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

Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)

Trabajo Fin de Máster

Sistema basado en SoC FPGA Zynq para aplicaciones en el campo de la astrofísica

Autora: Selenia María Medina Hernández
Tutor(es): Dr. Pedro Pérez Carballo
D. Pedro Hernández Fernández
D. Sergio González Martín-Fernández
Fecha: Julio 2022

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

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

Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)

Trabajo Fin de Máster

Sistema basado en SoC FPGA Zynq para aplicaciones en el campo de la astrofísica

HOJA DE FIRMAS

Alumna: Selenia María Medina Hernández

Tutor: Dr. Pedro Pérez Carballo
Tutor: D. Pedro Hernández Fernández
Tutor: D. Sergio González Martín-Fernández

Fecha: Julio 2022

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

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

Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)

Trabajo Fin de Máster

Sistema basado en SoC FPGA Zynq para aplicaciones en el campo de la astrofísica

HOJA DE EVALUACIÓN

Calificación: .....

Presidente:	Dr. Sebastián López Suárez	Fdo.:
Secretario:	Dr. Alfonso Medina Escuela	Fdo.:
Vocal:	Dra. Ernestina Martel Jordán	Fdo.:

Fecha: Julio 2022

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

AGRADECIMIENTOS

A mi familia, por su apoyo incondicional.

A mi tutor, Pedro Pérez Carballo, por la acertada orientación y por su paciencia.

A Sergio González Martín-Fernández, por su gran aportación en el proyecto y por ilustrar.

A Pedro Hernández Fernández y a David S. Miranda Guillén, por el soporte.

RESUMEN

El Instituto de Astrofísica de Canarias (IAC) participa en el Experimento Q-U-I JOin TEnerife (QUIJOTE) que tiene como objetivo caracterizar la radiación de Fondo Cósmico de Microondas (FCM) y otras emisiones galácticas y extra-galácticas en el rango de frecuencias 10-40 GHz. Las mediciones de QUIJOTE complementan, a baja frecuencia, las mediciones obtenidas por el satélite Plank (ESA).

Debido a los avances del Experimento QUIJOTE, el IAC solicita al Instituto Universitario de Microelectrónica Aplicada (IUMA) el diseño de un Sistema de Adquisición de Datos (DAS) escalable de dieciséis canales simultáneos a costos razonables. Según el IAC, el DAS tiene que amplificar y capturar a alta velocidad y con alta resolución, añadir marcas de tiempo real, calcular promedios y enviar por Ethernet las señales procesadas. Además, tiene que permitir elegir las ganancias desde el *software*.

Dadas las especificaciones, el investigador Sergio González Martín-Fernández y el IUMA diseñan un módulo DAS de treinta y dos canales compuesto por una fuente con reserva energética, dos tarjetas de adaptación que integran dieciséis amplificadores de ganancia variable y dos conversores AD7768 de Analog Devices, y dos *ZedBoard* que utilizan el dispositivo Zynq-7000 de Xilinx.

En el presente Trabajo de Fin de Máster se realiza el diseño del *System-on-Chip* FPGA a integrar en dicho módulo DAS, es decir, se realiza el diseño de un SoC FPGA de adquisición y procesamiento de dieciséis canales de forma simultánea y de control de un módulo DAS. Este *System-on-Chip* se desarrolla utilizando técnicas de diseño de alto nivel (C++) y RTL (Verilog), herramientas de desarrollo Xilinx y Cadence, metodologías de *diseño basado en plataformas* y reutilizando IPs de Xilinx y Analog Devices.

El SoC FPGA presenta una arquitectura heterogénea, es decir, está conformado por una plataforma *hardware* y una aplicación *software* de tipo *standalone*. A nivel *hardware*, el sistema interpreta los datos generados por los conversores AD7768 en función de su configuración, añade marcas de tiempo real con resolución de segundos o milisegundos en formato de tiempo Unix a las muestras, realiza el *bypass* o calcula el promedio de las muestras y almacena en memoria *On-Chip* las muestras

procesadas. Por otro lado, a nivel *software*, el sistema guarda en ficheros de datos las muestras procesadas mediante un Sistema de Ficheros FAT local, concretamente en RAM, y, a través de Ethernet, envía dichos ficheros a un servidor remoto utilizando el protocolo TFTP.

Además, el SoC permite controlar y configurar el módulo DAS por medio de comandos transferidos por un puerto UART y un menú de configuración interactivo. Por ejemplo, permite controlar la *captura* de las señales, elegir las ganancias de amplificación, poner en *standby* los canales deseados, etc. Asimismo, el SoC es capaz de apagar el módulo de forma controlada en caso de caída de la red eléctrica

Al final del presente Trabajo de Fin de Máster se obtiene un DAS de dieciséis canales simultáneos basado en FPGA que amplifica con diferentes ganancias, captura a alta velocidad (256kSP/s) y con una resolución de 24 bits, añade marcas de tiempo real Unix de segundos o milisegundos a las muestras, calcula el promedio de las muestras y envía a una estación remota a través de Ethernet las señales procesadas en formato ficheros de datos. Además, se logra un DAS que puede ser controlado y configurado por medio de comandos y un menú de configuración y que se detiene en caso de corte del suministro eléctrico.

ABSTRACT

The Instituto de Astrofísica de Canarias (IAC) participates in the Q-U-I JOin TEnerife Experiment (QUIJOTE) which aims to characterize the Cosmic Microwave Background (CMB) radiation and other galactic and extra-galactic emissions in the 10-40 GHz frequency range. The QUIJOTE measurements complement at low frequencies the measurements obtained by the Plank satellite (ESA).

Due to the progress of the QUIJOTE experiment, the IAC asked the Instituto Universitario de Microelectrónica Aplicada (IUMA) to design a scalable Data Acquisition System (DAS) with sixteen simultaneous channels at reasonable costs. According to the IAC, the DAS has to amplify and capture at high speed and high resolution, add real-time stamps, calculate averages and send the processed signals over Ethernet. In addition, it has to allow the gains to be chosen from the software.

Given the specifications, the researcher Sergio González Martín-Fernández and the IUMA designed a thirty-two-channel DAS module composed of a power supply with power reserve, two adaptation cards that integrate sixteen variable-gain amplifiers and two Analog Devices AD7768 converters, and two ZedBoard that use the Xilinx Zynq-7000 device.

In this Master's Thesis, the FPGA System-on-Chip to be integrated in this DAS module is designed, i.e. an FPGA SoC for the acquisition and processing of sixteen channels simultaneously and for the control of a DAS module. This System-on-Chip is developed using high-level design techniques (C++) and RTL (Verilog), Xilinx and Cadence development tools, platform-based design methodologies (PBD) and reusing Xilinx and Analog Devices IPs.

The FPGA SoC has a heterogeneous architecture, i.e. it is composed of a hardware platform and a standalone software application. At the hardware level, the system interprets the data generated by the AD7768 converters according to their configuration, adds real time stamps with second or millisecond resolution in Unix time format to the samples, bypasses or averages the samples and stores the processed samples in On-Chip memory. On the other hand, at the software level, the system

stores the processed samples in data files using a local FAT File System, specifically in RAM, and sends these files via Ethernet to a remote server using the TFTP protocol.

In addition, the SoC allows the DAS module to be controlled and configured by means of commands transferred via a UART port and an interactive configuration menu. For example, it allows to control the capture of the signals, to choose the amplification gains, to put the desired channels in stand-by, etc. The SoC is also capable of shutting down the module in the event of a power failure.

At the end of this Master's Thesis, an FPGA-based DAS with sixteen simultaneous channels is obtained, which amplifies with different gains and captures at high speed (256kSP/s) and with a resolution of 24 bits, adds Unix real time stamps of seconds or milliseconds to the samples, calculates the average of the samples and sends the processed signals in data file format to a remote station via Ethernet. In addition, a DAS is achieved which can be controlled and configured by means of commands and a configuration menu and which stops in the event of a power failure.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

ACRÓNIMOS

ACP	*Accelerator Coherency Port*
ADC	*Analog-Digital Converter*
AXI	*Advanced eXtensible Interface*
BEM	*Back-End Module*
BICEP	*Background Imaging of Cosmic Extragalactic Polarization*
CDC	*Clock Domain Crossing*
CMB	*Cosmic Microwave Background*
COBE	*Cosmic Background Explorer*
CRC	*Cyclic Redundancy Check*
DAS	*Data Acquisition System*
EDLC	*Electrical Double Layer Capacitor*
FEM	*Front-End Module*
FGI	*Forty-GHz Instrument*
FMC	*FPGA Mezzanine Card*
FCM	Fondo Cósmico de Microondas
FPGA	*Field Programmable Gate Array*
IAC	Instituto de Astrofísica de Canarias
IP	*Intellectual Property*
IUMA	Instituto Universitario de Microelectrónica Aplicada
LPC	*Low Pin Count*
LUT	*Look-Up Table*
MFI	*Multi-Frecuency Instrument*
OCM	*On-Chip Memory*
PL	*Programmable Logic*
PS	*Processing System*
QUIET	*Q/U Imaging ExperimenT*
QUIJOTE	*Q-U-I JOint TEnerife*

RTC	*Real Time Clock*
RTL	*Register Transfer Level*
SoC	*System-on-Chip*
TGI	*Thirty-GHz Instrument*
TMS	*Tenerife Microwave Spectrometer*
UPS	*Uninterruptable Power Supply*
WMAP	*Wilkinson Microwave Anisotropy Probe*
XFC	*Xilinx Constraints File*
XSA	*Xilinx Support Archive*

MEMORIA

Capítulo 1. INTRODUCCIÓN

En este capítulo, se exponen los antecedentes y objetivos que persigue el presente Trabajo de Fin de Máster. Además, se presenta la estructura del documento.

1.1. Introducción

Durante miles de años, el ser humano ha observado el universo con el fin de responder a la gran pregunta: *¿de dónde venimos?* Según el modelo cosmológico de concordancia la respuesta es el *Big Bang* [1] [2].

El *Big Bang*, también conocido como la *Gran Explosión*, es la teoría cosmológica que explica el origen y la evolución del universo. Este modelo postula que el universo observable tiene aproximadamente 13,7 billones de años y, que en ese entonces, se concentraba en unos pocos milímetros de diámetro y era extremadamente caliente y denso [3]. La teoría del *Big Bang* expone que el universo sufrió una 'explosión' o comienzo de una expansión exponencial que ocasionó el descenso progresivo de la temperatura y densidad. Explica que, 380 mil años después, empezaron a formarse los primeros átomos neutros, mayoritariamente el hidrógeno, y que esto originó la libre propagación de la radiación por todo el universo [4]. Esta radiación es conocida como Fondo Cósmico de Microondas (*Cosmic Microwave Background* o CMB) y es la radiación electromagnética más antigua del universo [5]. Avanzando en la cronología del cosmos, se inicia la creación de las estrellas y galaxias, a partir de las regiones más densas en materia, y el resto de las estructuras astronómicas ya conocidas. Actualmente, está demostrado que el universo continúa en expansión y que, además, si se orientara un radiotelescopio de gran sensibilidad hacia cualquier dirección del cielo sería posible detectar radiación de Fondo Cósmico [6].

La radiación de Fondo Cósmico de Microondas fue postulada en 1948 por los físicos Gamow, Alpher y Herman [7] y, en 1965, detectada por primera vez y de forma accidental por los ingenieros de *Bell Telephone Labs*, Arno Allan Penzias y Robert Woodrow Wilson [8]. Su descubrimiento proporcionó una evidencia más de la Teoría del *Big Bang* y motivó el desarrollo de instrumentos de

radioastronomía ultrasensibles con la finalidad de caracterizar la radiación de Fondo y, por ende, ampliar la comprensión del cosmos.

Misiones espaciales como *Cosmic Background Explorer* (COBE) [9], primer satélite construido especialmente para el estudio de la cosmología, *Wilkinson Microwave Anisotropy Probe* (WMAP) [10] y, la más reciente, PLANCK [11] se han dedicado al análisis de la variación de temperatura y polarización del CMB presentando cada cual mayor precisión que el anterior en la medida (Figura 1). Asimismo, experimentos en la Tierra como *Q/U Imaging ExperimenT* (QUIET) [12] ubicado en Chile, *Background Imaging of Cosmic Extragalactic Polarization* (BICEP) [13] ubicado en la zona del Tratado Atlántico y *Q-U-I JOint TEnerife* (QUIJOTE) [14] ubicado en Tenerife.

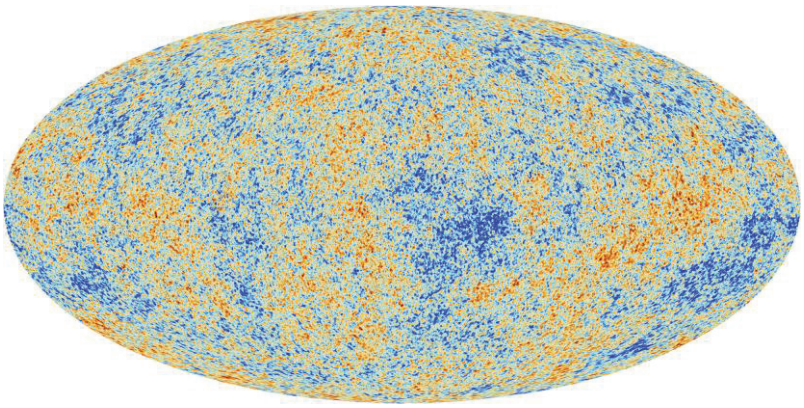

Figura 1. Radiación de Fondo Cósmico de Microondas mapeado por el satélite Planck en 2018

Hoy en día, la radiación posee características propias de radiación de cuerpo negro, tiene una temperatura de 2.725 Kelvin por encima del cero absoluto (-454.77 grados Fahrenheit o -270.43 grados Celsius), pertenece al rango del espectro en frecuencia de microondas con una frecuencia de 160.2 GHz y presenta una longitud de onda de 1.9 mm. Además, posee un nivel de polarización del orden de microkelvins cuyo patrón, generalmente, se descompone en un gradiente rotacional libre único o modo-E (nombrado en analogía a los campos electroestáticos) y en una divergencia rotacional libre o modo-B (nombrado en analogía a los campos gravitacionales) [15]. Los modos-E fueron por primera vez detectados en 2002 por el Experimento DASI [16]. Sin embargo, los modos-B aún permanecen sin ser medidos, pero se estima que posean una amplitud inferior a 0.1μK.

La detección de los modos-B del Fondo Cósmico sería un hito sin precedentes para la cosmología, puesto que confirmaría que las ondas gravitatorias fueron creadas durante la inflación cósmica o expansión exponencial del universo primigenio. Incluso, la determinación de su amplitud proporcionaría información sobre la escala de energía a la que se produjo la *Gran Explosión* [17]. Esto es una tarea extremadamente compleja, debido a la inmensa contaminación galáctica y a la combinación de los modos-E con los modos-B por las lentes gravitacionales. Por consiguiente, la caracterización de la polarización de la radiación CMB, no solo requiere una medición precisa del Fondo Cósmico,

sino que también requiere la caracterización de las emisiones de fondo galáctico (sincrotrón galáctico).

1.2. Experimento Q-U-I JOint TEnerife (QUIJOTE)

Este experimento terrestre es una colaboración científica entre el Instituto de Astrofísica de Canarias (IAC), el Instituto de Física de Cantabria (IFCA-CSIC-UC), el Departamento de Ingeniería de Comunicaciones de la Universidad de Cantabria (DICOM-UC), el observatorio *Jodrell Bank* de Manchester, el laboratorio *Cavendish* de Cambridge y la compañía IDOM [18] que se inicia en noviembre de 2012 y se localiza en el Observatorio del Teide (Izaña, Tenerife) a una altitud de 2400 metros sobre el mar.

Ante las premisas citadas, QUIJOTE se crea con el fin de complementar a baja frecuencia (10-40 GHz) y corregir de la contaminación galáctica las mediciones obtenidas a alta frecuencia (más de 100 GHz) por el satélite Planck. Por estas razones, se centra en la caracterización de la polarización del CMB, en especial los modos-B, y señales galácticas y extra-galácticas a las frecuencias 11, 13, 17, 19, 30 y 40 GHz, con una sensibilidad de 1μK (en las frecuencias más altas), con una resolución angular de 1º y cubriendo un área del cielo de 5 000 grados cuadrados [19].

QUIJOTE está compuesto por dos etapas diferentes denominadas FASE I y FASE II. La FASE I comprende la construcción del primer telescopio (QT1) y dos instrumentos: el Instrumento de Multi-Frecuencia (*Multi-Frecuency Instrument* o MFI) y el Instrumento de Treinta GHz (*Thirty-GHz Instrument* o TGI). La FASE II abarca la construcción del segundo telescopio (QT2) y un instrumento: el Instrumento de Cuarenta GHz (*Forty-GHz Instrument* o FGI) [19].

El *Multi-Frecuency Instrument*, con el fin de caracterizar la emisión galáctica, consta de cuatro polarímetros independientes que operan en el rango de frecuencias 10-20 GHz, donde dos funcionan a 11 y 13 GHz y otros dos a 17 y 19 GHz con un ancho de banda de 2GHz. Cada polarímetro está formado por un módulo *front-end* (FEM) a 20K y por un módulo *back-end* (BEM) a 300K [19].

El módulo *front-end* dispone de una bocina, antena o píxel, un modulador polar giratorio, un duplexor de polarización (*OrthoMode Transducer* o OMT) y dos amplificadores de bajo ruido (*Low Noise Amplifiers* o LNA). El módulo *back-end* está compuesto por LNA, interruptores de fase, filtros paso banda, detectores DC y amplificadores CC, entre otros; y proporciona ocho salidas (ocho canales) de tensión continua. Con estos valores, se obtienen los parámetros de *Stokes* I, Q, y U a las frecuencias 11 y 13 GHz o 17 y 19 GHz, los cuales permiten conocer el estado de la polarización galáctica, es decir, la intensidad o temperatura de la onda electromagnética (I) y los modos-E y B (Q, U) que caracterizan a la polarización (Figura 2).

Dichas tensiones son recopiladas por una Electrónica de Adquisición de Datos (*Data Acquisition Electronics* o DAE) a 300K y, en este punto, son digitalizadas a través de Conversores Analógico-Digital (*Analog-Digital Converter* o ADC) de 24 bits y enviadas a una estación central a través de Ethernet [20].

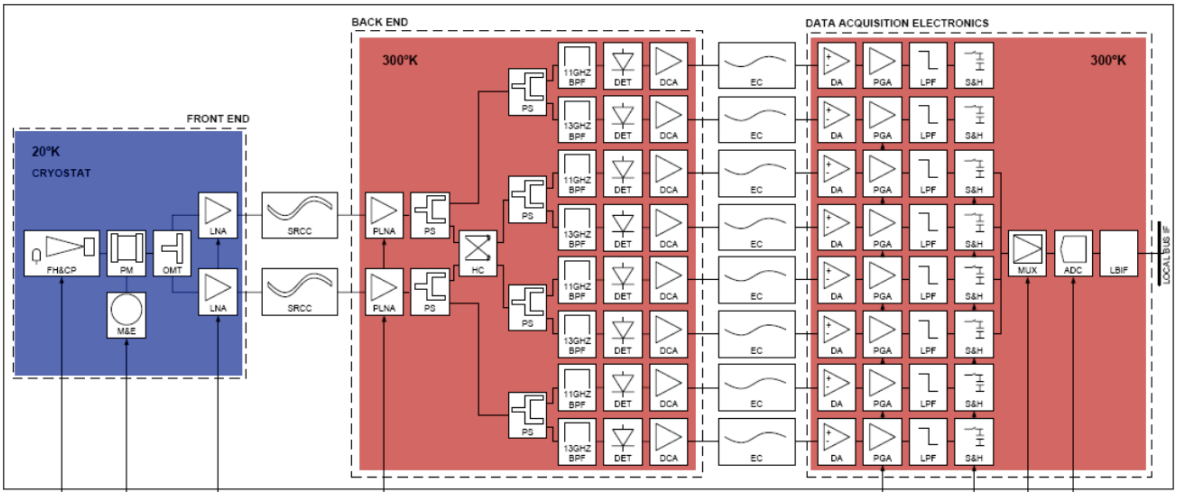

Figura 2. Un diagrama de bloques detallado del polarímetro de 11 y 13 GHz [20].

A diferencia del MFI, el *Thirty-GHz Instrument* se centra en la medición de la polarización de los modos-B del Fondo Cósmico. Por ello, está conformado por treinta y un polarímetros independientes que operan en el rango 26 y 36 GHz con un ancho de banda de 8 GHz [19] [21].

Cada polarímetro del TGI también consta de un módulo *front-end* y un módulo *back-end*. Sin embargo, los polarímetros del TGI presentan diferencias de concepto con respecto al de los MFI. Estos no poseen un modulador polar giratorio, sino un polarizador fijo combinado con dos interruptores de fase de 90º y 180º a 298 K, y los módulos *back-end* ofrecen cuatro canales de tensión continua para la obtención de los coeficientes de *Stokes*, en lugar de ocho, puesto que cada polarímetro trabaja a una frecuencia en concreto [22] (Figura 3). En la Figura 4, se observan los parámetros de *Stokes* obtenidos a partir de la captura a 1 Hz de los canales de salida del instrumento TGI, cuyas amplitudes pico-pico son del orden de milivoltios.

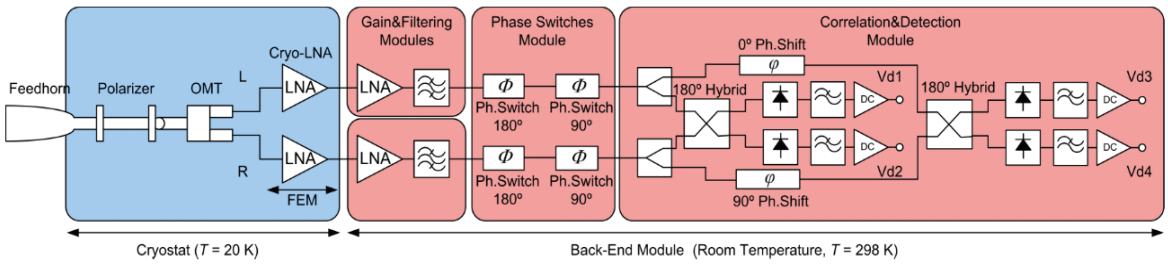

Figura 3. Un diagrama de bloques de un polarímetro del TGI [22].

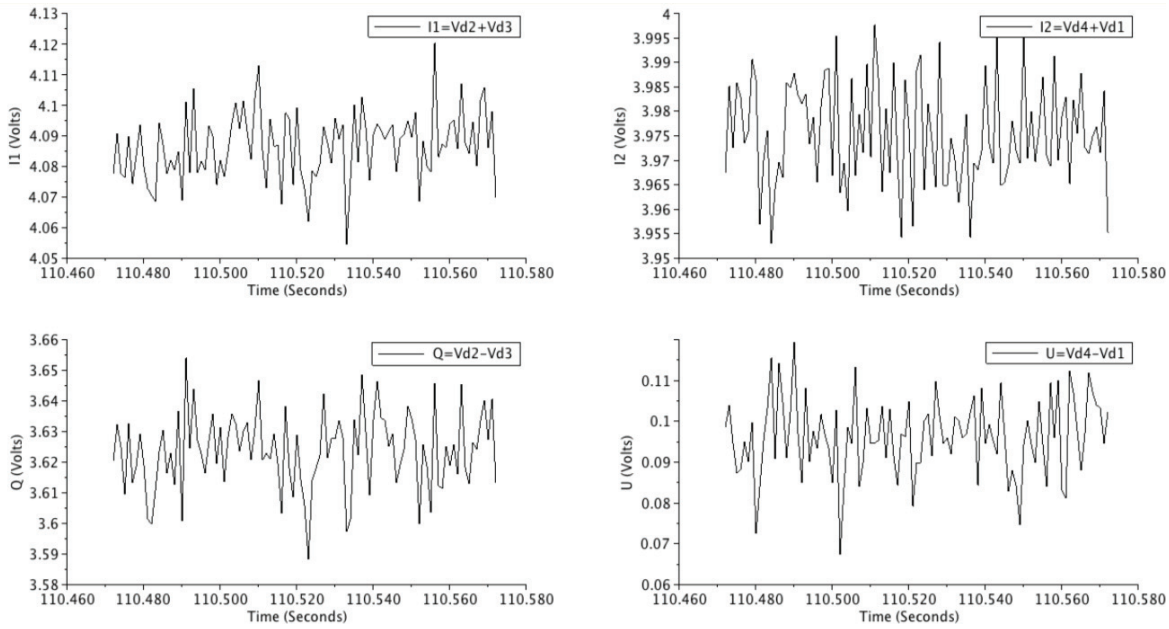

Figura 4. Parámetros de Stokes Q, U, I1 e I2 medidos a partir del TFI [23].

Los polarímetros del *Forty-GHz Instrument* están basados en los del TGI y, también, se dedican a la polarización de los modos-B. Este posee treinta y un polarímetros y, a diferencia del TFI, operan en el rango 35 y 47 GHz con un ancho de banda de 10 GHz [21].

En la Tabla 1, se exponen las principales diferencias entre los instrumentos.

Tabla 1. Especificaciones nominales de los tres instrumentos QUIJOTE: MFI, TGI y FGI [21].

	MFI	**TGI**	**FGI**
Frecuencia Nominal (GHz)	11 13 17 19	30	40
Ancho de banda (GHz)	2 cada uno	8	10
Nro. de bocinas	4	31	31
Nro. de canales por bocina	8	4	4
***Beam FWHM* (∘)**	0.92 0.92 0.6 0.6	0.37	0.38
Temperatura del sistema (K)	25	35	45
Sensibilidad (Jy$\sqrt{s}$)	0.42 0.59 0.44 0.54	0.06	0.06

Por último, con respecto a los telescopios QT1 y QT2, son telescopios de microonda de montura altazimutal armados sobre una plataforma que gira alrededor de un eje vertical a una frecuencia máxima de 15 rpm y 10 rpm que soportan un espejo primario parabólico con una apertura de 2,25m y otro secundario hiperbólico de 1.89 m dispuestos de forma *cross-dragonian* y capaces de funcionar hasta una frecuencia de 90 GHz y 200 GHz [19] (Figura 5).

Figura 5. Telescopios del experimento QUIJOTE [24].

1.3. Antecedentes

QUIJOTE continúa con su actividad y con su evolución. Actualmente, los instrumentos *Thirty-GHz* y *Forty-GHz* están combinados (14 píxeles de 30 GHz y 15 píxeles de 40 GHz) e integrados en el telescopio QT2. Además, de forma paralela, se están desarrollando dos nuevos instrumentos: el *Multi-Frecuency*-2 (MFI-2) que presenta mejores prestaciones que el actual MFI , y el *Tenerife Microwave Spectrometer* (TMS) que complementa las mediciones del experimento a baja frecuencia (10-20 GHz) [25].

El Instituto de Astrofísica de Canarias, frente a los avances del proyecto, observa la necesidad de ampliar el número de polarímetros (hasta 60 píxeles) y, en consecuencia, obtener un Sistema de Adquisición de Datos (*Data Acquisition System* o DAS) capaz de amplificar, capturar y procesar las salidas de estos (hasta 240 señales) con alta velocidad de muestreo y resolución (hasta 208 kSP/s y al menos 24 bits). No obstante, debido a la escasez de oferta en instrumentación con dichas prestaciones, así como la gran inversión económica que implica su elaboración, el IAC plantea la confección de un DAS a costos razonables.

Por consiguiente, el Instituto de Astrofísica de Canarias propone el estudio, diseño y desarrollo de una unidad escalable o *rack* capaz de amplificar, capturar y procesar hasta 60 señales analógicas débiles (15 píxeles) con unas dimensiones de 19" y 3U y una fuente de alimentación de 230 Vac. Además, sugieren que cada módulo o *slot* del *rack* sea capaz de tratar hasta 16 señales analógicas débiles (4 píxeles) y de sincronizarse con el resto de los módulos con el fin de tratar múltiples pixeles de forma simultánea (Figura 6).

Con respecto al tratamiento, recomiendan que cada módulo:

- Amplifique por los factores 2 (6 dB), 4 (12 dB), 6 (16 dB), 8 (18 dB) y 10 (20 dB) y que permita seleccionar dichos factores vía *software*.
- Digitalice con alta velocidad de muestreo (hasta 208 kSP/s) y resolución (al menos 24 bits).
- Procese utilizando una *Field Programmable Gate Array* (FPGA). Como procesamiento proponen: almacenar temporalmente los datos capturados (más de 4095 datos), promediar cada *n* muestras y añadir marcas de tiempo real en segundos o milisegundos.
- Transmita los resultados obtenidos a través del bus Ethernet.

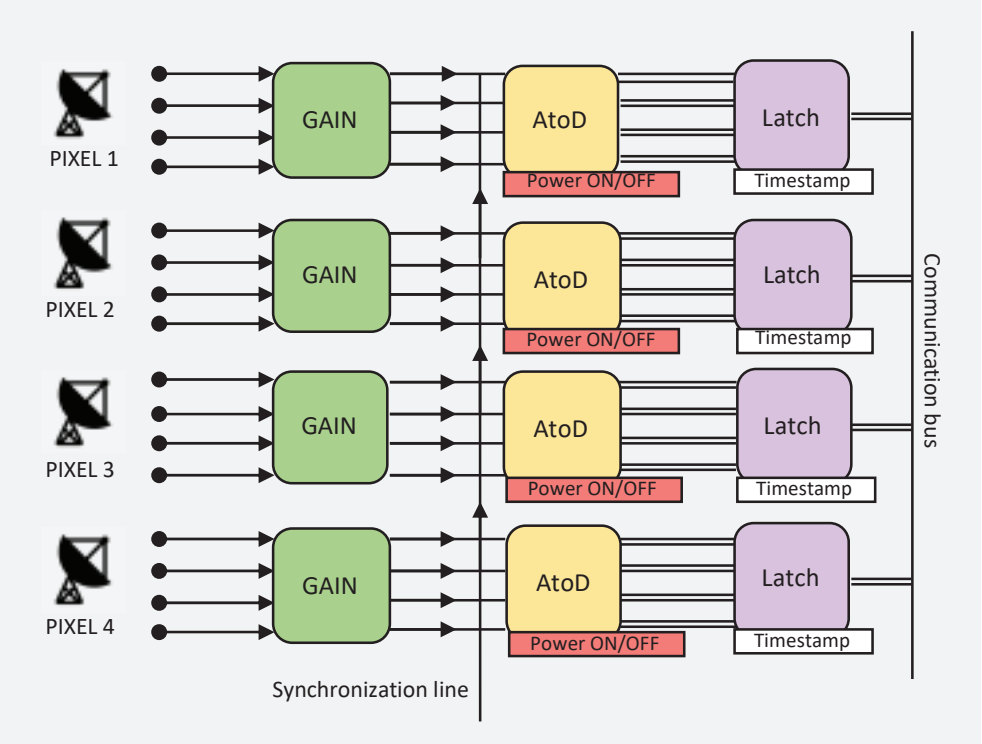

Figura 6. Esquema de un módulo del DAS propuesto por el IAC.

En base a los requerimientos especificados por el IAC, se ha desarrollado un módulo DAS de forma colaborativa entre el investigador Sergio González, encargado del diseño del equipo, la fuente de alimentación y el sub-sistema analógico, y el Instituto Universitario de Microelectrónica Aplicada (IUMA), encargado del diseño del sub-sistema de procesamiento digital basado en *System-on-Chip* (SoC) FPGA. El sistema diseñado se denomina *16/32 Channel Simultaneous Gain Controller 24 bits Fast A/C Converter*.

Este módulo DAS se ha elaborado con la idea de tratar hasta 32 señales (8 píxeles) y, debido a sus dimensiones y a las prestaciones exigidas en el procesamiento, está compuesto por componentes electrónicos que presentan un consumo de corriente moderado.

Este DAS está conformado por los siguientes elementos (Figura 7):

- Una fuente de alimentación que, a partir de una tensión externa de 230 Vac, suministra las tensiones requeridas por los diferentes componentes del módulo DAS. Esta fuente, en caso de corte de la red eléctrica, posee una autonomía de 3.5 minutos y genera una señal de alerta (AC *fail*).

- Dos tarjetas compuestas por dieciséis amplificadores de instrumentación y dos conversores analógico-digital AD7768 de ocho canales cada uno de Analog Devices [26], denominada *AD7768 AMP DIFF 16 CH*. Cada tarjeta permite la amplificación por los factores 1 (0 dB), 50 (34 dB), 100 (40 dB), 200 (46 dB) y 300 (50 dB) y la conversión analógico-digital de hasta dieciséis señales de entrada.

  La elección de estos factores se debe a que las señales a recibir del BEM son del orden de milivoltios, que, sin una amplificación notable, presentan problemas de rango en la conversión A/D. Los amplificadores elegidos son de bajo ruido y consumo, de entrada diferencial y compatibles con la selección del factor de forma. Los conversores A/D presentan una velocidad de muestreo máxima de 256 kSP/s, una resolución de 24 bits, bajo consumo de corriente, utiliza técnicas de conversión *sigma-delta* y entradas diferenciales. Además, son capaces de sincronizarse entre sí con el fin de digitalizar más de ocho señales de forma simultánea y son configurables por pines presentes en el dispositivo o a través del protocolo SPI.

- Dos placas basadas en SoC FPGA *ZedBoard* de Avnet [27] que utilizan el dispositivo Xilinx SoC Zynq XC7Z020-1CLG484C [28]. Ambas placas están integradas en el módulo DAS de forma que pueden controlar el suministro de tensiones interno, recibir las dieciséis señales acondicionadas (amplificadas y digitalizadas), seleccionar las ganancias de amplificación y configurar los conversores, a través del conector *FPGA Mezzanine Card* (FMC) que integran, y presentar los datos procesados en la salida Ethernet. La selección de estas *ZedBoard* se debe a que permiten diseñar un *System-on-Chip* FPGA de alto rendimiento y de bajo consumo de potencia a bajo coste.

  Estas disponen de dos núcleos ARM Cortex-A9 capaces de operar hasta a una frecuencia de 866 MHz, da la posibilidad de utilizar uno de estos procesadores o ambos para dar soporte a multiprocesamientos simétricos (SMP) o asimétricos (AMP) y, su lógica programable (*Programmable Logic* o PL), incluye recursos lógicos como *Look-Up Table* (LUT) empleados para la creación de funciones lógicas, registros y memoria local tipo bloque (BRAM), así como bloques DSP multifuncionales para la implementación de operaciones de multiplicación y acumulación.

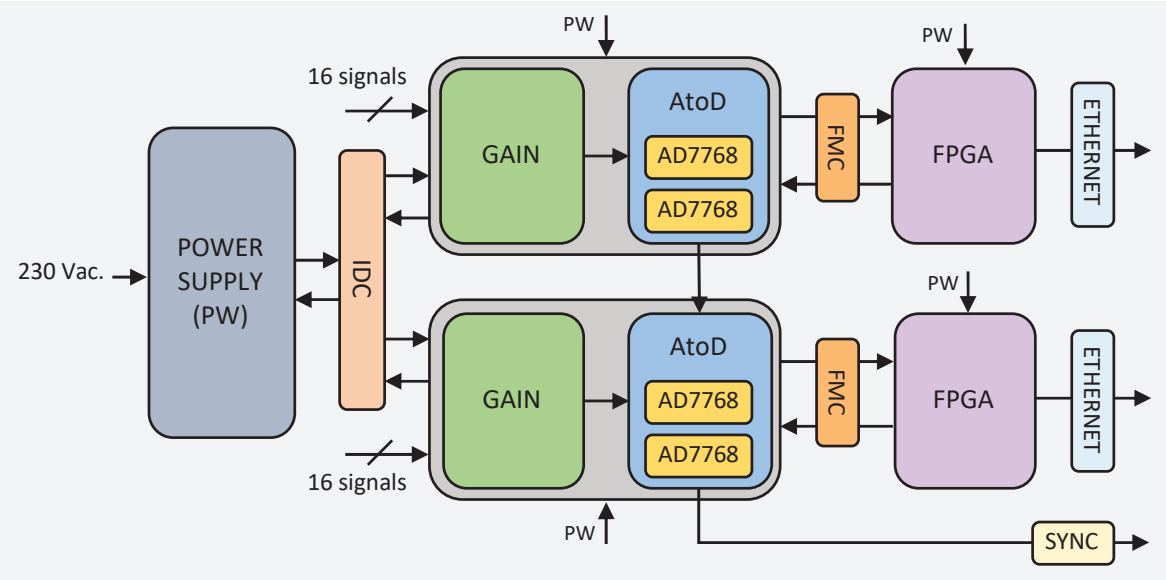

Figura 7. Esquema resumido del módulo DAS.

1.4. Objetivos

El objetivo del presente Trabajo de Fin de Máster consiste en diseñar un SoC basado en FPGA utilizando técnicas de diseño de alto nivel (C++) y RTL (Verilog) que, con un consumo de potencia moderado, realice las siguientes funciones:

F1. Adquiera de forma simultánea hasta dieciséis señales digitalizadas por los ADCs AD7768 de Analog Devices.

F2. Procese de forma digital y simultánea hasta dieciséis señales digitales. Este procesamiento debe cubrir los requisitos dictados por el IAC (ej.: promediar cada *n* muestras, insertar marcas de tiempo real con resolución de segundos o milisegundos a las muestras, etc.).

F3. Permita configurar la tarjeta *AD7768 AMP DIFF 16 CH*, es decir, que permita seleccionar las ganancias de amplificación de las dieciséis señales de entrada y configurar los dos ADCs AD7768 de Analog Devices por PIN y SPI. Esta configuración debe poder realizarse desde una aplicación *software*.

F4. Permita controlar la fuente de alimentación interna del módulo DAS. Este control debe poder realizarse desde una aplicación *software*.

F5. Supervise el estado de la red eléctrica externa al módulo DAS y que, en caso de caída, responda de forma controlada.

Alcanzar este objetivo supone lograr los objetivos operativos expuestos a continuación:

O1. Estudiar en detalle el módulo DAS desarrollado para definir los requisitos del SoC FPGA a diseñar.

O2. Estudiar las características del conversor AD7768 de 24-bits de Analog Devices y del dispositivo SoC Zynq-7000 de Xilinx.

O3. Estudiar la metodología de diseño SoC multi-dominio: soluciones *hardware-software*, síntesis y verificación de alto nivel (C++), co-verificación a través de entornos de desarrollo mixtos, verificación a nivel RTL, etc.

O4. Diseñar y verificar una plataforma *hardware* que integre las características exigidas, reutilice bloques *hardware* de propiedad intelectual (*Intellectual Property* o IP) y que, durante su diseño, se utilicen estándares de desarrollo *hardware*.

O5. Diseñar y verificar una aplicación *software* empotrada.

O6. Integrar la solución *software* con la solución *hardware*.

O7. Validar el SoC FPGA diseñado, comprobando que los resultados obtenidos son admisibles en función de las métricas de referencia.

1.5. Estructura del documento

El actual documento presenta la siguiente estructura de capítulos:

Capítulo 1. Se exponen los antecedentes y objetivos presente del Trabajo de Fin de Máster.

Capítulo 2. Se estudian los aspectos básicos de los componentes del módulo DAS y se exponen los requisitos del SoC FPGA a diseñar relativos al equipo.

Capítulo 3. Se expone la arquitectura del SoC FPGA que ofrece solución al presente Trabajo de Fin de Máster.

Capítulo 4. Se explica el flujo de diseño seguido y se describen las herramientas de desarrollo utilizadas, así como los dispositivos de instrumentación utilizados.

Capítulo 5. Se diseña la plataforma *hardware* del SoC FPGA que ofrece solución al presente Trabajo de Fin de Máster.

Capítulo 6. Se diseña la aplicación *software* empotrada y se realiza la integración *hardware-software* que da solución al presente Trabajo de Fin de Máster.

Capítulo 7. Se valida el SoC FPGA diseñado en el presente Trabajo de Fin de Máster y se valida el sistema completo (SoC FPGA + módulo DAS).

Capítulo 8. Se exponen las conclusiones del Trabajo de Fin de Máster y trabajos futuros.

Capítulo 2. MÓDULO 16/32 CHANNEL SIMULTANEOUS 24 BITS FAST A/C CONVERTER

En este capítulo, se estudian las características básicas de los componentes electrónicos que conforman el módulo DAS y se concluye con los requisitos del System-on-Chip FPGA a diseñar relativos al equipo.

2.1. Introducción

Tal y como se expuso en el capítulo anterior, el módulo *16/32 Channel Simultaneous Gain Controller 24 bits Fast A/C Converter* es un módulo de Adquisición de Datos (DAS) que permite el tratamiento de hasta 32 señales del orden de milivoltios, que, principalmente, está compuesto por los siguientes componentes (Figura 8):

- Una fuente de alimentación con acumuladores de energía.
- Dos placas de amplificación y conversión de señales conformadas por amplificadores de ganancia variable diferenciales y conversores A/D basados en el ADC AD7768.
- Dos tarjetas *ZedBoard,* que incluyen el dispositivo SoC FPGA Zynq-7000*,* dedicadas al procesamiento en el dominio digital.

A continuación, con la finalidad de identificar los requisitos del SoC FPGA a diseñar relativos al equipo, se exponen las principales características o puntos básicos a conocer de los componentes del DAS.

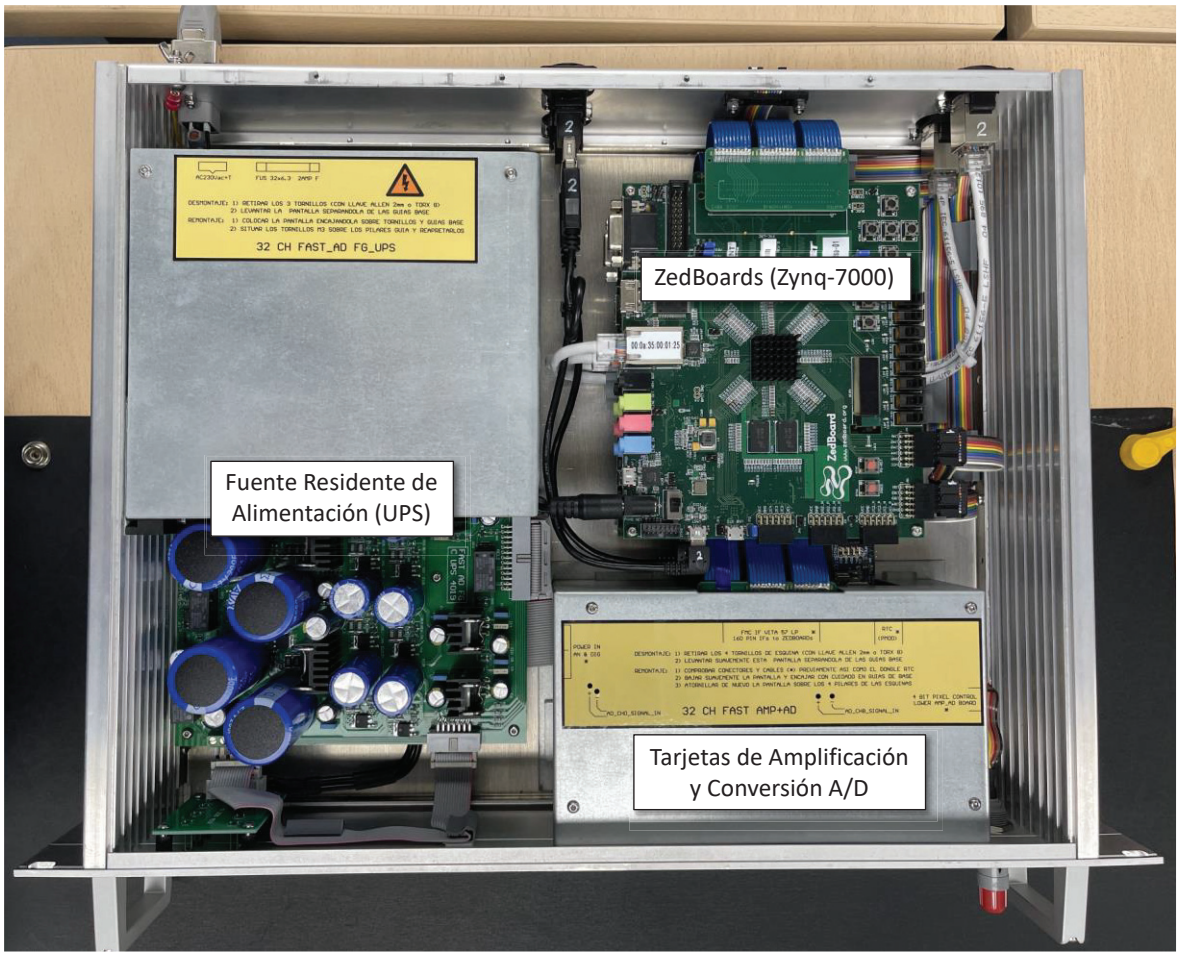

Figura 8. Componentes principales del módulo DAS.

2.2. Fuente de alimentación con funcionalidad UPS

La fuente de alimentación presenta características propias de un Sistema de Alimentación Ininterrumpido (*Uninterruptable Power Supply* o UPS). Está conformada por supercondensadores (*Electrical Double Layer Capacitor* o EDLC) de alta capacidad, reguladores lineales de baja caída (*Low-Dropout regulator* o LDO *regulator*) de bajo ruido y precisa estabilización, diodos *Schotty*, etc.

Esta UPS es capaz de generar, a partir de 230 Vac, cinco tensiones diferentes de alimentación para dos conjuntos AMP+A/D+FPGA de 16 canales de forma estable y ofrecer una continuidad de 3.5 minutos en dichas salidas de tensión en caso de corte de la red externa. Además, presenta tres masas diferentes y aisladas en origen: AGND, CGND y PGND.

Las tensiones que genera son las siguientes:

– +/-7.3 Vcc con corriente sostenida de 250 mA para los amplificadores.

– +5.4 Vcc (AVDD) con corriente sostenida de 250 mA y +3.3 Vcc (IOVDD) con corriente sostenida de 150 mA para los conversores A/D AD7768.

– +9.7 Vcc con corriente sostenida de 1.3 A para cada FPGA.

La tensión de +9.7 Vcc es directamente suministrada tras la alimentación a 230 Vac y el encendido través de un interruptor ON/OFF de la UPS. Esto enciende ambas *ZedBoard*. La fuente cuenta con dos relés (3.3 Vcc) que permiten habilitar o deshabilitar la generación de las tensiones +/-7.3 Vcc (AMP) y +5.4 Vcc (A/C). La secuencia de arranque y apagado de dicho DAS son inversas y, según recomienda el fabricante, debería ser la siguiente: primero los conversores A/D (+5.4 Vcc) y, después, los amplificadores de ganancia variable (+/-7.3 Vcc). Tanto el control de los relés como el suministro de las tensiones: +/-7.3 Vcc y +5.4 Vcc se realiza a través de un conector IDC (*Insulation-Displacement Connector*).

Por último, al sufrir un corte en la red eléctrica, la UPS genera una señal de alerta de 3.3V (AC *fail*), que puede ser supervisada a través dicho conector. Esta debe ser atendida con el fin de iniciar un protocolo de apagado controlado del sistema. El DAS dispone, aproximadamente, de 3.5 minutos para realizar dicha operación. Además, durante dicho tiempo, se enciende un piloto LED en la parte frontal del módulo que permite alertar al usuario de fallo en el suministro eléctrico.

2.3. Tarjeta AD7768 AMP DIFF 16 CH

Con el propósito de acondicionar las señales provenientes de los módulos *back-end*, el sistema diseñado dispone de dos tarjetas *AD7768 AMP DIFF 16 CH*. Además de distintos componentes pasivos, cada tarjeta está compuesta por dieciséis amplificadores de ganancia variable y dos conversores A/D AD7768 de 24 bits [26]. La combinación de dichos componentes electrónicos permite, en primer lugar, la amplificación y, en segundo lugar, la digitalización de hasta 16 señales analógicas de entrada por tarjeta.

Cada una de las tarjetas tiene un conector IDC y un conector FMC de tipo LPC. Desde el FMC y a través del IDC de ambas tarjetas, se puede controlar los relés que activan y desactivan las tensiones: +/-7.3 Vcc (AMP) y +5.4 Vcc (A/C); y supervisar la señal AC *fail*. Además, a través de los conectores FMC, cada tarjeta puede ser reconfigurada y puede transmitir hasta 16 señales digitalizadas.

A continuación, se exponen las características básicas de la etapa de amplificación y del conversor AD7768, así como la distribución de los pines en el conector FMC.

2.3.1. Amplificadores de Ganancia Variable

Con miras a cubrir los requisitos de amplificación especificados por el IAC (ver apartado 1.3 Antecedentes), la tarjeta integra los amplificadores operacionales de instrumentación: INA849 de

Texas Instruments [29] y AD8429 de Analog Devices [30]. Estos componentes permiten la amplificación de señales analógicas del orden de milivoltios (ultra-bajas figuras de ruido, alto CMRR, ultra-baja corriente de polarización, entrada diferencial, etc.), poseen un bajo consumo de potencia y son de bajo costo. Además, incorpora el operacional diferencial LTC6363 de Analog Devices [31], que permite convertir la salida *single-ended* del AD8429 y INA849 a una diferencial. Por último, dada la necesidad de poder elegir vía *software* la ganancia de amplificación, integra el multiplexor ADG1604 [32] que, a partir de sus entradas (EN, A0 y A1), permite seleccionar los factores de amplificación: 1 (0 dB), 50 (34 dB), 100 (40 dB), 200 (46 dB) y 300 (50 dB) según la Tabla 2. Dichas entradas, son controlables por medio del conector FMC de la tarjeta (Figura 9).

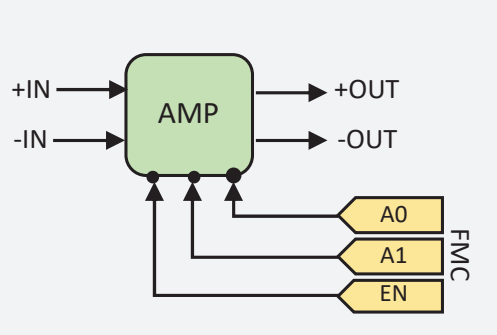

Figura 9. Amplificación de una entrada analógica (AMP+DIFF+MUX).

Tabla 2. Tabla de verdad del factor de amplificación.

EN	A1	A0	GAIN
0	0	0	1 (0 dB)
1	0	0	50 (34 dB)
1	0	1	100 (40 dB)
1	1	0	200 (46 dB)
1	1	1	300 (50 dB)

El esquema de la Figura 9 se repite dieciséis veces (uno por entrada analógica). No obstante, la selección del factor de amplificación, en lugar de ser exclusiva para cada entrada analógica (Figura 9), es por cada dos entradas analógicas consecutivas (Figura 10).

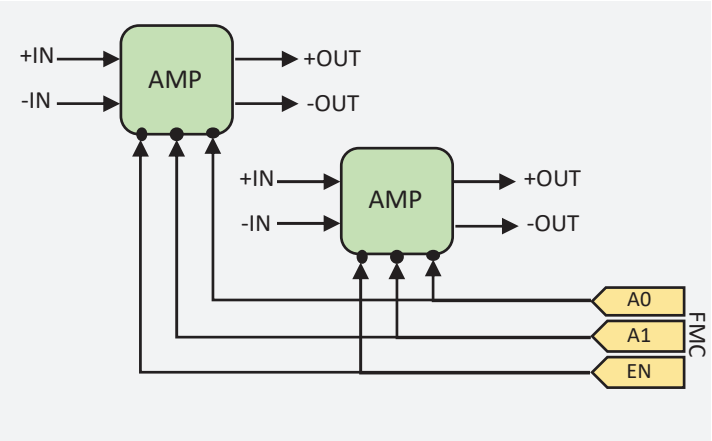

Figura 10. Esquema resumido de amplificación de dos entradas analógicas.

2.3.2. Conversor AD7768 de 24 bits de Analog Devices

Una vez amplificadas las señales analógicas a valores de tensión nominales, estas son sometidas a una conversión A/D, dada la naturaleza digital que su procesamiento. Nuevamente, dadas las exigencias de diseño planteadas por el IAC (ver apartado 1.3 Antecedentes), en el DAS se utiliza el ADC AD7768 de Analog Devices [26] como conversor analógico-digital.

Este conversor emplea la tecnología *sigma-delta* en la conversión A/D, dispone de ocho canales de muestreo simultáneos diferenciales, presenta una resolución de 24 bits y posee una velocidad de muestreo máxima de 256 kSP/s. Además, el AD7768 es de bajo consumo (Figura 11).

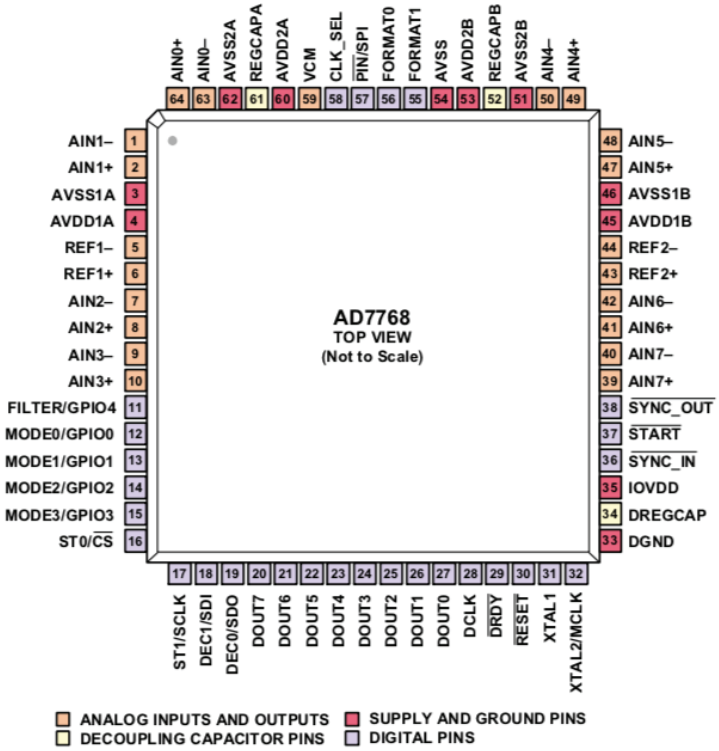

Figura 11. Configuración de pines del AD7768 [26].

A continuación, se describen aspectos básicos de la interfaz de datos y los modos de configuración del AD7768.

2.3.2.1. Interfaz de datos del A/D AD7768 de 24 bits.

El conversor AD7768 permite configurar diferentes aspectos de la Interfaz de Salida de Datos, como el número de líneas de datos, la frecuencia de salida de los datos, etc. Algunos de estos aspectos fueron fijados durante el tiempo de diseño de las tarjetas de adaptación, otros, en cambio, pueden ser reconfigurados durante el funcionamiento del sistema.

A continuación, se exponen en detalle las características de la Interfaz de Salida de Datos del AD7768, que permite explicar la solución elegida.

Formato de la interfaz de datos.

El conversor AD7768 cuenta con tres formatos de interfaz de datos, que determinan el número de líneas de datos a emplear para la transmisión de los datos digitalizados. En este caso, este formato está fijado al formato que utiliza una línea de datos por canal (Figura 12). Este formato permite recibir las muestras de los ocho canales de forma simultánea y procesar las señales de entrada, también, de forma simultánea.

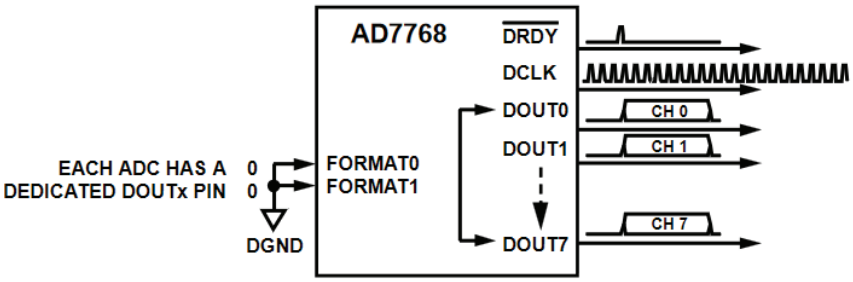

Figura 12. Formato de interfaz de dato fijado (FORMATO = 00) [26].

Datos de salida del conversor.

Los datos de salida del conversor tienen un ancho de 32 bits. Los 8 bits más significativos conforman la cabecera, que contiene información acerca del estado de la conversión y el número del canal, y los 24 bits menos significativos representan el dato digitalizado (Figura 13).

En la Tabla 3, se describen los diferentes campos de los datos de salida o muestras.

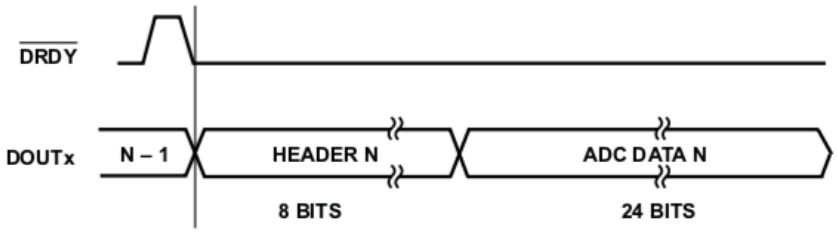

Figura 13. Datos de salida del conversor [26].

Tabla 3. Descripción de la trama de los datos de salida del conversor.

Bits	Nombre del campo	Descripción
31	ERROR_FLAGGED	Error grave en la conversión analógica-digital.
30	*Filter not settled*	Filtro no establecido. No se ha alcanzado el tiempo de establecimiento del filtro.
29	*Repeated data*	Dato repetido.
28	*Filter type*	Tipo de filtro empleado en la conversión: banda ancha (0) o *sinc5* (1).
27	*Filter saturated*	Recorte en la señal muestreada a escala completa o filtro saturado.
26:24	*Channel ID*	Canal a través del cual el dato ha sido muestreado: 000 (canal 0), 001 (canal 1), 010 (canal 2), etc.
23:0	*Data*	Dato de 24 bits representado en formato complemento a dos.

Resolución de la conversión.

La resolución del conversor AD7768 se calcula a partir de la fórmula (**1**). En este caso, ambos conversores están alimentados con una tensión de referencia (V_{REF}) de 5 V. Esta V_{REF} es la máxima admitida por el A/D y permite una entrada analógica de ± 5 V. Por lo tanto, la resolución de los datos de salida es 596.05 nV/LSB. En la Tabla 4, se observa una lista de valores convertidos con esta resolución.

$$\mathbf{LSB(V)} = \frac{\mathbf{2 \times V_{REF}}}{\mathbf{2^{24}}} \qquad (1)$$

Tabla 4. Entradas de tensión y salidas codificadas ideales.

Descripción	Entrada analógica (± 5 V)	Salida digital (24 bits)
Fondo de escala − 1 LSB	$+4.9999999$ V	0x7FFFFF
Mitad de escala + 1 LSB	$+596.05$ nV	0x000001
Mitad de escala	0 V	0x000000
Mitad de escala − 1 LSB	-596.05 nV	0xFFFFFF
−Fondo de escala + 1 LSB	-4.9999999 V	0x800001
−Fondo de escala	-5 V	0x800000

Modos de funcionamiento del conversor.

El conversor AD7768 presenta dos modos de funcionamiento:

1. Operación de Conversión Estándar (*Standard Conversion Operation*).

2. Operación de Conversión de Un Solo Disparo (*One-Shot Conversion Operation*).

En ambos modos de operación se realiza la conversión A/D de manera continua. La diferencia entre ambos modos radica en la transferencia de los datos. En el modo Estándar se envían los datos en flujo (*stream*) (Figura 14) y en el modo Un Solo Disparo se envía un dato cuando se recibe un pulso negativo de "sincronización" a través del pin SYNC_IN (el envío no es inmediato, existe un tiempo de establecimiento) (Figura 15).

En el modo de Operación de Conversión Estándar se generan de forma síncrona: una señal de reloj o DCLK, una señal de "dato preparado" o DRDY y uno/varios flujos de datos o DOUT. Dado que el formato a utilizar es el "una línea de dato por canal", se pueden generar hasta ocho *streams* de datos. El conversor, con el fin de indicar la transferencia completa de un dato, produce un pulso en DRDY al tiempo que envía el último bit de dicho dato (bit menos significativo). Cada bit del dato ocupa un ciclo de DCLK y, por tanto, se envía un dato completo cada 32 ciclos de este reloj (Figura 14).

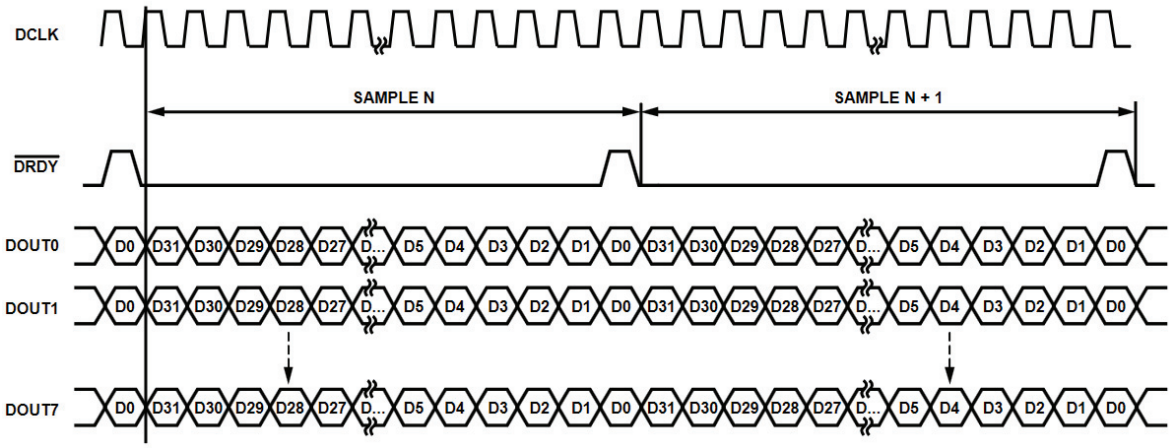

Figura 14. Salida de datos en Operación de Conversión Estándar y en FORMATO 00.

El modo de Operación de Un Solo Disparo también tiene un funcionamiento síncrono: DCLK, DRDY y DOUT. Sin embargo, en lugar de enviar un *stream* de datos, el conversor envía un único dato tras recibir un pulso negativo a través del pin SYNC_IN, lo cual significa que el A/D se comporta como un pseudo-esclavo. La transmisión del dato ocupa 32 ciclos de DCLK y, durante estos, DRDY se establece a nivel alto (Figura 15). Debido a la espera periódica de la señal de sincronización, este método reduce la velocidad de muestreo del conversor.

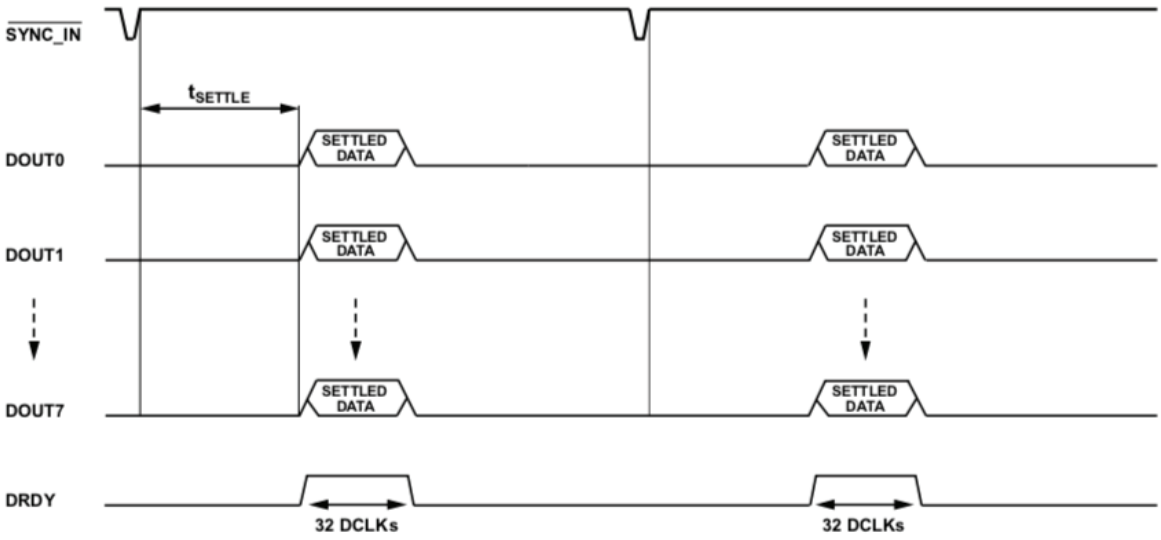

Figura 15. Salida de datos en Operación de Conversión de Un Solo Disparo y en FORMATO 00 [26].

Frecuencia de muestreo y salida de datos

Por recomendación del fabricante, el conversor AD7768 requiere de un reloj maestro (MCLK) de 32.768MHz para la conversión A/D. Esta señal de reloj es suministrada por un oscilador de cuarzo integrado en la tarjeta de adaptación.

El A/D cuenta con tres frecuencias de muestreo o modos de *power* que dependen de MCLK (Tabla 5). Además, dispone de cuatro frecuencias de salida de datos (DCLK) que, también, dependen del MCLK (Tabla 5). No obstante, la elección de esta última está limitada por la ecuación (**2**), la cual indica la frecuencia DCLK mínima requerida dependiendo de la frecuencia de muestreo y "número de canales por cada línea de datos" elegidos:

$$DCLK\ (minimum) = Output\ Data\ Rate \times Channels\ per\ DOUTx \times 32 \qquad (2)$$

Tabla 5. Frecuencias de muestreo o modos de power y salida de datos.

Modos de *Power*				**DCLK**
Modo	**Divisor**	**Velocidad de muestreo (MCLK = 32.768 MHz)**		
Fast Power	MCLK/4	256 kSP/s (8.192 MHz)		MCLK/1 MCLK/2 MCLK/4 MCLK/8
Median Power	MCLK/8	128 kSP/s (4.096 MHz)		
Low Power	MCLK/32	32 kSP/s (1.024 MHz)		

Verificación de Redundancia Cíclica o CRC

De forma optativa, el A/D AD7768 realiza una verificación de redundancia cíclica o CRC de los datos digitalizados. Esta comprobación la puede efectuar cada 4 o 16 datos digitalizados, lo cual permite detectar cambios bruscos o accidentales en la información transferida. Esta funcionalidad es únicamente configurable a través del protocolo SPI.

El CRC se envía en la cuarta o dieciseisava cabecera (8 bits más significativos). En la Figura 16, se observa la transmisión del CRC cada 4 datos digitalizados.

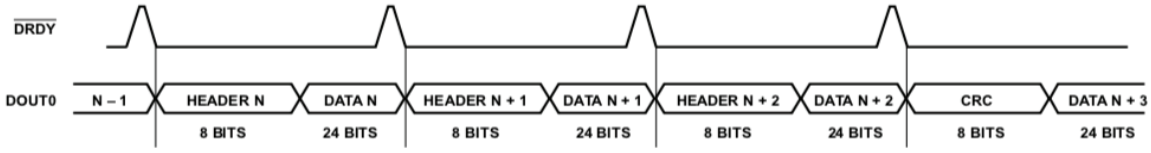

Figura 16. Envío del CRC cada 4 datos digitalizados [26].

Sincronización de los AD7768

Con el fin de realizar la conversión A/D de hasta 16 señales débiles por tarjeta, se han integrado dos conversores AD7768 conectados de forma que pueden sincronizarse en la digitalización de los distintos canales de captura (Figura 17).

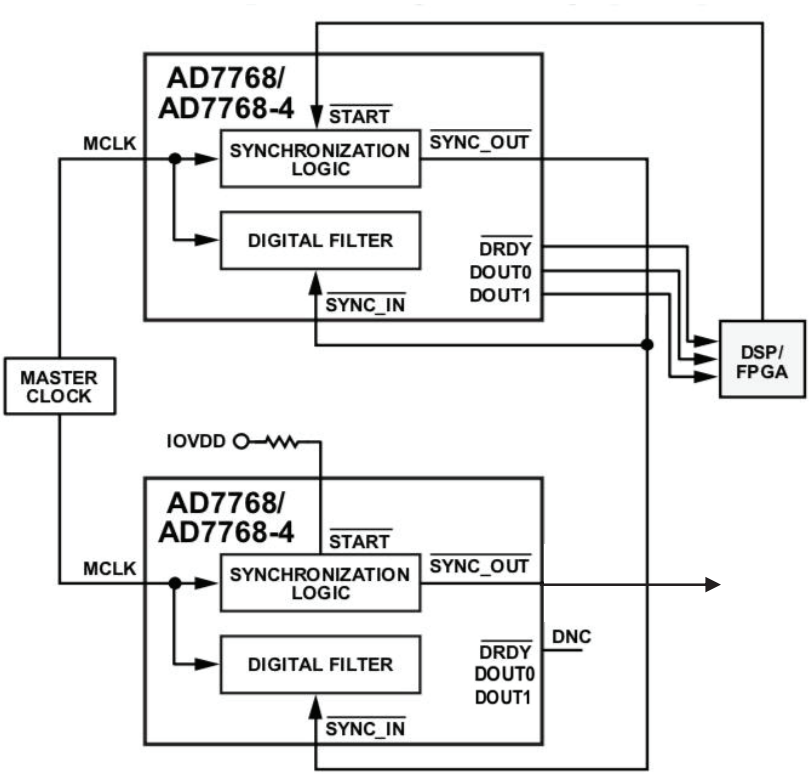

Figura 17. Sincronización de dos conversores AD7768 (adaptada de [26]).

Este conexionado de los ADC permite la sincronización en el muestreo con la segunda tarjeta y, por ende, obtener un sistema de hasta 32 canales de captura simultáneos. De igual forma, esto se

puede extender hacia otros módulos (más de 32 canales de captura). Tan solo es necesario que un conversor maestro genere la señal de sincronización ($\overline{\text{SYNC\_OUT}}$) hacia el resto de ADC ($\overline{\text{SYNC\_IN}}$) para sincronizar la captura entre tarjetas y/o módulos.

Esta señal de sincronización es generada por el ADC maestro tras recibir un pulso negativo a través del pin $\overline{\text{START}}$ o por medio del registro SPI_SYNC del protocolo SPI.

Por esta razón, la tarjeta dispone de una entrada denominada *master/slave* o M/S, que permite indicar si el primer conversor de la placa se comporta como maestro (0) o como esclavo (1), y una entrada *start* conectada al pin $\overline{\text{START}}$ del primer conversor. Ambas entradas se controlan a través del conector FMC.

La placa que contenga el ADC maestro es la placa denominada "maestra" y, se encarga de generar la señal de reloj (MCLK) hacia el resto de las placas del sistema. Por este motivo, cada tarjeta cuenta con una entrada MCLK_IN y una salida MCLK_OUT. Además, cada tarjeta presenta una entrada SYNC_IN y una salida SYNC_OUT, conectada según la configuración de la Figura 17, que permite la sincronización entre tarjetas y/o módulos (Figura 18). En esta figura, se observa cómo la tarjeta de la parte inferior es la maestra y, la de la parte superior, la esclava.

Figura 18. Frontal del módulo. Sincronización de tarjetas/módulos.

2.3.2.2. Configuración del ADC AD7768 de 24 bits.

El conversor AD7768 puede ser configurado a través de pines de entrada dedicados o mediante el protocolo SPI. La selección del modo de configuración se realiza por medido del pin $\overline{\text{PIN}}$/SPI del conversor. La tarjeta cuenta con una entrada $\overline{\text{PIN}}$/SPI, conectada a los pines $\overline{\text{PIN}}$/SPI de ambos conversores, que es accesible a través del conector FMC. Esto significa que ambos ADCs serán configurados en modo PIN o en modo SPI.

Configuración por PIN

La configuración del ADC por pines se define fijando la entrada $\overline{\text{PIN}}$/SPI a cero. Este modo de configuración permite un ajuste básico del conversor.

Por modo PIN se puede configurar las siguientes funcionalidades (Figura 20):

- Canales en *standby* o no operativos.
- Tipo de filtro. Este es común para los ocho canales.
- Tasa de decimación.
- Formato de salida de datos.
- Modos de configuración. Estos modos permiten elegir el modo de *power*, la frecuencia de DCLK y la operación de conversión (Figura 19).

MODE3	MODE2	MODE1	MODE0	Power Mode	DCLK Frequency	Data Conversion
0	0	0	0	Low power	MCLK/1	Standard
0	0	0	1	Low power	MCLK/2	Standard
0	0	1	0	Low power	MCLK/4	Standard
0	0	1	1	Low power	MCLK/8	Standard
0	1	0	0	Median	MCLK/1	Standard
0	1	0	1	Median	MCLK/2	Standard
0	1	1	0	Median	MCLK/4	Standard
0	1	1	1	Median	MCLK/8	Standard
1	0	0	0	Fast	MCLK/1	Standard
1	0	0	1	Fast	MCLK/2	Standard
1	0	1	0	Fast	MCLK/4	Standard
1	0	1	1	Fast	MCLK/8	Standard
1	1	0	0	Low power	MCLK/1	One-shot
1	1	0	1	Median	MCLK/1	One-shot
1	1	1	0	Fast	MCLK/2	One-shot
1	1	1	1	Fast	MCLK/1	One-shot

Figura 19. Modos de configuración [26].

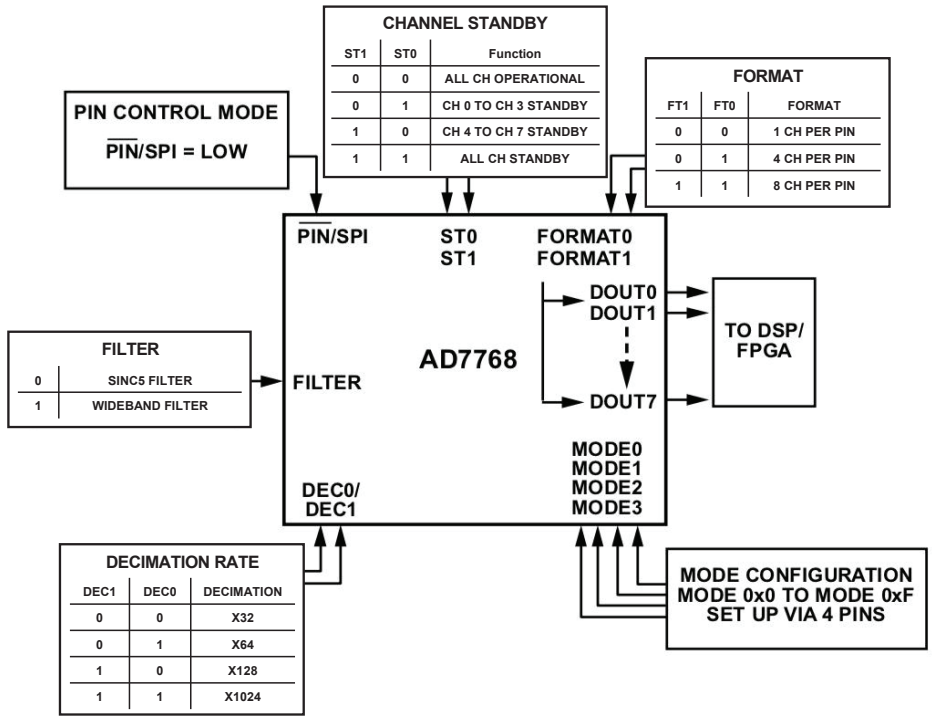

Figura 20. Configuración del AD7768 por pines (adaptada de [26]).

En la tarjeta de adaptación, los conversores comparten los pines: MODE0, MODE1, MODE2, MODE3, FILTER y, como antes mencionado, $\overline{\text{PIN}}$/SPI. Mientras que, los pines: ST y DEC se mantienen independientes para cada ADC: ST00, ST01, ST10, ST11, DEC00, DEC01, DEC10 y DEC11. Estos pines son accesibles a través del conector FMC de la tarjeta.

Con esta distribución de pines, al configurar por PIN, ambos ADCs convertirán a la misma velocidad de muestreo, transmitirán los datos a la misma frecuencia, convertirán en modo Estándar o Un Solo Disparo y utilizarán el mismo tipo de filtro. Además, permite elegir para cada ADC los canales en *standby* y la tasa de decimación.

Configuración por SPI.

La configuración del ADC a través del protocolo SPI se define fijando la entrada $\overline{\text{PIN}}$/SPI a uno (3.3 V). A diferencia del anterior modo, este permite una configuración completa del conversor y precisa de una interfaz de comunicación de tipo SPI. Este protocolo serial síncrono está conformado por una señal de reloj (SCLK), una señal de entrada de datos (SDI), una señal de salida de datos (SDO) y una señal de selección de dispositivo o esclavo ($\overline{\text{CS}}$) (Figura 21).

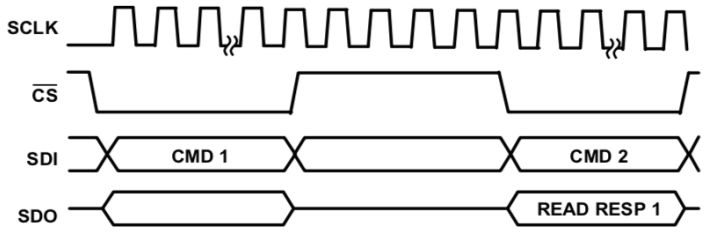

Figura 21. Protocolo de comunicación SPI [26].

Cada AD7768 dispone de estas entradas y salidas en su interfaz de comunicación y se comunica siguiendo el estándar SPI. La tarjeta que integra ambos ADCs presenta las entradas y salidas: SCLK, SDI y SDO y $\overline{\text{CS}}$ (una para cada AD7768), las cuales están conectadas al conector FMC. En la hoja de características del conversor se listan los registros de configuración SPI [26].

2.3.3. CONECTOR *FPGA MEZZANINE CARD* (FMC)

Cada placa *AD7768 AMP DIFF 16 CH* integra un conector *FPGA Mezzanine Card* (FMC) de tipo *Low Pin Count* (LPC) del estándar ANSI/VITA 57.1, que presenta 68 señales *single-ended* o 34 señales diferenciales [33].

En la Figura 22, se muestra la distribución de los pines de control de la UPS y configuración de la tarjeta *AD7768 AMP DIFF 16 CH*, así como salida de las señales digitales, en el conector FMC.

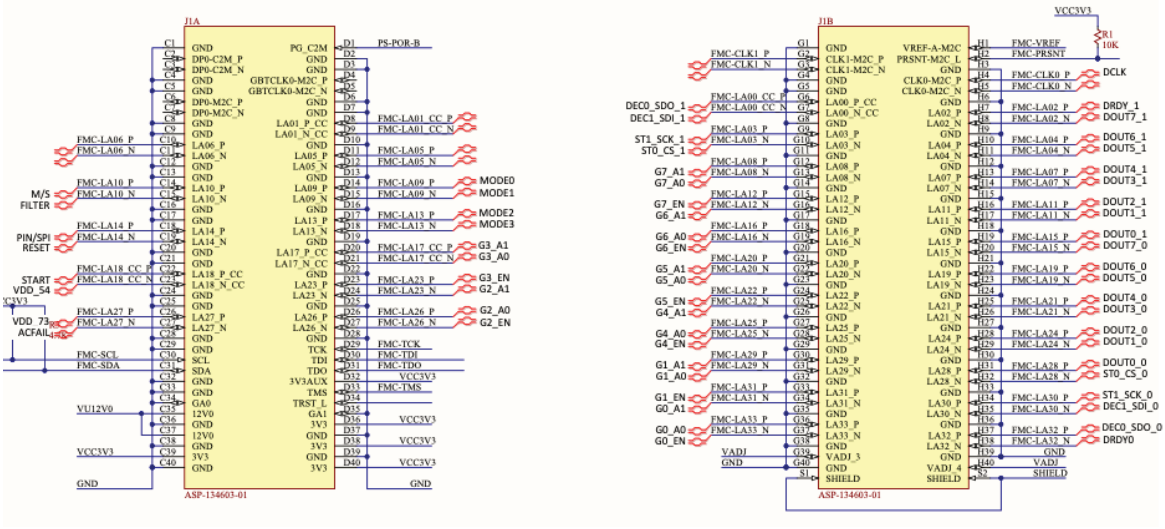

Figura 22. Distribución de E/S en el conector FMC.

2.4. Tarjeta *ZedBoard*

La tarjeta *ZedbBoard* es la pieza clave del presente Trabajo de Fin de Máster, ya que incorpora los elementos necesarios para la implementación de la lógica del sistema.

El módulo incluye dos placas FPGA *ZedBoard* que integran el dispositivo Xilinx Zynq-7000 (Zynq XC7Z020-1CLG484C) [28]. Esta placa de prototipado ha sido elegida por su bajo coste y por sus altas prestaciones, así como por integrar características de interés para el DAS [28], [34] (Figura 23), entre las cuales se destacan las siguientes:

- Conector *FPGA Mezzanine Card* (FMC) de tipo *Low Pin Count* (LPC) que conecta directamente con la Lógica Programable, soporta transferencias de hasta 2 Gbps y diferentes niveles de tensión (Figura 24).
- Múltiples memorias: DDR3 de 512 MB, Flash QuadSPI de 256 MB, SD de 4 GB y *On-Chip Memory* (OCM) de 256 KB distribuida en cuatro secciones de 64 KB.
- Un puerto Ethernet RJ45 que admite transferencias de 10 Mbps, 100 Mbps o 1 Gbps.
- Un puerto USB-JTAG que permite programar y depurar el diseño SoC.
- Cinco conectores PMOD.

En el módulo, cada *ZedBoard* está alimentada con una tensión de 9.7 Vcc (generada por la propia fuente de alimentación del módulo), está conectada, por medio del conector FMC que integra, a una placa *AD7768 AMP DIFF 16 CH* y tiene los puertos Ethernet y USB-JTAG externalizados hacia la parte trasera del módulo.

Como se ha comentado anteriormente, la tarjeta *ZedBoard* incluye un dispositivo Zynq-7000 (característica decisiva para su elección). Este posee una capacidad de procesamiento razonable (dos núcleos ARM *Cortex-A9* de 866 MHz), alta capacidad de integración, lo que permite implementar un *System-on-Chip* reconfigurable de alto rendimiento y de bajo consumo, y soporte de flujo de diseño tanto en alto nivel (C/C++/SystemC) como a nivel RTL (Verilog/VHDL) a través de herramientas de desarrollo propias de Xilinx (Vivado HLS y Vivado). Además, posee una gran variedad de recursos lógicos disponibles en la Lógica Programable que permiten la optimización eficiente del SoC (LUT, DSP, bloques de Propiedad Intelectual verificados, interfaces periféricas, etc.). Todo ello, hace que esta *ZedBoard* sea un dispositivo atractivo en cuanto a la relación precio-prestaciones.

En los siguientes apartados, se describe la arquitectura de este SoC. Su estudio es un aspecto tecnológico esencial en el presente Trabajo, puesto que permite explicar la solución arquitectural elegida. Se trata de un *System-on-Chip* profundamente complejo (ver el *Technical Reference Manual* o TRM [35]) y, por tanto, una descripción detallada se sale del alcance del proyecto.

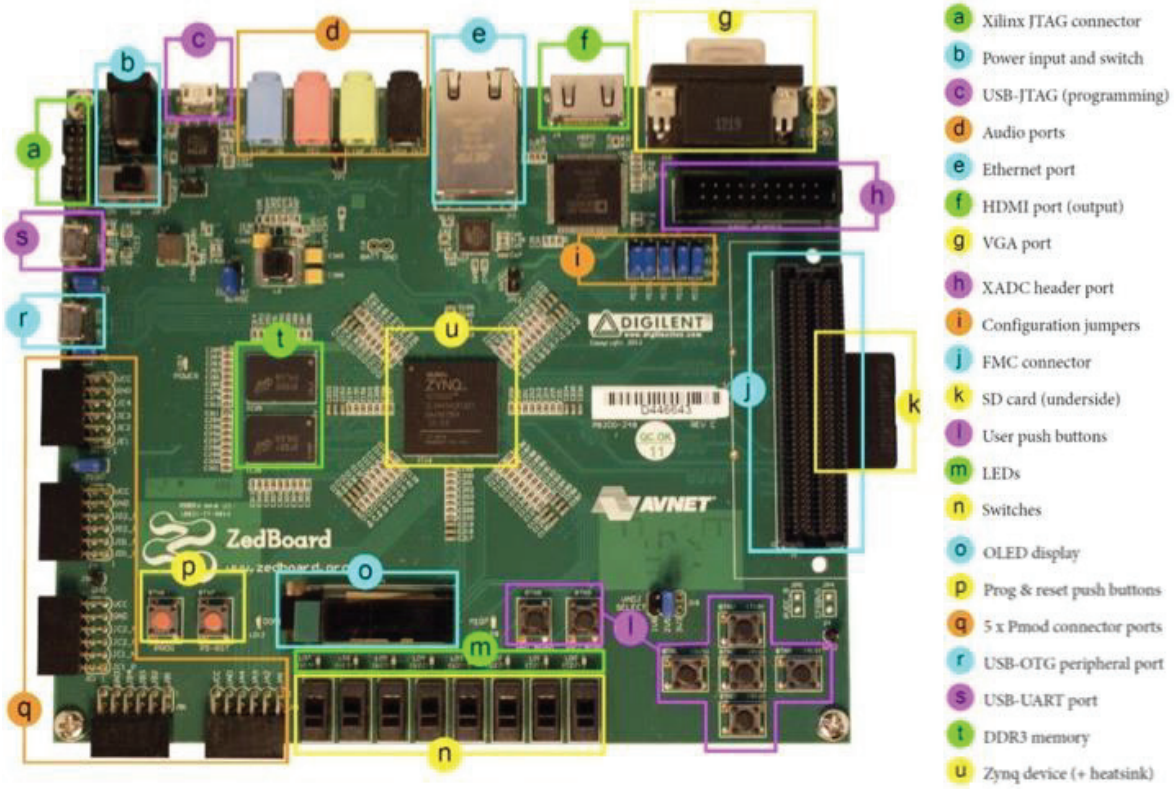

Figura 23. ZedBoard Zynq-7000 (adaptada de [34]).

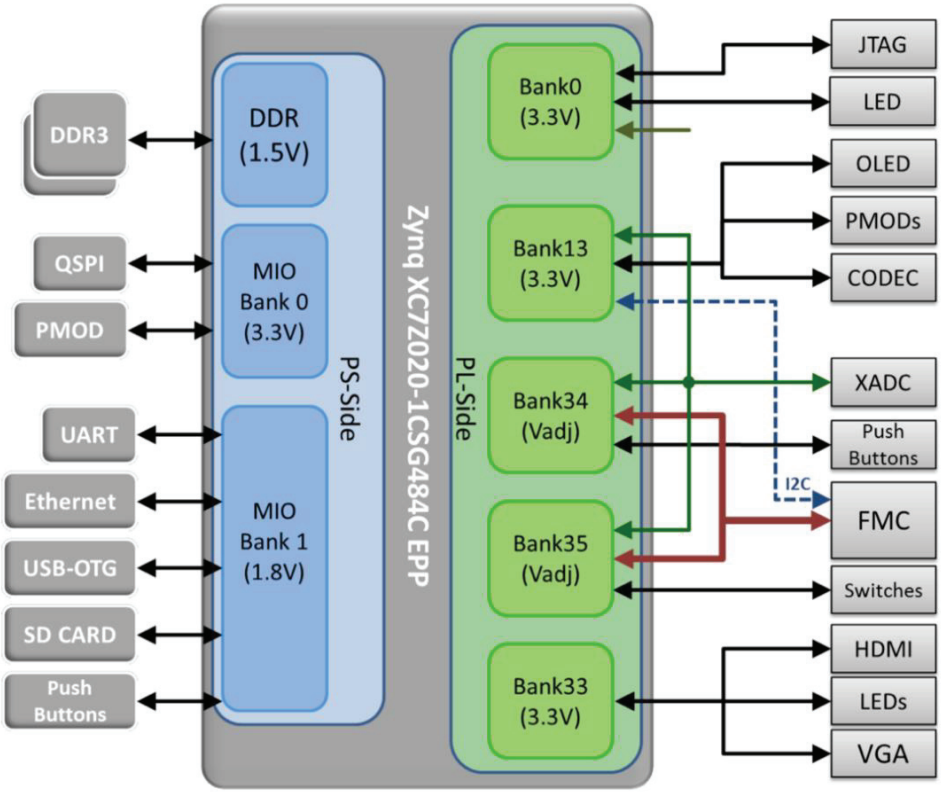

Figura 24. Asignación de bancos de E/S del dispositivo Zynq Z7020 [34].

2.5. Arquitectura del dispositivo Zynq-7000

La arquitectura del dispositivo Zynq-7000 está conformada por un Sistema de Procesamiento (*Processing System* o PS), que ejecuta aplicaciones *software*, que pueden incluir un sistema operativo, y por una Lógica Programable (*Programmable Logic* o PL), que permite implementar subsistemas de flujo de datos, aritmética y lógica de alta velocidad en el dominio *hardware* (Figura 25).

El Sistema de Procesamiento y la Lógica Programable pueden utilizarse de forma independiente. No obstante, la combinación PS-PL ofrece las virtudes de ambas partes: la flexibilidad de las soluciones *software* y las altas prestaciones alcanzadas a través del uso del paralelismo intrínseco del dominio *hardware*. Una integración PS-PL efectiva, requiere que ambos sistemas se comuniquen por medio de buses de alto rendimiento. En este caso, el Zynq-7000 utiliza buses AXI-4 (Interfaz Extensible Avanzada 4 o AXI-4) [36].

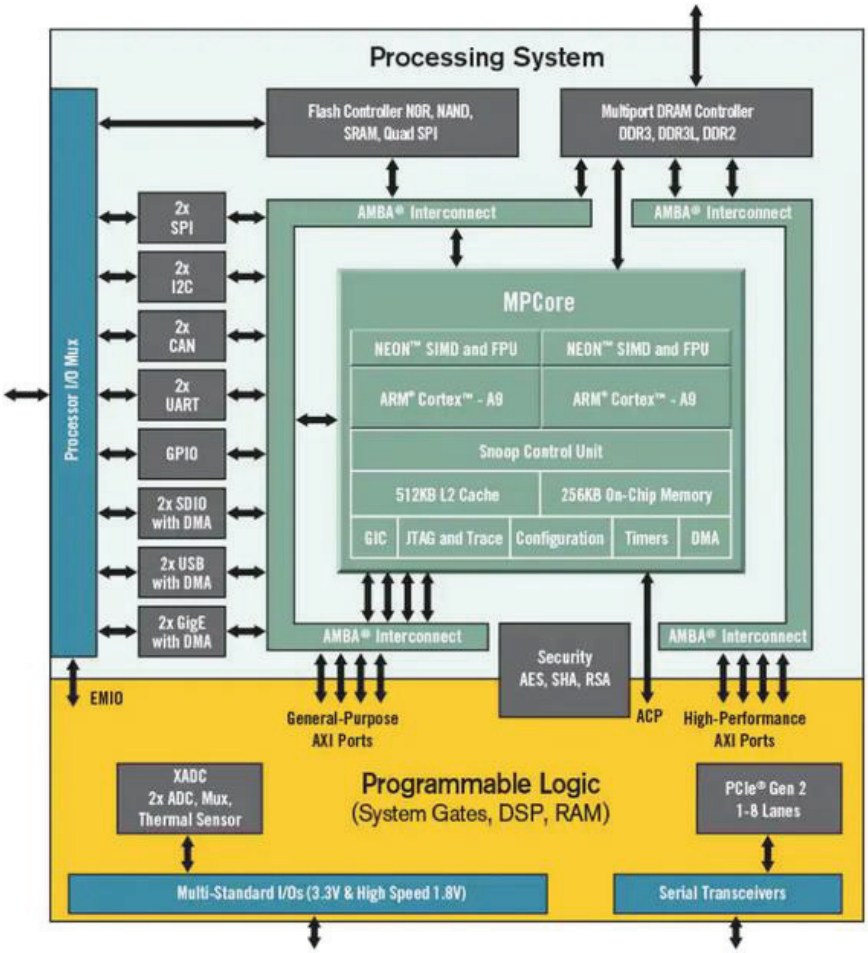

Figura 25. Esquema de la arquitectura del chip Zynq-7000 [37].

2.5.1. SISTEMA DE PROCESAMIENTO (PS)

El Sistema de Procesamiento del dispositivo Zynq-7000, a grandes rasgos, está compuesto por una Unidad de Procesamiento de Aplicaciones (APU), memorias *caches* y *On-Chip*, buses de interconexión ARM AMBA AXI, interfaces periféricas y de memoria externa, un controlador DMA de 8 canales y circuitos de generación de señales de reloj [28].

2.5.1.1. Unidad de Procesamiento de Aplicaciones (APU)

La APU del Zynq XC7Z020 es un multi-procesador que integra:

- Dos núcleos ARM Cortex-A9, que operan hasta a una frecuencia de 866 MHz.
- Dos Motores de Procesamiento de Multimedia (*Media Processing Engine* o MPE) NEON, que aplica el modelo de computación *Single Instruction Multiple Data* (SIMD) para acelerar los algoritmos DSP.
- Dos Unidades de Punto Flotante (*Floating Point Unit* o FPU).
- Una Unidad de Gestión de la Memoria (*Memory Management Unit* o MMU).

- Memorias *caches* L1 de 32 KB (almacenan instrucciones y datos), L2 de 512 KB (compartida entre los Cortex A9) y *On-Chip* de 256 KB (RAM).
- Una Unidad de Control de *Snoop* (SCU), que mantiene la coherencia entre las memorias *caches* L1 y L2, gestiona el acceso de ambos núcleos a la *cache* L2, etc [28].

Esta APU puede ejecutar tanto soluciones *standalone (software* empotrado) como soluciones basadas en la arquitectura ARM-32 con soporte de SO (ej.: Linux), desarrolladas a través del Entorno de Desarrollo *Software Vitis* de Xilinx.

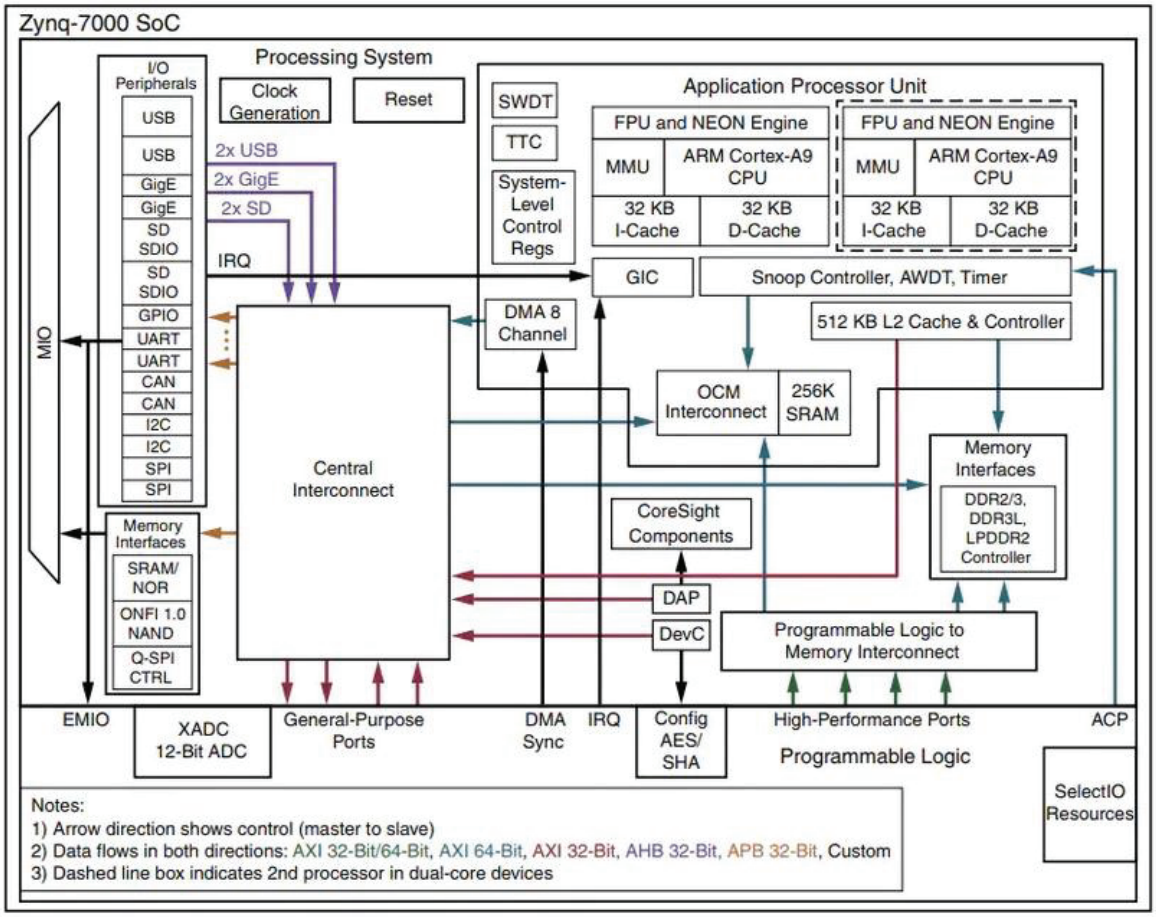

Figura 26. Diagrama de bloques del chip Zynq-7000 (editada [28]).

2.5.1.2. Interfaces de Comunicación Periféricas.

El Zynq-7000 cuenta con distintas interfaces de comunicación periféricas estándar, las cuales se listan en la Tabla 6. Estas interfaces utilizan un Multiplexor de Entrada/Salida (MIO) para la comunicación PS-componentes externos y un Multiplexor de Entrada/Salida Extendido (EMIO) para la comunicación PL-componentes externos. Estas entradas y salidas soportan las tensiones: 1.8 V, 2.5 V y 3.3 V [34].

Tabla 6. Interfaces de comunicación periféricas [36].

Interfaces periféricas	Siglas	Unidades
Serial Peripheral Interface.	SPI	2
Inter-Integrated Circuit.	IIC	2
Controller Area Network.	CAN	2
Universal Asynchronous Receiver-Transmitter	UART	2
General Purpose Input Output	GPIO	54 pines MIO 64 pines EMIO
Secure Digital Input Output	SDIO	2
Universal Serial Bus	USB	2
Gigabit Ethernet	GigE	2

2.5.2. LÓGICA PROGRAMABLE (PL)

La Lógica Programable del Zynq XC7Z020 es equivalente a una FPGA Artix-7 [36]. Está compuesta por:

1. Bloques Lógicos Configurables (*Configurable Logic Block* o CLB). Conjunto de elementos lógicos dispuestos sobre una matriz bidimensional y conectados a otros recursos lógicos similares por medio de interconexiones reprogramables. Cada CLB está compuesto por dos *slices* y colocado junto a una matriz de conmutación (*switch matrix*).
2. *Slice*. Unidad conformada por distintos recursos que permiten implementar circuitos lógicos combinacionales y secuenciales. Cada *slice* está compuesto por 4 *Lookup Table*, 8 *Flip-Flop* y otros elementos lógicos.
3. *Lookup Table* (LUT). Recurso lógico flexible capaz de implementar (i) funciones lógicas de hasta seis entradas, (ii) Memorias ROM, (iii) Memorias RAM, y (iv) registros de desplazamiento. Estas LUTs pueden ser combinadas para obtener funciones lógicas, memorias y registros de desplazamiento de mayor capacidad.
4. *Flip-Flop* (FF). Elemento lógico secuencial que implementa la funcionalidad de un registro de 1 bit.
5. Matrices de Conmutación *(Switch Matrix)*. Recurso lógico que facilita el enrutamiento (i) entre los elementos de un CLB y (ii) entre un CLB y otros elementos de la PL.
6. Bloques de Entrada o Salida (*Input/Output Blocks* o IOB). Proporcionan una interfaz entre los elementos lógicos de la PL y los circuitos externos a la FPGA.
7. Recursos de propósito especial:
    - DSP48E1s. *Slices* especializados en aritmética de alta velocidad con longitudes de palabra media y/o larga. Están compuestos por una unidad de pre-cálculo (sumador

o sustractor), un multiplicador con capacidad de segmentación, una unidad de post-cálculo (sumador o sustractor) y una unidad de comparación de patrones.

- Bloques de RAM (BRAM). Facilitan la implementación de Memorias RAM, Memorias ROM y FIFOs. Estos BRAMs se emplean para almacenar los datos de forma local en el *hardware*.

En la Tabla 7, se muestran los recursos lógicos disponibles en la Zynq XC7Z020.

Tabla 7. Recursos lógicos de la Zynq XC7Z020 [28], [36].

Recursos lógicos de la Zynq XC7Z020	
Lógica Programable equivalente	FPGA Artix-7
Bloques Lógicos Configurables (CLB)	85K
Lookup Table (LUT)	53200
Flip-Flop (FF)	106400
Bloques RAM (BRAM) de 36Kb	140 (4.9 Mb)
DSP48E1 *slices*	220

2.5.3. ARQUITECTURA DE INTERCONEXIÓN PS Y PL

La combinación del Sistema de Procesamiento (PS) y la Lógica Programable (PL) es el modelo de uso más eficaz de la Zynq. Por este motivo, es importante tanto el análisis de cada subsistema (PS y PL) como la interconexión entre ambos [36].

La principal interconexión PS-PL está compuesta por nueve interfaces AMBA AXI: dos interfaces maestras de propósito general (M_AXI_GPx), dos interfaces esclavas de propósito general (S_AXI_GPx), un *Accelerator Coherency Port* (S_AXI_ACP) y cuatro interfaces esclavas de alto rendimiento (S_AXI_HPx). A continuación, se exponen sus características:

a. Interfaz AXI de Propósito General (AXI_GP). Bus de datos de 32 bits de baja o media velocidad entre el Sistema de Procesamiento y la Lógica Programable. Es una interfaz directa y sin *bufferizado*.

b. *Accelerator Coherency Port* (ACP). Conexión asíncrona simple entre la PL y la Unidad de Control de *Snoop* (SCU) dentro de la APU con un ancho de 64 bits. Se emplea para lograr coherencia entre las memorias *cache* (L1 y L2) de la APU y los elementos de la PL.

c. Interfaz AXI de Alto Rendimiento (AXI_HP). Bus de datos de 32 o 64 bits de alta velocidad entre el Sistema de Procesamiento (esclavo) y la Lógica Programable (maestra). Es una interfaz que emplea *buffers* FIFO para la lectura y escritura en "ráfaga" o *stream* y se emplean

para el Acceso Directo a Memoria externa (DDR) u *On-Chip* (*On-Chip Memory* u OCM) (Figura 27).

Otras señales que comunican el PS y la PL son: temporizadores *watchdog*, señales de *reset*, interrupciones del *hardware*, señales de reloj, una interfaz JTAG, Entradas o Salidas Multiplexadas Extendidas (EMIO), una interfaz XADC, etc.

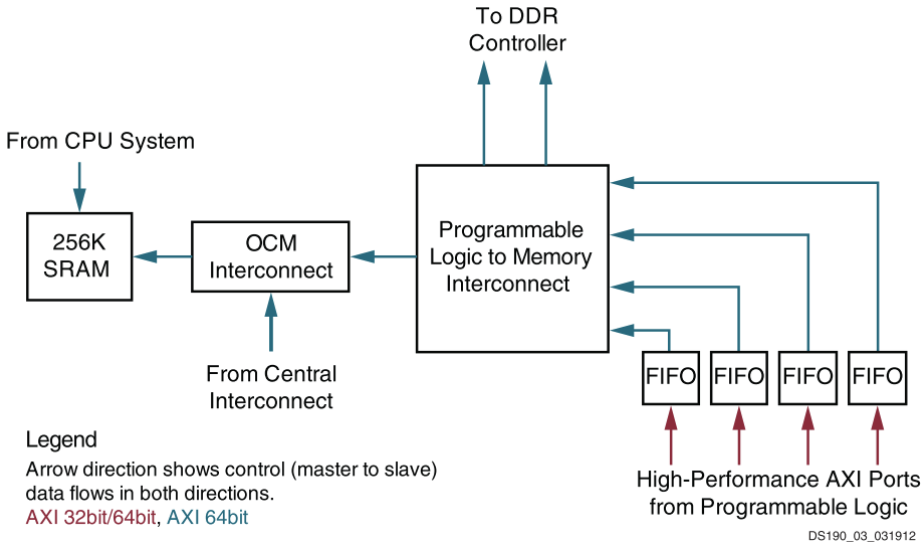

Figura 27. Interfaz AXI de Alto Rendimiento (PL y subsistema de memoria del PS) [28].

2.5.3.1. Interfaz Extensible Avanzada AMBA (AMBA AXI)

Por último, dada su importancia en los diseños *System-on-Chip Zynq*, se clasifican y se describen las interfaces AMBA *Advanced eXtensible Interface* (AXI) [38]:

La interfaz Extensible Avanzada o AXI es una interfaz orientada a buses de alto rendimiento, síncrona, de alta frecuencia, multi-maestra y multi-esclava, diseñada básicamente para las comunicaciones *On-Chip*. Estas interfaces AXI están integradas en la familia de buses *On-Chip* de ARM AMBA y, su última versión, se clasifican en:

- AXI4MM: es una interfaz de alto rendimiento que permite la lectura y escritura en memoria. Transfiere hasta 256 datos indicando una única dirección.
- AXI4-Lite: es una interfaz no apta para transferencias en modo "ráfaga" o *stream*. Transfiere un único dato al indicar una dirección y, por tanto, se emplea para la lectura y escritura en registros de control o de estado.
- AXI4-Stream: es una interfaz que soporta las transferencias en modo "ráfaga" o *stream* y, por tanto, se utiliza para la transferencia de datos a alta velocidad. No requiere de indicar una dirección de memoria y el número de datos a transferir es ilimitado.

Las interfaces AXI4MM y AXI4-Lite están conformadas por cinco canales, que permiten una comunicación bidireccional (maestro-esclavo) de manera simultánea:

- Canal de dirección de lectura (AR).
- Canal de dirección de escritura (AW).
- Canal de dato de lectura (R).
- Canal de dato de escritura (W).
- Canal de respuesta de escritura (B).

Mientras que, la interfaz AXI4-Stream tan solo permite la comunicación unidireccional y, por ende, básicamente solo está conformado por:

- Canal de receptor preparado (TREADY).
- Canal de dato (TDATA).
- Canal de dato válido (TVALID).
- Canal de fin de paquete (TLAST).

2.6. Conclusiones

En este capítulo, se han estudiado en profundidad las características que permiten el control y configuración del módulo DAS. Dicho con otras palabras, se ha estudiado el control de la UPS, la supervisión del estado de la fuente externa (AC *fail*) y el modo de actuar en caso de caída de la red, la selección de las ganancias de amplificación de las señales de entrada, el funcionamiento y configuración del conversor AD7768, la sincronización de múltiples AD7768, la distribución de las señales en el conector FMC y las principales características del dispositivo Zynq-7000 y la placa de prototipado *ZedBoard*.

Con este análisis del módulo DAS, se concluye que el SoC FPGA a diseñar debe recopilar los siguientes requisitos relativos al equipo:

1. Con respecto a la UPS, el *System-on-Chip* debe:
    a. Controlar el encendido y apagado de la etapa analógica (7.3 V) y digital (5.4 V).
    b. Supervisar el estado del suministro eléctrico externo (AC *fail*).
    c. Apagar la etapa digital (5.4 V) y, después, la etapa analógica (7.3 V), en caso de corte de la red externa (AC *fail*).
2. El SoC FPGA debe controlar las señales de selección de ganancias (ocho grupos de señales: *EN*, *A0* y *A1*).
3. Con respecto a la conversión analógica-digital, el *System-on-Chip* debe:

a. Configurar por modo PIN o SPI ambos conversores AD7768.

b. Definir el conversor *maestro* de la Tarjeta de Amplificación y Conversión A/D.

c. Sincronizar por modo PIN o SPI ambos conversores AD7768.

d. Configurar ambos conversores AD7768 con una velocidad de muestreo de 256 kSP/s, con un DCLK de 8.192 MHz (MCLK/4) y en modo conversión continua (*Standard Operation*). Esta velocidad de muestreo atiende a las especificaciones y esta frecuencia de DCLK reduce las derivas de *offsets* de los conversores.

4. Con respecto al procesamiento digital de las señales de entrada, el SoC debe:

   a. Interpretar los datos generados por los AD7768 en función de su configuración.

   b. Procesar de forma digital y simultánea dieciséis señales digitales. Esto se debe a que los AD7768 aunque un canal esté en *standby*, este genera ceros a la salida.

   c. Procesar en modo flujo, ya que la conversión de las señales es continua.

   d. Estar basado en el dispositivo Zynq-7000 (Zynq XC7Z020-1CLG484C).

5. Con respecto al control y configuración del módulo DAS, el *System-On-Chip* debe:

   a. Configurar y controlar el módulo DAS a través del conector FMC. Este es accesible desde la Lógica Programable. .

Capítulo 3. ARQUITECTURA DEL SISTEMA

En este capítulo, se expone la arquitectura del System-on-Chip que da solución al presente Trabajo de Fin de Máster, se describen las tareas de control y procesamiento, así como su distribución, a efectuar por el SoC y se especifican los IP y el software a desarrollar.

3.1. Introducción

En este capítulo, se expone la arquitectura *System-on-Chip* que ofrece solución al presente Trabajo de Fin de Máster. Por tanto, se explican los conceptos teóricos que justifican la metodología de diseño seguida, como el *diseño basado en plataformas*, se define el funcionamiento del sistema, se determinan las tareas de control y procesamiento a efectuar y se mapean las operaciones o tareas en los dominios *hardware*/*software* atendiendo a las exigencias del diseño. Además, como cierre del capítulo, se exponen los elementos *hardware* y *software* necesarios a diseñar que, integrados, dan lugar a la arquitectura final.

3.2. Diseño basado en plataformas

El estilo de diseño utilizado en el presente Trabajo de Fin de Máster está orientado a la obtención de una plataforma *hardware-software* heterogénea, aplicando metodologías de diseño de alto nivel (C++), RTL (Verilog) y técnicas de reutilización de IPs.

Al tratarse de un sistema empotrado que interactúa con un conjunto de sensores, se debe considerar tanto el desarrollo del *hardware* como del *software* de forma integrada (co-diseño *hardware-software*). Por tanto, el objetivo global es crear la partición adecuada para implementar, en cada uno de los dominios, aquella parte del sistema de forma que la combinación final resulte eficiente. Esta es la visión que se ha incluido a la hora de realizar la partición *hardware-software* del sistema.

Debido a la heterogeneidad de la solución y a su complejidad, conlleva una amplitud importante del espacio de diseño a analizar y sus implicaciones en el tiempo de desarrollo del producto (*Time-To-Market* o TTM). Por tanto, el *diseño basado en plataformas* sienta las bases de flujos de diseño económicamente viables. Se trata de una metodología estructurada, que logra resultados superiores en las limitaciones de tiempo del diseño. Igualmente establece puntos de transformación y verificación del diseño, lo que facilita la interacción de los equipos de trabajo.

De la misma manera, fomenta la reutilización del diseño en todos los niveles de abstracción (tanto en el dominio *hardware* como en el dominio *software*), lo que permite el diseño de un producto electrónico mediante la integración de los componentes de la plataforma de una manera rápida a través de una integración eficaz del entorno de diseño.

El **diseño basado en plataformas** permite alcanzar el diseño de la arquitectura final a partir de una plataforma. Según Peter Marwedel [39], se entiende como una plataforma al conjunto de arquitecturas que cumplen con un conjunto de restricciones impuestas para permitir la reutilización de componentes *hardware* y *software*. La plataforma incluye tanto la referencia *hardware* como el conjunto de APIs *software* que permiten obtener distintas arquitecturas derivadas. Por tanto, la plataforma en sí es una capa de abstracción que facilita el refinamiento del diseño en capas inferiores.

El principio básico del diseño basado en plataformas es la identificación del diseño como un proceso *meet-in-the-midle* [40], donde los refinamientos sucesivos de las especificaciones se mapean con abstracciones de implementaciones potenciales, y la identificación de capas definidas con precisión donde tienen lugar los procesos de refinamiento y abstracción.

La estrategia utilizada permite que, a partir de una plataforma inicial de granularidad gruesa, se refine y transforme hacia los detalles de la implementación final, en capas más bajas del proceso de transformación.

Cada capa admite una etapa de diseño que proporciona una abstracción de las capas inferiores que genera estimaciones precisas del rendimiento. Esta información se incorpora en parámetros apropiados que anotan las opciones de diseño en la capa actual de abstracción. Por tanto, podemos definir una plataforma como una biblioteca de componentes junto con sus reglas de composición. Un diseño en cada nivel de abstracción es una instancia de la plataforma, es decir, una composición legal de un conjunto de elementos de librería. La librería no solo contiene bloques computacionales que llevan a cabo el cálculo adecuado, sino también componentes de comunicación que se utilizan para interconectar los componentes funcionales. Con esta visión se separan los aspectos funcionales de los aspectos de comunicación entre bloques, facilitando la rápida transformación de la plataforma en diseños derivados en función de los cambios solicitados sobre el diseño de referencia.

Cada elemento de la librería tiene una caracterización en términos de parámetros de rendimiento para obtener la funcionalidad que puede soportar. Para cada nivel de plataforma, hay un conjunto de métodos utilizados para mapear las capas superiores de abstracción en la plataforma y un conjunto de métodos utilizados para estimar el rendimiento de las abstracciones de nivel inferior. El proceso *meet-in-the-midle* es la combinación de dos esfuerzos [41]:

- *Top-down*: mapear una instancia de la plataforma superior con sus restricciones en una instancia de la plataforma inferior, con restricciones propagadas y apropiadas a la implementación.
- *Bottom-up*: construir una plataforma definiendo sus componentes y sus prestaciones mapeadas en el espacio de diseño.

Cuanto más grande es el paso a través de las plataformas, más difícil es predecir el rendimiento, optimizar en los niveles más altos de abstracción y proporcionar un límite inferior ajustado. De hecho, el espacio de diseño para este enfoque puede ser en realidad más pequeño que el obtenido con pasos más pequeños porque se vuelve más difícil explorar alternativas de diseño significativas y la restricción de la búsqueda impide la exploración completa del espacio de diseño.

En la Figura 28, se muestra el proceso de diseño partiendo del espacio de aplicaciones y su mapeado y exploración del espacio del diseño hacia la instancia de la plataforma.

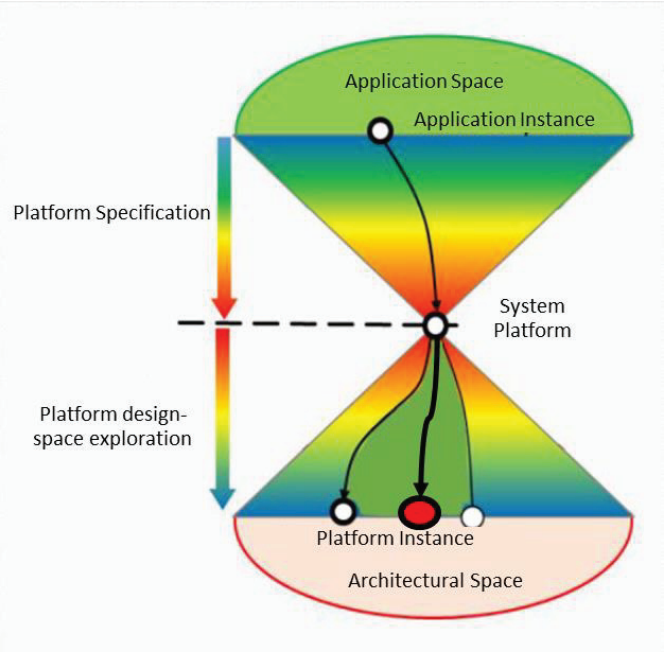

Figura 28. Proceso de diseño basado en plataforma (adaptada de [40]*).*

Por tanto, podemos deducir que el sistema final se desarrolla como una instancia de una arquitectura de la plataforma de referencia, derivada de una determinada familia específica de micro-

arquitecturas especializadas. Es decir, en lugar de ensamblarse a partir de una colección de bloques de funcionalidades desarrollados de forma independiente, se derivarán de una familia específica de micro-arquitecturas, posiblemente orientadas hacia una clase particular de problemas, que pueden ser ampliados o reducidos por el desarrollador del sistema. Los elementos de esta familia son una especie de "denominador de *hardware*" que podría compartirse entre múltiples aplicaciones. Cada elemento de la familia se puede obtener a través de la personalización de un conjunto apropiado de parámetros que controlan la micro-arquitectura.

La flexibilidad, entendida como la capacidad de soportar diferentes aplicaciones por una instancia de plataforma, está garantizada por componentes programables. La programabilidad puede considerarse como programabilidad del *software*, para indicar la presencia de un microprocesador, DSP o cualquier otro componente programable por *software*, o la programabilidad *hardware* para indicar la presencia de bloques lógicos reconfigurables, como es el caso de las FPGA. La arquitectura y la plataforma de implementación puede encontrarse en un único *chip*, que se conoce como sistema en *chip* o SoC.

Un aspecto clave en la plataforma es la reutilización del *software* de aplicación. La plataforma ve una interfaz de alto nivel al *hardware* o *Application Programm Interface* (API). Una capa de *software* se utiliza para realizar esta abstracción. Esta capa accede a partes esenciales de la plataforma, que incluye distintos *cores* y el subsistema de memoria, ya sea mediante librerías o a través de un sistema operativo (RTOS o SO empotrado), el subsistema de E/S a través de los *drivers* de dispositivos y los *drivers* de red través de la red subsistema de comunicación.

Para una implementación eficiente del sistema, en este trabajo se ha elegido como soporte del diseño basado en plataforma el SoC Zynq 7000 y el conjunto de herramientas de desarrollo de Xilinx. Desde el punto de vista conceptual, el entorno Vivado incluye el conjunto de IPs y los métodos de integración, verificación y síntesis necesarios, en distintos niveles de abstracción, incluyendo el diseño de alto nivel. Tanto los bloques existentes, como los diseñados para personalizar cada una de las instancias de la plataforma, soporta comunicaciones AMBA AXI4, lo que constituye el elemento central para facilitar su integración *hardware*/*software* al establecer un método de acceso unificado a los datos y registros. Igualmente, cada bloque/IP diseñado posee una API que facilita su programación desde un lenguaje de alto nivel (C/C++).

Los bloques diseñados se integran en la plataforma de referencia como elementos claves a partir de los que es posible crear plataformas derivadas para el diseño de sistemas DAS para distintas aplicaciones. La inclusión de nuevos bloques que amplíen la funcionalidad de la plataforma, como por ejemplo incluir un bloque que realice la FFT de las señales, es directa, aportando valor a la plataforma completa.

Como se deduce, inicialmente es necesario construir la plataforma de forma incremental a partir de los bloques disponibles y de la funcionalidad especificada. Este proceso de construcción requiere de procesos de verificación, integración *hardware*/*software* y de validación de la funcionalidad desarrollada. Por tanto, el diseño basado en plataformas facilita un desarrollo ágil del sistema, especialmente para el caso de su implementación sobre dispositivos FPGA.

3.3. Diagrama de Tareas del Sistema

Frente a los requerimientos, se diseña un sistema que ejecuta el diagrama de tareas mostrado en la Figura 29. Este comienza con una configuración inicial del módulo (ej.: encender la UPS, definir los factores de amplificación a 1 (0 dB), configurar los ADCs en modo PIN, definir que promedie cada 4 muestras, etc.); después de dicha configuración, entra en un bucle de *gestión de órdenes de control y reconfiguración del módulo* y de *supervisión de AC fail* (bucle *capture off*); y, en caso de "habilitar la captura", realiza el procesamiento de las muestras (bucle *capture on*).

Como tareas de procesamiento de las capturas, el sistema realiza las siguientes:

(i) Interpretar los datos generados por los conversores.

(ii) Añadir marcas de tiempo real a las muestras capturadas.

(iii) Promediar las muestras capturadas.

(iv) Almacenar en memoria las muestras procesadas.

(v) Guardar en ficheros de datos las muestras procesadas.

(vi) Transferir los ficheros de muestras procesadas a un servidor por Ethernet.

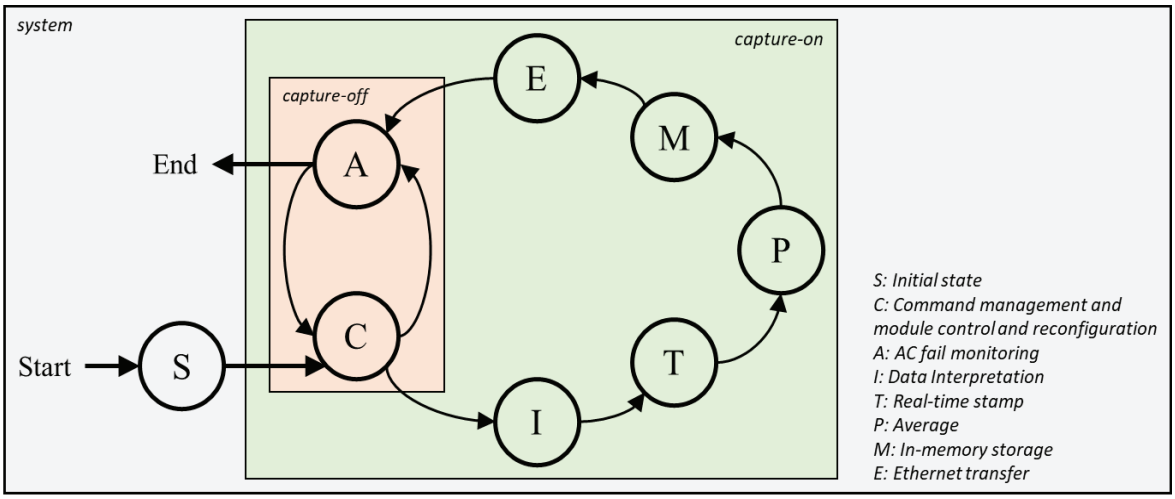

Figura 29. Diagrama de tareas del sistema propuesto.

Con este esquema, se obtiene un sistema que permite procesar las señales de entrada al módulo tal y como solicita el IAC y que permite controlar y reconfigurar el procesamiento de dichas señales a través de *órdenes de control y reconfiguración* durante su funcionamiento (ej.: habilitar o deshabilitar la adquisición y procesamiento de las muestras, modificar las ganancias de amplificación, poner en *standby* diferentes canales de captura, redefinir el número de muestras a promediar, etc.). Además, se obtiene un sistema que permite supervisar el estado de la red externa (AC *fail*) y que, por ende, en caso de fallo, permite detener el módulo de forma controlada.

3.4. Mapeado de Tareas del Sistema

El sistema expuesto realiza las siguientes tareas:

T1. Controlar y configurar el módulo DAS.

T2. Supervisar el estado de la red externa (AC *fail*).

T3. Atender a las *órdenes de control y reconfiguración del módulo*.

T4. Interpretar los datos generados por los ADCs.

T5. Añadir marcas de tiempo real a las muestras capturadas.

T6. Promediar las muestras capturadas.

T7. Almacenar en memoria las muestras procesadas.

T8. Guardar en ficheros de datos las muestras procesadas.

T9. Transferir los ficheros de muestras procesadas a un servidor por Ethernet.

Como primera aproximación y atendiendo al *diseño basado en plataformas*, se realiza el mapeado de dichas tareas en los dominios *hardware* y *software*, es decir, se diseña un *System-on-Chip* que integra una lógica reconfigurable basada en FPGA, una APU que ejecuta una aplicación *software* e interconexiones *hardware*/*software* efectivas.

En la Tabla 8, se observa la distribución de las tareas citadas. Como se puede apreciar, se distribuyen en el dominio *hardware* las tareas que requieren acceso inmediato al módulo DAS (ej.: controlar y configurar del módulo) o una implementación a alta velocidad y determinista (ej.: insertar marcas de tiempo real). Por otro lado, en el dominio *software* se mapean aquellas tareas con carácter secuencial (ej.: primero se escriben las capturas procesadas en memoria y después en ficheros) o que requieren mayor margen de flexibilidad (ej.: gestión de ficheros).

Tabla 8. Distribución de las tareas principales sobre el Zynq-7000.

Tareas	Dominio *software*	Dominio *hardware*
T1. Controlar y configurar del módulo	**X**	**X**
T2. Supervisar el estado de la red externa (AC *fail*)	**X**	**X**
T3. Atender a las *órdenes de control y reconfiguración*	**X**	
T4. Interpretar los datos generados por los ADCs		**X**
T5. Añadir marcas de tiempo real a las muestras capturadas		**X**
T6. Promediar las muestras capturadas		**X**
T7. Almacenar en memoria las muestras procesadas	**X**	**X**
T8. Guardar en ficheros de datos las muestras procesadas	**X**	
T9. Enviar ficheros de datos por Ethernet	**X**	

3.5. Arquitectura del Sistema

En la Figura 30, se muestra el diagrama de la arquitectura *System-on-Chip* FPGA que pretende cubrir los objetivos del sistema ya definido. Esta arquitectura heterogénea responde al modelo de uso eficaz del dispositivo Zynq, es decir, se acoge a las virtudes del *hardware* (aritmética y lógica de alta velocidad, paralelismo, etc.) y del *software* (alta flexibilidad, secuencialidad, etc.).

Tal y como se observa, en la Lógica Programable (diseño en el dominio *hardware*) se instancian IPs de control y configuración del módulo y de procesamiento de muestras. A continuación, se presenta la relación de dichos bloques *hardware* y su funcionalidad en el sistema:

- IP de Control de la UPS: Encender y/o apagar la UPS y supervisar la señal *AC Fail*.
- IP de Control de Ganancia: Seleccionar los factores de amplificación.
- IP de Control por PIN: Configurar y controlar por modo PIN los conversores.
- IP de Control por SPI: Configurar y controlar por modo SPI los conversores.
- IP Maestro/Esclavo: Definir el primer conversor como “maestro” o “esclavo”.
- IP Interfaz AD7768: Interpretar los datos generados por los ADCs AD7768 en función de su configuración y controlar la entrada de datos digitales al *hardware*.
- IP de Marca de Tiempo Real: Añadir marcas de tiempo real en segundos o milisegundos a las muestras capturadas.
- IP de Cálculo de Promedios: Promediar cada *n* las muestras capturadas.

- IP de Acceso Directo a Memoria: Almacenar en memoria las muestras inmediatamente después de ser procesadas y sin interferir en la actividad de los *cores*.
- IP de Control por I2C: Conocer la fecha y hora actual en segundos a través de un *Real Time Clock* (RTC) conectado a un PMOD. El Zynq-7000 carece de circuito RTC interno.
- IP de Interconexión AXI: Conectar los bloques *hardware* con el Zynq PS.

Por otro lado, en el Sistema de Procesamiento (diseño en el dominio *software*), se ejecuta una aplicación *software standalone* que permite realizar las siguientes operaciones:

- Configurar los IPs que conforman la plataforma *hardware* tanto para controlar y configurar el módulo DAS como para configurar el procesamiento de las señales digitales:
  - Configurar el IP de Control de UPS para encender o apagar la UPS.
  - Utilizar el IP de Control de UPS para supervisar el estado de la fuente externa.
  - Configurar el IP de Control de Ganancias para seleccionar las ganancias.
  - Configurar el IP de Control por PIN o SPI para configurar en modo PIN o SPI los conversores AD7768, así como para sincronizar ambos ADCs.
  - Configurar el IP Maestro o Esclavo para definir el primer conversor de la Tarjeta de Adaptación como *master* o *slave*.
  - Configurar el IP de Control I2C para conocer la fecha y hora real con resolución de segundos desde un módulo RTC.
  - Configurar el IP Interfaz AD7768 para interpretar de forma correcta las señales digitales generadas por los conversores AD7768, así como para controlar la entrada de muestras a la plataforma *hardware*.
  - Configurar el IP de Marca de Tiempo Real con la marca de tiempo real a añadir a las muestras. Esta debe ser con resolución de segundos o milisegundos. Dicha marca se puede obtener desde un módulo RTC (segundos) o servidor *Network Time Protocol* (NTP) (segundos o milisegundos).
  - Configurar el IP Cálculo de Promedios con el número de muestras a promediar.
  - Configurar el IP de Acceso Directo a Memoria para que almacene en memoria *On-Chip* las muestras procesadas.
- Guardar en ficheros de datos las muestras procesadas almacenadas en memoria mediante un Sistema de Ficheros FAT (*FAT File System* o FFS) en RAM.
- Enviar dichos ficheros de datos hacia un servidor por Ethernet a partir de un cliente.
- Leer los *comandos* recibidos por el puerto UART.

Las interfaces AXI4 AMBA del *hardware* ilustradas, se han elegido en base a sus características (Interfaz Extensible Avanzada AMBA (AMBA AXI)) y exigencias del diseño.

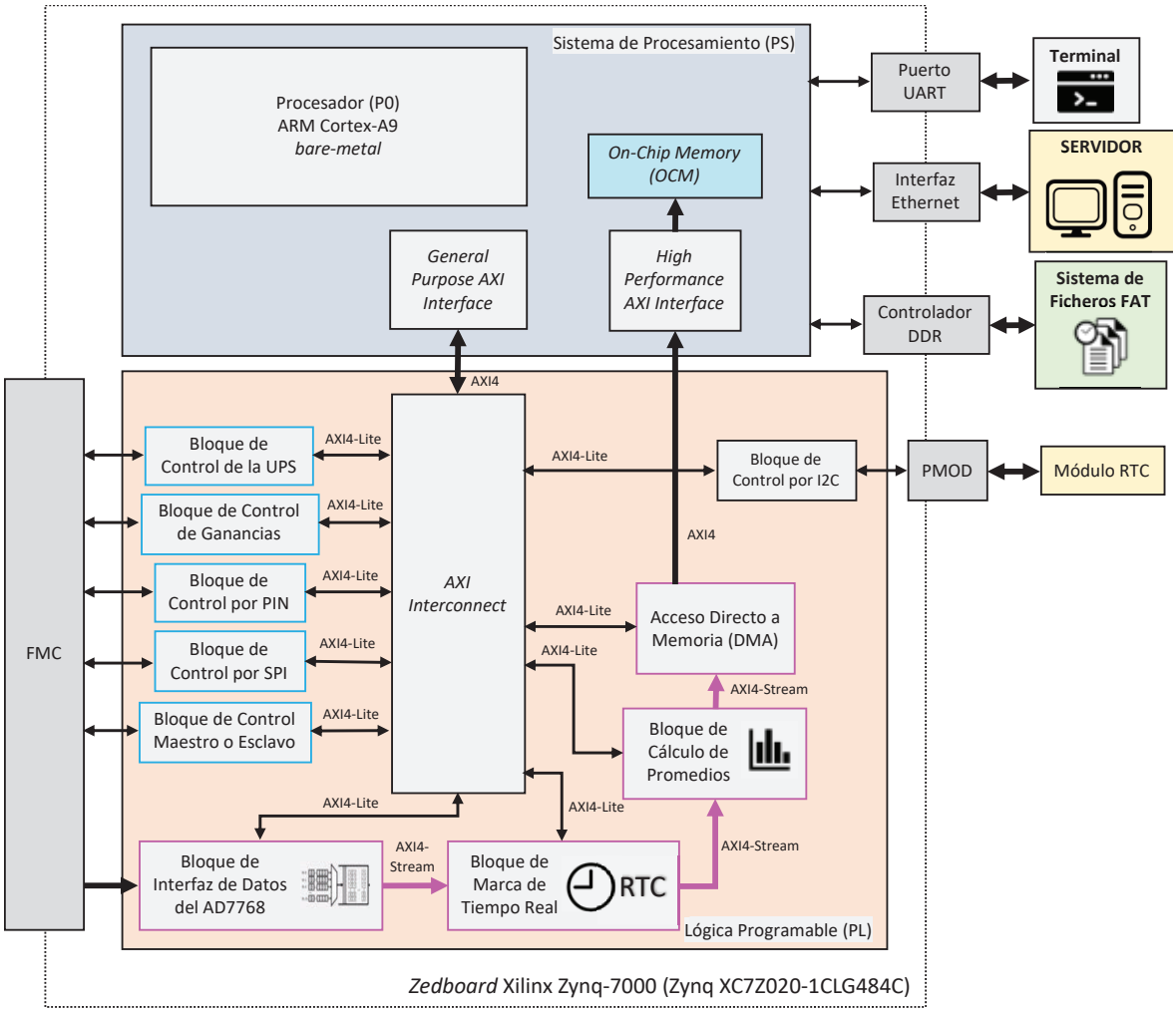

Figura 30. Diagrama de la arquitectura SoC propuesta.

3.6. Conclusión

Con esta arquitectura *System-on-Chip* se puede arrancar y configurar el módulo DAS, así como controlarlo y reconfigurarlo a través de órdenes de control y reconfiguración del módulo transferidas por la UART, supervisar el estado de la fuente externa (AC fail) y, en caso de corte, apagar el DAS de forma controlada, procesar las señales de entrada tal y como reclama el IAC, almacenar las señales procesadas en ficheros de datos y salvar dichos ficheros en un servidor remoto. Esto último, facilita el post-procesado científico de las señales.

Además, se puede conocer la fecha y hora real en segundos a partir de un *Real Time Clock* externo. Esto proporciona cierta autonomía al módulo, puesto que se pueden insertar marcas de tiempo real en segundos sin depender de un servidor *Network Time Protocol* (NTP). Asimismo, el SoC está preparado para añadir marcas de tiempo real a las muestras tanto en segundos como en milisegundos.

En los próximos apartados, se expone en detalle la solución *hardware* (Lógica Programable) y *software* (Sistema de Procesamiento) del *System On-Chip* descrito.

Capítulo 4. METODOLOGÍA DE DISEÑO

En este capítulo, se expone la metodología de diseño seguida para la elaboración de la plataforma SoC FPGA del presente Trabajo de Fin de Máster. Se explican en detalle las herramientas de desarrollo hardware y software, así como los dispositivos de instrumentación (generadores de señal y analizadores lógicos) utilizados.

4.1. Introducción

En el presente capítulo, se expone el flujo de diseño seguido para la elaboración de la plataforma *System On-Chip* FPGA que implementa la arquitectura presentada.

Esta metodología de desarrollo tiene como objetivo la obtención de un SoC FPGA empotrado, verificado y validado, apoyándose en los procedimientos y herramientas de diseño propias de Xilinx, así como otras herramientas ampliamente empleadas en entornos de desarrollo *hardware*, tales como: herramientas de simulación de Cadence y dispositivos de instrumentación de Digilent. Además, esta metodología contempla el uso de diseños ya verificados previamente por desarrolladores de Xilinx y Analog Devices.

El flujo de diseño incluye el uso de lenguajes de descripción *hardware* estandarizados (Verilog) y de programación C++ con el fin de valerse de las ventajas de desarrollo de ambos: la flexibilidad del *software* y el alto rendimiento y paralelismo del *hardware*.

A continuación, se describe la metodología de diseño utilizada, las herramientas de desarrollo y los equipos de instrumentación utilizados.

4.2. Flujo de diseño

El flujo de diseño utilizado en el presente Trabajo de Fin de Máster se muestra en la Figura 31. Este complejo esquema persigue el diseño de un SoC FPGA empotrado, verificado y validado

utilizando metodologías y herramientas de desarrollo propias de Xilinx y aprovechando recursos verificados por Xilinx y Analog Devices. Además, como herramientas de apoyo a la verificación y validación, emplea el simulador SimVision de Cadence y el multi-instrumento *Analog Discovery 2*.

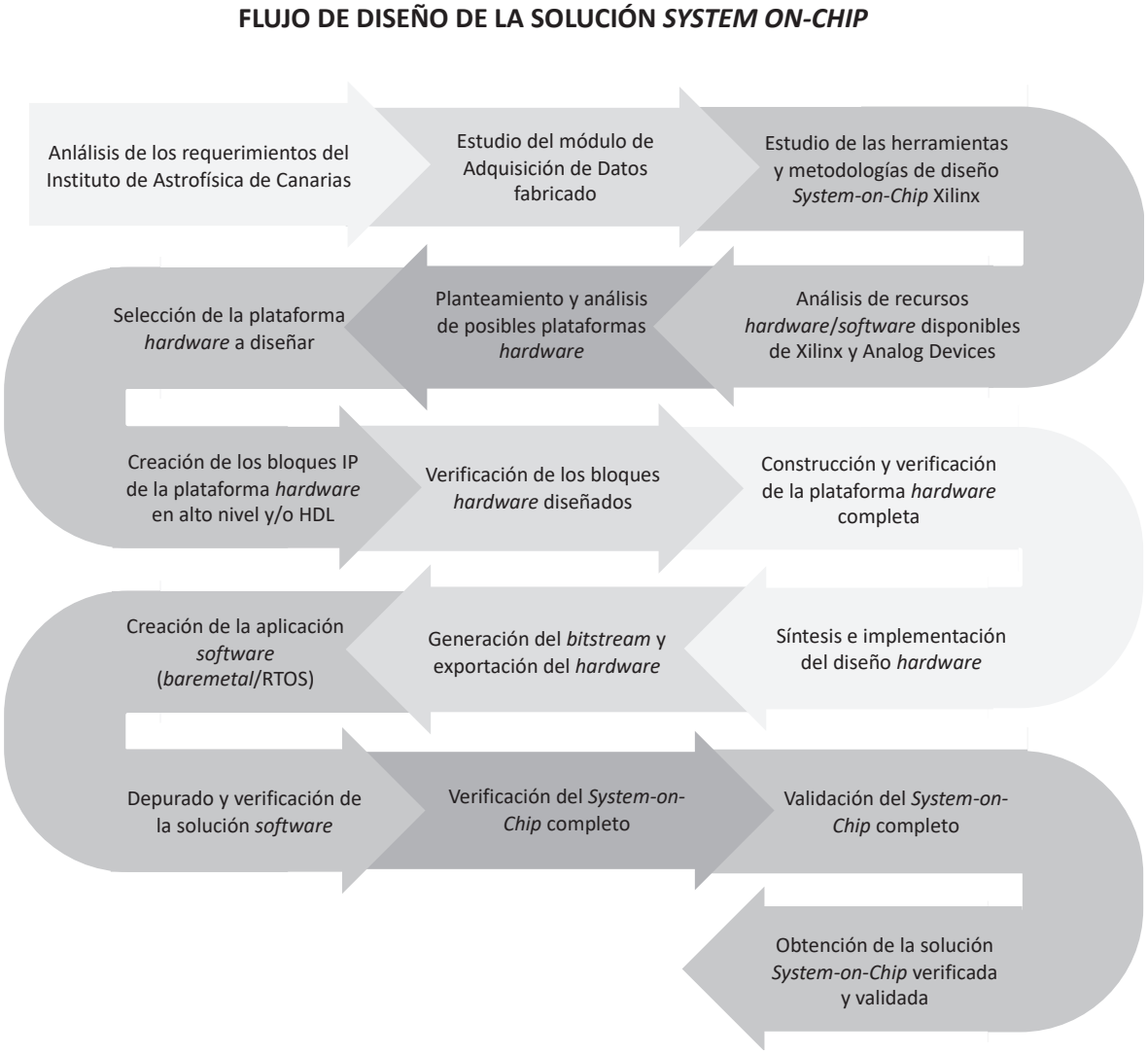

Figura 31. Flujo de diseño de la solución System on-Chip.

El diseño de la solución *System On-Chip* parte del estudio de los requisitos funcionales y de procesamiento de un Sistema de Adquisición de Datos establecidos por el Instituto de Astrofísica de Canarias. Tras este análisis, se elabora un módulo DAS que integra múltiples componentes electrónicos optimizados para la aplicación. Se prosigue con el estudio del módulo y con el análisis en detalle de dichos componentes: ADC AD7768, Zynq-7000, etc. Todo ello, define el marco de diseño de la solución SoC del presente Trabajo.

Seguidamente, se estudian las herramientas de desarrollo y los recursos disponibles, en este caso, de Xilinx y Analog Devices, tales como: Vivado HLS, Vivado, Vitis y repositorios GitHub de

Analog Devices. Conocidas las exigencias y el abanico de recursos, se plantean distintas arquitecturas que definen el espacio de soluciones *hardware* del SoC y, a través dichas herramientas, se analiza su viabilidad (escalabilidad, disponibilidad de recursos *hardware*, limitaciones, tiempos de respuesta, etc.). Se elige la plataforma *hardware* a diseñar, se crean los bloques IP en alto nivel (C++) necesarios a través de Vivado HLS y/o en lenguaje de descripción *hardware* (Verilog) por medio de Vivado, reutilizando aquellos IPs disponibles en los repositorios de Xilinx y de Analog Devices.

A continuación, se procede con la verificación de los bloques *hardware* creados por medio de bancos de pruebas y análisis de resultados utilizando el simulador SimVision de Cadence y, en caso necesario, se verifican las interfaces AXI de los IP a través de IP de Verificación de Xilinx. Se construye la plataforma *hardware* y se realiza la verificación global de la plataforma *hardware* mediante un *testbench* integrado y haciendo uso del Simulador de Vivado. Después, se definen el conjunto de restricciones a aplicar al diseño *hardware* (definir la frecuencia de los relojes del sistema, distribuir las E/S de la solución *hardware* en los pines físicos de la placa *ZedBoard*, definir las tensiones de E/S físicas, etc.) y se sintetiza e implementa el diseño sobre el Zynq. Esto proporciona informes de resultados (recursos utilizados y disponibles, consumo de potencia, *timing* del enrutamiento, etc.). En el supuesto de obtener resultados favorables en dichos *reports*, se genera el *bitstream* y se exporta el *hardware* a un fichero XSA (*Xilinx Support Archive*).

En Vitis y a partir de dicho *hardware* exportado, se crea la aplicación *software* (*baremetal*). Las partes del *software* de alta complejidad se someten a validaciones cruzadas con el *hardware* y, por tanto, se añaden Analizadores Lógicos Integrados (ILA) en los puntos de la plataforma SoC FPGA *hardware* a analizar y se verifican dichas secciones de código a través del *Hardware Manager* de Vivado durante el depurado de la aplicación. De igual forma, se verifica la integración *hardware-software* completa.

Posteriormente, se valida el *System On-Chip* completo monitorizando las E/S digitales de la *ZedBoard* e inyectando patrones digitales predefinidos con el multi-instrumento *Analog Discovery 2*. Para ello se utiliza el programa *WaveForms* de Digilent que permite configurar distintos parámetros de los patrones de entrada (frecuencia, niveles lógicos, etc.) durante el depurado del código y, desde el *software*, se imprimen por consola mensajes de “éxito” o “error” a fin de detectar fácilmente posibles errores.

Finalmente, se valida el SoC integrado en el módulo. Esto consiste en validar algunos aspectos claves del funcionamiento del DAS, como, por ejemplo, el módulo se enciende o apaga, se seleccionan los factores de amplificación, en caso de caída de la red externa se procede con el protocolo de apagado del módulo, los conversores están sincronizados y ajustados, las señales analógicas inyectadas son procesadas, etc. De nuevo, durante este proceso, desde el *software* se imprime el estado de estas acciones para identificar posibles fallas.

4.3. Herramientas de desarrollo

Tal como se indicó anteriormente, los entornos de diseño (EDA) utilizados son: *Vivado High-Level Synthesis* [42], *Vivado Integrated Design Environment* [43], *Vitis Integrated Development Environment* [44] y *Cadence SimVision Debug* [45].

4.3.1.1. Vivado High-Level Synthesis (Vivado HLS)

Vivado High-Level Synthesis o *Vivado HLS* [42] es un entorno de síntesis de alto nivel de Xilinx que transforma una especificación algorítmica C, C++ o SystemC en una implementación a nivel *Register Transfer Level* (RTL) expresada en un Lenguaje de Descripción *Hardware* (*Hardware Description Language* o HDL), que puede ser sintetizada en una FPGA de Xilinx. Dicho de otro modo, *HLS*, a partir de una capa de abstracción de alto nivel, permite crear *hardware* de alto rendimiento, lo cual reduce el tiempo y la dificultad que supone el desarrollo *hardware* en niveles inferiores, como por ejemplo RTL. Además, ofrece la posibilidad de obtener múltiples implementaciones del *hardware* a partir de un único código C, C++ o SystemC mediante el uso de directivas que controlan el proceso de síntesis.

El flujo de diseño de *Vivado HLS* se muestra en la de la Figura 32. Tal y como se observa, *HLS* requiere de: (i) función C, C++ o SystemC que ejecute la funcionalidad del *hardware* a diseñar; (ii) restricciones que especifique el periodo y la incertidumbre del reloj y el modelo de la FPGA; (iii) de forma opcional, directivas que definan el comportamiento u optimicen la implementación; y (iv) banco de pruebas (en inglés, *testbench*) del algoritmo. El flujo de diseño realiza las siguientes opciones con estos *inputs*:

(i) Compila, ejecuta y simula la función C, C++ o SystemC;

(ii) Sintetiza dicha función en una implementación RTL VHDL o Verilog;

(iii) Simula la implementación RTL obtenida utilizando el *testbench* original, comparándola con la especificación original,

(iv) Empaqueta el *hardware* generado en un bloque de *Intellectual Property* o IP, el cual puede ser utilizado en otros entornos de diseño Xilinx; y

(v) Genera informes de resultados de la síntesis, C/RTL co-simulación y encapsulado.

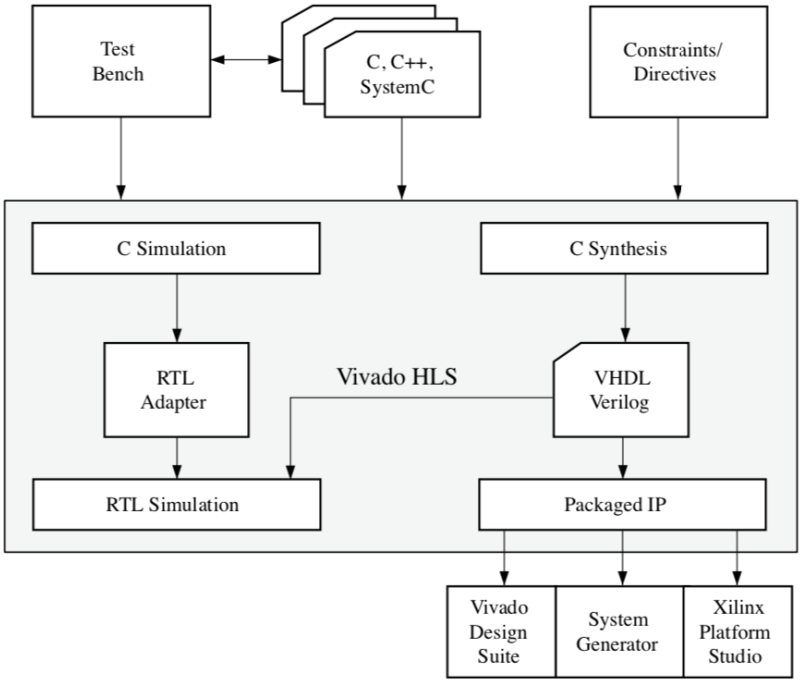

Figura 32. Flujo de diseño de Vivado HLS [42].

4.3.1.2. Vivado Integrated Design Environment (Vivado IDE)

Vivado Integrated Design Environment o *Vivado IDE* [43] es un entorno de Xilinx que, a través de una Interfaz Gráfica de Usuario (*Graphical User Interface* o GUI) (Figura 33), ofrece herramientas de desarrollo integrado, que incluye las siguientes tareas: conformado por las siguientes tareas:

i. **Diseñar plataformas *hardware*** a nivel RTL (VHDL, Verilog o SystemVerilog) o mediante la integración de bloques IP propios de Xilinx y/o elaborados en Vivado HLS. Vivado IDE proporciona un catálogo de bloques de *Intellectual Property* verificados.

ii. **Simular el diseño** HDL (*Hardware Description Language*). La simulación se realiza mediante el Simulador Vivado y requiere de un *testbench* a nivel RTL del *hardware*. Vivado IDE provee bloques IP de Verificación que simula interfaces AXI y *testbench* ejemplo. Esta también permite el uso de otros simuladores (ej.: *Cadence Xcelium*).

iii. **Sintetizar e implementar**. Realizada la síntesis y la implementación del *hardware*, la herramienta genera un esquemático detallado del *hardware* sintetizado, una vista de su implementación sobre la FPGA, múltiples informes de resultados, tales como: consumo de potencia y de recursos, *timing* del ruteado, etc.

iv. **Generar el *bitstream*** de la implementación del *hardware* y programar la FPGA.

v. **Depurar la plataforma *hardware***. Vivado IDE dispone de analizadores lógicos integrados (*Integrated Logic Analyzer* o ILA) y de herramientas de gestión para el acceso al *hardware* (*Hardware Manager*).

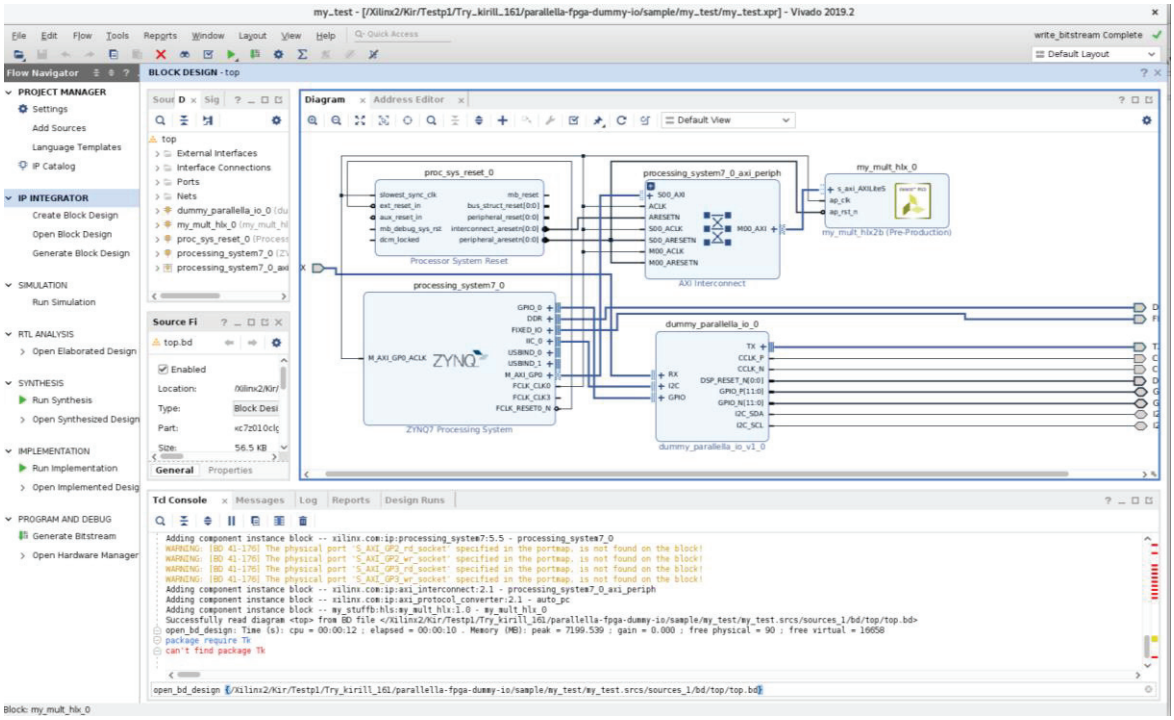

Figura 33. Vivado Integrated Design Environment o Vivado IDE.

4.3.1.3. Vitis Integrated Development Environment (Vitis IDE)

Vitis Integrated Development Environment o *Vitis IDE* [44] es un entorno propio de Xilinx de desarrollo de aplicaciones *software* empotradas dirigidas a procesadores integrados en los dispositivos SoC y MPSoC de Xilinx. Esta herramienta toma como referencia la plataforma *hardware* creada en Vivado IDE y está basada en entorno de desarrollo de código abierto Eclipse. Vitis IDE dispone de un editor de código C/C++, un gestor de proyectos, diferentes opciones de compilación, un control de versiones del código fuente, un navegador de errores, y un entorno que integra los diferentes depuradores de código proporcionados por Xilinx.

El flujo de desarrollo del *software* integrado en Vitis IDE se muestra en la Figura 34. Incluye las siguientes tareas:

(i) exportar el *hardware* diseñado en Vivado IDE a un archivo XSA que incluye la descripción de la plataforma integrada y el fichero *bitstream* de configuración;

(ii) crear una plataforma *software* en Vitis e importar el fichero XSA;

(iii) crear un dominio (*Board Support Package* (BSP) o sistema operativo compuesto por una colección de *drivers*), es decir, configurar el entorno *software*;

(iv) elaborar la aplicación *software* basada en el *hardware* y en el dominio;

(v) depurar la aplicación en Vitis;

(vi) depurar a nivel de sistema en caso de ejecutar varias aplicaciones, una por procesador integrados en el MPSoC;

(vii) generar la imagen de arranque que inicializa el sistema y lanza la aplicación principal.

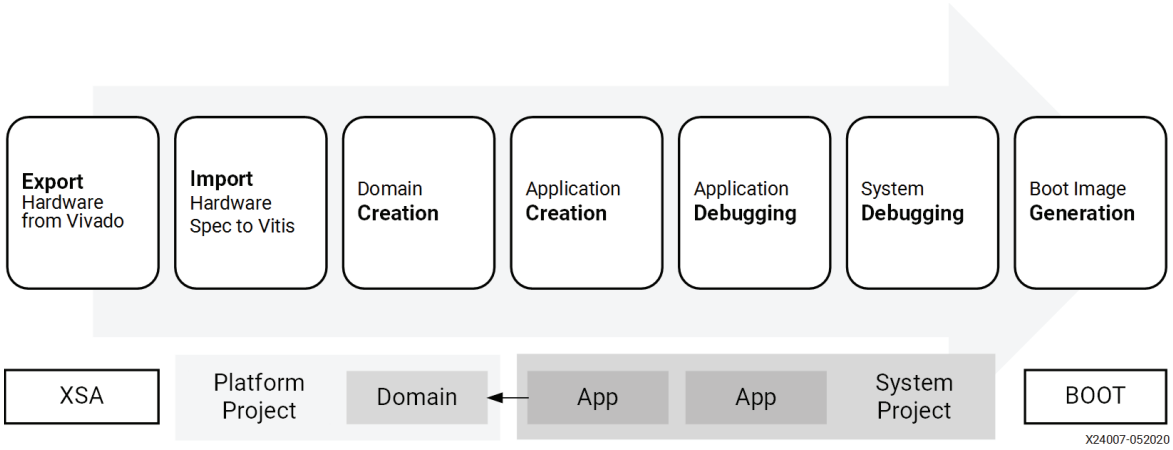

Figura 34. Flujo de desarrollo de aplicaciones software integradas en Vitis IDE [44].

4.3.1.4. Cadence SimVision Debug

Cadence SimVision Debug de *Cadence Xcelium Parallel Logic Simulation* [45] es un entorno de depuración y simulación de diseños digitales, analógicos o de señal mixta descritos en Verilog, SystemVerilog, VHDL y/o SystemC. *SimVision* soporta flujos a nivel de señal y de transacciones, y, a través de una interfaz gráfica, permite analizar el comportamiento del diseño y del *testbench*. Esta herramienta, a fin de afrontar la compleja tarea de depuración, consta de varias ventanas, entre las cuales se destacan:

1. *Propiedades*: permite administrar los objetos de depuración (cursores, marcadores, etc.).
2. *Navegador del diseño*: permite supervisar las señales y variables del diseño.
3. *Forma de Onda*: traza las señales simuladas sobre un eje XY *(*Figura 35*)*.
4. *Navegador de código fuente*: da acceso al código fuente del diseño.

Además, permite invocar la simulación desde la interfaz gráfica, compilando los ficheros modificados y generando el ejecutable necesario para la realización de la simulación durante el proceso de depurado del bloque IP.

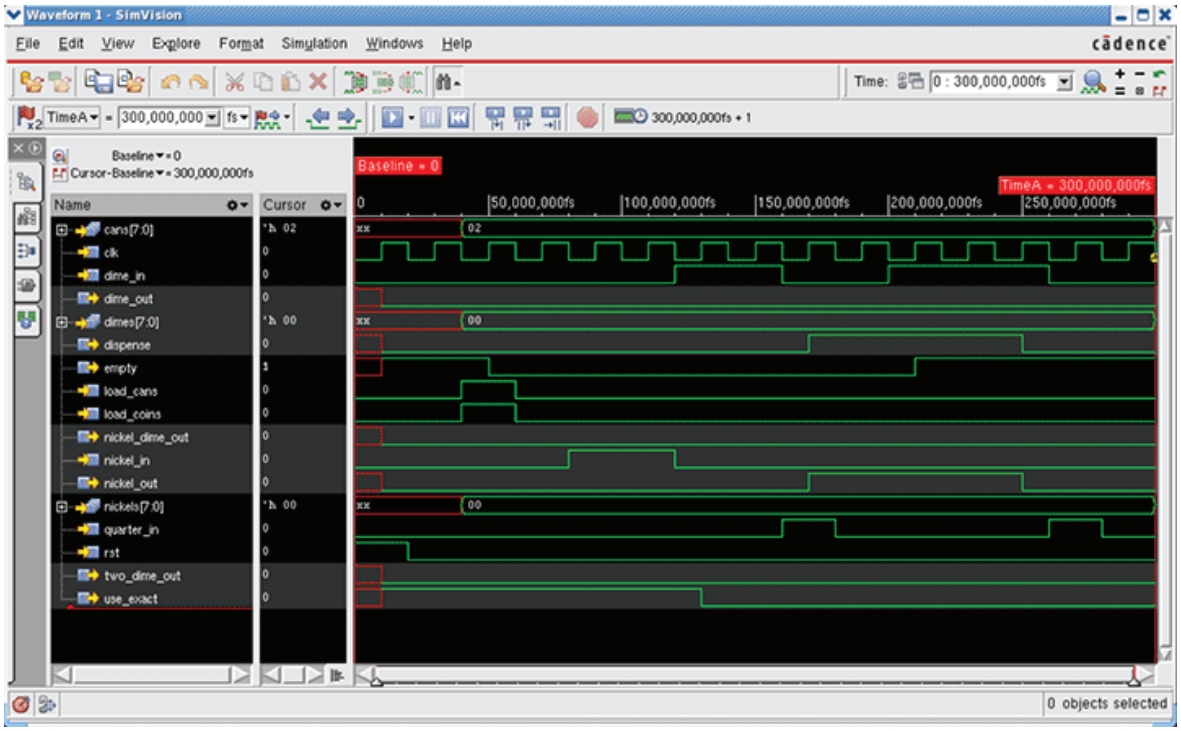

Figura 35. Ventana de Forma de Onda de SimVision [45].

4.4. Instrumentación: Analog Discovery 2 de Digilent

Analog Discovery 2 de Digilent [46] es un dispositivo multi-función de reducido tamaño y de bajo costo que integra trece instrumentos de prueba y medida, que permiten medir, visualizar, generar, registrar y controlar circuitos de señal mixta. Además, Digilent ofrece el *software* libre *WaveForms*, el cual permite controlar y configurar el *Analog Discovery 2* usando una interfaz gráfica (Figura 36).

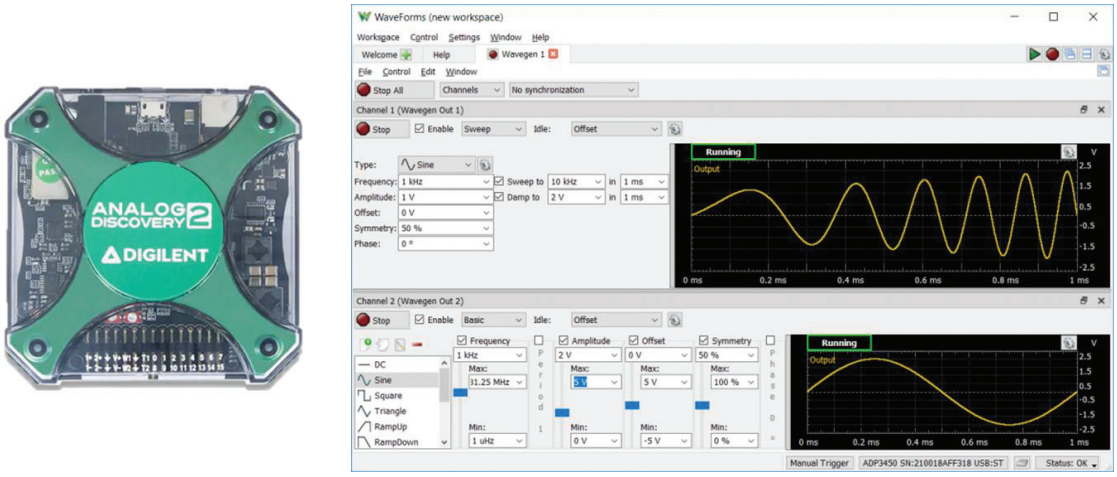

Figura 36. Analog Discovery 2 y el entorno WaveForms (adaptada de [46]*).*

A continuación, se describen cinco de los instrumentos que incluye:

- Osciloscopio: Dos canales, 1MΩ, ±25 V, 14 bits, 100 MSP/s y 300 MHz + BW.
- Generador de Formas de Onda: Dos canales, ±5 V, 14 bits, 100 MSP/s y 12 MHz + BW.
- Generador de Patrones: Dieciséis canales, 3.3 V CMOS y 100 MSP/s.
- Analizador Lógico: Dieciséis canales, 3.3 V CMOS y 100 MSP/s.
- Analizador de Protocolos: SPI, IIC, UART y Parallel.

4.5. Conclusión

En este capítulo, se han descrito las herramientas de desarrollo *System on-Chip* que ofrece Xilinx y sus respectivos flujos de diseño, que facilitan la elaboración de una solución *on-Chip* Zynq de alto rendimiento. Además, se han expuesto, como apoyo a la verificación y validación del *System on-Chip*, el simulador *SimVision* de Cadence y el multi-instrumento *Analog Discovery 2* de Digilent.

El uso de estas herramientas y metodologías, así como la reutilización de bloques *hardware*, librerías y *drivers* de Xilinx y Analog Devices, es fundamental en el presente Trabajo, dado que aporta un alto grado de fiabilidad a la solución final y permite reducir el tiempo de desarrollo.

Capítulo 5. PLATAFORMA *HARDWARE*

En este capítulo, se diseñan los diferentes bloques hardware que conforman la plataforma hardware que da solución al presente Trabajo de Fin de Máster. Además, a partir de estos IPs, se construye, sintetiza e implementa la plataforma hardware final. Muchos de los IPs se han desarrollado utilizando metodologías de diseño de alto nivel (algoritmos en C++) o metodologías de diseño mixtas (algoritmos en C++ y RTL). La elección de la metodología ha dependido de las exigencias del diseño. Otros IPs, en cambio, se han implementado mediante la reutilización de bloques hardware ya verificados (repositorios).

5.1. Introducción

A lo largo del presente Trabajo de Fin de Máster se han estudiado diversas implementaciones de la plataforma *hardware* de la solución *System-on-Chip* expuesta. Entre las múltiples opciones, se destacan dos versiones sustancialmente diferentes: una de procesamiento independiente por canal de captura (arquitectura paralela) y otra de procesamiento semejante entre canales (arquitectura serie).

Ambas arquitecturas presentan sus ventajas y desventajas con respecto al consumo de recursos *hardware*, la flexibilidad del procesamiento, el ancho de banda de la salida de datos y la complejidad en la interpretación de los datos procesados. La arquitectura paralela (Figura 37), dado que cada canal de captura es tratado de forma individualizada, exige un mayor consumo de lógica y permite procesar de forma personalizada cada canal, lo cual produce una mayor tasa de datos procesados y, además, requiere un algoritmo de reconstrucción de datos más complejo. Por el contrario, la arquitectura serie (Figura 38), ya que todos los canales son tratados de igual forma, permite procesar las muestras como paquetes de muestras y, por tanto, presenta un menor consumo de lógica, produce una menor tasa de datos procesados y exige un algoritmo de reconstrucción más sencillo.

Debido al uso de una tarjeta de prototipado de bajo coste y al imperativo de transferir los datos procesados hacia un servidor, entre los aspectos expuestos, el más determinante es el ancho de banda de la salida de datos.

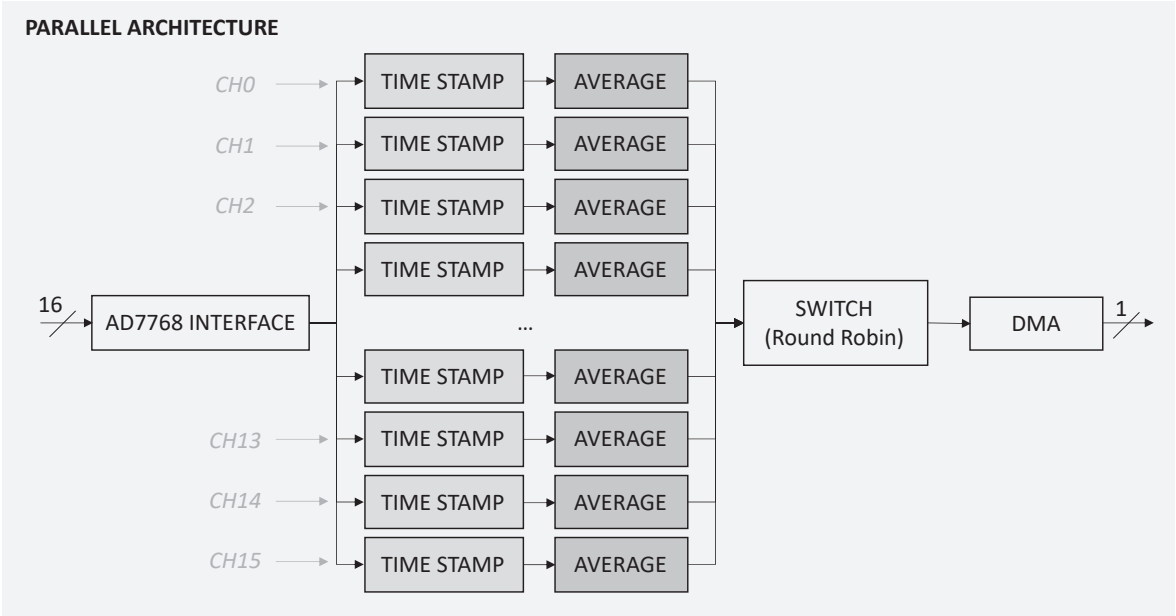

Figura 37. Arquitectura de procesamiento paralela.

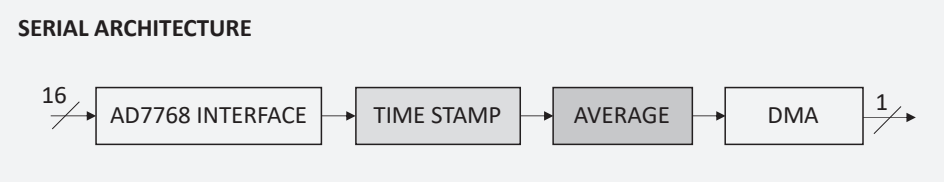

Figura 38. Arquitectura de procesamiento serie.

Durante este Trabajo se han desarrollado ambas plataformas (paralela [47] y serie). No obstante, en el presente documento únicamente se expone el diseño de la *arquitectura serie*.

La obtención de la plataforma *hardware* del *System-on-Chip* exige el estudio, diseño y verificación de los diferentes bloques *hardware* de procesamiento de datos y de control del sistema. Además, requiere la integración e interconexión de dichos componentes *hardware* y, la correcta, síntesis e implementación de la integración final.

5.2. Bloques *hardware* de procesamiento de datos

A continuación, se expone el diseño de los IPs de *procesamiento de datos* del sistema.

5.2.1. IP Interfaz AD7768

El objeto del IP Interfaz AD7768 es interpretar los datos generados por los ADCs y transmitir las muestras e información descriptiva de las mismas (ej.: canal a través del cual fue capturada). Este etiquetado de las capturas es imprescindible para el post-análisis científico de las señales de fondo, puesto que permite identificar las señales de entrada.

La interpretación de la información digital debe adaptarse al formato de la interfaz de salida de datos y configuración de los AD7768, así como comprobar que los datos interpretados son válidos (ej.: analizar el CRC de los datos, corroborar la recepción de 32 bits de datos, etc.).

Analog Devices ofrece IPs desarrollados en *Verilog* HDL que dan soporte al desarrollo de plataformas *hardware* Zynq-7000 de adquisición que integran el ADC AD7768 [48]. Entre los diferentes recursos *hardware* que proporciona se encuentra el IP AD7768_IF (Figura 39). Esta interfaz dedicada admite como entradas la salida de datos de un AD7768 (DCLK, DRDY y ocho salidas DOUT) y su configuración (ej.: conversión *One-Shot* o *Standard*; formato; implementación del CRC, etc.) (Figura 40) y, como salida, ofrece buses de datos y validación seriales y paralelas (un hilo por canal) (Figura 41). Con este IP es posible interpretar los datos generados por un ADC y comprobar la validez de las muestras.

Dado que este bloque *hardware* cubre gran parte de la funcionalidad del IP Interfaz AD7768 objeto, se diseña un IP que integra dos IP AD7768_IF (uno por ADC), una interfaz AXI4-Lite que permite ajustar las interfaces a la configuración de los AD7768 desde el procesador ARM Cortex A9 y una (u ocho) interfaz AXI4-Stream que permite transferir a la plataforma *hardware* las muestras como un *stream* de datos, dada la conversión continua de los ADC. Este nuevo IP se denomina *IP Interfaz AD7768*.

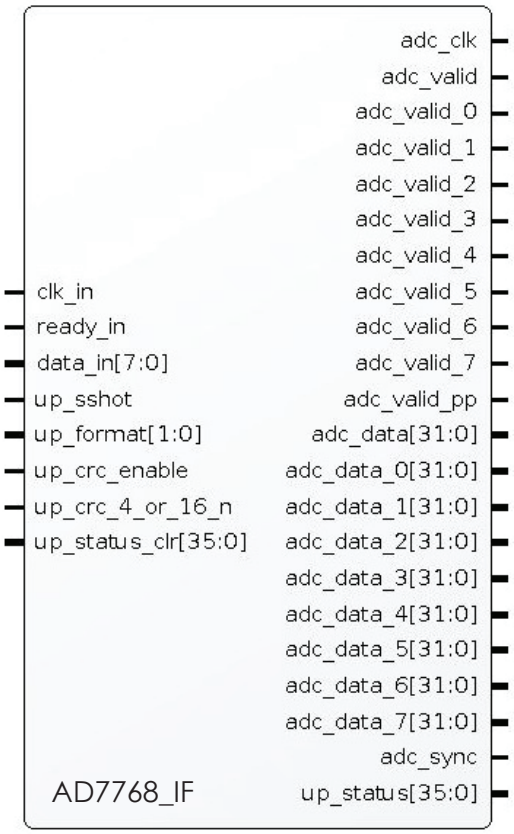

Figura 39. IP AD7768_IF de Analog Devices [48].

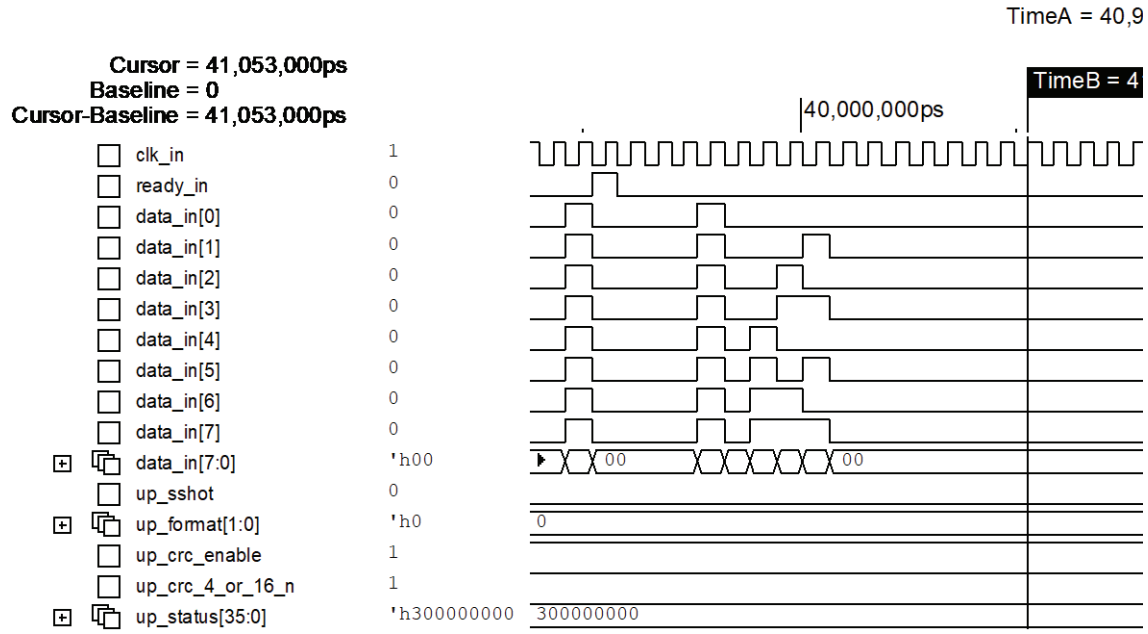

Figura 40. Entrada de señales en el IP AD7768_IF de Analog Devices. DCLK = 8,192MHz, Standard Operation, FORMAT = 00, CRC every four samples.

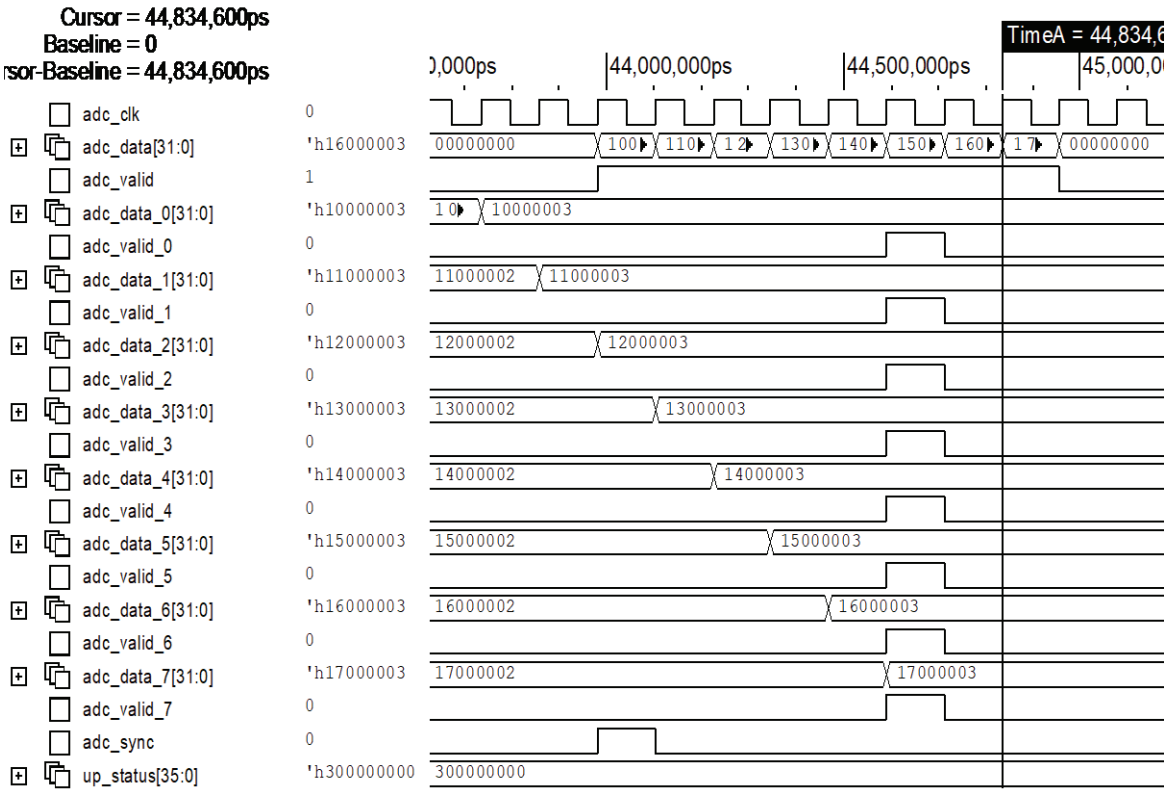

Figura 41. Salida de señales del IP AD7768_IF de Analog Devices. DCLK = 8,192MHz, Standard Operation, FORMAT = 00, CRC every four samples.

En vista de que el IP AD7768_IF proporciona salidas de datos y validación seriales y paralelas, se evalúan las ventajas y desventajas de un procesamiento serial y uno paralelo en términos de complejidad, escalabilidad, consumo de recursos *hardware*, potencia y ancho de banda de salida.

Uno de los factores claves es el correcto procesamiento de las muestras. Es de vital importancia conocer el canal que capturó la muestra a fin de distinguir las señales durante el procesamiento. El IP AD7768_IF mantiene la filosofía de salida de datos del AD7768, es decir, cada dato de salida está conformado por una cabecera de 8 bits y por 24 bits de datos y, en caso de habilitar la Verificación de Redundancia Cíclica (CRC), cada 4 o 16 muestras, la cabecera del dato es sustituida por el resultado del CRC. Por consiguiente, en el supuesto de implementar CRC en la conversión, lo cual es imprescindible en este ámbito, y de recibir capturas no-válidas, supone mayor dificultad identificar el origen de las muestras en una transmisión serial que en una paralelizada (un hilo por canal).

Otro de los aspectos más importantes a considerar en el diseño de este *System-on-Chip* es la posibilidad de escalar y paralelizar el procesamiento por canal, dado que el procesamiento a efectuar es similar en los dieciséis canales de captura (interpretar datos generados, añadir marcas de tiempo y calcular promedios) y dada la capacidad de paralelismo que ofrece el *hardware* de la FPGA. Dicho procesamiento paralelo permite personalizar el procesamiento por canal de muestreo. No obstante, requiere un mayor consumo de recursos *hardware*.

Un último factor indispensable para considerar es la tasa de datos procesados generados. Realizar un procesamiento paralelo o independiente por canal, supone adición de marcas de tiempo real, también, por canal. Esto provoca un aumento drástico del ancho de banda de salida necesario. Una forma de reducir la tasa de datos es procesar las dieciséis señales por igual (ej.: las dieciséis señales son muestreadas a la misma frecuencia de muestreo, las dieciséis señales son promediadas por el mismo factor, etc.). Esto permite procesar las muestras como paquetes de dieciséis muestras, ya que los dieciséis canales capturan y se procesan a la vez, y, por tanto, añadir una única marca de tiempo real por paquete. La captura y procesamiento por igual en los dieciséis canales, permite tener múltiples canales de captura idénticos.

El procesamiento descrito es un procesamiento tipo serie, puesto que los paquetes de muestras son transferidos uno detrás de otro, y, por tanto, utiliza menos recursos *hardware* frente al paralelo.

Por estas razones, en esta oportunidad, se opta por utilizar la interfaz de salida paralela de los IPs AD7768_IF e implementar una arquitectura de procesamiento de tipo serie.

5.2.1.1. Comunicación y control del IP

El IP Interfaz AD7768 admite como entrada las salidas de datos de ambos AD7768. Por tanto, a fin de interpretar y validar las muestras capturadas, se requiere un fichero de restricciones XCF

(Xilinx Constraints File) que vincule las entradas *dclk*, *drdy0*, *drdy1*, *din0* y *din1* a los pines del FMC de la *ZedBoard* que conecten con la salida de datos de cada ADC (DCLK, DRDY y DOUTx).

Con el objeto de configurar y/o controlar el IP Interfaz AD7768 desde una aplicación *software*, se dota de una interfaz AXI4-Lite que permite:

i. Controlar la transmisión de las muestras al *hardware* (*enable*), puesto que la conversión A/D es continua tras la alimentación de los ADC (1 bit).

ii. Habilitar o deshabilitar el CRC (1 bit).

iii. Definir cada cuántas capturas genera el CRC el conversor (4/16 muestras) (1 bit).

La configuración de estos parámetros requiere 3 bits, es decir, un registro AXI4-Lite. Por lo tanto, en Vivado HLS se crea una función C++ que permite escribir en un registro de estado de 32 bits por AXI4-Lite y que retorna la configuración del IP Interfaz AD7768 (Código 9). La distribución de los parámetros en dicho registro se observa en la Figura 42.

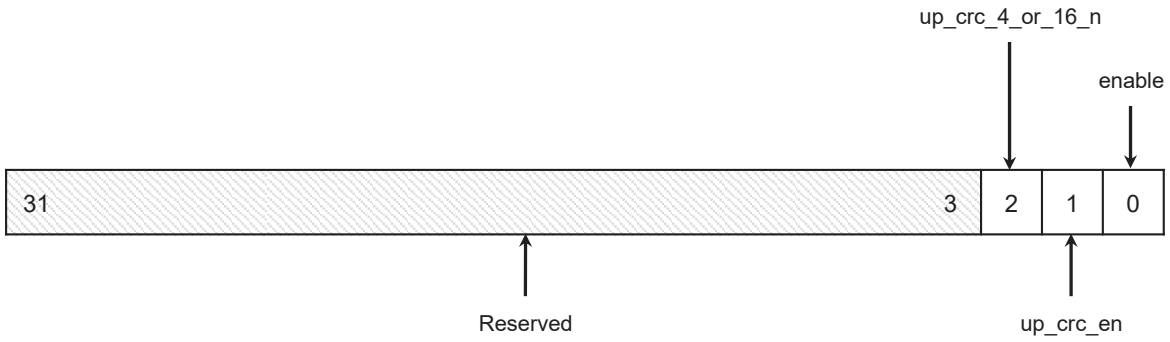

Figura 42. Registro de control del IP Interface AD7768. Dirección 0h10.

Tabla 9. Descripción de los registros de control del IP Interface AD7768.

Bits	Identificador	Descripción
0	enable	Habilitar o deshabilitar la transferencia de las muestras al *hardware*. 0 = No transferir muestras. 1 = Transferir muestras.
1	up_crc_en	Indicar si los AD7768 realizan Verificación de Redundancia Cíclica. 0 = CRC no habilitado. 1 = CRC habilitado.
2	up_crc_4_or_16_n	Indicar cada cuántas muestras (4/16) los AD7768 generan el CRC. 0 = cada 16 muestras. 1 = cada 4 muestras.

Las entradas de configuración de la operación de conversión (*up_sshot*), del formato (*up_format*) y del estado inicial de errores (*up_status_clr*) de ambos IP AD7768_IF están fijados a cero,

puesto que la conversión analógica-digital siempre es *standard* o continua, el formato de las interfaces de salida de datos de ambos ADC está fijado al FORMAT 00 y el estado de errores de partida siempre es cero.

Por otra parte, a fin de identificar y transmitir las muestras capturadas por los dieciséis canales, se dota al IP Interfaz AD7768 de una interfaz AXI4-Stream que etiqueta, empaqueta y serializa las muestras recibidas a través de las salidas paralelas de ambos IP AD7768_IF. Por tanto, en Vivado HLS se crea una función C++ que realiza las siguientes operaciones (Código 10):

(i) Recibir las dieciséis rutas de datos de 32 bits y de validación.
(ii) Realizar un *polling* entre las rutas de datos y de validación.
(iii) Etiquetar las muestras recibidas según el conversor y el canal de captura con 4 bits.
(iv) Concatenar las dieciséis muestras identificadas en una trama de 512 bits.
(v) Enviar dicha trama a través de una interfaz AXI4-Stream básica (*TDATA*, *TREADY* y *TVALID*).

De este modo, ya que los dieciséis canales estarán en captura continua, configurados de igual forma y sincronizados, las tramas estarán conformadas por las dieciséis muestras capturadas de forma simultánea. Dichas etiquetas reemplazan los 8 bits de cabecera de las muestras y, por lo tanto, ahora las muestras están conformadas por 28 bits (4 bits de etiqueta y 24 bits de dato). En la Figura 43, se observa el formato de la trama de datos de salida.

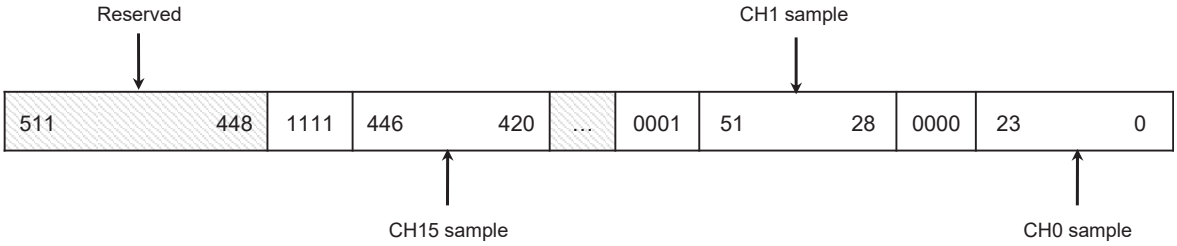

Figura 43. Formato de las tramas de datos de salida del IP AD7768 Interface.

5.2.1.2. Integración del bloque

La interfaz AXI4-Lite opera a la frecuencia del PS, puesto que los registros de control del IP se escriben desde una aplicación *software* que se ejecuta en el *Processing System*. Sin embargo, los IPs AD7768_IF y la interfaz AXI4-Stream funcionan a la frecuencia de los AD7768, dado que el bloque *hardware* de Analog Devices opera y entrega las muestras de forma síncrona a dicha frecuencia. Por consiguiente, para asegurar el registro de la configuración de los IPs AD7768_IF por AXI4-Lite, se utilizan técnicas *Clock Domain Crossing* (CDC) en los registros de configuración, que permiten sincronizar las señales del dominio de reloj del PS (clk_PS) con el dominio de reloj del ADC (clk_adc) (Figura 44).

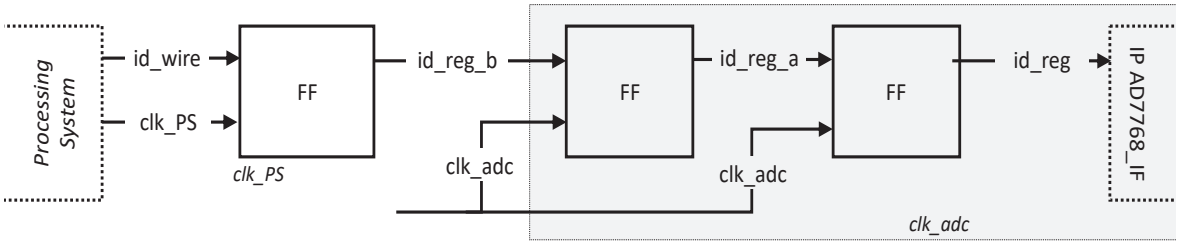

Figura 44. Registro de una señal aplicando Clock Domain Crossing.

En el Código 11, se expone la descripción Verilog del IP Interfaz AD7768, y, en la Figura 45, se muestran las interfaces de E/S de dicho bloque *hardware*. Este IP ha sido sintetizado para operar a una frecuencia de 100MHz.

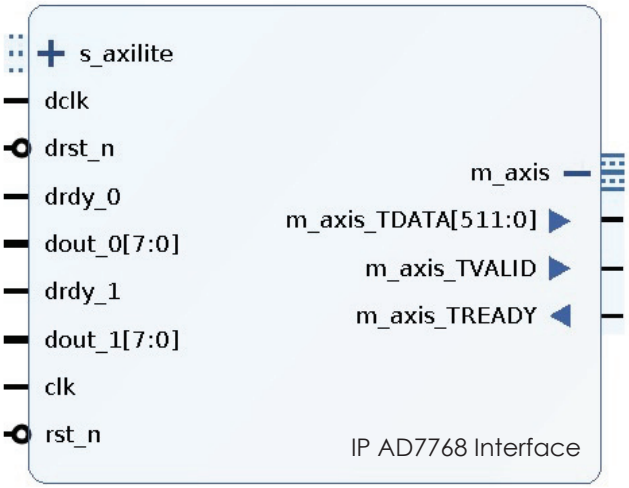

Figura 45. IP Interfaz AD7768.

Tabla 10. Descripción de las E/S del IP Interfaz AD7768.

Descripción de las E/S del IP Interfaz AD7768

clk	Señal del reloj de la interfaz AXI4-Lite.
rst_n	Señal de reset activa a nivel bajo de la interfaz AXI4-Lite.
s_axilite	Interfaz AXI4-Lite de configuración del IP.
dclk	Entrada de la señal DCLK del AD7768-0.
drdy0	Entrada de la señal DRDY del AD7768-0.
drdy1	Entrada de la señal DRDY del AD7768-1.
dout0	Entrada de las señales DOUT del AD7768-0.
dout1	Entrada de las señales DOUT del AD7768-1.
m_axis	Salida AXI4-Stream. Salida de las tramas de muestras.

En la Figura 46 se muestran los recursos *hardware* consumidos por el bloque. En la Figura 47 y en la Figura 48 se observa la respuesta del IP Interfaz AD7768. En este caso, el IP ha sido configurado para que tenga en cuenta el CRC generado por los ADC cada cuatro muestras. La señal de reloj *dclk* (ADC) es de 8.192MHz y *clk* (PS) de 100MHz.

Tal y como se aprecia, el IP, tras recibir las muestras de los dieciséis canales, (i) interpreta; (ii) valida; y (iii) transmite al *hardware* de forma correcta las dieciséis capturas en 1.89µs. Este tiempo de respuesta cumple con las restricciones temporales del sistema, puesto que al asignar marcas de tiempo real en milisegundos el error temporal cometido es despreciable (0.189%). Además, se observa el adecuado funcionamiento de la señal *enable* del IP.

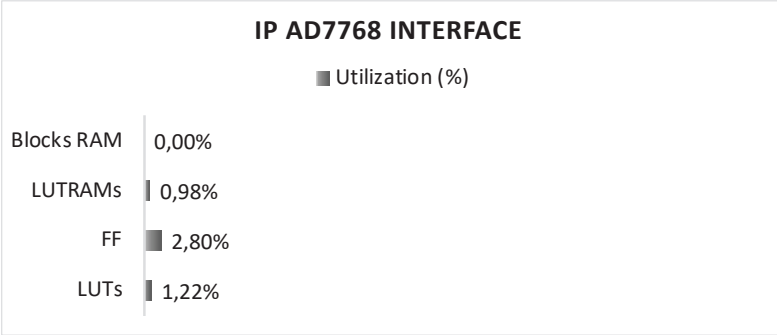

Figura 46. Recursos hardware consumidos por el IP Interfaz AD7768.

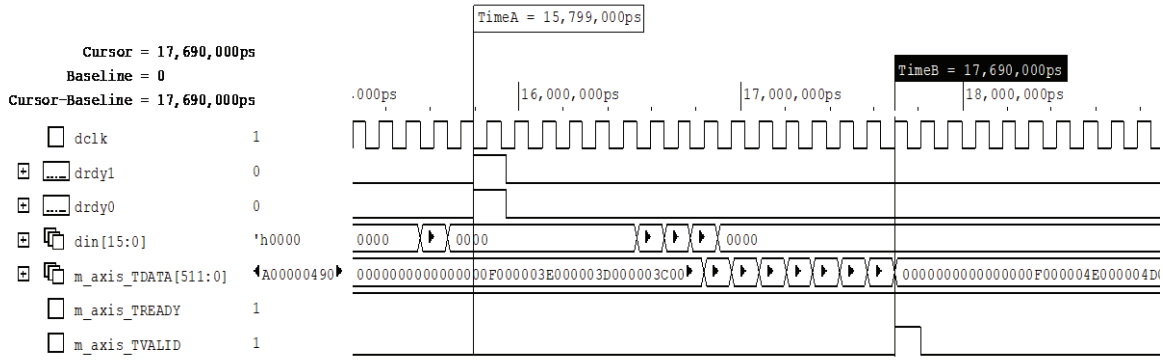

Figura 47. Verificación del IP Interfaz AD7768. Medición de tiempo de respuesta.

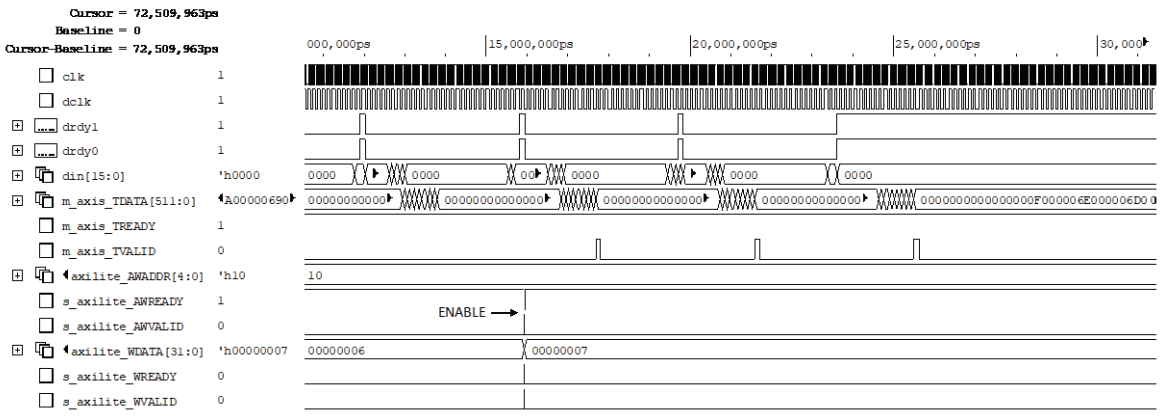

Figura 48. Verificación del IP Interfaz AD7768. Habilitar transferencia de datos.

5.2.2. IP DE MARCA DE TIEMPO REAL

La misión del IP de Marca de Tiempo Real es añadir marcas de tiempo real con precisión de segundos o milisegundos a las muestras. Con este bloque *hardware* se atiende a la petición de inserción de marcas de tiempo real especificada por el IAC.

Para implementar esta funcionalidad es preciso obtener y transferir al bloque *hardware* la fecha y hora real. Por lo tanto, dado que el dispositivo Zynq-7000 carece de circuito *Real Time Clock*, se propone dotar al sistema con al menos una de las dos siguientes opciones:

a) Obtener la fecha y hora real desde un servidor *Network Time Protocol* (NTP). Esta opción permite conocer el tiempo real en segundos y en milisegundos, solo si, el sistema dispone de conectividad externa mediante el correspondiente *stack* TCP/IP.

b) Obtener la fecha y hora real desde un *Real Time Clock* (RTC) externo conectado a la *ZedBoard*. Esta otra opción permite el funcionamiento del sistema en modo *sin conexión*. Si bien, solo permite conocer el tiempo real con resolución de segundos.

El presente Trabajo opta por la solución b) ya que dota al equipo de un funcionamiento autónomo, sin conexión a un servidor NTP. Para ello se obtiene la información de tiempo real en segundos a partir de un *Real Time Clock* externo. El *System On-Chip* implementado incorpora la funcionalidad de marcas de tiempo real en segundos y milisegundos.

En este caso, a fin de facilitar el tratamiento del tiempo real, se propone añadir *time stamps* en formato Tiempo Unix con resolución de segundos o milisegundos, el cual se define como la cantidad de segundos o milisegundos transcurridos desde la medianoche UTC del 1 de enero de 1970. Esto requiere la conversión previa a Tiempo Unix el tiempo real obtenido desde el RTC.

Dicho Tiempo Unix en segundos es un número real escalar de 32 bits y en milisegundos de 64 bits. Por tanto, se fija un ancho de 64 bits de *time stamp*, ya que admite ambas unidades de tiempo.

En la Figura 49, se observa la arquitectura del IP de Marca de Tiempo Real propuesto, el cual añade *time stamps* en segundos o milisegundos en formato Unix de 64 bits a las tramas de muestras.

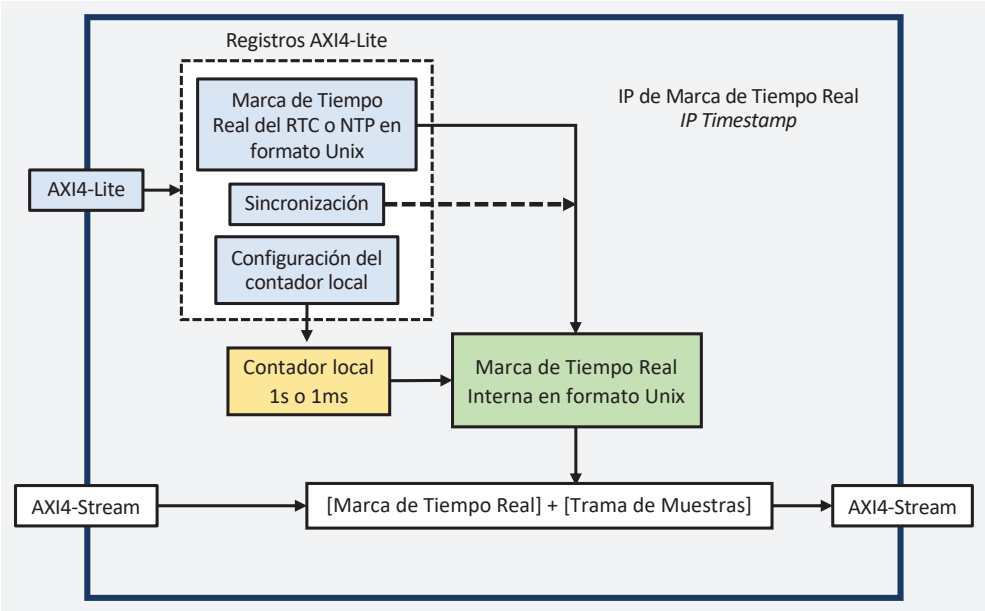

Figura 49. Diagrama de bloques del IP de Marca de Tiempo Real.

Tal y como se aprecia, con el propósito de transferir al bloque *hardware* el tiempo real, se confiere al IP de una interfaz AXI4-Lite que permite escribir en registros internos la fecha y hora real en formato Unix. A continuación, a fin de reducir las lecturas del tiempo real del módulo RTC se transcribe en un nuevo registro interno e incrementa a nivel local la marca Unix indicada cada segundo o milisegundo. La deducción del transcurso de un segundo o milisegundo se realiza mediante un contador local que utiliza el reloj del sistema. Después, con el objeto de sincronizar la marca Unix interna con el Tiempo Unix de los registros AXI4-Lite y con el propósito de configurar el contador local, se declaran nuevos registros AXI4-Lite dedicados. Por último, a fin de añadir la marca de tiempo real a las tramas de muestras ofrecidas por el IP Interfaz AD7768, se dota al IP de una estructura de "inserción de marcas" que recibe a través de una interfaz AXI4-Stream dichas tramas y que las envía, también por AXI4-Stream, con el *time stamp*. Esto permite insertar marcas de tiempo real a las dieciséis muestras capturadas de forma simultánea por los dieciséis canales, así como mantener el flujo de los datos.

A continuación, se describe el contador local, las interfaces AXI4-Lite y AXI4-Stream y la integración del IP de Marca de Tiempo Real.

5.2.2.1. Contador local

El IP de Marca de Tiempo Real requiere de una estructura *hardware* que permita registrar el transcurso de un segundo o milisegundo. Esto permite incrementar a nivel local la marca de tiempo real y, por ende, reducir las lecturas del tiempo real desde el módulo RTC.

Los bloques *hardware* de la Lógica Programable precisan de una señal de reloj que sincronice la funcionalidad del IP. Dicha señal es de frecuencia conocida de 100MHz.

Por lo tanto, se diseña un contador que cuenta *n* ciclos de reloj de dicha señal de reloj y genera un pulso al contar los *n* ciclos. De esta forma, por ejemplo, al contar 100 000 000 ciclos de una señal de reloj de 100MHz, se obtiene un pulso cada un segundo, y al contar 100 000, se obtiene un pulso cada un milisegundo. Este pulso controla el incremento de la marca de tiempo real interna.

En el Código 12, se muestra la descripción Verilog del contador local. Este presenta como entradas la señal de reloj de frecuencia conocida (*clk_in*), el reloj del sistema, dos señales de control (*enable* y *reset* a nivel bajo) y el límite de cuenta (*n*), y como salida, la señal que emite un pulso al contar los *n* ciclos de reloj (*clk_out*).

5.2.2.2. Interfaces de comunicación

Con la finalidad de configurar el IP de Marca de Tiempo Real desde una aplicación *software*, se dota a dicho bloque *hardware* de una interfaz AXI4-Lite que permite:

i. Indicar la fecha y hora real leída del RTC o servidor NTP en formato Unix (64 bits).

ii. Sincronizar la marca Unix interna con el Tiempo Unix indicado (1 bit).

iii. Especificar el límite de cuenta del contador local (*n*), el cual guía el incremento del *time stamp* interno cada segundo o milisegundo (32 bits).

Esta interfaz, además de facilitar la integración PS-PL, satisface los requisitos temporales que exige la transmisión y sincronización de una marca de tiempo real en segundos o milisegundos.

La configuración del IP de Marca de Tiempo Real requiere 97 bits, es decir, cuatro registros AXI4-Lite. Por tanto, a través de Vivado HLS se crea una función C++ que permite escribir en cuatro registros de estado y/o control de 32 bits por AXI4-Lite y que retorna la fecha y hora real leída desde el RTC en formato Unix, el bit de sincronización y el límite de cuenta (*n*) a especificar en el contador local.

En el Código 13, se describe dicha interfaz y, en la Figura 50, Figura 51, Figura 52 y Figura 53, se observa la distribución de dichos parámetros de configuración en los registros AXI4-Lite.

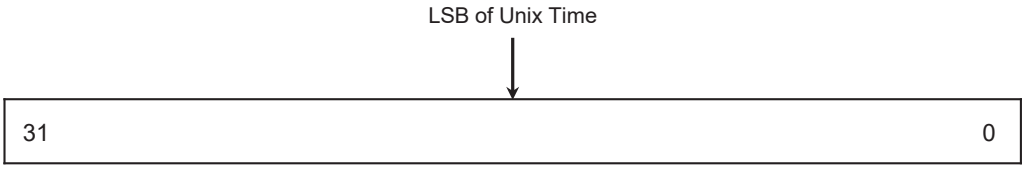

Figura 50. Registro de configuración del Tiempo Unix (LSB). Dirección 0h10.

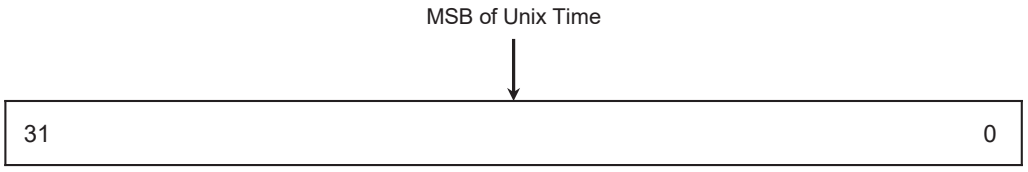

Figura 51. Registro de configuración del Tiempo Unix (MSB). Dirección 0h14.

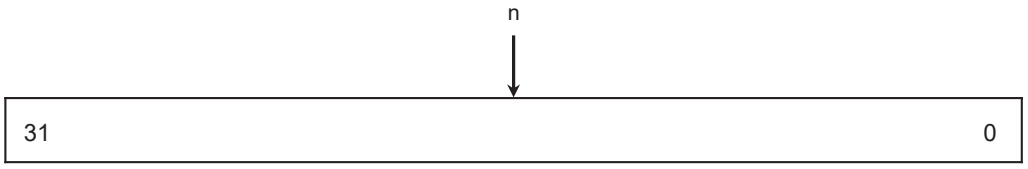

Figura 52. Registro de configuración del contador local. Dirección 0h18.

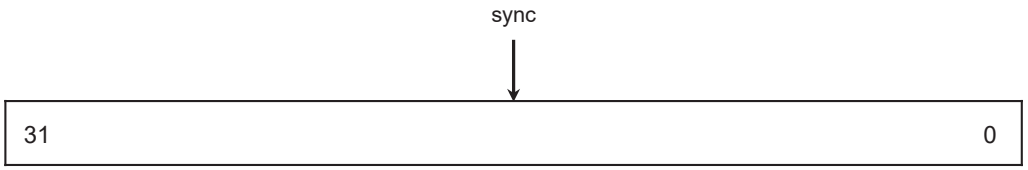

Figura 53. Registro de control de la sincronización. Dirección 0h1C.

Por otra parte, a fin de añadir las marcas de tiempo real a las tramas de muestras ofrecidas por el IP Interfaz AD7768, en Vivado HLS se crea una función en C++ que permite recibir las tramas por AXI4-Stream (512 bits), así como la marca Unix a insertar (64 bits), y que envía por AXI4-Stream las tramas marcadas (512 bits) (Figura 54). Ambas interfaces AXI4-Stream son interfaces AXI4-Stream básicas (*TDATA*, *TREADY* y *TVALID*) (Código 14). Esta función de "inserción de marcas" se instancia en el bloque *hardware* final.

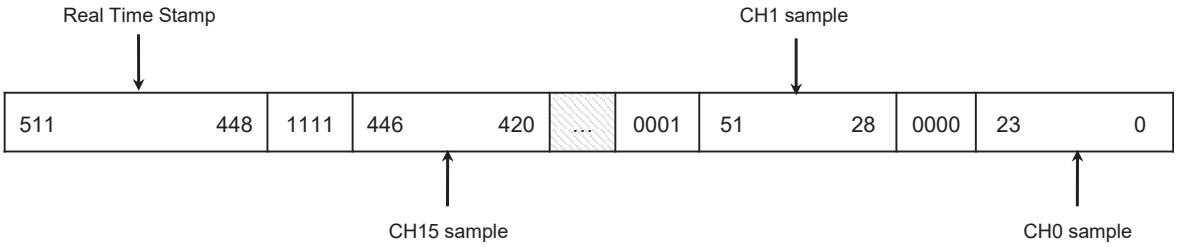

Figura 54. Formato de la trama de muestras con la marca de tiempo real.

5.2.2.3. Integración del bloque

Finalmente, el IP de Marca de Tiempo Real requiere una estructura *hardware* que permita trascribir en un registro interno e incrementar a nivel local el Tiempo Unix indicado por AXI4-Lite. En el Código 1, se observa la descripción Verilog de dicho bloque *hardware*.

```
// Synchronize internal registry with the Unix time indicated by AXI4-Lite
```

```
always @(posedge clk) begin
    i_clk_rst_n = 1'b1;

    if (i_clk) begin // Increase time stamp
        unixTime_reg = unixTime_reg + 1'd1;
    end

    if (syncTime) begin // Synchronization and reset local counter
        unixTime_reg = unixTime;
        i_clk_rst_n = 1'b0;
    end
end

assign unixTimestamp = unixTime_reg; // time stamp to insert to samples
```

Código 1. Sincronización e incremento local de la marca de tiempo real.

Esta, al recibir el *bit* de sincronización por AXI4-Lite, almacena en un registro interno el Tiempo Unix indicado y realiza un *reset* del contador local. Cuando se desactiva el *reset* se incrementa el Tiempo Unix indicado cada vez que el contador local registra el transcurso de un segundo o un milisegundo. Esta marca Unix interna es el *time stamp* a insertar a las muestras.

Con este IP *hardware*, se logra sincronizar la fecha y hora real del IP de Marca de Tiempo Real con la del RTC. Además, permite disponer del Tiempo Unix real en segundos o milisegundos a nivel local, lo que reduce las lecturas del módulo RTC.

En el Código 15, se expone la descripción Verilog del IP de Marca de Tiempo Real y, en la Figura 55, se muestran las interfaces de E/S de dicho bloque *hardware*. Este IP ha sido sintetizado para operar a una frecuencia de 100MHz.

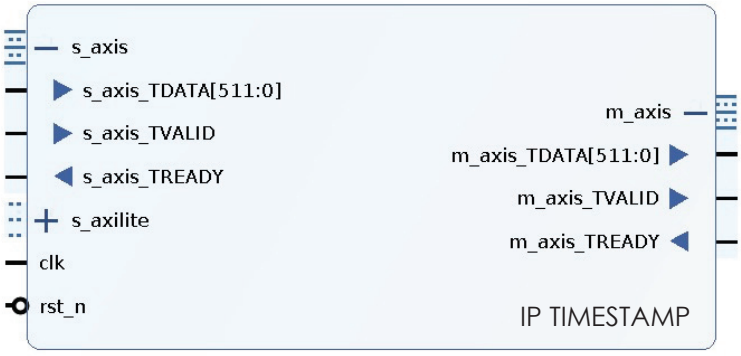

Figura 55. IP de Marca de Tiempo Real.

Tabla 11. Descripción de las E/S del IP de Marca de Tiempo Real.

Descripción de las E/S del IP de Marca de Tiempo Real	
clk	Señal del reloj.
rst_n	Señal de reseteo a nivel bajo.
s_axilite	Interfaz AXI4-Lite de configuración del IP.
s_axis	Entrada AXI4-Stream. Entrada de las tramas de muestras.
m_axis	Salida AXI4-Stream. Salida de las tramas de muestras con la marca de tiempo real.

Por último, se exponen los recursos *hardware* consumidos (Figura 56), se verifica la funcionalidad del IP y se miden los tiempos de respuesta.

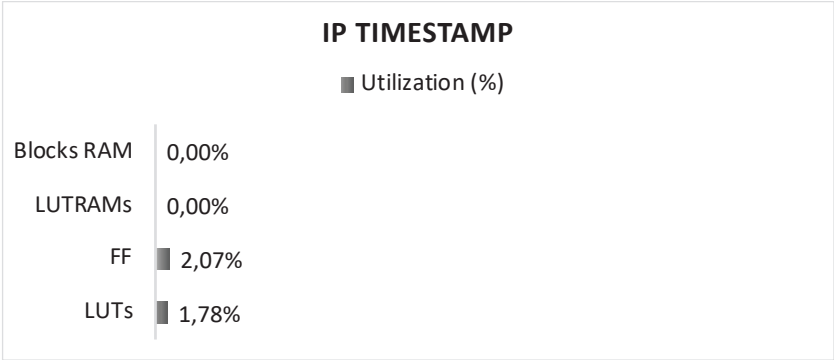

Figura 56. Recursos hardware consumidos por el IP de Marca de Tiempo Real.

En la Figura 57, Figura 58 y Figura 59, se observa la simulación mostrando la respuesta del IP de Marca de Tiempo Real. A modo de verificación, el IP ha sido configurado con un Tiempo Unix de 2ms (las 00:00:00h 002ms del 1 de enero de 1970) y de forma que incremente dicha marca Unix cada milisegundo (n = 100 000). La señal de reloj (*clk*) del bloque *hardware* es de 100MHz.

Como se aprecia, la configuración del IP ocupa 120ns. Este tiempo de configuración es válido, dado que la transmisión del tiempo Unix real con precisión de milisegundos, así como con precisión de segundos, al IP supone un error temporal en la marca Unix interna despreciable.

Por otro lado, la adición de la marca Unix a las muestras ocupa 20ns. Este tiempo de respuesta es válido, puesto que permite añadir marcas de tiempo real en milisegundos, así como en segundos, con un error temporal ínfimo.

Finalmente, se observa el incremento exacto de la marca Unix interna cada milisegundo.

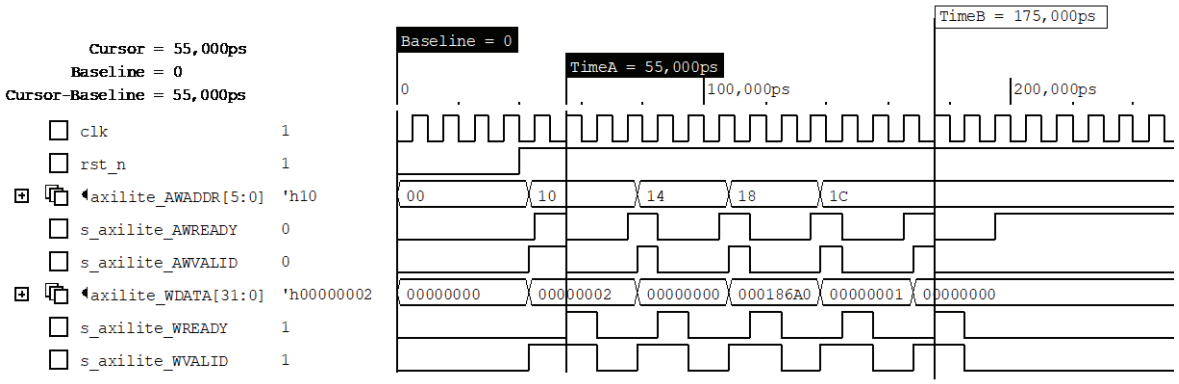

Figura 57. Verificación del IP de Marca de Tiempo Real. Tiempo de configuración.

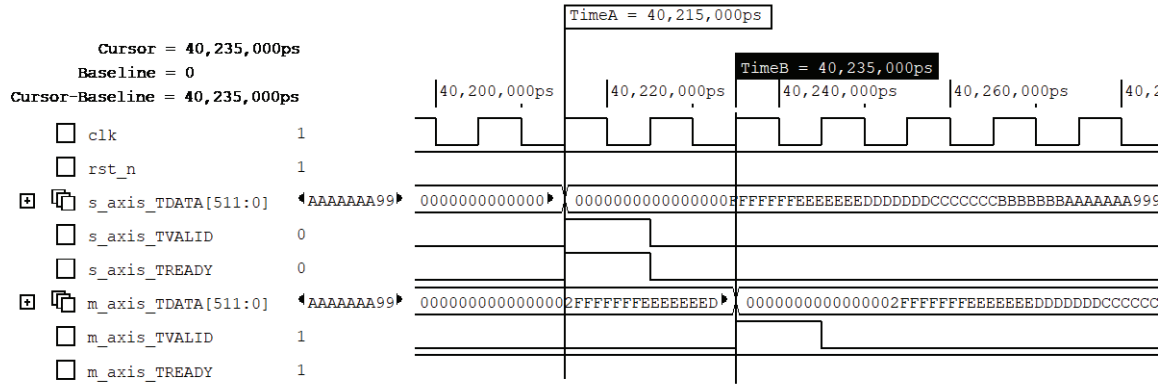

Figura 58. Verificación del IP de Marca de Tiempo Real. Añadir marca Unix a las muestras.

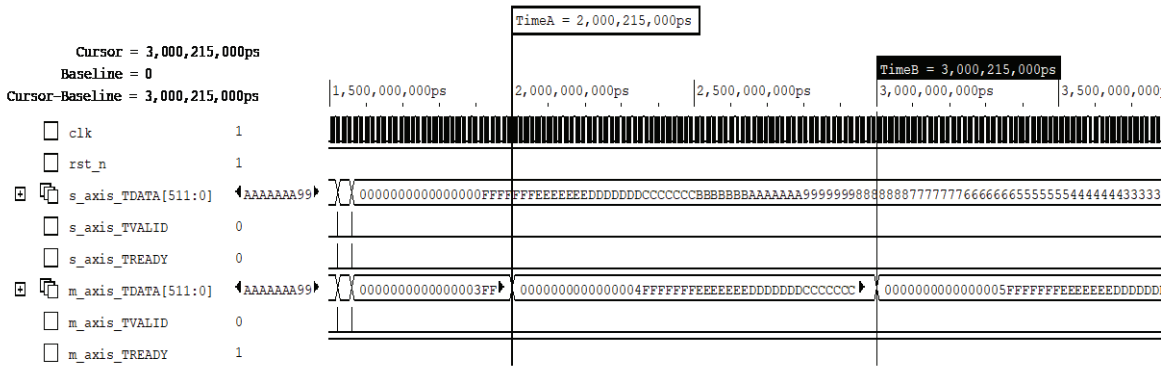

Figura 59. Verificación del IP de Marca de Tiempo Real. Incremento de la marca interna.

5.2.3. IP DE CÁLCULO DE PROMEDIOS

El IP de Cálculo de Promedios tiene como objeto realizar el promedio temporal de *n* muestras. Con este bloque *hardware* se cubre el requerimiento de promediar las señales de entrada cada *n* muestras.

En este caso, el promedio es la *media aritmética de un conjunto de valores, que se obtiene a partir de la sumatoria de dichos valores dividida entre el número de muestras* [49] (**3**).

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \cdots + x_n}{n} \quad (3)$$

Ante esta descripción, se diseña un IP que acumula en un registro interno las *n* muestras y, tras recibir las *n*, divide por el número de muestras recopiladas. Además, con el objetivo de reducir recursos de cómputo y restringir el consumo de lógica programable, que limita *n* a un número potencia de dos entre 2 y 4 096. Esto permite sustituir la operación de división por un desplazamiento de bits y evitar la implementación en *hardware* de divisores complejos.En la Figura 60, se observa la arquitectura del IP de Cálculo de Promedios expuesto.

Tal y como se aprecia en la Figura 60, se dota al IP de una interfaz AXI4-Lite que permite definir el número de muestras a promediar (*n*), así como elegir si promediar o hacer un *bypass* de los datos de entrada. Se confiere al bloque *hardware* de una estructura de procesamiento, que recibe por AXI4-Stream las tramas de muestras y envía, también por AXI4-Stream, las tramas con las muestras promediadas o sin promediar (*bypass*). Esto permite realizar la media aritmética de los dieciséis canales de muestreo de forma simultánea. El número de capturas a promediar (*n*) es igual para los dieciséis canales y la opción de *bypass* ofrece la posibilidad de obtener las muestras en crudo.

Dado que el IP de Cálculo de Promedio es el último de la cola de procesamiento, este debe empaquetar la información procesada, es decir, debe crear paquetes de N tramas procesadas. Esto permite inferir la cantidad de información a guardar por acceso a memoria. Por lo tanto, se declara un nuevo registro AXI4-Lite dedicado al valor de N.

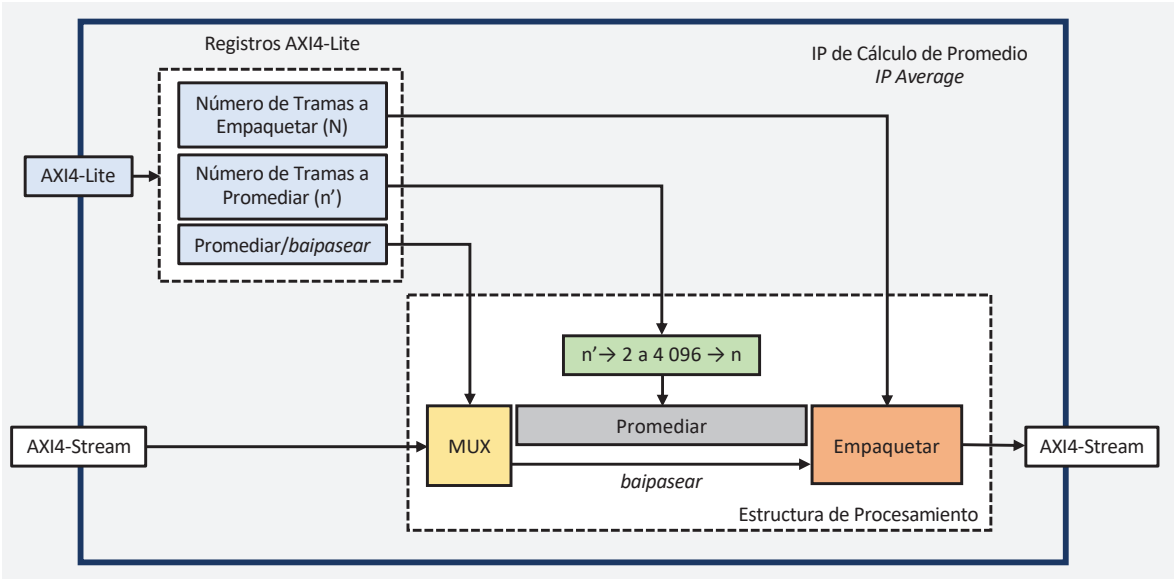

Figura 60. Diagrama de bloques del IP de Cálculo de Promedio.

A continuación, se describe la estructura de procesamiento, la interfaz AXI4-Lite y la integración del IP de Cálculo de Promedios.

5.2.3.1. Estructura de procesamiento

La estructura de procesamiento del IP de Cálculo de Promedios debe promediar o hacer un *bypass* de las tramas de muestras ofrecidas por el IP de Marca de Tiempo Real y empaquetar la información procesada. Por lo tanto, esta estructura incluye la siguiente funcionalidad:

- Recibe por medio de una interfaz AXI4-Stream básica (*TDATA*, *TREADY* y *TVALID*) las tramas de muestras con la marca de tiempo *Unix*. Esta posee un ancho de 512 bits.
- Permite elegir entre promediar (1) o hacer un *bypass* (0) de las tramas de muestras.
- Permite indicar el número de tramas a promediar (*n'*).
- Permite configurar el número de tramas de muestras procesadas por paquete (*N*).
- Con el objeto de mantener el flujo de los datos, envía por AXI4-Stream las tramas con las muestras procesadas. Esta interfaz AXI4-Stream incluye la señal *TLAST*, que permite indicar el final de un paquete.
- Sitúa a nivel alto la señal *TLAST* cada *N* tramas. Esto permite crear paquetes de *N* tramas de muestras procesadas.

En el caso de elegir realizar el promedio:

- Limita el número de tramas a promediar indicado por AXI4-Lite (*n'*) a un número potencia de dos entre 2 y 4 096 (*n*).
- Extrae de las *n* tramas las muestras por canal.
- Acumula en un banco de dieciséis registros internos las *n* muestras por canal.

Este banco de registros internos soporta la recopilación de hasta 4 096 muestras por canal. Cada registro tiene un ancho de 36 bits, ya que las muestras digitalizadas poseen una resolución de 24 bits y están codificadas en complemento a dos *(4)*.

$$\textbf{Ancho de los Registros Internos } = \frac{\log(2^{24} \cdot 4096)}{\log(2)} = 36 \qquad (4)$$

Tras recibir las *n* tramas:

- Desplaza a nivel de bits dichos registros (el número de bits a desplazar depende de *n*) para obtener el promedio de las *n* muestras de los dieciséis canales.
- Crea una trama con las muestras promediadas de los dieciséis canales de captura.
- Insertar a la trama con las muestras promediadas la marca de tiempo Unix de la última trama de muestras promediada.

En el caso de elegir realizar un *bypass*:

– Mantiene el flujo de los datos. Dicho de otro modo, conecta directamente la entrada AXI4-Stream a la salida AXI4-Stream.

En el Código 11, se observa la descripción de dicha estructura en C++, la cual será exportada a Verilog a través de Vivado HLS e instanciada en el bloque *hardware* final.

En el Código 16, se observa la descripción en C++ de esta estructura, la cual ha sido sintetizada a una descripción RTL Verilog a través de Vivado HLS. Esta última se *instancia* en el bloque *hardware* final. En la Figura 61 se muestra el formato de las tramas procesadas.

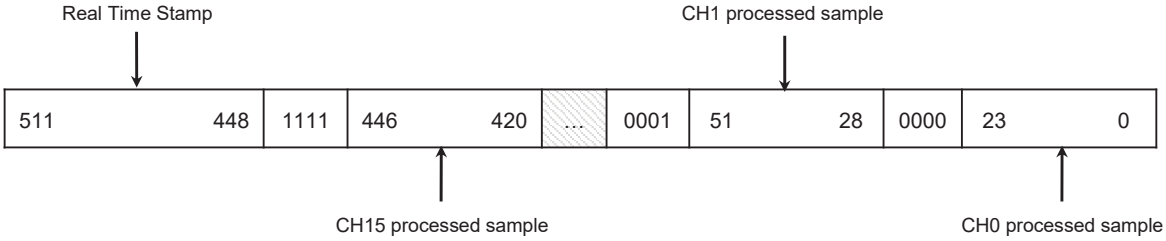

Figura 61. Formato de las tramas de muestras procesadas.

5.2.3.2. Interfaces de comunicación

Con la finalidad de configurar el IP de Cálculo de Promedios desde una aplicación *software*, se dota a dicho bloque *hardware* de una interfaz AXI4-Lite que permite:

i. Elegir si promediar o hacer el *bypass* de las tramas (1 bit): (0) *bypass*, (1) promediar.

ii. Indicar el número de tramas a promediar (*n'*) (32 bits).

iii. Indicar el número de tramas procesadas por paquete (*N*) (32 bits).

La configuración del IP de Cálculo de Promedios precisa de 65 bits, es decir, tres registros AXI4-Lite. Por tanto, en Vivado HLS se crea una función C++ que permite escribir en tres registros de control de 32 bits por AXI4-Lite y que retorna la selección de si promediar o hacer el *bypass*, el número de tramas a promediar (*n'*) y el número de tramas procesadas por paquete (*N*).

En el Código 17, se describe dicha interfaz y en la Figura 62, Figura 63 y Figura 64, se observa la distribución de dichos parámetros de configuración en los registros AXI4-Lite.

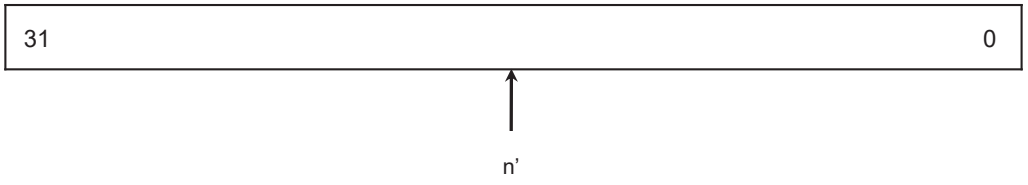

Figura 62. Registro de configuración del número de tramas a promediar (n'). Dirección 0h10.

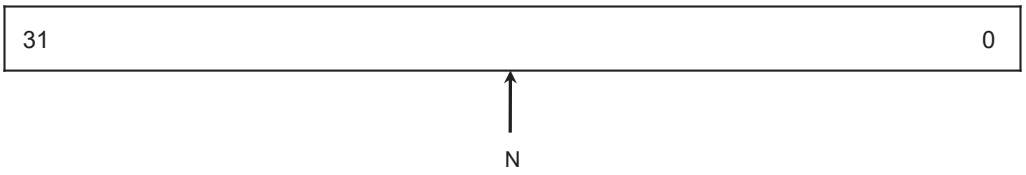

Figura 63. Registro de configuración del número de tramas procesadas por paquete. Dirección 0h14.

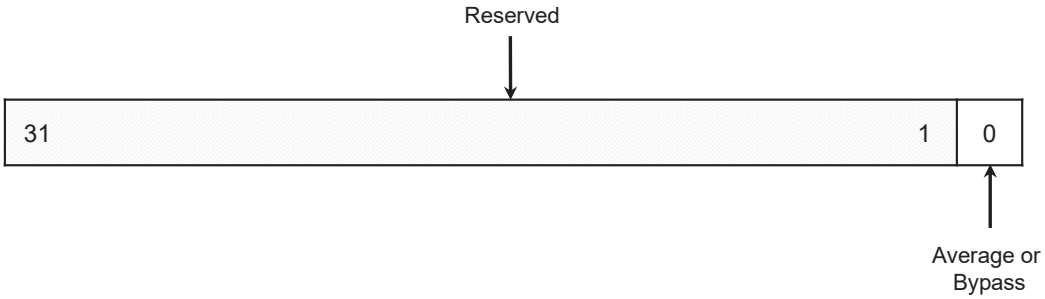

Figura 64. Registro de configuración. Selección de promedio o bypass. Dirección 0h18.

5.2.3.3. Integración del bloque

En el Código 18, se expone la descripción Verilog del IP de Cálculo de Promedios, que integra la estructura de procesamiento y la interfaz AXI4-Lite de configuración descrita, y, en la Figura 65, se muestran las interfaces de E/S de dicho bloque *hardware*. Este IP ha sido sintetizado para operar a una frecuencia de 100MHz.

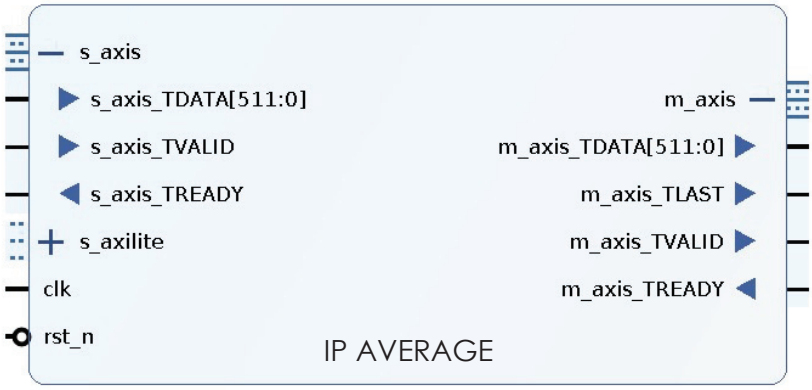

Figura 65. IP de Cálculo de Promedios.

Tabla 12. Descripción de las E/S del IP de Cálculo de Promedios.

Descripción de las E/S del IP de Cálculo de Promedios	
clk	Señal del reloj.
rst_n	Señal de reseteo a nivel bajo.
s_axilite	Entrada AXI4-Lite de configuración del IP.
s_axis	Entrada AXI4-Stream. Entrada de tramas con las marcas de tiempo real.

m_axis	Salida AXI4-Stream. Salida de tramas de muestras procesadas.

Por último, se exponen los recursos *hardware* consumidos (Figura 66), se verifica la funcionalidad del IP y se miden los tiempos de respuesta.

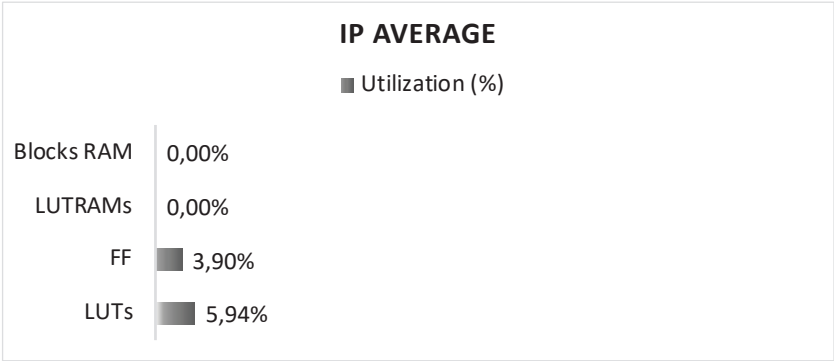

Figura 66. Recursos hardware consumidos por el IP de Cálculo de Promedios.

En la Figura 67 y Figura 68, se observa la respuesta del IP de Cálculo de Promedios. En este caso, el IP ha sido configurado para realizar el promedio de las tramas de muestras ($s = 1$), promediar cada dos tramas ($n' = 2$) y empaquetar cada cuatro tramas procesadas ($N = 4$). La señal de reloj (*clk*) del bloque *hardware* es de 100MHz.

Ante esta configuración por AXI4-Lite, el bloque *hardware* responde correctamente. Como se aprecia, el IP promedia cada dos y genera la señal *TLAST* cada cuatro tramas de muestras procesadas.

Asimismo, el bloque *hardware* realiza el cálculo del promedio de forma acertada (tarda 30ns tras recibir las *n* tramas de muestras) y añade correctamente la marca de tiempo Unix a la trama con las muestras procesadas.

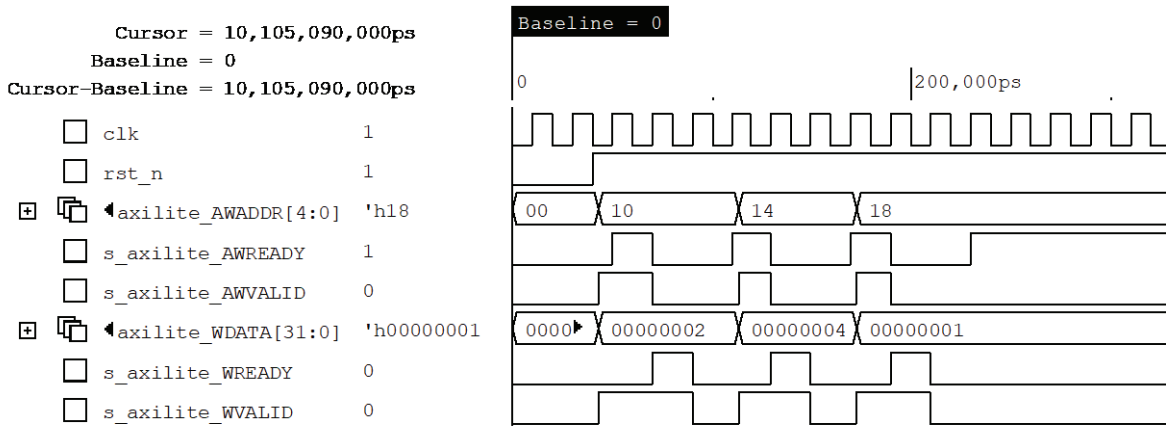

Figura 67. Verificación del IP de Cálculo de Promedios. Configuración.

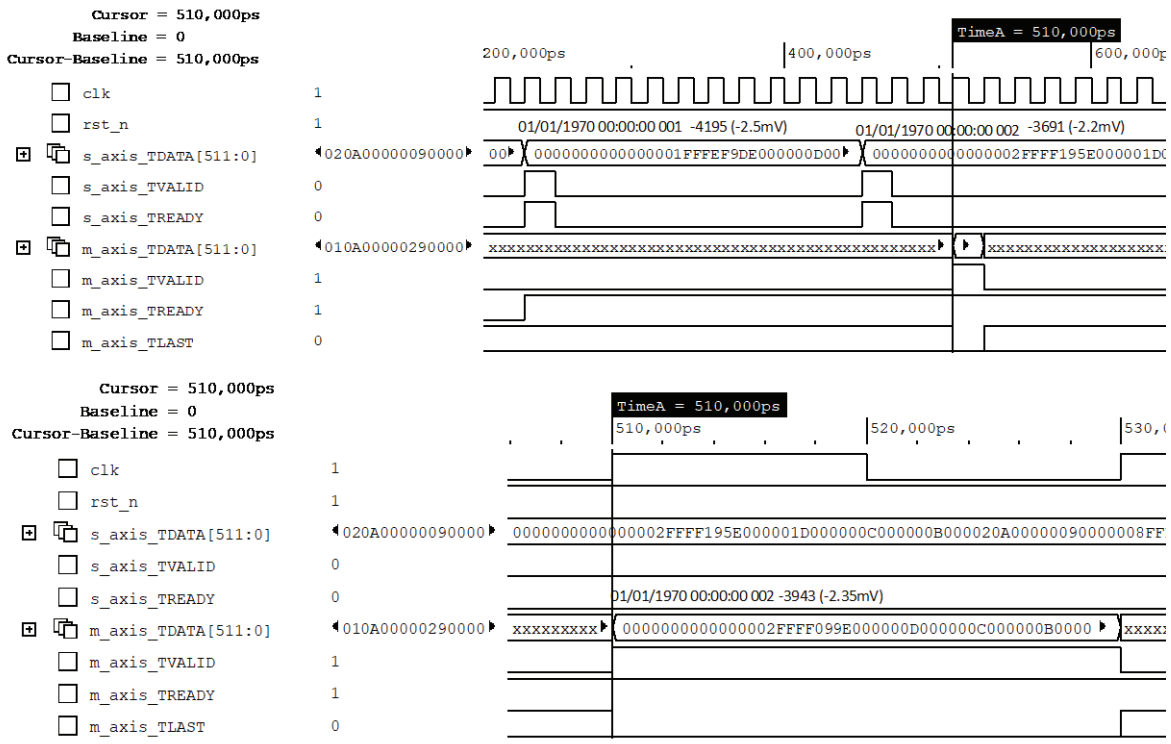

Figura 68. Verificación del IP de Cálculo de Promedios. Cálculo del promedio.

5.2.4. IP AXI DE ACCESO DIRECTO A MEMORIA

Con la finalidad de guardar en memoria *On-Chip* (OCM) a alta velocidad las muestras procesadas y sin interrumpir la actividad del ARM Cortex A9, se utiliza un *Direct Memory Access* o DMA.

Xilinx ofrece el IP AXI *Direct Memory Access* [50]. Este IP proporciona transferencias directas a alta velocidad entre periféricos AXI4-Stream y memoria *On-Chip* por medio de interfaces AXI de Alto Rendimiento (AXI_HP). El IP AXI DMA permite operar en modo Registro Directo o Simple DMA. Este modo ofrece transacciones DMA simples (*AXI4-Stream to Memory Map* o *Memory Map to AXI4-Stream*) con un reducido consumo de recursos *hardware*. Este IP de Xilinx admite datos con un ancho de 8, 16, 32, 64, 128, 512 y 1 024 y soporta direcciones de memoria de hasta 64 bits. Por último, el AXI DMA permite enviar paquetes de hasta 67 108 863 *bytes* y, en el modo Simple DMA *AXI4-Stream to Memory Map*, precisa que se le indique por medio de la señal *TLAST* el final de un paquete (tras recibir *TLAST*, envía el paquete a memoria) (Figura 69).

Channel	Clock Frequency (MHz)	Bytes Transferred	Total Throughput (MB/s)	Percent of Theoretical
MM2S(2)	100	10,000	399.04	99.76
S2MM(3)	100	10,000	298.59	74.64

Figura 69. Velocidades de transacción del IP AXI DMA [50].

Dada las características, se utiliza dicho *core* AXI DMA para almacenar en memoria *On-Chip* la información procesada. Por tanto, se configura el IP de la siguiente forma (Figura 70):

- Modo Simple DMA. Permite realizar transferencias *AXI4-Stream to Memory Map* indicando la dirección destino y el tamaño en *bytes* de la información a guardar. Esto reduce la complejidad del proceso de guardado y el consumo de recursos *hardware*.
- Profundidad de 524 288 *bytes*. Permite enviar paquetes de hasta 8 192 tramas procesadas.
- Ancho de direcciones de 32 bits. La memoria *On-Chip* trabaja con direcciones de 32 bits.
- Ancho de datos de 512 bits. Ancho de las tramas con las muestras procesadas.
- Deshabilitar la opción *Memory Map to AXI4-Stream*. Esta opción no es necesaria.

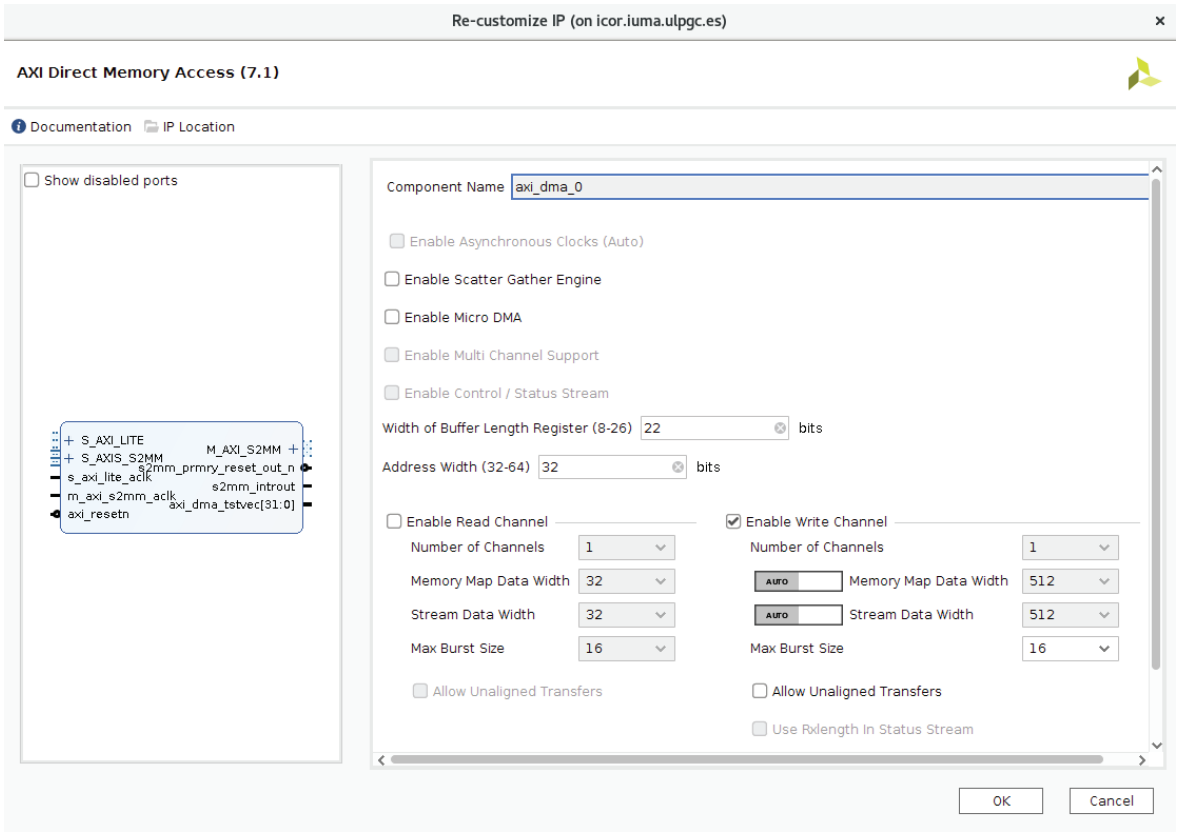

Figura 70. Configuración del IP AXI Direct Memory Access de Xilinx.

5.2.5. IP FIFO GENERATOR

Con el objeto de procesar a la frecuencia nominal (100MHz) las capturas procesadas por el IP Interfaz AD7768, los datos transferidos por la interfaz AXI4-Stream de este deben cruzar de dominio de reloj, desde el dominio de reloj del ADC (8.192MHz) a la frecuencia nominal citada (100 MHz). Por esta razón, se *instancia* a la salida de dicho IP un bloque FIFO con las entradas y salidas conectadas a los respectivos dominios de reloj: 8.192 MHz de escritura (entrada) y 100 MHz de lectura

(salida). Además, con la finalidad de recoger las tramas procesadas mientras el DMA está ocupado, se *instancia* otro bloque FIFO al final de la cadena de procesamiento (entrada del DMA).

Xilinx ofrece el IP FIFO *Generator*. Este IP permite almacenar y recuperar de forma ordenada la información, funcionando hasta 500MHz y con un consumo de recursos *hardware* reducido (*slices* y BRAM). Soporta interfaces FIFO Nativas, AXI4 *Memory Mapped* y AXI4-Stream de lectura y escritura, y dominios de reloj independientes. Además, admite anchos de palabra de 0 a 512 *bytes* y una profundidad de 16 a 131 072 posiciones [51].

Dada las características, se utiliza uno de estos *cores* FIFO para el cruce de dominios de reloj descrito y otro para recopilar las muestras procesadas.

Por lo tanto, el IP FIFO *Generator* (Figura 71) dedicado al cruce de dominios de reloj se configura con interfaces AXI4-Stream de lectura y escritura con relojes independientes, y se ajusta con un ancho de palabra de 512 bits o 64 *bytes* (ancho de las tramas de muestras) y con una profundidad de 16. Esta profundidad de la FIFO es suficiente, puesto que los datos son leídos 12.2 veces más rápido que escritos (100 MHz frente a 8.192 MHz).

El *core* FIFO *Generator* destinado a recolectar las tramas procesadas, también se configura con interfaces AXI4-Stream de entrada y salida, y con un ancho de palabra de 512 bits o 64 *bytes* (ancho de las tramas procesadas). Sin embargo, dichas interfaces, esta vez, deben operar a la misma frecuencia (100 MHz). En cuanto a la profundidad, se opta por definir una de 4 096. Esto permite acumular hasta 8 ms de captura en el supuesto de muestreo más desfavorable (256 kSP/s).

5.3. Bloques *hardware* de control

A continuación, se expone el diseño de los IPs de *control* del sistema, incluyendo el control de la UPS, el control de ganancias, el control de modo PIN, control por SPI o el control del bloque por IIC del RTC.

5.3.1. IP AXI GPIO de Xilinx

El IP AXI GPIO de Xilinx se utiliza de forma recurrente como bloque *hardware* de control. Por consiguiente, a continuación, se exponen brevemente sus características.

El *core* AXI GPIO de Xilinx [52] permite conectar una interfaz AXI4-Lite con una interfaz de entrada o salida de propósito general que utilice señales independientes. Este *core* soporta hasta dos canales configurables. Dicha configuración permite especificar el ancho del canal de 1 a 32 bits, configurar los pines GPIO como triestado, solo salida o solo entrada; y realizar el *reset* bit a bit al valor por defecto.

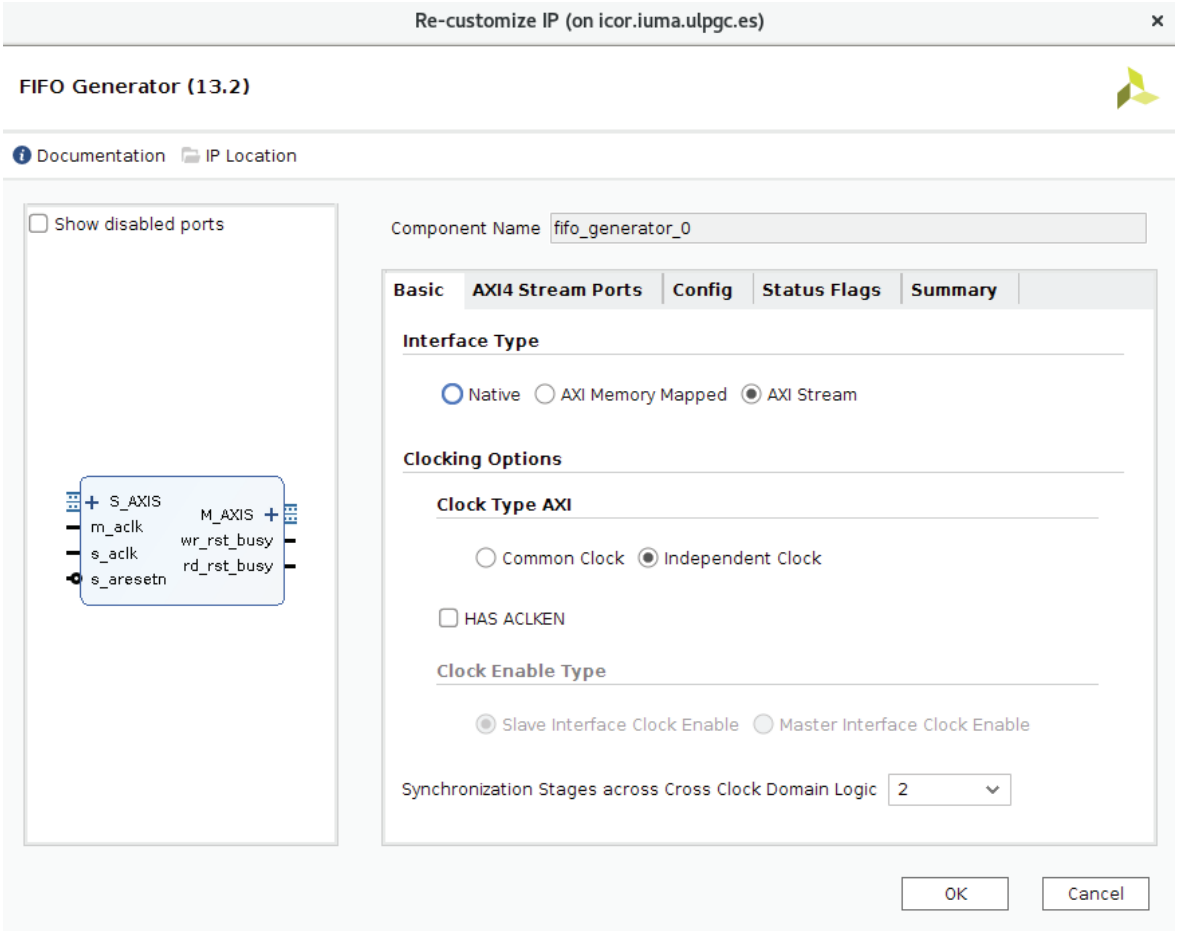

Figura 71. Configuración del IP FIFO Generator dedicado al cruce de dominios de reloj.

5.3.2. IP DE CONTROL DE LA UPS

Con el objeto de controlar la UPS del módulo, así como supervisar el estado de la red externa (AC *Fail*) que lo alimenta, desde el *software*, se utiliza un IP AXI GPIO de Xilinx [52].

Este IP se configura de la forma siguiente (Figura 72):

- Se habilita *dual channel*. El canal GPIO se dedica a la supervisión de la señal AC *Fail* y el canal GPIO2 al control de las tensiones de 7.3 V (AMP) y 5.4 V (ADC).
- GPIO en modo *all inputs* y con un bit de ancho. La señal AC *Fail* debe ser leída de un pin del conector FMC. Este GPIO permite leer el estado de dicho pin (0: no fallo, 1: fallo).
- GPIO2 en modo *all output* y con dos bits de ancho. El control de la tensión de 7.3 V (AMP) se realiza por medio de un pin del FMC y, el de la tensión de 5.4 V (ADC), por otro. Este GPIO2 permite la escritura en dichos pines (Tabla 13).
- Valor por defecto del GPIO2 a cero. Por defecto, ambos deben estar en *OFF*.

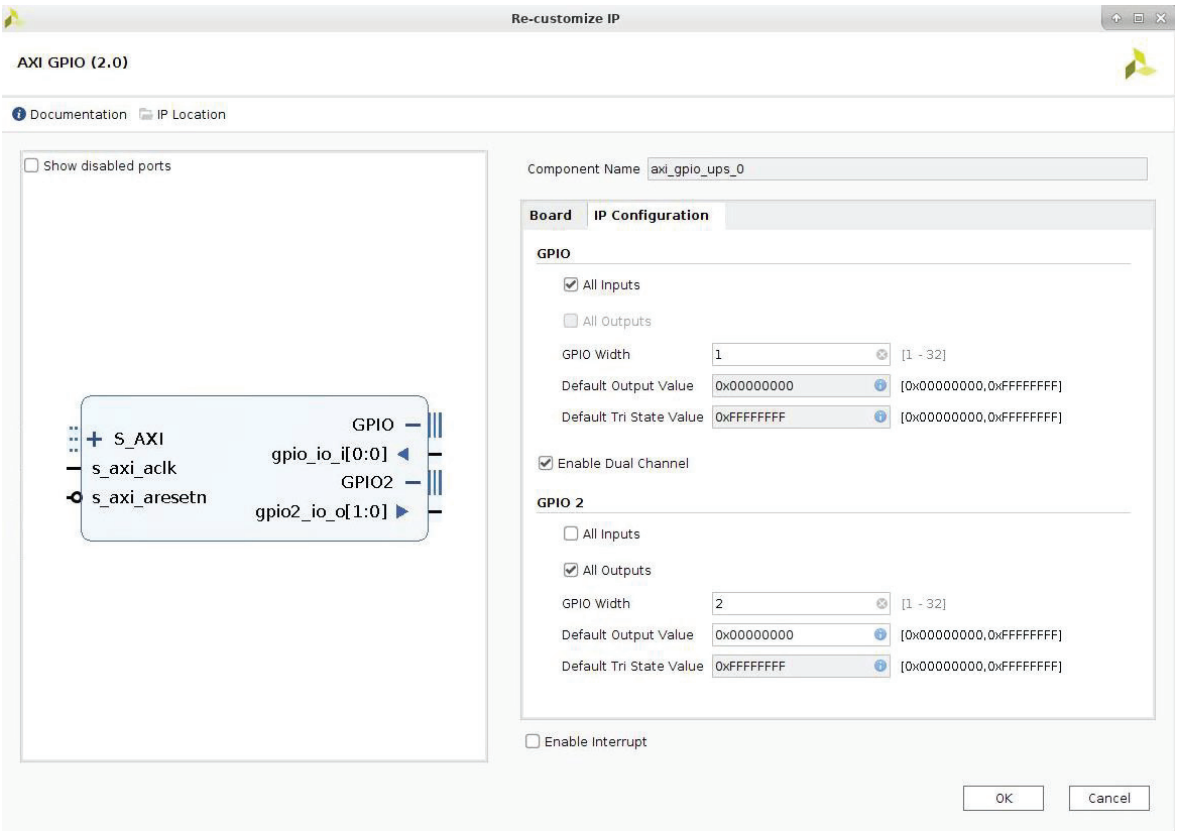

Figura 72. Configuración del IP AXI GPIO de la UPS.

Tabla 13. Control de las tensiones de 7.3 V (AMP) y 5.4 V(ADC) desde el GPIO2.

GPIO2[1]	GPIO2[0]	7.3V (AMP)	5.4V (ADC)
0	0	*OFF*	*OFF*
0	1	*OFF*	*ON*
1	0	*ON*	*OFF*
1	1	*ON*	*ON*

5.3.3. IP DE CONTROL DE GANANCIAS

Con la finalidad de seleccionar las ganancias de los amplificadores de la *etapa de amplificación* desde una aplicación *software*, se diseña un bloque *hardware* que, por medio de una interfaz AXI4-Lite, escriba en los pines del FMC dedicados a dicho propósito.

En este caso, dado que a través de tres pines del FMC (*A1*, *A0* y *EN*) se selecciona el factor de amplificación de dos canales, se requiere la escritura en 24 pines del FMC (ocho grupos de tres señales: *A1*, *A0* y *EN*) para seleccionar las ganancias de amplificación de dieciséis canales.

Por lo tanto, en Vivado HLS se crea una descripción de alto nivel en C++ que permite escribir en un registro de 32 bits por AXI4-Lite y que, a través de 24 salidas de un bit, genera los ocho

selectores de ganancia (Código 2). Dicha descripción de alto nivel C++ ha sido sintetizada a una descripción RTL Verilog que opera a una frecuencia de 100 MHz (Figura 74).

```
#include <ap_int.h>
void GainControl(
        ap_uint<32> s_axi,
        ap_uint<1> &GR1A1,
        ap_uint<1> &GR1A0,
        ap_uint<1> &EN1,
        ...
        ap_uint<1> &GR8A1,
        ap_uint<1> &GR8A0,
        ap_uint<1> &EN8){
    // Directives.
    #pragma HLS INTERFACE s_axilite port=s_axi
    #pragma HLS INTERFACE ap_none port=GR1A1
    #pragma HLS INTERFACE ap_none port=GR1A0
    #pragma HLS INTERFACE ap_none port=EN1
    ...
    #pragma HLS INTERFACE ap_none port=GR8A1
    #pragma HLS INTERFACE ap_none port=GR8A0
    #pragma HLS INTERFACE ap_none port=EN8
    #pragma HLS INTERFACE ap_ctrl_none port=return
    // GAIN 1:
    GR1A1 = s_axi.range(0, 0);
    GR1A0 = s_axi.range(1, 1);
    EN1 = s_axi.range(2, 2);
    ...
    // GAIN 8:
    GR8A1 = s_axi.range(28, 28);
    GR8A0 = s_axi.range(29, 29);
    EN8 = s_axi.range(30, 30);
}
```

Código 2. IP Control de Ganancias.

Las salidas de este bloque *hardware* se conectan a los 24 pines del conector FMC sujetos a seleccionar el factor de amplificación de los amplificadores según la Tabla 14. En la Figura 73, se observa el registro AXI4-Lite con la distribución de las distintas señales a controlar.

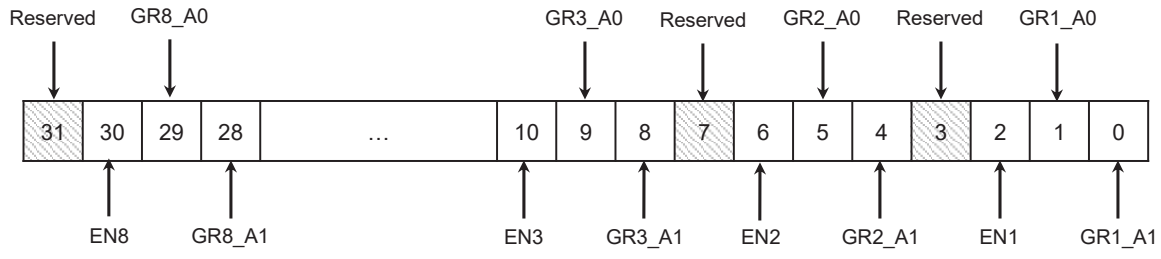

Figura 73. Registro de selección de ganancias de los amplificadores. Dirección 0h10.

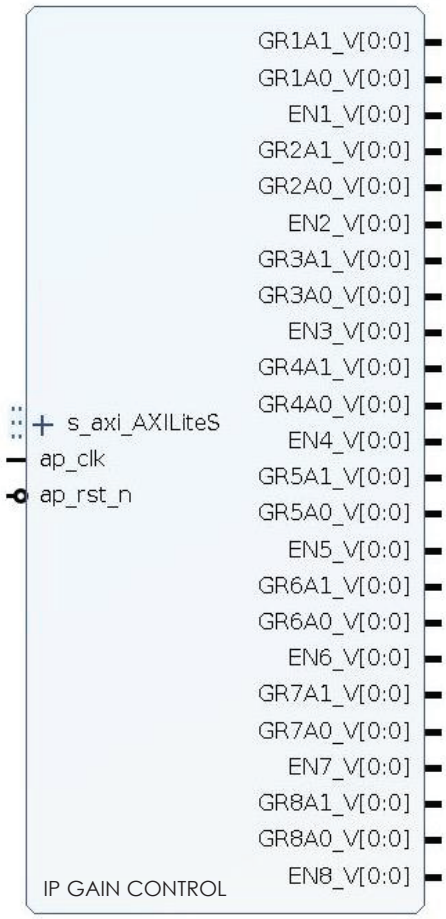

Figura 74. IP de Control de Ganancias.

Tabla 14. Descripción de las E/S del IP de Control de Ganancias.

Descripción de las E/S del IP de Control de Ganancias

clk	Señal del reloj.
rst_n	Señal de *reset*. Activa a nivel bajo.
s_axi_AXILiteS	Interfaz de configuración AXI4-Lite.
GR1A1, GR1A0 y EN1	Selección de ganancia de amplificación de: Canal 0 y Canal 1.
GR2A1, GR2A0 y EN2	Selección de ganancia de amplificación de: Canal 2 y Canal 3.
GR3A1, GR3A0 y EN3	Selección de ganancia de amplificación de: Canal 4 y Canal 5.
GR4A1, GR4A0 y EN4	Selección de ganancia de amplificación de: Canal 6 y Canal 7.
GR5A1, GR5A0 y EN5	Selección de ganancia de amplificación de: Canal 8 y Canal 9.
GR6A1, GR6A0 y EN6	Selección de ganancia de amplificación de: Canal 10 y Canal 11.
GR7A1, GR7A0 y EN7	Selección de ganancia de amplificación de: Canal 12 y Canal 13.
GR8A1, GR8A0 y EN8	Selección de ganancia de amplificación de: Canal 14 y Canal 15.

Por último, en la Figura 75, se verifica la funcionalidad del IP sintetizado a nivel RTL.

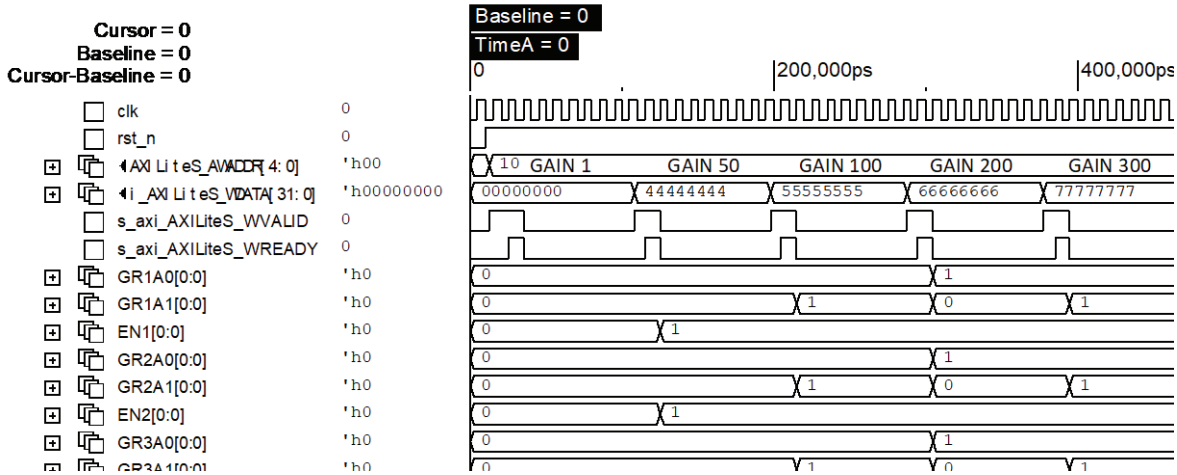

Figura 75. Verificación del IP de Control de Ganancias.

5.3.4. IP DE CONTROL EN MODO PIN

Con el fin de configurar y controlar los AD7768 por modo PIN desde la aplicación *software*, se diseñan o utilizan los siguientes IPs:

- Se diseña un IP que, por medio de una interfaz AXI4-Lite, permita escribir en los pines del FMC dedicados a la configuración por PIN de los ADC.
- Se utilizan dos IP AXI GPIO de Xilinx, uno para controlar el *reset* y otro la sincronización de los AD7768 en modo PIN. Estos GPIO permiten escribir en los pines *start* y *reset* del FMC a través de dos interfaces AXI4-Lite. Si bien, se podría haber utilizado un único IP para esta doble funcionalidad. No obstante, se opta por separarlas para evitar interferencias entre las señales de sincronización y *reset*.

La configuración por PIN requiere la escritura en 16 pines del FMC (Configuración por PIN). Por lo tanto, en Vivado HLS se crea una descripción de alto nivel en C++ que permite escribir en un registro de 32 bits por AXI4-Lite y que, a través de 16 salidas de un bit, genera la configuración por PIN de los AD7768 (Código 3). Dicha descripción de alto nivel C++ ha sido sintetizada a una descripción RTL Verilog que opera a una frecuencia de 100 MHz (Figura 77).

```
#include <ap_int.h>
void PinModeControl(
        ap_uint<32> s_axi,
        ap_uint<1> &PIN_or_SPI,
        ap_uint<1> &ST0_1, // ST1A
        ...
        ap_uint<1> &MODE3,
        ap_uint<1> &FILTER){

    // Directives.
```

```
    #pragma HLS INTERFACE s_axilite port=s_axi
    #pragma HLS INTERFACE ap_none port=PIN_or_SPI
    #pragma HLS INTERFACE ap_none port=ST0_1
    ...
    #pragma HLS INTERFACE ap_none port=MODE3
    #pragma HLS INTERFACE ap_none port=FILTER
    #pragma HLS INTERFACE ap_ctrl_none port=return
    // Implementation.
    PIN_or_SPI = s_axi.range(0, 0);
    ST0_1 = s_axi.range(1, 1);
    ...
    MODE3 = s_axi.range(14, 14);
    FILTER = s_axi.range(15, 15);
}
```

Código 3. IP de Control por PIN.

Las salidas de este bloque *hardware* se conectan a los 16 pines del conector FMC sujetos a la configuración por PIN de los conversores según la Tabla 15.

En la Figura 76, se observa la distribución de los parámetros de configuración por PIN en el registro AXI4-Lite, en la Figura 77 se muestran las interfaces de E/S y, por último, en la Figura 78, se verifica la funcionalidad del IP sintetizado a nivel RTL.

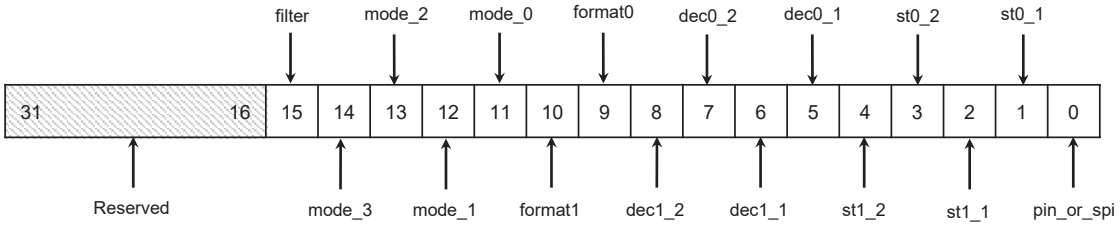

Figura 76. Registro de configuración por PIN. Dirección 0h10.

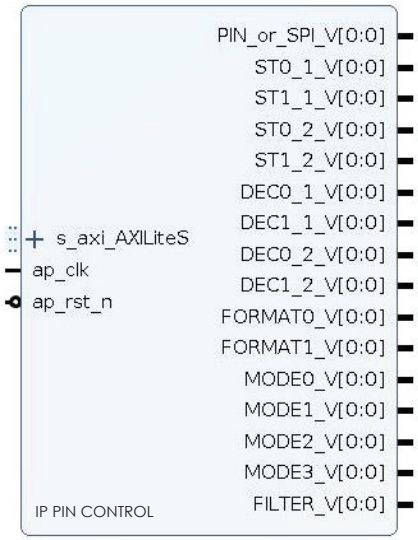

Figura 77. IP de Control por PIN.

Tabla 15. Descripción de las E/S del IP de Control por PIN.

Descripción de las E/S del IP de Control por PIN

clk	Señal del reloj.
rst_n	Señal de *reset*, activa a nivel bajo.
s_axi_AXILiteS	Interfaz de configuración AXI4-Lite.
PIN_or_SPI	Selección del modo de configuración de los ADC: PIN (0) o SPI (1).
ST0_1 y ST1_1	Selección de canales en *standby* del ADC 1.
ST0_2 y ST1_2	Selección de canales en *standby* del ADC 2.
DEC0_1 y DEC1_1	Selección de la tasa de decimación del ADC 1.
DEC0_2 y DEC1_2	Selección de la tasa de decimación del ADC 2.
FORMAT0 y FORMAT1	No conectar. Selección del formato de salida de datos de los ADC.
MODE0, MODE1, MODE2 y MODE3	Selección del modo de conversión (*standard* o *one-shot*), divisor de MCLK y modo de *power de los ADC.*
FILTER	Selección del filtro de los ADC: *wideband* (0) o *sinc5* (1).

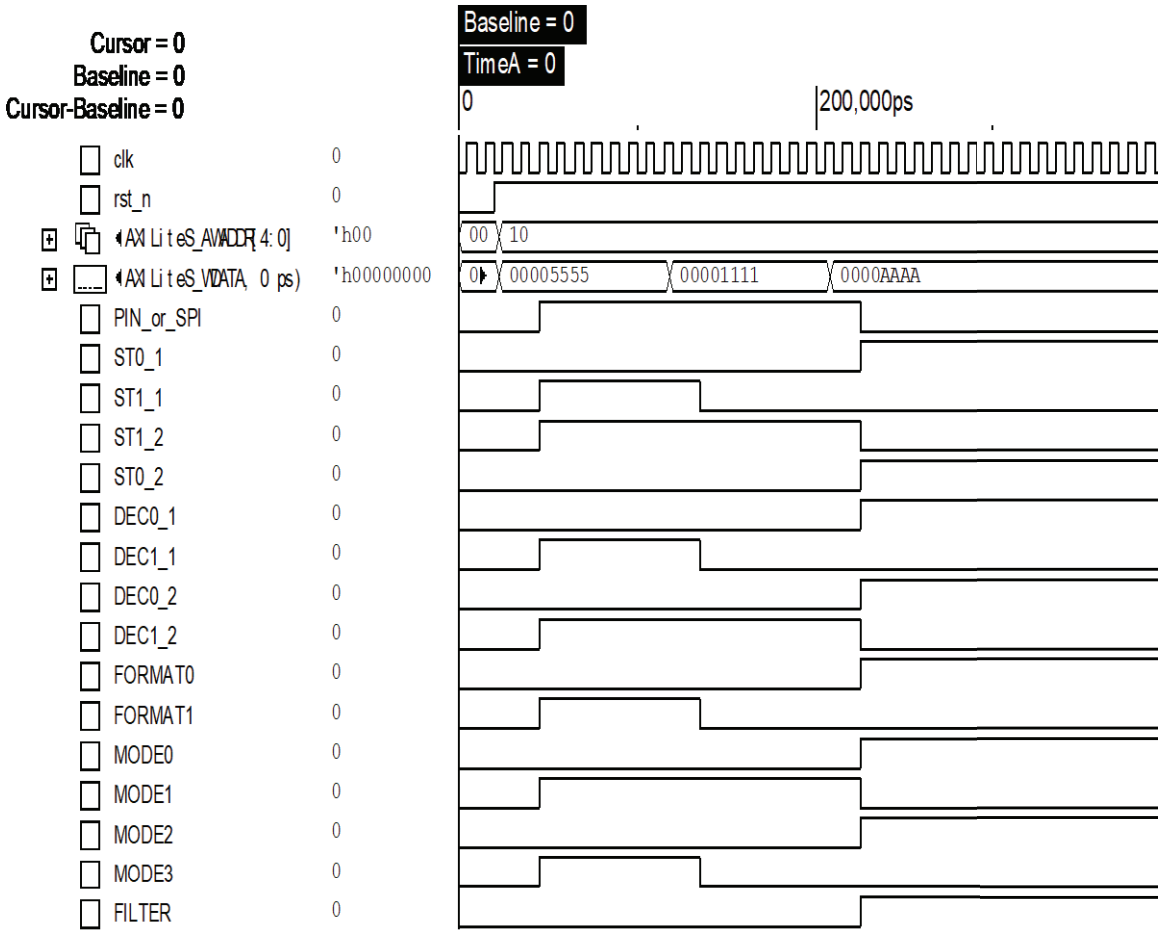

Figura 78. Verificación del IP de Control por PIN.

Los AXI GPIO se configuran de la siguiente forma (Figura 79 y Figura 80):

– Deshabilitar *dual channel*. En ambos casos, únicamente es preciso un canal GPIO.

– GPIO en modo *all output* y con un bit de ancho. El control del *reset* y de la sincronización de los ADC es por medio de un pin del FMC (*reset* y *start*). Estos GPIO permiten generar los estados correspondientes en el pin *reset* y pin *start*, respectivamente.

– Valor por defecto de los GPIO a uno. El *reset* y el *start* son activos a nivel bajo.

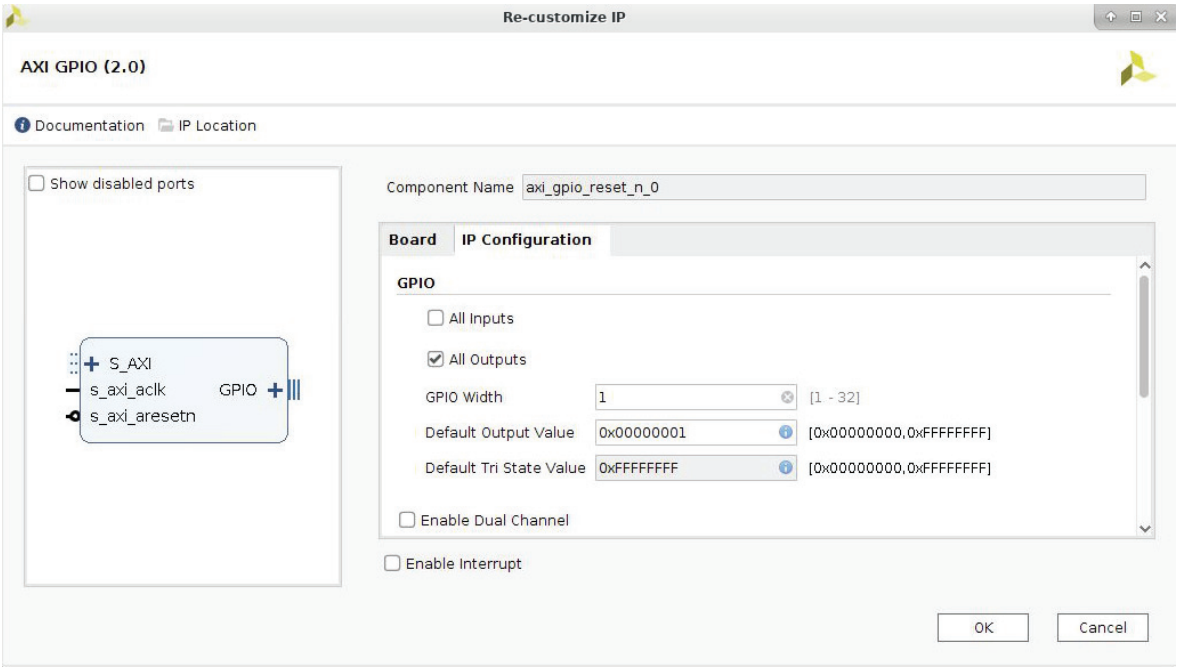

Figura 79. Configuración del AXI GPIO de control de reset por PIN.

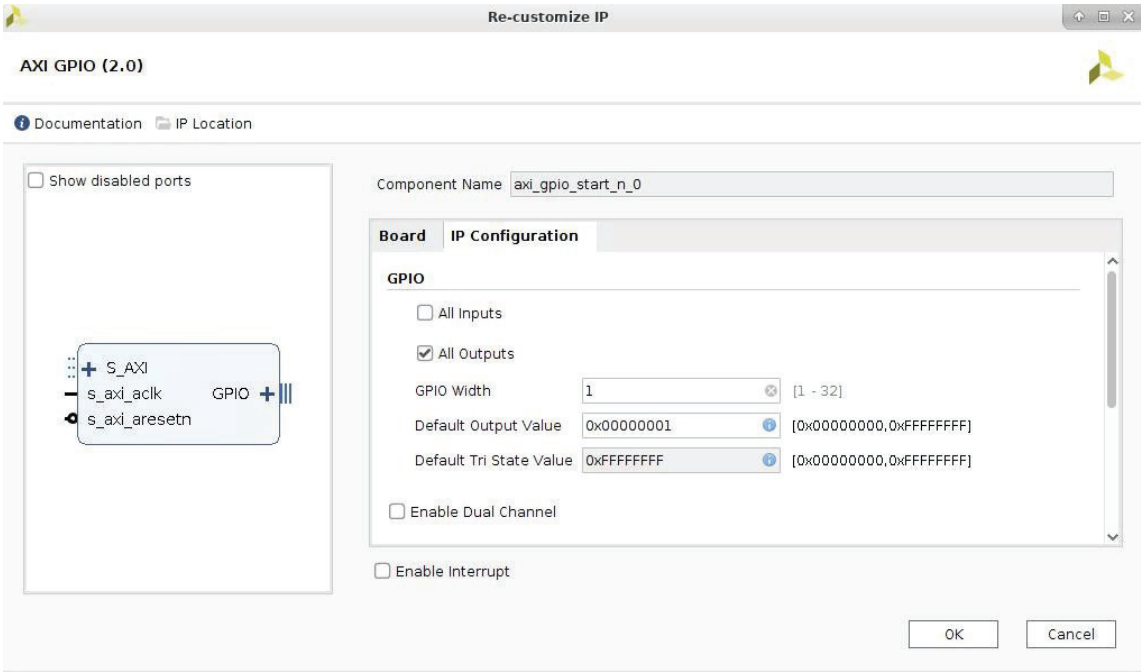

Figura 80. Configuración del AXI GPIO de control de sincronización por PIN.

5.3.5. IP DE CONTROL POR SPI

Con la finalidad de configurar y controlar por SPI, desde una aplicación *software* los dos ADC AD7768, se utiliza el IP AXI *Quad Serial Peripheral Interface* de Xilinx [53].

Este *core* permite conectar una interfaz AXI4 con un dispositivo SPI esclavo que soporte el protocolo *Standard*, *Dual* o *Quad*. El IP admite la interfaz de usuario AXI4, la cual permite transacciones en ráfaga, así como la AXI4-Lite. Además, permite configurar la frecuencia, fase y polaridad de la señal de reloj SPI (*sclk*), definir un ancho de transacción de 8, 16 ó 32 bits y, en el modo *Standard*, permite comportarse como maestro y comunicarse con hasta 32 esclavos seleccionables a nivel bajo.

Dadas las características de la Tarjeta AD7768 AMP DIFF 16 CH, se utilizan dos IP AXI *Quad-SPI* de Xilinx (uno por ADC). Ambos IP se configuran de la forma siguiente:

- Deshabilitar el modo *only-read* o *eXecute In Place* (XIP).
- Interfaz AXI4-Lite o deshabilitar el modo *Performance*.
- Modo *Standard*. La interfaz SPI de los AD7768 es *Standard* (*sclk*, *mosi*, *miso* y *ss*).
- Ancho de transacción de 16 bits. Los AD7768 operan con transacciones de 16 bits (un bit R/W, siete bits de dirección y ocho bits de datos).
- Ratio de frecuencia de 16. El IP AXI4 *Quad*-SPI genera la señal de reloj SPI (*sclk*) a partir de una señal de reloj externa. Los AD7768 requieren un *sclk* de al menos 100ns de periodo (10MHz máximo). Dado que la señal de reloj utilizada en el sistema es de 100MHz (PS), con este valor del divisor se obtiene un *slck* de 160ns (6.25MHz).
- Número de esclavos 1. Cada IP AXI *Quad*-SPI se comunica con un ADC.
- Habilitar modo *master*. La FPGA actúa como dispositivo maestro y los ADCs como esclavos.
- Habilitar el uso de FIFO con una profundidad de 16. Permite configurar hasta 16 registros SPI del AD7768 de forma consecutiva.

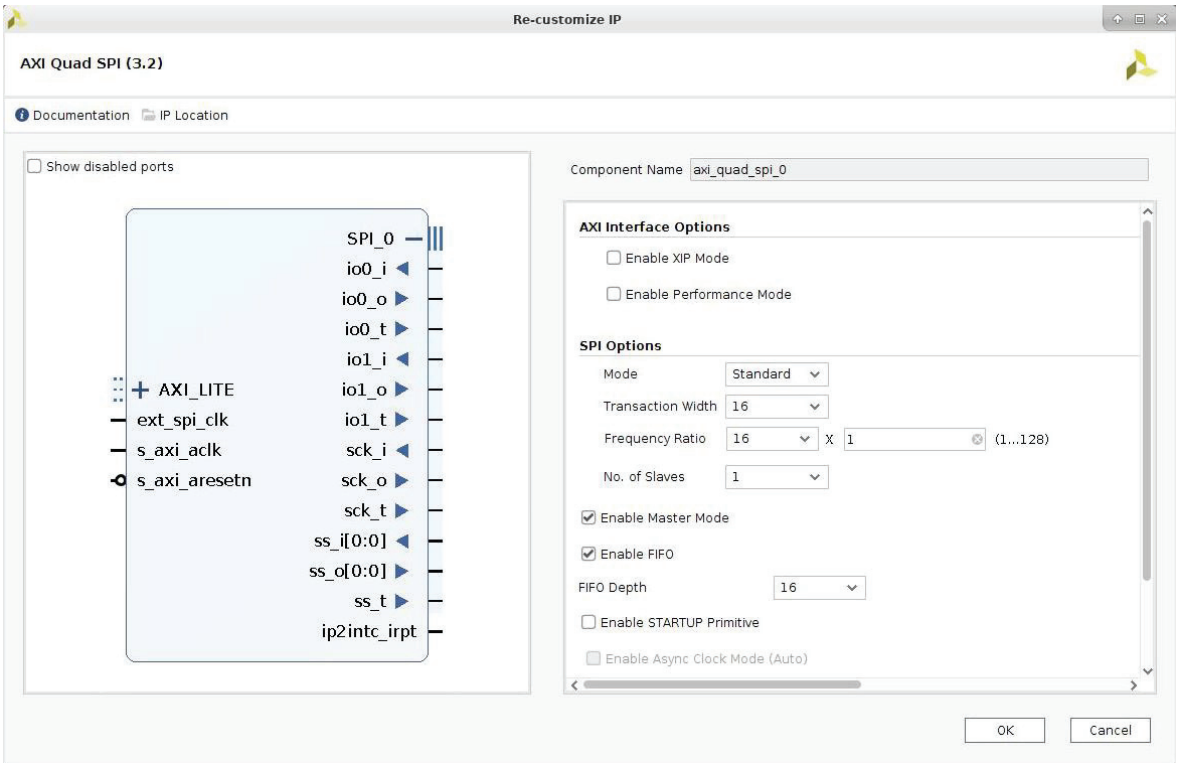

Figura 81. Configuración de ambos IP AXI Quad-SPI de Xilinx.

Las interfaces SPI de ambos *cores* deben conectarse a los pines del FMC conectados a las interfaces SPI de los conversores. Las E/S del IP AXI *Quad*-SPI son tri-estado (Figura 81) y las E/S SPI de los ADC son simples. Por tanto, se realizan las conexiones lógicas de la Tabla 16.

Tabla 16. Conexiones lógicas entre un IP AXI Quad-SPI y una interfaz SPI del AD7768.

Salida IP AXI Quad-SPI		Entrada SPI del AD7768
io0_o o MOSI	→	**SDI o MOSI**
io1_i o MISO	→	**SDO o MISO**
sck_o	→	**SCLK**
ss_o	→	**$\overline{\text{CS}}$**

5.3.6. IP de Selección Master or Slave

Con el propósito de establecer como maestro o esclavo al primer AD7768 de la placa desde el *software*, se utiliza un IP AXI GPIO de Xilinx [52].

Este IP se configura de la forma siguiente (Figura 82):

– Deshabilitar *dual channel*. Únicamente es preciso un canal GPIO.

– GPIO en modo *all output* y con un bit de ancho. El establecimiento del ADC como *master* o *slave* es por medio de un pin del conector FMC. Este GPIO permite escribir un 0 (*slave*) o un 1 (*master*) en dicho pin.

– Valor por defecto del GPIO a cero. Por defecto, el ADC debe ser *slave*.

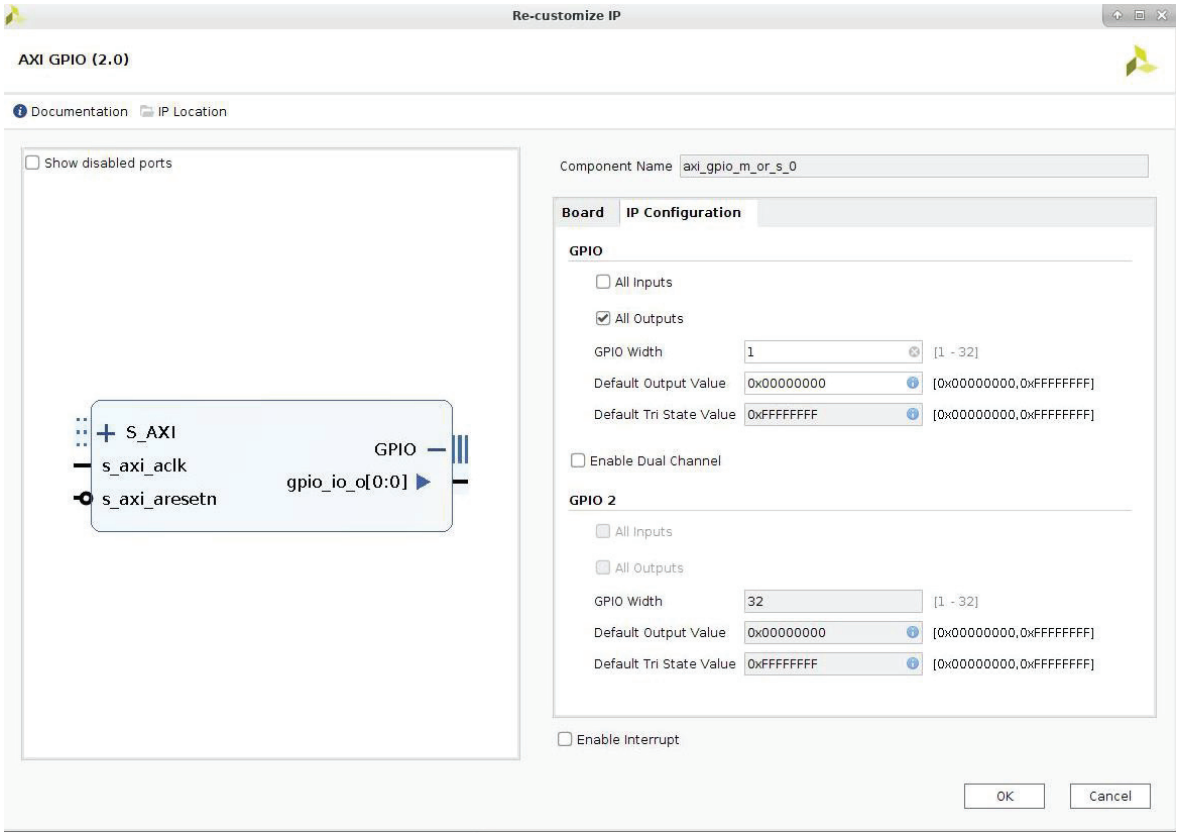

Figura 82. Configuración del AXI GPIO master o slave.

5.3.7. IP DE CONTROL RTC POR IIC

Con el objeto de añadir marcas de tiempo real en segundos sin depender de un servidor *Network Time Protocol* (NTP), se lee la fecha y hora real con resolución de segundos a partir de un *Real Time Clock* (RTC) conectado a un PMOD de la *ZedBoard*.

En este caso, el RTC utilizado es el PMOD *Real Time Clock/Calendar* de *Digilent* que integra el MCP79410 de Microchip [54]. Este RTC permite conocer la fecha y hora real por medio del protocolo *Inter-Integrated Circuit* (IIC). Además, está dotado por un conector J2 de ocho pines, lo que facilita la conexión en uno de los PMOD de la *ZedBoard*. En la Figura 83, se observa los registros que contienen la fecha y hora actual en formato BCD y, en la Figura 84, la secuencia de transferencia de datos por el bus IIC.

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Function	Range	Reset State
Time and Configuration Registers											
00h	ST	10 Seconds			Seconds				Seconds	00-59	00h
01h		10 Minutes			Minutes				Minutes	00-59	00h
02h		12/$\overline{24}$	10 Hour AM/PM	10 Hour	Hour				Hours	1-12 + AM/PM 00 - 23	00h
03h			OSCON	VBAT	VBATEN	Day			Day	1-7	01h
04h			10 Date		Date				Date	01-31	01h
05h			LP	10 Month	Month				Month	01-12	01h
06h	10 Year				Year				Year	00-99	01h

Figura 83. Registros de fecha y hora real del PMOD RTCC de Digilent [54].

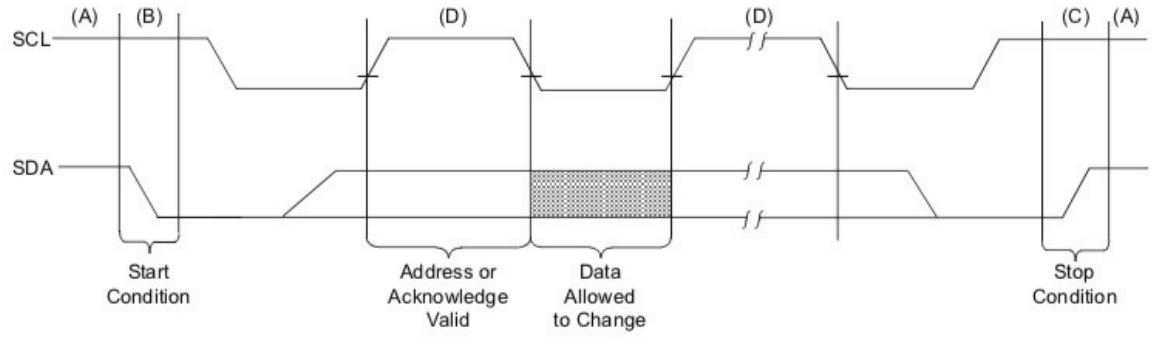

Figura 84. Características del bus IIC del PMOD RTCC. Secuencia de transferencia de datos [54].

Este *Real Time Clock* es un dispositivo IIC esclavo. Por tanto, a fin de conocer el tiempo real en segundos desde una aplicación *software*, se emplea el IP AXI I2C de Xilinx [55].

Este *core* permite conectar una interfaz AXI4-Lite con un gran número de dispositivos *Inter-Integrated Circuit* (IIC). Hay que destacar que, el IP permite realizar operaciones esclavas, maestras y multi-maestras y generar, así como detectar, las señales de START y STOP y el bit *acknowledge* (ACK). Además, permite configurar la frecuencia de la señal de reloj IIC (1KHz - 1MHz) y admite direccionamientos de 7 o 10 bits de ancho.

El IP se configura de la forma siguiente (Figura 85):

- Frecuencia de la señal de reloj IIC (SCL) de 100KHz o modo *Standard*. Los PMOD de la *ZedBoard* están fijados a 3.3 V y el *Real Time Clock* soporta una frecuencia de reloj máxima de 400KHz al estar alimentado por 2.5 V < Vcc < 5.5 V. Esta frecuencia del SCL, además de estar soportada por el RTC, permite transacciones de 12.5KB/s, lo que cumple con los requisitos temporales que exige la transmisión de una marca en segundos.
- Direcciones de 7 bits de ancho. Los registros del PMOD RTCC son de 7 bits de ancho.
- Señal SDA activa a nivel alto. La señal SDA del PMOD RTCC opera a nivel alto.

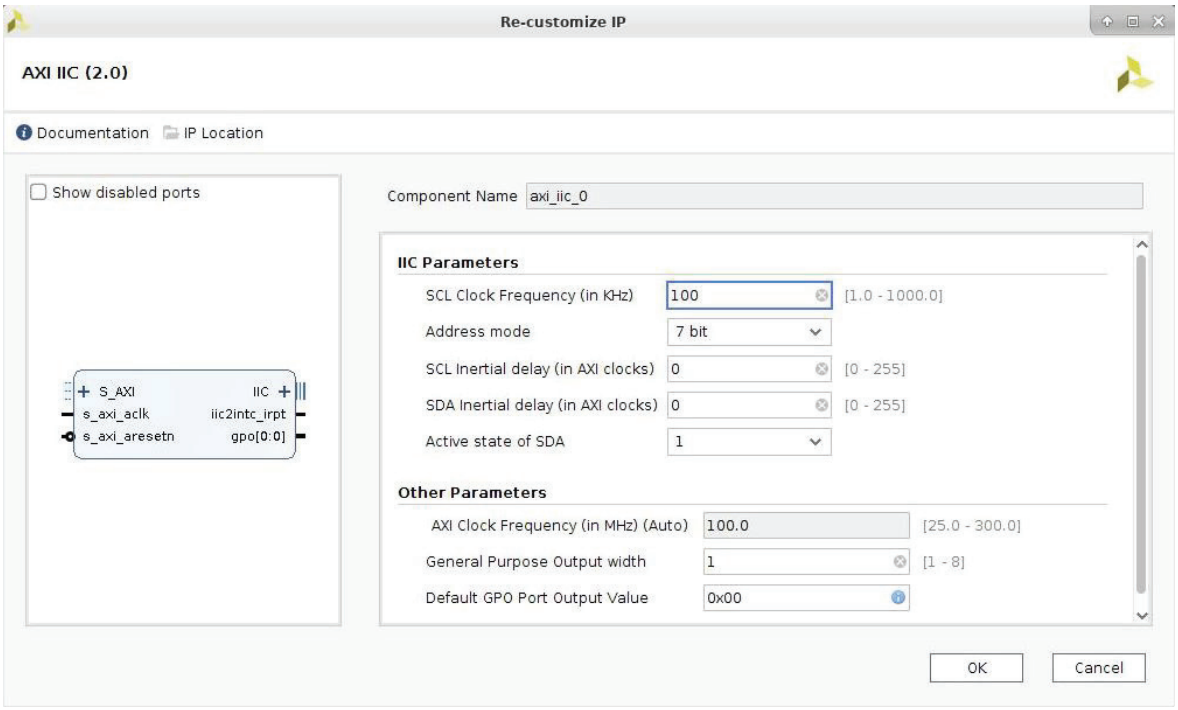

Figura 85. Configuración del IP IIC de Xilinx.

La interfaz IIC del IP se conecta al PMOD JC, dado que el módulo RTC está conectado a dicho conector (Figura 86).

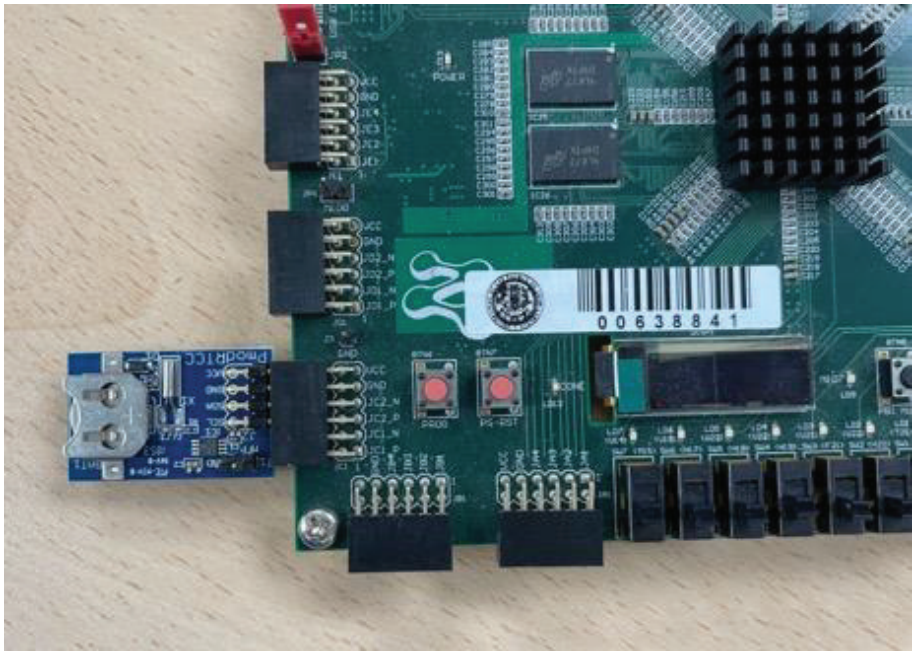

Figura 86. PMOD JC. Conexión del PMOD RTCC.

5.4. Bloque del Sistema de Procesamiento 7 o PS-7

El IP del Sistema de Procesamiento 7 o PS-7 de Xilinx [56] tiene como objeto conectar a nivel *hardware* la Lógica Programable con el Sistema de Procesamiento. La instanciación de este IP permite controlar los bloques *hardware* desde una aplicación *software* y guardar en memoria *On-Chip* las muestras procesadas en la plataforma.

Hay que destacar que, este *core* permite habilitar o deshabilitar periféricos de E/S (UART, SD, GigE, etc.), así como puertos AXI de E/S (AXI GP, AXI HP y AXI ACP), configurar las E/S multiplexadas y multiplexadas extendidas, generar señales de reloj de entre 10KHz y 250MHz de frecuencia y gestionar interrupciones PS-PL y PL-PS. En definitiva, permite la integración *hardware-software* en una unidad y ofrecer una solución SoC flexible y extensible.

A continuación, se expone la configuración del IP PS-7 (Figura 87):

- Habilitar un puerto AXI GP maestro. Este puerto AXI GP permite controlar y configurar desde el *software* los IP de la plataforma por medio de sus interfaces AXI4-Lite.
- Habilitar un puerto AXI HP esclavo. Esta interfaz de Alto Rendimiento permite enviar a alta velocidad las muestras procesadas (salida del IP AXI DMA) al PS.
- Generar una señal de reloj de 100 MHz. Este reloj sincroniza la funcionalidad del *System-on-Chip* completo.
- Habilitar una señal de *reset* de propósito general. Esta señal de *reset* permite restablecer el *hardware* diseñado a los valores por defecto.
- Habilitar una conexión *Gigabit Ethernet*. A través de esta interfaz se podrán transferir los ficheros de datos hacia un servidor por Ethernet.
- Habilitar un puerto UART. A través de este puerto, se puede controlar, así como configurar el sistema.

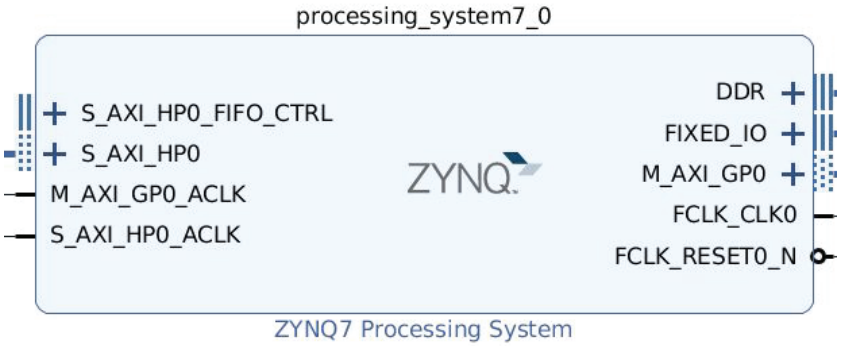

Figura 87. IP del Sistema de Procesamiento 7.

5.5. Integración del *hardware*

Con la finalidad de integrar la plataforma diseñada se crea un nuevo proyecto en Vivado que incluye un diseño a nivel de bloques con los IPs de procesamiento de datos, IPs de control e IP del Sistema de Procesamiento descritos.

A continuación, se expone la conexión de dichos bloques *hardware* y los pines a conectar con el exterior. En la Figura 88, se observa la conexión de los bloques de procesamiento de datos. Esta cadena de IPs permite el procesamiento de hasta dieciséis canales de captura simultáneos.

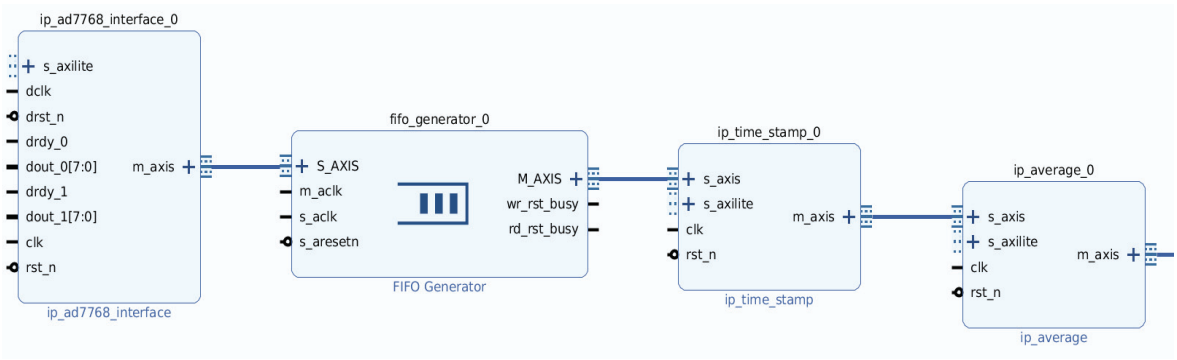

Figura 88. Conexión bloques hardware de procesamiento de dieciséis canales de captura.

De esta parte, únicamente se conectan al exterior las entradas del IP Interfaz AD7768 y se conecta la señal de reloj de los ADCs a la interfaz esclava del IP FIFO *Generator* dedicado al cruce de dominios de reloj (Figura 89).

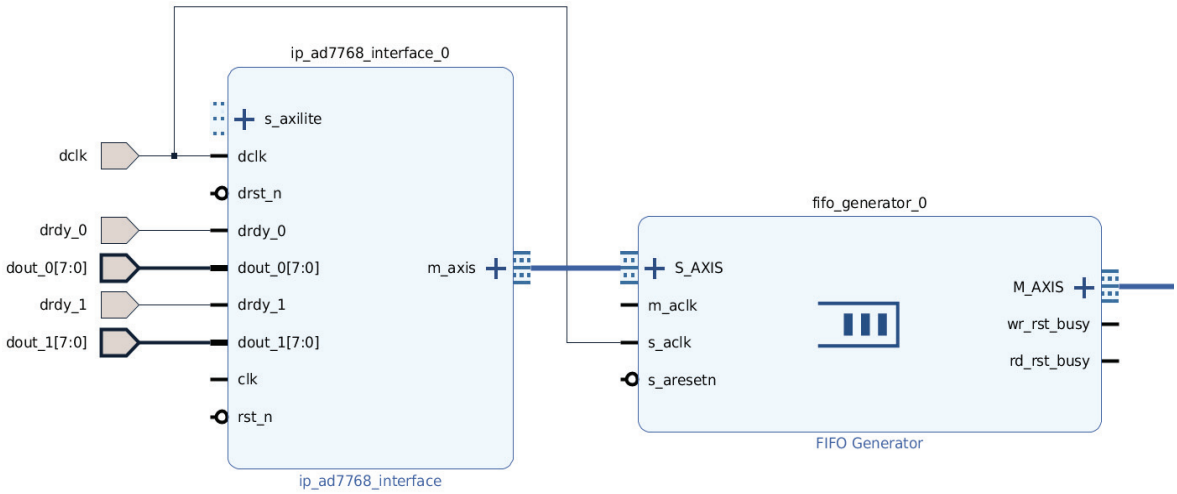

Figura 89. Conexión a los pines externos de las entradas de los IP Interfaz AD7768.

En la Figura 90, se muestra la conexión de la salida del IP de Cálculo de Promedios (último bloque de la cadena de procesamiento) con la entrada del IP AXI *Direct Memory Access* a través del IP FIFO *Generator* destinado a recolectar las tramas procesadas. Además, se observa la vinculación

de la salida del controlador AXI DMA con el puerto AXI HP, así como la conexión del puerto DDR, del IP PS-7.

El IP AXI *Memory Interconnect* es instanciado automáticamente por Vivado. Este IP simplemente interconecta la salida del IP AXI DMA con el puerto AXI de Alto Rendimiento.

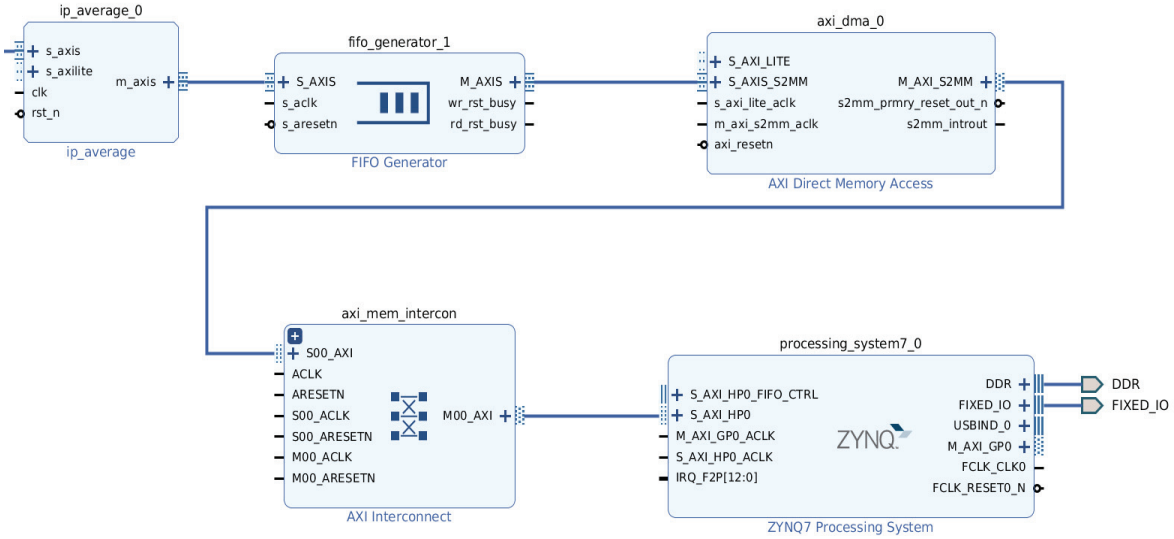

Figura 90. Conexión IP de Cálculo de Promedios, FIFO Generator, AXI DMA y PS-7.

Referente a los bloques *hardware* de control, se conectan al exterior por completo las salidas de los IPs: Control de la UPS, Control de Ganancias (Figura 91), Control por PIN, *Master*/*Slave* y AXI IIC. De los IP AXI Quad-SPI, únicamente se conectan al exterior las E/S necesarias (Figura 92). Dichas E/S se vinculan con los respectivos pines FMC y PMOD.

Por último, Vivado automáticamente conecta las interfaces AXI4-Lite de los IP con el puerto AXI GP a través del IP AXI *Interconnect*, lo que permite la comunicación *software*-IP, vincula la señal de reloj y *reset* del IP PS-7 con las entradas de *clk* y *rst_n* de los bloques *hardware* e instancia el IP de Control de *Reset*.

En la Figura 93, se observa la integración *hardware* final.

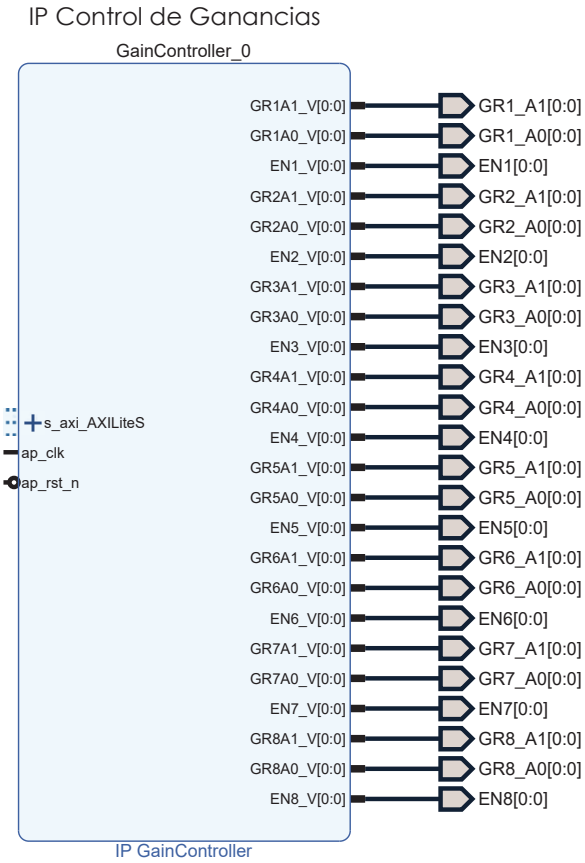

Figura 91. Conexión con el exterior de los pines del IP Control de Ganancias.

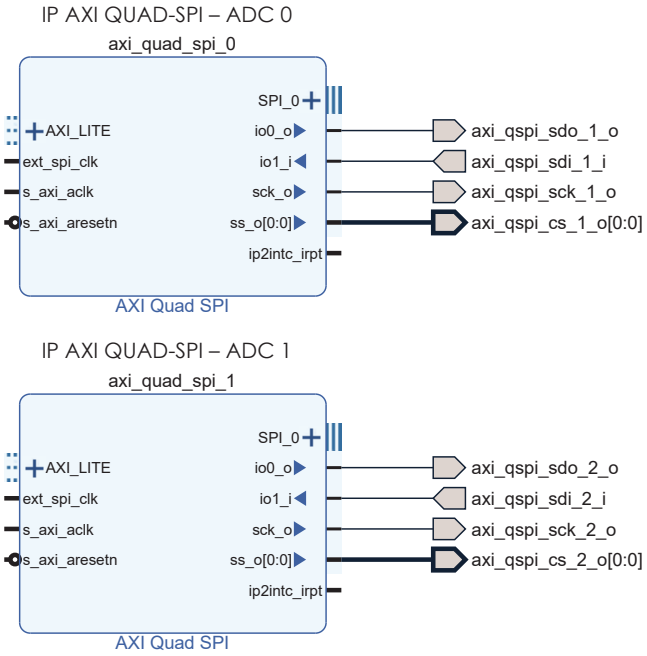

Figura 92. Conexión con el exterior de los pines de los IPs AXI Quad-SPI.

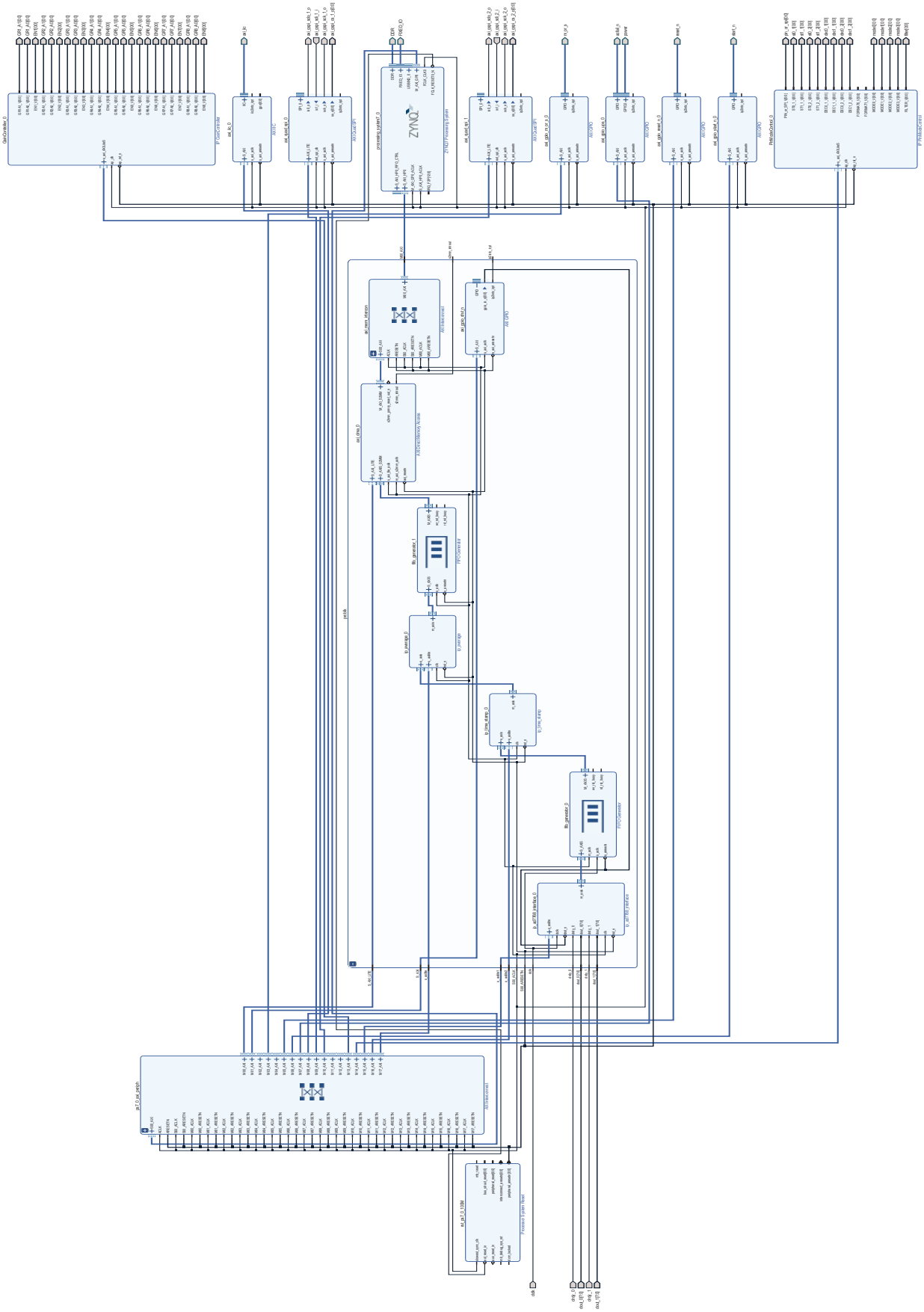

Figura 93. Integración del hardware final.

5.5.1. CONEXIONES EXTERNAS DE LA INTEGRACIÓN HARDWARE

La vinculación de los pines conectados al exterior con los pines del FMC y PMOD se realiza a través de un fichero de restricciones o *constraints*. En la Figura 94 y en la Figura 95, se observa la distribución requerida para los conectores FMC y PMOD respectivamente.

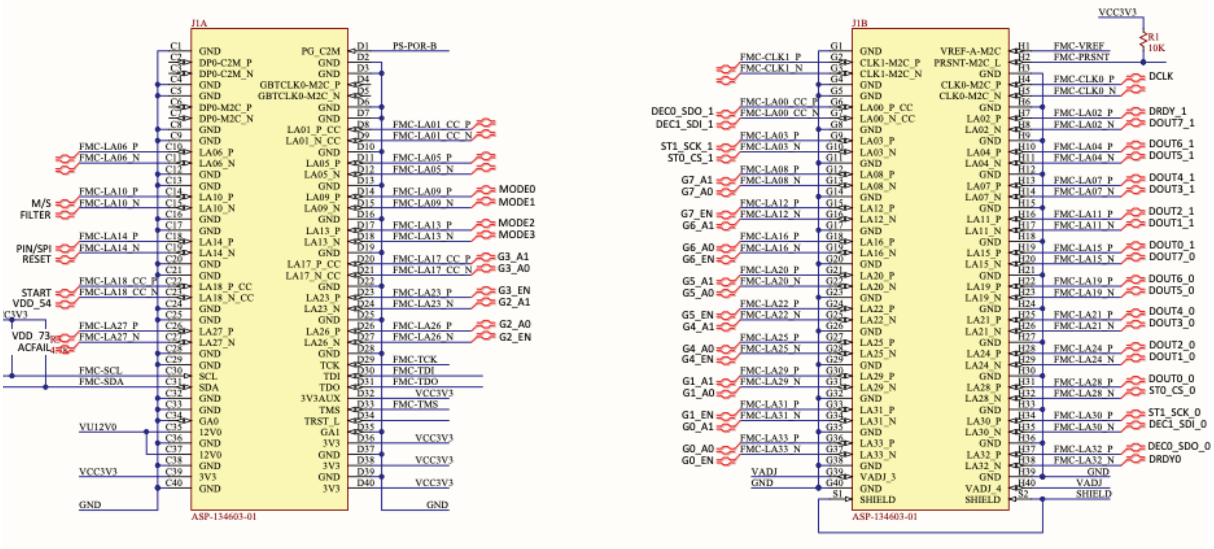

Figura 94. Distribución de E/S en el FMC.

Pmod	Signal Name	Zynq EPP pin
JC1 Differential	JC1_N	AB6
	JC1_P	AB7
	JC2_N	AA4 → SDA
	JC2_P	Y4 → SCL
	JC3_N	T6
	JC3_P	R6
	JC4_N	U4
	JC4_P	T4

Figura 95. Distribución de E/S en el PMOD.

Tal y como se aprecia, las E/S *dec0_1* y *sdo_1*, *dec0_2* y *sdo_2*, *st0_1* y *cs_1*, *st0_2* y *cs_2*, *dec1_1* y *sdi_1*, *dec1_2* y *sdi_2*, *st1_1* y *sck_1*, *st1_2* y *sck_2* comparten pin en el FMC. Las E/S *dec0_1* , *dec0_2*, *st0_1*, *st0_2*, *dec1_1*, *dec1_2*, *st1_1* y *st1_2* operan en el modo PIN y las *sdo_1*, *sdo_2*, *cs_1*, *cs_2*, *sdi_1*, *sdi_2*, *sck_1* y *sck_2* en el modo SPI. Dichos pares de E/S deben ser multiplexados en función del modo de configuración de los AD7768 (*pin_or_spi*), ya que únicamente es posible direccionar un pin conectado al exterior a través de un pin del FMC.

Por tanto, se genera el *wrapper* HDL de la integración y se crea un *wrapper* de nivel superior que instancia dicho *wrapper* y los multiplexores.

La multiplexación de un pin de entrada (ej.: *sdi_1*) con un pin de salida (ej.: *dec1_1*) en función del modo de configuración de los conversores (*pin_or_spi*), se realiza instanciando un IOBUF de

Xilinx (Código 4). Mientras que, la multiplexación de dos pines de salida (ej.: *dec0_1* y *sdo_1*), se realiza aplicando una estructura *assign* condicional (Código 5).

```
IOBUF #(.DRIVE(12),
        .IBUF_LOW_PWR("TRUE"),
        .IOSTANDARD("LVCMOS33"),
        .SLEW("FAST"))

IOBUF_DEC1_1_QSPI_SDI_1 (
            .O(axi_qspi_sdi_1_i),
            .IO(dec1_1_or_qspi_sdi_1),
            .I(dec1_1),
            .T(pin_or_spi));
```

Código 4. Multiplexación de pines E/S. IOBUF de Xilinx.

```
assign dec0_1_or_qspi_sdo_1 = pin_or_spi ? axi_qspi_sdo_1_o : dec0_1;
```

Código 5. Multiplexación de pines S/S. Estructura assign condicional.

Tras disponer de un pin conectado al exterior ya sea a través del pin FMC y de un pin del PMOD se crea el fichero de restricciones de la integración *hardware* final. En el Código 6, se observa un fragmento de los direccionamientos.

```
## ------------------------------------------------------------------------
## FMC Expansion Connector - Bank 34 & Bank 35
## ------------------------------------------------------------------------
set_property PACKAGE_PIN L18 [get_ports dclk]
set_property PACKAGE_PIN M20 [get_ports dec1_2_or_qspi_sdo_2]
set_property PACKAGE_PIN M19 [get_ports dec0_2_or_qspi_sdi_2]
set_property PACKAGE_PIN P18 [get_ports {dout_1[7]}]
set_property PACKAGE_PIN P17 [get_ports drdy_1]
set_property PACKAGE_PIN P22 [get_ports st0_2_or_qspi_cs_2]
set_property PACKAGE_PIN N22 [get_ports st1_2_or_qspi_sck_2]
set_property PACKAGE_PIN M22 [get_ports {dout_1[5]}]
set_property PACKAGE_PIN M21 [get_ports {dout_1[6]}]
...
```

Código 6. Fragmento del fichero de restricciones. Direccionamiento de pines externalizados.

Además, ya que por medio del propio fichero de restricciones se fijan las tensiones de los pines de la *ZedBoard*, se fija una tensión de 3.3 V en los pines del FMC y PMOD (Código 7).

```
# ------------------------------------------------------------------------
# IOSTANDARD Constraints
# ------------------------------------------------------------------------
# Set the bank voltage for IO Bank 33 to 3.3V
set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 33]]
```

```
# Set the bank voltage for IO Bank 34 to 3.3V
set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 34]]

# Set the bank voltage for IO Bank 35 to 3.3V
set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 35]]
```

Código 7. Fragmento del fichero de restricciones. Tensión de los pines FMC y PMOD.

Finalmente, en este *constraints* se declara el periodo estimado más desfavorable de la señal de reloj de salida de datos de los AD7768 (*dclk*) de aproximadamente de 31ns (Código 8). Con esta declaración, se obtiene una síntesis e implementación del *hardware* que soporta el dominio de reloj más adverso.

```
# ----------------------------------------------------------------------------
# Clock constraints
# ----------------------------------------------------------------------------
create_clock -period 31.000 -name dclk [get_ports dclk]
set_clock_groups -asynchronous -group [get_clocks dclk] -group [get_clocks clk_fpga_0]
set_false_path -from [get_clocks dclk] -to [get_clocks clk_fpga_0]
set_false_path -from [get_clocks clk_fpga_0] -to [get_clocks dclk]
```

Código 8. Fragmento del fichero de restricciones. Restricciones de reloj.

5.5.2. Resultados de la síntesis e implementación de la integración hardware

A través del entorno Vivado, se realiza la síntesis y, después, la implementación de la integración *hardware* final. En la Figura 96, se observa los resultados de *timing* de la síntesis y, en la Figura 97, los de la implementación. Ambos resultados de *timing* son válidos para la frecuencia de funcionamiento de 100MHz requerida.

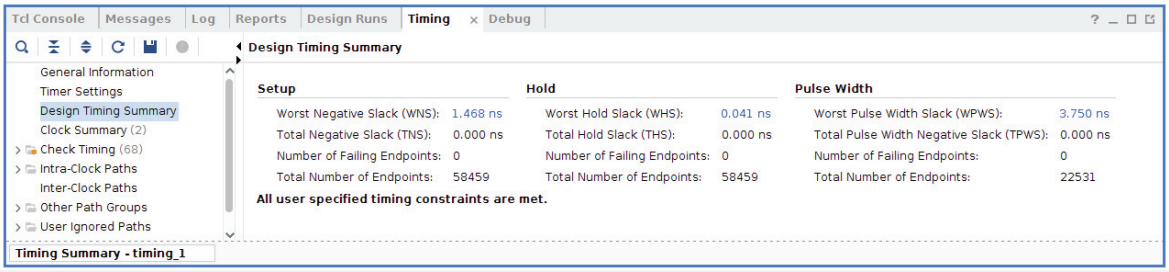

Figura 96. Resultados de timing de la síntesis.

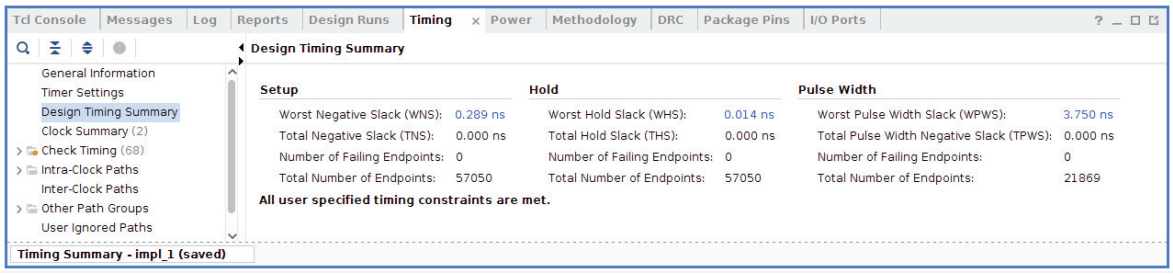

Figura 97. Resultados de timing de la implementación.

La plataforma final presenta un consumo de potencia de 1.836 W (Figura 98) y el consumo de recursos lógicos expuesto en la Figura 99.

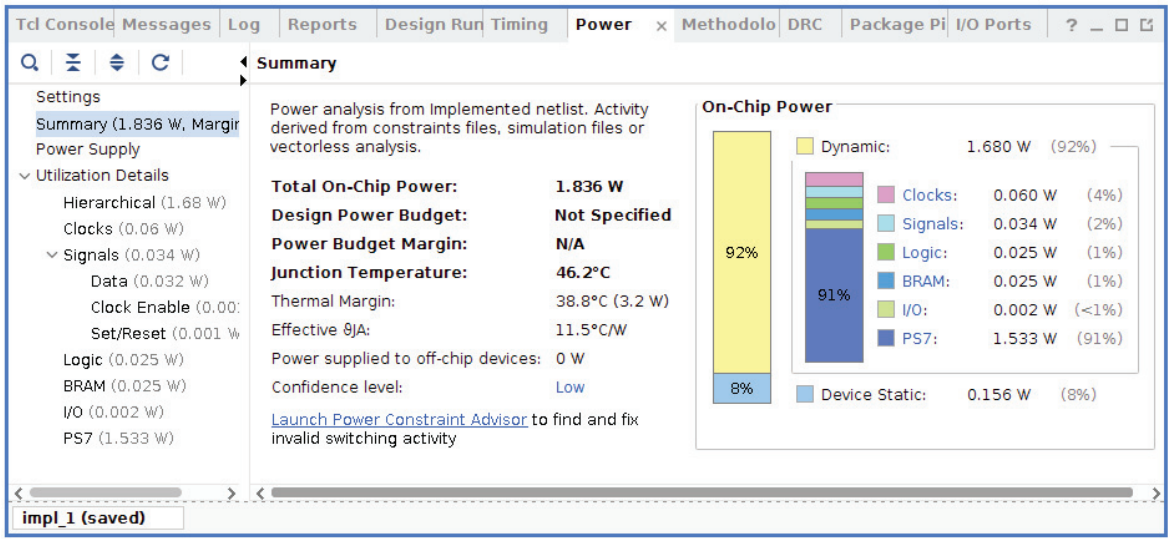

Figura 98. Potencia consumida de la integración.

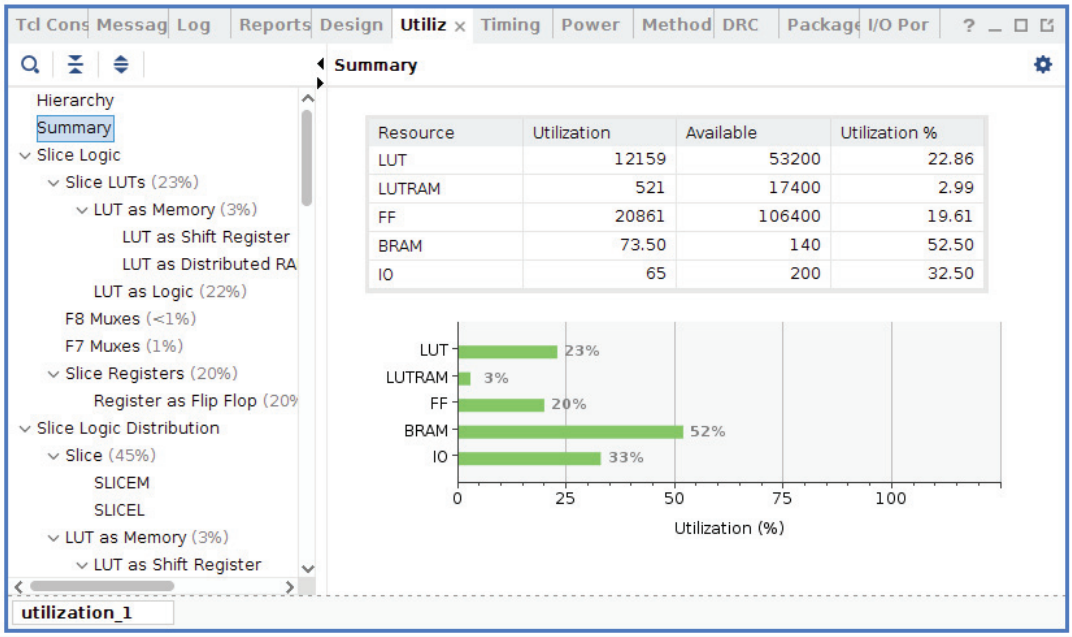

Figura 99. Recursos lógicos consumidos por la integración.

Por último, en la Figura 100, se muestra el esquemático de la plataforma *hardware* final y, en la Figura 101, la distribución de los recursos lógicos en el Zynq.

5.6. Conclusiones

En este capítulo, se han descrito detalladamente los IP que conforman la plataforma *hardware*. Se ha verificado la funcionalidad, así como validado los tiempos de respuesta, de los IP desarrollados y se ha expuesto la configuración de los bloques *hardware* de Xilinx utilizados. Los resultados de la verificación y validación de los IP diseñados han resultado ser favorables. Además, se ha expuesto el uso del PMOD *Real Time Clock*/*Calendar* de *Digilent* que integra el MCP79410 de Microchip como dispositivo RTC.

Se ha explicado la integración del *hardware* que ofrece solución al actual Trabajo de Fin de Máster y se ha expuesto la potencia consumida, así como los recursos *hardware* utilizados, que supone la implementación de dicha integración. El consumo energético del *hardware* implementado (1.836 W) es soportado con creces por la UPS del módulo y el moderado consumo de recursos lógicos admite la instanciación de Analizadores Lógicos Integrados (ILA), lo que permite depurar y validar la plataforma *hardware*, aplicando metodologíad de *cross-validation*. Esto supone la inclusión de ILAs en las partes del *hardware* a analizar, el uso del *Analog Discovery* 2 para analizar las E/S de la *ZedBoard*, así como la construcción del *software*. En el próximo capítulo, se describe la integración *hardware-software* y, finalmente, se valida la integración *System on-Chip* completa.

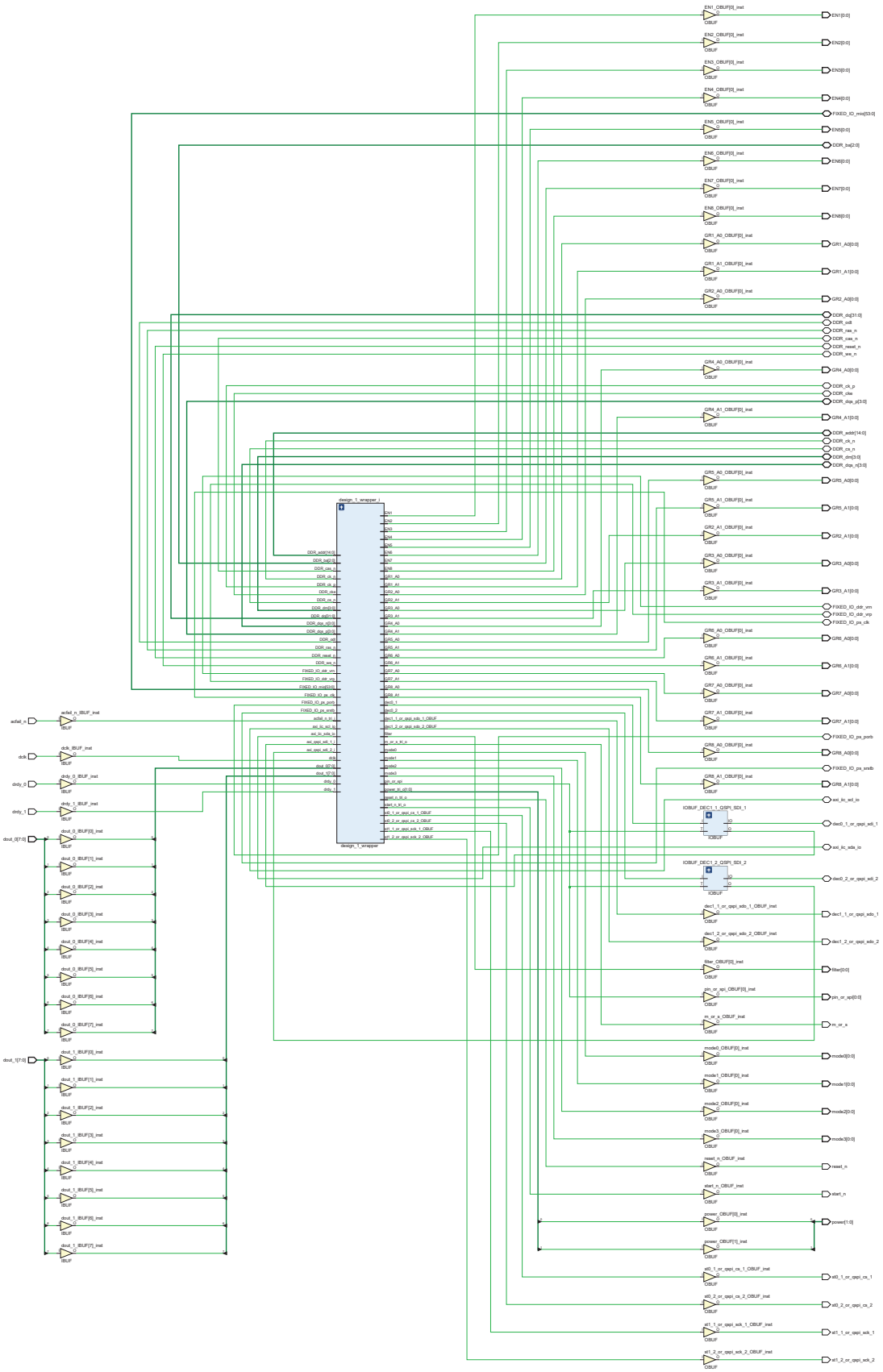

Figura 100. Esquemático de la integración hardware final.

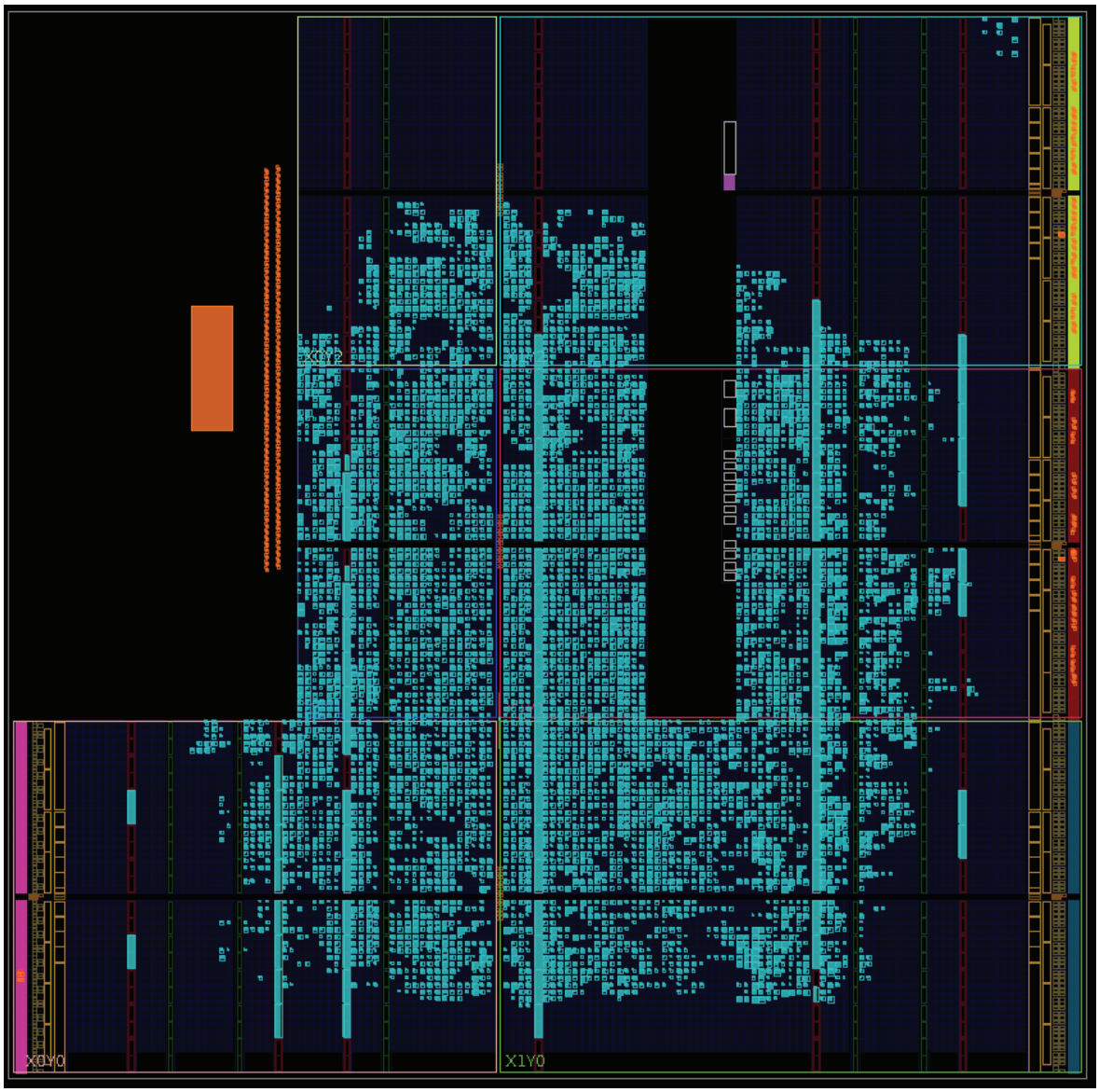

Figura 101. Implementación de la integración en el Zynq.

Capítulo 6. INTEGRACIÓN *HARDWARE-SOFTWARE*

En este capítulo, se realiza la integración final de la solución System-on-Chip del presente Trabajo de Fin de Máster. En otras palabras, se exporta la plataforma hardware diseñada, se configura el Board Support Package, se exponen las librerías desarrolladas y utilizadas, y se crea la aplicación software.

6.1. Introducción

La solución SoC FPGA final requiere la integración de la plataforma *hardware* ya diseñada con una aplicación *software* que permita realizar las operaciones siguientes (ver con más detalle en el apartado 3.5. del Capítulo 3):

i. Configurar y controlar la plataforma *hardware*.
ii. Configurar y controlar el módulo DAS.
iii. Almacenar en *On-Chip-Memory* las muestras procesadas en el *hardware*.
iv. Guardar en ficheros de datos las muestras procesadas.
v. Enviar dichos ficheros a un servidor por Ethernet.
vi. Reconfigurar y controlar el sistema mediante *órdenes de control y reconfiguración* transferidas por un puerto UART.

Por esta razón, se debe generar el *bitstream* y exportar el *hardware*, crear un nuevo proyecto en Vitis IDE, configurar en el *Board Support Package* (BSP) las librerías y *drivers* disponibles a reutilizar, desarrollar las librerías y *drivers* necesarios y diseñar la aplicación *software* en cuestión.

6.2. Generación del *bitstream* y exportación de la plataforma *hardware*

Con el objeto de integrar la plataforma *hardware* con la aplicación *software* a diseñar, a través de Vivado IDE se genera el *bitstream* y se exporta a un fichero *Xilinx Shell Archive* (XSA) la plataforma *hardware* diseñada. Este fichero XSA permite crear una aplicación *software* empotrada en la

plataforma *hardware* por medio de Vitis IDE, ya que recopila la descripción completa del *hardware* (ej.: *drivers* del *hardware*, mapa de memoria, configuración de los bloques *hardware*, etc.).

6.3. Creación del *Application Project*

Tras obtener el fichero XSA de la plataforma *hardware*, así como el *bitstream*, se procede con la creación del *Application Project* a través de Vitis IDE (Figura 102). Este proyecto Vitis permite la integración *hardware-software*, puesto que crea una plantilla básica de la plataforma *hardware* para la aplicación empotrada, extrayendo la información del *hardware* a partir del fichero XSA exportado desde Vivado IDE.

En este caso, se crea una aplicación *standalone* (versión 7.2) que se implementa sobre la CPU-0, es decir, sobre el primer microprocesador ARM Cortex A9 que integra el Zynq. Por consiguiente, con este *proyecto de aplicación* se obtiene un conjunto de capas *software* simples y de bajo nivel que ofrecen acceso a aspectos básicos del procesador (ej.: *caches*, interrupciones y excepciones).

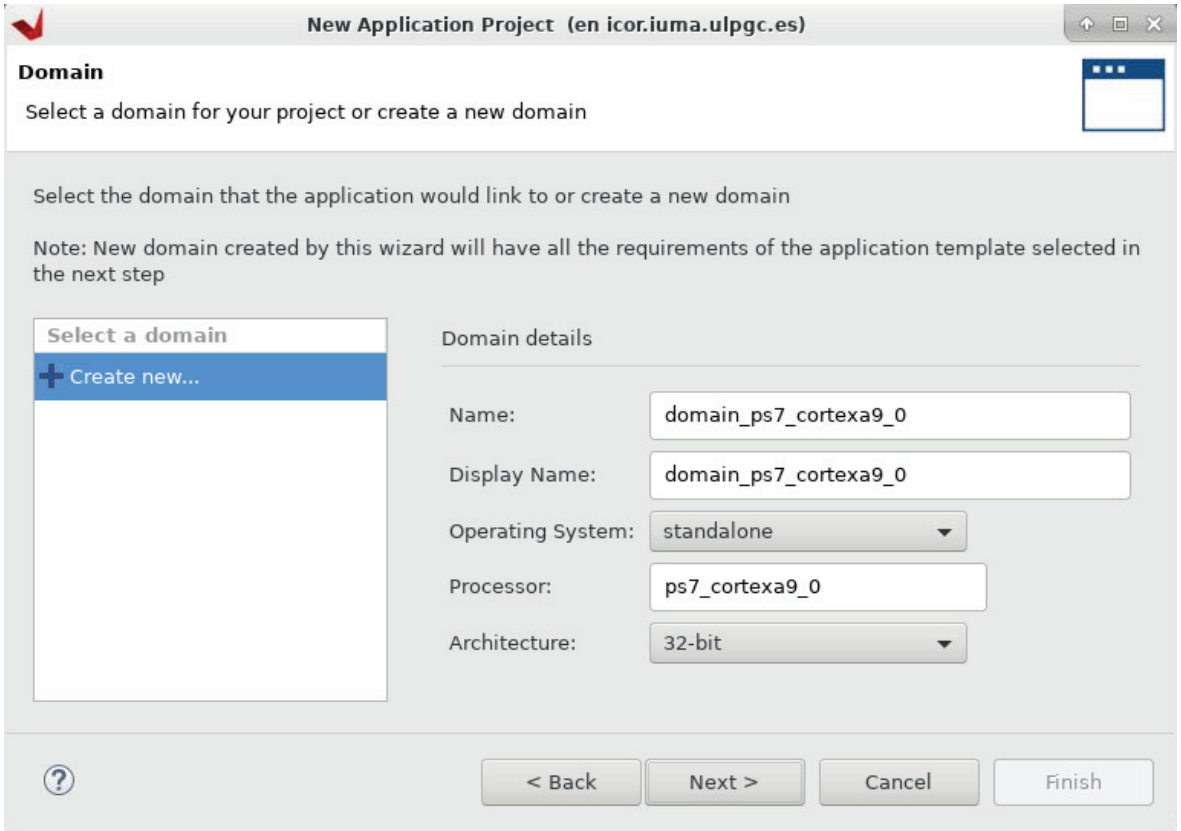

Figura 102. Creación del Application Project. Selección del dominio y Sistema Operativo.

6.4. Configuración del *Board Support Package* (BSP)

El *Board Support Package* (BSP) es un conjunto de *drivers* y librerías que proporcionan acceso al *hardware* desde una capa alta de abstracción *software*. Dichos controladores se generan a partir de la Capa de Abstracción *Hardware* (*Hardware Abstraction Layer* o HAL), que contiene información sobre la configuración del *hardware*, durante el proceso de creación del *Application Project* en Vitis IDE. Por el contrario, las librerías deben ser incluidas y, en caso requerido, ajustadas por el usuario mediante la configuración del *Board Support Package* a través de Vitis IDE.

En esta oportunidad, a fin de construir un Sistema de Ficheros FAT (*FAT File Systems* o FFS) en RAM, se incluye y se configura la librería *xilffs.h*. Esta librería ofrece una capa alta de abstracción que permite la construcción de FFS a través de interfaces SSD o eMMC, así como basados en RAM.

Además, con el objetivo de enviar los ficheros de datos por Ethernet hacia un servidor, también se incluye la librería *standalone lwip211.h*. Esta librería proporciona un *stack* TCP/IP ligero, es decir, permite establecer conexiones Ethernet con un consumo de recursos reducido.

En la Figura 103, se observa la configuración de la librería *xilffs.h*. Como se aprecia, es posible ajustar diferentes parámetros. En este caso, solo se configura el parámetro *fs_interface* con un valor de *2*, lo que permite montar el FFS en RAM (SSD: 1, RAM: 2). El Sistema de Ficheros FAT se define con el tamaño inicial por defecto: 3.145MB.

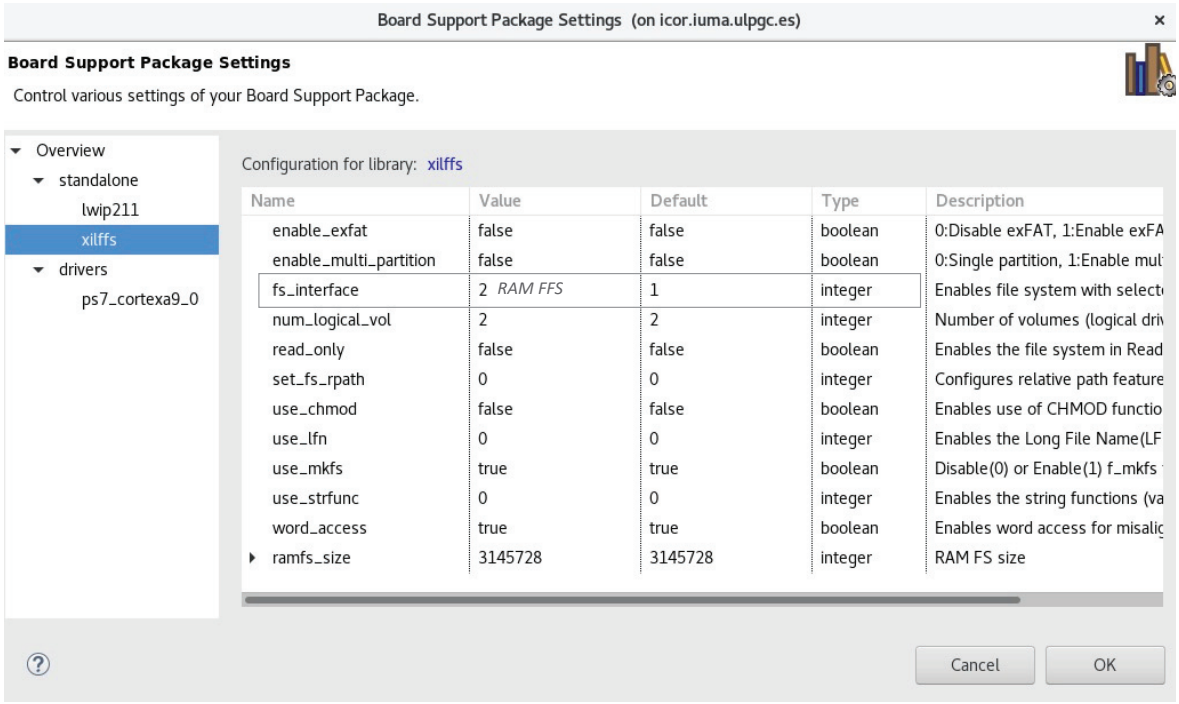

Figura 103. Configuración de la librería xilffs.h.

6.5. Aplicación *baremetal*

A continuación, tras la configuración del *Board Support Package*, se procede con el diseño de la aplicación *bare-metal* de la presente solución *System-on-Chip*.

6.5.1. LIBRERÍAS Y *DRIVERS* UTILIZADOS

La aplicación *baremetal* utiliza diferentes librerías y *drivers* propios de Xilinx, librerías desarrolladas por Digilent y librerías empotradas al actual *System-on-Chip*, que han sido elaboradas en el presente Trabajo de Fin de Máster. En la Tabla 17, se describen brevemente dichas librerías y *drivers*.

Tabla 17. Librerías y controladores utilizados en la aplicación software.

Librería/driver	Descripción	Creador
xparameters.h.	Librería que incluye los genéricos de la plataforma *hardware*.	Xilinx
xil_io.h	Librería que gestiona las E/S del *core* sin interfaces especiales.	Xilinx
xil_types.h	Librería que incluye tipos de variables básicos para el *software*.	Xilinx
xil_printf.h	Librería que incluye funciones similares a la función *print()*.	Xilinx
xstatus.h	Librería que incluye los genéricos de los estados del *software*.	Xilinx
xtime_l.h	Librería que ofrece acceso al *Global Counter* de 64 bits en la PMU. Esta es utilizada para medir los tiempos de ejecución del *software*.	Xilinx
xgpio.h	*Driver* del IP AXI *General Purpose* I/O.	Xilinx
xaxidma.h	*Driver* del IP AXI *Direct Memory Access*.	Xilinx
xilffs.h	Librería dedicada a la gestión de Sistemas de Ficheros FAT.	Xilinx
lwip211.h	Librería que proporciona un *stack* TCP/IP ligero.	Xilinx
xuartps_hw.h	*Driver* de la UART del Sistema de Procesamiento (PS).	Xilinx
PmodRTCC.h	*Driver* del *PMOD Real Time Clock and Calendar* (RTCC) de *Digilent*. Este utiliza el *driver* del IP AXI I2C de Xilinx.	Digilent
axiqspi.h	La librería *axiqspi.h* ha sido desarrollada en el presente Trabajo de Fin de Máster, dado que el *driver XSpi.h* de Xilinx no cumplía con las necesidades del protocolo SPI del AD7768. Esta nueva librería,	TFM

Librería/driver	Descripción	Creador

	además de adaptar el protocolo SPI del IP AXI *Quad*-SPI de Xilinx al del AD7768, incluye funciones empotradas a la configuración de dicho ADC. A continuación, se exponen algunas de estas funciones:
	`void qspi_write_ad7768_standby(u8 slave, u8 data):`
	Escribe en el registro SPI *standby* (0h00) de un conversor AD7768.
	`int qspi_read_ad7768_offset(u8 slave, u8 msb_offset):`
	Retorna el *offset* de un canal de un conversor AD7768.
	`int qspi_ad7768_soft_reset(u8 slave):`
	Resetea por modo SPI un conversor AD7768.	

menu.h	La librería *menu.h* ha sido desarrollada en el presente Trabajo de Fin de Máster. Esta ofrece un menú interactivo que permite la configuración del módulo DAS a través del puerto UART del PS.
	Hay que destacar que, este *menú* tras cada reconfiguración del sistema realiza una lectura del *hardware* reconfigurado y muestra el valor leído. Esto permite verificar que el sistema ha sido reconfigurado correctamente. A continuación, se expone el menú de configuración y los parámetros que permite configurar:
	`########## DAS Module Configuration ##########`
	`a) Configuration by PIN mode.`
	`- Set standby channels.`
	`- Set channels decimation rate.`
	`b) Configuration by SPI mode.`
	`- Set standby channels.`
	`- Set operative channels.`
	`- Set channels decimation rate.`
	`- Set channels offset.`
	`- Set channels internal gain`
	`c) Gains configuration`
	`d) Average configuration.`
	`e) Exit.`
	`##############################################`
		TFM

Librería/driver	Descripción	Creador

deepspace.h	La librería *deepspace.h* ha sido desarrollada en el presente Trabajo de Fin de Máster y es la librería principal de la aplicación *software*. Esta incluye los genéricos y funciones que permiten configurar el *software* y el *hardware* completo.
	A continuación, se exponen algunos de estos genéricos y funciones:
	`#define OCM_BASE_ADDR 0x00000000:`
	Dirección base de la OCM. En esta dirección se escriben las muestras procesadas en el *hardware*.
	`void FatFileSystemInitialization(void):`
	Crea y monta el Sistema de Ficheros FAT.
	`void IpTimestampWrite(u32 reg, u32 data):`
	Escribe un valor en un registro del IP de Marca de Tiempo Real.
		TFM

6.5.2. Inicialización y configuración del sistema on-Chip

Antes de arrancar y configurar el módulo e iniciar la adquisición y procesamiento de las muestras, se debe inicializar y configurar el subsistema *hardware*, el Sistema de Ficheros FAT y el cliente. Por esta razón, inicialmente se realizan las siguientes tareas:

1. Inicializar los *drivers* de los IP AXI GPIO (ej.: *driver* del IP de Control de la UPS).

```
XGpio xGpioUps;
status = XGpio_Initialize(&xGpioUps, GPIO_UPS_DEVICE_ID);
if(status != XST_SUCCESS){
    xil_printf("Error! XGPIO UPS initialization. \r\n");
    return XST_FAILURE;
}
```

2. Inicializar el *driver* y deshabilitar las interrupciones del IP AXI *Direct Memory Access*.

```
XAxiDma xAxiDma;
XAxiDma_Config *xAxiDmaConf;

xAxiDmaConf = XAxiDma_LookupConfig(DMA_DEVICE_ID);
if(!xAxiDmaConf){
    xil_printf("Error! XDMA Lookup Configuration. \r\n");
    return XST_FAILURE;
}

status = XAxiDma_CfgInitialize(&xAxiDma, xAxiDmaConf);
if (status != XST_SUCCESS){
```

```
    xil_printf("Error! XDMA Configuration Initialization. \r\n");
    return XST_FAILURE;
}

XAxiDma_IntrDisable(&xAxiDma,XAXIDMA_IRQ_ALL_MASK,XAXIDMA_DEVICE_TO_DMA);
```

3. Inicializar y configurar los IP AXI *Quad*-SPI. Esta configuración consiste en adecuar el protocolo SPI de los *cores* al protocolo SPI de los conversores (ej.: establecer a los *cores* como maestros SPI, habilitar la selección del esclavo SPI a través del pin $\overline{CS}$, etc.).

```
void QuadSpiInitialization(void){
    qspi_write_softreset_reg(QSPI_ADC0_BASEADRR, 0x0000000A);
    qspi_write_softreset_reg(QSPI_ADC1_BASEADRR, 0x0000000A);
    usleep(1); // 100 clock cycles.
    qspi_write_ctrl_reg(QSPI_ADC0_BASEADRR, 0x000001E6);
    qspi_write_ctrl_reg(QSPI_ADC1_BASEADRR, 0x000001E6);
}
```

4. Inicializar el *driver* del RTCC y verificar que responde. Esta verificación es crítica, dado que sin el RTCC no es posible conocer la fecha y hora real, ni tampoco, añadir marcas de tiempo real a las muestras. La verificación se realiza leyendo un registro del RTCC, cuyo contenido es conocido y distinto de cero.

```
RTCC_begin(&RTCC, RTC_BASEADDR, 0x6F);

// Is RTC present?
RTCC_ReadIIC(&RTCC, 0x03, ptr, 1);
if(!((ptr[0] & 0x07))){
    xil_printf("Error! RTC couldn't be found! \r\n");
    return XST_FAILURE;

} else{
    xil_printf("Success! RTC is present! \r\n");
}
```

5. Crear y montar el Sistema de Ficheros FAT. Dada la configuración de la librería *xilff.h* anterior, este FFS se monta en memoria RAM con un tamaño de 3.145MB.

```
FRESULT res;
BYTE memWork[FF_MAX_SS]; // Memory work area in bytes
TCHAR *path = "0:/"; // Path

res = f_mkfs(path, FM_FAT, 0, memWork, sizeof memWork);
if (res != FR_OK) {
    xil_printf("Error! Failed to format FAT FS! \r\n");
    return XST_FAILURE;
}
```

```
res = f_mount(&fatfs, path, 1);
if (res != FR_OK) {
    xil_printf("Error! Failed to mount FAT FS after format! \r\n");
    return XST_FAILURE;
}
```

6. Establecer la conexión cliente-servidor. En este caso, se utiliza el Protocolo de Transferencia de Archivos Trivial (*Trivial File Transfer Protocol* o TFTP) y, por tanto, el cliente y el servidor son de tipo TFTP.

   Este protocolo es una versión básica del Protocolo de Transferencia de Archivos (*File Transfer Protocol* o FTP), es decir, permite el envío y recepción de ficheros sin mecanismo de autentificación utilizando el protocolo de transporte UDP. Esto lo hace un protocolo ligero y práctico para el traspaso de archivos de reducido tamaño entre dispositivos conectados a una red [57].

   La librería *lwip211* de Xilinx, a modo de ejemplo de uso, proporciona múltiples aplicaciones de tipo *standalone* y FreeRTOS, entre las cuales se halla una aplicación *bare-metal* que implementa un cliente TFTP. Este *software* crea y escribe en un servidor TFTP tres ficheros de texto, lee dichos ficheros del servidor TFTP y verifica que los ficheros escritos y leídos coinciden [58].

   Ya que esta aplicación realiza la transferencia de archivos hacia un servidor TFTP a partir de un cliente TFTP, lo cual es una de las tareas a desempeñar por la actual aplicación *bare-metal*, parte de este *software* se ha personalizado para la aplicación diseñada.

   Por lo tanto, con la finalidad de establecer el enlace cliente-servidor TFTP, en la aplicación de Xilinx se indica la dirección MAC de la *ZedBoard* y la dirección IP del servidor TFTP. Opcionalmente es posible obtenerla por DHCP.

   En este caso, se utiliza como servidor TFTP un PC con una distribución Linux Ubuntu 18.0.4 (Figura 104).

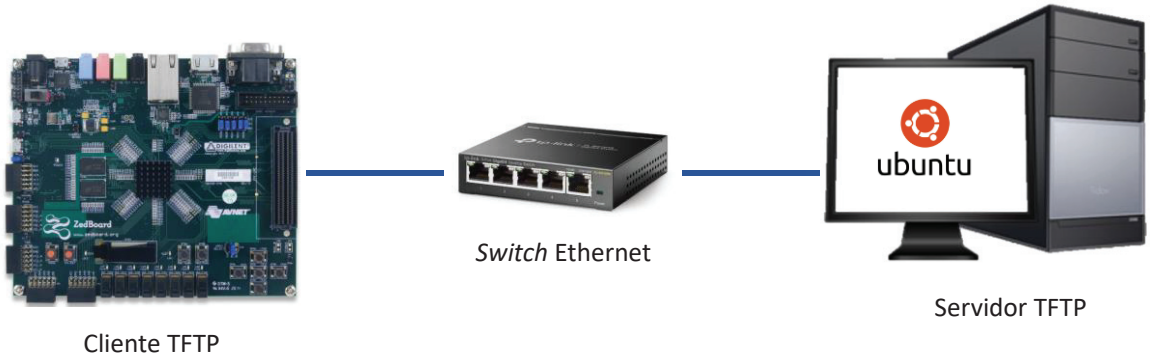

Figura 104. Conexión Cliente-Servidor TFTP.

```
#define TFTP_SERVER_IP          "192.168.100.1" // vlsiws20
unsigned char mac_ethernet_address[] = {0x00,0x0a,0x35,0x00,0x01,0x24};
```

7. Configurar el IP de Interfaz AD7768. Esta configuración consiste en bloquear la entrada de muestras hacia el *hardware* y deshabilitar el Control de Redundancia Cíclica.

```
IpAd7768InterfaceWrite(REG_0H10, DISABLE | NO_CRC);
xil_printf("Success! IP AD7768 Interface configured! \r\n");
```

8. Configurar el IP de Marca de Tiempo Real. Esta configuración atiende a los pasos siguientes:

   - Obtener la fecha y hora real del módulo RTCC.

```
    timeRTCC = GetRtcTime(&RTCC, RTCC_TARGET_RTCC);
```

   - Convertir a formato de tiempo *Unix* con resolución de segundos la fecha y hora real obtenida.

```
    unixTimeSec = GetUnixTimeSecFromRtc(timeRTCC);
```

   - Configurar el IP *Timestamp* con la marca de tiempo *Unix* obtenida, así como definir un incremento del contador local de segundos.

```
    unixTimeSecHigh = unixTimeSec >> 32; // MSB (32 bits)
    unixTimeSecLow = unixTimeSec; // LSB (32 bits)

    IpTimestampWrite(REG_0H10, unixTimeSecLow);
    IpTimestampWrite(REG_0H14, unixTimeSecHigh);
    IpTimestampWrite(REG_0H18, CNT_SECONDS);
```

   - Sincronizar la marca de tiempo *Unix* del IP con la marca de tiempo *Unix* registrada.

```
    IpTimestampWrite(REG_0H1C, 1);
    IpTimestampWrite(REG_0H1C, 0);
```

9. Configurar el IP de Cálculo de Promedios. En este caso, se opta por inicializar la plataforma de procesamiento de forma que promedie cada 4 muestras y empaquete cada 2 048 muestras.

```
    IpAverageWrite(REG_0H10, AVERAGE_EVERY_4_SAMPLES);
    IpAverageWrite(REG_0H14, DATA_PER_PACKET); // 2048
    IpAverageWrite(REG_0H18, AVERAGE);
```

En la Figura 105, se muestra el diagrama de flujo que resume la inicialización y configuración del *System-on-Chip*. Tal y como se observa, en caso de éxito en la inicialización de los *drivers* de los GPIOs y DMA, obtención de respuesta del módulo RTCC, creación y montaje del FFS FAT y conexión cliente-servidor TFTP, el sistema está listo para la inicialización y configuración del módulo, así como para iniciar la adquisición y el procesamiento de las muestras. En caso contrario, el sistema no está preparado, y finaliza la ejecución del programa emitiendo un mensaje de *Error!* especificando el motivo del fallo.

El resto de los procesos de inicialización y configuración del SoC (ej.: configuración IP de Marca de Tiempo Real), también son imprescindibles para que el sistema esté preparado. No obstante, dado que se tratan de escrituras directas en el *hardware*, no requieren de supervisión de errores.

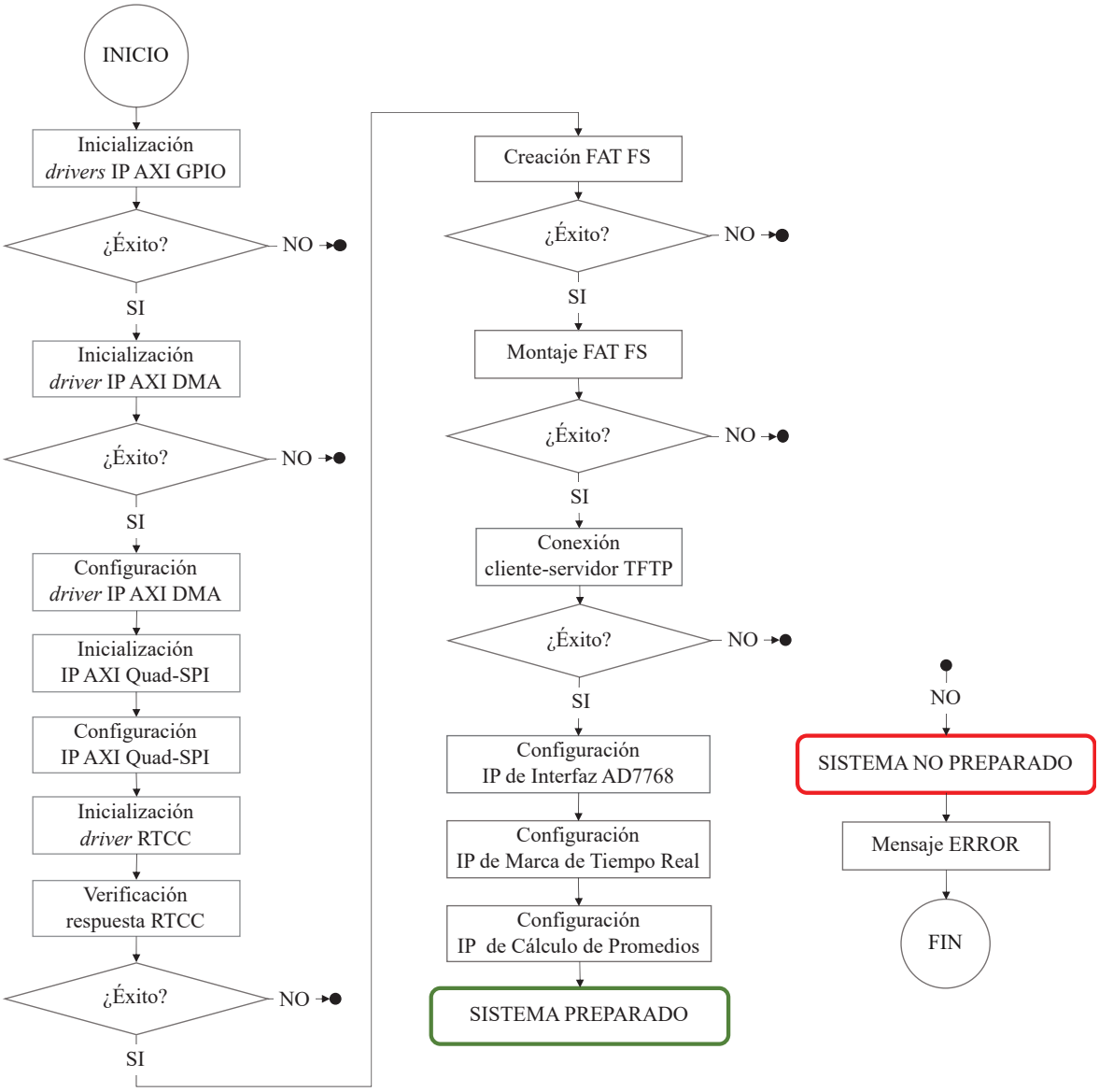

Figura 105. Diagrama de flujo de la inicialización y configuración del System-on-Chip.

6.5.3. Arranque y configuración del módulo DAS

Una vez que el *System-on-Chip* está preparado, se procede con el arranque y configuración del módulo DAS. Por recomendación del fabricante, antes de encender el equipo, este es inicializado con la siguiente configuración de arranque:

- Factor de amplificación de 1 (0 dB) en los 16 canales de captura.

```
    IpGainControllerWrite(REG_0H10, GAIN_ALL_CHANNELS_1);
```

- Modo de configuración de los ADC AD7768 por modo PIN.

```
IpPinModeControlWrite(REG_0H10, PIN_MODE);
```

- Pines de $\overline{\text{RESET}}$ y $\overline{\text{START}}$ a nivel alto.

```
XGpioResetNWrite(1);
XGpioStartNWrite(1);
```

- Dado que la *ZedBoard* tiene un módulo RTCC conectado, se debe definir al primer ADC de la tarjeta de adaptación como MASTER.

```
XGpioMasterOrSlaveWrite(MASTER);
xil_printf("Success! DAS module initialized! \r\n");
```

Tras esta inicialización, se continua con el arranque y configuración gradual del módulo DAS. Esto consiste en realizar las siguientes tareas:

1. Alimentar la etapa digital de la tarjeta AD7768 AMP DIFF 16 CH (+5.4Vcc).

```
XGpioUpsPowerWrite(CR_54VDD);
```

2. Resetear los conversores A/D AD7768.

```
XGpioResetNWrite(0);
usleep(200);
XGpioResetNWrite(1);
usleep(200);
```

3. Configurar los conversores A/D AD7768. Como requisito indispensable, los conversores son configurados, en este caso por Modo PIN, de la siguiente forma:
    - Velocidad de muestreo de 256 kSP/s (*fast power*).
    - DCLK de 8.192 MHz (MCLK/4).
    - Modo de conversión continua (*standard*).

    Y, a modo de configuración inicial:
    - Tasa de decimación de *x32*.
    - Dieciséis canales de captura (*no standby*).
    - Filtro *sinc5*.

```
IpPinModeControlWrite(REG_0H10, PIN_MODE | FAST_MCLK4_STANDARD_MODE |
                        NO_STANDBY | DEC_RATE32 | F_SINC5);
```

```
    // First pulse
    XGpioStartNWrite(0);
    usleep(200);
    XGpioStartNWrite(1);
    usleep(200);

   // Second pulse
    XGpioStartNWrite(0);
    usleep(200);
    XGpioStartNWrite(1);
    usleep(200);
```

4. Definir las ganancias de las 16 señales de entrada. También, a modo de configuración inicial, se define en los dieciséis canales un factor de amplificación de 1 (0 dB).

```
    IpGainControllerWrite(REG_0H10, GAIN_ALL_CHANNELS_1);
```

5. Alimentar la etapa analógica de la tarjeta AD7768 AMP DIFF 16 CH (+7.3 Vcc).

```
    XGpioUpsPowerWrite(CR_54VDD | CR_73VA);
```

En este punto, el sistema está listo para comenzar la captura y procesamiento de hasta dieciséis señales de entrada. La adquisición y el procesamiento de las muestras están sujetos a la configuración por defecto realizada del SoC y de la tarjeta AD7768 AMP DIFF 16 CH.

Por consiguiente, en caso de no reconfigurar el sistema, las señales de entrada serán capturadas y procesadas de la forma siguiente:

Tabla 18. Configuración del Sistema de Adquisición y Procesamiento de Datos.

Configuración del Sistema de Adquisición y Procesamiento de Datos	
Factor de Amplificación	1 (0 dB)
Tipo de Conversión	Continua (*standard*)
Velocidad de Muestreo	256 000 SP/s
Frecuencia de DCLK	MCLK/4 (8.192 MHz)
Tasa de Decimación	x32
Canales Operativos	ch. 0 al 15
Filtro	*sinc5*
Marca de Tiempo Real	Resolución de segundos
Promedios	Cada 4 muestras
Número de Muestras por Paquete	2 048

6.5.4. FUNCIONAMIENTO NORMAL DEL SISTEMA

El estado normal de funcionamiento del sistema está compuesto por dos estados: *capture off* y *capture on*. Durante el estado *capture off* el sistema únicamente gestiona las *órdenes de control y reconfiguración del módulo* recibidas por la UART y supervisa la señal AC *fail* y, durante el estado *capture on*, el sistema, además de realizar las tareas anteriores, realiza la adquisición y procesamiento de las señales de entrada (Figura 106).

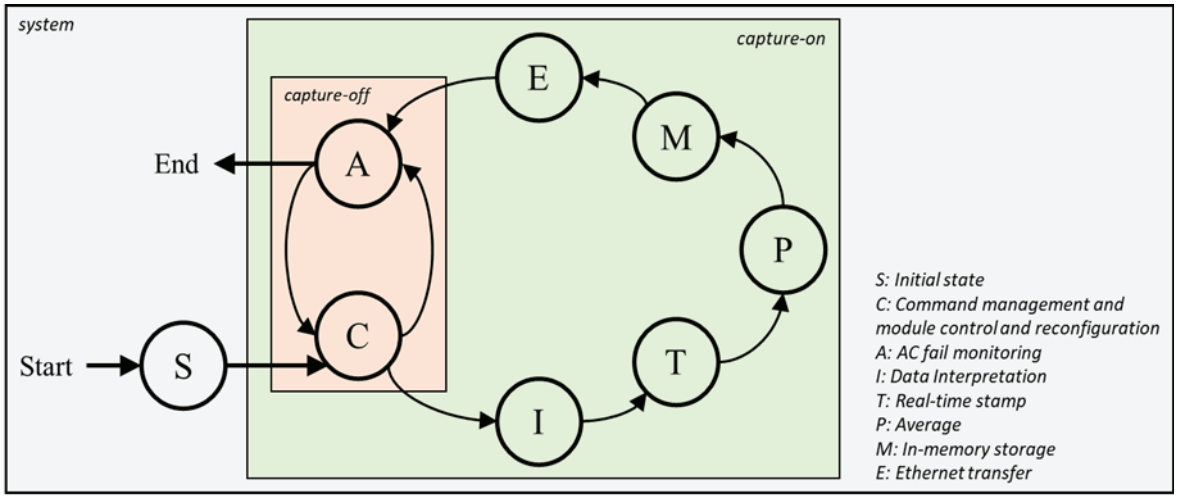

Figura 106. Estados y tareas del funcionamiento normal del sistema.

Una vez finaliza el arranque y configuración del módulo, el sistema entra en el estado *capture off*, es decir, queda a la espera de recibir *órdenes de control o reconfiguración del módulo* al tiempo que supervisa el estado de la fuente externa de alimentación del módulo (AC *fail*). En la Tabla 19, se observan las *órdenes de control y reconfiguración* que dispone el sistema.

Tabla 19. Órdenes de control y reconfiguración del módulo.

Orden	Descripción
Tecla `'C'` o `'c'`	**Habilitar captura**. El sistema entra en el estado de *capture on*. Habilita la entrada de datos digitalizados a la FPGA y, por tanto, habilita la adquisición y procesamiento de las muestras generadas por ambos ADCs.
Tecla `'N'` o `'n'`	**Deshabilitar captura**. El sistema entra en el estado de *capture off*. Deshabilita la entrada de datos digitalizados a la FPGA y, por tanto, deshabilita la adquisición y procesamiento de las muestras generadas por ambos ADCs.
Tecla `'R'` o `'r'`	**Reconfiguración del módulo**. Permite configurar el sistema a través de un menú de configuración interactivo. Los parámetros que permite configurar son los siguientes: 1) Configurar ADCs por PIN. a) Definir canales en *standby*. b) Definir tasa de decimación. 2) Configurar ADCs por SPI. a) Definir canales en *standby*.

	b) Definir tasa de decimación. c) Definir *offset* interno. d) Definir ganancia interna. 3) Configurar ganancias de amplificación. 4) Configurar promedio. Esta orden solo está disponible cuando la captura está deshabilitada (*capture off*).
Tecla `'S'` o `'s'`	**Encender módulo**. Arranca y configura con la configuración por defecto el módulo.
Tecla `'E'` o `'e'`	**Apagar módulo**. Apaga el módulo. Esta orden solo está disponible cuando la captura está deshabilitada (*capture off*).

Únicamente, el sistema entra en el estado *capture on* cuando recibe la petición "habilitar captura" y, entra en el estado *capture off*, cuando recibe "deshabilitar captura". Por lo tanto, es el usuario quién controla la *captura* de las muestras. A continuación, se explica el funcionamiento normal de la aplicación *software*.

En caso de recibir la orden “habilitar captura”, se reconfigura el IP Interfaz AD7768 de modo que permita la entrada de los datos generados por los ADCs a la FPGA. Esto habilita el movimiento de las muestras como un flujo de datos hacia la plataforma *hardware* y procesamiento de estas a medida que ingresan en los IPs: Interfaz AD7768, Marca de Tiempo Real y Cálculo de Promedios. Tras el procesamiento, las muestras se localizan en el *core* AXI FIFO Generator de salida procesadas, empaquetadas y a la espera de ser enviadas a la OCM por el DMA.

Las escrituras del DMA se gestionan desde el *software* y consisten en indicar la dirección destino en memoria y el tamaño en *bytes* del paquete de datos a transferir.

Después de almacenar un determinado número de paquetes de muestras procesadas en OCM, se continúa con el guardado de las mismas en un *fichero de datos*. Una vez salvadas, se prosigue con la transferencia de dicho *fichero* hacia el servidor TFTP.

Completado el envío, dicha colección de muestras procesadas se hallan almacenadas de forma segura en el servidor TFTP y, por tanto, se procede con el borrado del *fichero* transferido del FFS, lo cual evita la saturación del Sistema de Ficheros, y con la comprobación de AC *fail*.

Finalmente, en caso de que la fuente externa de alimentación del equipo esté estable, se repiten los procesos de gestión de las *órdenes de control y reconfiguración del módulo* recibidas, almacenamiento de muestras procesadas en memoria *On-Chip* y en *ficheros de datos*, transferencia de *ficheros* hacia el servidor TFTP, borrado de *ficheros* transferidos del FFS y, otra vez, comprobación de AC *fail*. En caso contrario, se apaga el módulo de forma controlada, es decir, se deshabilita la entrada de datos digitales a la FPGA (reconfiguración del IP Interfaz AD7768), se vacían los *buffers* de datos del *hardware* y se apaga la UPS.

Hay que destacar que, en supuesto de AC *fail*, no es posible recuperar el sistema por órdenes. Tan solo es posible recuperarlo reiniciando el módulo por medio del interruptor ON/OFF.

A continuación, se observa el esqueleto de la sección de código que atiende al funcionamiento normal del sistema y se desarrolla cada segmento.

```
int main(){
    /* System setup */
    while(/* ac fail status == OK! */){
        /* 1) Manage control and reconfiguration commands */
        if (/* 2) system status == capture ON */){
            /* 3) Send processed samples to OCM */
            /* 4) Read OCM and write processed samples into a datafile */
            /* 5) Send datafile to TFTP server */
            /* 6) Delete sended datafile from RAM FFS */
        }
        /* 7) Read AC fail status */
    }
    /* 8) Shutdown the DAS module */
    return XST_FAILURE;
}
```

1. Gestión de las *órdenes de control y reconfiguración*. Se realiza una lectura del puerto UART del PS y, en función del comando recibido y a través de una estructura *switch-case*, se ejecuta una *funcionalidad*. Por ejemplo, al recibir la orden `'E'`, se ejecuta la función `ModuleShut-Down()`, que apaga la UPS del módulo; al recibir la orden `'R'`, se ejecuta la función `menu()`, que, a través del menú interactivo, permite reconfigurar el módulo; etc.

```
        receivedByte = XUartPs_ReadReg(STDIN_BASEADDRESS,XUARTPS_FIFO_OFFSET);
        switch(receivedByte){
            case 'r':
            case 'R':
                ...
                menu();
                break;
            case 'c':
            case 'C':
                ...
                EnableAD7768Interface();
                break;
            case 'n':
            case 'N':
                DisableAD7768Interface();
                ...
                break;
            case 's':
            case 'S':
                ModuleInitialization();
                ModuleStartUp();
```

```
                break;
            case 'e':
            case 'E':
                ModuleShutDown();
                break;
            case 0:
                break;
            default:
                xil_printf("%c is an unknown command! \r\n", (char) receivedByte);
                break;
        }
    }
```

2. Comprobación del estado del sistema. Se realiza una lectura del bit *enable* del IP Interfaz AD7768. Este bit permite habilitar o deshabilitar la entrada de datos digitales a la FPGA. Si es '0', el sistema está en estado *capture off* y, si es '1', está en *capture on*.

```
if (IpAd7768InterfaceRead(REG_0H10) & MASK_EN_DIS_INTERFACE){}
```

3. Almacenamiento de muestras procesadas en memoria *On-Chip*. Se realiza un *flush* de la región de OCM a escribir y se configura el IP AXI DMA con la dirección destino y el tamaño en *bytes* de datos a almacenar. Una vez configurado, el IP transfiere los *bytes* especificados a la dirección de OCM definida. En este caso, el DMA envía 131 072 *bytes* por transferencia, es decir, 2 048 tramas de muestras procesadas.

```
void StreamToLocalMemory(u32 memoryAddr, u32 numDataPerPacket){
    u32 destAddr;
    u8 *destAddrPtr;
    u32 pktLenByt;

    destAddr = memoryAddr;
    destAddrPtr = (u8 *)destAddr;
    pktLenByt = DMA_M_AXI_S2MM_DATA_WIDTH*numDataPerPacket/8;

    Xil_DCacheFlushRange((UINTPTR)destAddrPtr, pktLenByt*2);
    XAxiDma_SimpleTransfer(&xAxiDma,(UINTPTR)destAddrPtr,pktLenByt,
                           XAXIDMA_DEVICE_TO_DMA);

    while(XAxiDma_Busy(&xAxiDma,XAXIDMA_DEVICE_TO_DMA));

}
```

4. Guardado de muestras procesadas en un fichero de datos. Se crea el nombre del fichero y se escriben los datos almacenados en la memoria *On-Chip* en dicho fichero. El Sistema de Ficheros FAT soporta el *short filename* (SFN) u 8.3 *filename* y, por tanto, se opta por un nombre de fichero conformado por el carácter *d* y siete dígitos (ej.: d0000032.dat), lo que permite la creación de 10 millones de *filenames* diferentes. Los ficheros tienen nombres secuenciales y, en este caso, un tamaño de 131 072 *bytes* (2 048 por 64 *bytes* paquete completo de muestras procesadas por fichero).

```
    int LocalMemoryToRamFile(char fileName[], u32 offset, u32 srcAddr,
                                u32 numDataPerPacket){
        ...
        frResult = f_write(&fil, (const void*)srcAddrPtr, dataToWrittenByt,
                    &dataToWrittenByt);
        if(frResult){
            xil_printf("Error! Write file: %s\r\n", fileName);
            return XST_FAILURE;
        }
        ...
        return XST_SUCCESS;
}
```

5. Transferencia del fichero de datos al servidor TFTP. Se adapta la aplicación cliente TFTP de Xilinx de forma que envía al servidor TFTP en modo *octet* el fichero especificado. Por tanto, se define como archivo a transferir el fichero de datos creado. Este modo de envío no altera el contenido del fichero.

```
    static void tftp_client_send_req(tftp_opcode opcode, ip_addr_t *ip){
       ...
       char fname[20];
       char *mode = "octet";
       ...
       sprintf(fname, fSend);
       ...
       tftp_prepare_request(pbuf->payload, fname, opcode, mode);
       ...
       err = udp_sendto(pcb, pbuf, ip, TFTP_PORT);
       ...
       pbuf_free(pbuf);
   }
```

6. Borrado del fichero transferido del Sistema de Ficheros FAT.

```
   f_unlink(fSend);
```

7. Supervisión de AC *fail*. Se realiza una lectura del estado de AC *fail* a través del IP de Control de la UPS y se actualiza el *status* del sistema.

```
    status = XGpioUpsAcFailRead();
    if(status!= XST_SUCCESS){
        xil_printf("Error! AC Fail!\r\n");
    }
```

8. Apagado del módulo DAS. Se realizan las siguientes operaciones:
    - Deshabilitar la entrada de datos digitales a la FPGA. Se establece el bit de *enable* del IP Interfaz AD7768 a cero.

```
dataRead = IpAd7768InterfaceRead(REG_0H10);
dataWrite = dataRead & 0xFFFFFFFE;
IpAd7768InterfaceWrite(REG_0H10, dataWrite);
```

- Vaciar los *buffers* de datos del *hardware*.
- Apagar la UPS. A través del IP de Control de la UPS, se apaga la etapa analógica (7.3 V) y, tras dos segundos, se apaga la etapa digital (5.4 V).

```
void ModuleShutDown(void){
    XGpioUpsPowerWrite(CR_54VDD);
    sleep(2);
    XGpioUpsPowerWrite(0);
}
```

6.6. Conclusiones

En este capítulo, se ha realizado la integración *hardware-software* del SoC FPGA que ofrece solución al presente Trabajo de Fin de Máster. Se ha explicado el procedimiento de integración de la plataforma *hardware* diseñada con una aplicación *software* de tipo *standalone* y la integración de las librerías y *drivers* disponibles mediante la configuración del BSP.

Se ha desarrollado el *software* empotrado que cumple con los requisitos de la solución final. Dicho de otro modo, se ha desarrollado una aplicación *software* que permite arrancar y configurar el módulo DAS, controlar y configurar el sistema mediante *órdenes de control y configuración*, supervisar el estado del suministro eléctrico externo y apagar el módulo DAS de forma controlada en caso de corte, almacenar las señales procesadas en ficheros de datos y enviar dichos ficheros a un servidor remoto.

De esta parte, hay que destacar que se ha optado por utilizar el protocolo TFTP para el envío de ficheros de datos al servidor remoto y que, por tanto, se ha optado por integrar un cliente TFTP.

Capítulo 7. VALIDACIÓN DE LA INTEGRACIÓN

En este capítulo, se realiza la validación por partes del SoC FPGA diseñado en el presente Trabajo de Fin de Máster y la validación del SoC sobre el módulo DAS físico, es decir, la validación del sistema final.

7.1. Introducción

Es requisito indispensable comprobar que la solución SoC FPGA diseñada en el presente Trabajo de Fin de Máster se comporta de la forma definida. Por consiguiente, en este capítulo, se realiza la validación del SoC tanto de forma aislada utilizando analizadores lógicos y generadores de patrones digitales como sobre el módulo DAS físico. Además, se analiza el rendimiento del sistema en cuanto al envío de datos al servidor, ya que el ancho de banda de salida es un aspecto determinante de la solución.

7.2. Validación del sistema

La presente solución *System-on-Chip* se han sometido a múltiples pruebas de validación, que permiten probar el diseño realizado. A continuación, se listan y se describen los siguientes escenarios de validación:

V1: *Validación de la Inicialización y Configuración del SoC*. Se comprueba la correcta inicialización del *hardware*, detección y lectura de registros del módulo RTC, creación y montaje del Sistema de Ficheros FAT, configuración de los IPs, en especial el IP Marca de Tiempo Real y conexión cliente-servidor TFTP.

V2: *Validación del Arranque y Configuración del Módulo DAS*. Se asegura el adecuado arranque y configuración del módulo DAS.

V3: *Validación de la Adquisición y Procesamiento de las Señales*. Se comprueba la apropiada interpretación de las muestras capturadas por dieciséis canales, adición de marcas de tiempo real, cálculo de promedios por canal, escritura de las muestras procesadas en OCM, creación de ficheros de datos, transferencia de ficheros al servidor TFTP y borrado de ficheros enviados del FFS (RAM).

V4: *Validación de la Supervisión de AC Fail*. Se asegura la correcta supervisión de la señal AC *fail* durante el funcionamiento *normal* del sistema y el apropiado apagado del módulo DAS en caso de detectar un fallo en la fuente externa de alimentación.

V5: *Validación del Sistema*. Se valida el funcionamiento del sistema sobre el módulo DAS físico. Se valida el arranque y la configuración por defecto del módulo y se valida que el sistema se mantiene a la espera de recibir una *orden de control o reconfiguración* al tiempo que supervisa el estado de la fuente externa (AC *fail*). Además, se valida la reconfiguración del módulo a través del menú interactivo, resaltando la configuración de los ADCs por SPI. Se valida el control de la *captura* y el control del encendido y apagado del módulo. Por último, se valida el apagado controlado del módulo en supuesto de caída de la red externa.

7.2.1. V1: Inicialización y Configuración del SoC

Con el objeto de comprobar la correcta inicialización y configuración de la plataforma, se conecta el módulo RTC al conector PMOD JC, se establece la conexión Ethernet utilizando un *switch* Ethernet y se conectan los puertos JTAG y UART de la *ZedBoard* al PC (Figura 107). Seguidamente, se programa la FPGA con la plataforma desarrollada y se lanza la ejecución.

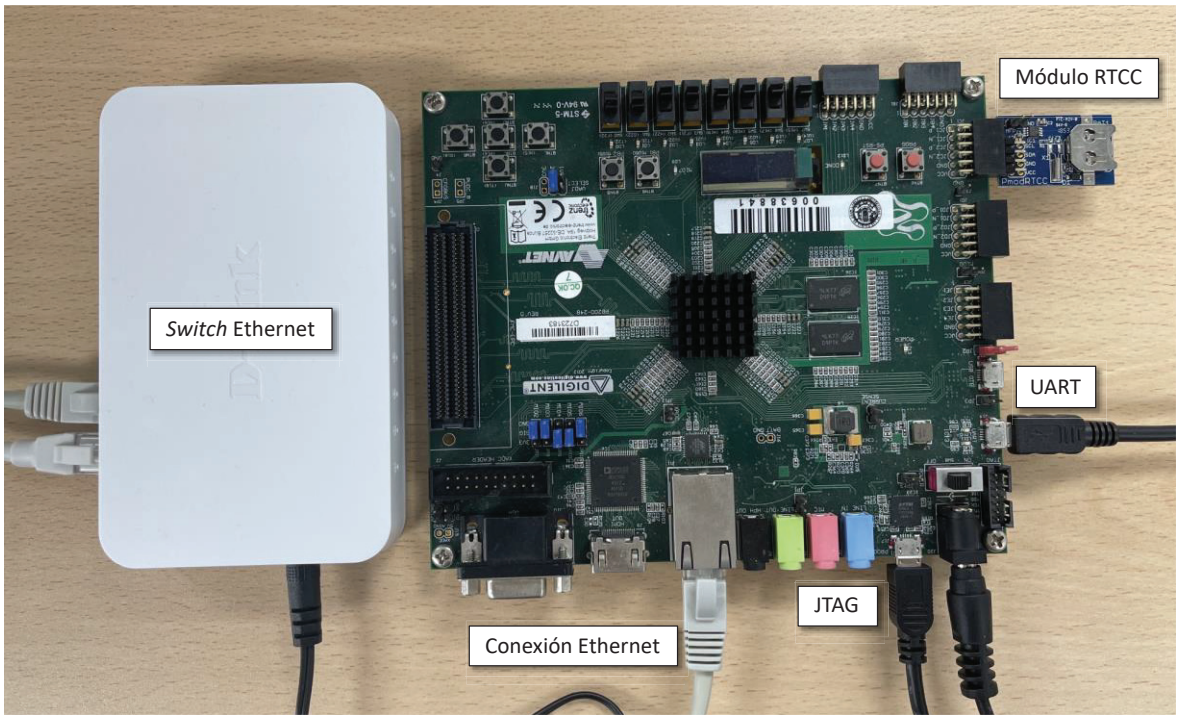

Figura 107. Entorno de validación de la inicialización y configuración de la plataforma

En el Listado 1, se expone el resultado de la inicialización y configuración del *System-on-Chip*. Tal y como se aprecia, se inicializan correctamente los *drivers* de los IPs GPIO e IP DMA, así como los IPs *Quad*-SPI, se detecta el módulo RTC, se crea y monta satisfactoriamente el FFS en RAM, se configuran los IPs de Interfaz AD7768, Marca de Tiempo Real y Cálculo de Promedios, y se vincula con éxito la *ZedBoard* (cliente TFTP) con el servidor TFTP con una velocidad de 1 Gbps.

En este caso, a fin de probar el supuesto más crítico, el IP de *Time Stamp* ha sido configurado de forma que añada marcas de tiempo real con resolución de milisegundos. Por tanto, ha sido ajustado con un tiempo *Unix* en milisegundos aproximado, es decir, con el tiempo *Unix* en segundos, obtenido a partir del RTC, multiplicado por mil. Como se observa, el IP de *Time Stamp*, pese a ser configurado con un tiempo real estricto, se configura rigurosamente con la marca *Unix* en milisegundos indicada.

MINICOM

```
################################################################
#                        TFM PROJECT                           #
#              Selenia Maria Medina Hernandez                  #
#     Master en Electronica y Telecomunicacion Aplicadas       #
################################################################

Success! XGPIO UPS initialized!
Success! XGPIO Master or Slave initialized!
Success! XGPIO Start initialized!
Success! XGPIO Reset initialized!
Success! AXI Quad-SPI initialized!
Success! AXI Direct Memory Access initialized!
Success! RTC is present!
Success! FAT File System initialized!
Success! IP AD7768 Interface configured!
Success! IP Average configured!
Time IP Time Stamp: Mon 2022-05-02 17:03:20 000
```

```
                                                                    MINICOM
      Success! IP Time Stamp configured!
      Initializing Lightweight IP...
      Start PHY autonegotiation
      Waiting for PHY to complete autonegotiation.
      autonegotiation complete
      link speed for phy address 0: 1000
      Configuring default IP ###.###.###.#
      Board IP:       ###.###.###.#
      Netmask :       255.255.255.0
      Gateway :       ###.###.###.#
```

Listado 1. Resultado de la inicialización y configuración del SoC.

7.2.2. V2: Arranque y Configuración Inicial del Módulo DAS

Ahora, tras la satisfactoria inicialización y configuración de la plataforma y con la finalidad de asegurar el adecuado arranque y configuración del módulo, se analizan las salidas del conector FMC dedicadas a dicho propósito utilizando los analizadores lógicos del instrumento *Analog Discovery 2* y un *FMC Pin Header Board* [59], (Figura 108).

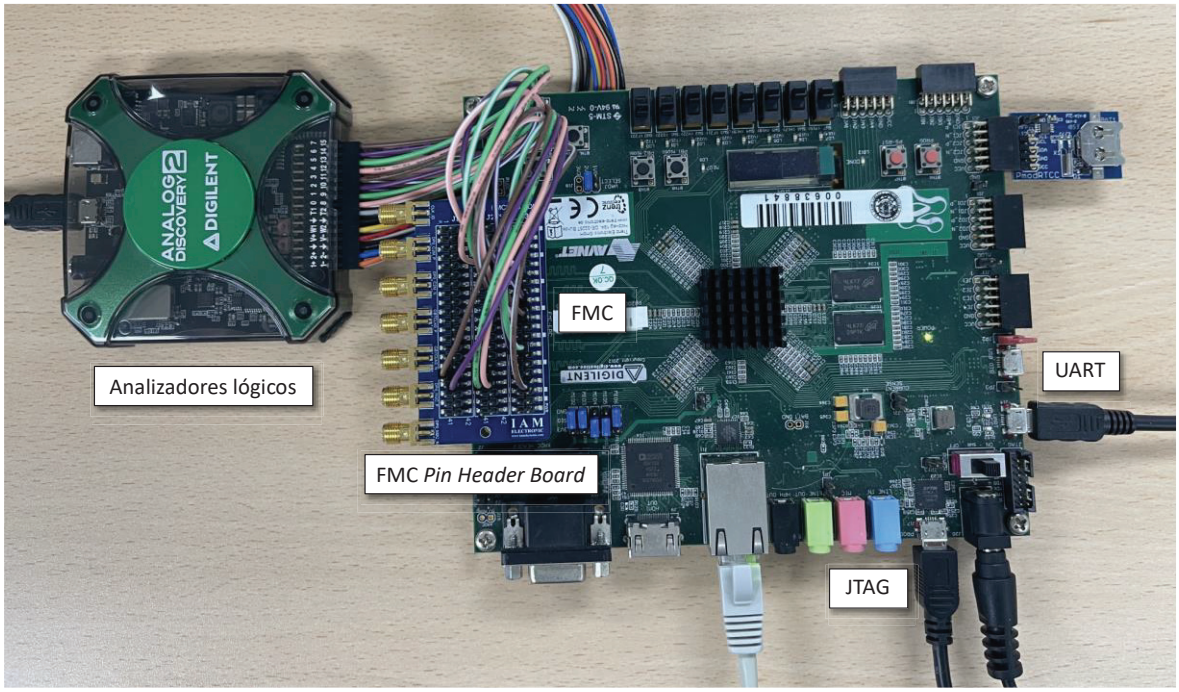

Figura 108. Entorno de validación del arranque y configuración del módulo DAS.

Después de dicho estudio, se corrobora la correcta propagación de la configuración de arranque (factor de amplificación 1/0 dB, modo PIN y ADC-0 como *master*), el pulso de *reset*, la configuración por modo PIN (*fast power*, MCLK/4, *standard*, *x32* y *no-standby*) y el doble pulso de *start* de sincronización de los AD7768, la ganancia escogida (100/40dB) y el encendido dirigido de la etapa digital (5.3 V) y analógica (7.3 V).

A modo de ejemplo, en la Figura 109, se visualiza el análisis lógico de las señales de selección de ganancias de los canales CH0 y CH1, configuración en Modo Pin, *reset* y *start* de los conversores, y control de la UPS.

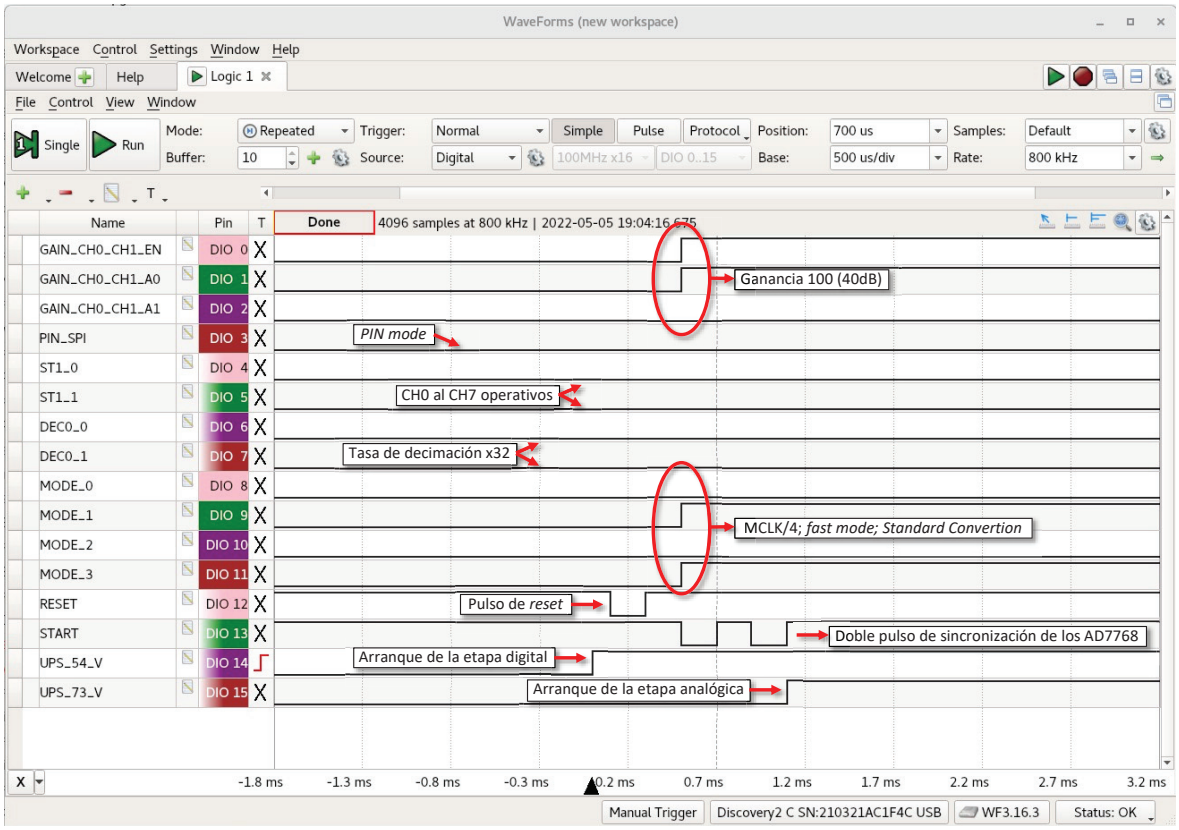

Figura 109. Análisis lógico del arranque y configuración del módulo DAS.

7.2.3. V3: Adquisición y Procesamiento de Señales

Con el fin de validar la adquisición y procesamiento de los datos generados por los conversores AD7768, se inyectan a las entradas dedicadas *dclk*, *drdy* y *dout* del FMC, las señales que generarían dos de estos ADCs durante la captura utilizando los generadores de patrones de dos *Analog Discovery 2* y un *FMC Pin Header Board* (Figura 110).

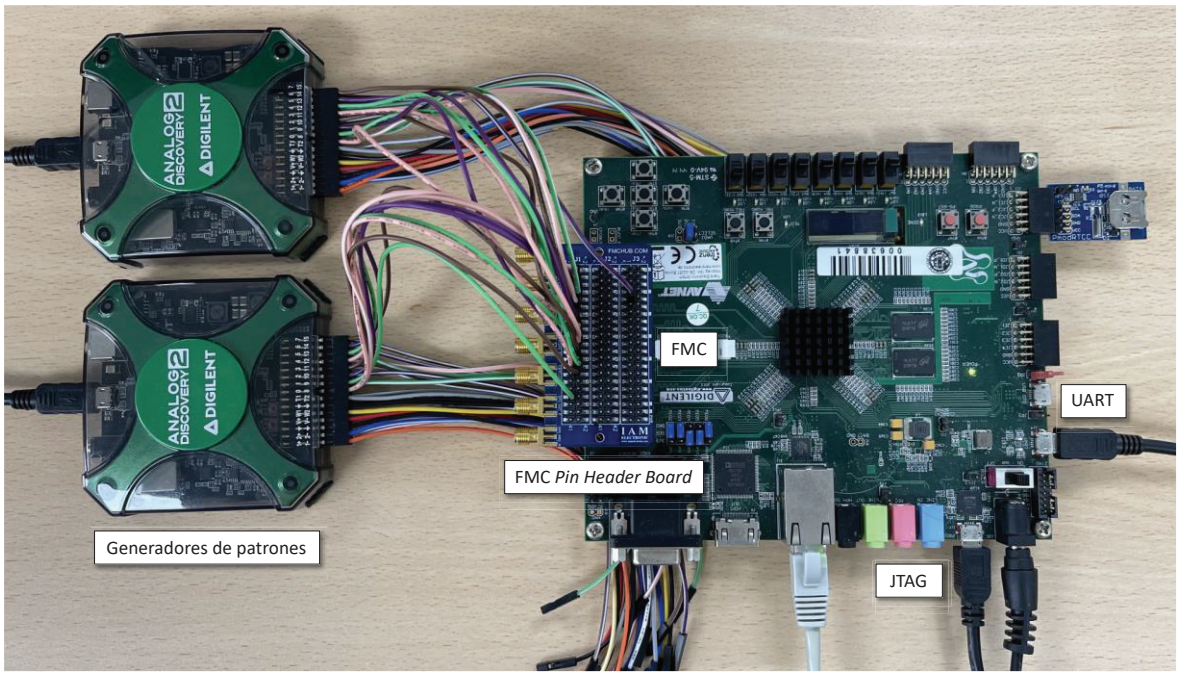

Figura 110. Entorno de validación de la adquisición y procesamiento de las señales.

En este caso, se opta por generar una señal de reloj (DCLK) de 8.33MHz, ya que esta frecuencia es la más próxima a 8.192MHz dentro del rango de frecuencias admitido por el *Analog Discovery 2*, y las salidas de datos (DOUT) mostradas en la Tabla 20. Así, por ejemplo, el canal cero (DOUT0) genera de forma continua las muestras de valores 1.0mV, 1.2mV, 1.4mV y 1.1mV (Figura 111).

La sincronización de las señales de *ready* (DRDY) y datos (DOUT) con dicha señal de reloj, facilita la generación 260kSP/s.

La plataforma se configura de forma que promedie cada cuatro capturas. Por tanto, el promedio de dichas señales es el expuesto en la última columna de la Tabla 20.

Tabla 20. Muestras generadas por los Analog Discovery 2.

	MUESTRA 1	MUESTRA 2	MUESTRA 3	MUESTRA 4	PROMEDIO
CANAL 0	1.0 mV 0h0000068D	1.2 mV 0h000007DD	1.4 mV 0h0000092C	1.1 mV 0h00000735	1.175 mV 0h000007B3
CANAL 1	0 mV	0 mV	0 mV	0 mV	0 mV
CANAL 2	0 mV	0 mV	0 mV	0 mV	0 mV
CANAL 3	0 mV	0 mV	0 mV	0 mV	0 mV
CANAL 4	0 mV	0 mV	0 mV	0 mV	0 mV
CANAL 5	110.2 mV 0h0502D234	120.0 mV 0h0503126E	115.6 mV 0h0502F598	118.4 mV 0h050307F2	116.05mV 0h0502F88B
CANAL 6	0 mV	0 mV	0 mV	0 mV	0 mV

	MUESTRA 1	MUESTRA 2	MUESTRA 3	MUESTRA 4	PROMEDIO
CANAL 7	-2.5 mV 0h07FFEF9D	-2.2 mV 0h07FFF195	-2.8 mV 0h07FFEDA6	-2.1 mV 0h07FFF23C	-2.4 mV 0h07FFF045
CANAL 8	1.0 mV 0h0000068D	1.2 mV 0h000007DD	1.4 mV 0h0000092C	1.1 mV 0h00000735	1.175 mV 0h000007B3
CANAL 9	0 mV	0 mV	0 mV	0 mV	0 mV
CANAL 10	0 mV	0 mV	0 mV	0 mV	0 mV
CANAL 11	0 mV	0 mV	0 mV	0 mV	0 mV
CANAL 12	0 mV	0 mV	0 mV	0 mV	0 mV
CANAL 13	110.2 mV 0h0502D234	120.0 mV 0h0503126E	115.6 mV 0h0502F598	118.4 mV 0h050307F2	116.05 mV 0h0502F88B
CANAL 14	0 mV	0 mV	0 mV	0 mV	0 mV
CANAL 15	-2.5 mV 0h07FFEF9D	-2.2 mV 0h07FFF195	-2.8 mV 0h07FFEDA6	-2.1 mV 0h07FFF23C	-2.4 mV 0h07FFF045

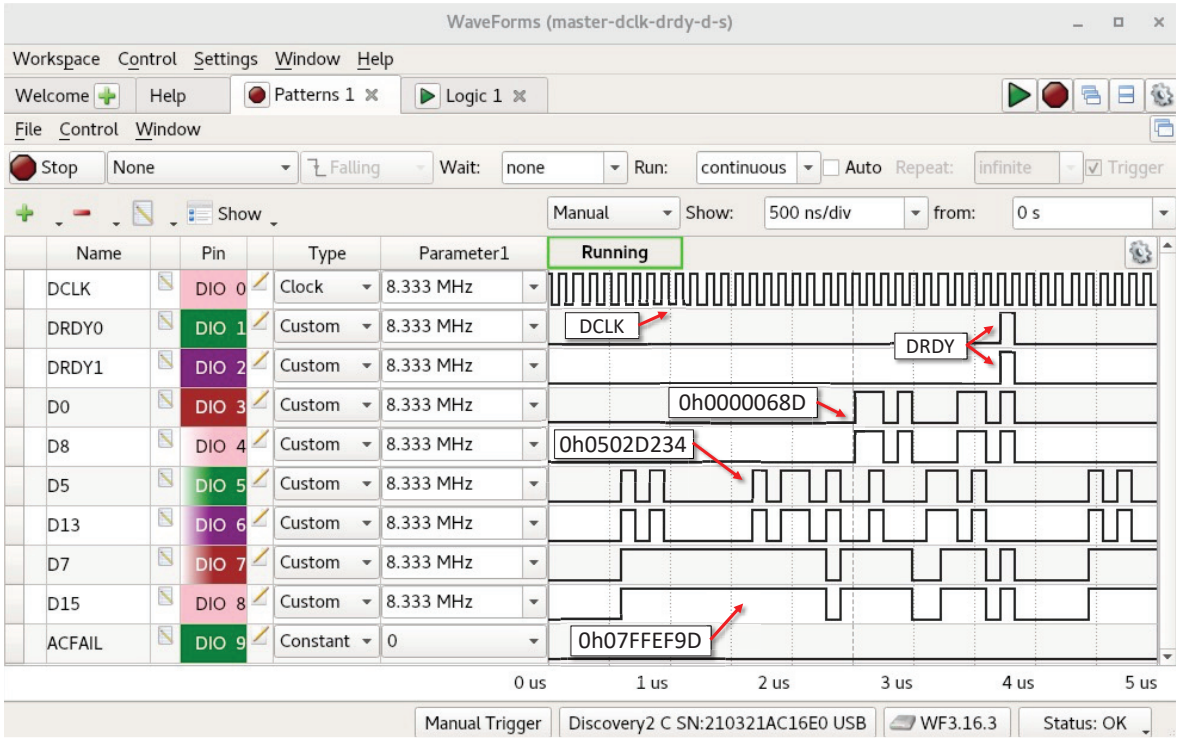

Figura 111. Generación de las señales dclk, drdy y dout.

A continuación, se da comienzo a la adquisición y procesamiento de dichos patrones digitales. En la Figura 112, se observa el contenido de la OCM después de una transferencia de datos del DMA, es decir, después de la escritura de 2 048 tramas de muestras procesadas (131 072 *bytes*).

En esta ventana, se constata que la mitad de la *On-Chip Memory* (131 072 *bytes*) ha sido escrita y que cada cuatro direcciones (512 bits o 64 *bytes*) se halla una nueva trama de muestras procesadas.

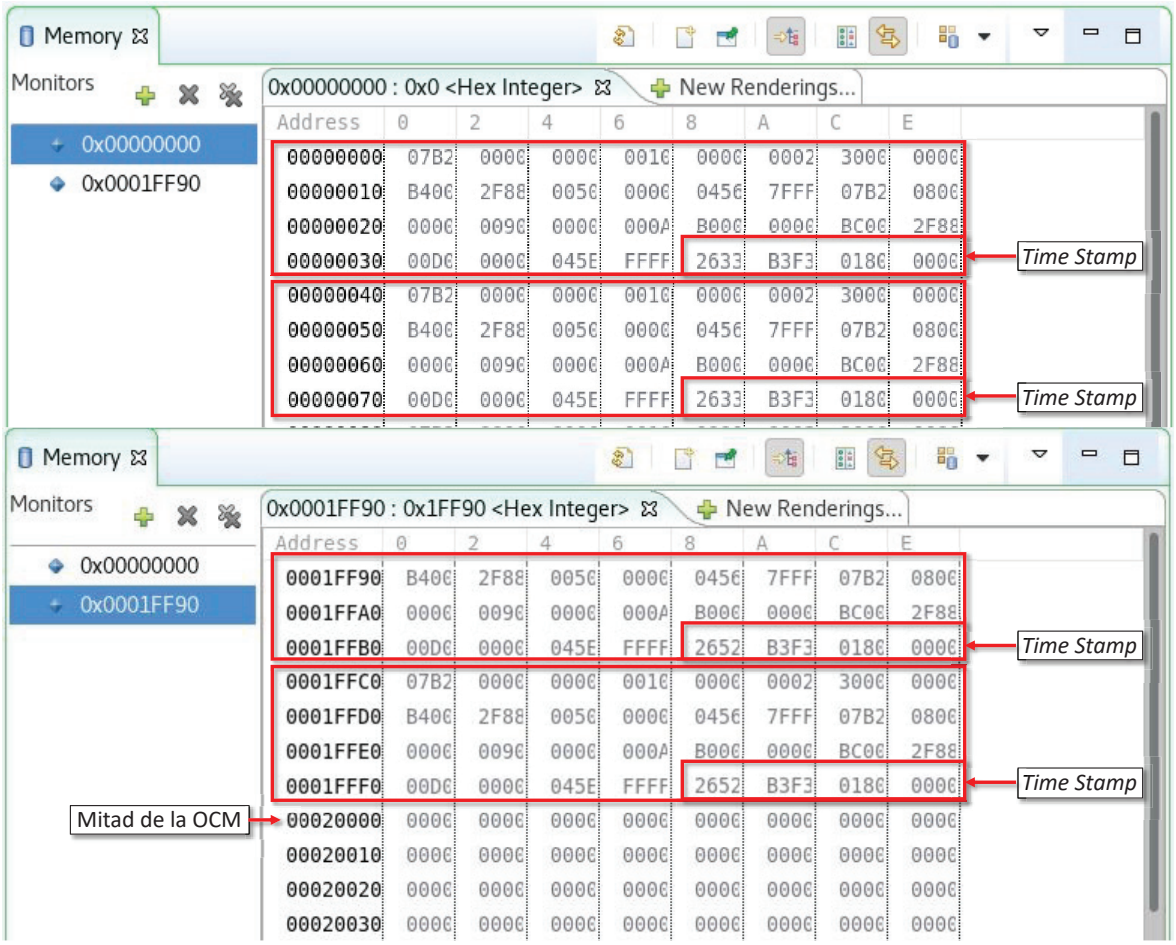

Figura 112. Contenido de la On-Chip Memory.

Una vez corroborado el correcto almacenamiento de los datos en la OCM, se prosigue con la adquisición y procesamiento de las señales digitales. En el Listado 2, se aprecia el detalle de los ficheros almacenados en el servidor TFTP tras dos segundos de captura (aprox.) y, en el Listado 3, se visualiza el contenido del primer fichero de datos (*d0000000.dat*) con el comando *hexdump* de Linux.

Con un DCLK de 8.33MHz y promediando cada cuatro muestras, se generan 65 000 tramas de muestras procesadas por segundo y, como consecuencia, 31.7 ficheros de datos por segundo (5). Con el Listado 2, se comprueba que se han generado y transferido los 64 ficheros de datos esperados después de dos segundos de captura (aprox.) y que presentan un tamaño de 131 072 *bytes*. Además, se confirma que los ficheros escritos en el servidor TFTP han sido borrados del FFS, puesto que este tiene un tamaño de 3.145 MB y no se ha desbordado tras la creación de tal cantidad de ficheros que, de forma conjunta, ocupan 8.38 MB.

En el Listado 3, se muestra que las tramas de muestras procesadas escritas en *On-Chip Memory* (Figura 112), se están guardando de forma correcta y segura en el servidor TFTP en formato fichero de datos.

$$\text{Tasa de tramas procesadas } = \frac{260\,000 \text{ SP/s}}{4} = 65\,000 \text{ tramas/s}$$
$$\text{Tasa de ficheros } = \frac{65\,000 \text{ tramas/s}}{2\,048 \text{ tramas/fichero}} = 31.7 \text{ ficheros/s} \quad (5)$$

CONSOLA

```
vlsiws20:/srv/tftp$ ls -la
drwxr-xr-x 2 tftp tftp 2863104 may 11 17:30 .
drwxr-xr-x 3 root root    4096 mar 10 19:31 ..
-rw-rw-rw- 1 tftp tftp  131072 may 11 17:30 d0000000.dat
-rw-rw-rw- 1 tftp tftp  131072 may 11 17:30 d0000001.dat
-rw-rw-rw- 1 tftp tftp  131072 may 11 17:30 d0000002.dat
...
-rw-rw-rw- 1 tftp tftp  131072 may 11 17:30 d0000061.dat
-rw-rw-rw- 1 tftp tftp  131072 may 11 17:30 d0000062.dat
-rw-rw-rw- 1 tftp tftp  131072 may 11 17:30 d0000063.dat
vlsiws20:/srv/tftp$
```

Listado 2. Listado de ficheros guardados en el servidor TFTP.

CONSOLA

```
vlsiws20:/srv/tftp$ hexdump d0000000.dat
...
001ff80 07b2 0000 0000 0010 0000 0002 3000 0000
001ff90 b400 2f88 0050 0000 0456 7fff 07b2 0800
001ffa0 0000 0090 0000 000a b000 0000 bc00 2f88
001ffb0 00d0 0000 045e ffff 2652 b3f3 0180 0000
001ffc0 07b2 0000 0000 0010 0000 0002 3000 0000
001ffd0 b400 2f88 0050 0000 0456 7fff 07b2 0800
001ffe0 0000 0090 0000 000a b000 0000 bc00 2f88
001fff0 00d0 0000 045e ffff 2652 b3f3 0180 0000
0020000
vlsiws20:/srv/tftp$
```

Listado 3. Contenido del primer fichero de datos (d0000000.dat).

Ahora, con el objeto de comprobar el acertado procesamiento de las señales digitales, se diseña una aplicación C++ que interpreta y escribe en un único fichero de texto de forma legible el contenido de los ficheros de datos (Listado 4), así como diferentes *bash scripts* que filtran por canal y agrupan según la marca de tiempo real las muestras procesadas (Listado 5).

```
CH[{channel}] TS[{ddd yyyy-mm-dd HH:MM:SS mmm}] DV[{sample} V]
```

Listado 4. Formato del fichero de texto generado.

```
CH[{channel}] TS[{ddd yyyy-mm-dd HH:MM:SS mmm}] D[{sample} V]

CH[{channel}] CAP[{num. samples}] TS[HH:MM:SS mmm]
```

Listado 5. Filtrado por canal y agrupación por marcas de tiempo de las muestras procesadas

En el Listado 6 y Listado 7, se expone el resultado de la adquisición y procesamiento de las muestras inyectadas. Como se aprecia, dichas muestras han sido interpretadas, etiquetadas por canal, marcadas con la *marca de tiempo real* en milisegundos y promediadas cada cuatro de forma correcta. También, al concentrar las muestras en función de las *marcas de tiempo* y al filtrarlas por canal, se comprueba que efectivamente se procesan 260kSP/s por canal, ya que cada 65 muestras procesadas se incrementa la *marca de tiempo real* un milisegundo (6).

$$\text{Tasa de muestras procesadas} = \frac{260\ \text{SP/ms}}{4} = 65\ \text{SP/ms} \tag{6}$$

CONSOLA

```
vlsiws20:~/tfm-smedina$ ./capture-converter.csh
vlsiws20:~/tfm-smedina$ more data/capture.txt
S[00] CH[00] TS[Wed 2022-05-11 17:28:18 355] DV[+0.001174 V]
S[01] CH[01] TS[Wed 2022-05-11 17:28:18 355] DV[+0.000000 V]
S[02] CH[02] TS[Wed 2022-05-11 17:28:18 355] DV[+0.000000 V]
S[03] CH[03] TS[Wed 2022-05-11 17:28:18 355] DV[+0.000000 V]
S[04] CH[04] TS[Wed 2022-05-11 17:28:18 355] DV[+0.000000 V]
S[05] CH[05] TS[Wed 2022-05-11 17:28:18 355] DV[+0.116050 V]
S[06] CH[06] TS[Wed 2022-05-11 17:28:18 355] DV[+0.000000 V]
S[07] CH[07] TS[Wed 2022-05-11 17:28:18 355] DV[-0.002400 V]
S[08] CH[08] TS[Wed 2022-05-11 17:28:18 355] DV[+0.001174 V]
S[09] CH[09] TS[Wed 2022-05-11 17:28:18 355] DV[+0.000000 V]
S[10] CH[10] TS[Wed 2022-05-11 17:28:18 355] DV[+0.000000 V]
S[11] CH[11] TS[Wed 2022-05-11 17:28:18 355] DV[+0.000000 V]
S[12] CH[12] TS[Wed 2022-05-11 17:28:18 355] DV[+0.000000 V]
S[13] CH[13] TS[Wed 2022-05-11 17:28:18 355] DV[+0.116050 V]
S[14] CH[14] TS[Wed 2022-05-11 17:28:18 355] DV[+0.000000 V]
S[15] CH[15] TS[Wed 2022-05-11 17:28:18 355] DV[-0.002400 V]
...
vlsiws20:~/tfm-smedina$
```

Listado 6. Resultado de la adquisición y procesamiento de las muestras generadas.

CONSOLA

```
vlsiws20:~/tfm-smedina$ ./capture-viewer.csh 00 | more
CH[00] CAP[65] TS[17:28:18 355]
CH[00] CAP[65] TS[17:28:18 356]
CH[00] CAP[65] TS[17:28:18 357]
...
vlsiws20:~/tfm-smedina$ ./capture-viewer.csh 01 | more
CH[01] CAP[65] TS[17:28:18 355]
CH[01] CAP[65] TS[17:28:18 356]
CH[01] CAP[65] TS[17:28:18 357]
...
vlsiws20:~/tfm-smedina$ ./capture-viewer.csh 14 | more
CH[14] CAP[65] TS[17:28:18 355]
CH[14] CAP[65] TS[17:28:18 356]
CH[14] CAP[65] TS[17:28:18 357]
...
vlsiws20:~/tfm-smedina$ ./capture-viewer.csh 15 | more
CH[15] CAP[65] TS[17:28:18 355]
CH[15] CAP[65] TS[17:28:18 356]
CH[15] CAP[65] TS[17:28:18 357]
...
```

Listado 7. Resultado de la adquisición y procesamiento de las muestras generadas.

Por último, debido a que el índice de creación de ficheros de datos depende del promedio (5), se procede con el análisis de la escritura de ficheros en el servidor TFTP al promediar por 16, 8, 4, 2 y al realizar el *bypass* de las muestras a fin de examinar la flexibilidad del protocolo e interfaz Ethernet. En esta ocasión, se utiliza el analizador de protocolos *Wireshark* para Linux.

En la Figura 113, se visualiza cómo el ancho de banda de la conexión Ethernet se duplica tanto en cuanto se promedia por 16, 8 y 4. Debido a las limitaciones del ancho de banda del servidor TFTP (máx.: 48 Mbps) el resto de las configuraciones se ven perjudicadas. Con ello, se valida que el protocolo TFTP cubre las necesidades de ancho de banda del sistema.

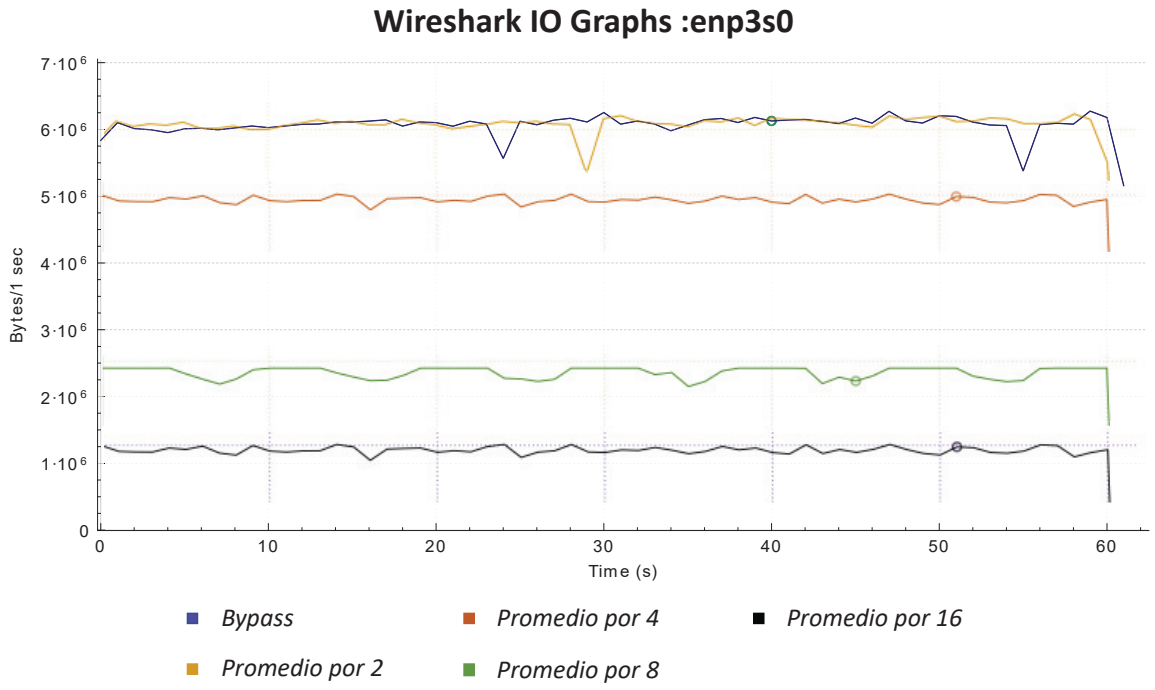

Figura 113. Análisis del protocolo TFTP y ancho de banda de la interfaz Ethernet.

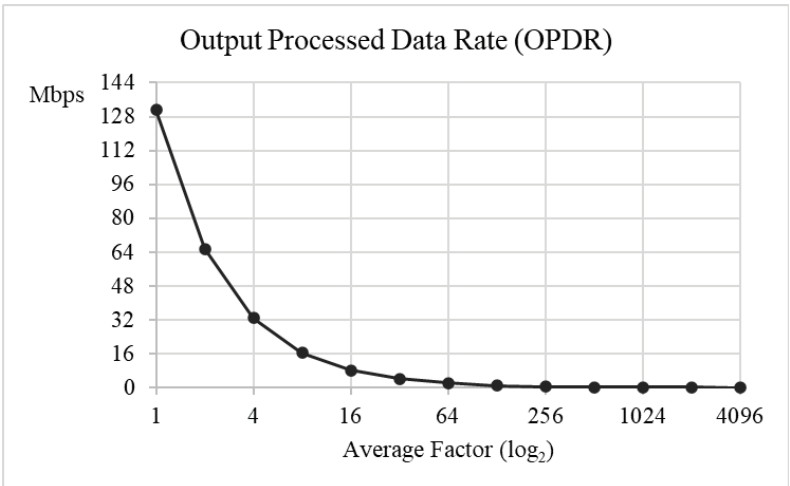

Figura 114. Tasa de datos de salida del sistema en función del factor de promedio

En la Figura 114, se observa la tasa de datos de salida del sistema cuando muestrea a una velocidad de 256 KSPS (*fast mode*, DCLK = MCLK/4 y conversión continua) a través de los dieciséis canales en función del factor del promedio. Como se aprecia, al obtener los datos en crudo (*bypass*) de los dieciséis canales, el sistema genera un tráfico de 16.384MB/s (131.072Mbps) de datos procesados. Extrapolando estos datos, durante una hora de captura se generan 58.98 MB de datos procesados. Dicho con otras palabras, el sistema genera 7 500 ficheros de datos por minuto, donde cada fichero guarda 2 048 muestras procesadas por canal. A medida que se reduce el factor de promedio, la tasa de datos de salida del sistema se duplica, siendo significativa para valores de promedios menores de 4.

7.2.4. V4: Supervisión de AC Fail.

Con el fin de comprobar la correcta supervisión del estado de la fuente externa de alimentación del módulo (señal AC *fail*) durante el funcionamiento *normal* del sistema, después de varios minutos de adquisición y procesamiento de las señales digitales del apartado de validación anterior, se simula una caída de la red externa. Esto consiste en inyectar una señal constante de 3.3V en el pin del FMC dedicado a *acfail* utilizando un generador de patrones del *Analog Discovery 2* y el *FMC Pin Header Board* (Figura 115).

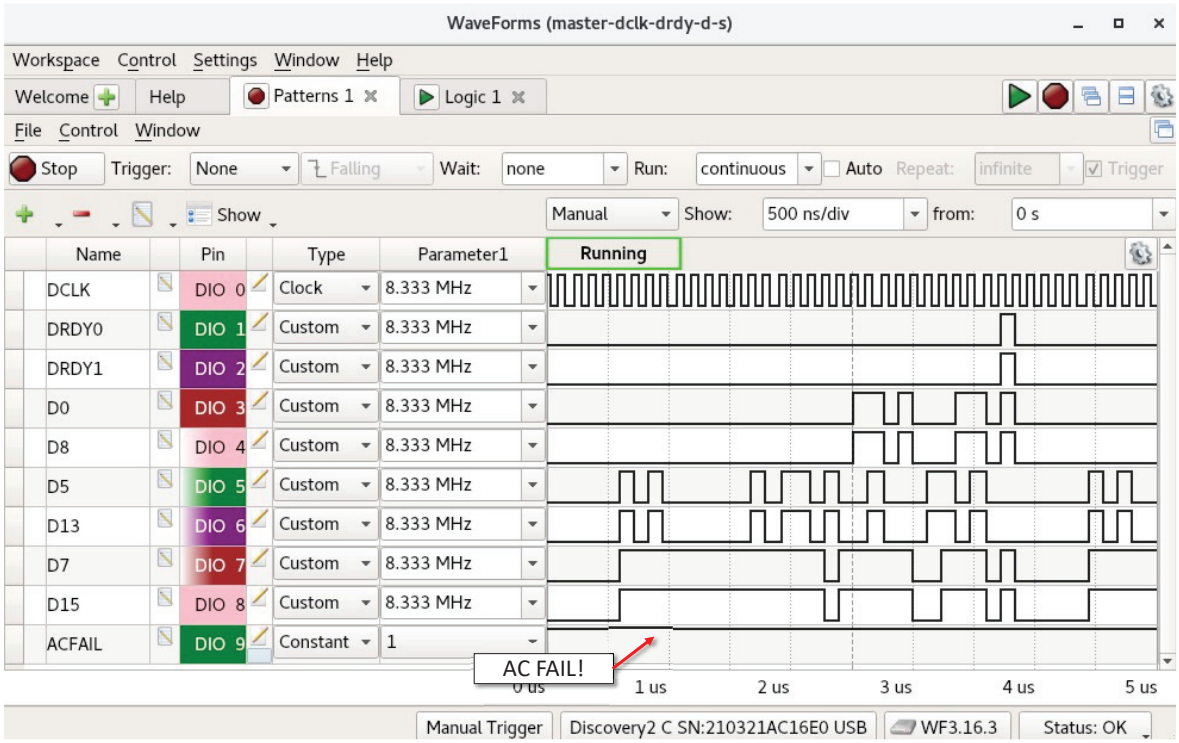

Figura 115. Emulación de caída de la red externa.

Tal y como se observa en el Listado 8, el sistema detecta el fallo en la fuente externa al módulo durante el funcionamiento *normal* del mismo y, debido al fallo, apaga de forma controlada el módulo, es decir, detiene la adquisición y procesamiento de las señales de entrada (*capture off*), vacía los *buffers* de datos del *hardware* y apaga la UPS del módulo (Figura 116).

Por consiguiente, se valida la supervisión de la señal AC *fail* y el apagado del módulo DAS en caso de caída de la red externa.

MINICOM

```
Start capture!
Error! AC Fail!
End capture!
Success! Hardware platform flushed!
DAS Module shutdown!

######################################################################
#                          TFM PROJECT                               #
#              Selenia Maria Medina Hernandez                        #
#     Master en Electronica y Telecomunicacion Aplicadas             #
######################################################################
```

Listado 8. Detección de caída de la red externa y apagado controlado del módulo DAS.

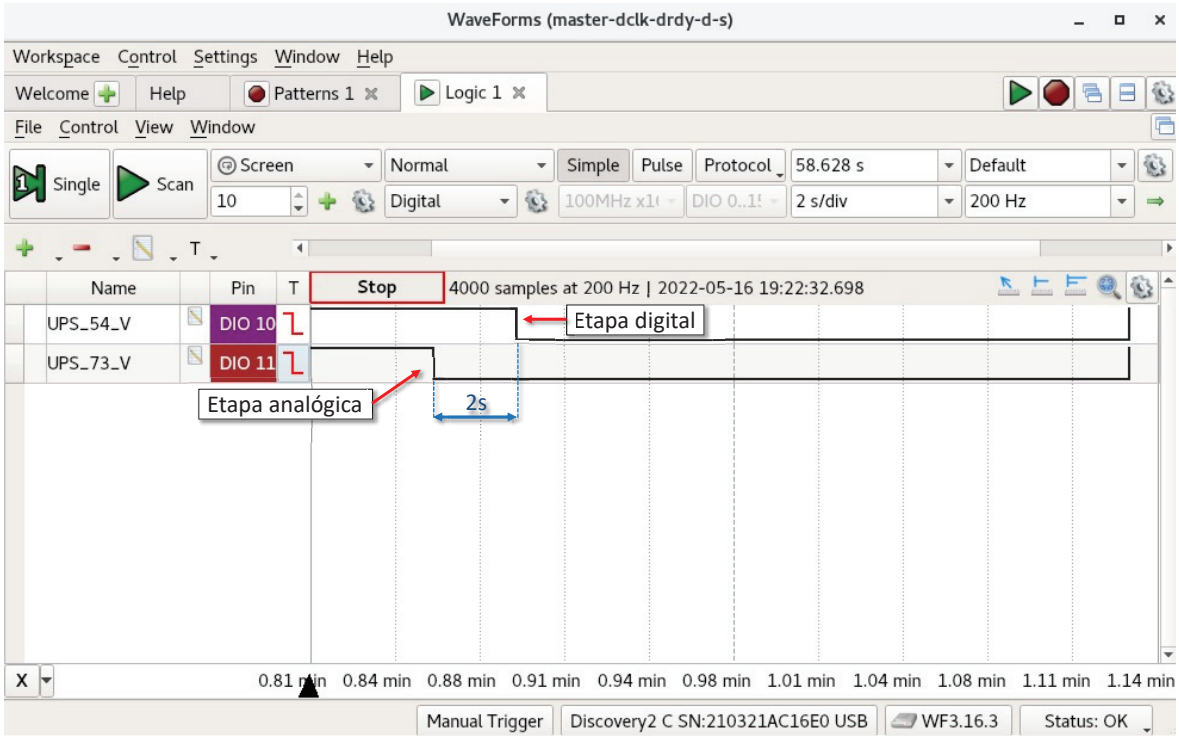

Figura 116. Apagado de la UPS del módulo DAS por caída de la red externa.

7.2.5. V5: VALIDACIÓN DEL SISTEMA.

Con el objeto de validar el funcionamiento del sistema sobre el módulo DAS físico, se conecta el módulo RTC al conector PMOD JC, se establece la conexión Ethernet utilizando un *switch* Ethernet y se conecta la salida JTAG y UART de la *ZedBoard-1* del módulo DAS a un PC. Seguidamente, se programa el SoC (*hardware* y *software)* con el sistema desarrollado y se lanza la ejecución.

Como se aprecia en el Listado 9, el sistema se inicializa, configura y arranca correctamente. En la Figura 117, se observa el correcto encendido de las etapas analógica (V_{an}) y digital (V_{dd}) y configuración de la *placa de adaptación-1* como maestra.

```
                                                                    MINICOM

      ################################################################
      #                        TFM PROJECT                           #
      #              Selenia Maria Medina Hernandez                  #
      #     Master en Electronica y Telecomunicacion Aplicadas       #
      ################################################################

      Success! XGPIO UPS initialized!
      Success! XGPIO Master or Slave initialized!
      Success! XGPIO Start initialized!
      Success! XGPIO Reset initialized!
      Success! AXI Quad-SPI initialized!
      Success! AXI Direct Memory Access initialized!
      Success! RTC is present!
      Success! FAT File System initialized!
      Success! IP AD7768 Interface configured!
      Success! IP Average configured!
      Time IP Time Stamp: Thu 2022-06-30 16:38:43 000
      Success! IP Time Stamp configured!
      Initializing Lightweight IP...
      Start PHY autonegotiation
      Waiting for PHY to complete autonegotiation.
      autonegotiation complete
      link speed for phy address 0: 1000
      Configuring default IP ###.###.###.#
      Board IP:       ###.###.###.#
      Netmask :       255.255.255.0
      Gateway :       ###.###.###.#
      Success! DAS module initialized!
      Success! DAS module started up!
```

Listado 9. Inicialización, configuración y arranque del sistema.

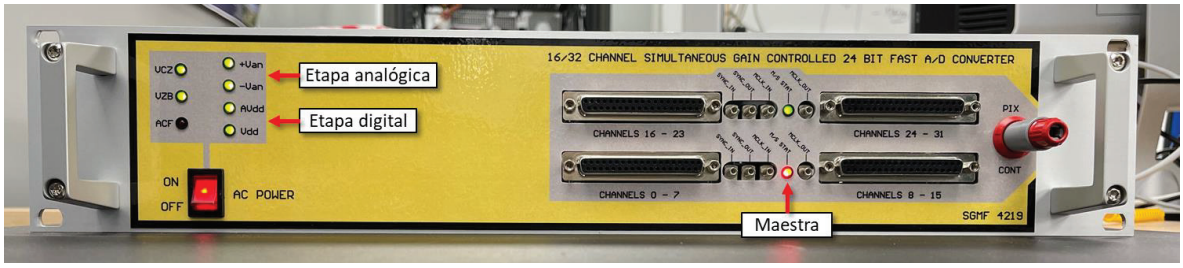

Figura 117. Módulo DAS configurado y arrancado.

Como se muestra en el Listado 10, en este punto, el sistema se mantiene a la espera de recibir *órdenes de control o reconfiguración del módulo* al tiempo que supervisa la señal AC *fail*, dado que,

por ejemplo, al enviar un comando desconocido (`'K'`) este responde. Por lo tanto, se comprueba que el sistema se encuentra en el estado *capture off*.

MINICOM

```
...
Success! DAS module initialized!
Success! DAS module started up!
K is an unknown command!
```

Listado 10. Reconocimiento de comando desconocido.

A continuación, se prueba la reconfiguración del módulo y, por esta razón, se envía el comando `'R'`. Como se observa en el Listado 11, el sistema, al recibir la petición de *reconfiguración*, imprime por consola el menú de reconfiguración.

MINICOM

```
...
R: DAS Module Reconfiguration!

########## DAS Module Configuration ##########

(a): Configuration by PIN mode.
    - Set standby channels.
    - Set channels decimation rate.

(b): Configuration by SPI mode.
    - Set standby channels.
    - Set operative channels.
    - Set channels decimation rate.
    - Set channels offset.
    - Set channels internal gain.

(c): Gains configuration.

(d): Average configuration.

(e): Exit.

##############################################
```

Listado 11. Menú de reconfiguración del módulo DAS.

En este caso, se reconfiguran los conversores por SPI estableciendo los canales CH0 y CH10 en *standby*. Con el Listado 12, se valida la configuración de los ADCs por SPI, puesto que el registro *standby* del ADC0 ahora contiene el valor 0h01 y el del ADC1 el valor 0h04 (Figura 118). Además, se elige promediar cada 8 muestras y amplificar los canales del CH12 al CH15 con una ganancia de 200 (46 dB). En la Tabla 21, se muestra la nueva configuración del sistema.

```
                                                                    MINICOM
      ...
      R: DAS Module Reconfiguration!
      ...

      Configuration by SPI mode:
      ...
      Set standby channels.
      Please, type a number from 00 (CH0) to 15 (CH15): 00 (press ENTER)

      ADC0 - Standby Register Value: 0x01
      ADC1 - Standby Register Value: 0x00
      ...
      Set standby channels.
      Please, type a number from 00 (CH0) to 15 (CH15): 10 (press ENTER)

      ADC0 - Standby Register Value: 0x01
      ADC1 - Standby Register Value: 0x04
      ...
      Average EVERY 8 SAMPLES
      ...
      Gain Configuration:
      ...
      (a):  CH0 and CH1. (Status) Gain: 1 (0 dB) Analog Input +/- 5.0V
      (b):  CH2 and CH3. (Status) Gain: 1 (0 dB) Analog Input +/- 5.0V
      (c):  CH4 and CH5. (Status) Gain: 1 (0 dB) Analog Input +/- 5.0V
      (d):  CH6 and CH7. (Status) Gain: 1 (0 dB) Analog Input +/- 5.0V
      (e):  CH8 and CH9. (Status) Gain: 1 (0 dB) Analog Input +/- 5.0V
      (f):  CH10 and CH11. (Status) Gain: 1 (0 dB) Analog Input +/- 5.0V
      (g):  CH12 and CH13. (Status) Gain: 200 (46 dB) Analog Input +/- 25mV
      (h):  CH14 and CH15. (Status) Gain: 200 (46 dB) Analog Input +/- 25mV
      ...
```

Listado 12. Reconfiguración del módulo DAS.

AD7768 REGISTER MAP DETAILS (SPI CONTROL)

AD7768 REGISTER MAP

See Table 63 and the AD7768-4 Register Map Details (SPI Control) section for the AD7768-4 register map and register functions.

Table 37. Detailed AD7768 Register Map

Reg.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset	RW
0x00	Channel standby	CH_7	CH_6	CH_5	CH_4	CH_3	CH_2	CH_1	CH_0	0x00	RW
		ADC-0	CH7	CH6	CH5	CH4	CH3	CH2	CH1	**CH0**	
		ADC-1	CH15	CH14	CH13	CH12	CH11	**CH10**	CH9	CH8	

Figura 118. Registro Standby del AD7768 [26].

Tabla 21. Nueva configuración del módulo DAS.

Nueva configuración del Sistema de Adquisición y Procesamiento de Datos	
Factor de Amplificación	CH0 al CH11: 1 (0 dB) , CH12 al CH15: 200 (46 dB)
Tipo de Conversión	Continua (*standard*)
Velocidad de Muestreo	256 000 SP/s
Frecuencia de DCLK	MCLK/4 (8.192MHz)
Tasa de Decimación	x32
Canales en *Standby*	CH0 y CH10
Filtro	*sinc5*

Nueva configuración del Sistema de Adquisición y Procesamiento de Datos	
Marca de Tiempo Real	Resolución de milisegundos
Promedios	Cada 8 muestras
Número de Muestras por Paquete	2048

Ahora, con el fin de validar el control de la *captura*, así como la adquisición y procesamiento del sistema, en el canal CH13 se inyecta una señal cuadrada de 1kHz y con una tensión de ± 23.7mV (Figura 119) y, a continuación, se envía el comando `'C'` y, tres segundos después, el comando `'N'`. Con el Listado 13, se valida el control de la *captura*, ya que en el servidor TFTP existen 47 ficheros de datos (*7*), y, por tanto, se valida que el sistema entra en el estado *capture on* y que regresa al estado *capture off* al recibir los respectivos comandos.

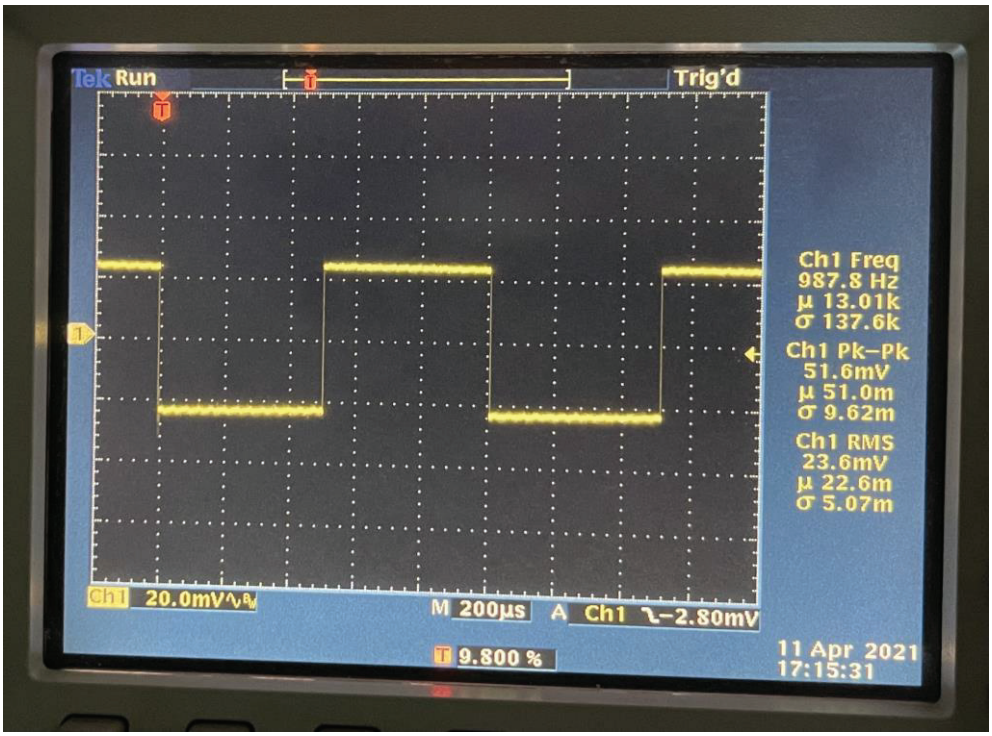

Figura 119. Señal cuadrada a inyectar en el canal CH13.

CONSOLA

```
vlsiws20:/srv/tftp$ ls -la
drwxr-xr-x 2 tftp tftp 2863104 jun 30 16:41 .
drwxr-xr-x 3 root root    4096 mar 10 19:31 ..
-rw-rw-rw- 1 tftp tftp  131072 jun 30 16:41 d0000000.dat
-rw-rw-rw- 1 tftp tftp  131072 jun 30 16:41 d0000001.dat
-rw-rw-rw- 1 tftp tftp  131072 jun 30 16:41 d0000002.dat
...
-rw-rw-rw- 1 tftp tftp  131072 jun 30 16:41 d0000045.dat
-rw-rw-rw- 1 tftp tftp  131072 jun 30 16:41 d0000046.dat
vlsiws20:/srv/tftp$
```

Listado 13. Contenido del servidor TFTP tras la captura.

$$\text{Tasa de tramas procesadas } = \frac{256\,000\ \text{SP/s}}{8} = 32\,000\ \text{tramas/s}$$
$$\text{Tasa de ficheros } = \frac{32\,000\ \text{tramas/s}}{2\,048\ \text{tramas/fichero}} = 15.63\ \text{ficheros/s} \quad (7)$$

Luego, con el Listado 14, se valida la adquisición y procesamiento del sistema, ya que se observa que los canales CH0 y CH10 están detenidos o en *standby*, y que la señal cuadrada inyectada por el canal CH13 se promedia cada 8 muestras (8) y se amplifica por 200 (46 dB).

CONSOLA

```
vlsiws20:~/tfm-smedina$ ./capture-converter.csh
vlsiws20:~/tfm-smedina$ ./capture-value.csh 00
CH[00] TS[Thu 2022-06-30 16:40:49 185] DV[+0.000000 V]
CH[00] TS[Thu 2022-06-30 16:40:49 185] DV[+0.000000 V]
CH[00] TS[Thu 2022-06-30 16:40:49 185] DV[+0.000000 V]
CH[00] TS[Thu 2022-06-30 16:40:49 185] DV[+0.000000 V]
...
vlsiws20:~/tfm-smedina$ ./capture-value.csh 10
CH[10] TS[Thu 2022-06-30 16:40:49 185] DV[+0.000000 V]
CH[10] TS[Thu 2022-06-30 16:40:49 185] DV[+0.000000 V]
CH[10] TS[Thu 2022-06-30 16:40:49 185] DV[+0.000000 V]
CH[10] TS[Thu 2022-06-30 16:40:49 185] DV[+0.000000 V]
...
vlsiws20:~/tfm-smedina$ ./capture-value.csh 13
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[+4.732941 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[+4.732219 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[+4.732521 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[+4.732251 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[+4.735789 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[+4.716495 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-2.942200 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.668892 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.673990 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.671961 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.664053 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.669297 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.667363 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.665758 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.663926 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.663175 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.661878 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.660243 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.658886 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.657841 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.656354 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[-4.675519 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[+3.398876 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[+4.734926 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[+4.734131 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[+4.734733 V]
CH[13] TS[Thu 2022-06-30 16:40:49 185] DV[+4.734418 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[+4.734329 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[+4.734011 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[+4.733954 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[+4.733242 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[+4.733766 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[+4.732966 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[+4.732269 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[+4.732475 V]
```

CONSOLA

```
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[+4.734052 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[+4.727652 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[+4.743576 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[-3.858320 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[-4.679496 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[-4.671901 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[-4.671581 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[-4.664018 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[-4.668294 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[-4.667327 V]
CH[13] TS[Thu 2022-06-30 16:40:49 186] DV[-4.665806 V]
...
vlsiws20:~/tfm-smedina$
```

Listado 14. Salida del Módulo DAS. Adquisición y procesamiento de las señales de entrada.

$$\text{Tasa de promedios } = \frac{256\,000\ \text{SP/s}}{8} = 32\,000\ \text{A/s} = \ 32\ \text{A/ms} \qquad (8)$$

Seguidamente, con el objeto de probar el control del apagado y encendido del sistema, se envía el comando `'E'` y, minutos después, el comando `'S'`. Como se observa en la Figura 120, el módulo DAS se apaga correctamente y, en la Figura 121 y en el Listado 15, se aprecia el adecuado arranque y configuración del sistema.

Figura 120. Apagado del módulo DAS.

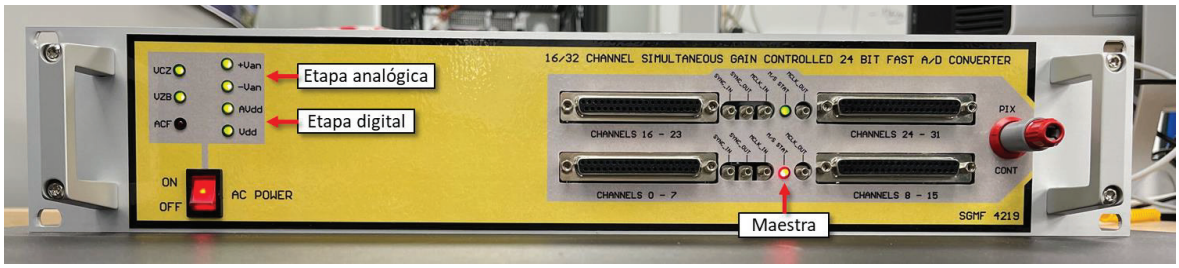

Figura 121. Encendido del módulo DAS.

MINICOM

```
...
E: DAS Module Shutdown!
S: DAS Module Initialized and Start Up!
Success! DAS module initialized!
Success! DAS module started up!
```

Listado 15. Apagado y encendido del módulo DAS.

Finalmente, con miras a probar que el sistema responde y, de forma correcta, ante una posible caída de la red externa, se apaga el módulo DAS de forma manual a través del interruptor ON/OFF. Con el Listado 16, se valida la detección del fallo en la fuente externa y el apagado controlado del sistema. En la Figura 122, se observa que existe una caída de la red y que tanto la etapa analógica (V_{an}) y como la digital (V_{dd}) están apagadas.

```
                                                                    MINICOM

      Start capture!
      Error! AC Fail!
      End capture!
      Success! Hardware platform flushed!
      DAS Module shutdown!

      ################################################################
      #                          TFM PROJECT                         #
      #              Selenia Maria Medina Hernandez                  #
      #     Master en Electronica y Telecomunicacion Aplicadas       #
      ################################################################
```

Listado 16. Detección de AC fail y apagado controlado del módulo DAS.

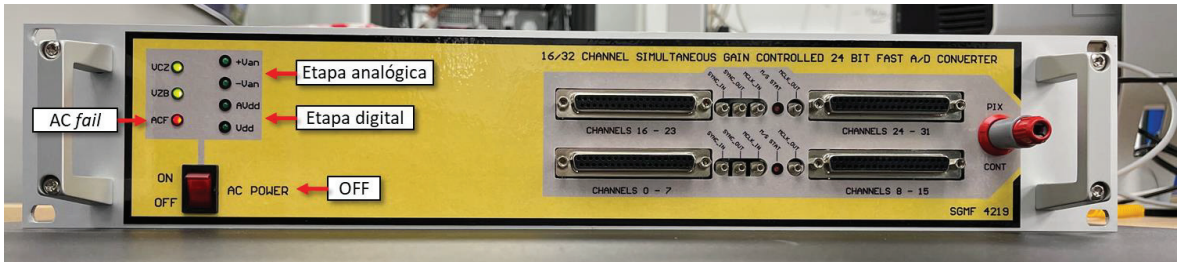

Figura 122. Apagado del módulo DAS por fallo en la fuente externa.

7.3. Conclusiones

En este capítulo, se ha validado el *System-on-Chip* FPGA desarrollado en el presente Trabajo de Fin de Máster. Se ha comprobado que, de forma correcta, el SoC se inicializa y configura, arranca y configura con la configuración por defecto definida el módulo DAS, captura y procesa las dieciséis señales de entrada, transfiere al servidor remoto los ficheros con las señales procesadas, supervisa el estado del suministro eléctrico externo (AC *fail*), apaga de forma controlada el módulo DAS en caso de corte y permite controlar y configurar el sistema mediante *órdenes de control y configuración del módulo* (ej.: seleccionar las ganancias de amplificación de las señales de entrada, controlar la *captura*, encender y apagar el DAS, configurar el *offset* de los ADCs por canal, etc.).

La validación del sistema ha resultado ser exitosa tanto de forma aislada utilizando analizadores lógicos y generadores de patrones digitales como sobre el módulo DAS físico.

Capítulo 8. CONCLUSIONES

En este capítulo, se exponen las conclusiones y trabajos futuros del actual Trabajo de Fin de Máster, así como la comparativa con otros Sistemas de Adquisición de Datos.

8.1. Conclusiones

Durante el presente Trabajo de Fin de Máster ha sido necesario abordar distintos aspectos para el desarrollo de una solución o producto comercial.

Ha sido preciso estudiar el contexto final de uso del módulo DAS, siendo en esta oportunidad la necesidad de un Sistema de Adquisición de Datos capaz de amplificar, capturar y procesar múltiples señales analógicas del orden de milivoltios de forma simultánea, con el objetivo de tomar ciertas decisiones en el diseño de la solución SoC FPGA, como la arquitectura del sistema (paralela o serie).

El módulo DAS utilizado se encuentra en proceso de construcción y, por tanto, de validación. Por esta razón, la solución final se ha desarrollado siguiendo una metodología de diseño incremental, compaginando el diseño del SoC con la validación del equipo. Primero se ha diseñado el *hardware-software* necesarios para el arranque y configuración del módulo DAS y adquisición y procesamiento elemental y, después, se ha diseñado el SoC con la funcionalidad completa. Este modo de desarrollo, además, ha permitido ampliar el conocimiento sobre los componentes electrónicos del equipo, como el conversor AD7768, y, por tanto, ha permitido afinar la solución SoC final. No obstante, esto último, ha supuesto el rediseño del *hardware* reprogramable.

La utilización del *diseño basado en plataformas* desde el comienzo del proyecto ha facilitado la metodología de diseño incremental citada. Esto ha consistido en desarrollar múltiples plataformas dedicadas a una operación específica y, una vez probada su operatividad, combinarlas de forma coherente en una única plataforma.

A grandes rasgos, se parte de una plataforma compuesta por bloques AXI GPIO dedicados al arranque de la fuente residente de alimentación y sincronización y reseteo de los ADCs, una unidad

de configuración de los ADCs por modo PIN, una unidad de selección de ganancias, un conjunto de unidades elementales de adquisición y procesamiento de los datos generados por los ADCs y el bloque del Sistema de Procesamiento (PS).

Con esta plataforma, se procede con el arranque y configuración elemental del módulo y con la captura y procesamiento básico de las señales de entrada. Esto permite tanto validar como estudiar en profundidad gran parte del funcionamiento del equipo. Una vez que esta plataforma está operativa, se continúa añadiendo el resto de las funcionalidades, como la unidad de configuración de los ADCs por modo SPI, y afinando el flujo de procesamiento de las señales digitales. Con ello, tras tener operativas las nuevas funcionalidades, se obtiene una nueva plataforma con la funcionalidad aumentada.

En las fases finales del Trabajo de Fin de Máster se ha integrado el *stack* TCP/IP utilizado para el envío de ficheros de datos con las señales procesadas a una estación remota empleando el protocolo TFTP. Esto ha requerido el desarrollo de un Sistema de Ficheros FAT local, en este caso en DRAM por razones de eficiencia, para el almacenamiento de las muestras procesadas en ficheros de datos.

Durante el desarrollo del proyecto, se han estudiado múltiples arquitecturas de procesamiento. En una primera instancia, se analiza una arquitectura paralela que soporta el procesamiento personalizado por canal de los ADCs, por ejemplo, capturar con promedios de 8 por el canal CH0 y, al mismo tiempo, capturar con promedios de 64 por el canal CH13. Esto ofrece flexibilidad al sistema. Sin embargo, produce una elevada cantidad de datos a la salida, debido a la necesidad de añadir marcas de tiempo real por canal (65.5 MB/s en el caso más desfavorable [47]).

En otra aproximación, se estudia una arquitectura serie que marca las 16 muestras capturadas de forma simultánea con la misma marca de tiempo real. Esto reduce la flexibilidad del sistema, dado que este se ve limitado a capturar y procesar con la misma configuración por los 16 canales. No obstante, esta arquitectura reduce el ancho de banda de salida en un factor de 4 frente a la paralela y presenta ventajas cuando el objetivo es obtener múltiples canales de captura paralelos idénticos, es decir, en función de la utilidad final, una arquitectura u otra puede tener unos puntos a favor u otros.

El desarrollo de la plataforma sobre el dispositivo SoC FPGA Zynq-7000 ha permitido implementar arquitecturas heterogéneas de procesamiento *hardware-software*. En este Trabajo de Fin de Máster se ha hecho énfasis en el diseño *hardware* de la arquitectura de adquisición y procesamiento del sistema, así como de control, y en el diseño *software* de una aplicación *standalone* que complementa dichas tareas (ej.: control del sistema por comandos, envío de ficheros a un servidor, etc.). En definitiva, se ha desarrollado una solución SoC FPGA eficiente que explota el paralelismo que ofrece el *hardware* y la flexibilidad del *software*.

En paralelo al diseño de este *System-on-Chip*, se han estudiado y desarrollado otras soluciones que parten de la plataforma *hardware* desarrollada. Como, por ejemplo, una que implementa Open-

AMP, en la que el primer microprocesador (CPU-0) ejecuta el SO Linux y el segundo (CPU-1) una aplicación *bare-metal* dedicada a las tareas que exigen un control estricto del tiempo de computación. Esta al disponer del SO Linux permite volcar directamente los ficheros en un Sistema de Ficheros en red (NFS o SMB) y controlar y configurar el sistema a través de una interfaz de usuario de tipo CLI.

Actualmente, también a partir del *hardware* desarrollado, se está construyendo un sistema optimizado que reduce el ancho de banda de salida. Con esta finalidad, se pretende dotar a la plataforma *hardware* de un bloque compresor de datos en modo flujo a la salida de la cadena de procesamiento.

Todo ello, confirma que la plataforma diseñada es una plataforma de referencia y que permite desarrollar nuevas soluciones que permiten la evolución del producto.

Con respecto a la solución SoC FPGA desarrollada, se concluye que en el presente Trabajo de Fin de Máster se ha obtenido un *System-on-Chip* FPGA capaz de adquirir y procesar hasta dieciséis señales digitales de forma simultánea con un consumo de potencia moderado y que permite controlar y configurar un módulo DAS. Este SoC integrado en el módulo DAS utilizado ofrece un Sistema de Adquisición y Procesamiento de Datos de dieciséis canales simultáneos que amplifica con diferentes ganancias, captura a alta velocidad (256kSP/s) y con una resolución de 24 bits, añade marcas de tiempo real Unix con resolución de segundos o milisegundos a las muestras, calcula el promedio de las muestras y envía a una estación remota a través de Ethernet las señales procesadas en formato ficheros de datos. Además, con este SoC se obtiene un DAS que puede ser controlado y configurado por medio de comandos y un menú de configuración (ej.: controlar la *captura*, seleccionar las ganancias de amplificación de las señales, poner en *standby* los canales deseados, apagar o encender el equipo, etc.) y que se detiene de forma controlada en caso de corte del suministro eléctrico.

Comparativa con otros DAS

En la literatura existen referencias a Sistemas de Adquisición de Datos. Sin embargo, es imposible realizar una comparativa directa con el DAS desarrollado, ya que dichas soluciones están diseñadas en base a criterios y objetivos científicos diferentes.

Por esta razón, se han identificado un conjunto de parámetros comunes al presente DAS que permiten caracterizar cada uno de estos sistemas. Dichos parámetros son los siguientes:

a) Resolución de la conversión A/D en bits (B).
b) Frecuencia de muestreo (F).
c) Número de canales de captura (C).
d) Simultaneidad entre canales de captura (S).
e) Reconfigurabilidad del *hardware* (R).

f) Marcas de tiempo real a las muestras (M).

g) Ancho de banda de salida (BW).

h) Potencia consumida del SoC FPGA (P).

i) Escalabilidad (E).

j) Conectividad TCP/IP (T).

k) Sistema de almacenamiento de datos (DS).

l) Interfaces de control del sistema (I).

Tabla 22. Comparativa entre DAS.

	DAS (TFM)	**DAS-1** [60]	**DAS-2** [61]	**DAS-3** [62]	**DAS-4** [63]
Resolución A/D en bits	**24-bits**	12-bits	12-bits	16-bits	