Universidad de Alcalá
Escuela Politécnica Superior

Grado de Ingeniería en Electrónica de Comunicaciones

Trabajo Fin de Grado

Diseño de una arquitectura basada en FPGA para la emulación de sistemas de adquisición de consumo eléctrico en redes de energía

Autor: Miguel Tapiador Luque

Tutor: Álvaro Hernández Alonso

2020/2021

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado de Ingeniería en Electrónica de Comunicaciones

Trabajo Fin de Grado

Diseño de una arquitectura basada en FPGA para la emulación de sistemas de adquisición de consumo eléctrico en redes de energía

Autor: Miguel Tapiador Luque

Tutor: Álvaro Hernández Alonso

TRIBUNAL:

Presidente: Ignacio Fernández Lorenzo

Vocal 1º: Emilio José Bueno Peña

Vocal 2º: Álvaro Hernández Alonso

FECHA: Julio 2021

Resumen

En este trabajo se realiza el diseño e implementación de una arquitectura SoC (System-on-Chip) para un emulador de un contador inteligente para la experimentación con técnicas NILM, así como el sistema de adquisición de datos que recoge la información enviada por el emulador. El sistema se implementa en la Zybo z7-20, con un dispositivo SoC de la familia Zynq 7000 de Xilinx, Inc; para tanto el envío como la recepción de datos se utilizan dos conversores con interfaz Pmod conectados entre sí. En el procesador se cargarán inicialmente las bases de datos que utilizará el emulador y se enviarán a un terminal una vez pasen por todo el sistema para el análisis de los resultados; en la FPGA se implementan tanto los drivers de los conversores como el controlador que los gestiona.

Palabras clave: emulador, NILM, Zynq, adquisición de datos.

Abstract

This work involves the design and implementation of a SoC (System-on-Chip) architecture for a smart meter emulator for experimentation with NILM techniques, as well as the data acquisition system that collects the information sent by the emulator. The system is implemented in the Zybo z7-20 platform, with a SoC device of the Zynq 7000 family from Xilinx Inc; for both sending and receiving data are used two converters with Pmod interface connected to each other. The processor will initially load the databases that will be used by the emulator and send them to a terminal for the analysis; both the drivers of converters and the controllers that manage them are implemented in the FPGA.

Key words: emulator, NILM, Zynq, data acquisition.

Resumen extendido

El crecimiento demográfico y el aumento del consumo energético que eso conlleva, así como el deterioro medioambiental, dan lugar a la necesidad de hacer un consumo más eficiente de la energía. Ahí es donde entran las técnicas NILM (Non-Intrusive Load Monitoring). Éstas son técnicas computacionales de desagregación de la energía, es decir: dado el valor total de consumo energético, son capaces de separar que dispositivos electrónicos consumen en cada momento, permitiendo un mayor conocimiento de lo que se utiliza y un consumo más responsable. Gracias a la aparición de los contadores inteligentes, dispositivos que permiten monitorizar el consumo total de la vivienda o edificio al que estén conectados y enviarlo de forma remota para su análisis, estas técnicas se han impulsado en los últimos años.

El sistema realizado en este trabajo se implementa sobre una plataforma Zybo z7-20, una arquitectura SoC, haciendo uso del dispositivo FPGA de la familia Zynq 7000 disponible, por lo que para su implementación se utilizará tanto un procesador como lo lógica programable; además de la Zynq también se utilizarán dos conversores con interfaz Pmod, también de Digilent, para sacar y meter los datos de la tarjeta. El sistema se divide principalmente en dos partes: emulación y adquisición de datos.

La primera parte trata de emular a un contador inteligente que envía los datos de consumo que recoge mediante una base de datos; dicha base contiene datos de voltaje y corriente de una vivienda muestreados a 12kHz. Estos datos se transfieren desde archivos .mat a ficheros .h mediante Matlab; una vez en forma de ficheros, se cargan como librerías en el proyecto del procesador, desde donde se envían a memorias FIFO implementadas en la FPGA a través de AXI-Lite4, protocolo de comunicación que enlaza la parte del procesador y la configurable de la FPGA. Una vez en las memorias, un controlador gestiona su envío al conversor digital-analógico a través de un driver, lo que finaliza la parte del emulador.

La segunda es la de adquisición de datos. El conversor digital-analógico de la parte anterior se conecta directamente con el conversor analógico-digital de esta parte. El mismo controlador que gestiona el driver del otro conversor también gestiona el del conversor analógico-digital, pues trabajarán paralelamente: cuando se envíe un dato a través del DAC, el ADC debe convertirlo y recibirlo en el controlador antes de que se envíe un nuevo dato por el primer conversor. Una vez dentro del sistema de nuevo, el controlador va guardando los datos en unas memorias FIFO diferentes a las de la anterior etapa; el procesador se encargará de leer estas memorias antes de que rebosen para que no se pierdan datos. Una vez en el procesador, este se comunica a través del puerto serie con un terminal con Matlab, donde se mostrarán los datos originales y los convertidos para analizar la viabilidad del SoC para la experimentación con técnicas NILM.

Índice

Índice de figuras

Índice de tablas

1. Introducción

En este capítulo se pone en contexto el objetivo y desarrollo de este proyecto. Se hará una breve explicación de la motivación que ha llevado al desarrollo de este proyecto, así como la definición de los objetivos a conseguir y una descripción general de la estructura del documento.

1.1 Contexto

El aumento del consumo energético global y el crecimiento demográfico de las últimas décadas, además del problema medioambiental que esto supone, exige que se ponga el área de la energía en el foco de estudio. Dado que el agotamiento de los recursos fósiles es una realidad, es importante la investigación y el desarrollo de las energías renovables, pero también lo es hacer un consumo más eficiente y responsable de la energía. Ahí es donde entran las técnicas NILM.

Las técnicas NILM son técnicas computacionales capaces de identificar qué dispositivos electrónicos se encuentran consumiendo en cada momento dado el consumo total energético. Aunque los primeros estudios sobre esta línea datan de la última década del siglo XX, de la mano de George Hart, no es hasta la aparición de los contadores inteligentes en los últimos años que se han empezado a desarrollar. Estos dispositivos permiten recoger el consumo eléctrico total de la vivienda o edificio al que estén conectados y enviarlo de manera inalámbrica para su análisis.

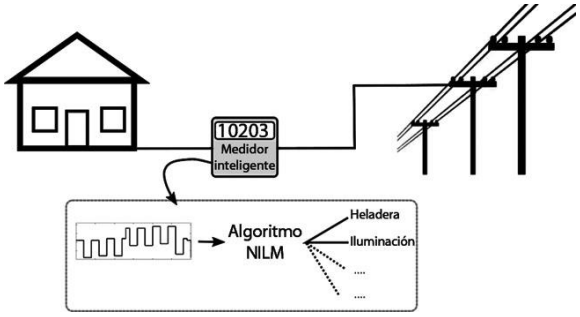

Figura 1.1 Esquema simplificado del uso de técnicas NILM a través de contadores inteligentes [1]

Dado lo costoso que puede llegar a resultar experimentar con las técnicas NILM sobre un contador inteligente real conectado a una vivienda, en este proyecto se busca crear un entorno que simule dichos dispositivos utilizando bases de datos con los datos de consumo energético de diferentes fuentes y a diferentes frecuencias de muestreo. Una vez emulado el contador, también se lleva a cabo la adquisición de dichos datos para, en trabajos futuros, experimentar con la viabilidad de los SoCs, entorno sobre el que se desarrolla el proyecto, a la hora de aplicar técnicas NILM.

1.2 Objetivos

En este punto se expondrán los principales objetivos que se buscan cumplir en el desarrollo del proyecto. Dado que el sistema se puede dividir en dos partes diferenciadas, emulador y sistema de adquisición de datos, cada uno tiene sus objetivos:

- **Emulador:** el objetivo de esta parte del sistema es la de emular un contador inteligente a través de bases de datos de consumo energético. Dado que existen multitud de bases de datos con diferentes fuentes y sobre todo diferentes frecuencias de muestreo, la frecuencia de muestreo de los datos debe ser configurable y debe mantenerse estable en todo momento, por lo que el flujo de datos que se destina hacia el conversor digital-analógico debe mantenerse constante.
- **Sistema de adquisición de datos:** el sistema de adquisición de datos debe funcionar a la misma frecuencia que el emulador y en paralelo a éste; tanto el conversor digital-analógico del emulador como el analógico-digital del sistema de adquisición irán conectados entre sí y, cada vez que se envíe un dato al DAC para su conversión, dicho dato deberá ser convertido de nuevo por el ADC antes de que el primer conversor reciba un nuevo dato, pero respetando su tiempo establecimiento. Finalmente el sistema de adquisición también deberá enviar el paquete de datos convertido utilizado como prueba de concepto a un terminal a través del puerto serie, para poder comparar los datos tras pasar por el sistema completo con los originales de las bases de datos y estudiar la viabilidad del uso de la arquitectura SoC propuesta para la experimentación y ejecución de técnicas NILM.

1.3 Estructura del documento

En este punto se hará una descripción de la estructura que seguirá el documento y un breve resumen del contenido de cada capítulo.

- **Estado del arte:** en este capítulo se desarrollará con más profundidad el contexto que rodea al proyecto. Se explicarán con más detalle en qué consisten las técnicas NILM, las fases que llevan a cabo y sus diferentes clasificaciones, así como sus aplicaciones. Por último se explicará que son los dispositivos SoCs y las FPGAs, y porqué tienen potencial a la hora de utilizarlos en el área de las técnicas NILM.
- **Arquitectura propuesta y desarrollo:** en este punto del documento se explicará toda la arquitectura seguida a la hora de diseñar el sistema y todos los módulos que lo forman. La arquitectura se divide en dos partes: procesador y lógica configurable. En el procesador se cargan las bases de datos en forma de librerías .h ,obtenidas previamente desde la herramienta Matlab, desde donde se transfieren a la FPGA. Una vez en la FPGA, un controlador gestiona el envío y la recepción de datos a y desde los conversores de datos, para enviarlas de nuevo al procesador; desde aquí, el procesador envía los datos convertidos a un terminal con Matlab, donde se comparan los datos originales y los nuevos.
- **Resultados experimentales:** en este capítulo se mostrarán los resultados al ejecutar el sistema, tanto los finales en Matlab como los de cada uno de los módulos. Para ello se implementan en el proyecto varios analizadores lógicos a

diferentes puertos de la parte alojada en la FPGA de la tarjeta, tanto puertos propios del sistema como algunos añadidos únicamente para la fase de depuración, y así mostrar que se cumplen con los objetivos propuestos. Finalmente se mostrarán los recursos utilizados, en hardware, por cada uno de los módulos que componen toda la arquitectura y el consumo de potencia del sistema al completo.

- **Conclusiones y trabajos futuros:** en este último capítulo se analizarán los resultados del capítulo anterior y se expondrán las conclusiones pertinentes. También se plantearán las posibles mejoras que se pueden aplicar al sistema y las líneas de trabajo a seguir en futuros proyectos.

2. Estado del arte

En este capítulo se explicará el contexto sobre el que se desarrolla este trabajo. En primer lugar se explicará en qué consisten las técnicas NILM y su relación con los contadores inteligentes, así como las aplicaciones que éstos tienen. A continuación se hará una exposición sobre las bases de datos usadas para la experimentación con las técnicas NILM anteriormente explicadas y finalmente se describirán las arquitecturas SoC y el dispositivo Zynq, sobre el que se implementa este proyecto, y las ventajas que puede aportar su uso sobre las técnicas NILM.

2.1 Técnicas NILM, los contadores inteligentes y sus aplicaciones

Con el crecimiento demográfico y el aumento del consumo energético de las últimas décadas, surge la necesidad de una gestión más eficiente de la energía, tanto por motivos monetarios como medioambientales. En la figura 2.1 se muestra gráficamente el aumento de la población junto con el aumento del consumo energético en los últimos 200 años.

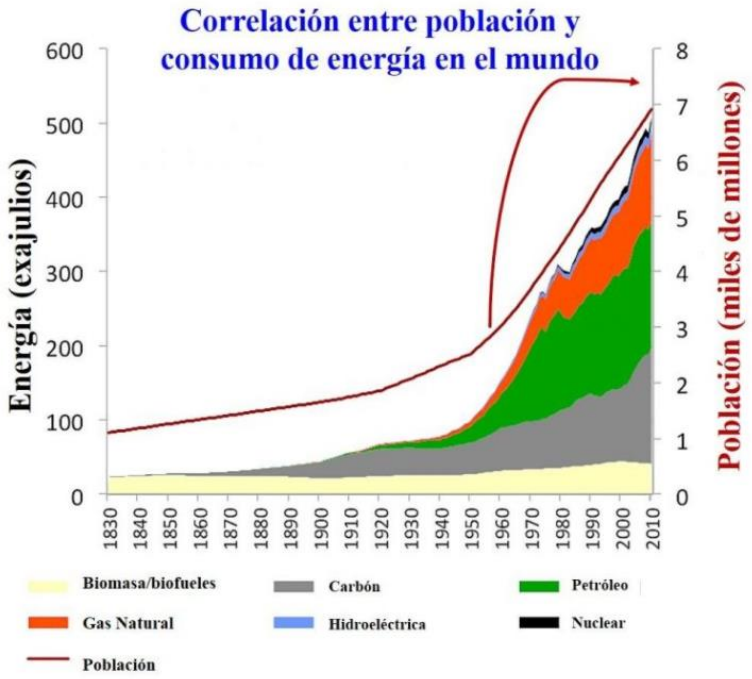

Figura 2.1 Crecimiento de la población mundial y su consumo energético [2]

Este hecho ha impulsado la búsqueda de nuevas fuentes de energía renovables frente a las fuentes clásicas de combustibles fósiles. Sin embargo, no solo es importante la búsqueda de nuevas fuentes, si no también un consumo más eficiente de la energía. En este punto es donde entran en juego las técnicas de disgregación de la energía.

Nonintrusive Appliance Load Monitoring, o NILM abreviado, son técnicas computacionales que a partir de la medida total del consumo energético de cualquier fuente, ya sea una vivienda, un edificio residencial o una fábrica industrial, son capaces de identificar los dispositivos eléctricos individuales que se encuentran consumiendo en un instante de tiempo determinado [3]; en otras palabras, son capaces de realizar una desagregación de la energía. Los primeros estudios sobre la desagregación de la energía datan de 1992, por George Hart. En dichos estudios Hart obtuvo la derivada de la señal y fue capaz de extraer las deltas, escalones producidos por la conexión y desconexión de cada dispositivo eléctrico [4]. En la figura 2.1 se muestra un ejemplo de la aplicación de las técnicas mencionadas sobre el gasto eléctrico de una vivienda.

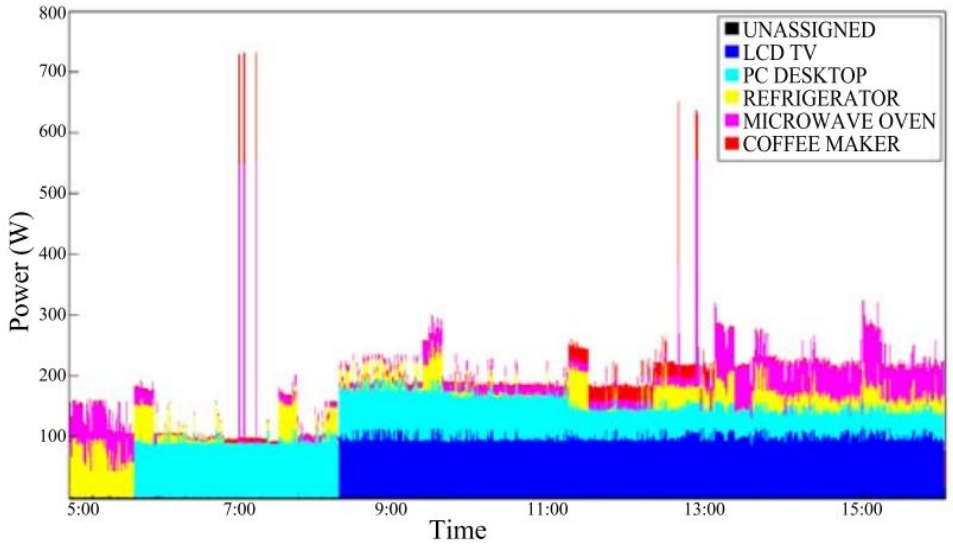

Figura 2.2 Ejemplo de aplicación de técnicas de disgregación de la energía [5]

Como ya se ha comentado, y en relación con la figura 2.1, el objetivo de la aplicación de estas técnicas es la de, dado una potencia medida total, dividir esa medida como la suma de la energía consumida por distintas fuentes; sobre esto, Hart sugiere que, dependiendo del perfil energético de cada dispositivo, se pueden clasificar en [5]:

- *Tipo1, Dispositivos On-Off:* dispositivos que se modelan con estados on/off. Ejemplos de estos dispositivos pueden ser una lámpara o un ordenador.

- *Tipo 2, Máquinas de estado finito*: dispositivos que pasan por diferentes estados de conmutación. Su ciclo de operación se puede representar como una máquina de estados que se repite diaria o incluso semanalmente. Un ejemplo de estos dispositivos es una lavadora.

- *Tipo 3, Dispositivos de consumo variable*: dispositivos caracterizados por tener un perfil de consumo no periódico. Un ejemplo de estos dispositivos son el proceso de carga de un teléfono móvil o una aspiradora.

- *Tipo 4, Dispositivos de consumo permanente:* dispositivos que están encendidos y en funcionamiento de manera continua o casi continua, como por ejemplo el frigorífico o el teléfono fijo.

En la figura 2.3 se muestra una gráfica con ejemplos del patrón de consumo de los diferentes tipos de dispositivos.

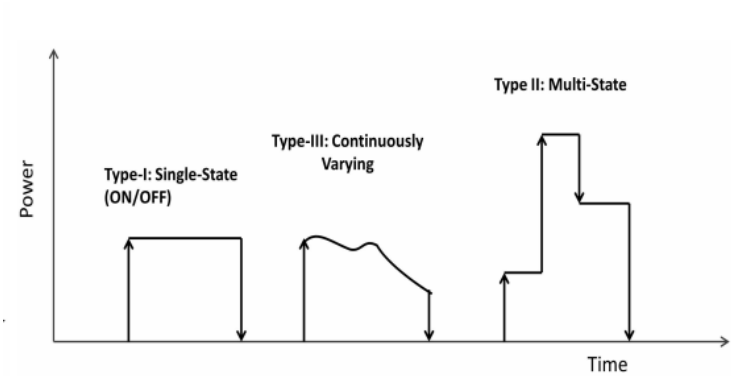

Figura 2.3 Patrones de consumo energético según el tipo de dispositivo [6]

Como ya se ha comentado, las técnicas NILM consisten en detectar que dispositivos están conectados en cada momento y cuál es su aportación al consumo total de energía. En la figura 2.4 es muestra un esquema con las diferentes fases por las que se pasa a la hora de aplicar técnicas NILM; dichas fases se describen a continuación [7]:

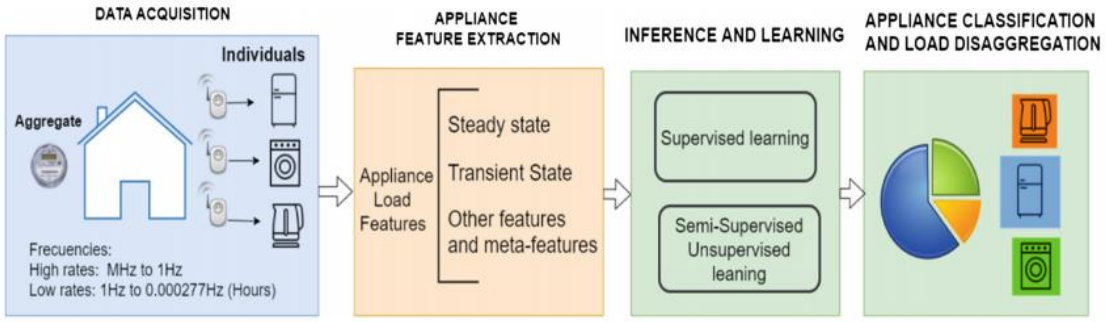

Figura 2.4 Esquema general de las diferentes fases a la hora de aplicar técnicas NILM [7]

- **Fase de adquisición de datos:** la primera fase a la hora de aplicar técnicas NILM es la de la adquisición de datos. La principal fuente de datos, y gracias a la cual el estudio de estas técnicas computacionales se ha impulsado en los últimos años, son los contadores inteligentes. Los contadores inteligentes o

Smart Meters son dispositivos electrónicos cuya función es la de registrar los datos reales de consumo energético de la vivienda o edificio en el que estén en funcionamiento, permitiendo medir el consumo en cualquier momento del día y de forma remota. Las medidas básicas que provee el contador inteligente son voltaje, medido en Voltios, corriente, medida en Amperios, y potencia aparente, medida en Voltios-Amperios, la cual es producto de corriente por voltaje.
Dentro de la adquisición de datos, estos se clasifican en dos tipos, los de alta frecuencia de muestro y baja frecuencia de muestreo.

- *Altas frecuencias de muestreo*: los datos se recogen a frecuencias de muestro desde 1kHz hasta 500kHz [4]. Aunque a estas frecuencias es más sencilla la clasificación de los distintos dispositivos, gracias a la presencia de estados estacionarios que no pueden observarse a frecuencias más bajas, tienen la desventaja de que requieren de un hardware más sofisticado y caro, haciendo más complicado su entrada en el hogar medio; además en ocasiones no es suficiente con único contador inteligente, sino que se requieren sensores específicos para los diferentes dispositivos, haciendo el proceso más intrusivo.
- *Bajas frecuencias de muestro*: los datos se recogen a frecuencias de 1Hz hacia abajo. Aunque la clasificación es más complicada de llevar a cabo, la principal ventaja de trabajar a estas frecuencias es la de poder implementarlo a través de contadores inteligentes.

- **Fase de extracción de características:** una vez adquiridos los datos, el siguiente paso es extraer información de ellos. Al igual que la fase anterior, aquí se distinguen dos vertientes, por detección de eventos o por algoritmos de predicción de estados (o sin detección de eventos).

  - *Algoritmos de predicción de estados*: estos algoritmos tratan de predecir el estado de un conjunto de dispositivos en cada muestreo de los datos de consumo recogidos [8]. Si dicho conjunto es muy grande, el número de posibles combinaciones aumenta también, por lo que el proceso se vuelve más complejo. Dado el problema que tienen los datos muestreados con bajas frecuencias debido a la ausencia de los transitorios, los algoritmos de predicción de estados están ligados a estos.
  - *Algoritmos de predicción de eventos*: estos algoritmos tratan de detectar los estados transitorios de los diferentes dispositivos conectados. Aunque también es posible utilizar estos algoritmos para los estados estacionarios (lo que los haría también adecuados para frecuencias de muestreo bajas), es mucho más sencillo separa los diferentes dispositivos por sus estados transitorios, por lo que estos algoritmos están ligados a las altas frecuencias de muestreo.

- **Inferencia y aprendizaje:** una vez extraídas las características del consumo total de la energía, en esta fase se trata de determinar cuáles son los dispositivos que se encuentran consumiendo en cada momento. Al igual que las fases

anteriores, en esta también se hacen diferentes clasificaciones: métodos supervisados y semi-supervisados o no supervisados.

- *Métodos supervisados:* aunque los métodos supervisados pueden dividirse en diferentes categorías, el funcionamiento general es el mismo: partiendo de bases de datos preestablecidas, estos métodos tratan de comparar las características extraídas en las fases anteriores del proceso con las presentes en las bases de datos y de esta manera encontrar las combinaciones más similares. De esta manera, estos algoritmos son capaces de encontrar el dispositivo que ha generado una determinada salida de consumo.
- *Métodos semi-supervisados o no supervisados:* aunque de momento no dan resultados tan precisos como los métodos supervisados, el interés en estos métodos radica en que apenas necesitan información previa para ser aplicados y el hardware necesario es menos costoso e intrusivo, así como una fase mucho más corta de aprendizaje, características importantes a la hora de llevar las técnicas NILM al hogar medio.

- **Desagregación del consumo y clasificación:** una vez extraídas las características del consumo energético y clasificado los diferentes dispositivos que participan en dicho consumo en cada momento del muestreo, en esta última fase se identifica y desglosa cada uno del balance total. Esto permite al usuario tener mayor conocimiento de su perfil de consumo y por lo tanto más control sobre él.

Una vez descritas las fases, se puede hablar de las aplicaciones y ventajas que tiene el uso de estas técnicas de la desagregación de la energía. Como ya se ha comentado, el alto consumo energético empieza a convertirse en un problema; mediante la aplicación de las técnicas NILM, el usuario puede obtener una información mucho más detallada de su consumo, obteniendo qué dispositivos son los que más consumen y el uso que hace de ellos. Esto puede dar lugar a ejercer un consumo más responsable y eficiente. Sin embargo, este no es el único uso que puede dársele a estas técnicas.

Como se muestra en la figura 2.5, aunque solo haga referencia a los últimos 20 años ya se puede observar un envejecimiento paulatino de la población; este hecho es más acusado cuanto más se retrocede en el tiempo, debido tanto al aumento de la calidad de vida y los avances médicos en países desarrollados, junto con el descenso de la natalidad infantil,

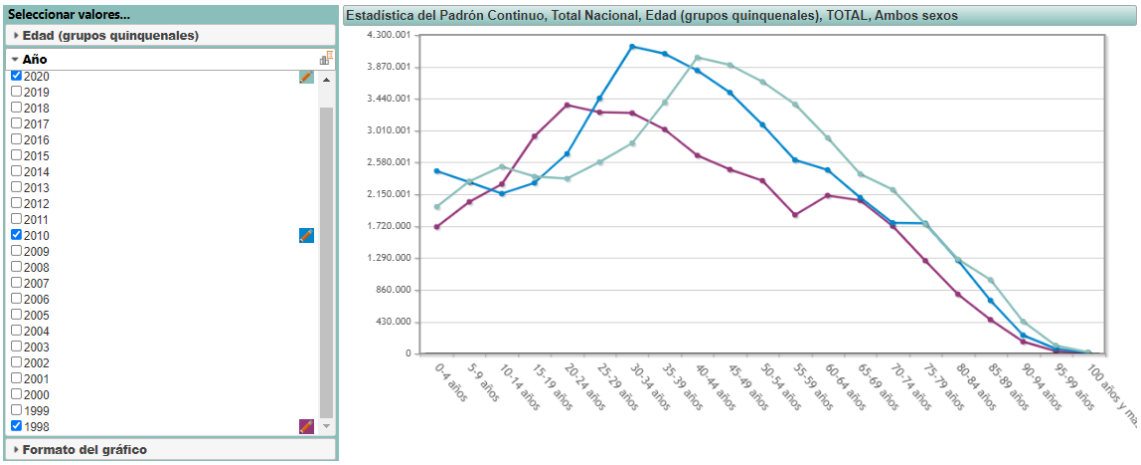

Figura 2.5 Censo de población total española por franjas de edad. Datos obtenidos del Instituto Nacional de Estadística

El potencial del uso de técnicas NILM en esta área radica en aumentar la autonomía y cuidado de las personas mayores, que buscan ser lo más independientes posibles, mediante la monitorización de sus actividades. A través del uso de los algoritmos y los sensores inteligentes, es posible encontrar los patrones de comportamiento del individuo en su vivienda a través del empleo que le da a sus dispositivos electrónicos, permitiendo así encontrar anomalías en su rutina, así como estimar su localización directa.

Para poder estudiar el desempeño de las técnicas NILM sin la necesidad de obtener datos en tiempo real, un proceso costoso, existen bases de datos a disposición de los investigadores. Dichas bases de datos varían en frecuencia de muestreo, fuente de los datos (desde viviendas individuales, usos comerciales o industria) y duración, habiendo opciones para todo tipo de estudios. En el caso de este trabajo, se ha utilizado la base de datos BLUED [9], con una frecuencia de muestreo de 12kHz y una duración de una semana, datos obtenidos de un edificio residencial. La primera parte del proyecto consiste en, utilizando esta base de datos, emular el comportamiento de un contador inteligente; la segunda parte del proyecto se centra en el proceso de la adquisición de dichos datos.

En el siguiente punto se hará una descripción de los llamados System on Chip, SoC abreviados, así como las ventajas que pueden aportar a la hora de aplicar técnicas NILM.

2.3 Arquitecturas SoC, FPGAs y dispositivos Zynq

Un System on Chip (SoC) es esencialmente un circuito integrado que unifica, en un solo chip, una unidad central de procesamiento o CPU y un conjunto de IPs y periféricos, así como los buses con sus respectivos protocolos para permitir la comunicación entre las diferentes partes que lo componen. Un System on Chip puede estar diseñado para una aplicación específica, lo que supone una reducción de costes y una mayor eficiencia que utilizar otro sistema, ya sea SoC o no. En la figura 2.6 se muestra un ejemplo de una arquitectura SoC, concretamente la de la tarjeta Zybo z7-20. Esta plataforma, de Digilent Inc., es la que se utilizará para el desarrollo de este trabajo y se explicará con más detalle más adelante.

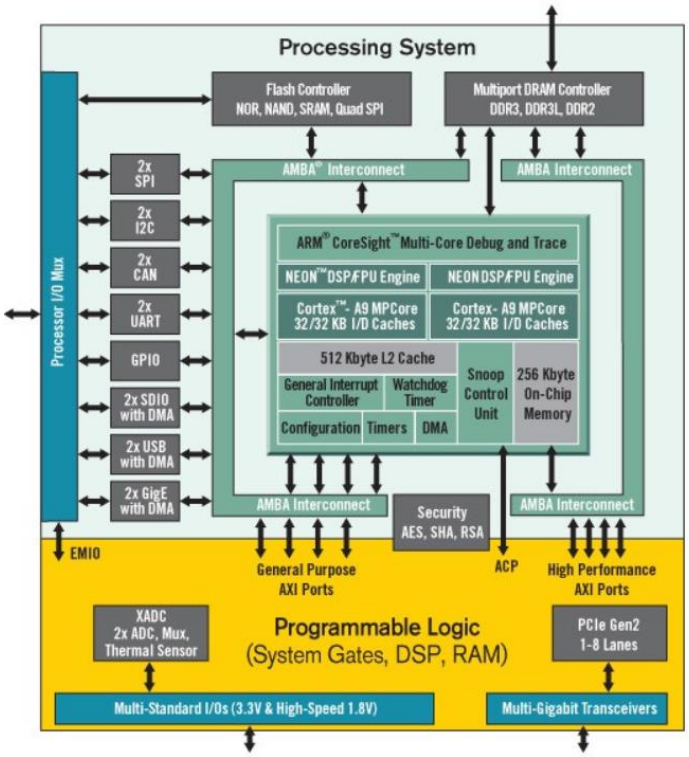

Figura 2.6 Arquitectura de un ejemplo de SoC, concretamente un dispositivo Zynq-7000 [10]

Aunque la arquitectura de un System on Chip puede diferir dependiendo de si está diseñado para aplicaciones específicas o sigue una arquitectura más estándar, las partes que lo componen suelen coincidir con las que explicarán a continuación [11]:

- **CPU:** la unidad central de procesamiento es la parte más importante del circuito integrado; forma el core del SoC y se encarga de controlar el resto de los periféricos de la arquitectura y de llevar a cabo las principales tareas de procesamiento.

- **Recursos del sistema:** este bloque cuenta con los osciladores para la generación de los relojes, las fuentes de alimentación y el control de sus modos de trabajo y el Watchdog Timer, entre otros. Este último se trata de un reloj cuya

finalidad es la de, dado un tiempo de Timeout, detectar y recuperarse de un mal funcionamiento del proyecto en ejecución.

- **Memorias:** el sistema debe contar con memorias, tanto memorias Flash para almacenar el código implementado como memorias RAM para datos y ROM para arrancar el sistema. Puede implementar según sea necesario otros tipos de memorias.

- **Periféricos digitales:** los SoC suelen contar con diferentes periféricos digitales, desde contadores y generadores PWM hasta conversores digital-analógicos y Procesadores Digitales de Señales o DSP, según sus siglas en inglés. En los últimos años han aparecido SoCs que implementan una FPGA dentro de la arquitectura, por lo que los recurses digitales suelen aunarse en esta parte del sistema; estos SoC son los llamados Zynq y se entrará en detalle en ellos más adelante.

- **Periféricos analógicos:** de igual manera que el SoC cuenta con periféricos digitales, también lo hace con periféricos analógicos. Estos pueden ir desde conversores analógico-digitales hasta fuentes de voltaje y corriente y otros componentes electrónicos analógicos como comparadores o amplificadores.

- **Periféricos de comunicación:** aparte de los protocolos que implemente internamente el SoC para comunicar sus diferentes partes en caso necesario, también cuenta con periféricos de comunicación para enlazar con sistemas externos. Algunos ejemplos de protocolos de comunicación que se suelen implementar son I2C, SPI, UART o USB para comunicaciones cableadas y Wifi o Bluetooth para comunicaciones inalámbricas.

- **Puertos I/O:** los SoC cuentan generalmente con bloques dedicados a puertos de entrada/salida para propósito general o GPIO, y en algunos casos para propósitos específicos como drivers para pantallas LCD y LEDs.

- **Direct Memory Access Controller:** DMA abreviado, se trata de un periférico que permite la transferencia directa de datos entre los periféricos y los bloques de memoria, liberando a la CPU de llevar a cabo dicha tarea. Esta transferencia de datos puede ser entre periféricos y memoria, periférico y periférico e incluso entre memoria y memoria, y en cualquier dirección; además permite transferencias únicas o en bloque.

Aunque dependiendo de su objetivo los SoC pueden incluir bloques más específicos, los bloques más generales son los que se han expuesto. Como se ha comentado, en los últimos años se han empezado a desarrollar SoC que incluyen en su arquitectura FPGAs, llamándose así AP SoCs, All Programmable System on Chip. En la figura 2.7 se muestra un ejemplo de la arquitectura de una FPGA de la serie 7 de Xilinx.

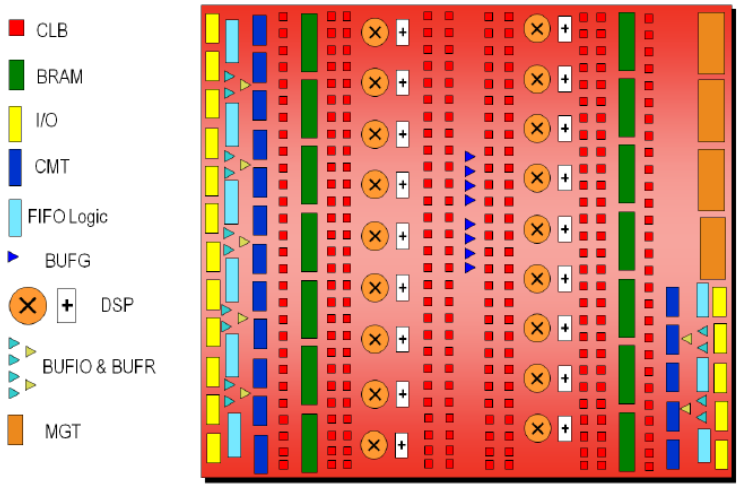

Figura 2.7. Arquitectura de una FPGA de la serie 7 de Xilinx

Las FPGAs o Field Programmable Gate Array son circuitos integrados basados en semiconductores cuya unidad más básica son los CLB. Los CLB o Configurable Logic Block son los elementos lógicos básicos que componen las FPGAs, permitiendo implementar cualquier funcionalidad lógica y ejecutar funciones de memoria. A su vez, están compuestos por elementos más pequeños:

- **Flip-flops:** los flip-flops o latches son los elementos de memorias más pequeños de las FPGAs. Capaces de representar dos estados de un único bit, funcionan como registros para guardar los estados lógicos de las diferentes señales entre ciclos de reloj.
- **LUTs:** las Look-Up Tables funcionan en los términos de electrónica digital como tablas de verdad. Dadas unas entradas, generan determinadas salidas dependiendo de la lógica configurada. Éstas permiten agilizar los procesos computacionales almacenando los posibles resultados y ahorrando realizar cálculos.
- **Multiplexores:** la última pieza básica de los CLBs son los multiplexores, elementos digitales que, dadas varias entradas y señales de control, selecciona una única señal de entrada como salida.

En la serie 7 de FPGAs de Xilinx las LUTs también se implementan como elementos de memorias RAM, llamadas LUTRAM. Otro elemento que se utilizará en el sistema diseñado que se explicará en el siguiente capítulo es la BRAM; se trata de memoria RAM de doble puerto instanciada dentro de la FPGA que se utiliza para almacenar cantidades relativamente grandes de datos.

Una de las mayores ventajas de las FPGAs es su capacidad para ser reprogramadas continuamente, lo que da mucha más flexibilidad frente a los ASICs (Circuito Integrado de Aplicación Específica), diseñados para realizar tareas específicas, y una gran eficiencia energética. Otra de las características principales de las FPGAs es la capacidad de trabajar en paralelo y a nivel de bit; a pesar de que trabajan a relojes de frecuencias mucho más bajas que las CPUs o las GPUs, en el rango de los MegaHerzios frente a los GigaHerzios de éstos, por su arquitectura y funcionamiento en paralelo son capaces de ejecutar operaciones en unos pocos ciclos de reloj, frente a las docenas de

instrucciones que necesitan los procesadores, suponiendo una mayor eficiencia de consumo.

Debido a todas estas características, las FPGAs son muy potentes a la hora de trabajar en el tratamiento digital de señales y en el ámbito de la Inteligencia Artificial y el Deep Learning [12], empezando a usarse incluso frente a las GPUs, que ocupaban este mercado. Es por esto que las FPGAs han despertado el interés en el mundo de las técnicas NILM, cuyos algoritmos muchas veces se basan en el Deep Learning y las redes neuronales.

Aunando tanto la tecnología de los SoCs como la de las FPGAs, Xilinx creó una familia de dispositivos llamada Zynq-7000, también referidos como AP SoCs (All Programmable System on Chip). Estos dispositivos cuentan con las ventajas computacionales a nivel del software que ofrece una CPU, junto con la programabilidad hardware que ofrece una FPGA, lo que los hace al menos muy interesantes para experimentar con técnicas NILM.

Este trabajo se desarrolla sobre una tarjeta de la familia Zynq-7000 descrita, concretamente la Zybo z7-20. Además de la tarjeta, se utilizarán dos Pmod, un controlador digital-analógico y otro analógico-digital, dispositivos desarrollados por Digilent con una interfaz para trabajar tanto con FPGAs como con microcontroladores. Se utilizará las base de datos BLUED comentada anteriormente cargándola como librería sobre el procesador del AP-SoC, el cual gestionarán el emulador y se encargará de transferir los datos a la FPGA para que este controle el funcionamiento de los conversores. En el siguiente capítulo se describirá en detalle la arquitectura diseñada para el emulador.

3. Arquitectura propuesta y desarrollo

En este capítulo se va a describir la arquitectura propuesta para el emulador, profundizando en su diseño e implementación. El emulador se divide en dos bloques principales: la parte hardware, implementada en lenguaje VHDL en la Programmable Logic o PL, que se encargará principalmente de la comunicación con los conversores utilizados en el proyecto y la gestión de la velocidad de estos; la parte software, implementada en lenguaje C en la Processing Subsystem o PS, cuya función es la de transmitir las bases de datos al controlador implementado en la PL y su recepción posteriormente , tras haber pasado por ambos conversores, así como transmitirlas a Matlab para comparar los resultados con los datos originales. Para la comunicación entre ambas partes se hace uso del protocolo AXI4-Lite, implementado a través de IPs prefabricadas provistas por Xilinx, en las cuales se instancian el resto de los módulos VHDL. En la figura 3.1 se muestra un diagrama con la distribución del sistema completo y la conexión entre las partes que con lo componen.

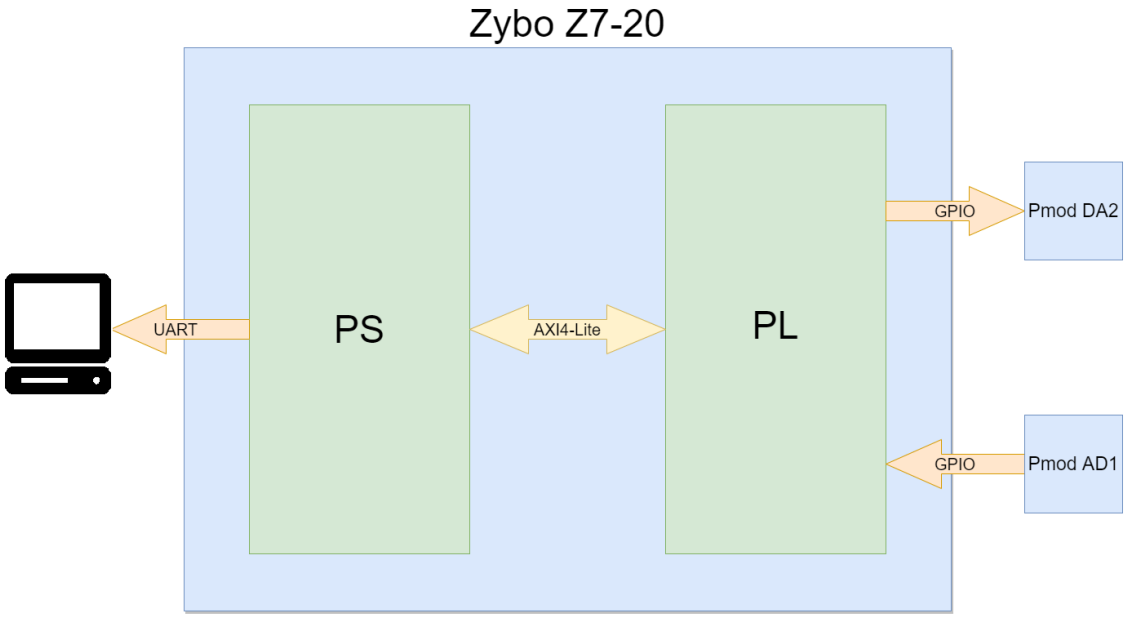

Figura 3.1 Diagrama general del sistema completo

En el siguiente punto de este capítulo se describirá el funcionamiento general del sistema y el cargo que desempeñan cada una de las partes que lo componen; posteriormente se describirá la arquitectura interna de dichas partes y como interaccionan entre sí.

3.1 Objetivos y funcionalidad

El objetivo del proyecto es, en primer lugar, ser capaz de emular el consumo eléctrico de una vivienda o edificio mediante el uso de bases de datos y un conversor digital-analógico; la segunda mitad del proyecto tiene la finalidad de emular un sistema de adquisición de datos para ejecutar posteriormente técnicas NILM, conectando entre sí ambos conversores. En la figura 3.2 se muestra un esquema con el flujo de acción del emulador.

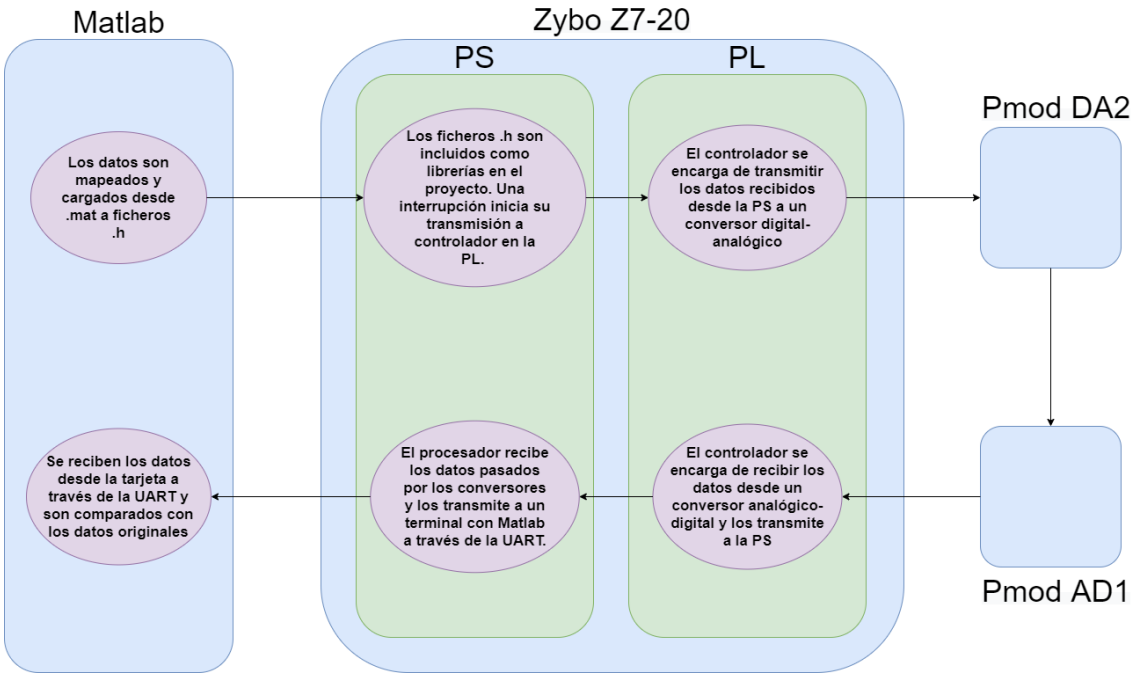

Figura 3.2 Flujo de funcionamiento del sistema

Las técnicas NILM se aplican sobre datos con frecuencias relativamente bajas, de entre 1 y 12kHz, por lo que el sistema debe asegurar que las muestras mantienen dicha frecuencia a lo largo de todo el proceso, tanto al emular el consumo eléctrico como al realizar la adquisición de dichos datos. Dadas las diferencias de frecuencia con las que trabajan el procesador, los conversores y finalmente las propias bases de datos, se implementan memorias FIFO dentro de la FPGA para guardar los datos mientras son enviados y recogidos a y desde los conversores. A continuación se procederá a describir con más detalle las diferentes fases mostradas en la figura 3.2:

- Las bases de datos que se utilizan en el emulador se encuentran en ficheros de formato .mat. Dado que dicho formato no puede ser utilizado directamente por el procesador de la tarjeta, se hace uso de la herramienta Matlab para convertir dichos datos a ficheros .h. Además, al convertirlos a dichos ficheros también se realiza el mapeado de los datos a un dominio de 12 bits para trabajar en el rango de los conversores.
- Los ficheros .h creados en el punto anterior se cargan en el proyecto implementado sobre el procesador de la tarjeta como librerías. Dicho proyecto

funciona en un bucle con espera activa a base de interrupciones. Cuando la interrupción correspondiente se activa, el procesador comienza a enviar los datos a las memorias de la parte lógica del emulador, asegurándose de que dichas memorias siempre tengan datos para enviar a los conversores y se cumpla la frecuencia configurada para la base de datos.

- Una vez que las memorias comienzan a cargarse con las bases de datos, el controlador implementado en la PL gestiona el envío de éstos hacia el conversor analógico digital a la frecuencia configurada desde el procesador. El controlador se mantendrá activo siempre y cuando haya al menos un dato en las memorias FIFO de entrada.
- Una vez que cada dato pasa por ambos conversores, el controlador lo recoge de nuevo y llena las memorias FIFO de salida, trabajando a la misma frecuencia y en paralelo con el envío de los datos hacia fuera. Una vez que las memorias FIFO de salida se llenan, se produce una interrupción en el procesador para que los datos sean recogidos. Dada la diferencia de frecuencia entre el procesador y las bases de datos, no se produce ninguna pérdida por rebosamiento de las memorias.
- Una vez que se produce dicha interrupción, el procesador recoge todos los datos contenidos en las memorias FIFO de salida y las envía a un terminal con Matlab a través de la UART.
- Una vez que Matlab recibe los datos, se deshace el mapeado de éstos que se hizo inicialmente para adaptarlos a los conversores; con los datos originales y los convertidos en el mismo formato, se muestran gráficamente para poder analizar los resultados y la viabilidad del emulador.

En los siguientes puntos se explicará en detalle cada una de las partes que componen el sistema.

3.2 Bases de datos y herramienta Matlab

La base de datos utilizada en el desarrollo del emulador se corresponde con la BLUED, Building-Level fUlly labeled Electricity Disaggregation Dataset. La base de datos contiene datos de alta frecuencia (12kHz) de corriente y voltaje de una vivienda, además de un listado de eventos marcados en tiempo. Sin embargo para el desarrollo de este proyecto solo se utilizarán los datos correspondientes a dos listados de datos de corriente a modo demostrativo.

Para el desarrollo del emulador, se trabaja con dos scripts de Matlab: Genera_base_datos_h.m, que trabaja directamente con las bases de datos originales en formato .mat, y Conexión_Uart_PS.m, que se comunica con la tarjeta a través de UART y recibe los datos tras la fase de adquisición de datos.

Como ya se ha comentado, dichos listados de datos se encuentran en forma de columnas en dos ficheros .mat, *BLUED_Cu_PhA_011.mat* y *BLUED_Cu_PhA_012.mat*; para poder hacer uso de los datos contenidos en estos archivos .mat se transfieren a través del primer script a ficheros .h, que podrán incluirse como librerías dentro del proyecto desarrollado en la parte del procesador.

Sin embargo estos datos no se pueden utilizar directamente para ser enviados a los conversores. Dado que el consumo de corriente de una vivienda puede llegar a oscilar entre 5 y -5 Amperios, y los conversores trabajan con datos de 12 bits, se realiza un mapeado con los valores de corriente antes de transferirlos a los ficheros .h. En la figura 3.3 se muestran las funciones utilizadas para el mapeado de dichos valores.

```
fichero = 'base_datos_1_convertido.h';
fid = fopen (fichero, 'w');
fprintf(fid,'uint16_t muestras_base_datos_conv_1 []={');
for i=1:L_1
    x = uint16(((y_1(i)+5)/10)*4095);
    if i==L_1
        fprintf(fid,'%d', x);
    else
        fprintf(fid,'%d,', x);
    end
end
fprintf(fid,'}; \n\r');
fclose(fid);
```

Figura 3.3 Función de mapeado de datos y transferencia a ficheros .h

Aunque los conversores trabajan con tramas de 12 bits, los datos se guardan en 16 bits para ajustarlos a los tamaños del compilador del procesador de la tarjeta. Una vez que los datos son transferidos al controlador situado en la FPGA, estos son reajustados de tamaño, y dado que se realizó el mapeado para tener valores máximos de 4095 (correspondientes a 5 Amperios de corriente), no se pierden bits en el proceso.

El segundo script de Matlab abre una comunicación vía UART con la tarjeta y recibe todos los datos una vez han pasado por los conversores. Una vez recibidos, se deshace el mapeado y se visualizan éstos y los originales para analizar los resultados del emulador.

3.3 Protocolo AXI e interconexión PS-PL

Para la interconexión y comunicación entre hardware y software del emulador se hace uso del protocolo AXI. Se trata de un protocolo de comunicación que forma parte de ARM AMBA, una familia de buses para controladores. La versión de la que se hace uso es la AXI4, concretamente AXI4-Lite.

AXI4-Lite es un subtipo de AXI4 desarrollado para comunicaciones simples con baja demanda de rendimiento: a diferencia de AXI4 Stream y AXI4 Memory Map, no permite transmisión en modo burst y el ancho del canal está limitado a 32 o 64 bits, 32 en el caso de Xilinx IPs, como es el caso del emulador. Cuenta con una interfaz sencilla tipo maestro-esclavo; en este proyecto, el maestro se corresponderá con el procesador de la PS y el esclavo con la IP implementada dentro de la PL. El protocolo cuenta únicamente con cinco canales: Read Address Channel y Read Data Channel para operaciones de lectura, y Write Address Channel, Write Data Channel y Write Response Channel para operaciones de escritura. En la figura 3.4 se muestra la arquitectura para las operaciones de lectura y escritura.

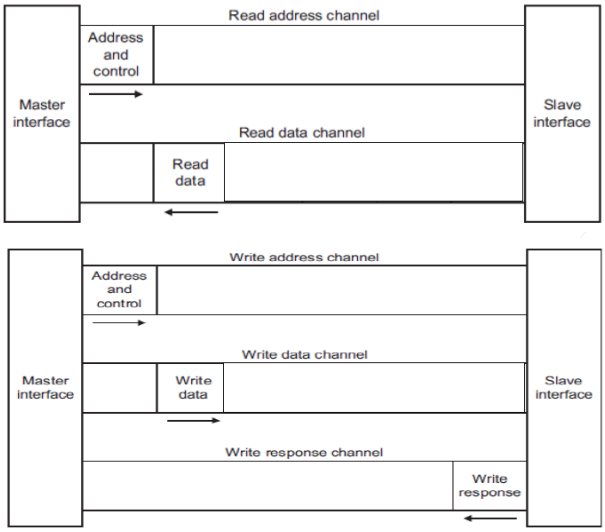

Figura 3.4 Arquitectura para operaciones de lectura y escritura de protocolo AXI4-Lite

Para hacer uso de los buses AXI Xilinx provee de IPs prefabricadas con el protocolo ya implementado en código VHDL, para utilizar en la parte lógica de cualquier proyecto. En el caso del emulador, se ha añadido una nueva IP con el protocolo AXI implementado dentro de la cual se han instanciado todos los módulos diseñados que se explicarán en siguientes puntos. De esta manera, el sistema cuenta con una única IP desde la que se comunican procesador y FPGA. En el siguiente punto se entra más en profundidad en la parte que le corresponde al procesador.

3.4 Processing Subsystem y SDK

El procesador es el encargado de gestionar la carga y descarga de las bases de datos en las memorias situadas en la parte lógica de la tarjeta. Una vez descargadas, tras haber pasado por los conversores, transfiere los datos a Matlab a través de la UART para poder observar los resultados.

Para la comunicación entre procesador y parte lógica a través de los buses AXI, Vivado crea unos drivers en fichero .h; en dicho fichero vienen definidas las funciones para leer y escribir en los registros del módulo AXI, donde están instanciados todos los módulos VHDL del emulador. Dichas funciones pueden observarse en la figura 3.5.

```
#define SPI_PMODS_MEMORIAS_IP_mWriteReg(BaseAddress, RegOffset, Data) \
    Xil_Out32((BaseAddress) + (RegOffset), (u32)(Data))
#define SPI_PMODS_MEMORIAS_IP_mReadReg(BaseAddress, RegOffset) \
    Xil_In32((BaseAddress) + (RegOffset))
```

Figura 3.5 Funciones de escritura y lectura de los registros del módulo AXI

Estas funciones hacen uso de funciones propias de Xilinx, Xil_Out32 y Xil_In32, para escribir en registros dadas las direcciones de éstos. Tantos las direcciones de los registros como las IDs de todos los periféricos y drivers utilizados en cada proyecto están definidos en el fichero xparameters.h, añadido por defecto por el compilador al crear el proyecto.

La función principal del sistema es la función main. En esta función se inicializan las variables utilizadas en el proyecto y se realiza la configuración de la UART para comunicarse con el terminal que ejecuta Matlab; finalmente configura las tres interrupciones de las que consta el proyecto. Una vez realizadas las configuraciones iniciales, el sistema se mantiene en un bucle a la espera de la activación de las entradas de interrupción. El código de la función descrita se muestra en la figura 3.6.

```
int main()
{
  int status = XST_SUCCESS;

  cont_muestras = 0;
  write_en = 0;

  UartPsConfiguracion(XPAR_XUARTPS_0_DEVICE_ID);

  status = SetupInterruptSystem(&Intc);
  if (status != XST_SUCCESS){
    return XST_FAILURE;
  }

  XScuGic_Connect(&Intc, INTC_INTERRUPT_ID_READ, (Xil_ExceptionHandler) Read_FIFOs, 0);
  XScuGic_SetPriorityTriggerType(&Intc, INTC_INTERRUPT_ID_READ, 0x90, 0x3);
  XScuGic_Enable(&Intc, INTC_INTERRUPT_ID_READ);

  XScuGic_Connect(&Intc, INTC_INTERRUPT_ID_INIT_WRITE, (Xil_ExceptionHandler) Write_FIFOs_init, 0);
  XScuGic_SetPriorityTriggerType(&Intc, INTC_INTERRUPT_ID_INIT_WRITE, 0xA0, 0x3);
  XScuGic_Enable(&Intc, INTC_INTERRUPT_ID_INIT_WRITE);

  XScuGic_Connect(&Intc, INTC_INTERRUPT_ID_NEXT_WRITE, (Xil_ExceptionHandler) Write_FIFOs_next, 0);
  XScuGic_SetPriorityTriggerType(&Intc, INTC_INTERRUPT_ID_NEXT_WRITE, 0xA0, 0x3);
  XScuGic_Enable(&Intc, INTC_INTERRUPT_ID_NEXT_WRITE);

  while(1)
  {

  }
}
```

Figura 3.6 Función main del procesador

Las tres interrupciones de las que consta el proyecto tienen la finalidad de gestionar la comunicación entre PS y PL, ya sea de escritura o lectura de las bases de datos. Se configuran de manera que la más prioritaria sea la de lectura y así no se produzca pérdida de datos por rebasamiento de las memorias de salida. Las interrupciones se describen con más detalle a continuación:

- **Write_FIFOs_init**: la primera interrupción se activa cuando se pulsa el botón 1 de la tarjeta. La función de esta interrupción es la de configurar la frecuencia de transmisión de la base de datos que será gestionada por el módulo controlador en la PL y la de realizar el primer envío de datos de las bases de datos. Dado que AXI4-Lite no soporta el modo burst, se utiliza un *bucle for* para realizar 1024 envíos de datos de 32 bits, de forma que se llenan completamente las memorias FIFO de entrada. Como cada envío es de hasta 32 bits, y las bases de datos son de 16 bits, los datos de ambas se concatenan en un solo dato que será separado posteriormente en hardware. Dado que el primer envío se corresponde con la frecuencia de las bases de datos y con los valores que contienen, se utilizan diferentes registros de offset para facilitar su gestión por parte de la FPGA. En la figura 3.7 se muestra el código correspondiente a la rutina de atención a la interrupción descrita.

```
void Write_FIFOs_init(void)
{
    u32 aux;
    int i;

    data_base_frq = FPGA_FRQ/PMODS_FRQ;
    SPI_PMODS_MEMORIAS_IP_mWriteReg(XPAR_SPI_PMODS_MEMORIAS_IP_0_S_AXI_BASEADDR, 4, data_base_frq);

    for ( i=0 ; i<1024 ; i++ ){
      aux = (muestras_base_datos_conv_2[i] << 16) + muestras_base_datos_conv_1[i];
      SPI_PMODS_MEMORIAS_IP_mWriteReg(XPAR_SPI_PMODS_MEMORIAS_IP_0_S_AXI_BASEADDR, 0, aux);
    }

    cont_muestras = 2;
    write_en = 1;
}
```

Figura 3.7 Rutina de atención a la interrupción para primera transmisión de bases de datos, PS-PL

- **Write_FIFOs_next**: la segunda interrupción se activa cuando las memorias FIFO de entrada se vacían hasta la mitad, rellenándolas de nuevo de manera que nunca queden vacías y el controlador sea capaz de enviar todos los datos sin ningún tiempo de espera, manteniendo la integridad de la frecuencia de las bases de datos. Para ello se hace uso de un contador, de manera que los envíos de 512 tramas se realizarán hasta finalizar los 16k de datos. En la figura 3.8 se muestra el código correspondiente a rutina de atención a la dicha interrupción.

```
void Write_FIFOs_next (void)
{
    u32 aux;
    int i;

    if(cont_muestras < 32 && write_en == 1)
    {
    for ( i=0 ; i<512 ; i++ ){
      aux = (muestras_base_datos_conv_2[i + 512*cont_muestras] << 16) + muestras_base_datos_conv_1[i + 512*cont_muestras];
      SPI_PMODS_MEMORIAS_IP_mWriteReg(XPAR_SPI_PMODS_MEMORIAS_IP_0_S_AXI_BASEADDR, 0, aux);
    }

    cont_muestras++;
    }
    else
    {
        write_en = 0;
    }
}
```

Figura 3.8 Rutina de atención a la interrupción para sucesivas transmisiones de bases de datos, PS-PL

- **Read_FIFOs**: la última interrupción se activa cuando las memorias FIFO de salida se llenan. Una vez que la memoria se llena, la interrupción recoge todos los datos y los va enviando uno por uno a Matlab a través de la UART. Dado que los datos se envían en paquetes de 8 bits, se realiza en 6 envíos, añadiendo el carácter NULL al final de cada dato de cada base. Esta interrupción es la más prioritaria para asegurar que las memorias FIFO de salida comience a vaciarse una vez que se llenan, de manera que los conversores no envíen un nuevo dato

a estas y no se pierda. En la figura 3.9 se muestra el código correspondiente a la rutina de atención a interrupción descrita.

```
void Read_FIFOs(void *InstancePtr){

  u8 prueba_uart_dato1[3], prueba_uart_dato2[3];
  int SentCount_d1 = 0, SentCount_d2 = 0;
  int i;
  for(i=0 ; i<1024 ; i++){
    y = SPI_PMODS_MEMORIAS_IP_mReadReg(XPAR_SPI_PMODS_MEMORIAS_IP_0_S_AXI_BASEADDR, 0);
    datos1[i] = y;
    datos2[i] = (y >> 16);

    prueba_uart_dato1[0] = datos1[i];
    prueba_uart_dato1[1] = datos1[i]>>8;
    prueba_uart_dato1[2] = '\0';

    prueba_uart_dato2[0] = datos2[i];
    prueba_uart_dato2[1] = datos2[i]>>8;
    prueba_uart_dato2[2] = '\0';

    while (SentCount_d1 < (sizeof(prueba_uart_dato1) - 1))
    {
      SentCount_d1 += XUartPs_Send(&Uart_Ps, &prueba_uart_dato1[SentCount_d1], 1);
    }

    while (SentCount_d2 < (sizeof(prueba_uart_dato2) - 1))
    {
      SentCount_d2 += XUartPs_Send(&Uart_Ps, &prueba_uart_dato2[SentCount_d2], 1);
    }

    SentCount_d1 = 0;
    SentCount_d2 = 0;

    }
  }
```

Figura 3.9 Rutina de atención a la interrupción para recepción de las bases de datos una vez convertidas, PL-PS

En el siguiente y último punto de este capítulo se describe la arquitectura diseñada para la parte hardware del emulador y cada uno de los módulos que la componen.

3.5 Programmable Logic y Vivado

Como ya se ha comentado, la parte hardware del proyecto, alojada en la PL, tiene como función principal comunicarse con los conversores de datos, así como gestionar tanto la velocidad de transmisión de datos como la frecuencia de muestro de éstos.

Esta parte del proyecto se divide a su vez en pequeños módulos en los que se entrará en profundidad más adelante: las memorias que almacenan tanto los datos originales como los datos tras pasar por los conversores, el controlador que gestiona tanto una parte de las memorias como los drivers de los conversores, y los propios drivers.

La figura 3.10 se corresponde con un diagrama de la IP AXI_IP_Core, que implementa el protocolo AXI4-Lite y tiene instanciados todos los módulos nombrados. En cuanto a las memorias, dependiendo de si son de entrada o salida, el protocolo AXI gestiona un lado y el otro lado es gestionado por el controlador.

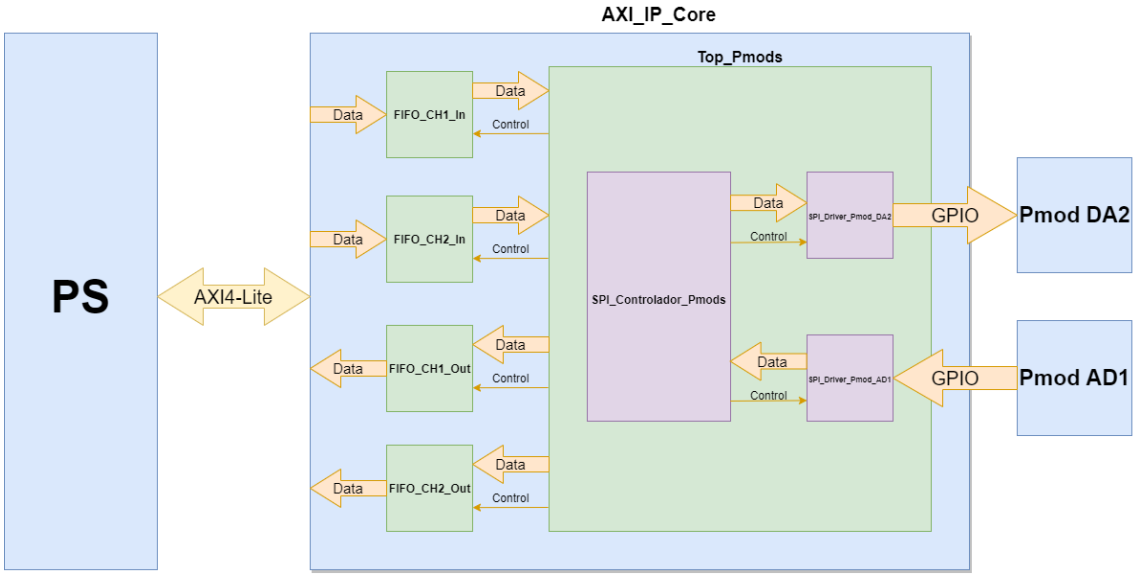

Figura 3.10 Diagrama de bloques de la parte hardware del sistema

3.5.1 Memorias FIFO

La primera parte perteneciente al hardware son las memorias FIFO. La función de las memorias es la de almacenar los datos de las bases de datos, tanto antes como después de pasar por los conversores: cada base tiene una memoria de entrada, para almacenar los datos originales, y una memoria de salida, para almacenar los datos convertidos antes de que la PS los recoja, haciendo un total de cuatro memorias. Dada la diferencia de frecuencia de trabajo entre la PL y la PS, el procesador se encarga de mantener las memorias de entrada siempre con datos para los drivers de manera que se mantenga la integridad de la frecuencia de trabajo de las bases de datos; a su vez, se encarga de vaciar las memorias de salida para asegurar que nunca se pierda ninguna trama. En las figuras 3.11 y 3.12 se muestran los diagramas temporales correspondientes a las operaciones de lectura y escritura de las memorias.

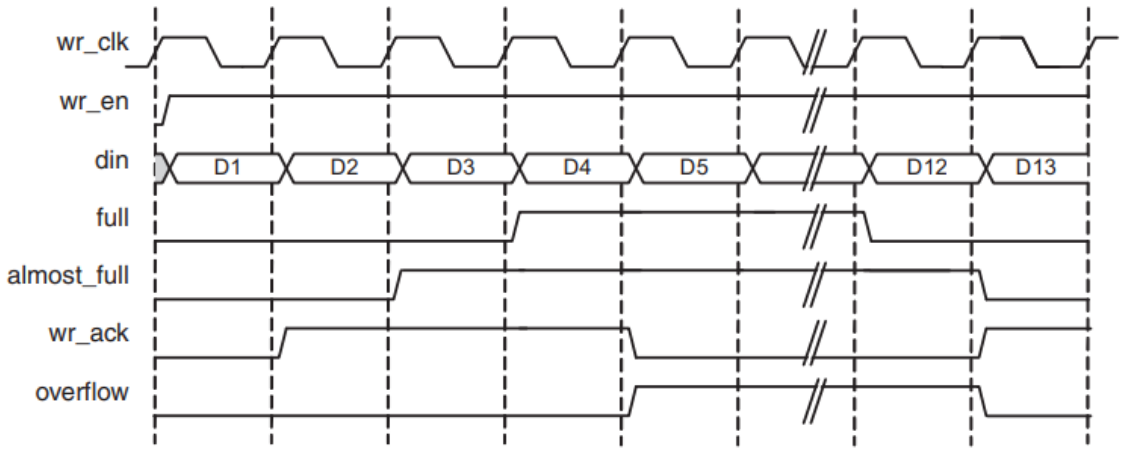

Figura 3.11 Diagrama temporal de memorias FIFO, operación de escritura [13]

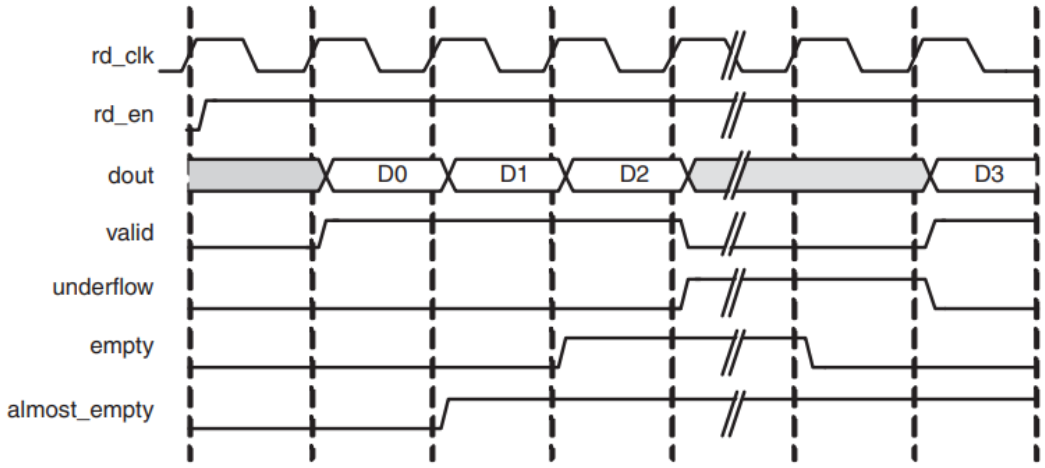

Figura 3.12 Diagrama temporal de memorias FIFO, operación de lectura [13]

Las memorias añadidas cuentan con 1024 datos de profundidad y 12 bits de ancho; dado que los datos que se reciben y se envían al procesador son de 16 bits, antes de guardar y después de sacar datos de las memorias, se ajustan los tamaños de cada trama. De los puertos de control disponibles para las memorias, solo se hace uso, aparte de las señales de Enable para operaciones de lectura y escritura, de las señales full, empty y prog_empty, programada para 512 bytes. En la tabla 3.1 se hace una descripción de todos los puertos de las memorias.

Tabla 3.1 Puertos de las memorias FIFO

Puerto	Tamaño	Dirección	Tipo de dato	Interfaz	Función
clk	1	In	Std_logic	Sistema	Reloj del sistema
srst	1	In	Std_logic	Sistema	Reset asíncrono activo a nivel alto
wr_en	1	In	Std_logic	Escritura	Señal de Enable para habilitar escritura en memoria
din	12	In	Std_logic_vector	Escritura	Entrada de datos a la memoria
full	1	Out	Std_logic	Escritura	Señal que se activa cuando la memoria se llena, activa a nivel alto
rd_en	1	In	Std_logic	Lectura	Señal de Enable para habilitar lectura de la memoria
dout	12	Out	Std_logic_vector	Lectura	Salida de datos de la memoria
empty	1	Out	Std_logic	Lectura	Señal que se activa cuando la memoria se vacía completamente, activa a nivel alto
prog_empty	1	Out	Std_logic	Control	Señal que se activa cuando la memoria se vacía hasta un valor configurable. Configurada para activarse cuando quedan 512 datos. Activa a nivel alto.

Dado que las memorias se dividen entre las de entrada y las de salida, la distribución de sus interfaces es diferente según el caso: en las de entrada, los puertos correspondientes a operaciones de escritura se conectan con el controlador descrito anteriormente, mientras que las de operaciones de lectura reciben sus datos a través del protocolo AXI directamente desde el procesador; en las memorias de salida, la

distribución es la contraria. En la figura 3.13 se muestra un diagrama con la distribución descrita.

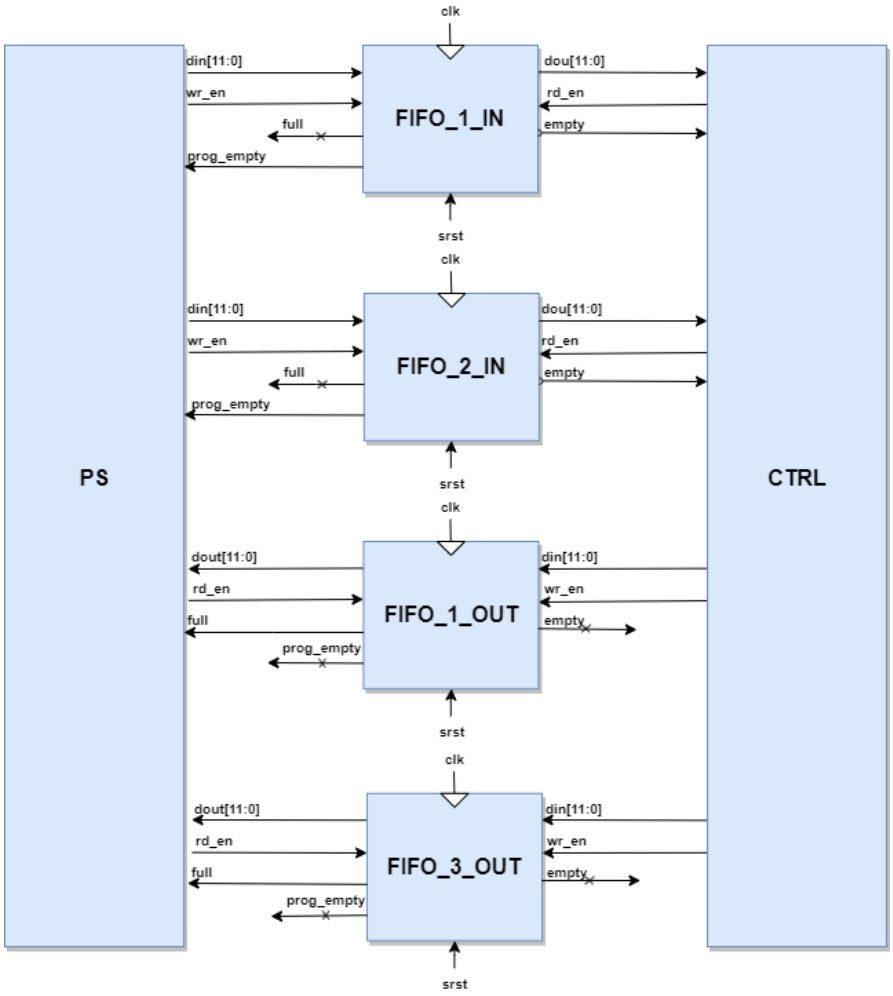

Figura 3.13 Diagrama de bloques con la distribución de las interfaces de las memorias FIFO

3.5.2 Módulo controlador de los drivers

El objetivo de este módulo es el de actuar como controlador general de toda la parte hardware del sistema. Por una parte controla los inicios de operación de ambos drivers con una frecuencia dada desde la parte software del sistema, recibida a través del protocolo AXI. Por otra controla una parte de las memorias FIFO, tanto de entrada como de salida; la otra parte de las memorias es gestionada desde el propio módulo AXI.

Interfaz

Como ya se ha comentado, la interfaz se divide principalmente en dos bloques: por un lado los drivers de los conversores y por otro las memorias FIFO.

Dentro de las memorias hay dos grupos, las de entrada y las de salida. Las de entrada son escritas por la parte software del sistema con las bases de datos, que se encarga de que nunca lleguen a vaciarse. En paralelo, el controlador se encarga de leer dichas memorias y mandarlas a los drivers de los conversores. Las FIFOs de salida son escritas por el controlador tras pasar por los conversores, mientras que la PS se encarga de leer dichos datos de manera que nunca lleguen a llenarse y no se pierda información. La figura 3.14 contiene un diagrama de bloques en el que se muestra la distribución de los puertos del módulo controlador.

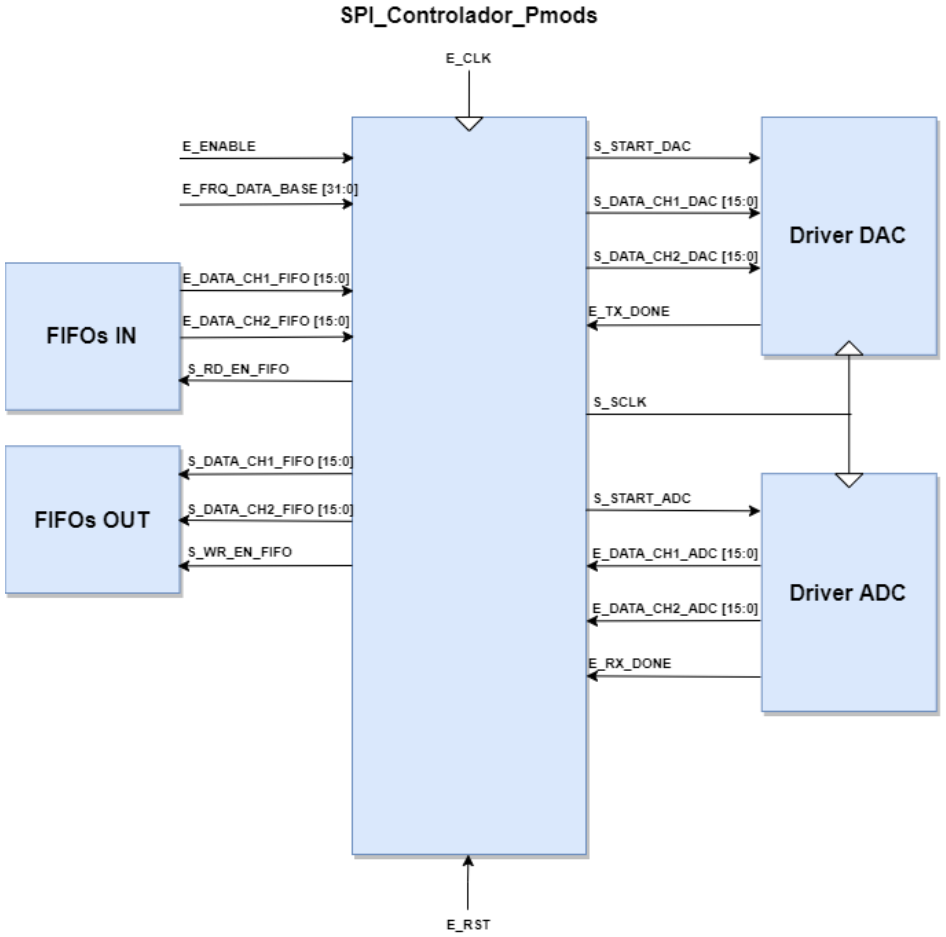

Figura 3.14 Diagrama del módulo controlador de los drivers y las FIFOs

La funciones y descripciones de todos los puertos del módulo controlador vienen definidos en la tabla 3.2.

Tabla 3.2 Puertos del módulo controlador de los drivers y las FIFOs

Puerto	Tamaño	Dirección	Tipo de dato	Interfaz	Función
E_CLK	1	In	Std_logic	Sistema	Reloj del sistema
E_RST	1	In	Std_logic	Sistema	Reset asíncrono activo a nivel alto
E_ENABLE	1	In	Std_logic	Sistema	Señal de Enable del controlador
E_FRQ_DATA_BASE	32	In	Std_logic_vector	Sistema	FRQ de trabajo de las bases de datos
E_DATA_CH1_FIFO	16	In	Std_logic_vector	FIFOs In	Datos recibidos desde FIFOs de entrada a transmitir al DAC, canal 1
E_DATA_CH2_FIFO	16	In	Std_logic_vector	FIFOs In	Datos recibidos desde FIFOs de entrada a transmitir al DAC, canal 2
S_RD_EN_FIFO	1	Out	Std_logic	FIFOs In	Señal de Read Enable para leer las FIFOs de entrada

S_DATA_CH1_FIFO	16	Out	Std_logic_vector	FIFOs Out	Datos enviados a FIFOs de salida, recibidas desde el ADC, canal 1
S_DATA_CH2_FIFO	16	Out	Std_logic_vector	FIFOs Out	Datos enviados a FIFOs de salida, recibidas desde el ADC, canal 2
S_WR_EN_FIFO	1	Out	Std_logic	FIFOs Out	Señal de Write Enable para escribir en las FIFOs de salida
S_SCLK	1	Out	Std_logic	Drivers	Reloj de 10MHz generado para los módulos drivers
S_START_DAC	1	Out	Std_logic	Driver DAC	Señal de inicio de transmisión para driver del DAC
S_DATA_CH1_DAC	16	Out	Std_logic_vector	Driver DAC	Datos enviados al driver del DAC para ser transmitidos al conversor, canal 1
S_DATA_CH2_DAC	16	Out	Std_logic_vector	Driver DAC	Datos enviados al driver del DAC para ser transmitidos al conversor, canal 2
E_TX_DONE	1	In	Std_logic	Driver DAC	Señal recibida por el driver del DAC para indicar final de operación
S_START_ADC	1	Out	Std_logic	Driver ADC	Señal de inicio de transmisión para driver del ADC
E_DATA_CH1_ADC	16	In	Std_logic_vector	Driver ADC	Datos recibidos por el driver del ADC tras ser convertidos por el conversor, canal 1
E_DATA_CH2_ADC	16	In	Std_logic_vector	Driver ADC	Datos recibidos por el driver del ADC tras ser convertidos por el conversor, canal 2
E_RX_DONE	1	In	Std_logic	Driver ADC	Señal recibida por el driver del ADC para indicar final de operación

Máquina de estados

La máquina de estados del módulo consta de siete estados: IDLE, RD_FIFO, TX, WAIT, RX, SAVE y WR_FIFO. La máquina se mantiene en reposo siempre y cuando las memorias FIFO de entrada estén vacías; si contienen al menos un dato, ésta se activa y manda dicho dato al conversor digital analógico. La máquina puede dividirse en dos fases: lectura de datos de las memorias de entrada para ser enviadas al conversor digital-analógico, y recepción del mismo dato desde el conversor analógico-digital y escritura de éste en las memorias de salida. En la figura 3.15 se muestra un esquema de la máquina de estados.

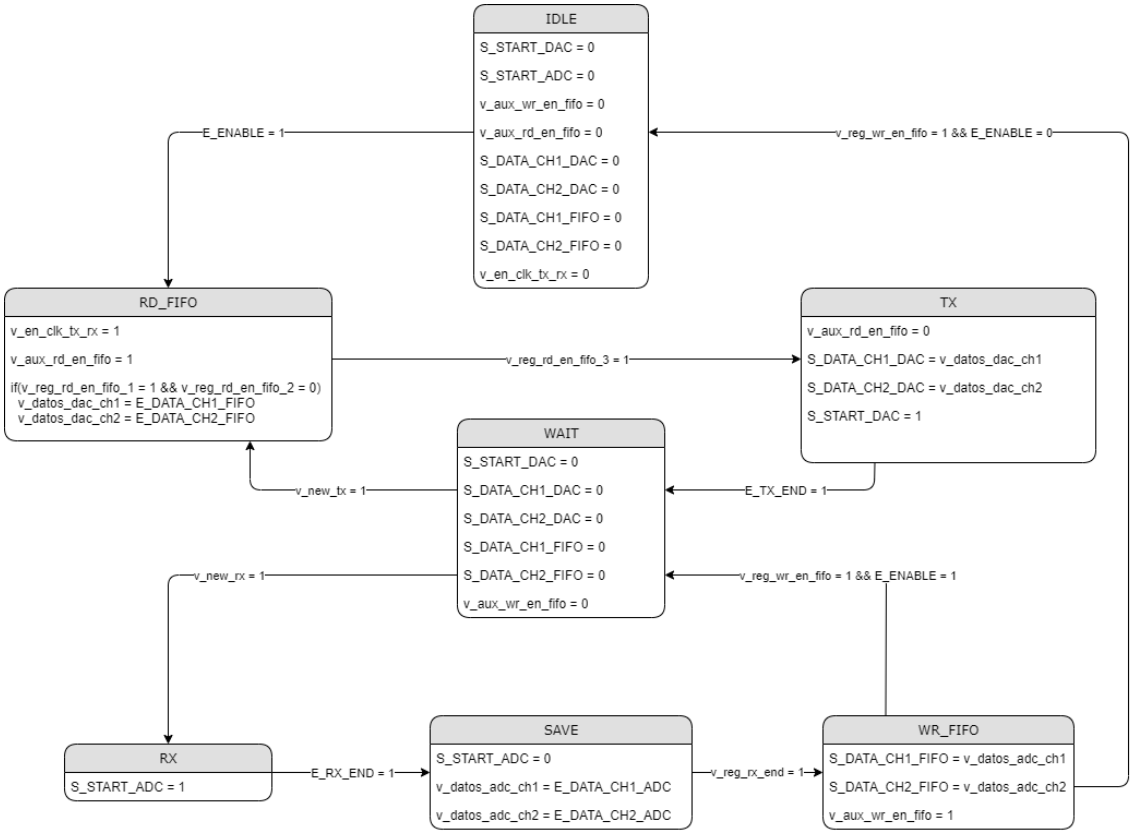

Figura 3.15 Máquina de estados del módulo controlador de los drivers y las FIFOs

A continuación se hará una breve descripción de cada uno de los estados de la máquina:

- **IDLE:** estado de reposo del sistema. Mantiene a los drivers de los conversores y a las memorias a la espera de órdenes. Pasa al siguiente estado, RD_FIFO, siempre que haya al menos un dato dentro de las memorias FIFO de entrada.
- **RD_FIFO:** lee un dato de cada memoria de entrada y lo registra. Una vez finalizada la transacción de lectura y el registro de los datos, pasa al estado TX.
- **TX:** estado en el que se transmiten los datos al driver del conversor digital-analógico, y se ordena el inicio de transacción de éste. Cuando se recibe la finalización de transacción del driver, se pasa al estado WAIT.
- **WAIT:** estado de espera de la máquina. Dado que las bases de datos trabajan a frecuencias de muestreo mucho menores que la FPGA y los conversores, entre 1 y 12kHz, es necesario un tiempo de espera entre los envíos. La frecuencia entre envíos y la frecuencia entre recepciones es la misma, sin embargo están desfasadas 180º entre ellas; de esta manera, siempre que se envíe un dato al conversor digital-analógico, se recibirá ese dato a través del conversor analógico-digital antes de que se produzca un nuevo envío. Por lo tanto, si la última transacción fue una recepción, el siguiente estado es de nuevo RD_FIFO; en caso contrario, el siguiente estado es RX.
- **RX:** estado de recepción de datos. Se inicia la transacción de datos del driver del conversor analógico-digital. Una vez que esa finaliza, se pasa al estado SAVE.
- **SAVE:** estado en el que se registran los datos recibidos del driver del ADC. Una vez guardados, se pasa al estado WR_FIFO.

- **WR_FIFO:** estado en el que se guardan los datos recibidos del ADC en las memorias FIFO de salida. Si las memorias de entrada quedan finalmente vacías, se vuelve al estado IDLE; de lo contrario se vuelve al estado WAIT, a la espera de un nuevo envío de datos.

Funcionamiento del controlador

Para mostrar el funcionamiento del controlador se han realizado simulaciones funcionales mediante la herramienta de Vivado Design Suit. A la hora de realizar dichas simulaciones, se han añadido, además del controlador, los módulos de los drivers de los conversores, de manera que se aprecia mejor el comportamiento descrito anteriormente.

Como se ha comentado en la descripción de la máquina de estados, ésta se puede dividir en dos fases. La primera fase de lectura de datos de las memorias de entrada y envío de estos al DAC, se muestra en la figura 3.16.

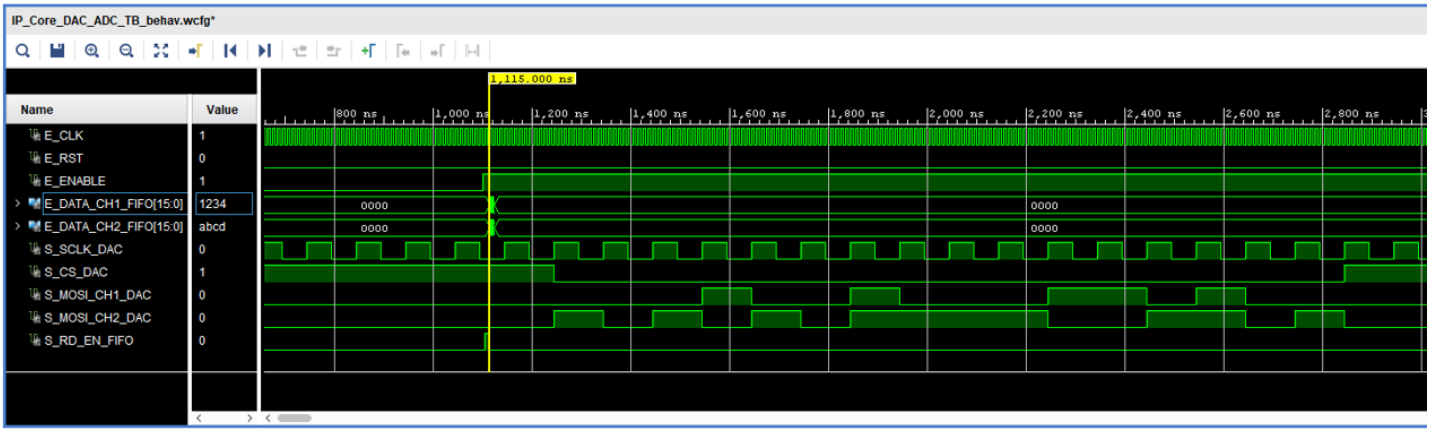

Figura 3.16 Simulación funcional correspondiente al controlador, proceso de lectura de las FIFOs de entrada y operación de escritura a los conversores digital-analógicos

Como se muestra en la figura, en primer lugar se leen los datos procedentes de las memorias de entrada; una vez leídos, en este caso “1234” y “abcd”, se inicia la transmisión de estos al conversor digital-analógico a través de su driver correspondiente.

La segunda fase de recepción de datos procedentes del ADC y su escritura en las memorias de salida, se muestra en la figura 3.17.

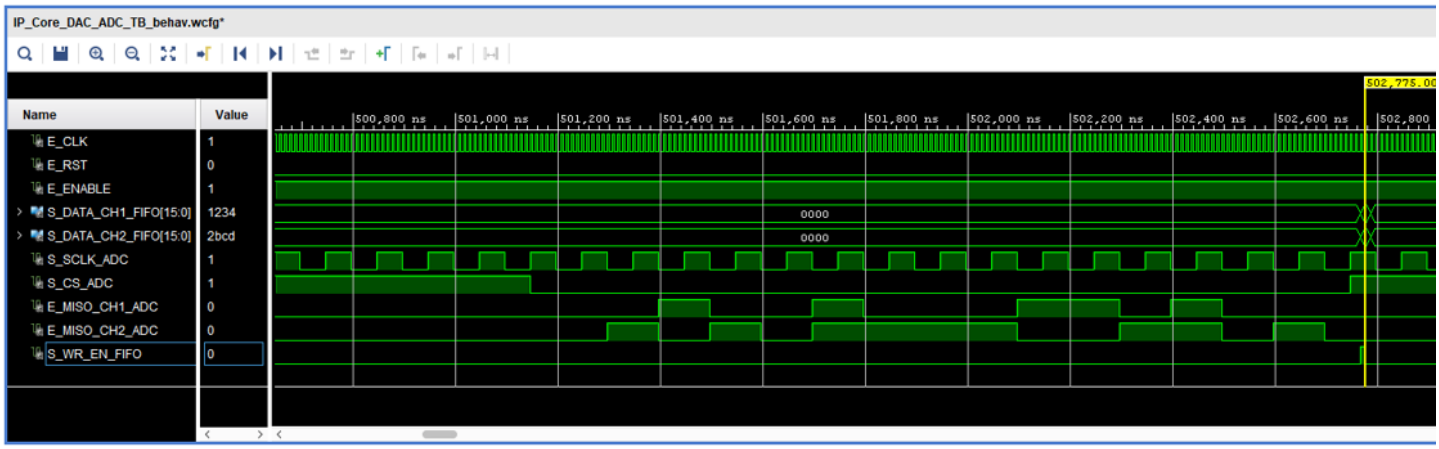

Figura 3.17 Simulación funcional correspondiente al controlador, proceso de escritura de las FIFOs de entrada y operación de lectura a los conversores analógico-digitales

En este caso, primero se produce la recepción de los datos procedentes del conversor analógico-digital para luego ser guardados en las memorias FIFO de salida. Dado que, como se mostrará más adelante en el diagrama temporal de este conversor, el primer bit se pierde, se recibe en una de las tramas “2bcd” en vez de su valor original. Sin embargo, dado que los datos que se enviarán son datos de 12 bits concatenados con cuatro “0”s, esto no será un problema.

La última figura de las simulaciones, la 3.18, muentra únicamente las señales de finalización de envío y recepcion de datos.

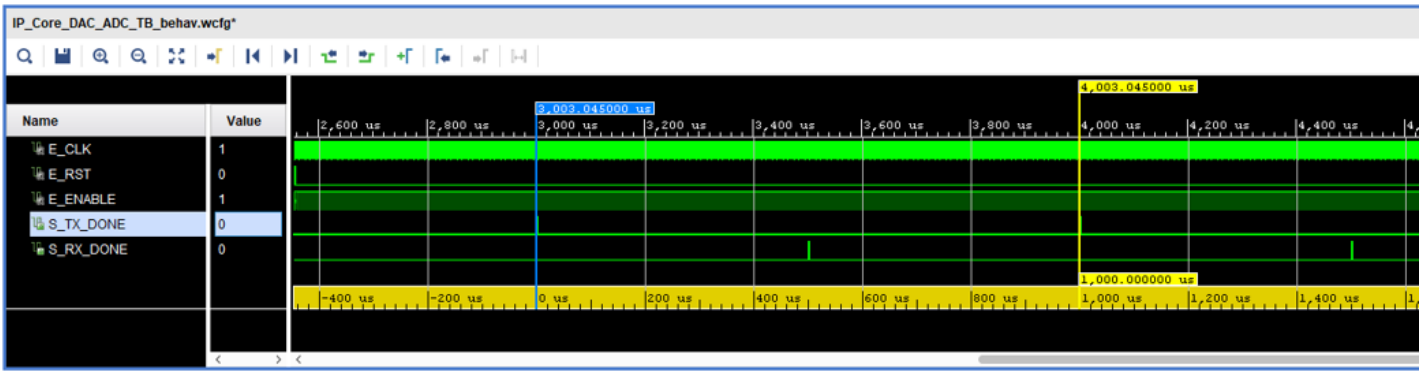

Figura 3.18 Simulación funcional correspondiente al controlador, muestra de las frecuencia de las bases de datos

Como se observa en la figura anterior, la frecuencia entre envíos y recepciones, desfasada entre ellos 180º, se ha configurado para este ejemplo para trabajar a 1kHz.

3.5.3 Módulo driver del conversor digital-analógico

El objetivo de este módulo es la comunicación con el Pmod DA2, fabricado por la compañía Digilent; se trata de un conversor digital analógico de dos canales simultáneos y 12 bits, con una interfaz GPIO.

El circuito integrado del Pmod hace uso de dos conversores DAC121S101, de Texas Instruments, que funcionan mediante un protocolo tipo SPI para gestionar la recepción de datos; el diagrama temporal que se debe cumplir se muestra en la figura 3.19.

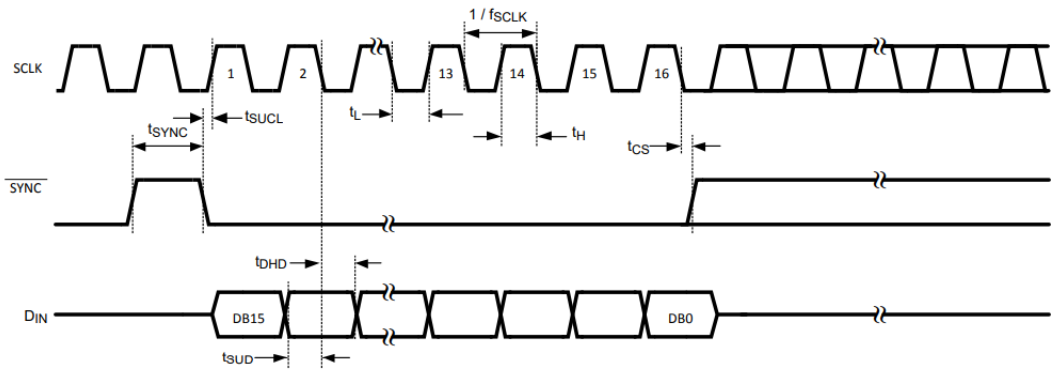

Figura 3.19 Cronograma temporal de conversores DAC121S101 [14]

Los conversores trabajan utilizando la tensión de alimentación como referencia para los niveles lógicos del canal de datos, por lo que las señales de datos (DIN) trabajarán entre 0 y 3.3V.

Interfaz

Dado que el driver es gestionado por el controlador descrito anteriormente, la interfaz del módulo se divide en dos bloques: con dicho controlador y con el Pmod DA2. En la figura 3.19 se muestra la distribución de los puertos entre los diferentes módulos.

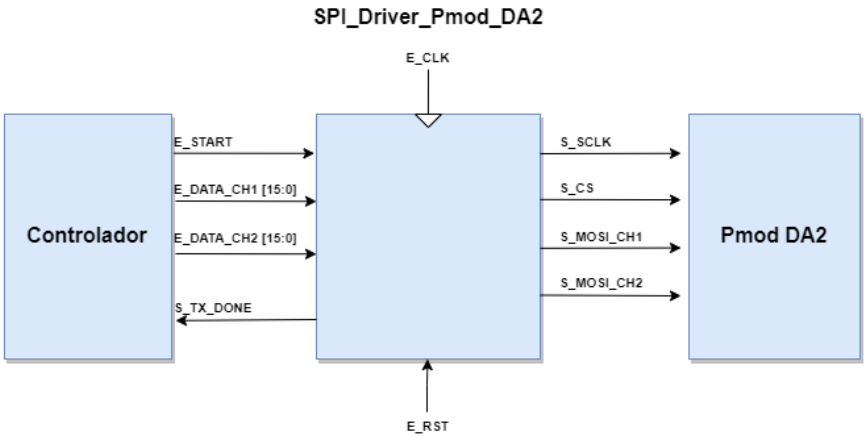

Figura 3.19 Diagrama de bloques del módulo driver del Pmod DA2

La descripción de cada puerto es explicada en la tabla 3.3, mostrada a continuación.

Tabla 3.3 Puertos del módulo driver del Pmod DA2

Puerto	Tamaño	Dirección	Tipo de dato	Interfaz	Función
E_CLK	1	In	Std_logic	Sistema	Reloj del sistema
E_RST	1	In	Std_logic	Sistema	Reset asíncrono activo a nivel alto
E_START	1	In	Std_logic	Controlador	Señal de inicio de transmisión
E_DATA_CH1	16	In	Std_logic_vector	Controlador	Datos recibidos para transmitir por canal 1
E_DATA_CH2	16	In	Std_logic_vector	Controlador	Datos recibidos para transmitir por canal 2
S_TX_DONE	1	Out	Std_logic	Controlador	Señal para indicar final de operación
S_SCLK	1	Out	Std_logic	PmodDA2	Reloj generado para PmodDA2
S_CS	1	Out	Std_logic	PmodDA2	Chip Selection para PmodDA2, señal Synk en los conversores integrados
S_MOSI_CH1	1	Out	Std_logic	PmodDA2	Canal de transmisión de datos a PmodDA2, canal 1
S_MOSI_CH2	1	Out	Std_logic	PmodDA2	Canal de transmisión de datos a PmodDA2, canal 2

Máquina de estados

La máquina de estados que rige el módulo consta únicamente de tres estados: IDLE, START y TX. En la figura 3.20 se muestra un esquema con la máquina de estados del módulo.

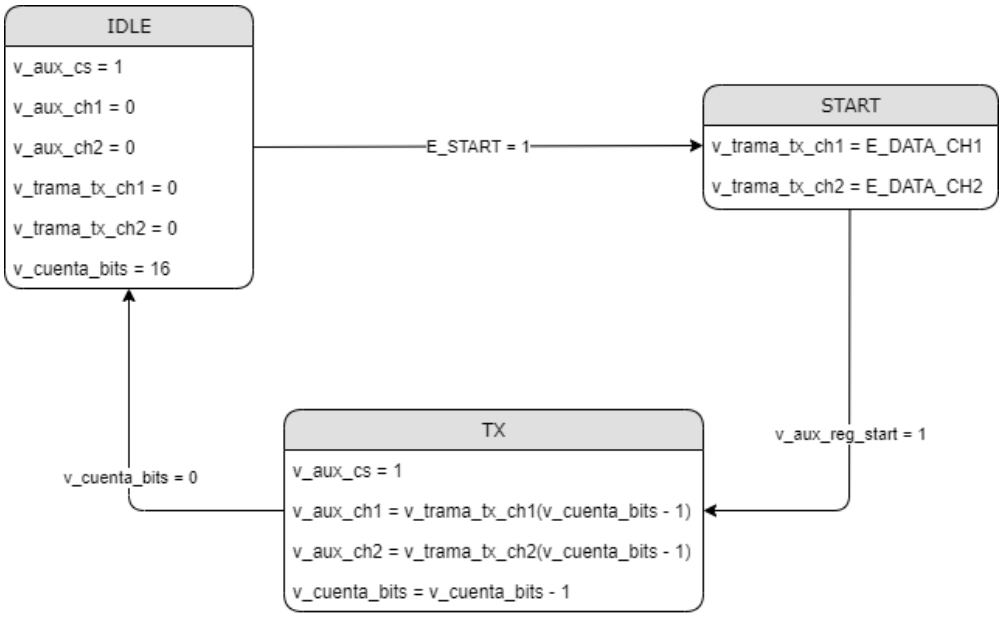

Figura 3.20 Máquina de estados del módulo driver del conversor digital-analógico

A continuación se hará una breve descripción de los estados de la máquina:

- **IDLE:** estado de reposo del sistema. El driver mantiene al conversor en reposo a la espera de órdenes del controlador de iniciar una transacción. Una vez que recibe la orden, pasa al estado START.
- **START:** estado en el que se registran los datos procedentes del controlador para ser transmitidos al conversor. Una vez registrados, se pasa al estado TX.
- **TX:** estado de transmisión. En este estado se transmiten uno a uno todos los bits de la trama de datos. Una vez que se han transmitido todos los bits, se avisa al controlador de que la transacción ha finalizado y se vuelve al estado de reposo a la espera de nuevas operaciones.

Funcionamiento del driver

En la figura 3.21 se muestra el resultado de la simulación funcional del módulo. Como se puede observar en la figura, el cronograma temporal de la figura 3.19 se corresponde con el resultado de la simulación. Dentro del módulo, la señal C_CS se corresponde con la señal SYNC, y, dado que el Pmod cuenta con dos conversores con los que se trabaja simultáneamente, DIN se corresponde con S_MOSI_CH1 y S_MOSI_CH2, uno para cada conversor.

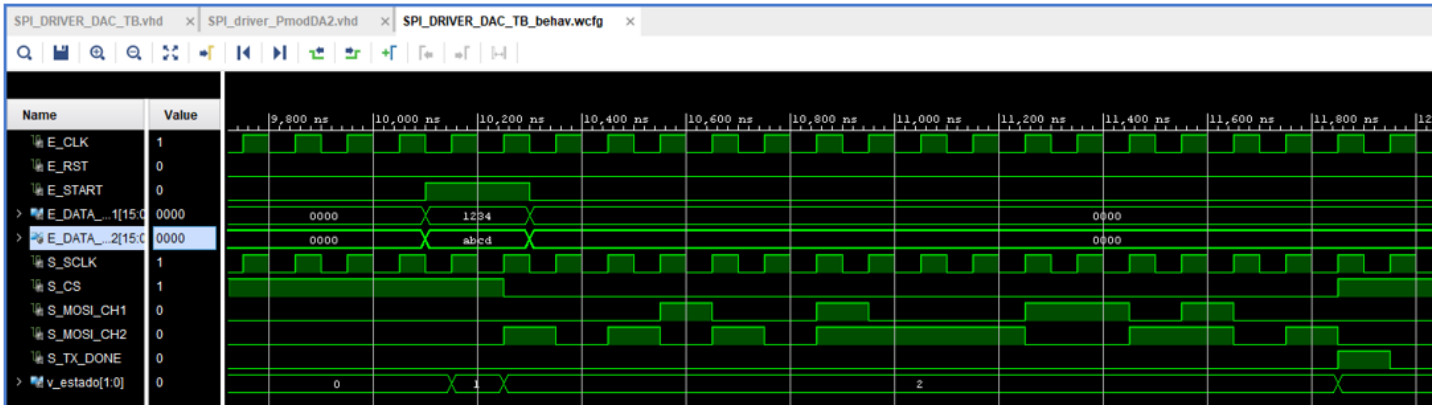

Figura 3.21 Simulación funcional del módulo driver del Pmod DA2

3.5.4 Módulo driver del conversor analógico-digital

El objetivo de este módulo es la de actuar como driver para el Pmod AD1. Se trata de un circuito integrado de Digilent; dicho CI consta de dos conversores AD7476A, conversores analógico-digitales con una interfaz GPIO y un protocolo de comunicación tipo SPI. El cronograma temporal para comunicarse con éstos se muestra en la figura 3.22.

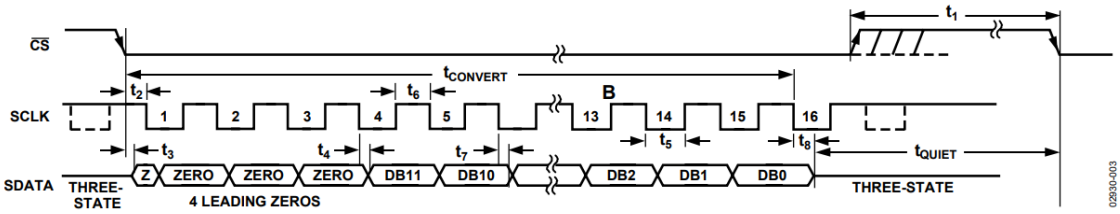

Figura 3.22 Cronograma temporal de conversores AD7476A [15]

Los conversores trabajan utilizando la tensión de alimentación como referencia para los niveles lógicos del canal de datos, por lo que las señales de datos (SDATA) trabajarán entre 0 y 3.3V.

Interfaz

La interfaz del módulo puede dividirse en dos bloques: por un lado, la que se corresponde con el controlador que comparte con el módulo driver del otro Pmod, y por el Pmod que se corresponde con este mismo módulo, el Pmod AD1. El diagrama con la distribución de los puertos se muestra en la figura 3.23.

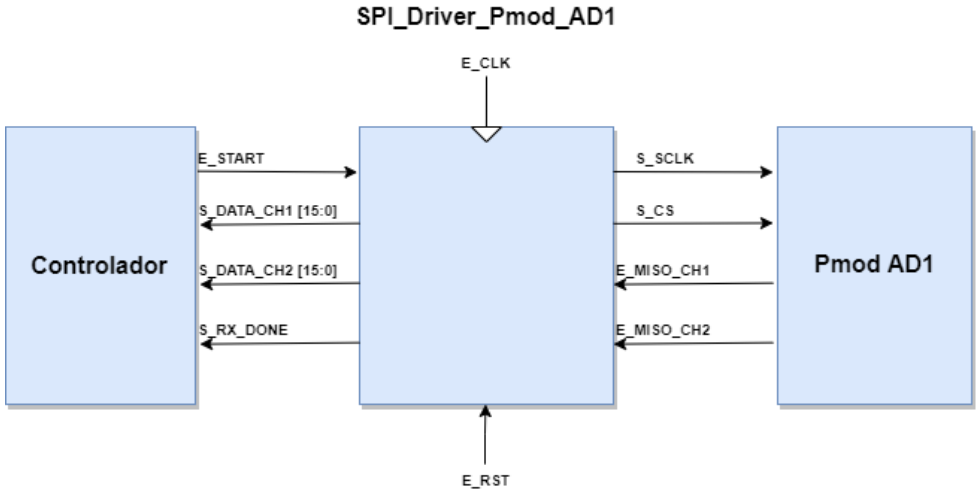

Figura 3.23 Diagrama de bloques del módulo driver del Pmod AD1

La función y características de los puertos del módulo se explican en la tabla 3.4.

Tabla 3.4 Puertos del módulo driver del Pmod AD1

Puerto	Tamaño	Dirección	Tipo de dato	Interfaz	Función
E_CLK	1	In	Std_logic	Sistema	Reloj del sistema
E_RST	1	In	Std_logic	Sistema	Reset asíncrono activo a nivel alto
E_START	1	In	Std_logic	Controlador	Señal de inicio de recepción
S_DATA_CH1	16	Out	Std_logic_vector	Controlador	Datos para transmitir a CTRL recibidos por canal 1
S_DATA_CH2	16	Out	Std_logic_vector	Controlador	Datos para transmitir a CTRL recibidos por canal 2

S_RX_DONE	1	Out	Std_logic	Controlador	Señal para indicar final de operación
S_SCLK	1	Out	Std_logic	PmodAD1	Reloj generado para PmodAD1
S_CS	1	Out	Std_logic	PmodAD1	Chip Selection para PmodAD1, señal CS en los conversores integrados
E_MISO_CH1	1	In	Std_logic	PmodAD1	Canal de recepción de datos de PmodAD1, canal 1
E_MISO_CH2	1	In	Std_logic	PmodAD1	Canal de recepción de datos de PmodAD1, canal 2

Máquina de estados

La máquina de estados del módulo consta únicamente de tres estados: IDLE, RX y END. En la figura 3.24 se muestra un esquema con la máquina de estados del módulo.

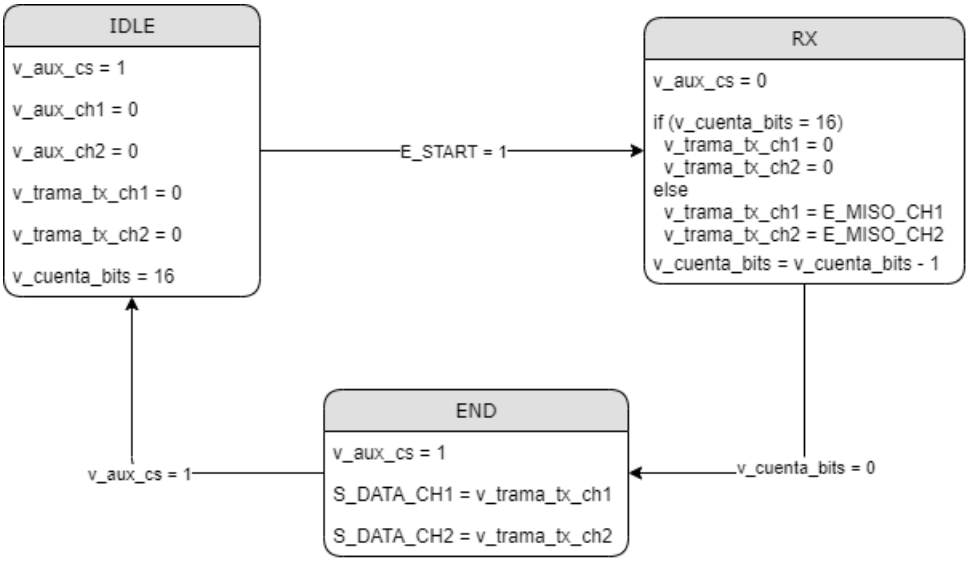

Figura 3.24 Máquina de estados del módulo driver del Pmod AD1

A continuación se hará una breve descripción de los estados de la máquina:

- **IDLE:** estado de reposo del sistema. El driver mantiene al conversor en reposo a la espera de órdenes del controlador de iniciar una transacción. Una vez que recibe la orden, pasa al estado RX.
- **RX:** estado en el que se inicia la recepción de datos desde el conversor analógico digital. Se reciben uno a uno todos los bits de la trama. Una vez que se han recibido todos, el sistema pasa al estado END.
- **END:** estado en el que se registran los datos recibidos por el ADC. Una vez registrados los datos, estos son enviados al controlador junto con el aviso de que la transacción ha finalizado; tras esto se pasa de nuevo al estado IDLE a la espera de nuevas operaciones.

Funcionamiento del driver

En la figura 3.25 se muestra el resultado de la simulación funcional del módulo. Como se puede observar en la figura, el cronograma temporal de la figura 3.22 se corresponde con el resultado de la simulación. Dentro del módulo, la señal C_CS se corresponde con la señal CS, y, dado que el Pmod cuenta con dos conversores con los que se trabaja simultáneamente, SDATA se corresponde con S_MISO_CH1 y S_MISO_CH2, uno para cada conversor.

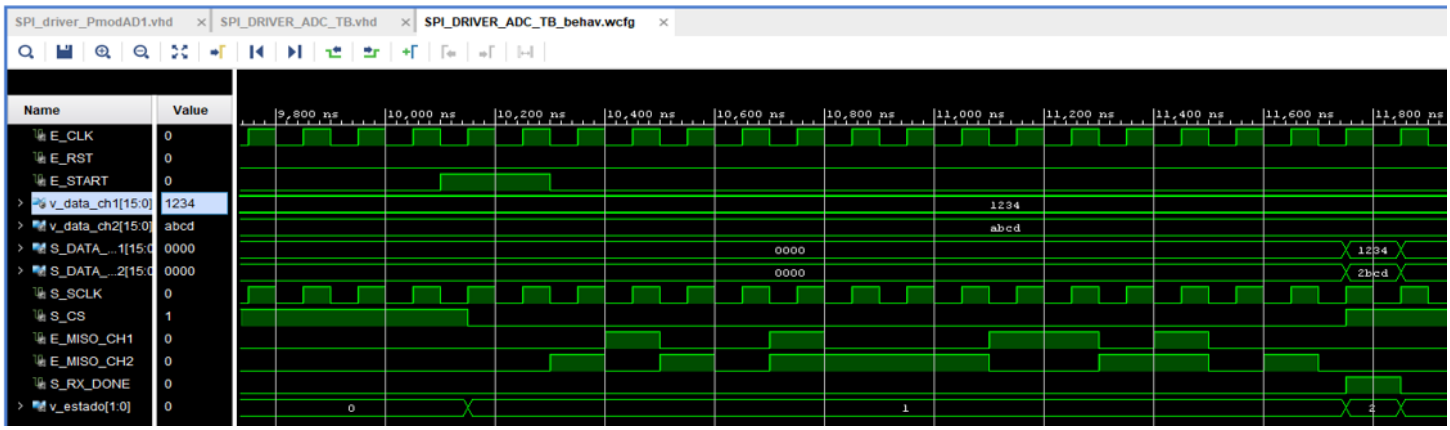

Figura 3.25 Simulación funcional del módulo driver del Pmod AD1

En este capítulo se ha abarcado todo lo relacionado con el diseño y la implementación del sistema completo, desde los drivers implementados en la FPGA que se comunican directamente con los conversores hasta las rutinas de atención a la interrupción del procesador que reciben los datos convertidos desde el hardware y los envían a un terminal a través del puerto serie. Debido a la necesidad de enviar estos datos a un terminal con Matlab para poder comparar resultados, la transferencia de datos es únicamente de un bloque de 16k de datos, no de las bases de datos completas ni en bucle; esta idea se explorará en el capítulo de trabajos futuros. En el siguiente capítulo se mostrarán los resultados experimentales obtenidos al ejecutar el sistema completo y las respuestas individuales de los módulos más importantes.

4. Resultados experimentales

Una vez explicado todo lo que se corresponde con la arquitectura del emulador diseñado, se han de realizar pruebas para comprobar el correcto funcionamiento de este. En primer lugar, se realizarán las simulaciones temporales post síntesis de los módulos diseñados para comprobar que cumplen con las condiciones temporales necesarias para su correcto funcionamiento. A continuación, se han añadido Analizadores Lógicos desde las librerías de Xilinx en forma de IPs prefabricadas para comprobar el funcionamiento del sistema en un experimento real. Finalmente se analizarán los resultados obtenidos comparando los datos originales con los datos tras pasar por el emulador en la herramienta Matlab.

4.1 Simulaciones temporales

En este punto se mostrarán los resultados de las simulaciones temporales post síntesis de los módulos que componen la parte hardware del sistema. Dado que los módulos, como se ha explicado en el capítulo anterior, están instanciados dentro de una IP que implementa el protocolo AXI-Lite4 y dependen de un reloj generado por el procesador para funcionar, no se pueden implementar por sí solos y por lo tanto obtener las simulaciones postimplementación. Es por ello que lo más cerca del funcionamiento real que podemos obtener en simulación es a través de la síntesis.

4.1.1 Conversor digital-analógicos

La figura 4.1 se corresponde con la simulación temporal post síntesis del módulo driver del DAC.

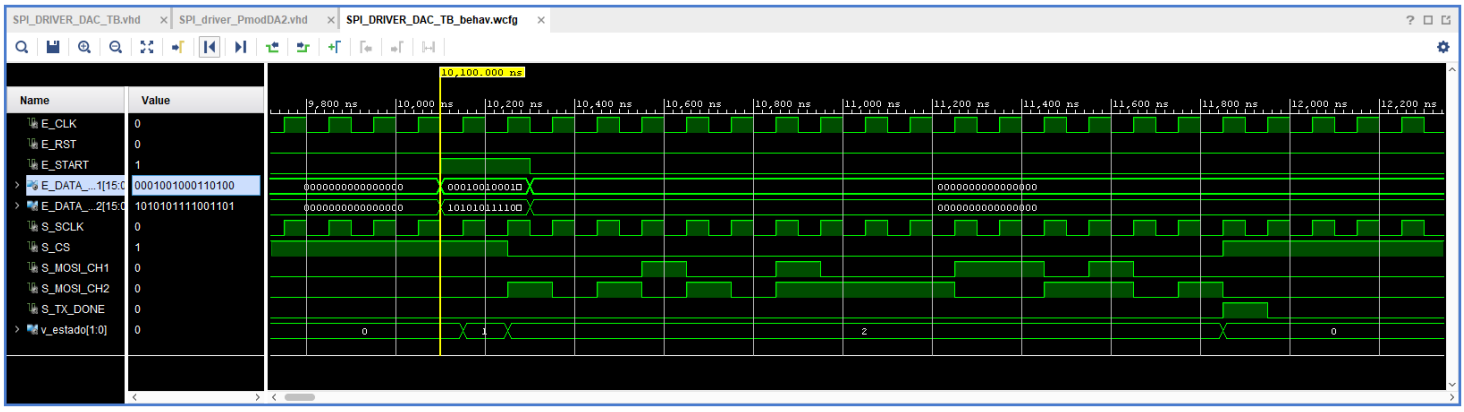

Figura 4.1 Simulación temporal post síntesis del módulo driver del Pmod DA2.

La figura 4.2 se corresponde con el módulo top que contiene tanto a controlador como a drivers; sin embargo, dicha simulación se centra en la operación de lectura de datos de las memorias y su posterior envío al conversor digital analógico.

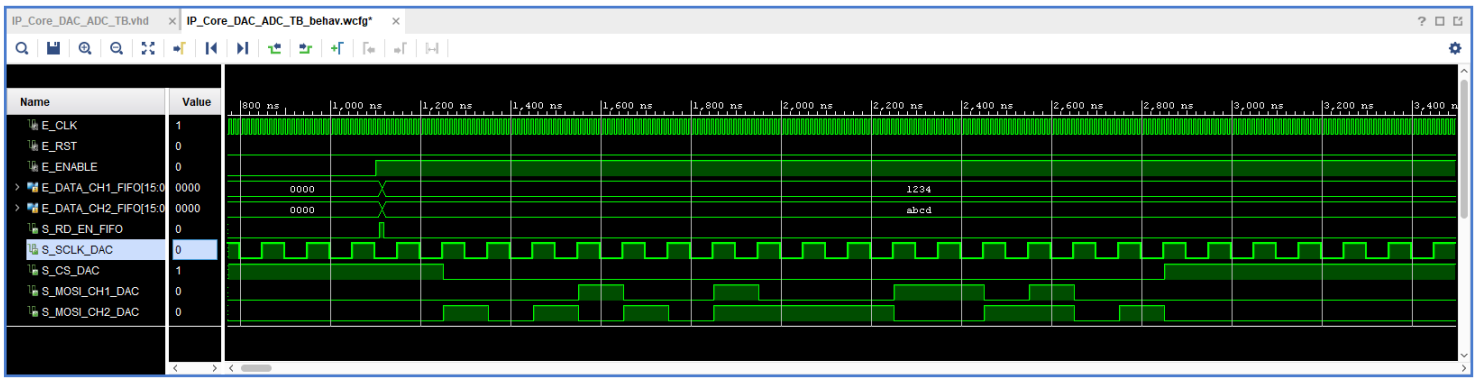

Figura 4.2 Simulación temporal post síntesis correspondiente al controlador, proceso de lectura de las FIFOs de entrada y operación de escritura a los conversores digital-analógicos.

Ambas figuras, la 4.1 y la 4.2, se ajustan a los cronogramas temporales establecidos en las especificaciones del capítulo anterior y mostrados mediante simulaciones funcionales en las figuras 3.21 y 3.17, por lo que se ajustan a los resultados esperados.

4.1.2 Conversor analógico-digital

La figura 4.3 se corresponde con la simulación temporal post síntesis del módulo driver del ADC.

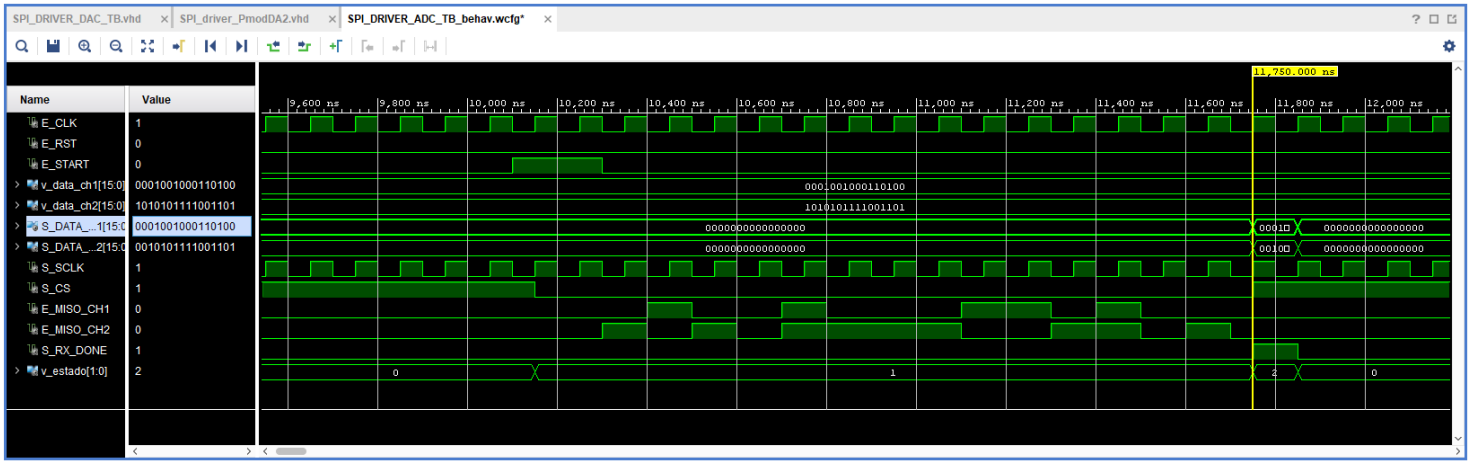

Figura 4.3 Simulación temporal post síntesis del módulo driver del Pmod AD1.

La figura 4.4 se corresponde con el módulo top que contiene tanto a controlador como a drivers; sin embargo, dicha simulación se centra en la operación de recepción de datos desde el conversor analógico-digital y la escritura de estos en las memorias FIFO de salida, a la espera de ser leídas desde el procesador.

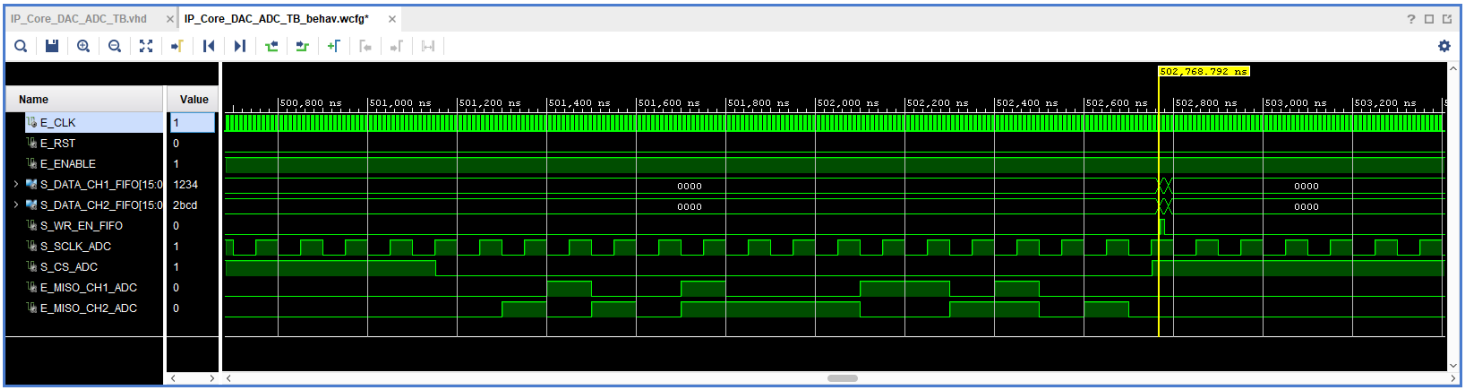

Figura 4.4 Simulación temporal post síntesis correspondiente al controlador, proceso de escritura de las FIFOs de entrada y operación de lectura a los conversores analógico-digitales.

Como se puede observar, tanto la figura 4.3 como la 4.4 se corresponden con las simulaciones funcionales mostradas en las figuras 3.25 y 3.16, respectivamente, por lo que cumplen con las especificaciones diseñadas. En el siguiente capítulo se mostrarán los resultados de un experimento real cargando el proyecto finalmente en la tarjeta Zynq, fuera del entorno de simulación.

4.2 Analizadores lógicos integrados

En este punto se mostrarán los resultados de diferentes partes del emulador mediante el uso de Analizadores Lógicos (ILA) implementados a través de IPs propias de Xilinx. Dichos analizadores se conectan a diferentes puertos del top de la parte hardware del proyecto que junta tanto la implementación del protocolo AXI como todos los módulos diseñados, como se muestra en la figura 4.5.

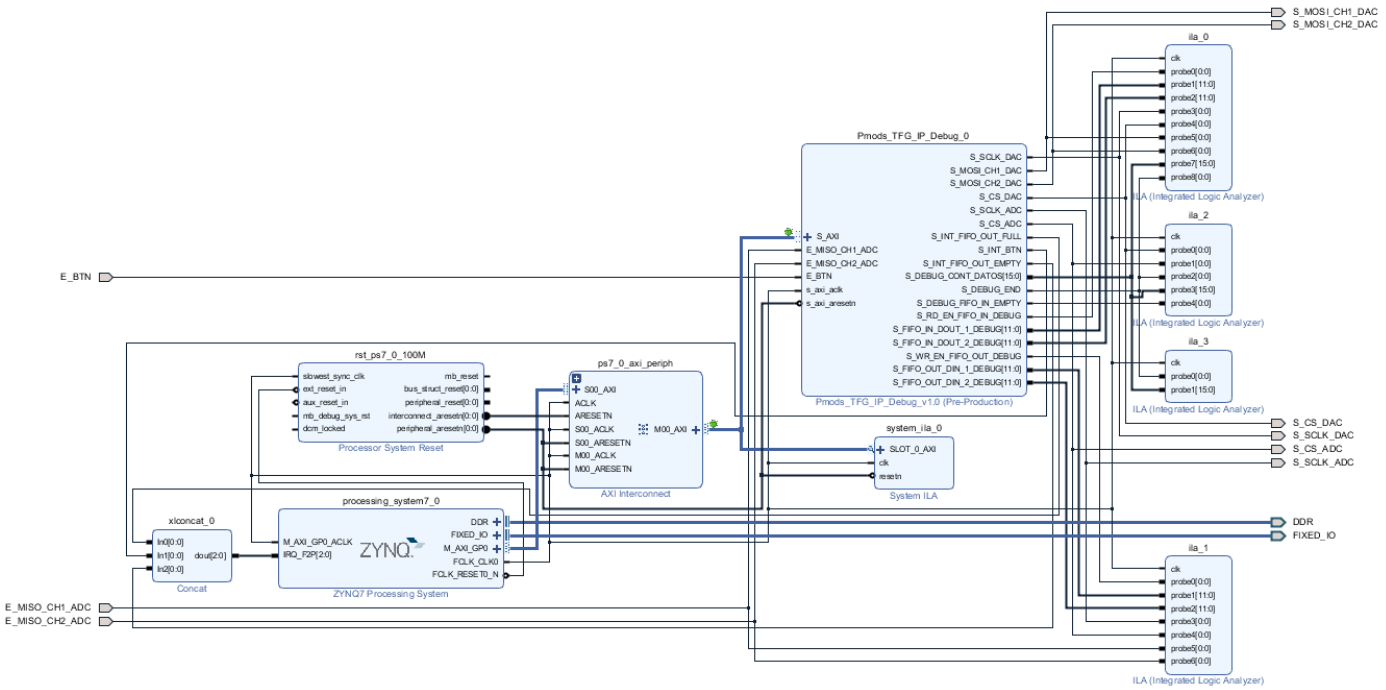

Figura 4.5 Diseño de bloques del emulador completo, incluyendo puertos y analizadores lógicos para proceso de depuración

Algunos de los puertos presentes en la figura 4.5 no forman parte del emulador ni son necesarios para su funcionamiento, sino que se han creado únicamente para el proceso de depuración y de demostración de los resultados experimentales.

4.2.1 Protocolo AXI

La primera ILA (Integrated Logic Analyzer) se conecta a los puertos del protocolo AXI. En este punto de depuración se observó un problema con respecto a la arquitectura propuesta en el capítulo anterior. Aunque el emulador funciona correctamente en el rango de frecuencias establecido (entre 1 y 12kHz), al añadir comunicación serie al proyecto para poder observar los resultados en otro terminal con Matlab, esto ralentizaba excesivamente el funcionamiento del emulador. Debido a esto, aunque a las frecuencias bajas del rango establecido el emulador funciona correctamente, a las frecuencias más altas el procesador no era capaz de enviar y recoger los datos de las memorias FIFO con la suficiente velocidad que el controlador de los drivers requería, llegando a vaciarse las memorias de entrada antes de que todos los datos de las bases fueran cargados en ellas. Es por ello que, aunque la arquitectura del hardware fue apenas alterada, se ha cambiado ligeramente el código software del proyecto para optimizar y acelerar el proceso.

De las tres interrupciones explicadas en el punto 3.4, Processing Subsystem y SDK, se han reducido a dos. Aunque la interrupción que configura la frecuencia de las bases de datos y hace la primera carga de datos a las memorias se mantiene intacta, las otras interrupciones se han juntado en una: cuando las memorias de salida tienen 256 datos o más, se activa la nueva interrupción, que se encarga en paralelo de leer dichos datos de las memorias de salida y de rellenar el mismo número de datos en las memorias de entrada, como se observa en la figura 4.6.

Figura 4.6 Analizador Lógico conectado a los buses del protocolo AXI, comunicación entre procesador y FPGA con UART

Sin embargo, como se observa en dicha figura, el tiempo entre la lectura de un dato y la escritura del siguiente es excesivamente alto. Esto se debe a que después de leer cada dato, éste se envía por el puerto serie al terminal que ejecuta la herramienta Matlab; dicha comunicación, aunque al máximo de velocidad estándar del protocolo UART, 115200 baudios, es lenta en comparación con el resto de proyecto, por lo que cuando la frecuencia de las bases de datos se acerca al límite superior establecido esta ralentización provoca que los datos no se envíen al hardware lo suficientemente rápido. Esta decisión se tomó pensando en reducir los recursos utilizados por el procesador,

eliminando la necesidad de almacenar los datos leídos, enviándose en cada lectura. En la figura 4.7 se muestra el mismo proceso que en la figura 4.6, pero eliminando la comunicación con el terminal después de cada lectura.

Figura 4.7 Analizador Lógico conectado a los buses del protocolo AXI, comunicación entre procesador y FPGA sin UART

Como se observa al comparar las figuras 4.6 y 4.7, al eliminar la comunicación serie entre lecturas se reduce considerablemente la latencia entre cada transacción. De esta manera, a frecuencias altas el emulador sigue funcionando correctamente. En la figura 4.8 se muestra el código de la nueva interrupción que aúna las anteriores de lectura y escritura de los datos tras la carga inicial.

```
void Read_FIFOs(void *InstancePtr){
  u32 aux;
  u8 prueba_uart_dato1[3], prueba_uart_dato2[3];
  int SentCount_d1 = 0, SentCount_d2 = 0;
  int i;

  for(i=0 ; i<256 ; i++){

  if(cont_muestras < 64 && write_en == 1)
    {
     aux = (muestras_base_datos_conv_2[i + 256*cont_muestras] << 16) + muestras_base_datos_conv_1[i + 256*cont_muestras];
     PMODS_TFG_IP_DEBUG_mWriteReg(XPAR_PMODS_TFG_IP_DEBUG_0_S_AXI_BASEADDR, 0, aux);
    }
  else
    {
      write_en = 0;
    }

    y = PMODS_TFG_IP_DEBUG_mReadReg(XPAR_PMODS_TFG_IP_DEBUG_0_S_AXI_BASEADDR, 0);

    if(cont_muestras_uart < 64){
    datos1[i + 256*cont_muestras_uart] = y;
    datos2[i + 256*cont_muestras_uart] = (y >> 16);
    }
    }

  cont_muestras++;
  cont_muestras_uart++;

  if(cont_muestras_uart == 64){
      for(i=0; i<16484; i++){
      prueba_uart_dato1[0] = datos1[i];
      prueba_uart_dato1[1] = datos1[i]>>8;
      prueba_uart_dato1[2] = '\0';

      prueba_uart_dato2[0] = datos2[i];
      prueba_uart_dato2[1] = datos2[i]>>8;
      prueba_uart_dato2[2] = '\0';

      while (SentCount_d1 < (sizeof(prueba_uart_dato1) - 1))
        {
        SentCount_d1 += XUartPs_Send(&Uart_Ps, &prueba_uart_dato1[SentCount_d1], 1);
        }

        while (SentCount_d2 < (sizeof(prueba_uart_dato2) - 1))
        {
          SentCount_d2 += XUartPs_Send(&Uart_Ps, &prueba_uart_dato2[SentCount_d2], 1);
        }

        SentCount_d1 = 0;
        SentCount_d2 = 0;
        }
```

Figura 4.8 Nueva interrupción para recepción de las bases de datos de salida una vez convertidas, cruzada con la escritura sucesiva de las bases de datos de entrada

Como se observa del código de la nueva interrupción mostrado en la figura 4.8, la transmisión con el terminal a través del puerto serie se realiza cuando se han terminado de leer los 16k de muestras de las bases de datos, almacenándose estos en arrays de

16384 datos; aunque esto aumenta el consumo de recursos del emulador, permite enviar los resultados a Matlab a las frecuencias más altas requeridas en el diseño.

4.2.2 Lectura de memorias de entrada y envío a conversores digital-analógicos

En este punto se presentarán los resultados de conectar los analizadores lógicos a los puertos correspondientes con las transacciones de datos, tanto entre memorias y controlador como controlador y drivers.

Como ya se describió en el capítulo sobre la arquitectura propuesta, siempre que haya datos en las memorias de entrada, el controlador leerá dichos datos y los enviará a la frecuencia configurada para las bases de datos hacia el conversor digital-analógico. Para ver los resultados de dicho proceso, un analizador lógico se ha conectado tanto a los puertos de datos y de control de escritura de las memorias de entrada como a todos los puertos correspondientes al conversor digital-analógico. Los resultados de dicho analizador se muestran en la figura 4.9.

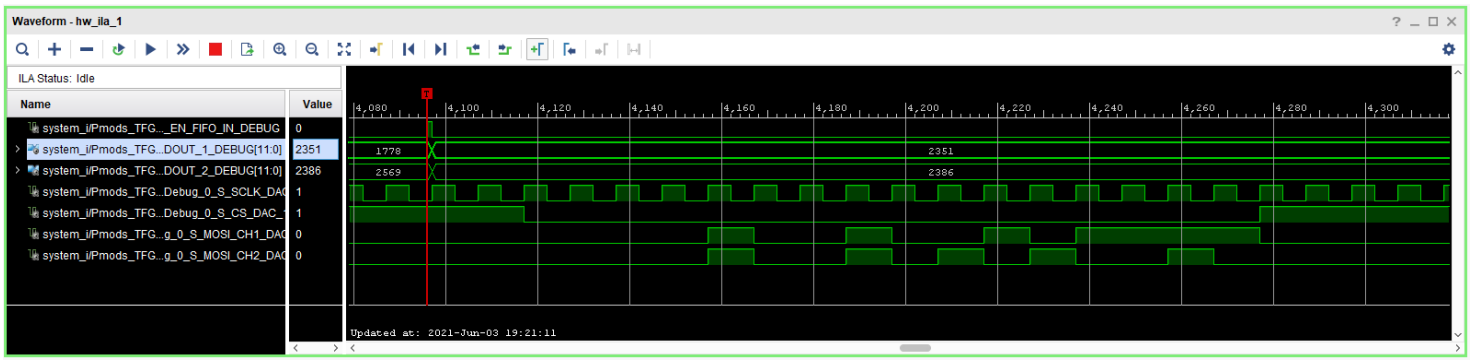

Figura 4.9 Analizador Lógico correspondiente al proceso de lectura de las FIFOs de entrada y operación de escritura a los conversores digital-analógicos

En paralelo con las lecturas de datos y su envío al conversor digital-analógico, tienen lugar la recepción de datos desde el conversor analógico-digital y su escritura en las memorias de salida, ambas a la misma frecuencia pero desfasada 180 grados. Dicho proceso se muestra en la figura 4.10.

Figura 4.10 Analizador Lógico correspondiente al proceso de escritura de las FIFOs de entrada y operación de lectura a los conversores analógico-digitales

La figura 4.10 se corresponde con la recepción de los datos enviados en la figura 4.9. Sin embargo, dada la limitación impuesta por la frecuencia de muestreo de las ILAs, que debe ser la misma que la frecuencia proporcionada por el procesador para toda la parte hardware del proyecto, 100MHz, no es posible mostrar en un mismo analizador lógico el proceso de envío y el de recepción, pues el tiempo que ocurre entre y otro (marcado por la frecuencia de las bases de datos), es del orden de kHz.

4.2.4 Frecuencia de las bases de datos

El último analizador lógico implementado en el sistema para depuración y muestra de resultados tiene el objetivo de comprobar que se cumple con la frecuencia marcada por el procesador para las bases de datos.

Para medir el tiempo entre muestras y comprobar que se cumple con la frecuencia requerida se han utilizado las señales de Chip Selection de ambos conversores; cada vez que se produce un flanco de bajada en cada uno de ellos, significa que se produce una transacción de datos, por lo que al medir el tiempo entre flancos de bajada se puede obtener la frecuencia a la que están siendo muestreadas las bases de datos. En la figura 4.11 se muestra el resultado descrito.

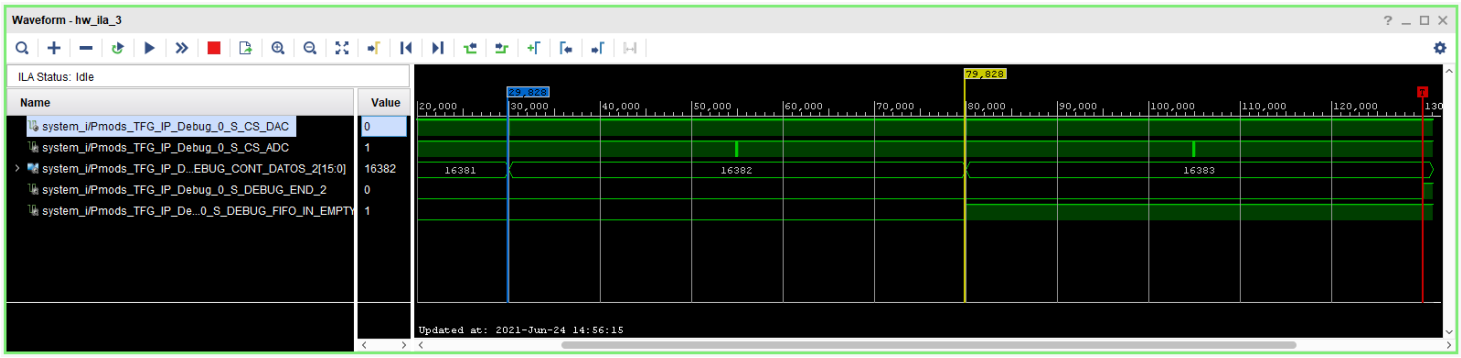

Figura 4.11 Analizador Lógico correspondiente a la frecuencia de trabajo de las bases de datos, 2kHz

La frecuencia configurada para obtener los resultados de la figura anterior es de 2kHz; dado que el analizador muestrea a 100MHz, el número de muestras entre flacos (50000), da lugar a un tiempo entre flancos de 500us, 2kHz. Al igual que ocurría en el punto anterior, trabajar a frecuencias bajas impide poder observar los resultados en los analizadores lógicos. El resto de las señales que se representan en el analizador lógico tienen la finalidad de comprobar que, además de que se cumple la frecuencia de muestro de las bases de datos, también se transfieren al completo los 16k de datos sin interrupciones, de manera que la frecuencia se mantiene estable durante todo el proceso.

Para ello se ha implementado un reloj dentro del hardware con la misma frecuencia que la de las bases de datos; dicho reloj se reinicia cada vez que se envía una pareja de datos al conversor digital-analógico. De esta manera, el reloj no debe finalizar la cuenta mientras queden datos que enviar hacia el conversor; dicho reloj lleva la cuenta de las veces que ha sido reiniciado (los envíos al conversor), cuenta que se registra en la señal DEBUG_CONT_DATOS, presente en el analizador lógico. Cuando finalmente los datos han sido enviados al completo y el reloj finaliza su cuenta, activa la señal DEBUG_END,

desencadenando la condición de trigger del analizador lógico. De esta manera se puede observar en las figuras 4.11 y 4.12 (frecuencia de muestreo de 12kHz, la máxima requerida), que la frecuencia de muestreo de las bases de datos se mantiene estable durante todo el proceso y que han enviado los 16k de muestras al completo.

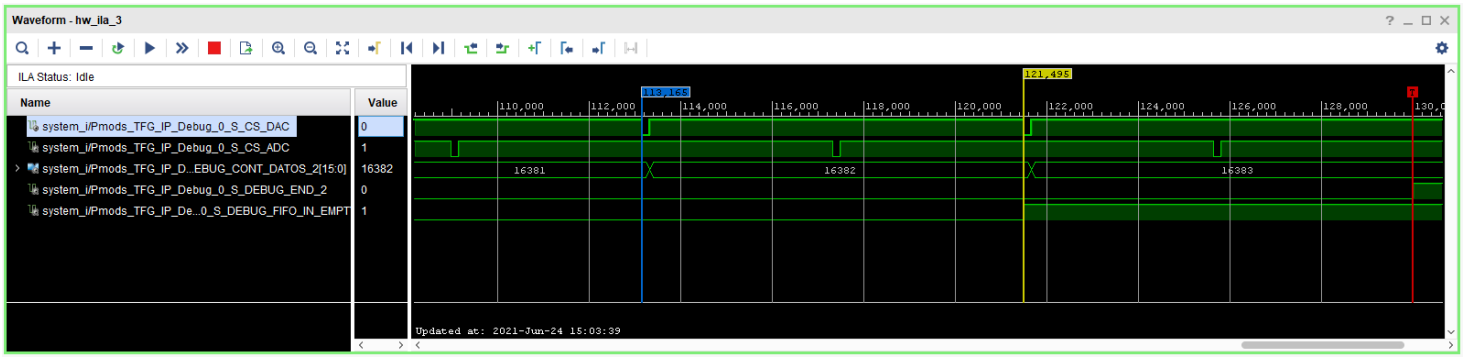

Figura 4.12 Analizador Lógico correspondiente a la frecuencia de trabajo de las bases de datos, 12kHz

Al igual que en el caso anterior, el número de muestras entre flancos (8330), esta vez más visibles, da lugar a 83,3us, 12.004kHz.

4.3 Utilización de recursos, consumo de potencia y frecuencia máxima de trabajo

En este punto se expondrán, tanto los recursos que consume el sistema al completo y cada una de sus partes, como la frecuencia máxima a la que puede llegar a trabajar el emulador. Para ello, dado que un gran número de los puertos y bloques mostrados en la figura 4.5 tienen como finalidad la depuración y la visualización de resultados experimentales de algunas partes del sistema y no son necesarios para el funcionamiento del emulador, se han eliminado del proyecto final, quedando como el mostrado en la figura 4.13.

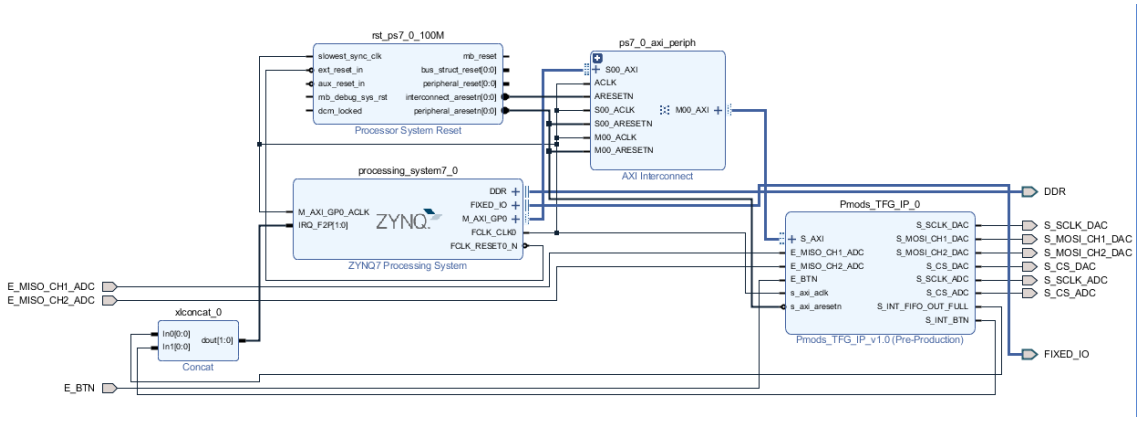

Figura 4.13 Diseño de bloques del emulador completo final

4.3.1 Utilización de recursos lógicos

En este punto se mostrarán los recursos de lógica programable de la FPGA utilizados por el sistema, de manera que se podrá observar cuánto espacio de la lógica queda libre para futuras implementaciones. La utilización de los recursos (explicados en el capítulo del estado del arte) se dividirá en lo utilizado por cada módulo, para luego realizar un recuento del total del sistema completo.

Los módulos de los drivers, el controlador que los gestiona y el módulo antirrebotes son los únicos módulos diseñados enteramente nuevos, sin ninguna generación por parte de Vivado. En las tablas 4.1, 4.2, 4.3 y 4.4 se muestran los recursos que consumen el driver del conversor digital-analógico, el del conversor analógico-digital, el controlador y el anti rebotes respectivamente.

Tabla 4.1 Recursos lógicos utilizados por el módulo driver del conversor digital-analógico

Recurso	Utilizado	Disponible	Utilizado %
LUT	45	53200	0.085
LUTRAM	0	17400	0
FF	36	106400	0.034
BRAM	0	140	0
IO	4	125	3.2

Tabla 4.2 Recursos lógicos utilizados por el módulo driver del conversor analógico-digital

Recurso	Utilizado	Disponible	Utilizado %
LUT	53	53200	0.099
LUTRAM	0	17400	0
FF	57	106400	0.054
BRAM	0	140	0
IO	4	125	3.2

Tabla 4.3 Recursos lógicos utilizados por el módulo controlador de los drivers y las memorias

Recurso	Utilizado	Disponible	Utilizado %
LUT	90	53200	0.169
LUTRAM	0	17400	0
FF	180	106400	0.169
BRAM	0	140	0
IO	0	125	0

Tabla 4.4 Recursos lógicos utilizados por el módulo anti rebotes del botón

Recurso	Utilizado	Disponible	Utilizado %
LUT	35	53200	0.066
LUTRAM	0	17400	0
FF	27	106400	0.025
BRAM	0	140	0
IO	1	125	0.8

Los recursos consumidos por las memorias FIFO se han aunado en una única tabla, dado que se generan con la misma IP de Vivado y con las mismas características.

Tabla 4.5 Recursos lógicos utilizados por el total de las cuatro memorias FIFO

Recurso	Utilizado	Disponible	Utilizado %
LUT	199	53200	0.374
LUTRAM	0	17400	0
FF	202	106400	0.190
BRAM	2	140	1.429
IO	0	125	0

Aunque el bloque TOP que implementa toda la lógica de la parte hardware del sistema contiene instanciados todos los módulos de las tablas anteriores, también contiene la lógica del protocolo AXI, así como la lógica de escritura y lectura de las memorias que le corresponden a dicho protocolo. El consumo de los recursos de esta lógica se muestra en la tabla 4.6.

Tabla 4.6 Recursos lógicos utilizados por la lógica del protocolo AXI

Recurso	Utilizado	Disponible	Utilizado %
LUT	45	53200	0.085
LUTRAM	0	17400	0
FF	64	106400	0.060
BRAM	0	140	0
IO	0	125	0

Si se observa la figura 4.13, hay dos bloques de los que aún no se ha referido a lo largo del documento; éstos son los bloques de AXI Interconnect y Processor System Reset. Esto se debe a que estos bloques son generados automáticamente por Vivado al añadir un bloque con lógica AXI y la generación de un reloj para la FPGA respectivamente. Aunque la lógica interna de estos bloques es irrelevante para el desarrollo del proyecto, consumen recursos, por lo que deben ser mencionados en este punto. Los recursos de éstos se muestran en las tablas 4.7 y 4.8.

Tabla 4.7 Recursos lógicos utilizados por el bloque de interconexión AXI generado automáticamente por Vivado

Recurso	Utilizado	Disponible	Utilizado %
LUT	264	53200	0.496
LUTRAM	51	17400	0.293
FF	375	106400	0.352
BRAM	0	140	0
IO	0	125	0

Tabla 4.8 Recursos lógicos utilizados por el bloque de reset del reloj generado automáticamente por Vivado

Recurso	Utilizado	Disponible	Utilizado %
LUT	16	53200	0.030
LUTRAM	1	17400	0.006
FF	33	106400	0.031
BRAM	0	140	0
IO	0	125	0

Finalmente, en la tabla 4.9 se muestra el consumo de recursos total de toda la lógica programable del sistema.

Tabla 4.9 Recursos lógicos totales utilizados por el sistema completo

Recurso	Utilizado	Disponible	Utilizado %
LUT	747	53200	1.404
LUTRAM	52	17400	0.299
FF	974	106400	0.915
BRAM	2	140	1.429
IO	9	125	7.2

Como se puede observar, exceptuando los puertos de entrada/salida, ocupados por los conversores y el botón que inicia la transferencia de datos, el porcentaje de lógica ocupada es muy pequeño, por lo que queda espacio para futuras implementaciones.

4.3.2 Consumo de potencia

Tras la implementación, Vivado también realiza un reporte en el que hace una estimación de la potencia empleada al ejecutar el sistema. Dicha estimación de potencia se muestra en la figura 4.14.

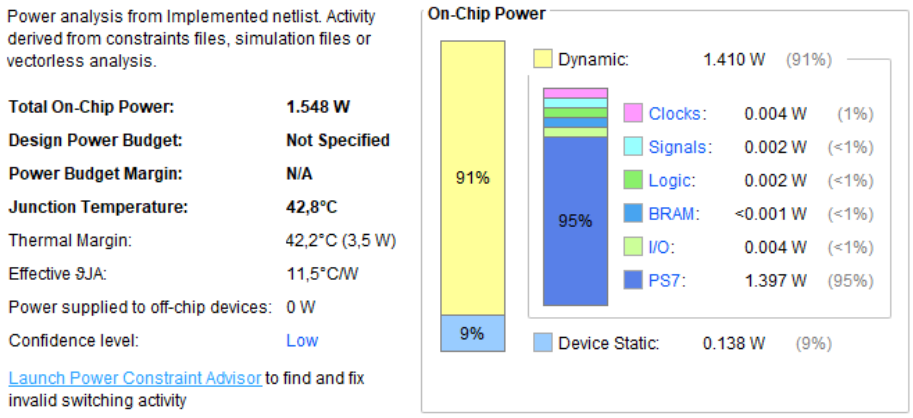

Figura 4.14 Consumo de potencia del sistema

Como se advierte en la figura anterior, la gran mayoría del consumo de potencia viene de la mano del procesador, siendo apenas un 5% del total la gastada por la parte lógica.

4.3.3 Frecuencia máxima de trabajo

En cuanto a la frecuencia máxima de trabajo, hay dos diferentes sobre las que comentar: la frecuencia máxima de la FPGA generada por el procesador, que controla toda la parte hardware del sistema, y la frecuencia máxima de muestreo de las bases de datos.

El reloj generado por el procesador es de 100MHz. Cuando Vivado realiza la implementación, pone a disposición del diseñador una serie de reportes, como el consumo de recursos utilizado en el punto anterior o, en este caso, el reporte de las características temporales. En la figura 4.14 se muestra el resumen del contenido del reporte referido a la frecuencia del sistema.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4,306 ns	Worst Hold Slack (WHS):	0,048 ns	Worst Pulse Width Slack (WPWS):	4,020 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	2289	Total Number of Endpoints:	2289	Total Number of Endpoints:	946

All user specified timing constraints are met.

Figura 4.15 Resumen del reporte temporal de la implementación del sistema

El menor periodo para la señal de reloj en el que el sistema funciona correctamente es la suma los peores valores. Dicho valor suma 8,37ns, por lo que la frecuencia máxima de funcionamiento del sistema es de 119MHz.

En cuanto a la frecuencia máxima de muestreo de las bases de datos, hay dos partes del sistema que la limitan: la frecuencia del protocolo AXI y los tiempos de establecimiento de ambos conversores.

Dado que el controlador de los drivers enviará y recibirá datos desde los conversores siempre y cuando tenga datos en las memorias de entrada, la frecuencia máxima vendrá determinada por la capacidad del protocolo AXI de mantener a las memorias de entrada con datos siempre disponibles para el controlador, así como de vaciar las memorias de salida para evitar que se pierdan datos por desbordamiento. Al acudir a la figura 4.7, resultado de conectar los puertos del protocolo AXI a un analizador lógico, se observa que entre cada transacción (envío y recepción de un dato), se llevan a cabo 49 pulsos de muestreo del analizador lógico; dado que el analizador muestrea con el reloj de 100MHz, eso significa que cada transacción toma unos 0.5us para completarse. Dado que esta transacción supone el envío de un nuevo dato a las memorias de entrada y la recepción de un dato desde las memorias de salida, el límite teórico para que el controlador siempre tenga un dato disponible en las memorias de entrada y nunca desborden las memorias de salida es de 2MHz.

Por otro lado, si se acude a las fichas técnicas de los conversores, referenciadas en la bibliografía, se observa que el conversor más limitante es el digital-analógico, con tiempo de establecimiento típico para la señal de salida de 12us, lo que da una frecuencia de 83,3kHz, siendo esta menor que la del protocolo AXI y por lo tanto la más limitante del sistema; sin embargo, dado que las técnicas NILM se aplican a frecuencias máximas de en torno a 12 o 16kHz, el sistema cumple con los requisitos de muestreo de las bases de datos.

4.4 Resultados finales en Matlab

En este punto se mostrarán los resultados finales de todo el sistema, obtenidos en Matlab.

Una vez que el emulador y el sistema de adquisición de datos terminan su tarea, todos los datos que han pasado por los conversores son enviados a un terminar a través del puerto serie; en dicho terminal y a través de un script de Matlab, a los datos originales de las bases de datos se les restan los datos pasados por los conversores, de manera que se obtiene la diferencia entre ambas señales y se tiene una estimación del error generado al pasar los datos por los conversores y el ruido asociado que ello conlleva. Finalmente, todas las señales comentadas (datos originales, datos convertidos, diferencia) se muestran en pantalla. En la figura 4.16 se observan dichos resultados, para una muestra de 200 datos, muestreados a una frecuencia de 1kHz. Aunque en Matlab se han obtenido las 16k de muestras, se muestran solo los 200 primeros de manera que pueda observarse más fácilmente la forma de cada señal.

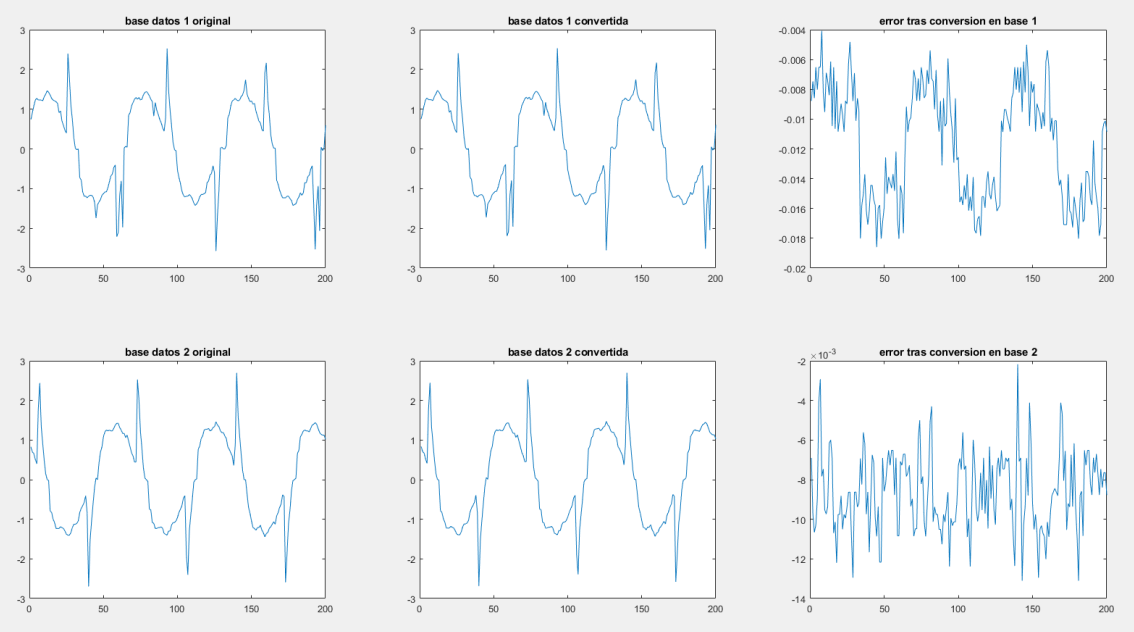

Figura 4.16 Gráfico de Matlab con la comparativa de los datos originales, los datos tras pasar por el emulador y la diferencia entre ellos, 200 muestras, 1kHz de frecuencia de muestreo

A través del mismo script se obtiene que la media del error absoluto es, para la primera base de datos, -0.0118, -0.0087 en el caso de la segunda. Como se observa en la figura, el orden del es del error es 10^{-3}, por lo que es prácticamente despreciable. En las figuras 4.17 y 4.18 se muestra el mismo gráfico, pero a frecuencias de 10 y 20kHz respectivamente.

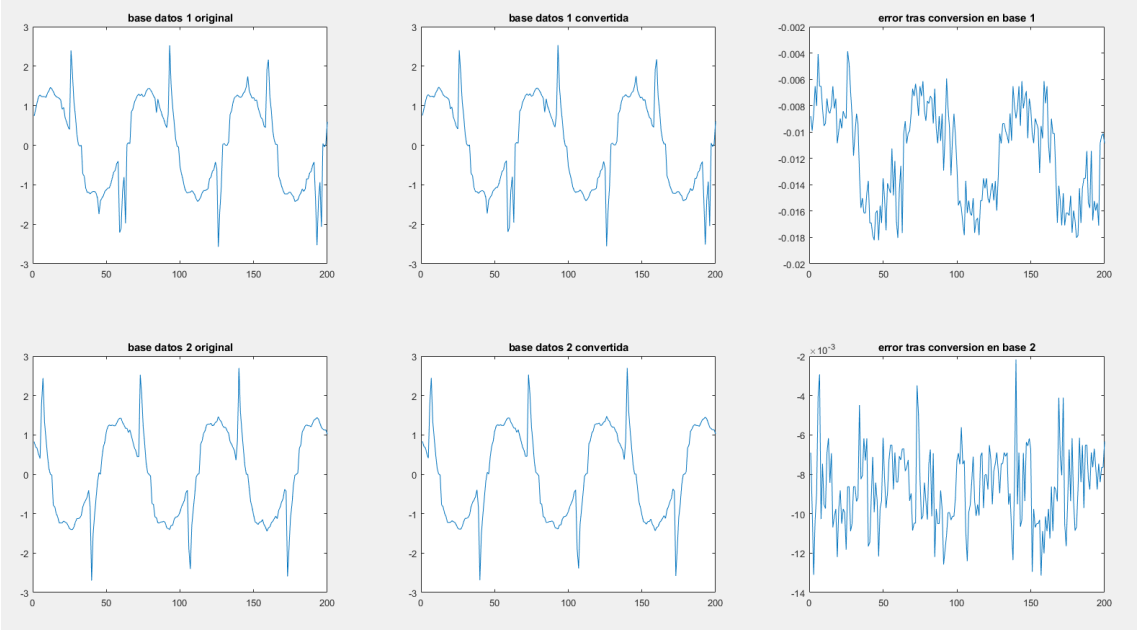

Figura 4.17 Gráfico de Matlab con la comparativa de los datos originales, los datos tras pasar por el emulador y la diferencia entre ellos, 200 muestras, 10kHz de frecuencia de muestreo

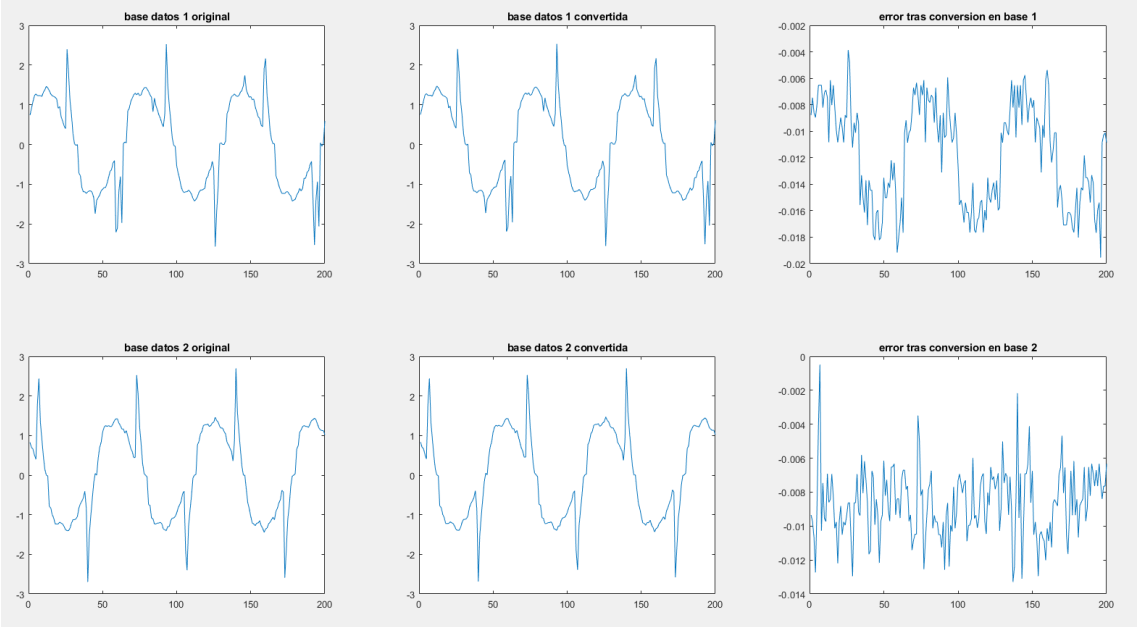

Figura 4.18 Gráfico de Matlab con la comparativa de los datos originales, los datos tras pasar por el emulador y la diferencia entre ellos, 200 muestras, 20kHz de frecuencia de muestreo

La media del error absoluto calculado al subir la frecuencia es, en todos los casos, la misma (dicha media ha sido calculada sobre el total de los datos, no sobre las 200 muestras que se ven gráficamente). Como se observa en las tres figuras la diferencia se mantiene estable en todo el rango de frecuencias, por lo que el sistema se comporta de la misma manera en todo el rango de frecuencias aceptables para la experimentación con técnicas NILM.

En este capítulo se ha mostrado el comportamiento de los módulos que conforman el sistema, así como los resultados finales obtenidos en Matlab; dichos comportamientos y resultados se ajustan a lo esperado. En el siguiente capítulo se comentarán las conclusiones del trabajo y las posibles mejores y líneas de trabajo futuras.

5. Conclusiones y trabajos futuros

En este capítulo se comentarán las conclusiones derivadas de los capítulos anteriores, analizando el comportamiento del sistema. También se hablará de posibles mejoras que podrían aplicarse y de líneas de trabajo a seguir en el futuro.

5.1 Conclusiones

El objetivo principal de este trabajo era ser capaz de emular el comportamiento de un contador inteligente para crear un entorno accesible en el que experimentar con técnicas de desagregación de la energía. Dicho entorno debe ser capaz de trabajar con diferentes bases de datos a diferentes frecuencias de muestreo, siempre en el rango de trabajos de las técnicas NILM. El sistema de adquisición de datos, aunque en primera instancia su objetivo no es otro que validar que el emulador se comporta correctamente y según lo esperado, en trabajos futuros puede servir como la primera fase de las técnicas computacionales sobre las que trata este trabajo, como ya se describió en el capítulo del estado del arte.

Dado que el sistema está diseñado como prueba de concepto para tratar de demostrar si este entorno es adecuado para emular el comportamiento de un contador inteligente, hacen falta incorporar cambios y mejoras para estar listo para dicha tarea. Sin embargo, observando los resultados finales obtenidos en el capítulo anterior, el emulador funciona de la misma manera en el rango de frecuencias buscado, sin apenas error en la señal recogida por el sistema de adquisición de datos, por lo que se puede concluir que resulta apto para ser utilizado para la experimentación con técnicas NILM.

5.2 Mejoras y trabajos futuros

En cuanto a la cuestión sobre los trabajos futuros, la primera mejora debería ser que el emulador funcionara de manera cíclica, sin parar en un número concreto de muestras. Debido a la necesidad de mandar las muestras a Matlab para comparar los resultados del emulador y el sistema de adquisición de datos, debía ser un número finito de muestras; sin embargo, para la experimentación con técnicas NILM en trabajos futuros, con pequeños cambios se puede hacer que el sistema funcione cíclicamente.

En cuanto a mejoras en el sistema, para establecer la comunicación y envío de los datos recogidos a otro terminal mientras se mantiene al emulador funcionando cíclicamente, se exploran varias opciones. Desde el lado de la comunicación procesador-FPGA, existe el limitante de que el protocolo AXI-Lite4 no puede establecer comunicaciones en modo ráfaga, si no que envía los datos de uno en uno, estableciendo una nueva comunicación con cada dato que escribe o lee, como se puede ver la figura 4.7 ya comentada, lo que limita la velocidad; para solventar este problema y dar más tiempo a la comunicación por el puerto serie, puede utilizarse la interfaz de AXI-Stream en lugar de AXI-Lite, lo que permitiría escribir y leer los paquetes de datos de las bases en modo ráfaga. Desde el lado de la comunicación terminal-procesador, para aumentar la velocidad se pueden utilizar otros protocolos más rápidos como Ethernet o USB; si se desea mantener el uso del puerto serie, otra solución viable es la de implementar la UART desde hardware en la FPGA, liberando al procesador de esta tarea. De esta manera, los datos podrían

enviarse directamente desde el hardware, sin necesidad de que se envíen antes al procesador.

Finalmente, el siguiente paso natural de este trabajo es la experimentación con técnicas NILM. Dado que tanto el emulador como el sistema de adquisición de datos han dado los resultados esperados, un posible trabajo en el futuro podría ser continuar en esta línea de investigación.

6. Pliego de condiciones

En este capítulo se hará un recuento de todo el material, tanto software como hardware, utilizado durante el desarrollo de este trabajo, así como la función que ha desempeñado cada elemento.

Los elementos software han sido: Vivado HLx Editions, Xilinx Software Development Kit, el paquete de Microsoft Office, Matlab y la base de datos BLUED. En el caso de Vivado HLx Editions y Xilinx SDK, han sido las plataformas sobre las que se ha desarrollado y depurado todo el sistema, Vivado la parte hardware y SDK la parte software; la base de datos BLUED ha sido utilizada como base del sistema, usando sus datos para emular el comportamiento de un contador inteligente conectado a una vivienda. Matlab se ha utilizado tanto para transferir el contenido de las bases de datos a ficheros .h, aptos para el procesador, como para recibir los datos desde el mismo tras pasar por todo el sistema y mostrar gráficamente los resultados. Finalmente, Microsoft Office se ha utilizado para redactar este mismo documento.

En cuanto a los elementos hardware utilizados, han sido: computadora de uso personal, tarjeta Zybo z7-20, Pmod DA2 y Pmod AD1. En la computadora de uso personal se ha desarrollado todo el trabajo y se han utilizado todos los elementos software comentados anteriormente. En la tarjeta Zybo z7-20 se ha implementado el sistema; los dos Pmods, ambos conversores, son los periféricos utilizados para sacar y volver a meter los datos en el entorno digital de la tarjeta.

En el siguiente capítulo se realizará el presupuesto de la realización de este trabajo recopilando los costes de los elementos nombrados.

7. Presupuesto

En este capítulo se hará un recuento del coste de los elementos enumerados en el punto anterior y con ello el presupuesto de la realización del presente trabajo.

El coste asociado al software se debe a las licencias de los programas utilizados. Estos programas son: Vivado HLx Editions, Xilinx Software Development Kit, el paquete de Microsoft Office, Matlab y la base de datos BLUED. El coste de cada producto se resume en la tabla 6.1.

Tabla 6.1 Costes asociados al Software utilizado

Licencia	Fabricante/Vendedor	Coste
Vivado 2017.4	Xilinx	0 €
Xilinx SDK 2017.4	Xilinx	0 €
Microsoft Office	Microsoft/UAH	0 €
Matlab	MathWorks/UAH	0 €
Base de datos BLUED	UAH	0 €
Total		0 €

Los programas de Xilinx son de uso gratuito y la base de datos ha sido descargada de la nube para realización de este trabajo. Tanto el paquete de Microsoft Office como Matlab son licencias de pago, sin embargo, se han utilizado licencias de estudiante para llevar a cabo el trabajo, por lo que en este caso son gratuitas. Los costes asociados a todo el software son nulos.

El material hardware utilizado es: tarjeta Zybo Z7-20, Pmods AD1 y DA2, y ordenador personal en el que se ha desarrollado y depurado todo el sistema. En la tabla 6.2 se resume el coste de las herramientas mencionadas.

Tabla 6.2 Costes asociados al Hardware utilizado

Producto	Fabricante/Vendedor	Coste
Zybo Z7-20	Digilent/Digi-key	250,44 €
Pmod AD1	Digilent/Digi-key	25,12 €
Pmod DA2	Digilent/Digi-key	16,74 €
Ordenador Personal	Varios vendedores/PCcomponentes	837,27 €
Total		1129,57 €

En cuanto a la mano de obra, el trabajo ha sido realizado por un ingeniero con un título en el Grado en Ingeniería en Electrónica de Comunicaciones. En la tabla 6.3 se muestra el desglose por horas trabajadas.

Tabla 6.3 Costes asociados a la mano de obra empleada

Concepto	Horas invertidas	Coste/hora	Coste total
Diseño e implementación	350	50€/h	17500€
Mecanografiado de este documento	100	15€/h	1500€
Total			19000€

Finalmente, en la tabla 6.4 se resume el coste total del trabajo realizado.

Tabla 6.4 Costes totales asociados a la realización del trabajo

Concepto	**Coste**
Material Software	0 €
Material Hardware	1129,57 €
Mano de obra	19000€
Honorarios por realización (7%)	1409,07€
Total sin IVA	21538,64€
IVA (21%)	4523,11€
Total	26061,75€

El coste total de la realización de este trabajo teniendo en cuenta los honorarios del Colegio de Ingenieros y el IVA asciende a 26061,75€.

8. Bibliografía

[1] Ignacio Carugati, Marcos Funes, Jesús Ureña, Patricio G. Donato, «Desagregación no intrusiva de consumos eléctricos en redes eléctricas inteligentes», Revista Ingeniería Eléctrica, no. 351, pp. 1-6, Marzo 2020.

[2] Ignacio Mártil de la Plaza (Abril 2021), «El reto de la energía: la transición hacia un nuevo modelo energético», BBVAOpenmind. [Online]. Available: https://www.bbvaopenmind.com/ciencia/medioambiente/el-reto-de-la-energia-la-transicion-hacia-un-nuevo-modelo-energetico/. [Último acceso: 15 Junio 2021].

[3] Diego Cocconi, Raúl Beinotti, Rebeca Yuan, Micaela Mulassano, Javier Bruno, Matías Beltramone, «Monitoreo de carga por métodos no invasivos en el hogar argentino utilizando redes neuronales» XX Workshop de Investigadores en Ciencias de la Computación, Universidad Nacional del Nordeste, Corrientes, Argentina, pp. 22-26, Abril 2018.

[4] José Alcalá, Jesús Ureña, Álvaro Hernández, Juan Jesús García, «Análisis no intrusivo de la actividad humana a través de la monitorización del consumo energético», Actas de las XXXVI Jornadas de Automática, Bilbao, pp. 731-736, Septiembre 2015.

[5] Francesca Paradiso, Federica Paganelli, Dino Giuli, Samuele Capobianco, «Context-Based Energy Disaggregation in Smart Homes», Future Internet 2016, vol. 8, pp. 1-22, Enero 2016.

[6] Ahmed Zoha, Alexander Gluhak, Muhammand Ali Imran, Sutharshan Rajasegarar, «Non-Intrusive Load Monitoring Approaches for Disaggregated Energy Sensing: A Survey», Sensors 2012, vol. 12, pp. 16838-16866, Diciembre 2012.

[7] Jorge Revuelta, Álvaro Lozano, Alberto López, Daniel Hernández de la Iglesia, Gabriel Villarrubia, Juan Manuel Corchado, Rita Carreira, «Non Intrusive Load Monitoring (NILM): A State of the Art» International Conference on Practical Applications of Agents and Multi-Agent Systems, pp. 125-138, Junio 2018.

[8] Jose Manuel Alcalá Orzáez, «Non-Intrusive Load Monitoring techniques for Activity of Daily Living recognition», Tesis doctoral, Escuela Politécnica Superior, Universidad de Alcalá, Alcalá de Henares, 2016.

[9] wiki.nilm.eu [Online]. Available: http://wiki.nilm.eu/datasets.html. [Último acceso: 20 Junio 2021].

[10] Robert Bielby (Octubre 2021), «Las Ventajas de los Dispositivos FPGA de la Serie 7 de Xilinx» [Online]. Available: https://www.ni.com/es-es/innovations/white-papers/13/advantages-of-xilinx-7-series-fpga-and-soc-devices.html. [Último acceso: 20 Junio 2021].

[11] Subbarao Lanka, Tushar Rastogi (30 Marzo 2014), «Designing with ARM Cortex-M based System-On-Chips (SoCs) – Part I: The basics» [Online]. Available: https://www.embedded.com/designing-with-arm-cortex-m-based-system-on-chips-socs-part-i-the-basics/. [Último acceso: 20 Junio 2021].

[12] Intel, «FPGA vs. GPU for Deep Learning» [Online]. Available: https://www.intel.es/content/www/es/es/artificial-intelligence/programmable/fpga-gpu.html. [Último acceso: 20 Junio 2021].

[13] Xilinx Inc, «FIFO Generator v13.2, LogiCORE IP Product Guide» 2017.

[14] T. Instruments, «DAC121S101 converter, Product Specification» 2015.

[15] A. Devices, «AD7476A converter, Product Especification» 2011.

[16] Xilinx Inc, “Vivado-Based Workshops”, Cursos y tutoriales en el uso de Vivado, SDK y la programación de tarjetas Zynq, [Online]. Available: https://www.xilinx.com/support/university/vivado/vivado-workshops.html. [Último acceso: Enero 2021].

Universidad de Alcalá