X. Cheng, and B. Ai, "3d geometry-based uav-mimo channel modeling and simulation," *China Communications*, vol. 15, no. 12, pp. 64–74, 2018.

[27] E. Biglieri, J. Proakis, and S. Shamai, "Fading channels: information-theoretic and communications aspects," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2619–2692, 1998.

[28] A. F. Molisch, *Wireless communications*. John Wiley & Sons, 2012.

[29] M. Mfeze and E. Tonye, "Comparative approach of doppler spectra for fading channel modelling by the filtered white gaussian noise method," *International Journal of Computer Science and Telecommunications*, vol. 6, no. 11, pp. 1–12, 2015.

[30] "RUDAT-6000-110 Programmable Attenuator Datasheet." https://www.minicircuits.com/pdfs/RUDAT-6000-110.pdf.

[31] "ADL5202 Amplifier Datasheet." https://www.analog.com/media/en/technical-documentation/data-sheets/ADL5202.pdf.

[32] Xilinx, "Petalinux Tools Documentation: Reference Guide (UG1144)," 2023.

[33] "Spirent Vertex Channel Emulator Technical Specifications." https://www.solvitsystem.co.kr/product/brochures/03/06_Spirent_Vertex_Channel_Emulator_Data%20Sheet.pdf.

[34] "Keysight Propsim FS8 Channel Emulator Datasheet." https://www.keysight.com/us/en/assets/7018-05285/data-sheets/5992-1613.pdf.

[35] Xilinx, "Zynq UltraScale+ Embedded Design Tutorial User Guide (UG1209)," 2020.