Trabajo Fin de Máster
Máster en Ingeniería Industrial

Desarrollo en plataforma FPSoC de estrategias FCS-MPC para el control de convertidores de potencia

Autor: Eduardo Zafra Ratia

Tutor: Sergio Vázquez Pérez

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Grupo de Tecnología Electrónica

Trabajo Fin de Máster
Máster en Ingeniería Industrial

Desarrollo en plataforma FPSoC de estrategias FCS-MPC para el control de convertidores de potencia

Autor:
Eduardo Zafra Ratia

Tutor:
Sergio Vázquez Pérez
Profesor Titular

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Máster: Desarrollo en plataforma FPSoC de estrategias FCS-MPC para el control de convertidores de potencia

Autor: Eduardo Zafra Ratia
Tutor: Sergio Vázquez Pérez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Me gustaría aprovechar estas líneas para agradecer a todas las personas que han hecho posible la realización de este Trabajo Fin de Máster que marca el final de mi etapa como estudiante de la Universidad de Sevilla.

Primero de todo, me gustaría agradecer a mi familia, en especial a mis padres, Fernando y María José, y a mi abuela María Dolores por su constante e incondicional apoyo todos estos años.

Por otro lado, agradecer a todos y cada uno de los miembros del Grupo de Tecnología Electrónica que han aportado su conocimiento y experiencia al desarrollo de este proyecto. En especial, a Sergio Vázquez Pérez, tutor de este proyecto y bajo cuya coordinación durante este casi ya año y medio de trabajo han sido innumerables las cosas que he podido aprender. También mencionar a otros miembros del grupo y/o departamento como Abraham Márquez, Hipólito Guzmán o Juan Antonio Sánchez, cuya ayuda ha sido también valiosa para sacar adelante este proyecto. No olvidarme tampoco de antiguos miembros como Francisco González o Guillermo Pérez con los que también he compartido incontables horas cacharreando en los laboratorios.

Por último, mencionar el apoyo de la administración pública por su financiación al proyecto "TEC2016-78430-R" bajo paraguas ha sido posible realizar este proyecto.

Eduardo Zafra Ratia
Grupo de Tecnología Electrónica

Sevilla, 2019

Resumen

La tendencia en la sociedad actual es la de un aumento persistente en la demanda de energía eléctrica, dentro de unos parámetros de calidad que las distintas normativas tienden a hacer cada vez más exigentes. Este hecho ha provocado una incesante búsqueda tecnológica de sistemas de producción energética que cada vez sean más eficientes y sostenibles.

En particular, el desarrollo de la electrónica de potencia ha sido crucial y ha protagonizado muchas de las mejoras tecnológicas en este ámbito, especialmente con la progresiva integración de las fuentes de energía renovable en la red eléctrica. Entre las numerosas líneas de investigación abiertas en este campo actualmente, se encuentran la adaptación de estrategias de control cada vez más avanzadas y complejas que procuren mejores resultados, o la transición de la tecnología del Silicio (Si) a otras tecnologías de semiconductores más recientes, como las de "Wide Bandgap". Uno de los principales lastres a la hora de permitir una adopción más rápida de todas estas tecnologías es el notable incremento que precisan en la capacidad de cálculo de la plataforma de control hardware que gestiona el sistema.

Es precisamente esta problemática la que se pretende abordar en este Trabajo Fin de Máster, mediante la implementación del control de un convertidor de potencia en una plataforma FPSoC ("Field Programmable System on Chip"). Básicamente, este tipo de plataformas se tratan de SoCs donde se combina la presencia de procesadores hardware con tejido FPGA, lo cual dota a estas plataformas de una flexibilidad y capacidades de cálculo extraordinarias. Para ello, se tomará como ejemplo de aplicación una estrategia de control de tipo FCS-MPC ("Finite Control Set - Model Predictive Control") para la regulación de las corrientes que un inversor trifásico de dos niveles proporciona a una carga trifásica.

En particular, el desarrollo de este trabajo se centrará en el codiseño HW/SW de una arquitectura eficiente para la plataforma "All Programmable Soc" Zynq-7000 de Xilinx, que ofrece la posibilidad de combinar la capacidad de procesamiento de dos núcleos ARM y de una FPGA Artix-7. Como se desarrollará en el resto del documento, la arquitectura propuesta para este Trabajo Fin de Máster combinará la ejecución de un sistema operativo embebido en uno de los núcleos, para aprovechar las soluciones de alto nivel que un OS puede ofrecer, con la ejecución de las tareas más críticas del control mediante aplicaciones "baremetal" en el segundo núcleo. Esto elimina la incertidumbre que un sistema operativo puede introducir en tareas software que se tienen que ejecutar en tiempo real. Al mismo tiempo, esta arquitectura hará uso de la FPGA para el grueso de los cálculos del algoritmo de control, explotando las grandes capacidades de cálculo en paralelo propio de este tipo de plataformas, y aprovechando al máximo los recursos del sistema. Además, el estudio de la implementación propuesta incluirá una disquisición sobre posibles alternativas de diseño en la parte FPGA y sus efectos en la velocidad de cálculo y la ocupación de área y recursos en la misma.

En resumen, el objetivo principal de este Trabajo Fin de Máster será dejar patente la viabilidad de este tipo de plataformas FPSoC para protagonizar el control de un convertidor de potencia, pudiendo convertirse en catalizadoras para la adopción de técnicas de control más ambiciosas y complejas, así como nuevas tecnologías de semiconductores, al permitir tiempos de ejecución muy reducidos. Se realizará una descripción detallada de todas las soluciones adoptadas desde la fase de diseño hasta la puesta en marcha de los equipos, que permita establecer una base de cara a futuros trabajos y a la extensión de este tipo de plataformas en este ámbito.

Abstract

The demand of electric energy in nowadays society is constantly increasing, while the power delivery quality restrictions also become stricter. These facts are the main cause for the never ending search of new energy generation systems and methods with increasing efficiency and sustainability.

Particularly, the development of Power Electronics has been a crucial fact to understand all the successful technological advances in this field, specially with the progressive renewable energy grid integration. Among all the ongoing researching efforts, new and more complex control strategies are being proposed in order to improve the quality of the obtained results. Besides, the appearance of "Wide Bandgap" semiconductors based on materials like Silicon Carbide (SiC) or Gallium Nitride (GaN) are allowing the transition from Silicon-based semiconductors with promising results. Nonetheless, the computational burden that the introduction of these new technologies represent is hampering their adoption as industry standards.

Proposing a solution to this problem is precisely the aim of this work, with the implementation of the control of a power converter in a FPSoC ("Field Programmable System on Chip") platform. Basically, these kind of platforms combine the calculation power of FPGA fabric with the high level solutions of hard processing systems. As a particular example, a Finite Control Set Model Predictive Control strategy for the current reference tracking of a two-level three-phase power converter is chosen for this implementation.

More specifically, an efficient HW/SW architecture is designed for the Zynq-7000 "All programmable SoC" by Xilinx. In short, this platform is composed of two ARM hard cores and an Artix-7 FPGA. As it will be described in this work, the proposed architecture will take advantage of the high level solutions provided by an embedded operating system, running in one of the cores. The other core will run a "baremetal" application to address critical tasks that require real time execution. Also, the FPGA will be used for the most complex calculations in the control algorithm in order to fully harness its great computational capabilities and fully exploit all the resources in the system. In addition, different alternatives for the FPGA implementation will be analysed and discussed when it comes to the speed-area trade-off.

In summary, the main goal of this work is to demonstrate the viability of FPSoC platforms to implement the control of a power converter, acting as catalysts for the adoption of new complex control algorithms and fast-switching semiconductor technologies with its reduced execution times. Also, a detailed description of all the solutions adopted in this work will be offered, with the goal of establishing a base for future works with these platforms.

Índice

Resumen III
Abstract V
Notación y Acrónimos IX

1 Introducción 1
1.1 Antecedentes 1
1.2 Breve resumen del proyecto 4

2 Modelo matemático del sistema y estrategia de control 5
2.1 Introducción al Control Predictivo orientado a aplicaciones de electrónica de potencia 5
2.2 Modelo matemático del sistema 6
2.3 Algoritmo de control 9

3 Plataforma ZYNQ-7000 y Kit de Evaluación Zedboard 11
3.1 All-Programmable SoC Zynq-7000 11
3.1.1 Processing System 12
3.1.2 Programmable Logic (Artix-7) 15
Configurable Logic Blocks (CLB) 16
DSP blocks 17
RAM blocks 18
3.2 Placa de evaluación Zedboard 20

4 Convertidor Analógico Digital ADS7953 25
4.1 Sensores y etapa de adaptación 25
4.1.1 Transductores LEM de tensión y corriente 25
4.1.2 Etapa de adaptación 27
4.2 Convertidor Analógico Digital ADS7953 27
4.3 Kit de evaluación ADS7953EVM-PDK 31

5 Implementación del control en plataforma Zynq-7000 35
5.1 Vivado Design Suite 35
5.1.1 Programación hardware mediante Vivado 35
Definición del sistema procesador. Block Design 37
Creación de un top level. Generación del "bit stream" 38
5.1.2 Programación software mediante Xilinx SDK 39
Hardware Platform. Programación de la FPGA 39
Board Support Packages. Drivers y librerías 39
Creación y ejecución de una aplicación 40
5.2 Arquitectura básica propuesta para el FPSoC 40
5.3 Petalinux OS 42
5.4 Descripción detallada del diseño 48

5.4.1 Programación de la parte hardware I. Estructura básica y bloques auxiliares 49
Configuración de la parte PS. Creación del "ARM wrapper" 50
Máquina de estados (FSM) 52
Contador 55
Disparos 55
Bloque de filtrado de Rebotes 55
Bloque de detección de flancos 55
Bloque Interrupción 55
5.4.2 Programación de la parte hardware II. Bloques del algoritmo de control 56
Aritmética de punto fijo para los cálculos 56
Bloque de BRAM para senos y cosenos 58
Bloque de transformadas 59
Predicción a k+1 60
5.4.3 Programación de la parte hardware III. Alternativas para la predicción a k+2 61
Metodología en paralelo 61
Metodología secuencial 62
5.4.4 Programación de la parte hardware IV. Top Level y constraints 63
5.4.5 Programación de la aplicación baremetal 64
Función principal "main" 64
Rutina de Interrupción 66
5.4.6 Programación de la aplicación de Linux 67
5.5 Interfaz gráfica 67

6 Resultados experimentales y análisis de los resultados 71
6.1 Resultados experimentales 71
6.1.1 Equipos de laboratorio 71
6.1.2 Resultados de los experimentos 73
6.2 Análisis de consumo de recursos y tiempo de cálculo 75

7 Conclusiones y Trabajo Futuro 79

Índice de Figuras 81
Índice de Tablas 83
Índice de Códigos 85
Bibliografía 87

Notación y Acrónimos

Si	Silicio
Sic	Carburo de Silicio
GaN	Nitruro de Galio
FPSoC	Field Programmable System on Chip
SoC	System on Chip
FPGA	Field Programmable Gate Array
DC/AC	Conversión de corriente continua a alterna
VSI	Voltage Source Inverter
PWM	Pulse Width Modulation
DSP	Digital Signal Processors
HDL	Hardware Description Language
HLS	High-Level-Synthesis
ADC	Convertidor Analógico - Digital
FCS-MPC	Finite Control Set - Model Predictive Control
$\mathbf{v} \in \mathbb{R}^n$	Vector **v** pertenece al espacio vectorial *n*
I_{nxn}	Matriz identidad de dimensión *nxn*
ESA	Exhaustive Search Algorithm
$\|\mathbf{v}\|_2$	Norma euclídea del vector **v**
$\mathbf{v}^*$	Referencia de la magnitud **v**
PS	Processing System / Parte software del FPSoC
PL	Programmable Logic / Parte hardware del FPSoC
GIC	Generic Interrupt Controller
OCM	On-Chip Memory
DMA	Direct Memory Access
SPI	Serial Peripheral Interface
SS/CS	Slave Select / Chip Select. Señal de selección de esclavo
MIO	Multiplexed Input/Output
EMIO	External Multiplexed Input/Output
CLB	Configurable Logic Block
LUT	Look-up Table
CLK	Señal de reloj
SAR	ADC de Aproximaciones Sucesivas
MOSI	Master Out - Slave In
MISO	Master In - Slave Out
SDO	Salida Digital del ADC
SDI	Entrada Digital del ADC
RTL	Register-Transfer Level
VHDL	Very-High-Speed-Integrated-Circuit Hardware Description Language
BSP	Board Support Package
AMP	Asymmetric Multiprocessing

FSBL	First Stage Bootloader
GUI	Graphical User Interface
UPS	Uninterruptible Power Supply

1 Introducción

Se dedicará este apartado introductorio a un repaso de los antecedentes y del estado del arte en el ámbito de la electrónica de potencia y del control de sistemas industriales. Este repaso servirá de justificación del presente proyecto, de cuyos puntos más destacados se ofrecerá también un breve resumen.

1.1 Antecedentes

Todos los procesos y técnicas que tienen lugar en la generación, transporte y transformación de energía eléctrica están sujetos a un proceso de mejora continua que permita mejorar la eficiencia de estos. La electrónica de potencia constituye una de las ramas de conocimiento clave cuyo desarrollo ha protagonizado muchos de los más recientes avances tecnológicos en este campo.

La electrónica de potencia se encarga del estudio de todas aquellas aplicaciones que tienen como objetivo la transformación y/o la adaptación de la forma en que se presenta la energía eléctrica, de manera controlada, fiable y eficiente. Los avances en esta tecnología han permitido su utilización en numerosas aplicaciones como las fuentes de alimentación, el control de motores eléctricos, la integración de las energías renovables en el sistema eléctrico o el control del flujo de potencia en la red (dispositivos FACTS). [1, 2, 3, 4, 5].

De igual forma, la presencia de la electrónica de potencia se antoja imprescindible en el desarrollo de nuevas aplicaciones como son los vehículos eléctricos (VE), los sistemas de almacenamiento de energía eléctrica (ESS), los sistemas de transmisión de corriente continua en alta tensión (HVDC) o las redes inteligentes (Smart grids) [6, 7, 8, 9].

Todas las tecnologías mencionadas hasta ahora cubren en algún punto sus necesidades de adaptación de la energía eléctrica mediante el uso de un convertidor de potencia, en cualquiera de sus diferentes formas o tipologías. En particular, se centrará el desarrollo de este Trabajo Fin de Máster en los convertidores de potencia DC/AC (Figura 1.1), concretamente en los inversores de fuente de tensión (VSI, del inglés "Voltage Source Inverter"), en su tipología trifásica de dos niveles. Este tipo de convertidores permiten la transformación de la energía eléctrica ofrecida por una fuente de continua a alterna. Es frecuente ver este tipo de conversión en muchas de las aplicaciones citadas anteriormente. Por ejemplo, para el aprovechamiento de la energía eléctrica generada en paneles solares en forma de corriente continua, y su inyección en alterna a la red, o para el del uso de la energía almacenada en baterías. Por tanto, la utilización de estos convertidores está muy extendida en la industria, especialmente para aplicaciones de baja potencia, debido a su sencillez y robustez, frente a otras tipologías más complejas, como los multiniveles, que encuentran su campo de aplicación en potencias más elevadas [10].

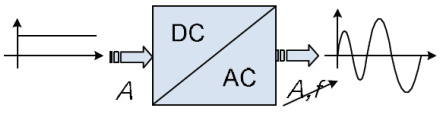

Figura 1.1 Convertidor DC/AC [10].

En cuanto al esquema de control de este tipo de convertidores, lo más habitual es encontrar controles de lazo cerrado (frecuentemente controladores de tipo proporcional-integral) en conjunción a técnicas de modulación como pueden ser la modulación por anchura de pulsos (PWM, del inglés "Pulse Width Modulation") o la

modulación Space Vector [11, 12]. Como se puede apreciar en la Figura 1.2, la metodología habitual de estas técnicas se basa en un controlador que genera los valores de referencia que debe de seguir el convertidor para cumplir el objetivo de control requerido (valor de corrientes, potencias o balanceo de las tensiones DC a la entrada, entre otros). Por otro lado, el modulador toma estos valores de referencia calculados y se encarga de generar los pulsos requeridos para los dispositivos de potencia que permitan generar, en valor medio dentro del intervalo de control, el valor de referencia calculado a la salida del convertidor.

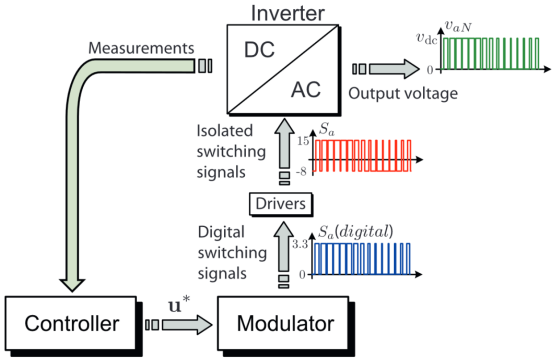

Figura 1.2 Esquema básico de control y modulación de un convertidor DC/AC [12].

En años recientes, existe una tendencia cada vez más notable de aplicar otras técnicas desarrolladas por la Teoría de Control para el control de convertidores de potencia. En particular, el Control Predictivo basado en Modelo (MPC, del inglés "Model Predictive Control") está demostrando ser un campo fértil a la hora de producir numerosos resultados en el campo de la investigación para este tipo de aplicaciones [13, 14]. Entre sus principales ventajas se encuentra su rápida respuesta dinámica y la flexibilidad de este método de control para ser adaptado a distintos tipos de sistemas de mayor o menor complejidad. La adición de restricciones, no-linealidades y objetivos de control múltiples pueden ser también resueltas de manera directa mediante MPC.

El funcionamiento básico de este tipo de control se basa en la realización de predicciones para futuros estados de las variables del sistema, en base a un modelo matemático representativo de éste. El horizonte de predicción define el número de estados futuros que es necesario calcular. Con los valores de estas predicciones, se puede minimizar una función de coste, que de manera genérica se define como el error entre alguna de estas variables en un instante futuro y la referencia a seguir. Es precisamente esta necesidad de calcular predicciones para las variables del sistema dentro del intervalo de control la que origina el principal inconveniente de estas técnicas de control: su coste computacional.

Todo algoritmo de control se basa en la realización de una serie de operaciones, cuya ejecución en un microprocesador requiere un tiempo determinado. Si el sistema al que se aplica el MPC crece en complejidad, o se aumenta el horizonte de predicción, el número de operaciones requeridas crece, y por tanto el coste computacional puede llegar a ser demasiado grande para ser ejecutado en un tiempo razonable para un control en tiempo real de un sistema. Este hecho supone un lastre importante en la adaptación de estas técnicas en el ámbito de la electrónica de potencia de una manera realmente extendida. Esto se debe a la relación directa que existe entre el contenido armónico de las tensiones y corrientes de salida en un convertidor de potencia y su frecuencia de conmutación. Aumentar la frecuencia de conmutación no sólo disminuye el contenido armónico total si no que desplaza buena parte del espectro hacia frecuencias más altas. Esto facilita el cumplimiento de la normativa en términos de contenido armónico, y permite que el filtro de salida sea de menor tamaño y coste. Un coste computacional del algoritmo de control muy alto limita la frecuencia con la que se puede ejecutar el mismo, y por tanto la frecuencia a la que se conmutan los dispositivos de potencia.

Adicionalmente, es cada vez mayor el interés investigador en las nuevas tecnologías de semiconductores que permite mantener las pérdidas energéticas frente al incremento de la frecuencia de conmutación. Mientras que la tecnología del Silicio (Si), en la que se han basado la gran mayoría de los semiconductores de potencia, empieza a dar muestras de agotamiento en la actualidad, surgen nuevas tecnologías basadas en materiales "Wide Bandgap" como son el Carburo de Silicio (SiC) o el Nitruro de Galio (GaN) [15, 16, 17]. Desde el punto de vista del material, la principal diferencia entre Si y los materiales como SiC o GaN es que estos últimos presentan una banda prohibida más ancha (Figura 1.3). Esto se traduce en que las pérdidas por conmutación en los dispositivos "Wide Bandgap" tiene un aumento mucho más contenido al aumentar la frecuencia de conmutación del semiconductor.

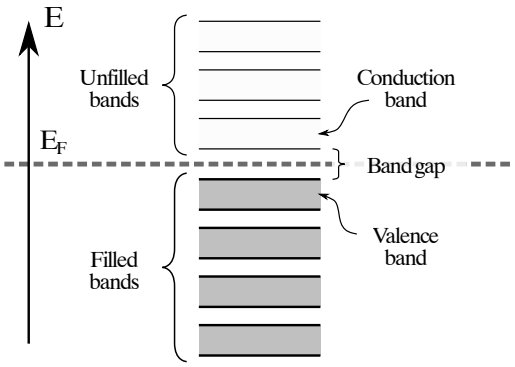

Figura 1.3 Bandas de un material semiconductor [18].

En la práctica, IGBTs basados en Si empiezan a tener unas pérdidas muy notables más allá de los 20 kHz de frecuencia de conmutación, mientras que en el caso de semiconductores basados en materiales "Wide Bandgap" se están desarrollando aplicaciones con frecuencias de conmutación en los centenares de kiloherzios [15, 19]. Efectivamente, esto implica que para el máximo aprovechamiento de estos dispositivos es necesario que la plataforma hardware de control sea capaz de ejecutar el algoritmo de control en un espacio de tiempo muy reducido. Concretamente, con los dispositivos "Wide Bandgap" existentes en el mercado que pueden llegar a trabajar en torno al megahercio de frecuencia de conmutación [20], se estaría rondando un orden de unos pocos microsegundos, o incluso menos, para poder realizar todas estas operaciones de control y obtener los pulsos a aplicar en el siguiente intervalo de control. Esto se antoja una tarea difícilmente abordable mediante el uso de plataformas hardware de control convencionales, como los DSPs. Aún más si la complejidad computacional del algoritmo a calcular es elevada, como se ha comentado para el caso del Control Predictivo.

Todo esto apunta a una misma dirección: la necesidad de explorar nuevas plataformas digitales de control que ofrezcan capacidades de cálculo que puedan afrontar el reto que suponen los nuevos avances tecnológicos en el campo de la electrónica de potencia. Posee especial interés el papel que las FPGA ("Field Programmable Gate Array") están teniendo en este ámbito, con publicaciones y aplicaciones propuestas a lo largo de los años, de cara a demostrar la viabilidad de estos sistemas como plataformas de control fiables y adecuadas para el control de convertidores de potencia [21, 22, 23, 24]. Más allá de estos esfuerzos llevados a cabo, la realidad es que la norma en el ámbito de la electrónica de potencia sigue siendo la de utilizar plataformas de tipo DSP en la gran mayoría de aplicaciones [25]. Esto se debe básicamente al bajo coste y bajo consumo de estas plataformas, así como al conocimiento extendido de los métodos de programación para este tipo de dispostivos, que habitualmente son programados en lenguajes de programación software estructurados como C o similar a éste. De igual manera, el extendido uso de estas plataformas ha permitido a la industria ofrecer productos cada vez más especializados para ciertos rangos de aplicaciones determinados, con la inclusión de periféricos diseñados para estas aplicaciones.

En este sentido, uno de los hechos que más han lastrado la adopción de las plataformas FPGA para protagonizar el control de convertidores de potencia es la poca familiarización por parte de los programadores en este ámbito de los lenguajes de descripción hardware (HDL, del inglés "Hardware Description Language") [26]. En general, no es trivial trasladar algunas de las soluciones más conocidas y aplicadas para DSPs a una FPGA.

Con el objetivo de abordar esta problemática, está siendo de bastante interés el desarrollo en los últimos años de nuevas herramientas como las "High-Level-Synthesis" o HLS. Éstas permiten la compilación de lenguajes de programación de alto nivel como podría ser código C, en código HDL. Este hecho ha convertido a las FPGA en plataformas más accesibles para muchos programadores, facilitando su uso en este tipo de aplicaciones [27, 28]. La posibilidad de trabajar en un nivel de abstracción mayor, así como la posibilidad de adaptar soluciones ya conocidas de manera más rápida, es un factor también crucial a la hora de reducir el "time to market" (TTM) del proyecto. Esto ha permitido facilitar la adopción de soluciones de programación hardware basadas en FPGA en algunos casos, frente a otras soluciones más convencionales. Como contrapartida, es habitual observar un incremento notable del consumo de recursos de la FPGA en diseños realizados mediante herramientas HLS, frente a diseños implementados directamente mediante HDL, lo cual indica que es todavía un campo en crecimiento y que sigue siendo objeto de estudio de cara a sus sucesivas mejoras [29, 30, 31].

Por otro lado, está siendo de especial relevancia la aparición de los sistemas FPSoC ("Field Programmable

System-On-Chip"). Estas plataformas combinan en un mismo SoC ("System-On-Chip") uno o más núcleos procesadores con tejido FPGA. Esto les permite combinar las potentes soluciones de alto nivel que se pueden desarrollar en lenguajes de programación más extendidos para sistemas procesadores con la potencia de cálculo de las plataformas FPGA [32, 33]. En particular, al incluir todos estos elementos en un mismo SoC, permiten ofrecer soluciones fiables, de altas prestaciones y bajo consumo para la interconexión y comunicación de los mismos. De igual forma, estas plataformas pueden ofrecer una gama de periféricos comparables a las de los sistemas DSP. Sin embargo, la presencia de todos estos recursos y la flexibilidad que ofrecen estas plataformas viene acompañada de una mayor complejidad para el diseñador, que ahora tiene que enfrentarse a la programación de una FPGA, de uno o más microprocesadores y a un codiseño HW/SW que tiene que solucionar la interconexión y la comunicación de todos estos elementos.

Por todo ello, las plataformas FPSoC se presentan como una alternativa muy interesante, todavía por explorar y que requieren una labor de investigación importante de cara a su implantación extendida para este tipo de aplicaciones.

1.2 Breve resumen del proyecto

Considerando todos estos antecedentes, se planifica la realización de este Trabajo Fin de Máster con el objetivo de desarrollar la base necesaria para implementar el control de convertidores de potencia en una plataforma FPSoC actual. Concretamente, todo lo desarrollado en este trabajo estará destinado a su implementación en un "All Programmable Soc" Zynq 7000 de Xilinx [34], que cuenta con una FPGA Artix-7 y dos núcleos procesadores ARM. En concreto se utilizará el kit de desarrollo "Zedboard" [35] del fabricante Avnet.

La arquitectura básica que se planteará en este proyecto será la de utilizar la FPGA para el cálculo de los algoritmos de control, absorbiendo la mayor carga computacional del sistema, mientras que los núcleos ARM se encargan de tareas como la lectura de los ADC o las comunicaciones. En particular, se utilizará uno de ellos para ejecutar un sistema operativo basado en Linux que resuelva de manera más simple todo lo relacionado con las comunicaciones hacia el exterior. Esto permitirá la monitorización del sistema desde una interfaz gráfica ejecutada en un PC. Por otro lado, el otro núcleo ARM se encargará de ejecutar las tareas críticas con una aplicación "baremetal" que gestione las medidas de los sensores a través del ADC y proporcione esta información a la FPGA para que realice los cálculos del algoritmo de control.

En cuanto a las medidas, el integrado Zynq-7000 incluye un ADC de 12 bits y 1 MSPS, pero el fabricante Avnet no tiene todos los canales de medida accesibles en su placa de evaluación Zedboard. Debido a esto, la toma de medidas del sistema real se llevará a cabo mediante un ADC externo de Texas Instruments: el ADS 7953, cuyo funcionamiento también se estudiará en detalle a lo largo de este trabajo.

La idea básica trata de, una vez definido matemáticamente el control a aplicar, llevar a cabo un repaso de las características básicas de la plataforma digital elegida y efectuar una descripción de la solución adoptada, desde las configuraciones básicas y la fase de diseño de los distintos elementos hasta su puesta en marcha en laboratorio. Finalmente, la viabilidad y el funcionamiento de la solución propuesta será verificada mediante las pruebas experimentales realizadas en el laboratorio. De igual manera, se ofrecerá un análisis del consumo de los recursos de la FPGA que permita discernir las diferencias entre los diseños propuestos.

2 Modelo matemático del sistema y estrategia de control

Este capítulo está destinado al establecimiento de la base matemática y de la teoría de control que son necesarias para la descripción del control implementado en este trabajo. Tal y como se ha especificado en la introducción de este documento, se ha optado por un un Control Predictivo basado en Modelo (MPC). Básicamente, la idea es aprovechar las ventajas que ofrecen estas técnicas de control de creciente popularidad en el ámbito de la electrónica de potencia, al mismo tiempo que se pueda demostrar la capacidad de cálculo de la plataforma elegida al implementar un tipo de control con un coste computacional elevado.

2.1 Introducción al Control Predictivo orientado a aplicaciones de electrónica de potencia

Son numerosas las técnicas del Control Predictivo que han sido exploradas y aplicadas para el control de dispositivos de potencia. Sus ventajas desde el punto de vista del control son numerosas, al presentar una respuesta dinámica rápida y al facilitar la incorporación al modelo de restricciones, no-linealidades y múltiples objetivos de control [36]. Todo ello, ha permitido el desarrollo de aplicaciones en los últimos años dentro del campo de la electrónica de potencia, entre las que se encuentran el control de convertidores de potencia, sistemas de alimentación ininterrumpida (SAI) o máquinas eléctricas [37, 14, 38].

De manera básica, un controlador MPC se basa en el modelo matemático del sistema a controlar, mediante el cual realiza predicciones de la evolución de las disintas variables del sistema en los posibles futuros estados. Conocidas estas plausibles evoluciones del sistema, se puede minimizar una función de coste que fuerce al sistema a describir el comportamiento requerido, por ejemplo, seguir un valor de referencia [13]. Conociendo los estados óptimos futuros, se puede conocer la acción de control óptima que se ha de aplicar en el siguiente intervalo de control para que el sistema tienda a esa situación óptima.

Las principales técnicas MPC aplicadas en este ámbito se pueden clasificar en dos grandes categorías: "Continuous Control Set MPC" (CCS-MPC) o "Finite Control Set MPC" (FCS-MPC). La primera de ellas se basa en el cómputo de una señal de control continua que es alimentada a un modulador que genera los pulsos necesarios a una frecuencia de conmutación fija. La segunda categoría se aprovecha de la naturaleza discreta de un convertidor de potencia, que tiene un número conocido de estados posibles que depende de su tipología (por ejemplo un inversor trifásico de dos niveles tendrá 8 estados posibles) y que definen los niveles de tensión generada a la salida del convertidor. En general, la formulación matemática de las técnicas CCS-MPC es más compleja, aunque habitualmente suelen presentar un coste computacional menor al poder hacer ciertos cálculos "offline". Por otro lado, las técnicas FCS-MPC pueden prescindir del modulador, al trabajar directamente con los posibles estados de conmutación del convertidor como señal de control [13].

Con esta información, se decide optar por una estrategia de control FCS-MPC, debido a que su formulación más intuitiva permite mantener el foco de este trabajo en el desarrollo de la plataforma hardware de control, más que en la formulación matemática del mismo. Especialmente teniendo en cuenta que computacionalmente las técnicas FCS-MPC tienen un coste más alto que las CCS-MPC.

Dentro de las técnicas FCS-MPC, se optará por un tipo de control "Optimal Switching Vector MPC" (OSV-MPC). Esta técnica de control tiene como señal de control los posibles estados de conmutación del convertidor de forma directa. Por contra, están las técnicas "Optimal Switching Sequence MPC" (OSS-MPC)

que consideran la variable temporal, al tomar como señal de control una secuencia de estados. Esto les permite fijar la frecuencia de conmutación de los dispositivos de potencia, lo cual no ocurre necesariamente en OSV-MPC [13]. Se opta por una estrategia OSV-MPC al poseer una formulación menos compleja.

2.2 Modelo matemático del sistema

Como se ha descrito en la sección anterior, una estrategia de control MPC requiere un modelo matemático del sistema en base al cual se puedan realizar predicciones futuras de las variables de estado del sistema.

En este caso, el elemento principal del sistema se trata de un convertidor de potencia. En particular, y dado su disponibilidad en el laboratorio, de cara a obtener resultados experimentales se tomará un inversor trifásico de dos niveles como convertidor a controlar. Un esquema básico del circuito electrónico del mismo puede apreciarse en la Figura 2.1.

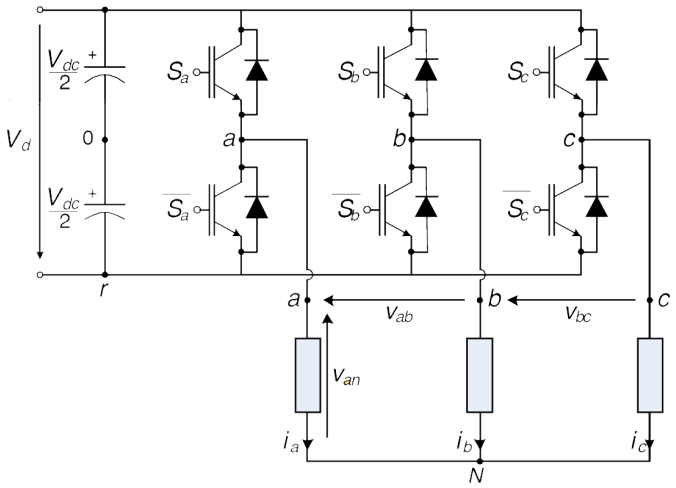

Figura 2.1 Circuito de un inversor trifásico de dos niveles [39].

Como carga se utilizará una carga resistiva-inductiva presente en el laboratorio, por lo que se modela el sistema con una carga RL. Este circuito puede resumirse en el esquema de la Figura 2.2, con la notación que se utilizará en los desarrollos matemáticos posteriores. Para mayor detalle, se puede consultar la Tabla 2.1 para conocer las variables y parámetros que definen el sistema.

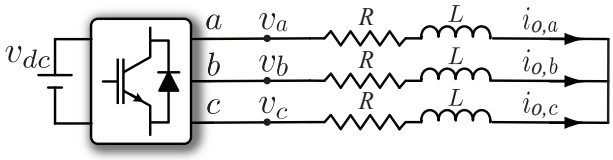

Figura 2.2 Circuito de un inversor trifásico de dos niveles con carga RL. Los valores de tensión se miden respecto al neutro (N) de la carga.

Como se puede observar, para modelar el sistema se utilizará la representación en ejes naturales *abc*, la cual se trata de una representación vectorial de tres componentes donde a cada una de ellas se le asigna el valor instantáneo de las tres tensiones (a, b y c) del sistema. Posteriormente, estas variables representadas en ejes *abc* pueden ser transformadas a ejes estacionarios $\alpha\beta$ mediante la aplicación de la transformada de Clarke y posteriormente a ejes síncronos *dq* mediante la transformada de Park, proceso que facilita en general la formulación matemática de las técnicas de control al reducir las variables del problema mediante la transformación de magnitudes tridimensionales a magnitudes contenidas en un plano.

Atendiendo a los posibles estados de los semiconductores de potencia, en función de las señales de disparo (S_a, S_b y S_c), se puede deducir que existen ocho posibles estados en el sistema, en lo que respecta a la tensión trifásica de salida generada, de los cuales sólo siete son diferentes. Conocida la tensión de continua en el dc-link, es posible conocer el valor de tensión instantáneo generado por el convertidor en cada una de las tres

Tabla 2.1 Variables y parámetros del sistema.

Variable	Descripción
$v_{abc} = [v_a\ v_b\ v_c]^T$	Vector de tensión de salida en coordenadas naturales *abc*
$i_{o,abc} = [i_{o,a}\ i_{o,b}\ i_{o,c}]^T$	Vector de corriente de salida en coordenadas naturales *abc*
L	Inductancia de la carga
R	Resistencia de la carga
v_{dc}	Tensión del dc-link
w	Frecuencia angular de sincronización del sistema

fases, a partir de los pulsos aplicados por el mismo. De esta forma, pueden ser precalculados en función del valor de v_{dc} y recogidos en la Tabla 2.2. Se incluyen los valores en ejes naturales *abc* y ejes estacionarios $\alpha\beta$. La naturaleza síncrona de los ejes dq impide su cálculo "offline". Es necesario conocer el ángulo de sincronización en cada instante y transformar los valores de $\alpha\beta$ a ejes dq.

Tabla 2.2 Valores de tensión de salida del convertidor en función del estado.

Estado	S_a,S_b,S_c	v_a/v_{dc}	v_b/v_{dc}	v_c/v_{dc}	v_α/v_{dc}	v_β/v_{dc}
S0	0,0,0	0	0	0	0	0
S1	1,0,0	$2/3$	$-1/3$	$-1/3$	0.8164966	0
S2	1,1,0	$1/3$	$1/3$	$-2/3$	0.4082483	0.70710678
S3	0,1,0	$-1/3$	$2/3$	$-1/3$	-0.4082483	0.70710678
S4	0,1,1	$-2/3$	$1/3$	$1/3$	-0.8164966	0
S5	0,0,1	$-1/3$	$-1/3$	$2/3$	-0.4082483	-0.70710678
S6	1,0,1	$1/3$	$-2/3$	$1/3$	0.4082483	-0.70710678
S7	1,1,1	0	0	0	0	0

Conociendo los valores de tensión de salida que genera el inversor, se puede proceder a extraer la ecuación que describe la dinámica del sistema, aplicando las leyes de Kirchoff al circuito y aplicando la ecuación dinámica de una bobina elemental:

$$v_{abc} = L\frac{di_{o,abc}}{dt} + R\,i_{o,abc} \tag{2.1}$$

donde v_{abc}, $i_{o,abc} \in \mathbb{R}^3$. Pudiendo conocer v_{abc} a partir de los valores de la Tabla 2.2.

Aplicando las transformadas de Clarke y de Park antes mencionadas, se puede obtener la representación en ejes síncronos dq del modelo del sistema, lo que arroja la siguiente ecuación:

$$v_{dq} = L\frac{di_{o,dq}}{dt} + J\,\omega L\,i_{o,dq} + R\,i_{o,dq} \tag{2.2}$$

donde $J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ y v_{dq}, $i_{o,dq} \in \mathbb{R}^2$, ya que ahora las tensiones y corrientes del sistema son magnitudes que se mueven en el plano. En particular, al elegir la representación en ejes dq del sistema, serán cantidades de continua, siendo el marco de referencia el que gira en función del ángulo de sincronización.

Es particularmente útil a la hora de formular un control de tipo MPC obtener la representación en espacio de estados del sistema, quedando la expresión matemática del mismo resumida en:

$$\frac{dx}{dt} = Ax + Bu \tag{2.3}$$

$$y = x \tag{2.4}$$

donde $x = i_{o,dq}$; $u = v_{dq}$; $A = -\left[\frac{R}{L}I_{2x2} + J\omega\right]$; $B = \frac{1}{L}I_{2x2}$. I_{2x2} es la matriz identidad de dimensión $2x2$.

Hasta este punto, se ha obtenido el modelo de tiempo continuo del sistema. Por la naturaleza del algoritmo de control digital empleado y su ejecución en sucesivos intervalos de control, es necesario discretizar el modelo matemático del mismo para un tiempo de sampleo (T_s) determinado. El modelo discretizado del sistema se puede expresar de la siguiente manera:

$$x(k+1) = A_d x(k) + B_d u(k) \tag{2.5}$$

$$y(k) = x(k) \tag{2.6}$$

donde $A_d = e^{AT_s}$ y $B_d = \int_0^{T_s} e^{A\tau} B d\tau$. Si A_d es invertible, la integral se puede resolver, quedando la expresión simplificada para estas matrices como:

$$A_d = e^{AT_s} \tag{2.7}$$

$$B_d = A^{-1}(e^{AT_s} - I)B \tag{2.8}$$

Estas operaciones pueden ser resueltas analíticamente o mediante métodos numéricos. Estos últimos son especialmente útiles ya que habitualmente la resolución analítica exacta de éstas ecuaciones se hace difícilmente abordable al aumentar la complejidad del sistema. Existen aproximaciones utilizadas con frecuencia en la literatura como el método de Euler, que utiliza la aproximación: $e^{AT_s} = (I - AT_s)^{-1}$, o el método de Tustin: $e^{AT_s} = (I + 1/2AT_s)(I - 1/2AT_s)^{-1}$, ambos con rangos de aplicación más o menos apropiados [40].

Para aplicar estos métodos numéricos, se puede utilizar una herramienta software como MATLAB. En particular, la función de MATLAB "c2d" permite la conversión de un modelo de tiempo continuo a discreto utilizando alguno de estos métodos de forma sencilla [41]. Un script de MATLAB para abordar la discretización del sistema se presenta en el Código 2.1. En este caso, se utiliza el método de discretización "zero-order hold" (ZOH), el cual asume que las magnitudes se mantienen constantes durante el periodo, lo cual es una aproximación apropiada para el modelo en ejes *dq* utilizado.

Código 2.1 Discretización de un modelo de espacio de estados en MATLAB.

```
function [a] = parametros(b)

% Parametros
Ts = b(1,1)
L = b(1,2)
R = b(1,3)
w = 2*pi*50

% Sistema
I = eye(2,2);
0 = zeros(2,2);
J = [0 -1; 1 0];

A = (-(R/L)*I)-w*J;
B = 1/L*I;
C = I;
D = 0;
Gsys = ss(A,B,C,D);
Gsys_z = c2d(Gsys, Ts);

% Salida de datos
k = 1
[Nf, Nc] = size(Gsys_z.a);
for i = 1:Nf
  for m = 1:Nc
```

```
    a(k,1) = Gsys_z.a(i,m);
    k = k+1
  end
end

[Nf, Nc] = size(Gsys_z.b);
for i = 1:Nf
  for m = 1:Nc
    a(k,1) = Gsys_z.b(i,m);
    k = k+1
  end
end
```

Ejecutando este script desde MATLAB, con los parámetros del sistema como entradas, se pueden obtener las matrices del sistema discreto, que serán las utilizadas al implementar el control digital.

2.3 Algoritmo de control

A partir de la representación en espacio de estados del modelo matemático del sistema, se puede aplicar el algoritmo FCS-MPC en cada intervalo de control. Una vez las medidas hayan sido obtenidas y transformadas a ejes síncronos dq, la tensión que está generando el convertidor a su salida puede ser conocida a través de los valores de la Tabla 2.2, ya que la tensión del dc-link es conocida y el estado aplicado actual es conocido. Con los valores actuales de todas las variables de estado del sistema, una predicción para el siguiente intervalo de control $(k+1)$ es calculada. Esta predicción se realiza para compensar el retraso digital [42].

A partir de este punto, y con el valor de las variables de estado en $k+1$, se realiza un Algoritmo de Búsqueda Exhaustiva (ESA, del inglés "Exhaustive Search Algorithm") el cual comprueba uno por uno todos los posibles estados en $k+2$ (Tabla 2.2). Para cada uno de estos estados, se obtienen los valores que las variables de estado alcanzarían, según el modelo del sistema, de optar por ese estado en el siguiente intervalo de control. En base a ellas, se procede a evaluar una función de coste que permita determinar la elección del estado óptimo. Existen en la literatura numerosas formulaciones para la función de coste [36]. En este caso, como el objetivo es la regulación de las corrientes en una carga conectada a la salida del convertidor, se formulará la función de coste como el error entre el valor de corriente que existiría de tomar el estado candidato y el valor de referencia de la corriente que corresponde.

Adicionalmente, se añade un término limitante de las conmutaciones, cuyo efecto sirve para penalizar con un mayor peso aquellos estados que supondrían conmutar un mayor número de fases. Esto puede tener la utilidad de limitar la frecuencia a la que conmutan los semiconductores de potencia (cada fabricante especifica una frecuencia de conmutación máxima), mientras que la frecuencia de sampleo, que es a la que se ejecuta el control, puede ser más alta. Esto mejora el control al disponer de mayor información sin sobrepasar la frecuencia a la que pueden conmutar los semiconductores.

Con todo ello, la función de coste queda:

$$g = \|i^*_{o,dq}(k+2) - i^p_{o,dq}(k+2)\|_2^2 + \lambda_1 \|S(k+1) - S(k)\|_2^2 \tag{2.9}$$

donde $i^*_{o,dq}$ es la referencia de corriente, expresada en ejes dq para el intervalo $k+2$, $i^p_{o,dq}$ es la predicción de corriente, expresada en ejes dq para el intervalo $k+2$, λ_1 es el peso que se le otorga al segundo término corrector en la función de coste, y $S(k)$,$S(k+1) \in \mathbb{R}^3$ son los estados para el intervalo k y $k+1$, respectivamente.

Una vez el valor para todos los estados candidatos ha sido evaluado, se comparan y se selecciona el menor de ellos, que corresponde al estado óptimo. Este estado será el que se aplique en el siguiente intervalo de control.

3 Plataforma ZYNQ-7000 y Kit de Evaluación Zedboard

Este capítulo se dedicará a repasar las características principales de la plataforma hardware de control en la que se implementará este trabajo. Se trata del Kit de evaluación y desarrollo Zedboard. Esta placa incluye como chip central un Zynq-7000 de Xilinx, en concreto la versión Z-7020. Sus características se repasarán en detalle en los siguientes apartados.

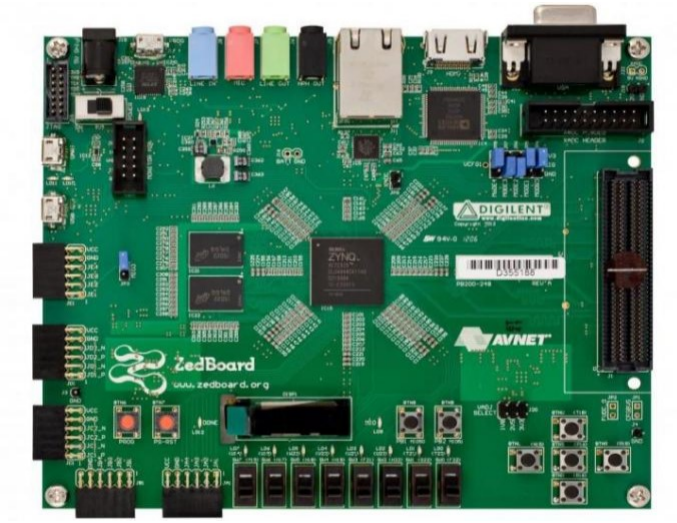

Figura 3.1 Kit de desarrollo Zedboard, del fabricante Avnet [35].

3.1 All-Programmable SoC Zynq-7000

En la literatura se utiliza el término Field Programmable System-on-Chip [32] para designar aquellas plataformas System-On-Chip que combinan uno o más microprocesadores junto con una parte lógica programable en un mismo chip.

La compañía Xilinx Inc. es reconocida por ser la empresa líder en el mercado de FPGAs, seguida por Altera (adquirida por Intel) y Lattice Semiconductor [43]. Entre la extensa gama de productos ofrecidos por Xilinx, existen las familias de FPGAs (de mayor a menor rendimiento): Virtex, Kintex y Artix. También tuvo una gran relevancia la familia de FPGAs Spartan (ya discontinuadas). En relación a este trabajo, existe también un segmento de plataformas FPSoC que Xilinx comercializa bajo el nombre de "All Programmable SoC Zynq-7000". De manera genérica, esta familia de plataformas "Zynq-7000" sale al mercado con el objetivo de ofrecer a programadores y diseñadores software y hardware una solución flexible que combine en una misma plataforma una parte de lógica programable (FPGA) basada en tecnología de 28 nm con microprocesadores Cortex-A9 single o dual-core [44]. El objetivo principal de esta plataforma es el de abordar un rango muy

amplio aplicaciones embebidas como el control y la automatización industrial, sistemas de comunicaciones inalámbricas o el procesamiento de audio y vídeo.

En cuanto a las características y periféricos incluidos en el integrado, Xilinx hace una distinción entre lo que denomina como "Processing System" (PS), la cual se compone de los microprocesadores ARM Cortex-A9 junto con las interfaces de memoria, cachés y demás periféricos; y la denominada como "Programmable Logic" (PL). Ésta última se trata de toda la parte de hardware programable que está basada en alguna familia de FPGAs actual ofrecida por Xilinx. Adicionalmente, se incluyen soluciones de interconexión de ambas partes basadas en el protocolo AXI de ARM [45].

A continuación se recogen algunas de las características más relevantes compartidas por todos los dispositivos de la familia Zynq-7000. En la Figura 3.2 se adjunta el diagrama de bloques de la arquitecura del Zynq-7000. Se recomienda consultar el documento [34] para un mayor detalle.

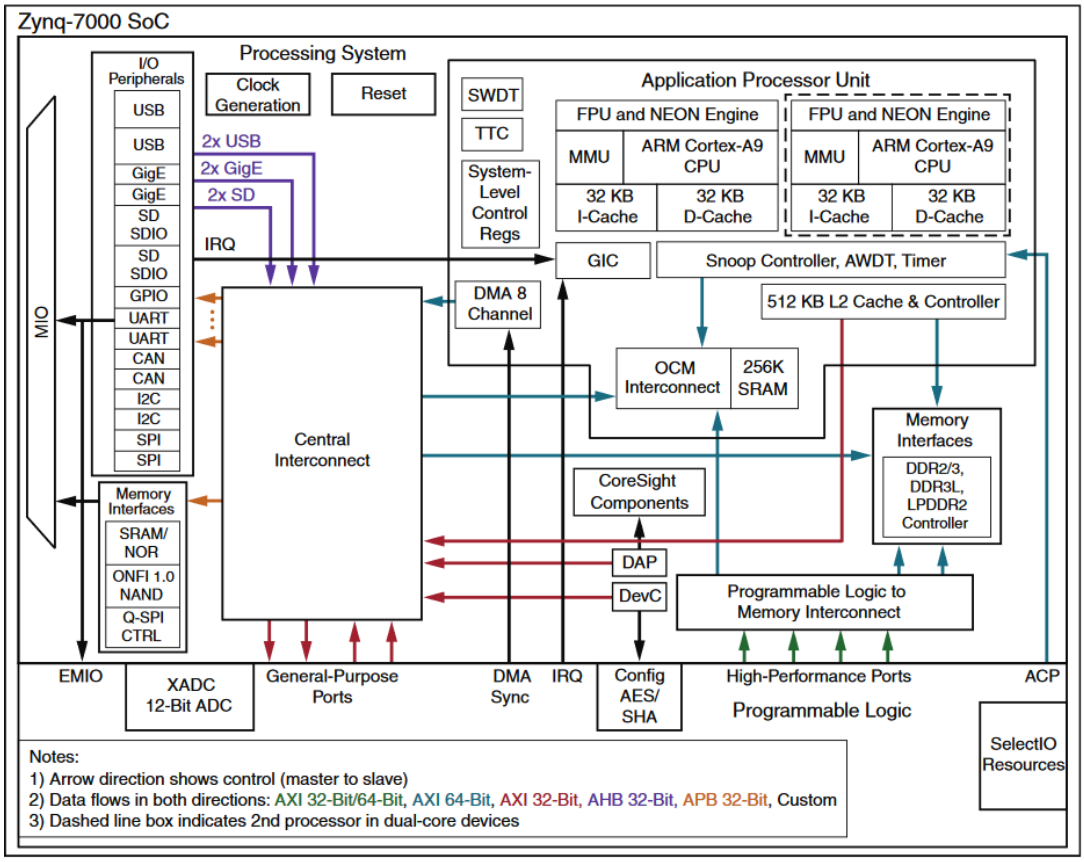

Figura 3.2 Diagrama de bloques de SoC Zynq-7000.

3.1.1 Processing System

- **ARM Cortex-A9**.
  - 2.5 DMIPS/MHz por CPU.
  - Frecuencia CPU hasta 1 GHz.
  - Soporte multiprocesador.
  - Arquitectura ARMv7-A.
  - Unidad de Punto Flotante Vectorial (VFPU) de precisión "single" y "double".
  - Temporizadores e Interrupciones.
    - Interrupciones tanto compartidas como privadas para cada CPU. El "Generic Interrupt Controller" (GIC) se encarga de la gestión, distribución y priorización de las interrupciones para las CPU.
    - Tres timers para Watchdog.

    - Un temporizador global.
    - Dos contadores de temporizador triple.

- **Cachés**.
  - Caché de Nivel 1 de 32 KB.
  - Caché de Nivel 2 de 512 KB.
  - Soporte de paridad de byte.
- **Memoria "On-Chip" (OCM)**.
  - RAM interna de 256 KB.
  - Soporte de paridad de byte.
- **Interfaces de Memoria Externa**.
  - Interfaces de 16 o 32 bits para DDR3, DDR3L, DDR2 o LPDDR2.
  - Soporte de ECC en modo 16 bits.
  - Espacio de direcciones de hasta 1 GB.
  - Interfaces de memoria estática.
    - Bus de datos de 8 bits para SRAM. Soporte hasta 64 MB.
    - Soporte de flash NOR paralelo.
    - Soporte de flash NOR serie basado en SPI de 1, 2 y 4 bits (Q-SPI), ó dos Q-SPI (8 bits).
    - Soporte de flash NAND ONFI1.0.
- **Controlador DMA de 8 canales**.
  - Memoria a Memoria.
  - Memoria a Periférico.
  - Periférico a Memoria.
    - Mediante una interfaz AXI de 64 bits, permite grandes transferencias de datos sin intervención directa del microprocesador [46].
- **Periféricos**.
  - Dos periféricos Ethernet MAC de velocidades 10/100/1000 con soporte del estándar IEEE 802.3 y 1588 rev 2.0.
    - Compatibilidad con DMA.
    - Interfaces GMII, RGMII y SGMII.
  - Dos periféricos USB 2.0 OTG.
    - IP core compatible con USB 2.0.
    - Soporte de modos OTG (On-The-Go) y Low, High y Full-Speed.
  - Dos interfaces de bus CAN con el soporte de CAN 2.0B.
    - Soporte para CAN 2.0-A y CAN 2.0-B y el estándar ISO 118981-1.
    - Soporte de modos OTG (On-The-Go) y Low, High y Full-Speed.
  - Dos controladores de SD/SDIO2.0/MMC3.31.
    - Soporte de hasta 4 líneas de datos [46].
  - Dos controladores SPI full-duplex con Slave Select (SS) para hasta 3 periféricos .
    - Funcionamiento configurable en modo maestro, esclavo o multi-maestro.
    - En modo maestro, el controlador SPI aporta la señal de reloj y las señales de Slave Select (SS). En modo esclavo, el reloj ha de ser aportado externamente para sincronizar las tramas.

    - FIFOs independientes para la lectura y escritura de 128 bytes y 8 bits de ancho. Los datos que se desean enviar se escriben mediante software en los registros correspondientes, y se envían en tramas de 8 bits de manera continua mientras haya datos en la FIFO. Para la lectura se hace de manera análoga.
    - El comienzo de una trama y la activación / desactivación se puede configurar como automática (gestiados por el controlador SPI) o manual (por software). Esto puede ser útil para enviar tramas de más de 8 bits [46].
  - Dos UART de alta velocidad (1 MHz).
    - Full-dúplex y configurable para distintos protocolos (bits de start, stop, paridad) y varias velocidades.
    - Líneas de lectura y escritura independientes, cada una con una FIFO de 64 bytes [46].
  - Dos interfaces maestro y esclavo I2C.
    - Funcionamiento configurable en modo maestro, esclavo o multi-maestro. Velocidad de hasta 400 Kb/s.
    - Soporte de direccionamiento de 7 y 10 bits.
    - FIFO de 16 bytes [46].
  - GPIO de cuatro bancos de 32 bits repartidos entre PS y PL. 54 de estos pines conectados a la PS.
  - 54 I/O multiplexadas (MIO) para asignación de pines.
    - El bloque MIO, que aparece en el esquema de la Figura 3.2 es un bloque que multiplexa 54 pines que salen de la PS entre los distintos periféricos que contiene la Zynq-7000. Esta multiplexación de los pines se puede decidir en la programación del dispositivo. En caso de que estos 54 pines no sean suficientes, se pueden asignar las señales de estos periféricos a señales EMIO (Extended-MIO), que son señales rutadas a la PL, permitiendo acceso a periféricos de la PS desde pines configurables en la PL [47, 46].

- **Interconexiones**.
  - Conectividad de banda ancha para la interconexión entre PL y PS.
  - Compatibilidad con protocolo AXI de ARM.

A modo de resumen de incluye la Figura 3.3, con una tabla extraída directamente del documento [34].

	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Processing System	Processor Core	Single-core ARM Cortex-A9 MPCore™ with CoreSight™			Dual-core ARM Cortex-A9 MPCore™ with CoreSight™						
	Processor Extensions	NEON™ & Single / Double Precision Floating Point for each processor									
	Maximum Frequency	667 MHz (-1); 766 MHz (-2)			667 MHz (-1); 766 MHz (-2); 866 MHz (-3)			667 MHz (-1); 800 MHz (-2); 1 GHz (-3)			667 MHz (-1) 800 MHz (-2)
	L1 Cache	32 KB Instruction, 32 KB data per processor									
	L2 Cache	512 KB									
	On-Chip Memory	256 KB									
	External Memory Support(1)	DDR3, DDR3L, DDR2, LPDDR2									
	External Static Memory Support(1)	2x Quad-SPI, NAND, NOR									
	DMA Channels	8 (4 dedicated to Programmable Logic)									
	Peripherals(1)	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO									
	Peripherals w/ built-in DMA(1)	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO									
	Security(2)	RSA Authentication, and AES and SHA 256-bit Decryption and Authentication for Secure Boot									
Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)		2x AXI 32b Master 2x AXI 32-bit Slave 4x AXI 64-bit/32-bit Memory AXI 64-bit ACP 16 Interrupts									

Figura 3.3 Características Zynq-7000 (PS). El integrado correspondiente a la Zedboard es el Z-7020.

3.1.2 Programmable Logic (Artix-7)

- **Configurable Logic Blocks (CLB)**.
  - Look-up tables (LUT).
  - Flip Flops.
  - Sumadores en cascada.
- **Block RAM de 36 Kb**.
  - Dual port.
  - Anchura de hasta 72 bits.
  - Configurable como block RAM dual de 18 Kb.
- **Bloques DSP**.
  - Multiplicación de 18 x 25 bits con signo.
  - Acumulador/Sumador de 48 bits.
  - Pre-Sumador de 25 bits.
- **Bloques de I/O programables**.
  - Soporte de LVCMOS, LVDS y SSTL.
  - I/O de 1.2 V a 3.3 V.
  - Delay programable.
- **Bloque PCI Express**.
  - Velocidades hasta Gen2.
  - Hasta 8 líneas.
- **Transceptor serie**.
  - Hasta 16 receptores y transmisores.
  - Tasa de datos de hasta 12.5 Gb/s.
- **Dos Convertidores Analógico-Digital (ADC) de 12 bits**.
  - Sensado de tensión y temperatura del chip.
  - Hasta 17 canales diferenciales de medida.
  - Tasa de medidas de hasta 1 MSPS.

A modo de resumen de incluye la Figura 3.4, con una tabla extraída directamente del documento [34].

Como se puede observar, la FPGA con la que cuenta el Zynq-7000 presente en la Zedboard (Z-7020) incluye como elementos más destacados 85 000 celdas de lógica programable, 53 200 LUTs y 106 400 Flip-flops. Adicionalmente, posee 140 bloques RAM de 36 Kb, lo que hace un total de 4.9 Mb de RAM en la parte FPGA para guardar datos. Cuenta también con 220 bloques DSP que permiten la realización de cálculos de multiplicación/suma de manera rápida y sin ocupar un gran número de LUTs y otros recursos de la FPGA.

Conviene repasar alguna de las características principales de estos elementos de la parte lógica programable, ya que se tratarán posteriormente en el análisis de los recursos consumidos por el control implementado en la FPGA.

	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Programmable Logic	Xilinx 7 Series Programmable Logic Equivalent	Artix®-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Kintex®-7 FPGA	Kintex-7 FPGA	Kintex-7 FPGA	Kintex-7 FPGA
	Programmable Logic Cells	23K	55K	65K	28K	74K	85K	125K	275K	350K	444K
	Look-Up Tables (LUTs)	14,400	34,400	40,600	17,600	46,200	53,200	78,600	171,900	218,600	277,400
	Flip-Flops	28,800	68,800	81,200	35,200	92,400	106,400	157,200	343,800	437,200	554,800
	Block RAM (# 36 Kb Blocks)	1.8 Mb (50)	2.5 Mb (72)	3.8 Mb (107)	2.1 Mb (60)	3.3 Mb (95)	4.9 Mb (140)	9.3 Mb (265)	17.6 Mb (500)	19.2 Mb (545)	26.5 Mb (755)
	DSP Slices (18x25 MACCs)	66	120	170	80	160	220	400	900	900	2,020
	Peak DSP Performance (Symmetric FIR)	73 GMACs	131 GMACs	187 GMACs	100 GMACs	200 GMACs	276 GMACs	593 GMACs	1,334 GMACs	1,334 GMACs	2,622 GMACs
	PCI Express (Root Complex or Endpoint)[3]		Gen2 x4			Gen2 x4		Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs									
	Security[2]	AES and SHA 256b for Boot Code and Programmable Logic Configuration, Decryption, and Authentication									

Figura 3.4 Características Zynq-7000 (PL). El integrado correspondiente a la Zedboard es el Z-7020.

Configurable Logic Blocks (CLB)

Es importante no confundir el concepto de "celda lógica programable" que aparece en la Figura 3.4 con el de "configurable logic block" o CLB. Mientras que estos últimos son el bloque lógico básico presente realmente en una FPGA, el concepto de "celda lógica programable" es una métrica que utiliza Xilinx de cara a la promoción de sus productos con el objetivo de unificar la medida del número de celdas lógicas presentes en sus FPGAs independientemente de la tecnología de las mismas. De esta forma, se puede comparar lo que ofrecen distintas familias de FPGAs de distintas generaciones a nivel de prestaciones. El motivo de todo esto es que las celdas lógicas de las sucesivas familias más modernas han aumentado en complejidad, y por tanto ha aumentado la capacidad que estas ofrecen a nivel de cálculo lógico y aritmético, por lo que no es justo comparar directamente el número de celdas lógicas sin considerar la potencia de las mismas. La metodología se basa en tomar como base una CLB básica y añadir multiplicadores al número de celdas para contabilizar capacidades mayores dentro de cada CLB.

De manera genérica, estos CLB incluyen una parte para la implementación de lógica combinacional, normalmente LUTs (almacenan una tabla de verdad que implementa una función lógica) con un número de entradas determinado (en las FPGA modernas se suelen ver de hasta 6 o más entradas). Así mismo, incluyen recursos de interconexionado y registros (flip-flops o latches) para la implementación de lógica secuencial que permita la sincronización del circuito con una señal de reloj.

Para ilustrar con mayor claridad en qué consiste un CLB de una FPGA, se puede extraer el ejemplo simplificado de la Figura 3.5:

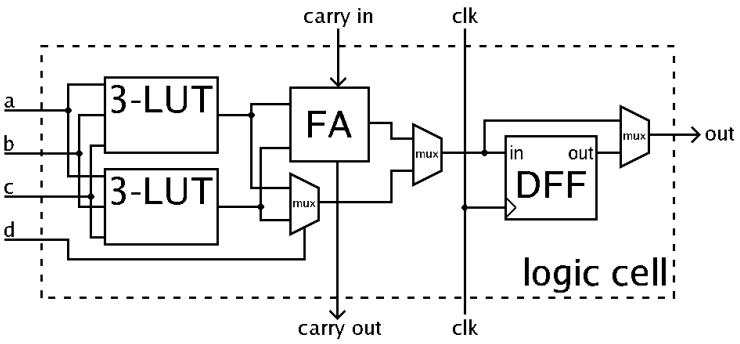

Figura 3.5 Ejemplo de un CLB básico en una FPGA [48].

Como se puede observar, este CLB en particular posee hasta 4 entradas y una salida. Tiene una primera parte de lógica combinacional con dos LUTs de 3 entradas para implementar la función lógica requerida y un bloque sumador (FA) que se puede bypassear mediante los multiplexores (mux). A la salida del último multiplexor hay un flip-flop de tipo D cuya función es sincronizar la salida (out) con la señal de reloj (clk).

En el caso particular de las FPGA de la familia Artix-7, el diseño de una CLB es más complejo. Se puede encontrar documentación detallada ofrecida por Xilinx sobre las CLB de sus FPGAs de la serie 7 [49].

Resumidamente, un CLB como la que se puede encontrar en la FPGA Artix-7 del chip Z-7020 (ver Figura 3.6), está formado por dos "slices" o bloques lógicos, que a su vez están compuestos por LUTs y flip-flops de manera similar a lo visto para el ejemplo simplificado. Las entradas y salidas (señales de datos y reloj) se gestionan mediante una matriz de conmutación que da acceso al resto de recursos de la FPGA. Las señales "CIN" y "COUT" corresponden a los "carry chain" que van de un slice a otro verticalmente.

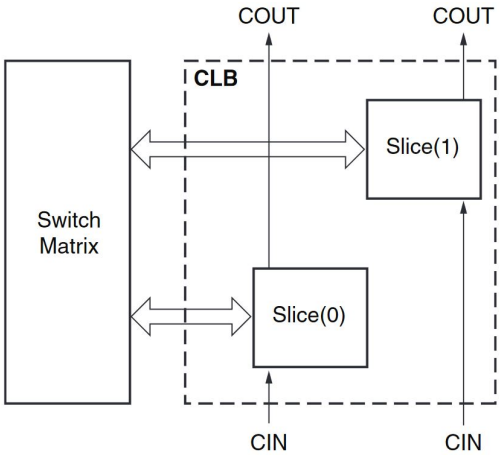

Figura 3.6 CLB en FPGA serie 7 de Xilinx [49].

Existen dos tipos de "slice" presentes en las FPGA de la serie 7 de Xilinx. Por un lado, están las "SLICEL" que sólo pueden ser utilizadas para operaciones lógicas y aritméticas, mientras que las "SLICEM" también pueden ser utilizadas como memoria ("distributed RAM"). Aproximadamente, dos tercios de las "slice" son de tipo "SLICEL" en esta familia de FPGAs.

De manera más detallada, cada "slice" de los dos presentes en un CLB está compuesto por cuatro LUTs de 6 entradas y 8 flip-flops (cada LUT tiene asociados dos de ellos). De estos 8 flip-flops, 4 se pueden configurar como latches (en este caso, los 4 restantes deben permanecer inactivos). Adicionalmente, cada CLB basada en "SLICEM" puede configurarse como 256 bits de RAM distribuida o registro de desplazamiento de 128 bits. A modo ilustrativo, se puede ver el diseño de una "SLICEL" en la Figura 3.7 y una "SLICEM" en la Figura 3.8.

Desde el punto de vista del programador, no es necesario instanciar estos bloques de manera específica en el diseño hardware para que sean inferidos por la herramienta de síntesis. Este tipo de recursos son habitualmente inferidos automáticamente por la herramienta para implementar buena parte de la lógica diseñada.

DSP blocks

A parte de todas las celdas lógicas presentes, las FPGAs de la serie 7 de Xilinx incluyen "slices" dedicados para la implementación de operaciones aritméticas. Estos "slices" son los bloques DSP48E1, y su utilización para realizar operaciones de tipo multiplicación o suma permite ahorrar un número importante de recursos CLB. Un sólo bloque DSP es capaz de realizar, de manera rápida y eficiente, una operación que consumiría un gran número de LUTs y registros.

De manera básica, estos bloques incluyen: un multiplicador de complemento a 2 de 25 x 18 bits, acumulador de 48 bits, pre-sumador para el ahorro de energía, unidad lógica y unidad aritmética SIMD (Single-Instruction-Multiple-Data) que permite la realización de sumas y restas en paralelo (dos de ellas si 24 bits o cuatro si 12 bits) [50]. En la Figura 3.9 se adjunta un esquema detallado de los componentes de un bloque DSP48E1.

La síntesis de este tipo de bloques en el diseño hardware no es tan inmediata como en el caso de los CLB. El número de DSPs es más limitado, por lo que es conveniente utilizarlos en aquellas operaciones que ahorren un mayor número de recursos de otro tipo. El manual de usuario para síntesis que ofrece Xilinx [51] ofrece las guías básicas a seguir en un diseño hardware para obtener los resultados de síntesis buscados. Por defecto, la herramienta infiere (si no implica el incumplimiento de algún otro tipo de "constraint" añadido al diseño) bloques DSP cuando detecta operaciones de multiplicación, multiplicación+suma, multiplicación+resta, y multiplicación+acumulador. Las operaciones de suma, resta y acumulación no son inferidas en bloques DSP

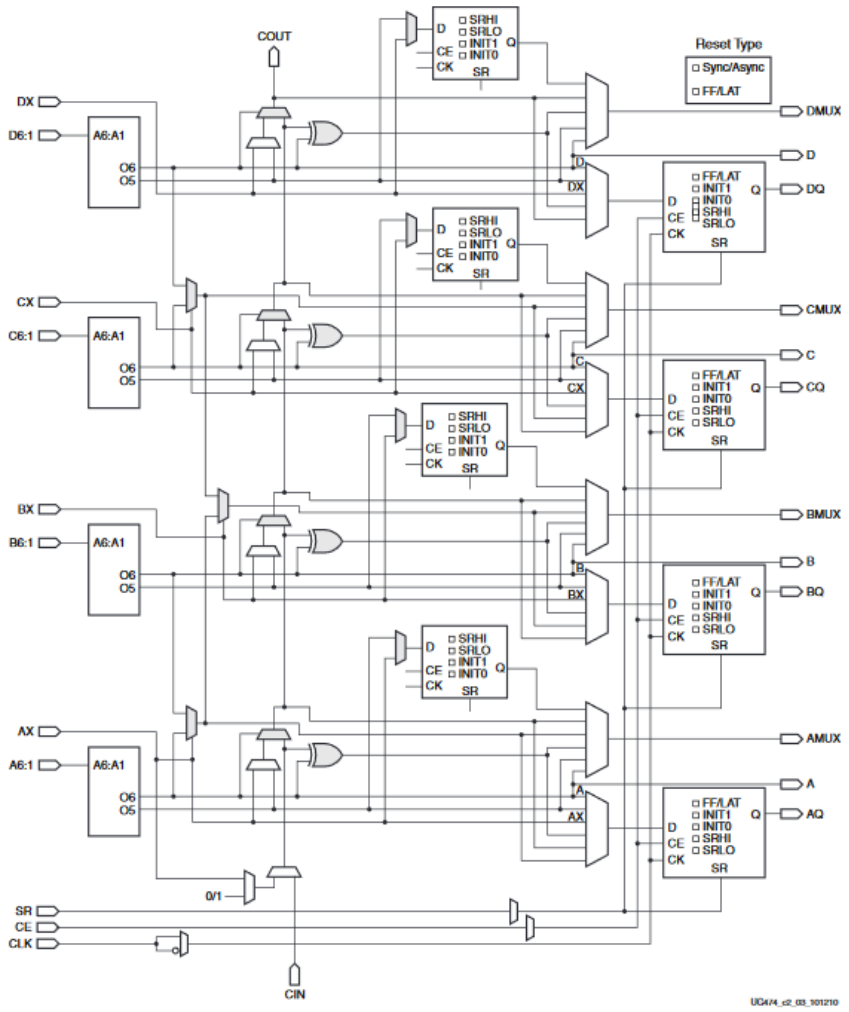

Figura 3.7 "SLICEL" de la serie 7 de FPGAs de Xilinx [49].

por defecto, sino que son implementadas mediante lógica de los CLB. Si se desea forzar que este tipo de operaciones se realizen en bloques DSP, es necesario añadir el atributo "USE-DSP48" ó "USE-DSP" al diseño HDL. Se puede aplicar este atributo a señales, arquitecturas y componentes y módulos y entidades. Este atributo puede tomar los valores de "logic" (permite la implementación de estructuras XOR en DSPs), "yes", y "no" (estos dos últimos permite la implementación de la lógica en DSPs o no).

RAM blocks

Las FPGAs de la serie 7 de Xilinx incluyen también una serie de recursos de memoria que pueden ser utilizados para el almacenamiento de datos sin necesidad de gastar otros recursos de la parte lógica. En particular, Xilinx ofrece para estas FPGAs la inclusión de bloques RAM de hasta 36 Kbits de datos. Como para el resto de recursos, Xilinx ofrece documentación para este tipo de bloques [52].

De manera resumida, estos bloques permiten la configuración de memorias de doble puerto, con escritura y lectura síncronas (cada una de estas operaciones requiere un flanco de reloj) e independientes. Pueden funcionar como un bloque de 36 Kb o como dos bloques de 18 Kb totalmente independientes. Las posibles configuraciones para un bloque completo son (posiciones de memoria X ancho de bits): 64K x 1 (utilizando dos bloques adyacentes), 32K x 1, 16K x 2, 8K x 4, 4K x 9, 2K x 18, 1K x 36, ó 512 x 72. En el caso de que se trate de un bloque de 18Kb: 6K x 1, 8K x2 , 4K x 4, 2K x 9, 1K x 18 ó 512 x 36.

Desde el punto de vista del diseño hardware, es también conveniente seguir las recomendaciones de [51], de cara a asegurar que la herramienta infiera correctamente un bloque de RAM en aquellos bloques del diseño donde sea conveniente. Es decir, en aquellos donde sea preferible la utilización de estos bloques frente al uso de "distributed RAM" que utiliza CLBs. En general, es más recomendable utilizar los recursos dedicados de memoria cuanto mayor es la cantidad de datos que se quieren almacenar.

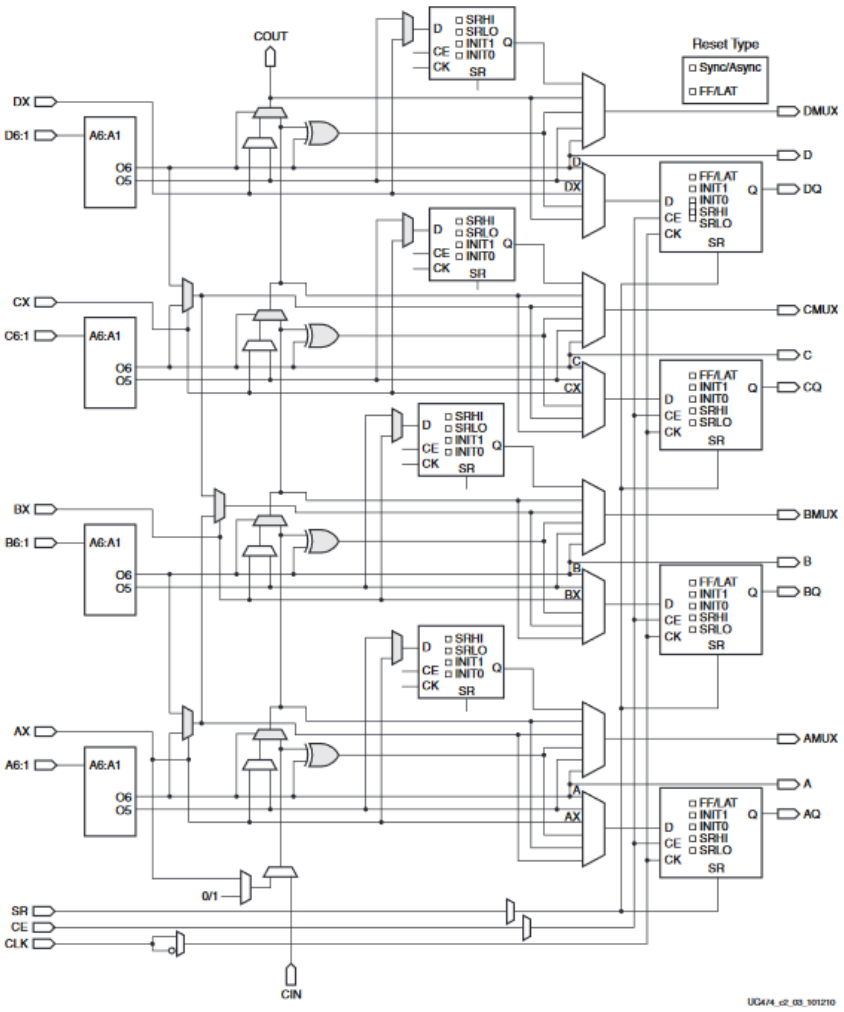

Figura 3.8 "SLICEM" de la serie 7 de FPGAs de Xilinx [49].

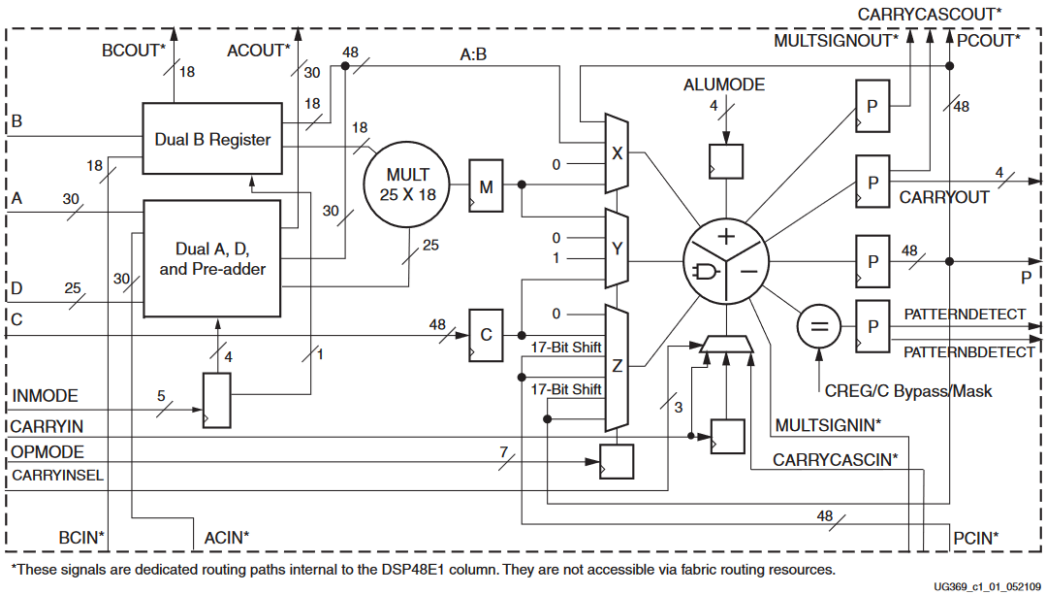

Figura 3.9 Bloque "DSP48E1" de la serie 7 de FPGAs de Xilinx [50].

3.2 Placa de evaluación Zedboard

Una vez descritas las características más relevantes del integrado Zynq-7000, se procederá a la revisión de la placa de evaluación Zedboard [53], del fabricante Avnet, en la que se basará el desarrollo de este proyecto y permitirá el desarrollo de aplicaciones para la plataforma Zynq-7000.

En el kit de evaluación que comercializa Avnet (ver Figura 3.10), se incluye la placa de desarrollo Zedboard, con el integrado Zynq-7000 (Z-7020) y todos los periféricos que el fabricante ha proyectado para esta placa. Los esquemáticos de la placa en su revisión "Rev. D" pueden consultarse en [54].

Así mismo, el kit de evaluación incluye una tarjeta SD de 4 GB, con una imagen de Linux pregrabada para poder utilizar a modo de demostración, fuente de alimentación de 12 V y 5 A, un cable micro USB para conectar al PC y un adaptador USB - Micro USB.

Figura 3.10 Kit de evaluación Zedboard [53].

En cuanto a las características principales incluidas por la placa, se muestran a continuación algunas de ellas y se recogen de manera esquemática en la Figura 3.11.

- **Xilinx® XC7Z020-1CLG484C Zynq-7000 AP SoC (Opciones de configuración)**.
  - QSPI Flash.
  - Cascaded JTAG.
  - Tarjeta SD.
    - Los dispositivos Zynq-7000 llevan a cabo un proceso de "booteo" multi-etapa en el que el PS es el maestro del mismo. Mediante jumpers en la placa conectados a los pines MIO[8:2] se puede configurar el dispositivo primario y el modo de "booteo".

- **Memoria**.
  - 512 MB de memoria DDR3 (128Mb x 32).
    - Dos integrados de 128 Mb x 16 del fabricante Micron que se conectan al controlador de memoria DDR en la PS del Zynq-7000. Esta interfaz puede llegar a los 533 MHz de velocidad de reloj, lo que implica una tasa de datos de 1066 Mbs (Dual Data Rate).
  - 256 Mb de memoria Flash (QSPI).
    - Memoria Flash de tipo NOR serie accesible por quad-SPI (4 bit) del fabricante Spansion. Se puede utilizar para "bootear" el Zynq 7000, programando y configurando tanto la PS como la PL. Alcanza velocidades de reloj de 104 MHz (que se traduce en 400 MHz en modo Q-SPI).

- **Interfaces**.
  - Programación USB-JTAG.
    - Mediante el módulo USB "High Speed JTAG Module" de Digilent, la Zedboard permite funcionalidad JTAG desde las herramientas software de Xilinx a través del puerto micro USB.
  - Ethernet 10/100/1G.
  - USB OTG 2.0.
    - La Zedboard implementa una interfaz de 8 bits de tipo ULPI mediante un chip transceptor USB que soporta velocidades de hasta 480 Mb/s.
  - Interfaz para tarjeta SD.
    - La tarjeta SD se puede utilizar de almacenamiento externo no volátil y como medio de "booteo". La Zedboard incluye el conector estándar de 9 pines que a su vez se conecta al periférico SD/SDIO de la PS.
  - USB 2.0 (USB-UART).
    - Mediante un chip puente USB-UART del fabricante Cypress, se permite el acceso al periférico UART de la PS a través de un puerto micro USB, que permite el intercambio de información desde una consola mediante puerto serie.
  - 5 puertos PMOD de 2x6 pines (1 PS, 4 PL).
    - Los puertos PMOD son conectores de 2 filas de 6 pines que trabajan a 3.3 V. Este conector tiene dos pines de alimentación y dos pines de tierra, mientras que el resto de pines son entradas y salidas digitales configurables. El conector asociado a la PS (PMOD JE) puede ser configurado para los periféricos del Zynq-7000 vistos anteriormente (a través del bloque MIO). El resto de conectores (PMODs JA, JB, JC y JD) se pueden asociar a señales de la PL, e indirectamente a periféricos de la PS, mediante EMIO.
  - Un puerto LPC FMC.
    - Ranura "Low Pin Count" (LPC-FMC) con 68 pines de entrada/salida de tipo "single-ended", también configurables como 34 pares diferenciales. La tensión a la que trabajan estos pines es configurable a 1.8 V, 2.5 V ó 3.3 V, mediante un jumper en la placa que permite la configuración de la tensión de alimentación "Vadj" a estos valores.
  - Un puerto AMS.
    - Este conector ofrece acceso a los pines del ADC en el Zynq-7000 (xADC). En particular, a 3 de las señales diferenciales del mismo: Vp/n, Vaux0p/n, y Vaux8p/n.
  - Dos botones de Reset (1 PS, 1 PL).
  - Siete botones de usuario (2 PS, 5 PL).
  - Ocho interruptores deslizadores (8 PL).
  - Nueve LEDs de usuario (1 PS, 8 PL).
  - Un LED indicativo de programación exitosa de la PL.
- **Osciladores**.
  - 33.333 MHz (PS).
    - Fuente de reloj principal de la PS, puede generar hasta 4 salidas adicionales de reloj basadas en PLL para la PL.
  - 100 MHz (PL).
    - Fuente de reloj principal de la PL, en el pin Y9 del banco 13.
- **Audio y vídeo**.
  - Salida HDMI (Soporte de 1080p60Hz y color de 16 bits).
  - Salida VGA (Color de 12 bits).

  - Pantalla LED de 128x32.
  - Salidas y entradas de audio (Códec de Audio I2S).

- **Alimentación**.
  - Interruptor de Encendido/Apagado.
  - Alimentación a 12 V y 5 A mediante un conector "barrel jack".
    * Mediante una serie de reguladores de tensión del fabricante Maxim, se generan las diferentes tensiones necesarias para el Zynq-7000 y el resto de integrados en la placa.

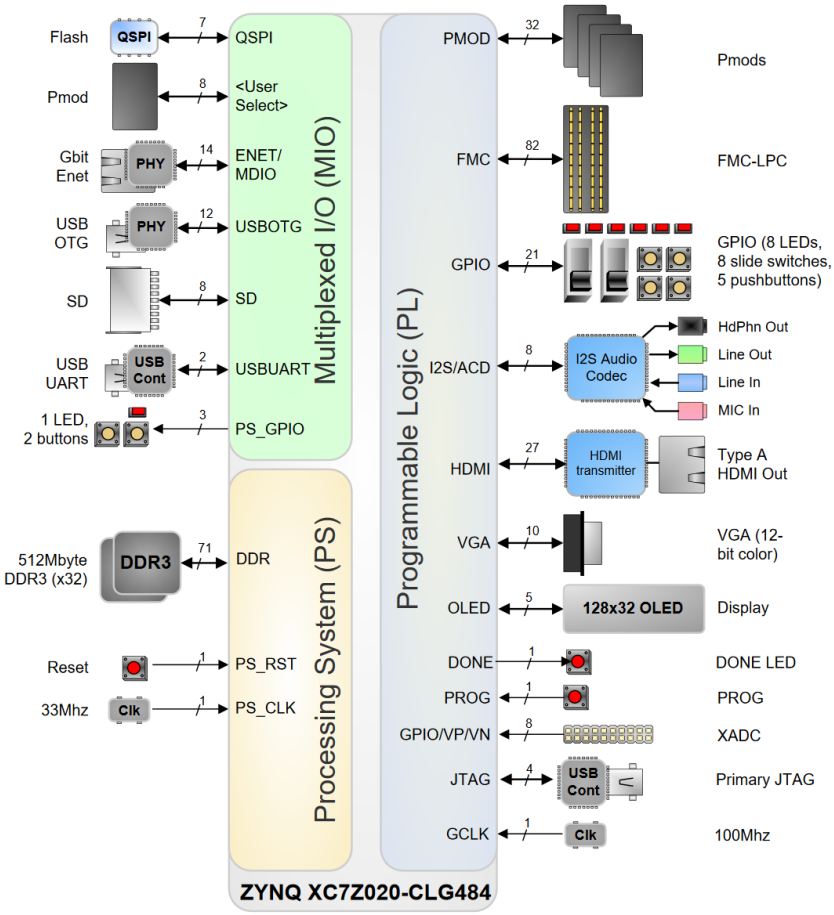

Figura 3.11 Diagrama de bloques de componentes en placa de desarrollo Zedboard [35].

También resulta interesante conocer la distribución de los diferentes pines, tanto de la parte PS como de la PL del Zynq-7000. Estos pines se distribuyen en bancos cuyos niveles de tensión pueden venir prefijados a 1.8 V, 3.3 V o ser configurable mediante "Vadj". Esta distribución se recoge en la Figura 3.12.

A cada uno de estos pines del Zynq-7000 les corresponde un código de una o dos letras y un número de dos cifras que lo identifica. De cara al diseño hardware que se implementará, será necesario asignar los puertos de entrada y salida requeridos a estos pines físicos. Esto se realiza mediante un fichero en formato ".xdc" de "constraints", que entre otras cosas codifica la asociación física de los puertos del diseño hardware ("placement constraints"). También puede incluir otro tipo de "constraints" de tiempo o de síntesis que ordenen a la herramienta ciertas guías para la síntesis e implementación del diseño.

Los pines correspondientes al bloque MIO quedan definidos en la configuración del mismo, y no es necesario añadirlos al fichero de constraints. Sin embargo, sí es necesario para aquellos pines asociados a la parte PL. Para facilitar la confección del fichero de "constraints", el fabricante de la Zedboard ofrece un fichero maestro [55] donde se definen todos los posibles pines disponibles. Tan solo es necesario cambiar el nombre del puerto al puerto de nuestro diseño al que se quiera asignar el pin y comentar aquellos pines que no se vayan a utilizar.

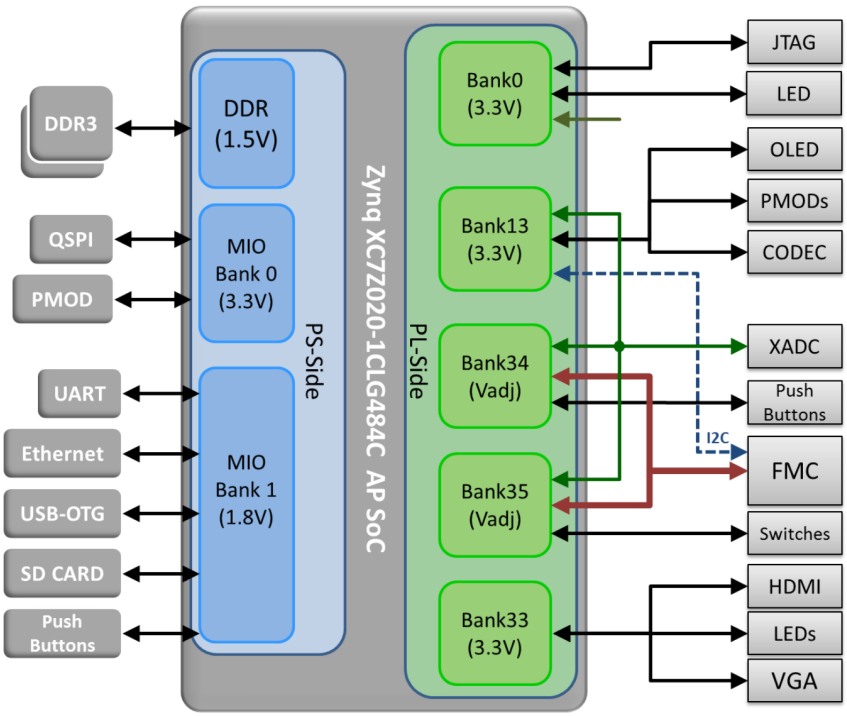

Figura 3.12 Distribución de pines del integrado Zynq-7000 en bancos [35].

4 Convertidor Analógico Digital ADS7953

En este capítulo se describirá la solución adoptada desde el punto de vista del hardware para la medición de las magnitudes eléctricas. Todo algoritmo de control en bucle cerrado que se pueda plantear necesita conocer el valor de un determinado número de variables que describen el estado del sistema en cada instante en el que se calcula la acción de control. Esto se traduce en la necesidad de medir ciertas magnitudes del sistema real a través de sensores. Para un convertidor de potencia será necesario, en general, medir las tensiones y las corrientes en distintos puntos del sistema. Para la aplicación que se quiere desarrollar en este proyecto se necesita, en concreto, medir la tensión de continua del dc-link y la corriente trifásica que pasa por las cargas.

Para poder llevar a cabo estas medidas es necesario un convertidor analógico digital (ADC, del inglés "Analog to Digital Converter") que tome la señal analógica adaptada a un determinado nivel de tensión de los sensores y la convierta a valores digitales con los que un microprocesador o una FPGA pueda trabajar y realizar cálculos. Habitualmente, el resultado será un número binario en formato de complemento a 2 con un número de bits determinado que se van guardando en registros de memoria según se realizan las mediciones. Posteriormente, los números digitales guardados en estos registros se tratan y se operan para adaptarlos al formato requerido.

Existen en el mercado numerosas alternativas de convertidores analógico digitales con características muy variadas. Incluso el propio Zynq-7000 ofrece un ADC que podría utilizarse para llevar a cabo este cometido. Se descarta en principio esta opción debido al limitado número de canales disponibles que el fabricante Avnet ha proyectado en su placa de evaluación. La idea es optar por un ADC externo que ofrezca un mayor número de canales de medidas, no sólo de cara a la aplicación desarrollada en este documento, si no para dejar una plataforma de control funcional de cara al futuro para otro tipo de aplicaciones que pueden utilizar un mayor número de medidas.

En el primer apartado este capítulo, se describirán los sensores presentes en los laboratorios donde se llevarán a cabo las pruebas con el equipo real. Se hará lo propio para la placa de adaptación de medidas que se utiliza para convertir la señal que proporcionan los sensores al rango de trabajo habitual de las plataformas de control convencionales que se suelen utilizar en el Grupo de Tecnología Electrónica del Departamento de Ingeniería Electrónica. Este primer apartado proporcionará el marco de trabajo de cara a la elección de un ADC apropiado cuyas principales características serán expuestas en el segundo apartado.

4.1 Sensores y etapa de adaptación

4.1.1 Transductores LEM de tensión y corriente

El equipo de pruebas presente en el laboratorio se trata de un convertidor trifásico de dos niveles basado en transistores de potencia de tipo IGBT (ver Figura 4.1). Posee un driver de potencia del fabricante Semikron, con una etapa de adaptación de fibra óptica que gestiona las señales de entrada y salida. En concreto, convierte las señales ópticas que llegan de la plataforma de control a señales eléctricas que puedan ser alimentadas a los dispositivos de potencia. Se opta por esta etapa intermedia en fibra óptica para aislar la plataforma de control del ruido eléctrico que generan las conmutaciones en el equipo de potencia. Así mismo, el convertidor posee un condensador en su dc-link con un sensor de tensión. Dos terminales permiten la conexión de una fuente de tensión de continua al dc-link, en el caso de requerir la imposición de una tensión constante.

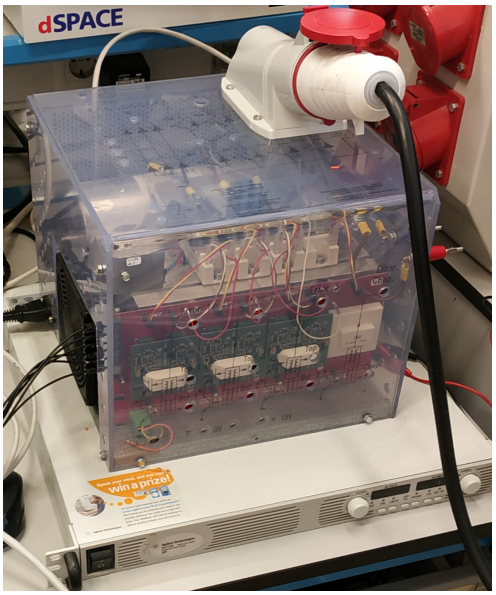

Figura 4.1 Convertidor trifásico de dos niveles en laboratorio.

Por otro lado, para completar el sistema representado por el circuito esquematizado en la Figura 2.1, se dispone de una carga RL trifásica configurable a diversos valores mediante reconexionado en paralelo de resistencias. Esta carga RL se conecta al convertidor a través de una manguera trifásica. En un punto intermedio, se dispone una PCB con tres sensores de corriente a través de los cuales pasan los cables de cada una de las tres fases y miden la intensidad que circula por cada una de ellas.

Tanto estos sensores como el sensor de tensión del dc-link, pertenecen a la misma familia de transductores del fabricante LEM. En concreto, los sensores utilizados son el LV 25-P [56] para tensiones y el LA 55-P [57] para corrientes. Ambos basan su principio de funcionamiento en el efecto Hall.

Básicamente, los transductores de tensión (ver Figura 4.2) ofrecen una salida en corriente (Is) que puede ser tanto positiva como negativa con valor nominal RMS de 25 mA (será positiva si la tensión en el terminal +HV es positiva). El rango de valores nominales de tensión a medir recomendado para este sensor es de 10 V hasta 500 V. El datasheet del fabricante para el LV 25-P recomienda que se configure el sensor de manera que colocando una resistencia R1, se limite la intensidad nominal por su primario a 10 mA (ésta sería la intensidad que circularía para la tensión nominal a medir por el sensor). En este caso, se configura a una tensión nominal de 440 V. El sensor requiere una tensión de alimentación simétrica (Uc), en el rango de $\pm$12 V a $\pm$15 V.

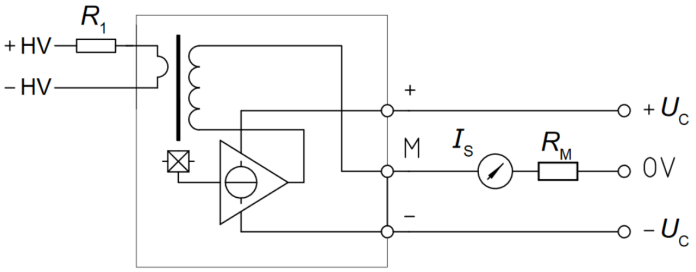

Figura 4.2 Esquema sensor LV 25-P [56].

En el caso de los transductores de intensidad (ver Figura 4.3), su funcionamiento es similar, en el sentido que proporciona igualmente una intensidad Is de salida que será positiva en el caso de que Ip fluya en el sentido indicado en la Figura 4.3. Esta corriente de salida estará en el rango de $\pm$ 50 mA (valor RMS). El rango de medidas recomendado por el fabricante es de 50 A RMS (70 A de pico). De esta manera, una corriente de 50 A RMS se correspondería con una corriente a la salida del sensor de 50 mA RMS. De no haber corriente, el sensor no ofrece corriente a la salida. La tensión de alimentación requerida es la misma que para el LV-25 P.

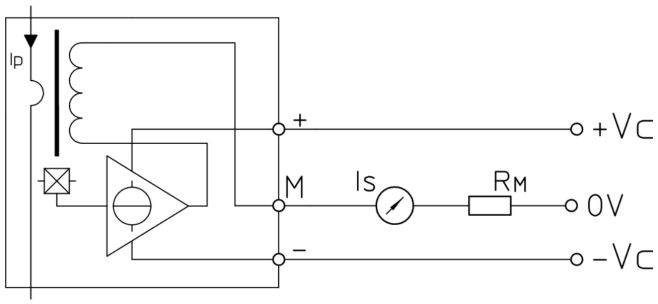

Figura 4.3 Esquema sensor LA 55-NP [57].

4.1.2 Etapa de adaptación

Para la etapa de adaptación se utilizará la placa de adaptación "GTE M2C-M Analog Interface" (ver Figura 4.4) diseñada por Abraham Márquez para el Grupo de Tecnología Electrónica. Su diseño se basa en amplificadores operacionales OPA4340 configurados con ganancia no inversora más offset.

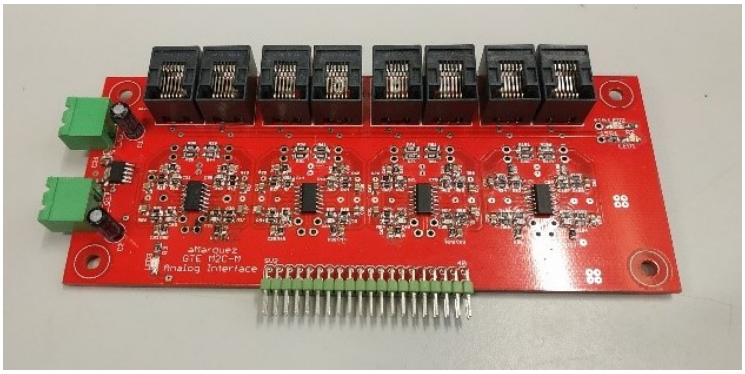

Figura 4.4 Placa de adaptación de medidas.

Básicamente, la placa dispone una serie de conectores RJ-11 de 6 pines que se utiliza como interface para la interconexión con los sensores. Como se ha podido ver en el anterior apartado, los sensores ofrecen la medida en corriente mediante un pin. Adicionalmente, requieren una alimentación simétrica de $\pm$ 15 V. De este modo, con un sólo conector RJ-11 de 6 pines se pueden alimentar hasta dos sensores con dos conexiones extra que se utilizan para la señal en corriente de los dos sensores. Por otro lado, cada integrado de OPA4340 tiene 4 entradas/salidas. En total, la placa tiene ocho conectores RJ-11 entradas, lo que se traduce en 16 medidas de sensores. Para ello, utiliza cuatro integrados OPA 4340. Esquemáticamente, la señal que proporciona un sensor se adapta en un canal de alterna según el circuito de la Figura 4.5.

Como se puede apreciar, la etapa de adaptación recibe la señal en corriente con una resistencia en paralelo conocida (200 Ω), de manera que conociendo el rango de corrientes que llega del sensor (rango máximo $\pm$ 25 mA), se pasa a trabajar en un rango de tensiones, que para este punto será de $\pm$ 5 V. Dicha tensión de entrada se pasa por la etapa del amplificador operacional en configuración no inversora más offset (ganancia de 1.5/5), la cual transforma esos $\pm$ 5 V a una tensión en el rango de 0 a 3 V, apropiada para su uso con la mayoría de DSPs utilizados en el departamento para este tipo de aplicaciones. Este offset impuesto es de 1.5 V en el caso de magnitudes de alterna, mientras que se reservan 4 entradas/salidas para medir tensiones de continua sin offset impuesto. De esta manera, en el caso de sensores de alterna, la señal se centra en torno a 1.5 V, mientras que la ausencia de señal en el caso de sensores de continua (sólo se contempla la medición de tensiones positivas) se traduce en 0 V a la salida que llega al ADC.

4.2 Convertidor Analógico Digital ADS7953

Una vez se han descrito los sensores y la etapa de adaptación a utilizar, se puede proceder a la introducción del ADC externo. El ADC elegido para este proyecto, deberá reunir una serie de requisitos básicos. El principal es que sea capaz de llevar a cabo las medidas necesarias con la velocidad suficiente para tener una tasa de

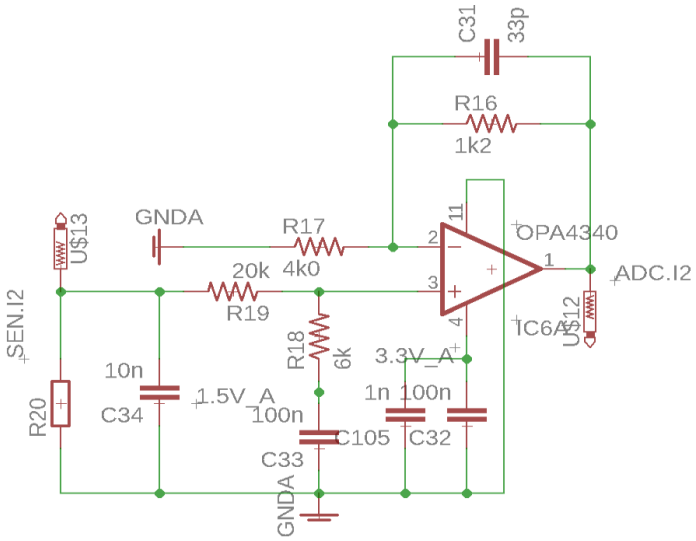

Figura 4.5 Esquemático circuito adaptación.

sampleo para nuestro control de al menos en el orden de varias decenas de kiloherzios. Por otro lado, de cara a aplicaciones más complejas que se puedan desarrollar en el futuro, será necesario medir tensiones e intensidades trifásicas en varios puntos, por lo que tener un número alto de canales de medida es algo importante. Otro factor crucial será la disponibilidad de placas de evaluación en el mercado que permitan trabajar con el ADC directamente sin necesidad de acometer la fabricación de una PCB para el integrado del ADC. Esto supondría un gasto de recursos y de horas de trabajo muy importante, tanto en el diseño de la placa como en su fabricación y testeo. Por ello, encontrar un módulo de evaluación cuyo funcionamiento esté garantizado por el fabricante desde el primer momento será determinante. A parte de todo ello, son también características relevantes la precisión del ADC, su resolución, los protocolos de salida que permita para la lectura de los datos y que las entradas analógicas que acepte estén en el rango de la etapa de adaptación que se quiere utilizar.

Buscando en el mercado entre los principales fabricantes de convertidores analógico digitales (Texas Instruments, Analog Devices o Maxim), se halla el ADS7953 de Texas Instruments, el ADC que se utilizará para este Trabajo Fin de Máster [58]. El motivo principal de su elección radica en la disponibilidad del kit de evaluación ADS7953EVM-PDK, que incluye una PCB con el integrado del ADC, y toda la circuitería necesaria para el acceso a las entradas analógicas, la interfaz digital y la alimentación [59]. Así mismo, este ADC reúne una serie de características que lo hacen apropiado para la aplicación que se quiere desarrollar en este Trabajo Fin de Máster y que se procede a resumir en las siguientes líneas:

- **Medidas y velocidad**.
  - Tasa de sampleo de hasta 1 MSPs.
  - Resolución de 12 bits. Formato digital de 0 a 4095 en binario.
  - 16 canales de medidas.
  - Principio de funcionamiento basado en aproximaciones sucesivas (SAR) con un Sample and Hold.
  - Modos de funcionamiento manual y automáticos.
  - GPIOs configurables.
- **Alimentación y entradas analógicas**.
  - Rango de alimentación: 2.7 a 5.25 V.
  - Rango de alimentación para interfaz digital de salida : 1.7 a 5.25 V.
  - Rango de tensión de referencia: 2 a 3 V (funcionamiento nominal a 2.5 V).
  - Rango de tensión de entradas analógicas: Modo de 0 a Vref o de 0 a 2Vref.
  - Entradas unipolares.

- **Interfaz digital de entrada/salida**.
  - Protocolo SPI (Serial Peripheral Interface).
  - Puertos SDI, SDO, SCLK y CS.
  - Velocidad máxima de reloj 20 MHz.

Tras esquematizar algunas de las características básicas del ADS 7953 se procederá a describir su funcionamiento. Como se extrae de la lista anterior, la interfaz digital del ADC se basa en el protocolo SPI. Éste se trata de un estándar de comunicación digital de tipo serie y síncrono de tipología maestro-eslavo. Concretamente, se trata de un estándar muy extendido cuya compatibilidad está prevista en la mayoría de plataformas digitales de tipo microprocesador, microcontrolador o DSP del mercado. En particular, la plataforma Zynq-7000 en la que se basará este proyecto posee dos periféricos SPI que permitirá la gestión de la interfaz con el ADC de manera simple, mediante el uso de drivers y librerías con funciones que Xilinx incorpora para la utilización de este periférico.

Básicamente, el estándar SPI hace uso de 4 pines: SCLK, MOSI, MISO y CS. La primera de ellas, SCLK, es la señal de reloj que impone la sincronía de la comunicación. Los pines MOSI y MISO son respectivamente los puertos de Master Out - Slave In y de Master In Slave Out. Estos pines sirven para la transferencia de información. El protocolo SPI es full-duplex en el sentido de que una transacción SPI se compone de dos envíos simultáneos de información. En este caso, el maestro de la comunicación envía al esclavo por el puerto MOSI al mismo tiempo que el esclavo envía datos al maestro por el puerto MISO. Por último, el pin de CS es un pin que maneja el maestro y que permite seleccionar el esclavo al que quiere transmitir, habitualmente a nivel bajo. Sirve también para indicar el comienzo de una trama SPI [60].

Como ventajas principales de este protocolo se pueden citar su velocidad, al permitir mayores velocidades que otras alternativas serie como el I2C, su flexibilidad en cuanto a que se pueden configurar parámetros de la trama como su longitud de bits o la polaridad y fase del reloj, o la simpleza del hardware necesario para implementar este protocolo. Por contra, es un estándar que requiere un gran número de líneas/pines que competidores como el I2C, no posee señal de acknowledgment desde los esclavos, no soporta de manera extendida los modos multi-maestro ni es capaz de gestionar la adición dinámica de nuevos nodos. Tampoco es un estándar que soporte grandes distancias en la comunicación [61].

Conociendo los conceptos básicos del protocolo SPI, se puede volver al estudio del funcionamiento del ADC. Como se puede apreciar en el diagrama de bloques funcional de éste (Figura 4.6), el ADS 7953 cuenta con un secuenciador de canales que le permite cambiar el canal al que apunta de manera secuencial, esto le permite ir midiendo un canal tras otro mientras por la interfaz digital va enviando la información correspondiente con el canal anterior. Adicionalmente, el ADC posee funciones de alarma mediante comparadores.

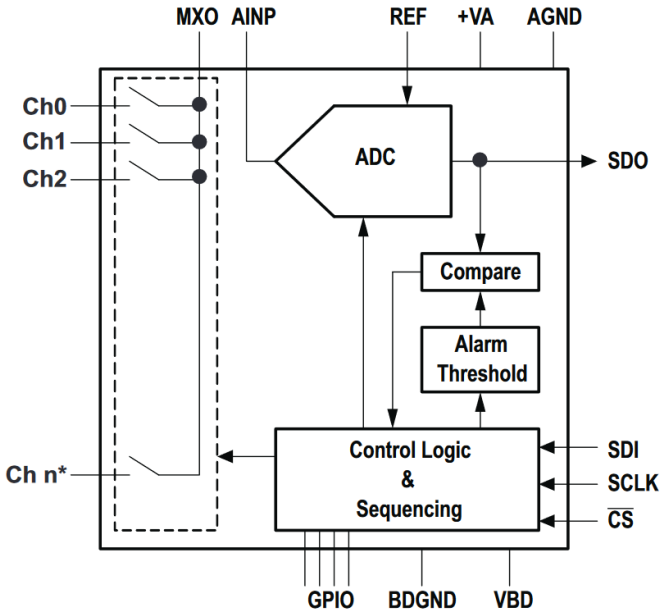

Figura 4.6 Diagrama de bloques funcional del ADS 7953 [58].

En cuanto a la secuencia de canales, el ADC admite dos modos de funcionamiento básicos: manual o automático. En el modo manual, el próximo canal a medir se programado en cada transacción SPI, al mismo tiempo que se lee la medida actual. En el modo automático, una secuencia de canales se deja pre-programada en el "registro de programa" de manera que el ADC conoce en todo momento cual es el siguiente canal a muestrear sin necesidad de indicarlo en cada trama. Existen dos modos automáticos de funcionamiento, el modo "Auto-1" permite programar una secuencia de canales en orden ascendente, mientras que el modo "Auto-2" permite dejar programado en el "registro de programa 2" el canal máximo a medir, de manera que la secuencia empieza en el canal 0 y va aumentando de uno en uno hasta este registro tope. Cabe destacar que los modos automáticos no ahorran el envío de la trama hacia el ADC ni permiten una conversión más rápida, ya que la trama maestro-esclavo se envía igualmente, sólo que ignorando los bits de canal. Por este motivo, se operará el ADC en modo manual por simplicidad y flexibilidad, centrando el estudio en la operación del ADC en este modo, en el que por defecto se inicializa el ADC en su encendido (apuntando al canal 0).

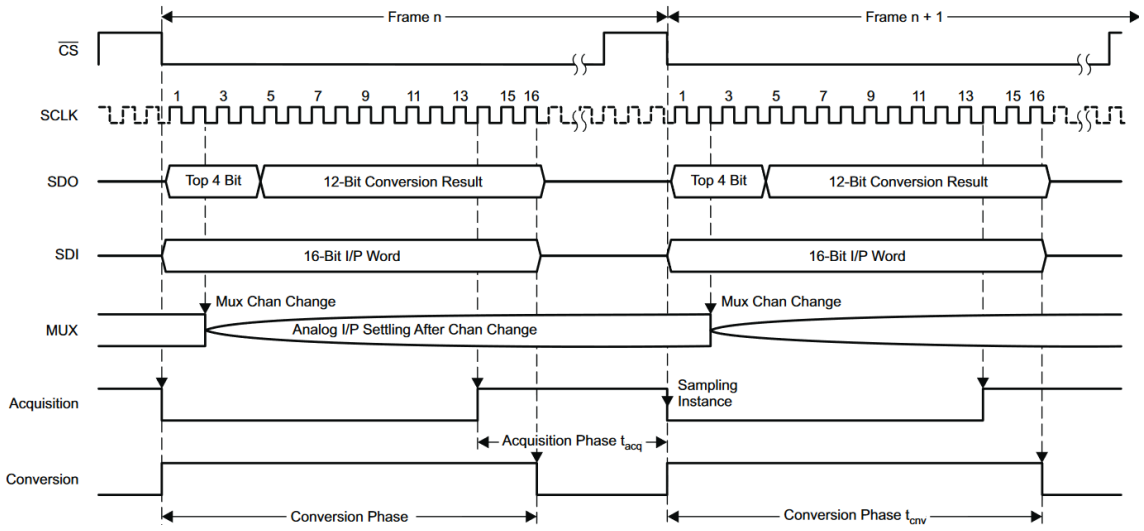

Figura 4.7 Diagrama temporal del proceso de conversión [58].

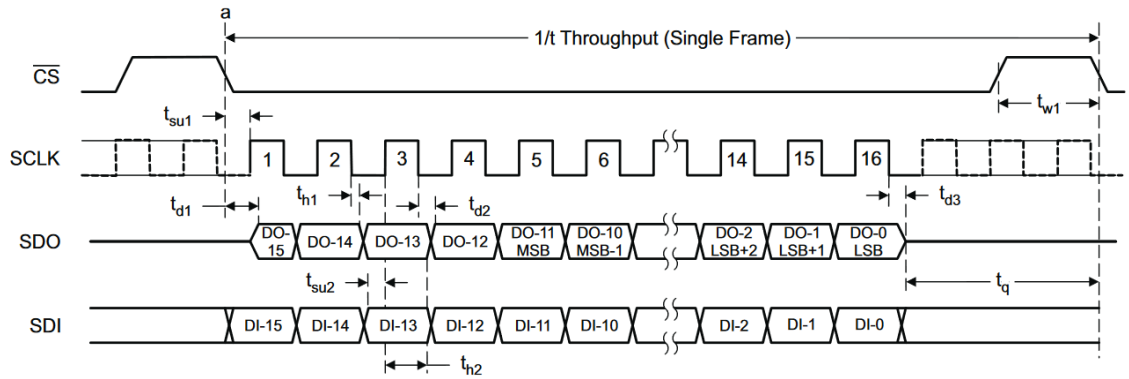

Figura 4.8 Diagrama temporal detallado de la interfaz digital [58].

Para mayor claridad, el proceso de conversión y gestión del ADC por SPI se aprecia en la Figura 4.7 y en la Figura 4.8. Como se puede observar, los procesos de adquisición y de conversión ocurren paralelamente a la transacción de datos a través de SPI. Una trama comienza con el paso a nivel bajo del pin CS. Cuando esto sucede, tanto los puertos SDO (MOSI) y SDI (MISO) comienzan a transmitir la trama de bits correspondiente. A través de SDI llega información con el próximo canal al que el multiplexor (MUX) debe apuntar y del modo de funcionamiento. En caso de haber programado un modo de funcionamiento automático, el multiplexor irá al canal pre-programado en la secuencia. Por el pin SDO, el ADC envía el resultado de la anterior conversión en formato 16 bits MSB-LSB, con el número de canal al que corresponde en los 4 bits MSB y el resultado de 12 bits en los 12 últimos bits de la transacción. El proceso de adquisición (HOLD de un canal) comienza automáticamente en el ciclo de reloj de la trama número 14 (se adquiere el canal al que apuntaba el MUX

antes del cambio de canal pedido en la trama actual) y acaba cuando llega un nuevo flanco de bajada de CS. En este momento, empieza la fase de conversión del canal adquirido y comienza el envío del canal que se convirtió en la trama anterior. En resumen, el dato de un canal pedido en la trama "n" estarán listos y se enviarán en la trama "n+2". Mientras que en la trama "n+1" el multiplexor apunta a ese canal y comienza el proceso de adquisición/conversión (Figura 4.9). En total, los procesos de adquisición y conversión duran en torno a 1 μs a la máxima velocidad de reloj de SCLK (20 MHz).

Conocido el funcionamiento del ADC y el formato de salida del mismo, resta conocer el formato que debe tener la trama de entrada al ADC que se envía por el pin SDI. Conocer la función de cada bit de esta trama de 16 bits es crucial para interactuar con el ADC correctamente y que las medidas se realicen de la manera en que son requeridas. En la Tabla 4.1 se recoge la estructura de esta trama.

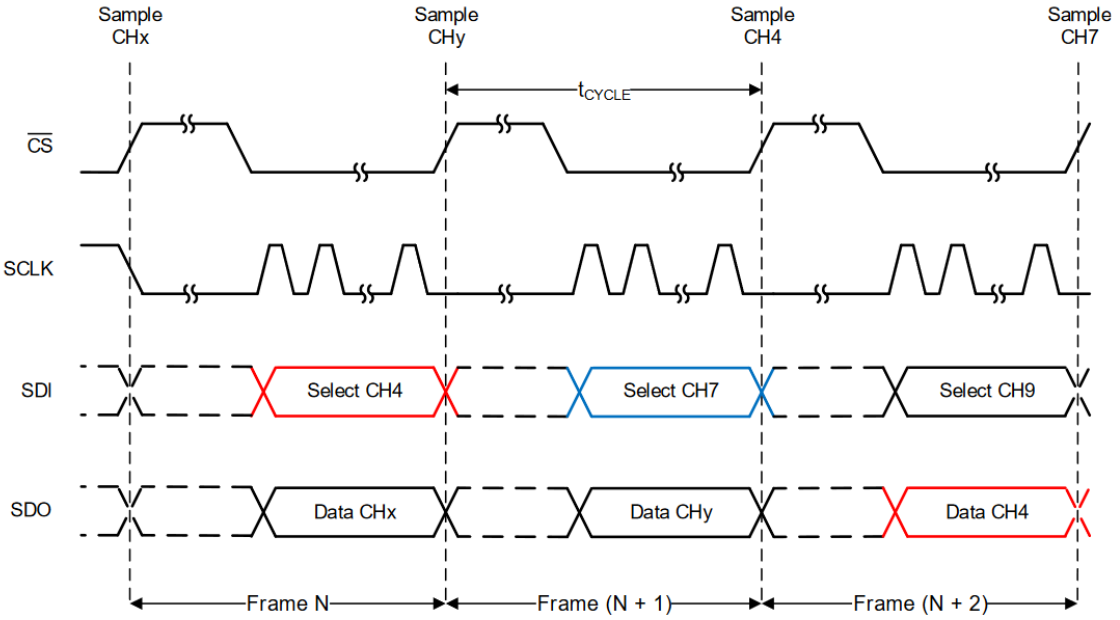

Figura 4.9 Diagrama temporal detallado de la secuencia de canales en modo manual [58].

Tabla 4.1 Estructura de la trama de entrada SDI para el modo manual [58].

Bits	Estado de Reset	Función
B15-12	0001	Selección de Modo: Manual (0001). Auto-1 (0010). Auto-2 (0011).
B11	0	Habilita programación bits B06-00: Habilitada (1). Deshabilitada (0).
B10-07	0000	Número de canal a ser seleccionado en la siguiente trama.
B6	0	Selecciona el rango de medida. 0 a Vref (0). 0 a 2Vref (1).
B5	0	Powerdown si 1.
B4	0	Selección 4 bits MSB de SDO. Canal (0). Valor de las GPIOs (1).
B03-00	0000	Valores para las GPIO (MSB-LSB)

4.3 Kit de evaluación ADS7953EVM-PDK

Tras un repaso de las características básicas del integrado del ADS 7953, se procederá a hacer lo propio para el kit de evaluación ADS7953EVM-PDK que el fabricante Texas Instruments comercializa para poder desarrollar aplicaciones con este ADC. Este kit de desarrollo soluciona toda la etapa de alimentación y permite el acceso a las diferentes funciones del ADC. Adicionalmente, este kit incluye otra placa auxiliar que permite la conexión a un PC para el testeo del ADC a través de la herramienta software "ADCPro".

Para el uso que se le dará en este trabajo, basta con conocer cómo configurar los jumpers que hay en la placa y el pinout de ésta, ya que el funcionamiento del ADC corresponde a lo descrito en el apartado anterior para el ADS7953. La configuración de los jumpers se recoge en la Tabla 4.2.

Tabla 4.2 Configuración jumpers placa ADS7953EVM-PDK [59].

Jumper	Config. Usada	Función
W1	Open	No utilizado en placas de producción.
W2	1-2	Selección de REFP. 2.5 V de la placa (1-2) ó externa en P1.18 (2-3).
W3	1-2	Selección de REFN. GND de la placa (1-2) ó externa en P1.20 (2-3).
W4	2-3	CH0 con filtro RC (1-2) o bypasseado (2-3).
W5	2-3	Selección pin CS. FSX (1-2) ó CNTL (2-3).
W6	2-3	Tensión DV_{DD}. +5 V (1-2) ó 3.3 V (2-3).
W7	1-2	Fuente de interrupción para placa auxiliar (no utilizado).

En lo que respecta al pinout de la placa, se tienen cuatro "headers" de pines. Los headers J1 en horizontal y P1 en vertical, ambos a la izquierda de la placa, contienen las entradas analógicas. El header P2 contiene los pines de la interfaz digital (a la derecha de la placa). El header P3 contiene los pines de alimentación de la placa (parte inferior). En la Figura 4.10 se muestra una imagen del kit de desarrollo junto con una placa de interconexión que permite la conexión por un lado de la placa de adaptación de medidas a los pines de entrada analógica del ADC y por otro lado la conexión de los pines de interfaz digital a los pines PMOD de la Zedboard. En concreto a los 3 puertos PMOD a la izquierda de la Zedboard. En particular la interfaz digital se conecta al puerto PMOD JE, que es el asignado al MIO de la parte PS. De esta manera, se podrá utilizar el periférico SPI del integrado Zynq-7000 sin necesidad de definir una interfaz EMIO.

Por otro lado, esta placa permite la conexión de una fuente de alimentación mediante un bloque terminal de alimentación y conecta la alimentación digital de la placa a los 3.3 V que la Zedboard suministra por los pines Vcc de los puertos PMOD. Adicionalmente, en esta placa se han instalado dos integrados SMP04 de Sample and Hold con 4 canales cada uno y alimentados a 15 V [62]. Estos integrados permiten hacer hold de las señales analógicas de entrada cuando comienza el proceso de medidas para eliminar el desfase que se produciría al medir de manera secuencial con el ADC. El manejo de estos integrados se produce con una única señal de S&H que al estar a nivel lógico alto permite el hold de las señales en los condensadores internos y en nivel lógico bajo permite que las señales sean sampleadas hasta que llegue una nueva orden de Hold.

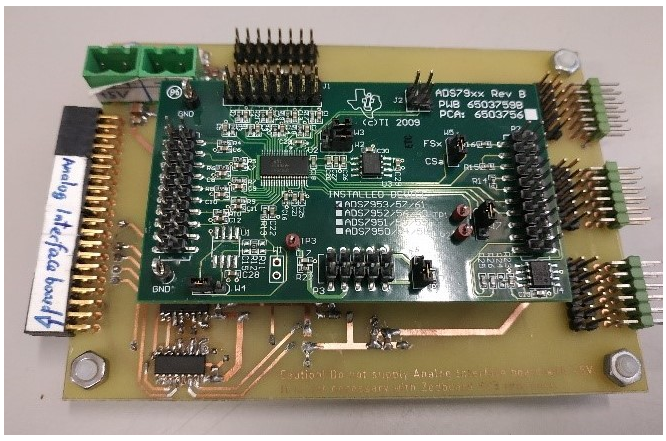

Figura 4.10 Kit de desarrollo ADS7953EVM-PDK con placa de interconexión para Zedboard y etapa de adaptación.

En detalle, el pinout de la placa se recoge en las tablas: Tabla 4.3, Tabla 4.4 y Tabla 4.5.

De todos estos pines, quedan cableados con la placa de interconexión realizada, los correspondientes al header P1. Es decir, con este montaje se tendrán totalmente accesibles ocho canales, seis de los cuales estarán asociados a canales de medida de magnitudes de alterna (con offset en placa de adaptación) y dos de ellos de continua (sin offset en placa de adaptación). En cuanto a la alimentación, no son necesarias todas las alimentaciones que se contemplan en la placa. Son estrictamente necesarias la alimentación analógica de +5VA, ambas tierras y la alimentación digital al nivel que se utilice la interfaz digital, que en este caso son los 3.3 V de la Zedboard. El resto son prescindibles y pueden permanecer sin conectar.

Tabla 4.3 Pines Alimentación de la placa ADS7953EVM-PDK [59].

Pin J3.x	Señal	Pin J3.x	Señal
J3.1	+VA (10 a 15 V)	J3.6	Analog GND (AGND)
J3.2	-VA (-10 a -15 V)	J3.7	+1.8 VD
J3.3	+5VA	J3.8	No utilizado
J3.4	-5VA	J3.9	+3.3 VD
J3.5	Digital GND (DGND)	J3.10	+5 VD

Tabla 4.4 Pinout interfaz Analógica de la placa ADS7953EVM-PDK[59].

Pin P1.x	Señal	Pin J1.x	Señal
P1.2	CH7	J1.2	CH15
P1.4	CH6	J1.4	CH14
P1.6	CH5	J1.6	CH13
P1.8	CH4	J1.8	CH12
P1.10	CH3	J1.10	CH11
P1.12	CH2	J1.12	CH10
P1.14	CH1	J1.14	CH9
P1.16	CH0	J1.16	CH8
P1.18	External REFN		
P1.20	External REFP		
Impares	Analog GND	Impares	Analog GND

Tabla 4.5 Pinout interfaz Digital de la placa ADS7953EVM-PDK [59].

Pin P2.x	Señal	Pin P2.x	Señal
P2.1	CNTL (CS si W5)	P2.11	SDI
P2.2	GPIO0	P2.12	GPIO3
P2.3	SCLK	P2.13	SDO
P2.4	DGND	P2.14	GPIO4
P2.5	CLK Out	P2.15	
P2.6	GPIO1	P2.16	
P2.7	FSX (CS si W5)	P2.17	
P2.8	GPIO2	P2.18	DGND
P2.9	FSR	P2.19	
P2.10	DGND	P2.20	

A continuación se muestra a modo de demostración el muestreo a 10 kHz de una señal senoidal de 50 Hz en los ocho primeros canales del ADC:

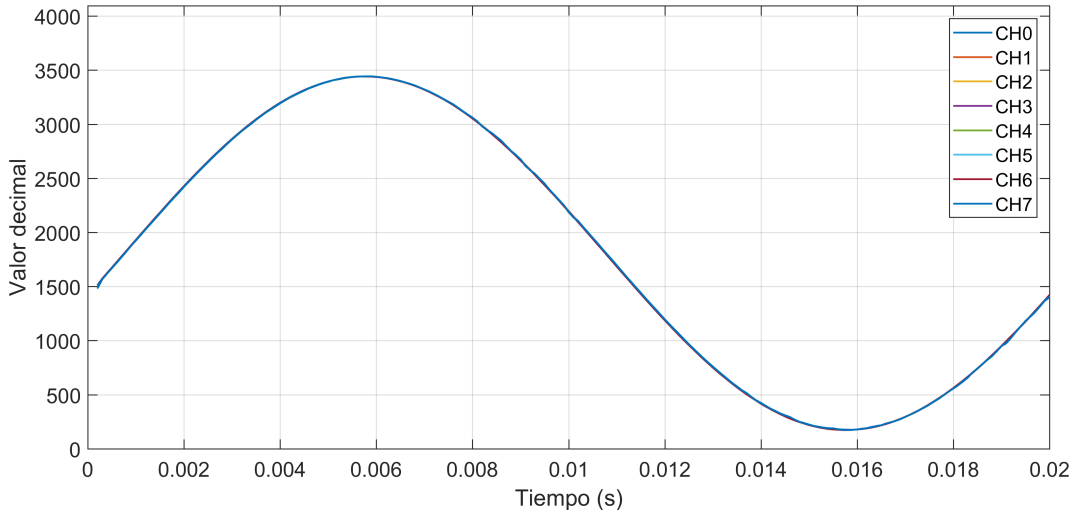

Figura 4.11 Señal senoidal de 50 Hz muestreada con el ADC.

5 Implementación del control en plataforma Zynq-7000

Este capítulo estará dedicado a la descripción de la arquitectura propuesta para implementar el control descrito en el segundo capítulo de este Trabajo Fin de Máster en una plataforma Zynq-7000. El diseño recogido en este capítulo será efectuado mediante las herramientas facilitadas por Xilinx. En particular, se utilizará el software Vivado Design Suite [63]. Como primer paso introductorio en este capítulo, se dedicará un apartado a describir brevemente el flujo de trabajo con este programa, desde el diseño de una aplicación hasta su ejecución en la placa. La cantidad de herramientas que este software pone a disposición del programador es enorme, por lo que se optará por hacer un breve repaso, focalizando la atención en aquellos puntos esenciales e imprescindibles para la ejecución de este proyecto. Posteriormente, se procederá a describir la solución propuesta, mostrando los puntos más importantes de la programación de los distintos elementos del sistema.

5.1 Vivado Design Suite

El flujo de trabajo propuesto por Xilinx en su herramienta Vivado Design Suite es bastante intuitivo ya que divide el diseño en dos grandes partes. Por un lado, mediante el software Vivado se puede diseñar la parte correspondiente a la lógica programable del FPSoC, incluyendo la programación de la FPGA y las configuraciones y parámetros básicos de los procesadores embebidos y sus periféricos. Este primer diseño, tras los procesos de síntesis e implementación, acaba con la generación del bitstream que contiene toda la configuración de la parte hardware. Este archivo, se puede exportar mediante Vivado, de manera que el diseño hardware pueda ser cargado por la segunda herramienta de Vivado: el SDK de Xilinx. Este programa es un entorno de programación software basado en Eclipse que está incluido en el Vivado Design Suite. Permite programar aplicaciones para ser ejecutadas por los microprocesadores. Así mismo, permite la programación de la placa, tanto de la parte FPGA como de los microprocesadores a través de JTAG, con capacidad de debug de las aplicaciones. También posee una herramienta para generar bootables a partir del diseño creado para el arranque a través de una tarjeta SD con un sistema operativo.

5.1.1 Programación hardware mediante Vivado

Una vez ejecutado el software Vivado Design Suite, el primer paso será crear un nuevo proyecto a través de la opción "Create Project" en la ventana "Quick Start". Se abrirá una nueva ventana donde es necesario indicar una serie de ajustes básicos para el proyecto. La configuración que aquí se usará es la de tipo de proyecto "RTL project", que creará un proyecto vacío donde se puedan ir añadiendo "sources" en algún lenguaje HDL y ejecutar los procesos de síntesis e implementación. En la siguiente ventana, es necesario elegir el chip o placa para el cual irá dirigido el diseño que se va a realizar. En este caso, lo más fácil es buscar la placa Zedboard en el apartado "Boards". Se elige la versión 1.4, que incluye los archivos de soporte última versión para la Zedboard modelo rev. D que se utilizará en este proyecto. Se incluye la Figura 5.1 para mostrar este proceso. Si se posee una revisión de la placa anterior, será necesario elegir la versión correspondiente. En caso de querer realizar un proyecto para una placa distinta y de no encontrar esta placa en las previstas en esta ventana, es posible que los "board files" necesarios estén en la página del fabricante. Para añadir los

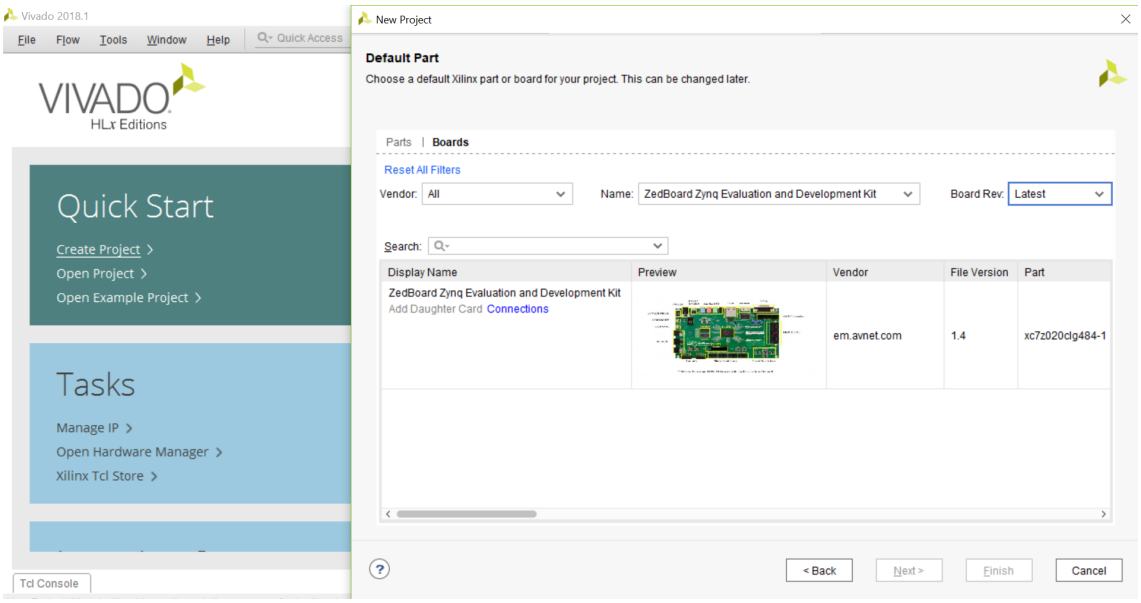

Figura 5.1 Creación de proyecto en Vivado. Elección de placa.

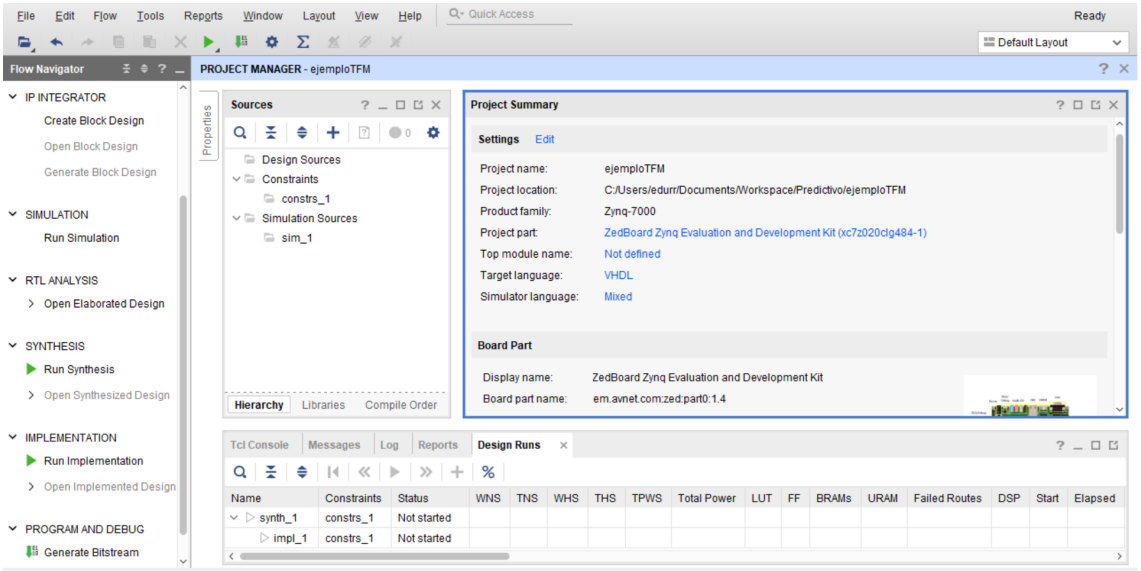

Figura 5.2 Ventana principal de Vivado (Versión 2018.1).

archivos necesarios para dar soporte a otra placa, se han de incluir los archivos descargados en la carpeta "*...Vivado*/2018.1/*data*/*boards*/*board_files*".

Una vez se han realizado todos estos pasos, se puede proceder a crear el proyecto, que ya contendrá todas las configuraciones básicas necesarias para la plataforma que se utilizará. Una vez creado, aparece la ventana principal de Vivado (ver Figura 5.2), donde se encuentran las principales herramientas para trabajar en el proyecto. En la columna de la izquierda, se encuentra el "Flow Navigator". Con las herramientas de esta columna, se pueden ir invocando los diferentes procesos necesarios para materializar el diseño:

- **IP INTEGRATOR.** La pestaña "IP integrator" permite crear y gestionar bloques de lógica que implementan una funcionalidad y se pueden añadir al diseño y trabajar con ellos modularmente como sistemas de "caja negra". Estos bloques pueden ser tanto creados por el usuario como importados de algún sitio ("IP cores").
- **SIMULATION.** Esta pestaña permite abrir la herramienta de simulación que incorpora Vivado. Mediante simulaciones, se puede verificar y comprobar el correcto funcionamiento a nivel lógico del diseño programado en HDL. Se puede ver los valores que las distintas señales del diseño van tomando a lo

largo del tiempo, según los estímulos que se hayan programado en el "test bench" elegido como "top level" de cara a la simulación.

- **RTL ANALYSIS.** La herramienta "RTL Analysis" permite un análisis básico del código HDL programado previo a los procesos de síntesis e implementación, que traducen el código realizado a recursos presentes en la parte lógica. Mediante esta herramienta se puede ver un primer esquemático con el diagrama de bloques que representa el diseño creado. Aunque no representa fielmente el circuito real que se acabará implementando en la FPGA tras los procesos de síntesis e implementación, a nivel visual es el más descriptivo y intuitivo. Permite una primera comprobación de como las herramientas de Xilinx están interpretando el código.
- **SYNTHESIS.** En esta pestaña, se tiene acceso a las herramientas de síntesis. Básicamente el proceso de síntesis se encarga de convertir el código HDL, que realiza una descripción RTL del diseño (se describe el flujo de datos y la lógica combinacional existente entre registros) a un netlist a nivel de puertas lógicas de acuerdo a una librería de primitivas (se pasa a un nivel de abstracción inferior). Una vez completado este proceso, se puede acceder al reporte de síntesis, que proporciona información útil sobre el timing, recursos utilizados, ruido o consumo del circuito sintetizado entre otros datos.
- **IMPLEMENTATION.** De manera similar a la pestaña anterior, esta herramienta da acceso al proceso de implementación. Éste se encarga de generar el diseño definitivo al traducir el netlist creado en síntesis al rutado exacto que se implementará en la FPGA seleccionada para el proyecto. Tras la finalización de este proceso, es posible acceder al reporte de implementación, que contiene información similar al de síntesis, pero más definitiva y con algunas optimizaciones extra que efectúa esta herramienta para el mapeado óptimo del circuito en la FPGA.
- **PROGRAM AND DEBUG.** Esta herramienta permite la generación del "bitstream", el cual es el archivo binario que realmente se programa en la FPGA.

En la parte central de la ventana, se tiene el "Project Manager". Aquí, se pueden añadir o crear los archivos fuente ("sources") al proyecto. Para este Trabajo Fin de Máster se utilizará lenguaje VHDL, por lo que los ficheros que se creen serán formato ".vhd". Estos se guardarán en la carpeta "Design Sources". Adicionalmente, en la carpeta "Constraints", se colocarán los ficheros ".xdc" de constraints. Tal y como se expuso en el capítulo tercero de este trabajo, estos ficheros sirven para codificar la localización física de cada uno de los puertos de entrada y salida del "top level" del diseño. También permiten incluir constraints de tiempo para las señales de reloj y otras directivas de síntesis. Para facilitar la confección de este fichero, se puede acudir al fichero maestro para la Zedboard, donde se definen todos los pines disponibles [55]. En la carpeta "Simulation Sources" se guardan los archivos fuente de los "test bench" de simulación creados. Para lanzar una simulación, es necesario elegir como top level el fichero con el "test bench" deseado. Esto se hace con la opción "Set as Top", que se encuentra en el menú desplegable que se abre al hacer click derecho en el fichero en cuestión.

En el cuadro inferior, se tiene acceso a una serie de pestañas. Por un lado, la consola ("Tcl Console") permite interactuar con el programa mediante comandos. Las pestañas "Messages", "Log" y "Reports" contienen información sobre los distintos procesos de síntesis e implementación realizados. Recogen avisos sobre posibles errores cometidos en el diseño e información sobre los pasos que sigue la herramienta a la hora de inferir el circuito a partir del diseño realizado. Esta información puede ser útil a la hora de depurar el diseño y optimizarlo. La pestaña "Design Runs" contiene una lista con todas las síntesis e implementaciones realizadas que han sido guardadas, con una serie de datos que resumen el resultado de las mismas.

Tras un repaso a las opciones básicas que presenta la interfaz de Vivado, se esquematizarán los pasos a seguir para, mediante las herramientas expuestas, llevar a cabo un diseño que se pueda ejecutar en la placa.

Definición del sistema procesador. Block Design

El primer paso para crear un diseño para una plataforma como la Zynq-7000 es el de definir el sistema procesador. Éste es el maestro en el arranque de la placa y debe instanciarse en el proyecto. La manera más inmediata de hacer esto es mediante la herramienta "Block Design" que se encuentra en la pestaña "IP Integrator". Esta herramienta permite crear un diseño de bloques donde, por ejemplo, combinar e interconectar el microprocesador con otros bloques de lógica. Estos bloques se pueden crear desde Vivado, aunque Xilinx también ofrece multitud de ellos en sus librerías.

Para añadir un bloque, hay que hacer click en "Add IP" y buscar en la lista desplegada el "IP core" requerido. Como mínimo, cualquier diseño debe de incluir el bloque "ZYNQ7 Processing System", por lo que se añade al diseño. Aparece un bloque con una serie de entradas y salidas a las que se pueden conectar otros bloques

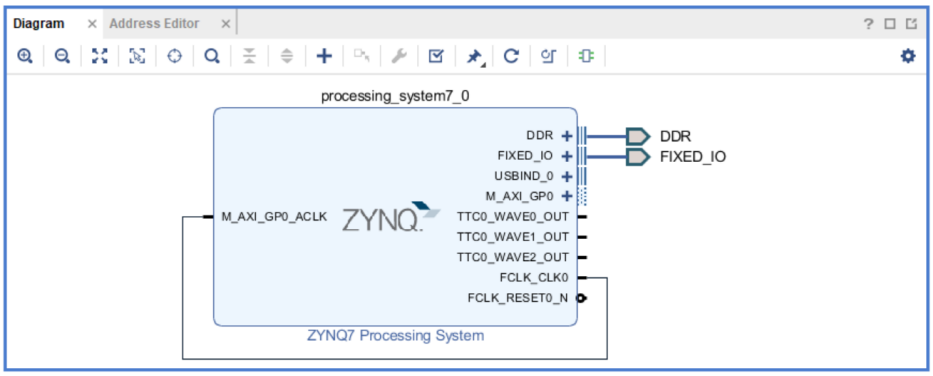

Figura 5.3 "Block Design" con bloque "ZYNQ7 Processing System".

de lógica programable. Haciendo click en el bloque que aparece, se accede a la ventana de configuración del sistema procesador, donde se pueden configurar numerosos aspectos, como los relojes, los periféricos o los puertos MIO y EMIO que se describieron en el capítulo 3 de este proyecto. La mayoría de parámetros vienen configurados de manera correcta para la placa que se seleccionó en la creación del proyecto, por lo que no es necesario tener que reconfigurar todos y cada uno de ellos. Tan solo aquellos que realmente sean requeridos para obtener una funcionalidad específica. Por ello, para este primer ejemplo no es necesario cambiar nada. En el diagrama de bloques creado, aparece la opción mediante un cuadro verde de efectuar la conexión automática de ciertos puertos del diseño. Efectuando esta operación, se puede observar como la herramienta crea de manera automática puertos para las conexiones DDR y FIXED_IO (ver Figura 5.3). La primera es la correspondiente a la memoria DDR3 que incluye la Zedboard, mientras que la segunda se corresponde con los puertos MIO, que se pueden reconfigurar en el bloque procesador. Estos pines en particular no se tienen que asociar en el fichero de "constraints", ya que la herramienta asigna automáticamente su localización física según la placa seleccionada en el proyecto. El resto de pines del bloque procesador, entre los que se encuentran salidas de timers, reset o buses de comunicación pueden quedar abiertos, salvo la entrada M_AXI_GP0_ACLK, que requiere ser asociada a una señal de reloj para poder validar el diseñó. Se cablea manualmente, clickando y arrastrando con el ratón, con la señal FCLK_CLK0.

Una vez realizado el diseño, se puede proceder a su validación mediante la opción "Validate Design". Si el diseño resulta validado con éxito, puede cerrarse la ventana de "Block Design", lo cual permite volver al "Project Manager". Desde esta ventana, se puede observar como la jerarquía del proyecto se ha actualizado apareciendo a la izquierda el diseño realizado en diagrama de bloques bajo la carpeta "Design Sources".

Creación de un top level. Generación del "bit stream"

Para poder utilizar este diseño, es necesario crear lo que se conoce como un "wrapper". Esto se trata de un top level codificado en VHDL que simplemente sirve de envoltura a este diseño. Para poder agilizar el proceso, Vivado ofrece la posibilidad de generar un "wrapper" automáticamente, haciendo click derecho en el diseño de bloques generado. Mediante la opción "Create HDL wrapper", se genera un fichero ".vhd" en el que se define el bloque como un componente y en el que se definen las entradas y salidas de éste como puertos de la entidad generada. Como es el único fichero VHDL en el proyecto, por defecto es seleccionado como el top level global del mismo. Es perfectamente posible definir otro fichero distinto como top level e instanciar como componente dentro de éste el "wrapper" que se ha generado. Igualmente, es posible definir más bloques de lógica en VHDL que se definan como componentes dentro del top level y que se conecten mediante señales a los puertos del "wrapper", pudiendo conectar de esta manera la parte del sistema procesador con la parte de la lógica programable.

Sin embargo, con el objetivo de demostrar la utilización de las herramientas, basta con el diseño planteado hasta ahora, pudiendo proceder a ejecutar los procesos de síntesis, implementación y generación del "bitstream" que aparecen en la columna de "Flow Navigator". En caso de clickar este último, la herramienta ejecutará automáticamente los tres procesos en ese orden, ya que sus resultados son necesarios para el siguiente.

Finalmente, estos tres procesos deben de finalizar sin ningún tipo de error, resultando el "bitstream" que se escribirá en la placa. En este punto se pueden consultar los distintos reportes de síntesis e implementación y consultar los distintos parámetros y resultados de estos procesos. Para este primer ejemplo, no poseen un gran interés, por lo que el paso siguiente será la exportación del diseño al SDK. Para ello, en el menú "File->Export" se debe de hacer click en la opción "Export Hardware...", marcando el cuadro de "Include

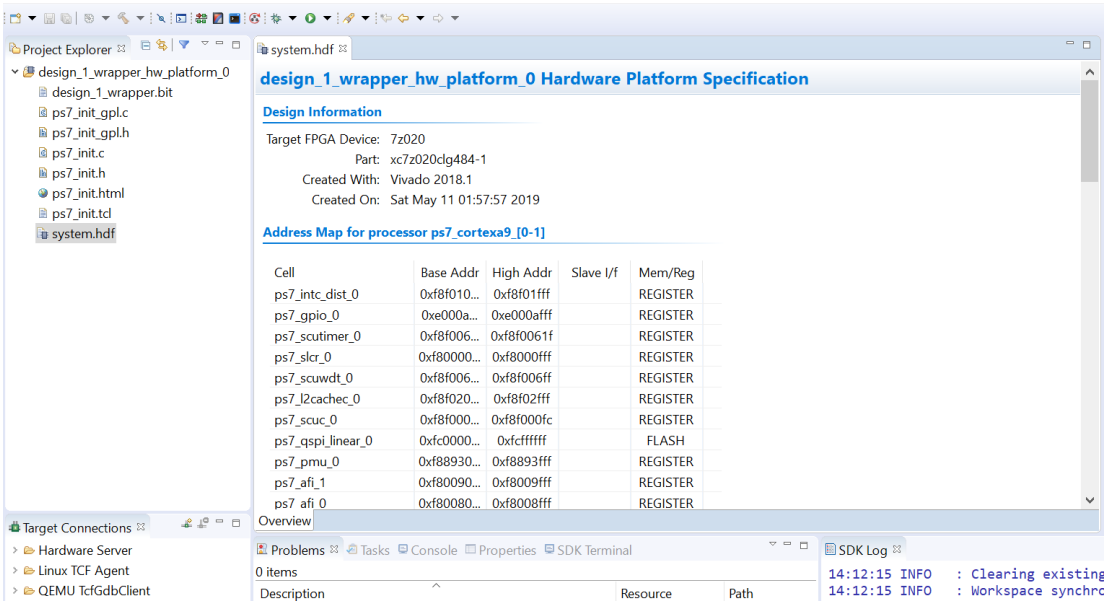

Figura 5.4 Ventana principal del Xilinx SDK.

Bitstream". Efectuando este paso, el hardware diseñado estará disponible para ser leído desde el SDK. A continuación, es posible abrir esta herramienta mediante el menú "File->Launch SDK". Esta opción abrirá el SDK de Xilinx, que cargará el diseño hardware creado de manera automática.

5.1.2 Programación software mediante Xilinx SDK

El software Xilinx SDK es un programa para el desarrollo de aplicaciones software basado en el conocido entorno de programación Eclipse. Por este motivo, estas líneas no tendrán como objetivo el estudio de todas y cada una de las funciones que esta herramienta ofrece, sino que simplemente se repasarán aquellos pasos o conceptos particulares que son necesarios de cara a diseñar y ejecutar una aplicación para una plataforma como la Zynq-7000.

Hardware Platform. Programación de la FPGA

Tras aplicar los pasos del apartado anterior y lanzar el SDK de Xilinx, se abre la ventana principal del entorno (Figura 5.4). Automáticamente, el programa ha cargado en el "Project Explorer" el diseño hardware que se exportó bajo el nombre de "design_1_wrapper_hw_platform_0". Este proyecto contiene información sobre cómo se ha configurado el hardware en Vivado y contiene una lista de todos los periféricos habilitados con su dirección en el mapa de memoria. Adicionalmente, se puede observar que contiene el fichero ".bit" generado en Vivado. En caso de volver a Vivado y cambiar partes del diseño hardware, al exportar el nuevo diseño la herramienta será por lo general capaz de actualizar esta plataforma de manera automática. Aunque dependiendo de la magnitud de los cambios, habrá ocasiones donde la herramienta cree un nuevo proyecto "...hw_platform" para el nuevo diseño.

En este punto, se podría programar la placa con el hardware diseñado, mediante la opción en la barra de herramientas "Program FPGA". Esto escribe el bitstream mediante el puerto JTAG a la FPGA. De esta forma, el diseño hardware quedaría programado, y el SoC configurado tal y como se ha especificado en Vivado. Sin embargo, los núcleos ARM no estarían ejecutando ninguna aplicación. Para ello, hay que crear un proyecto de aplicación y lanzarlo a través del JTAG.

Board Support Packages. Drivers y librerías

Antes de crear una nueva aplicación es necesario hacer un paso intermedio, crear un "Board Support Package" o BSP. Un BSP contiene todos los drivers y librerías de funciones que facilitan el acceso al hardware configurado. El SDK es capaz de generar uno automáticamente a partir de la plataforma hardware creada mediante el menú "New -> Board Support Package". Se abre una ventana de diálogo donde es necesario indicar la plataforma hardware que se utilizará como base, el núcleo al que irán dirigidas las aplicaciones creadas con este BSP y si serán aplicaciones "standalone" ("baremetal") o se ejecutarán en un sistema operativo embebido. En la siguiente ventana se puede comprobar la versión de los distintos drivers utilizados

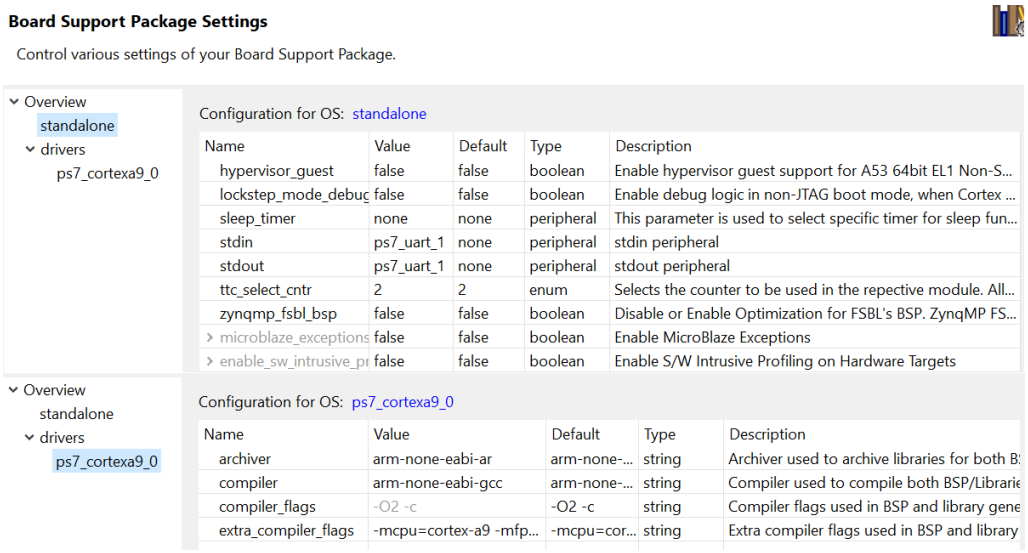

Figura 5.5 Configuraciones por defecto del "Board Support Package".

para cada periférico configurado en el diseño hardware. También se tiene acceso a algunas directivas de compilador, pulsando sobre la pestaña "drivers -> ps7_cortex9_0" (ver Figura 5.5). Para un ejemplo sencillo no es necesario cambiar nada, por lo que se puede proceder a generar el BSP pulsando sobre "Ok".

Una vez generado el BSP, aparece un nuevo proyecto en el "Project Explorer", con el nombre que se le asignó en su creación. Si se exploran las carpetas y archivos de este proyecto, se pueden ver bajo la carpeta "include" todos los archivos de cabecera ".h" de los drivers generados. Bajo la carpeta "libsrc" se encuentran los ficheros ".c" correspondientes con todas las funciones que facilitarán el desarrollo de las aplicaciones en código C, permitiendo la configuración e interacción con los periféricos de manera modular e intuitiva. Por ejemplo, en la carpeta "gpiops_v3_3" se encuentran funciones como "XGpioPs_CfgInitialize", "XGpioPs_Read" o "XGpioPs_Write". Éstas permiten la configuración, lectura y escritura de aquellos pines del MIO que están definidos como GPIO (configurable en el bloque PS en Vivado).

Creación y ejecución de una aplicación

Con todo ello, es ya posible crear la primera aplicación. Esto se hace mediante el menú "New -> Application Project", lo cual permite la aparición de la ventana de la Figura 5.6. En esta ventana, se deben de seleccionar la plataforma hardware y el BSP para el que se va a diseñar esta aplicación. También se debe elegir el tipo de lenguaje de programación (C ó C++) que se utilizará y el núcleo al que irá dirigida la aplicación. Configurados estos parámetros, la siguiente ventana permite elegir si el proyecto creado estará vacío o si seguirá algún tipo de plantilla de ejemplo. Una vez seleccionado el tipo de plantilla, se genera el proyecto de aplicación. La estructura de éste es la que se puede encontrar habitualmente en este tipo de entornos, con las carpeta "Binaries" que contienen los ejecutables ".elf" generados en la compilación, la carpeta "includes" con las librerías que se están usando en el proyecto, la carpeta "Debug" con los productos que genera el modo debug y la carpeta "src" con los archivos fuente. En esta carpeta se encuentra también el linker script "lscript.ld". Éste sirve para modificar los sectores de memoria presentes en la plataforma en los que se pueden ubicar partes del ejecutable, pudiendo también mapear las distintas secciones a la región de memoria que se desee.

Una vez codificada la aplicación, la manera de proceder es similar a la de otros entornos de programación basados en Eclipse. Se puede compilar la aplicación, debugear y ejecutarla en la plataforma. Es importante que la parte hardware se haya programado anteriormente mediante la acción "Program FPGA". Haciendo click derecho en el proyecto de aplicación, se obtienen las opciones "Run As" y "Debug As" que permiten lanzar la aplicación en el hardware y debugearla.

5.2 Arquitectura básica propuesta para el FPSoC

Hasta este punto, se ha expuesto el procedimiento básico para crear un diseño hardware sencillo y programar una aplicación para ser ejecutada por un núcleo del sistema procesador mediante las herramientas proporcionadas por Xilinx. Sin embargo, la plataforma Zynq-7000 tiene un mayor potencial, como por ejemplo la posibilidad de trabajar con un sistema operativo embebido o la capacidad de ejecutar aplicaciones

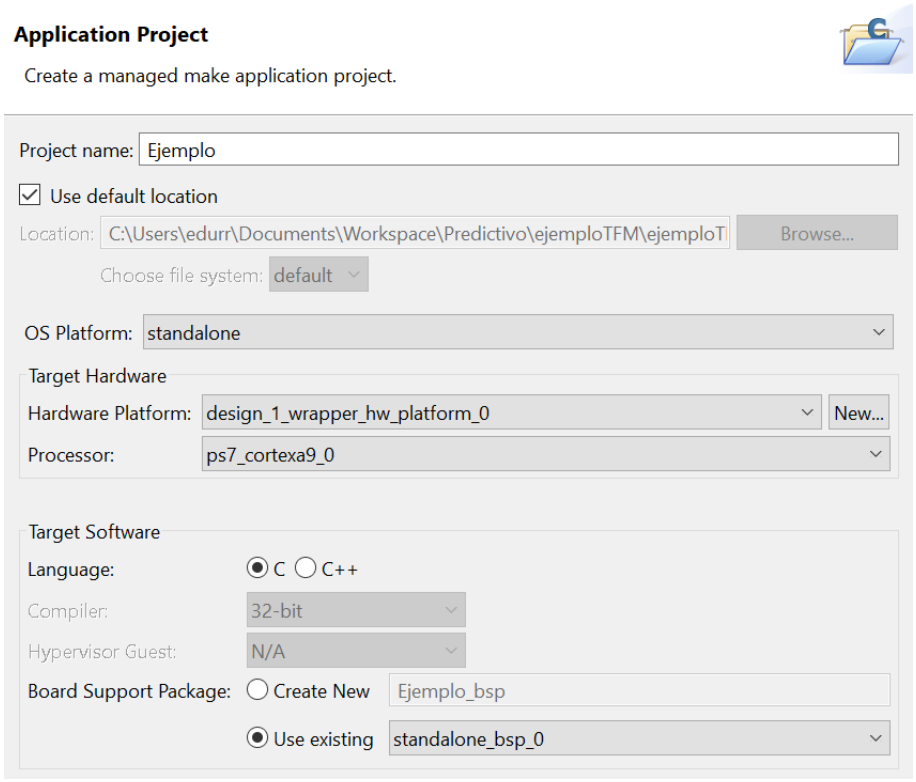

Figura 5.6 Creación de un proyecto de aplicación.

independientes en cada uno de sus núcleos. Por ejemplo, para poder ejecutar un sistema operativo embebido es necesario enfocar el problema de manera distinta. En este caso, la manera de abordar el problema es grabar en la tarjeta SD la imagen del sistema operativo, y todos los archivos necesarios para que la Zedboard arranque automáticamente leyendo la SD y cargando el sistema operativo y el bitstream diseñado.

Antes de seguir explorando cómo programar la plataforma, el primer paso a realizar es decidir el tipo de arquitectura que se quiere implementar en el sistema. Como se describió en la introducción de este proyecto, el objetivo es el de utilizar los tres grandes recursos presentes en el sistema de manera que trabajen independientemente ejecutando tareas diferentes. La arquitectura que se ha decidido implementar es la que se recoge en la Figura 5.7. En esta figura se puede apreciar como todos los cálculos del algoritmo de control FCS-MPC recaen sobre la FPGA, que absorbe de esta manera el grueso de la complejidad computacional. Así mismo, se encarga de gestionar los disparos de los IGBT del convertidor, ya que el estado óptimo se calcula en la FPGA. Además, como se vio en el capítulo tercero de este proyecto, la parte PL tiene acceso a un mayor número de pines, en concreto los correspondientes a los puertos PMOD, con cuatro de ellos por sólo un puerto accesible por MIO desde la PS.

Por otro lado, el núcleo 1 se encargará de ejecutar una aplicación "standalone" que se encargue de gestionar el ADC externo descrito en el capítulo cuarto a través de uno de los periféricos SPI de la Zynq-7000. La idea es programar una interrupción que salte periódicamente y empiece a disparar las medidas.

La temporización del sistema podría llevarse a cabo a través de uno de los timers en el Zynq-7000. Sin embargo, como la FPGA es la encargada de gestionar los disparos, y es importante que las conmutaciones se produzcan en el momento exacto, se decide que sea la FPGA la que se encargue de la temporización del sistema mediante un bloque contador. Este bloque señala el comienzo de cada intervalo de control, momento en el cual se lanza una interrupción desde la parte PL a la parte PS. Esta interrupción es la que gestiona la lectura periódica del ADC.

El trasvase de información entre el ARM1 y la FPGA se realiza a través de la memoria compartida, de manera que la FPGA tiene acceso a las medidas y el procesador puede acceder a variables internas de la FPGA cableadas a la memoria. La metodología seguida para esta compartición de memoria se describirá más adelante con mayor profundidad.

Hasta este punto, se han descrito tareas que son críticas y tienen que realizarse en tiempo real para controlar el sistema. Por ello, su asignación tanto a la FPGA como a uno de los núcleos ARM a través de una aplicación

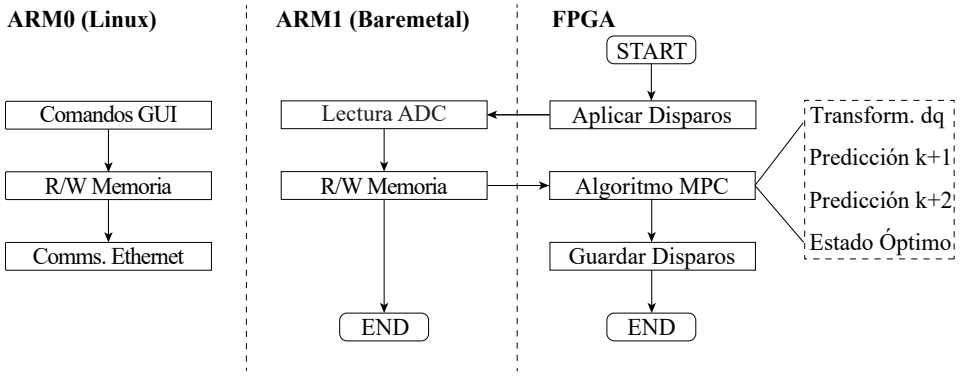

Figura 5.7 Diagrama de flujos con la división de tareas propuesta.

standalone, es apropiada, ya que la gestión del tiempo y la prioridad de tareas no son transparentes al usuario en estos casos. De este modo, la ejecución en tiempo real está asegurada siempre que se respete el tiempo de cálculo necesitado para medidas y algoritmo de control. No habrá ninguna tarea no prevista por el programador que salte y quite tiempo a las tareas programadas.

Este no es el caso para la monitorización del sistema. En este caso, se decide utilizar el ARM0 para la ejecución de un sistema operativo que gestione las comunicaciones con el exterior por un puerto Ethernet. La idea es que mediante una interfaz de usuario ejecutada en el PC, poder enviar comandos a la plataforma de control y que ésta pueda enviar datos de las distintas variables y magnitudes para ser representadas gráficamente en la interfaz de usuario. La ventaja de contar con un sistema operativo para esta operación es que se facilita la gestión de las comunicaciones gracias a soluciones de más alto nivel que hay presentes en un SO. Adicionalmente, contar con un SO basado en Linux como el que se instalará, abre la puerta a explorar desarrollos más avanzados en un futuro que permitan expandir la funcionalidad del sistema más allá de lo alcanzable por un sistema embebido baremetal.

Mediante la arquitectura propuesta, que se estudiará con mayor detalle en los siguientes apartados, se consigue explotar todos los recursos presentes en el sistema, asignando todas las tareas que se quieren llevar a cabo según su complejidad de calculo y prioridad. El objetivo de subsecuentes apartados será explicitar cómo materializar esta arquitectura y demostrar mediante resultados experimentales su viabilidad.

5.3 Petalinux OS

El primer paso a la hora de exponer la arquitectura propuesta será describir qué es necesario hacer para poder compilar y arrancar un sistema operativo en la Zedboard. En el proceso de arranque se programa la parte PL a través del bitstream, se arranca la aplicación del otro núcleo, y se deja el sistema operativo funcional para poder interactuar con él a través del puerto serie y ejecutar aplicaciones de Linux.

En concreto, una distribución de Linux suministrada por Xilinx llamada Petalinux será la elegida para este proyecto [64]. Se pueden encontrar compilaciones de este sistema operativo ya realizadas por Xilinx, y actualizadas según las correspondientes versiones del Vivado Design Suite [65]. Éstas pueden ser útiles a modo de demostración para probar alguna aplicación sencilla, pero para implementar la arquitectura que se ha expuesto en el anterior apartado será necesario hacer nuestra propia compilación de este SO con algunas modificaciones.

Básicamente, para la arquitectura propuesta, necesitamos configurar el sistema en el modo conocido como "Asymmetric Multiprocessing" (AMP) [66]. Este modo permite a cada núcleo ejecutar "stacks" independientes, de manera que coexisten dos ejecutables, cada núcleo operando uno de ellos, y que tienen la capacidad de compartir información a través de los recursos compartidos de la Zynq-7000. Este modo permite entre otras cosas, como es el caso de este proyecto, que un núcleo se pueda dedicar a la ejecución de un sistema operativo mientras que el otro núcleo ejecuta una aplicación baremetal.

Dos documentos serán necesarios para poder realizar las modificaciones necesarias para llevar a cabo los pasos necesarios para funcionar en este modo. Por un lado, la Wiki de Xilinx recoge los pasos detallados de cómo realizar una compilación personalizada del SO a través de las Petalinux Tools [67]. Por este motivo, el objetivo de este apartado no será repetir este tutorial paso por paso, sino señalar aquellos puntos relevantes y las modificaciones particulares requeridas para este proyecto. Por otro lado, el ya referenciado documento [66] se trata de una nota de aplicación de Xilinx donde se explica qué es necesario para poner en marcha un sistema

en modo AMP. Más allá de las guías básicas de este ejemplo, será necesario hacer algunas modificaciones extra para este proyecto.

Lo primero a tener en cuenta es que las Petalinux Tools sólo funcionan en Linux, por lo que para hacer las operaciones necesarias de cara a obtener una compilación personalizada de Petalinux, es necesario instalar el Vivado Design Suite en alguna de sus respectivas versiones junto con las Petalinux Tools correspondientes [64].

El primer paso será elegir la versión de Petalinux con la que se va a trabajar. La primera opción que se consideró para este proyecto es utilizar la versión 2018.1, que es la correspondiente a la versión de Vivado con la que se empezó a diseñar este proyecto. El principal problema que se encontró con esta versión es que el sistema de archivos que utiliza Xilinx cambió a partir de la versión 2017.1. En particular, se pasó del sistema "Legacy" al sistema "Flattened Image Tree", introducido en el kernel 2.6 de Linux, extendiendo la compatibilidad a un mayor número de sistemas e introduciendo mejoras de seguridad [68]. A nivel práctico, básicamente cambia la forma en la que se "wrappea" la imagen del sistema operativo. La problemática en este caso es que casi todos los tutoriales existentes estaban escritos con el anterior sistema de archivos como referencia. Este hecho dificultó su puesta en marcha de manera correcta y no se consiguió realizar una compilación del sistema en esta versión que funcionara bien, surgiendo habitualmente problemas en el proceso de arranque.

Otra opción que se consideró es utilizar la versión 2014.4. Para esta versión se pueden encontrar más referencias, como el TFG de Ángel Ruiz Martínez que desarrolló un sistema de adquisición de medidas con esta placa y un sistema Petalinux 2014.4 [69]. El principal problema que se encontró con esta versión es que las nuevas versiones de Vivado incorporan el compilador hard-float para aplicaciones de ARM, mientras que las versiones antiguas utilizaban el compilador soft-float. Por este motivo, cualquier aplicación diseñada en una versión de Vivado como la 2018.1 da error al intentar ser ejecutada con una distribución de Petalinux compilada en la versión 2014.4.

Finalmente, se optó por la versión 2016.4. Esta versión todavía utiliza el sistema de archivos anterior, por lo que pudo compilarse sin problemas y arrancar sin ningún tipo de error. Adicionalmente, el compilador hard-float ya había sido incorporado por parte de Xilinx a Vivado, por lo que las aplicaciones diseñadas para versiones más recientes funcionan sin problemas.

A continuación, se listarán qué archivos son necesarios copiar a la raíz de la tarjeta SD para poder arrancar el sistema desde la Zedboard (ver Figura 5.8). De estos archivos, algunos se tendrán que generar y otros se pueden utilizar los aportados por Xilinx en la compilación por defecto de Petalinux 2016.4 que se puede encontrar en [65].

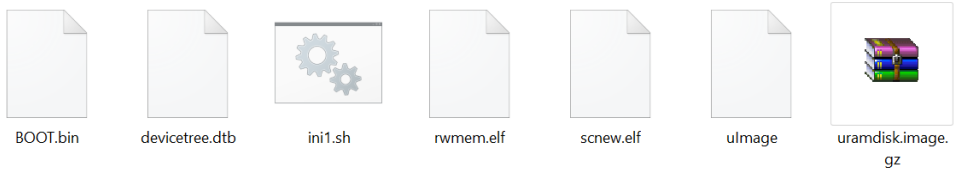

Figura 5.8 Archivos en la raíz de la tarjeta SD.

- **BOOT.bin**. Este es el archivo de arranque. Será lo primero que lea la tarjeta al encender. Contiene los archivos necesarios para bootear el sistema operativo y programar tanto la parte hardware como el otro núcleo si fuera necesario. Por tanto, si se cambia el diseño será necesario generar uno nuevo. El proceso de generación de este archivo se realiza desde el SDK de Xilinx en el que se ha diseñado el proyecto. Esto se hace desde el menú "Xilinx->Create Boot Image" (ver Figura 5.9). En esta ventana, se han de indicar en el cuadro "Boot Image partitions" los distintos archivos, en orden, que el bootable debe de ir cargando. El primero de ellos es el bootloader o FSBL, seguido por bitstream del diseño hardware, el archivo u-boot y el ejecutable de la aplicación compilada para el segundo núcleo. Para agilizar el proceso, se puede crear un archivo BIF que recoja todos los archivos que debe contener el bootable:

Código 5.1 Archivo BIF.

```
//arch = zynq; split = false; format = BIN
the_ROM_image:
```

```
{
[bootloader]C:\Users\edurr\Desktop\Saves_ZedLinux\pruebas_boot\fsbl.
    elf
C:\Users\edurr\Desktop\Saves_ZedLinux\pruebas_boot\toplevel.bit
C:\Users\edurr\Desktop\Saves_ZedLinux\pruebas_boot\u-boot.elf
C:\Users\edurr\Desktop\Saves_ZedLinux\pruebas_boot\appcpu19.elf
}
```

A continuación se procederá a explicar cómo crear los respectivos archivos del BOOT.bin.

- **FSBL ("First Stage Boot Loader")**. Ésta se trata de la primera aplicación que ejecuta el ARM0 al arrancar. Se encarga de programar la parte hardware y de escribir en la memoria RAM los sucesivos ejecutables en formato .elf que se han incluido en el BOOT.bin. Para obtener este ejecutable se debe de crear una aplicación con el ARM0 como target y con el diseño hardware creado como plataforma base. En la ventana donde se pueden seleccionar ejemplos de aplicación, elegir la opción "Zynq FSBL". Una vez creada la aplicación, basta con compilarla y extraer el ejecutable .elf de la carpeta "Debug". En versiones anteriores de Vivado era necesario utilizar un BSP modificado para habilitar el modo AMP, aunque en las versiones recientes no es necesario.
- **toplevel.bit**. Se trata del bitstream con el diseño hardware. Se puede obtener siguiendo los pasos del primer apartado de este capitulo.
- **u-boot.elf**. Se encarga de arrancar el sistema operativo Linux, en este caso en el ARM0. Se puede utilizar el suministrado en las compilaciones por defecto ofrecidas por Xilinx en su Wiki.

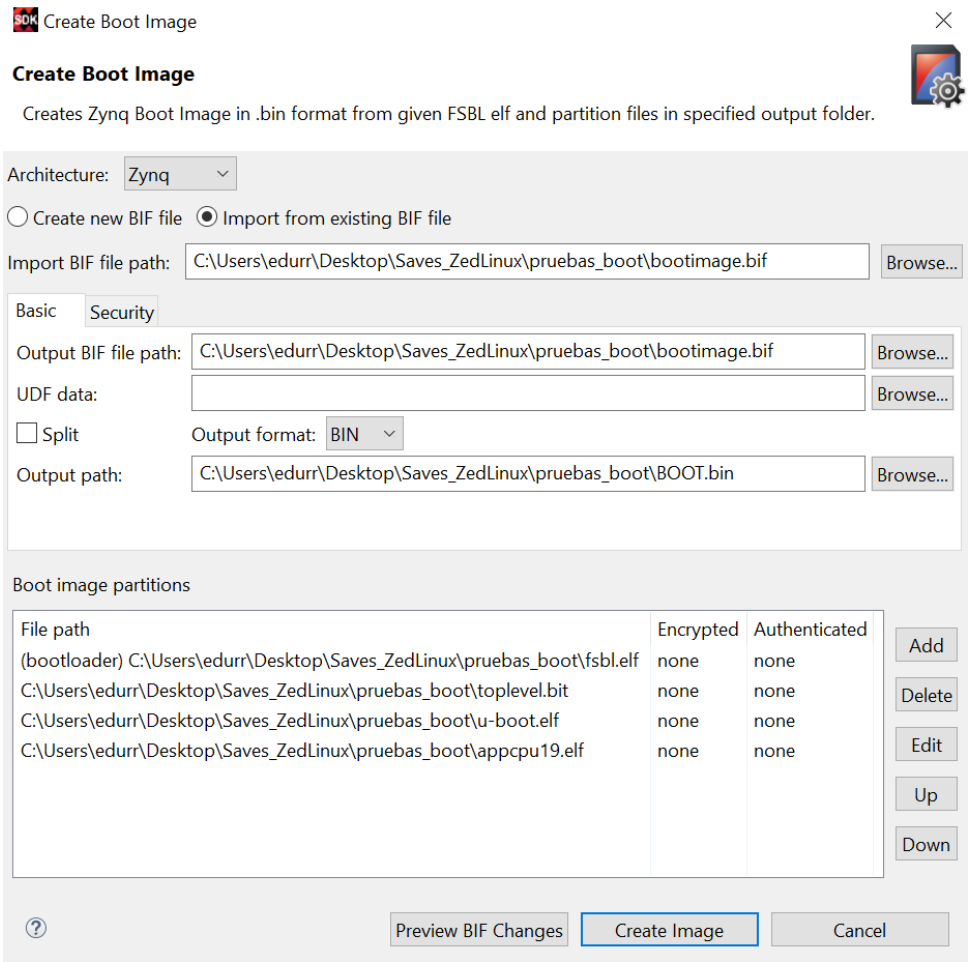

Figura 5.9 Ventana de creación del BOOT.bin.

  - **appcpu19.elf**. Es el ejecutable de la aplicación compilada para el ARM1. Es importante realizar una serie de modificaciones en la creación de esta aplicación para que el modo AMP pueda funcionar correctamente:
    * Indicar ARM1 como target.
    * En la creación del BSP, seleccionar BSP OS como standalone y añadir en el apartado extra_compiler_flags el siguiente comando: -DUSE_AMP=1.
    * Es necesario editar el "linker script" (lscript.ld) para limitar los sectores de memoria en los que se podrán escribir las diferentes secciones del ejecutable generado. En particular, es importante que el ejecutable del ARM1 no se escriba en partes de memoria a las que accederá el sistema operativo. De esta manera, se decide seccionar la memoria en dos partes. El SO tendrá acceso desde la dirección 0x0 hasta la 0x18000000 (esto se limita en el devicetree), mientras que en el lscript.ld se limita el acceso de la aplicación desde la dirección inicial 0x18000000. Se le asigna un tamaño total de 0x1FF00000.

- **devicetree.dtb**. Es el archivo que describe el hardware disponible al kernel de Linux y cómo comunicarse con él. Es necesario modificarlo para asegurar que no se produzcan interferencias a la hora de utilizar los periféricos por parte de los dos núcleos. El procedimiento para generar un devicetree customizado puede encontrarse en [70]. Resumidamente, el proceso consiste en generar el device tree en formato "human readable" mediante las Petalinux Tools, en base a nuestro diseño hardware (es necesario el bitstream). Hacer las modificaciones necesarias en estos archivos y compilar el devicetree en formato binario mediante el Device Tree Compiler (DTC) incluido en las PetaLinux Tools con el comando:

Código 5.2 Compilación DeviceTree.

```
./scripts/dtc/dtc -I dts -O dtb -o <devicetree name>.dtb <devicetree
    name>.dts
```

La creación de un proyecto de Device Tree en Petalinux arroja los archivos, en formato "human readable" de la Figura 5.10. Estos compondrán el Device Tree al ser compilados. Básicamente, es necesario editar estos archivos para limitar la sección de memoria que utilizará el Linux, el número de CPUs disponibles y evitar que interfiera con periféricos que utilizará el ARM1. En el caso de este proyecto, se pudo comprobar que existían problemas con los periféricos SPI y GPIO utilizados desde el segundo ARM al ejecutar en modo AMP, por lo que se efectúan las modificaciones que se exponen a continuación:

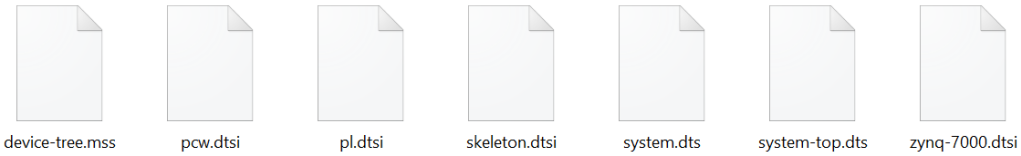

Figura 5.10 Archivos dts para creación del Device Tree.

  - **pcw.dtsi**. Este archivo recoge ciertas propiedades para los periféricos de la parte PS. En este caso, es necesario eliminar las sección donde se hace mención a la GPIO, en concreto:

Código 5.3 Eliminar esta sección del archivo pcw.dtsi.

```
&gpio0 {
  emio-gpio-width = <64>;
  gpio-mask-high = <0x0>;
```

```
    gpio-mask-low = <0x5600>;
  };
```

– **pl.dtsi**. Este archivo sirve para listar todos aquellos IP cores mapeados en memoria definidos en la parte PL. No es necesario modificar.

– **system-top.dts**. Contiene información acerca de la memoria, la salida por consola y argumentos de booteo. Es necesario modificar los registros base y final de la memoria que tendrá disponible el Linux, así como indicar en la línea de argumentos que sólo se utilizará uno de los ARM, de manera que el sistema operativo Linux no interfiera con el segundo y se permita la ejecución AMP. La línea "clk_ignore_unused" evita que el Linux resetee el reloj del SPI utilizado desde el ARM1. El archivo queda así:

Código 5.4 Archivo system-top.dtsi.

```
/dts-v1/;
/include/ "zynq-7000.dtsi"
/include/ "pl.dtsi"
/include/ "pcw.dtsi"
/ {
  chosen {
    bootargs = "maxcpus=1 clk_ignore_unused";
    stdout-path = "serial0:115200n8";
  };
  aliases {
    ethernet0 = &gem0;
    serial0 = &uart1;
    spi0 = &qspi;
    spi1 = &spi1;
  };
  memory {
    device_type = "memory";
    reg = <0x0 0x18000000>;
  };
};
```

– **zynq-7000.dtsi**. Este archivo contiene información adicional sobre los periféricos de la parte PS y las CPUs. Es necesario eliminar completamente el bloque dedicado a la GPIO, y en el bloque "slcr", eliminar de "clk-output-names" la mención: "gpio_aper".

Código 5.5 Eliminar esta sección del archivo zynq-7000.dtsi.

```
gpio0: gpio@e000a000 {
  compatible = "xlnx,zynq-gpio-1.0";
  #gpio-cells = <2>;
  clocks = <&clkc 42>;
  gpio-controller;
  interrupt-controller;
  #interrupt-cells = <2>;
  interrupt-parent = <&intc>;
  interrupts = <0 20 4>;
  reg = <0xe000a000 0x1000>;
};
```

Código 5.6 Sección "slcr" del archivo zynq-7000.dtsi.

```
    slcr: slcr@f8000000 {
      u-boot,dm-pre-reloc;
      #address-cells = <1>;
      #size-cells = <1>;
      compatible = "xlnx,zynq-slcr", "syscon", "simple-mfd";
      reg = <0xF8000000 0x1000>;
      ranges;
      clkc: clkc@100 {
      u-boot,dm-pre-reloc;
      #clock-cells = <1>;
      compatible = "xlnx,ps7-clkc";
      fclk-enable = <0xf>;
      clock-output-names = "armpll", "ddrpll", "iopll", "cpu_6or4x",
      "cpu_3or2x", "cpu_2x", "cpu_1x", "ddr2x", "ddr3x",
      "dci", "lqspi", "smc", "pcap", "gem0", "gem1",
      "fclk0", "fclk1", "fclk2", "fclk3", "can0", "can1",
      "sdio0", "sdio1", "uart0", "uart1", "spi0", "spi1",
      "dma", "usb0_aper", "usb1_aper", "gem0_aper",
      "gem1_aper", "sdio0_aper", "sdio1_aper",
      "spi0_aper", "spi1_aper", "can0_aper", "can1_aper",
      "i2c0_aper", "i2c1_aper", "uart0_aper", "uart1_aper",
      "lqspi_aper", "smc_aper", "swdt","dbg_trc", "dbg_apb";
      reg = <0x100 0x100>;
    };
```

- **ini1.sh**. Es un archivo de línea de comandos de Linux. Se puede customizar con aquellos comandos que se quieran ejecutar. No es necesario incluirlo, aunque en el caso de este proyecto se utiliza para llamar a la aplicación "rwmem.elf" [66]. El objetivo es escribir en la dirección de memoria 0xFFFFFFF0 la dirección de memoria en la que empieza la aplicación en el otro núcleo (0x18000000). Esto es porque cuando arranca el sistema, el segundo núcleo es programado con la aplicación provista en el BOOT.bin, pero queda en bucle leyendo la dirección 0xFFFFFFF0 a la espera de que se ordene el comienzo de su ejecución mediante la escritura de un valor distinto de 0 que será la dirección de memoria a la que salte (la dirección base de la aplicación del núcleo 1). Este proceso se puede hacer también desde la propia aplicación de Linux. El código es el siguiente:

Código 5.7 Código de ini1.sh.

```
#!/bin/bash

./rwmem.elf 0xFFFFFFF0 0x18000000
```

- **rwmem.elf**. Es el ejecutable de la aplicación de ejemplo aportada por Xilinx en su nota de aplicación sobre AMP [66]. Su funcionalidad es la de realizar lecturas y escrituras en la memoria compartida de la Zynq-7000. En este caso se utiliza para ser llamada desde ini1.sh. Se extrae de la carpeta "Debug" dentro del proyecto de aplicación. Se pueden tener varias guardadas en la raíz. Se ejecutará una vez arrancado el sistema operativo mediante comandos.

- **scnew.elf**. Es el ejecutable generado por el SDK de Xilinx al compilar la aplicación de Linux diseñada. Igualmente, se extrae de la carpeta "Debug" dentro del proyecto de aplicación y se puede ejecutar desde el sistema operativo mediante comandos.

- **uImage**. Es un "wrapper" que incluye la imagen del kernel del sistema operativo junto con el header del u-boot que permite su lectura en el arranque. Los pasos para su compilación se pueden encontrar en [68]. En este caso no es necesario hacer ninguna modificación respecto a la compilación por defecto que ofrece Xilinx, por lo que se utiliza esta misma.

- **uramdisk.image.gz**. Se trata también de un "wrapper" que contiene el header del u-boot para su lectura en el arranque y el archivo ramdisk.image que contiene información acerca del sistema de archivos del sistema operativo Linux. Igualmente, se puede extraer información acerca de su compilación de la Wiki de Xilinx, aunque en este caso no es necesario customizar este archivo, por lo que se puede utilizar el ofrecido por defecto por Xilinx.

Una vez se han obtenido todos estos archivos, se copian a la raíz de la tarjeta SD, y si se enciende la Zedboard con la configuración de jumpers de la Tabla 5.1, comenzará el proceso de arranque del sistema. El encendido del led azul de la Zedboard es indicativo de que los distintos archivos del BOOT.bin han sido programados correctamente. Es conveniente conectar el puerto Ethernet de la Zedboard a un router. Como el SO compilado tiene DHCP, el router le asignará una dirección IP automáticamente. Así mismo, para interactuar con el sistema es necesario conectar mediante un cable USB el puerto UART de la Zedboard al PC, el cual debe de reconocer este cable como un puerto COM. Mediante un terminal en el PC conectado a este puerto COM se podrán ver las salidas de texto del proceso de arranque del SO y empezar a interactuar con el sistema a través de comandos.

Tabla 5.1 Configuración de jumpers en Zedboard. El resto de jumpers quedan al aire.

Conectado a	Señal	Jumper	Conectado
JP11 (MIO6)	GND	JP6 (MIO0)	Conectado
JP10 (MIO5)	3V3	J18	2v5
JP9 (MIO4)	3V3		
JP8 (MIO3)	GND		
JP7 (MIO2)	GND		

Una vez terminado el proceso de arranque, se puede acceder al sistema introduciendo el nombre de usuario "root" sin contraseña. Esto dará acceso al sistema operativo Linux, tal y como se puede observar en la Figura 5.11. Se puede acceder a la tarjeta SD en la dirección "home/run/media/mmcblk0p1". Con la raíz de la tarjeta SD como directorio, se pueden ejecutar las distintas aplicaciones introducidas mediante comandos.

En cuanto a la posibilidad de Debuggear la aplicación baremetal del ARM1, si se intenta debuggear con las configuraciones por defecto a través del JTAG, el sistema es reseteado, por lo que se interrumpe la ejecución del SO y no se puede debuggear la aplicación en modo AMP. Para poder hacer esto, es necesario ir a la opción "Debug Configurations" del proyecto de aplicación que se quiere debuggear y deseleccionar las opciones: "Reset Entire System", "Program FPGA", "Run ps7_init" y "Run ps7_post_config". Esto permite que el sistema siga su ejecución normal en modo AMP, y tan sólo se escriba la aplicación del ARM1 que comienza su ejecución en paralelo sin problemas.

5.4 Descripción detallada del diseño

Una vez ya se ha especificado como poner en marcha el sistema ejecutando en modo AMP un sistema operativo en el ARM0, con una aplicación ejecutándose en el ARM1, se procederá a detallar con mayor profundidad los distintos aspectos de la arquitectura que ya se introdujo en el segundo apartado. Los puntos clave serán la configuración del hardware, el intercambio de información entre los distintos elementos a través de la memoria compartida y el diseño del algoritmo de control en VHDL, incluyendo posibles alternativas a la hora de abordar su diseño y gestionar la complejidad del algoritmo. El objetivo final será el de obtener tanto el bitstream que configura toda la parte hardware, como los dos ejecutables de las respectivas aplicaciones.

Como referencia, se aporta el diagrama de bloques de la Figura 5.12. Este diagrama recoge de manera esquemática los distintos bloques en los que se puede descomponer la arquitectura propuesta y que se irá explorando en mayor profundidad a lo largo de este apartado.

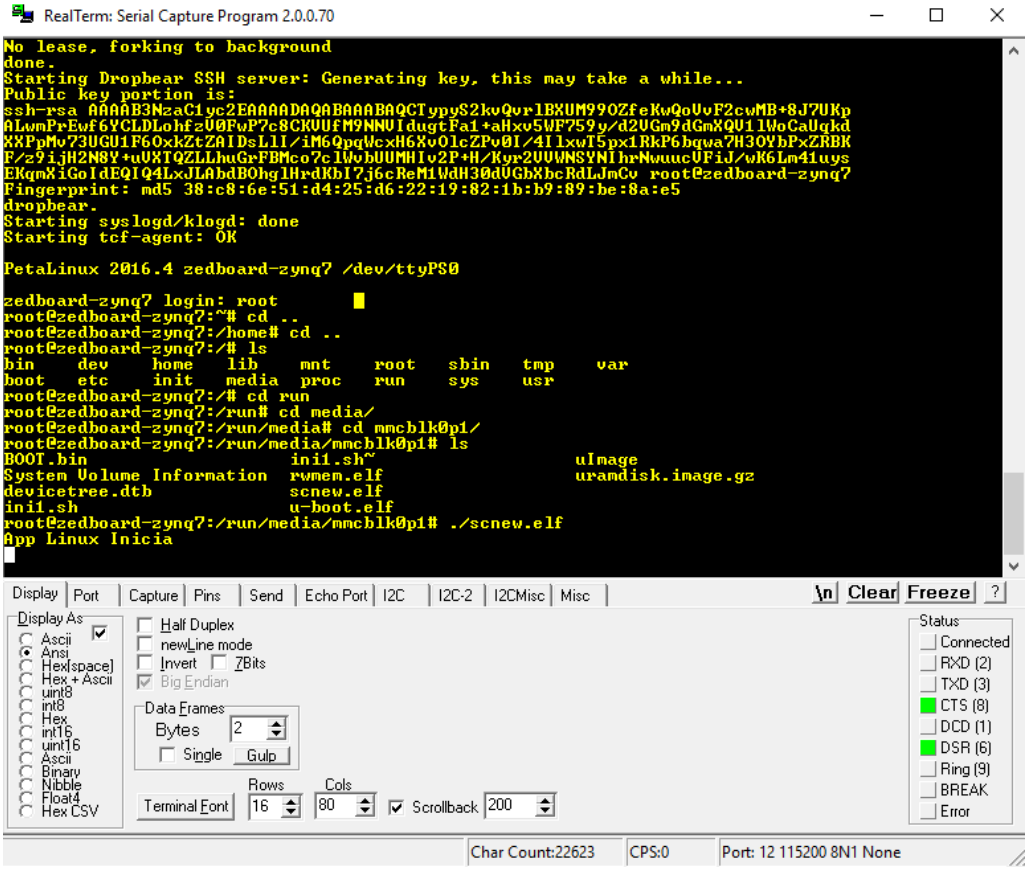

Figura 5.11 Interacción desde terminal con el SO ejecutándose en la Zedboard.

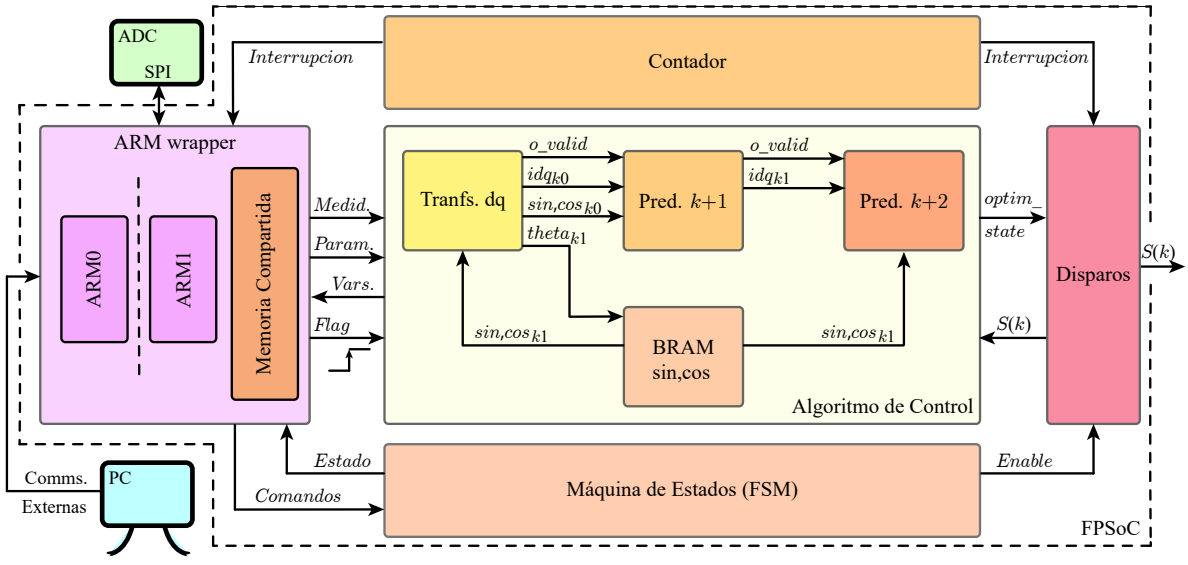

Figura 5.12 Diagrama de bloques de la arquitectura propuesta.

5.4.1 Programación de la parte hardware I. Estructura básica y bloques auxiliares

Este subapartado estará dedicado a la programación en lenguaje VHDL de toda la parte de la arquitectura que se implementa en la FPGA.

Configuración de la parte PS. Creación del "ARM wrapper"

El primer paso será realizar un diseño en diagrama de bloques para configurar la parte PS, de manera similar a lo descrito en el primer apartado de este capítulo. Lo primero de todo será añadir el bloque de la Zynq-7000 y configurarlo adecuadamente. Respecto a las configuraciones por defecto, se realizan los siguientes cambios:

- **PS-PL Configuration**. En esta pestaña es necesario habilitar el M AXI GP0 Interface. Esto creará un puerto AXI que permite interconectar los procesadores ARM con la parte PL para la compartición de memoria

- **Peripheral I/O Pins**. Se asigna el periférico "SPI1" que se utilizará para la gestión del ADC a los pines 10 (MOSI), 11 (MISO), 12 (SCLK) y 13 (CS). Estos pines están cableados en la Zedboard al puerto PMOD JE, que es solo accesible desde la PS a través de MIO. De igual manera, asignar los pines 0, 7, 8, 9, 14 y 15 a la GPIO MIO.

- **Clock Configuration**. Configurar la frecuencia de reloj del periférico SPI a 148. Con el clock source "IO PLL" por defecto, esto debe de dar unos 142 MHz de frecuencia real, que se ha calculado para poder configurar el SPI a unos 20 MHz más adelante mediante los divisores de frecuencia adecuados. En la pestaña "PL Fabric Clocks" habilitar la salida "FCLK_CLK0" a 100 MHz. Ésta será la señal de reloj para el protocolo AXI.

- **Interrupts**. Se debe activar la interrupción PL-PS "Core1_nIRQ" (ID31). Ésta es una interrupción privada del ARM1 que proviene de la parte PL. Es decir, se activará cuando la señal asociada de la FPGA esté a nivel alto y sólo el ARM1 la atenderá. Hay que tener en cuenta que el comportamiento de las interrupciones privadas está prefijado y no es configurable. En este caso, estas interrupciones son activas por nivel, por lo que se diseña un bloque VHDL que simplemente recibe esta señal y la mantiene a nivel alto durante el tiempo de interrupción necesario y la resetee adecuadamente.

Una vez configurado el bloque de la Zynq-7000, se abordará la compartición de memoria. El objetivo es poder transferir datos estre la FPGA y los ARM de una manera eficiente. Esta plataforma prevé los siguientes métodos para la interconexión de la parte PS y PL [71]:

- Dos puertos AXI de 32 bits maestro (PS Máster).
- Dos puertos AXI de 32 bits esclavo (PL Máster).
- Cuatro puertos "High Performance" de 32 ó 64 bits esclavos (PL Máster).
- Un puerto ACP (Accelerator Coherency Port) de 64 bits esclavo (ACP) (PL Máster).
- Cuatro relojes desde la PS a la PL.
- Interrupciones PS - PL.
- Interrupciones PL - PS.
- Interfaz DMA.

Tal y como se ha planteado la arquitectura de este proyecto, tiene más sentido otorgar a la parte PS la categoría de maestro en la comunicación, ya que es el que obtiene las medidas del ADC en tiempo de interrupción y avisa a la FPGA de la disponibilidad de estos datos para el comienzo de los cálculos para el algoritmo de control. Se utilizará entonces el puerto Máster AXI de 32 bits definido anteriormente en el bloque Zynq-7000. Otra posibilidad sería utilizar el controlador DMA para transferencias más rápidas y que alivien la carga del procesador. Sin embargo, siguiendo los resultados obtenidos en la publicación [32], la utilización de la DMA para escrituras y lecturas de datos entre procesador y FPGA empieza a rentar a partir de una cantidad importante de datos, la cual en principio no se va alcanzar con la información que se planea transmitir entre FPGA y procesador. En este caso, se transmitirá un número limitado de medidas y variables, en su mayoría enteros de 16 ó 32 bits que no suponen una cantidad total de Bytes muy grande.

Una vez elegido el protocolo en el que se basará la comunicación FPGA-ARM, es necesario implementarlo en el diseño. La manera más inmediata y sencilla de hacerlo es mediante la creación de un periférico customizado gracias a la opción de Vivado "Tools -> Create and Package New IP". Esta herramienta permite la creación de un IP core personalizado, que básicamente se compondrá de una serie de registros asociados a posiciones de la memoria compartida y de unos puertos de entrada y salida que permiten la conexión a un protocolo de comunicación, como puede ser AXI. Por suerte, la herramienta Vivado genera automáticamente todo el código VHDL necesario para gestionar el protocolo, encargándose de escribir y leer los datos de los distintos registros definidos y enviarlos a través del puerto. Una vez los datos han sido escritos por el maestro

en estos registros, están disponibles para ser cableados a señales en el diseño VHDL. Igualmente, señales en la FPGA pueden acceder a estos registros y escribir información en ellos, que el maestro leerá mediante una petición por el puerto AXI, gestionada mediante el código generado en la creación del periférico.

Una vez se ha hecho click en la opción "Create and Package New IP", se pueden configurar los distintos parámetros del periférico que se quiere crear. La primera opción a seleccionar es la de "Create a New AXI4 Peripheral". Esta opción permitirá crear un IP core con entrada/salida AXI. Además, esta opción incluye la generación de drivers y librerías para la interacción con este periférico desde la parte PS, lo que facilita la utilización del mismo. En la siguiente ventana, se debe de asignar un nombre y una carpeta para guardar el proyecto. En nuestro caso, el periférico creado se llamará "ADCreg_100". En cuanto a la ventana de la configuración de las interfaces, se seleccionarán las siguientes opciones:

- **Tipo de Interfaz**. Se selecciona una interfaz de tipo AXI Lite. Esta tipología del protocolo AXI es una versión reducida del mismo, ideal para aplicaciones donde no se espera el envío de grandes cantidades de datos.
- **Modo de Interfaz**. Será una interfaz esclava, de manera que la parte PS actúe como el maestro de la comunicación.
- **Ancho de Datos (Bits)**. Se selecciona un ancho de bits de 32 bits. Como las medidas del ADC son de 12 bits y se utilizan números enteros de 16 bits para la mayoría de los cálculos, se puede utilizar una sola posición de memoria para guardar dos variables.
- **Número de Registros**. Se eligen 128 registros para tener un número suficiente de cara a sucesivas ampliaciones del diseño.

Con estas opciones, se puede proseguir con la opción de añadir la IP al repositorio. Esto permitirá su adición al diseño de bloques, y por tanto, la posibilidad de interconectar este periférico AXI al puerto maestro AXI que se dejó configurado en el bloque ZYNQ7. Si se aplica la opción de ruteo automático, el software Vivado añade automáticamente los bloques "AXI Interconnect" para la gestión de varios periféricos AXI a través de un mismo puerto y el bloque "Processor System Reset" que gestiona los reset de la comunicación. Tan solo es necesario cablear la señal de reloj que se utilizará, que en este caso será la señal de reloj "FCLK_CLK1" del bloque ZYNQ7.

Una opción interesante que ofrece Vivado es la de editar el rango de direcciones asignado a cada periférico. En la pestaña "Address Editor", se puede configurar la dirección de memoria base y el rango asignado. Por defecto, Vivado asigna un rango amplio de direcciones, mayor que el realmente necesario. Éste se mantendrá por ahora porque es el único periférico en el diseño de bloques. El rango de direcciones quedaría tal y como se puede ver en la Figura 5.13.

Figura 5.13 Rango de direcciones asignado al periférico creado.

Otra posibilidad que ofrece Vivado es la de ver y editar el código VHDL generado en la creación del periférico. Esto es posible haciendo click derecho en el bloque y pulsando la opción "Edit in IP Packager". Esto abrirá un nuevo proyecto de Vivado donde se puede editar el IP core creado. Se tienen dos archivos de código VHDL. El primero de ellos, "ADCreg_100_v1_0_S00_AXI.vhd" contiene todo el código con la funcionalidad del protocolo AXI y los 128 registros que se crearon. El segundo, "ADCreg_100_v1_0.vhd" se trata del top level del proyecto. Estos archivos incluyen zonas para que el usuario pueda insertar código propio sin necesidad de tocar el código generado por la herramienta. Esto permite añadir funcionalidad extra a este periférico o crear puertos de entrada/salida que conecten a los registros.

En este caso, se han generado un diseño con 128 registros en el archivo "ADCreg_100_v1_0_S00_AXI.vhd". Esto se traduce a que se han definido 128 señales ("signal") de nombre "slv_regX" y tipo "std_logic_vector" con ancho 32 bits (definido mediante un "generic"). Se requiere para la arquitectura propuesta que estos

registros estén cableados a puertos de entrada y salida en el top level de este bloque. De esta manera, los datos escritos en los registros serán accesibles desde la FPGA, y también se podrán escribir otros registros desde la misma para que sea el procesador quien los lea. La metodología será, en la entidad de este primer archivo, definir en el campo "port" de la entidad, sucesivos puertos de entrada y salida del mismo tipo que las señales "slv_regX". En este caso, se han repartido los registros entre puertos de entrada y salida de manera que los diez primeros sean de salida, los diez siguientes de entrada y así sucesivamente hasta agotar los 128 registros disponibles. En la arquitectura de este diseño, será necesario realizar la asignación correspondiente entre las señales y los puertos de entrada y salida creados, según corresponda.

Adicionalmente, en el "process" donde se resetean los registros, se fuerza a que se reseteen tan solo los registros conectados a puertos de salida. Igualmente, en el "process" que gestiona la escritura de los registros desde el maestro, es conveniente comentar la línea que escribe en registros conectados a puertos de entrada del bloque, de manera que sólo la FPGA los pueda modificar, y no se escriban por error desde la parte procesadora. No es necesario hacer lo propio para la lectura, ya que es deseable que todos los registros estén disponibles para su lectura desde el procesador.

Editado el primer archivo del bloque, se procede a hacer lo propio con el el top level del mismo. En este caso lo que se tiene es el diseño anterior definido en la arquitectura como un componente ("component"). Lo único que resta es cablear los puertos de entrada y salida de este componente a los puertos de entrada y salida que se definan en el top level. En este caso, los puertos del top level serán aquellos a los que realmente se quiera acceder desde el diseño hardware, con la nomenclatura y el ancho deseados. Por ejemplo, se pueden definir dos puertos de salida de anchura 16 bits, que sean respectivamente el valor de la tensión y la corriente en una fase. Estos dos puertos se pueden cablear ambos a un mismo registro, asignado un valor a los 16 bits MSB y el otro a los 16 bits LSB. De esta manera, todas las variables que se van a querer leer o escribir en la memoria compartida quedan disponibles en el diseño hardware.

Hechas todas las modificaciones pertinentes al IP core, en la opción "Package IP" del "Flow Navigator" de Vivado, hay que ir aplicando los sucesivos pasos que aparecen hasta poder aplicar el último paso sin errores, lo cual significa que el IP core creado ha sido generado y actualizado con las últimas modificaciones. Ya se puede cerrar el nuevo proyecto de Vivado que se creó al editar el IP core.

De vuelta al proyecto original, Vivado detectará que se ha generado una nueva versión del periférico en el panel "Report IP Status", por lo que pide regenerar el diseño. Hecho esto, se procede a crear sucesivos puertos en el diagrama de bloque para las señales que entran y salen del bloque. Finalmente, cableando un puerto de interrupción ("interrupt") al puerto Core1_nIRQ, se puede validar el diseño final (ver Figura 5.14). Como se puede observar, los diferentes registros que se han definido se tratan de valores de medidas o de variables internas calculadas en el algoritmo de control. También existen algunos puertos de un solo bit que se utilizan como banderas para avisos. En particular la bandera "flagrep" se utiliza para informar del estado de reposo/funcionamiento del sistema, mientras que la bandera "flag1" (correspondiente con la variable "Flag" del diagrama de la Figura 5.12) se escribe a '1'cuando las medidas del ADC han acabado. Esto permite avisar a la FPGA de que las medidas están disponibles en memoria y puede comenzar los cálculos. Las señales "START", "RESET", y "RMVERR" son señales que se activan desde la PS según los comandos que llegan de la interfaz para interaccionar con la máquina de estados que gobierna el sistema.

De manera similar a como se describió en el primer apartado de este capítulo, se puede proceder a generar un "wrapper" de manera automática que sirva como top level para el diseño de bloques generado. Este "wrapper" se definirá como componente en el top level global del proyecto.

Máquina de estados (FSM)

Siguiendo con el resto de bloques que componen la arquitectura propuesta, se tiene la máquina de estados que gobierna el sistema. Básicamente, esta máquina de estados tiene como entradas las señales que provienen del "ARM wrapper", las cuales se escriben en memoria según los comandos que llegan desde la interfaz gráfica. Igualmente, se prevén tres botones en la placa a los que se les asigna la misma funcionalidad. Lo que se hace es pasar sendas señales en el top level por una puerta lógica OR, de manera que llega la orden a la máquina de estados independientemente de dónde provenga.

Como salida de la máquina de estados se tiene la señal que informa al resto de bloques sobre el estado de marcha/paro del sistema ("Rep"). Ésta se cableará a leds físicos de la Zedboard para obtener retroalimentación visual del estado, al igual que con el estado de error.

Los estados por los que puede pasar la FSM son: "Reposo", "Disparo", "estError", y sus tres respectivos estados auxiliares: "Reposoaux", "Disparoaux", "erroraux". Estos estados auxiliares son "copias" del correspondiente estado y se utilizan para poder actuar sobre la máquina de estados con un único botón de marcha/paro. De esta manera, si se da la orden de pasar por ejemplo, de reposo a marcha, se intercala

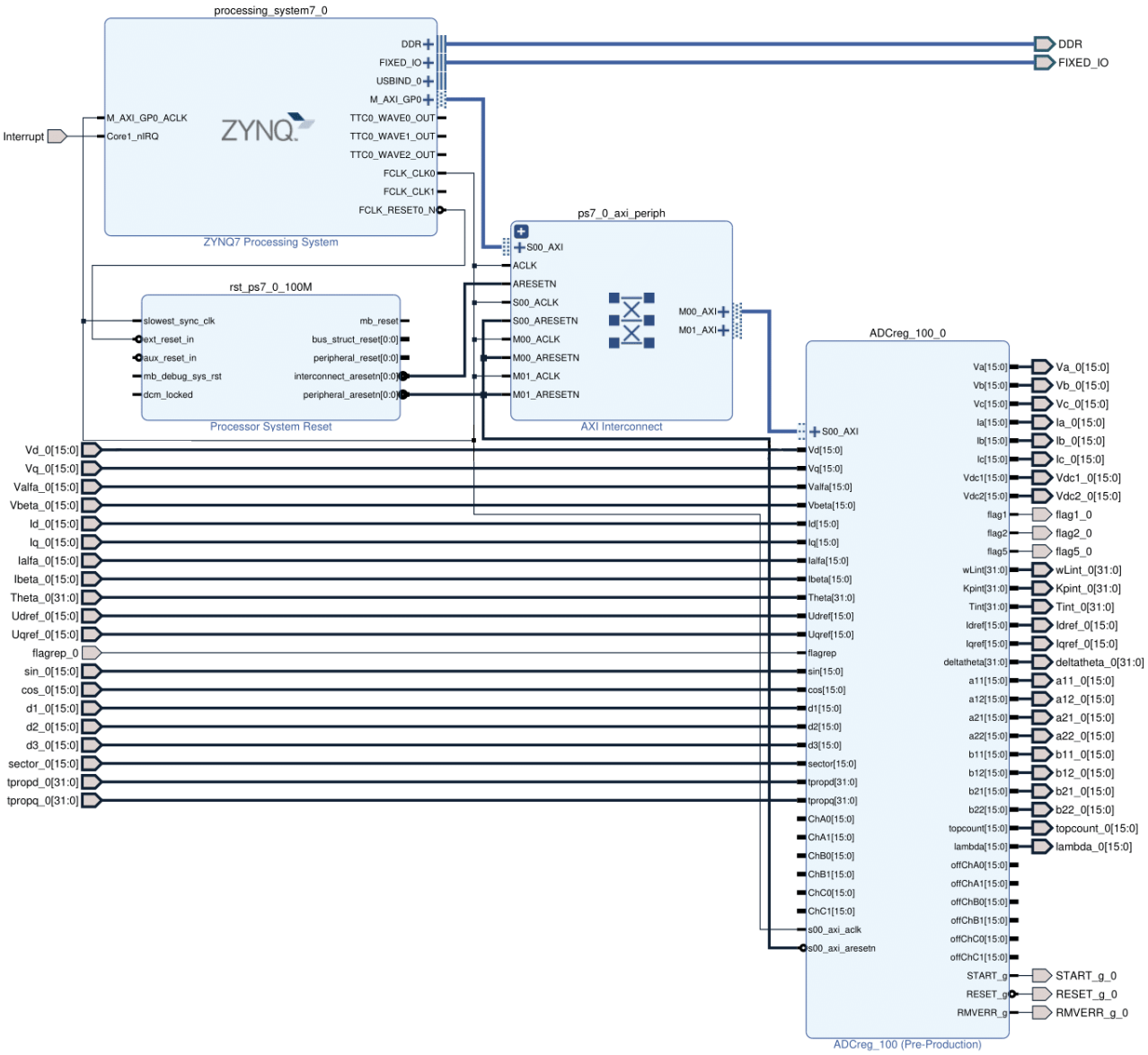

Figura 5.14 Diseño de bloques para la parte PS.

un estado auxiliar que espera a que la señal de entrada vuelva a '0', para que no se recorra la máquina de estados de manera automática numerosas veces mientras se actúa sobre el botón. Las entradas que provienen de botones se pasan por sus correspondientes bloques de filtrado de rebotes. Se ofrece en la Figura 5.15 un diagrama de la máquina de estados para mayor claridad. El estado por defecto tras reset del sistema es "Reposoaux".

Un apunte en cuanto a la placa de driver del inversor, es que requiere una señal de Enable a nivel lógico alto para comenzar a disparar. Esta señal de Enable será el negado de la señal de reposo. La placa que gestiona las señales que llegan al driver, genera la señal de habilitación de disparos al convertidor a partir de este Enable y de las propias señales de errores asociados a cada IGBT, mediante puertas NAND. Así mismo, la señal de error que sale del driver, se genera a partir del negado de la señal de habilitación del driver, como se puede apreciar en la Figura 5.16. Por este motivo, es importante que la señal de error que llega a la máquina de estados sea el AND de la señal de error que produce el driver y la señal de enable que se envía al driver desde la FPGA (esto se trata en el top level). De esta forma, se evita que en el estado de reposo se lea un error que no corresponde y envíe a la máquina de estados al estado de error. En resumen, cualquier error en el hardware de potencia (señales HALT_OUTx) deshabilitan los disparos y generan una señal de error (TX de fibra óptica a 0). Los RX de fibra óptica invierten la señal, por lo que se lee un '1' lógico desde la FPGA. La deshabilitación del Enable desde la FPGA también genera una señal de error, pero se descarta con

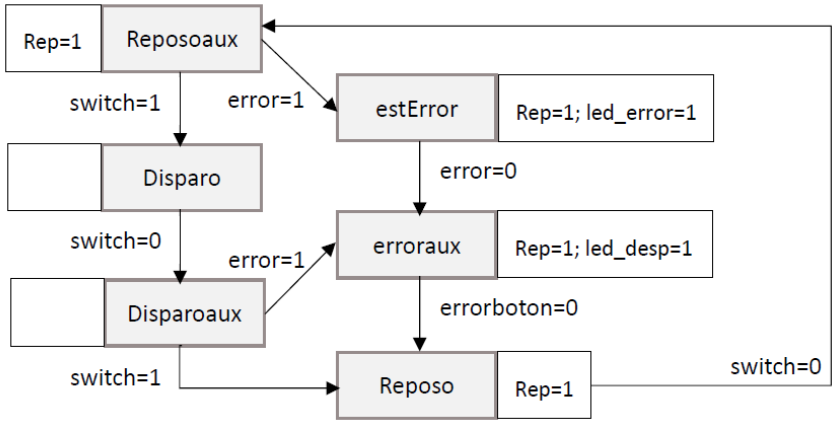

Figura 5.15 Diagrama de la máquina de estados.

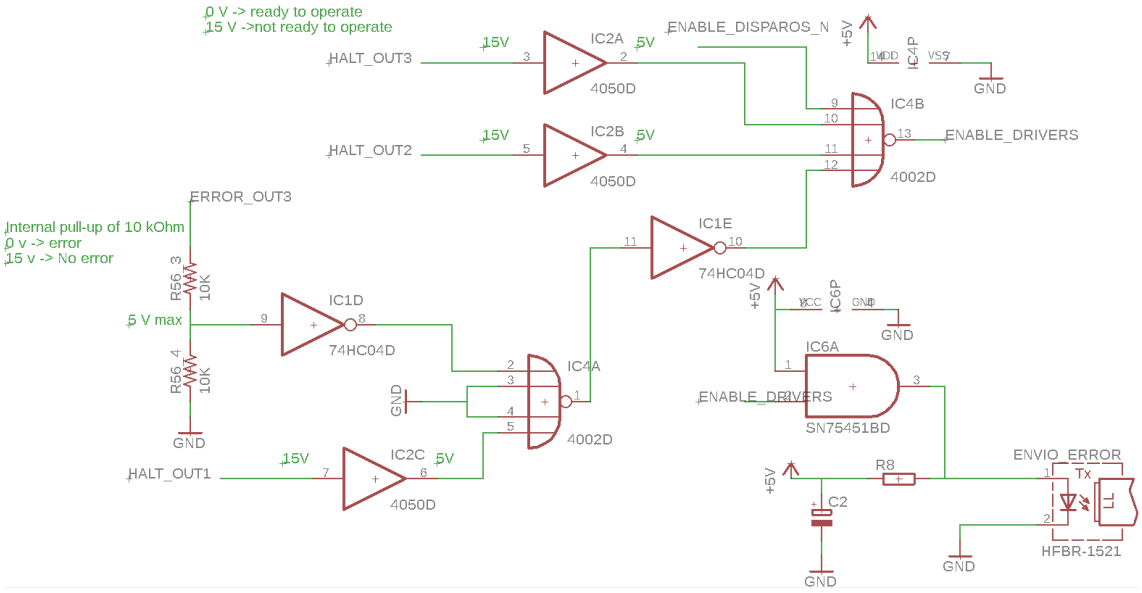

Figura 5.16 Señal de error generada en el driver del inversor.

la corrección comentada anteriormente.

Desde el punto de vista de la codificación en VHDL, se sigue la estructura recomendada de dividir la arquitectura en dos procesos. Uno de ellos contiene toda la lógica combinacional de la máquina de estados, mientras que el otro será un proceso síncrono que tendrá la señal de reloj y la de reset en su lista de sensibilidad. En concreto, el reset será un reset asíncrono, de manera que todas las señales vuelvan a su estado por defecto cuando se produzca un reset global del sistema. Por otro lado, cuando sucede un flanco de reloj ("rising_-edge(clk)"), es cuando realmente se asignan los distintos valores de las señales al valor que el proceso combinacional ha calculado. De esta manera, el proceso combinacional, si detecta un cambio en su lista de sensibilidad actualiza según corresponda el valor de "p_estado", por ejemplo. Cuando exista un flanco de reloj, el proceso síncrono actualiza la señal "estado" al valor de "p_estado". A nivel práctico, se está imponiendo que la herramienta infiera una serie de biestables ("flip-flops", también denominados en este contexto como "registros") al final de la lógica combinacional que consiguen que las salidas de la máquina de estados se sincronicen con la señal de reloj, evitando posibles "glitches". Codificar separando la lógica combinacional de la secuencial en dos procesos, suele ser la más recomendada en la mayoría de manuales de VHDL y es considerada como más intuitiva y descriptiva del diseño a nivel RTL que se está codificando.

Adicionalmente, es importante evitar que la herramienta infiera "latches" en las señales que se asignan según el estado que se tome. Esto se consigue asignando en el comienzo del proceso combinacional un valor por defecto a "p_estado", que en este caso podría ser "p_estado" igual a "estado". Si según la lógica de la máquina de estados esta variable debe de cambiar, se machaca la anterior asignación con el nuevo valor que

"p_estado" debe de tomar. Se hace lo propio para todas las señales asignadas en el proceso combinacional.

El resto de bloques VHDL programados seguirá en la medida de lo posible esta metodología de programación, dividiendo la arquitectura en dos procesos y asignando valores por defecto a las señales del combinacional para evitar "latches". No se especificará este hecho continuamente salvo en casos que sea necesario volver a puntualizar.

Contador

Este bloque se encarga de la temporización del sistema. Básicamente, se define una señal que se incrementa en una unidad en cada ciclo de reloj. El valor límite del contador se define según el tiempo de sampling que se quiere asignar a cada intervalo de control. De esta manera, se dispone de una entrada "topcount" que se cablea a una posición en la memoria compartida. Según la frecuencia de sampling que el usuario envía desde la interfaz, y conociendo que la frecuencia de reloj utilizada es de 100 MHz, se calcula el valor que debe de tomar "topcount".

Cuando el valor de la cuenta llega al valor tope, el contador se resetea y lanza una señal de fin de cuenta que se mantiene a nivél lógico '1'durante un flanco de reloj. Esta señal indica el final del intervalo de control y se utiliza para generar la señal de interrupción del puerto "Core1_nIRQ" que se dejó configurado en el bloque ZYNQ7 del "ARM wrapper". Adicionalmente, indica al bloque que gestiona las señales de disparos que estos se deben de actualizar.

Disparos

Este bloque aplica las señales de disparos que llegan del algoritmo de control cuando el contador indica que empieza el nuevo intervalo de sampleo. De esta manera, este bloque sabe en todo momento el estado actual de los IGBT del inversor. Éste se ofrece a través de un puerto de salida al algoritmo de control, ya que es necesario conocer este estado para el modelo predictivo, que requiere para sus cálculos conocer la tensión generada en el inversor (Ecuación 2.2).

En la implementación VHDL propuesta, el estado se trata siempre como un "std_logic_vector" de tres componentes, conteniendo la señal de disparo de la rama superior del inversor para las fases a, b y c. El driver del inversor se encarga de gestionar los tiempos muertos y generar la negada para la rama inferior. En el proceso combinacional se lee este vector y se asignan las tres señales de disparos correspondientes. Éstas se aplicarán en un flanco de reloj si coincide con una saturación del contador.

Las señales de disparos se cablean a puertos de salida del top level, que mediante el archivo de "constraints" se asocian a pines físicos en los puertos PMOD.

Bloque de filtrado de Rebotes

Este bloque se utiliza para filtrar los rebotes propios de los botones físicos que se utilizan como entradas para asegurar una lectura estable. Simplemente se habilita una cuenta que se resetea en cuanto se produce un cambio entre el valor actual y el anterior de la entrada. Esto se detecta mediante una puerta lógica XOR. Si se produce un cambio entre estas señales, no se asigna la salida. Si la cuenta llega a su valor tope (en torno a 2^{20} ciclos de reloj para una espera en el orden de milisegundos), quiere decir que la lectura es estable y se puede escribir el valor de la entrada a la salida del bloque.

Bloque de detección de flancos

Para la detección de flancos en señales, no es conveniente utilizar "rising_edge", ya que este se reserva para la detección de flancos de reloj en el proceso síncrono. Para ello, se codifica un bloque que genera una salida igual a "1" durante un periodo de reloj cuando la señal de entrada pasa de "0" a "1". Igualmente, se codifica otra salida del bloque que haga lo propio para el flanco de bajada, ya que es útil tener ambas posibilidades en un mismo bloque.

Para conseguir esta funcionalidad se tienen dos señales que básicamente samplean la señal de entrada en cada ciclo de reloj en el proceso síncrono. Una de ellas es el valor actual y la otra el anterior. Para ello, esta última realmente se asigna al valor de la primera señal para conseguir este retraso de un ciclo de reloj. Fuera de este proceso, como asignaciones concurrentes, la señal del flanco de subida será producto del AND de la negada de la señal retrasada un ciclo y la actual. Por otro lado, la del flanco de bajada será la AND de la negada de la señal actual y la retrasada un ciclo.

Bloque Interrupción

Este bloque construye la señal de interrupción que entra al bloque del procesador. Simplemente genera una señal que se pone a '1'cuando el contador termina su cuenta y vuelve a '0'cuando se activa la bandera de aviso "flag1" desde el procesador.

5.4.2 Programación de la parte hardware II. Bloques del algoritmo de control

Todos los bloques correspondientes al algoritmo de control se verán con mayor detalle en este subapartado, donde se profundizará en la metodología seguida para implementar todos los cálculos correspondientes en VHDL.

Aritmética de punto fijo para los cálculos

Uno de los puntos clave de este proyecto radica en la implementación de los cálculos del algoritmo de control en la FPGA. Resolver esto en un entorno de programación software con una aplicación que ejecuta una CPU es algo trivial utilizando aritmética de punto flotante y la representación real de las distintas variables. Adicionalmente, mediante el uso de librerías como la "math.h" en lenguaje C, se tienen acceso a funciones matemáticas que implementan el cálculo en aritmética flotante de muchas funciones trascendentes como las trigonométricas o logarítmicas. Por ejemplo, para el caso de este proyecto, para trabajar en ejes dq, es necesario transformar las magnitudes en ejes estacionarios $\alpha\beta$ a unos ejes síncronos que rotan a velocidad w (frecuencia angular). Esto quiere decir que es necesario calcular una transformación mediante una matriz de rotación cuyas componentes son los senos y los cosenos del ángulo de fase. En el caso de una aplicación de C, solucionar esto es trivial mediante las funciones previamente mencionadas.

En FPGAs, esto supone una complicación a solventar. Por un lado, en las librerías que se utilizan en el proyecto, "IEEE Standard Logic 1164", y "IEEE Standard Logic Numeric" no están definidos ni el tipo de datos "float" ni las operaciones de números flotantes. Estos se añadieron en la revisión de 2008 mediante el paquete "float_pkg". Sin embargo, implementar un diseño con cálculos en flotante en una FPGA implica un consumo de recursos mucho mayor que utilizando aritmética de punto fijo [72]. Otro punto a señalar, es que el estándar sí define el tipo de dato "real", que puede tomar valores del conjunto de los números reales y ser utilizado para operar de manera similar a los números flotantes en programación software. sin embargo, este tipo no es sintetizable y solo se puede utilizar en simulación. Esto quiere decir que a la hora de elaborar un diseño que se va a implementar en una FPGA, al no poder utilizar la representación real de las distintas variables, el hecho de utilizar aritmética flotante no implica una mayor facilidad para un diseñador acostumbrado a la programación de aplicaciones software. Sí tiene una serie de ventajas en cuanto a la precisión de los cálculos gracias al rango dinámico de las operaciones en flotante, pero no suele compensar ante el gasto mayor de recursos.

En definitiva, para todos los cálculos de esta implementación en la parte FPGA, se utilizará una aritmética de punto fijo en la que las distintas variables se escalan a un rango de número de enteros apropiado. De esta manera, las distintas operaciones se realizarán entre enteros, con signo ("signed") o sin signo ("unsigned") según corresponda. En el caso de números con signo, la representación utilizada es la de complemento a 2, que es la más habitual en operaciones de aritmética de punto fijo.

Básicamente, la metodología que se ha seguido es, conociendo el rango de medidas del ADC en enteros, y los valores de intensidad y tensión reales a los que corresponden, aplicar a las medidas una escala común y trabajar en ese rango de enteros. En este caso, como hay 3 medidas de intensidad y 1 medida de tensión (dc-link), lo que se ha hecho es dejar las corrientes en la escala de enteros tal y como se obtiene del ADC, simplemente corrigiendo el offset, y pasar la medida de tensión a la escala de las corrientes. Esto se hace en el ARM que gestiona las medidas, por lo que desde el punto de vista de la FPGA todas las medidas están ya correctamente adaptadas y en el rango de enteros adecuado.

De esta forma, se conoce que la placa roja de adaptación acepta corrientes de entrada del sensor en el rango de $\pm$ 25 mA, que se corresponden con corrientes de $\pm$ 25 A en la realidad. A la salida de la placa de adaptación, este rango se transforma en tensiones de 0 a 3 V. El rango de medida del ADC es configurable de 0 a 2.5 V ó 5 V. En el caso de que el rango sea el máximo, al valor de 1.5 V que se impone de offset a las señales de alterna, le correspondería teóricamente un valor entero de 1229 ($1.5/5 * 4095$). Igualmente, al valor máximo de 3 V le corresponde un valor de 2458. Por este motivo, la escala teórica de enteros quedaría (en el rango 0-5 V):

$$i_{ADC}^{unsigned} = i_{real} \frac{1229}{25} + Offset \tag{5.1}$$

siendo el Offset teórico igual a 1229. Como no se van a alcanzar valores de intensidad tan altos en esta aplicación, se puede medir en el rango de 0 a 2.5 V sin saturar la medida, doblando la precisión de la misma. Entonces, la escala de enteros quedaría:

$$i_{ADC}^{unsigned} = i_{real} \frac{2458}{25} + Offset \tag{5.2}$$

siendo el Offset (teórico) igual a 2458. Si se trabaja en el rango de enteros con el offset corregido, se tiene los siguiente:

$$i_{ADC}^{signed} = i_{ADC}^{unsigned} - Offset = i_{real} \frac{2458}{25} \tag{5.3}$$

En resumen, al valor real de la intensidad le corresponde un valor entero en nuestra implementación escalado por el factor $2458/25$. El offset se corrige al tomar la medida del ADC con el valor real obtenido al calibrar los sensores. Para el sensor de tensión, se puede hacer lo propio, conociendo que su rango máximo es de 0 a 440 V. Como se trata del sensor del bus de continua, la tensión a medir es siempre positiva y sin corrección de offset, por lo que se tiene:

$$vdc_{ADC}^{signed} = vdc_{ADC}^{unsigned} - 0 = vdc_{ADC}^{unsigned} = vdc_{real} \frac{2458}{440} \tag{5.4}$$

Es decir, el valor entero de la tensión está escalado $2458/440$ respecto del real. Para trabajar en el mismo rango que las corrientes, se reescala este número multiplicando por el factor $440/25$. Aunque el offset teórico de esta medida sea 0, es posible que debido al ruido de medida exista un pequeño offset que se corrige mediante la calibración.

Al trabajar con la misma escala de enteros, las magnitudes se pueden sumar y restar. Igualmente, si se multiplican su escala será el producto de las escalas. Desde el punto de vista de la codificación VHDL, estas magnitudes serán "std_logic_vector" de 16 componentes. Para efectuar operaciones con los operadores de la librería "IEEE Standard Logic Numeric" se definirán señales de tipo "signed" y se asignarán mediante una conversión de tipo. Con las señales de tipo "signed" se interpreta el vector como un entero con signo (siendo éste el bit MSB). De esta manera, ya se pueden utilizar los operadores de suma, resta y multiplicación. En el caso de sumas y restas es necesario que el ancho de bits sea el mismo, y vigilar que en las operaciones no se produzca un "overflow" no deseado, es decir que el resultado sea mayor que el número más grande alcanzable con los bits del vector (o que sea menor que el más negativo). Para el caso de las multiplicaciones, se pueden multiplicar números de distinto ancho de bits, pero el ancho de la señal donde se guarda el producto tiene que ser igual a la suma de los dos anchos de los factores. Hay que tener en cuenta que no todos los bits del resultado serán útiles debido a la extensión de signo que se realiza en la multiplicación entre números con signo en complemento a 2. Tras una multiplicación habrá que quedarse con los bits útiles según el ancho de la señal donde se quiera guardar realmente el resultado.

Una de las operaciones más habituales a lo largo de los cálculos del algoritmo es la de multiplicar una señal "signed" de 16 bits por una constante o parámetro. Para resolver estas multiplicaciones, lo que se hace es multiplicar la constante por 2^{15}. Si la constante está entre -1 y 1, podrá almacenarse en otro "signed" de 16 bits. Esto es lo habitual en el caso de la mayoría de cálculos que se tienen que realizar para las transformadas y el control. Se multiplica la señal por la constante escalada y se guarda en otra señal de 32 bits. En este resultado, se encuentra el producto multiplicado por 2^{15}. Si se quiere guardar el resultado en otro número de 16 bits, basta con descartar el bit MSB y quedarse con los 16 bits siguientes. Esto es equivalente a desplazar 15 bits a la derecha, y por tanto, dividir por 2^{15}. En el caso de que la constante a multiplicar sea mayor que 1 en valor absoluto, lo mejor es utilizar desplazamiento de bits hacia la izquierda en el caso de potencias de 2. Si no, se puede seguir el mismo método de multipicar por 2^{15} teniendo en cuenta que este factor necesitará más de 16 bits. Si se trata de una multiplicación de dos variables, lo mejor es multiplicar directamente arrastrando la multiplicación de las escalas. En cualquier caso, es posible que el producto final pueda ocupar más bits que los factores, porque con 16 bits no alcance a representar el resultado. Por ejemplo, en este sentido si se quisiera calcular una potencia, multiplicando una corriente por una tensión, la escala en enteros de la potencia será la multiplicación de ambas escalas, lo que hace posible que numéricamente sea necesario utilizar más bits para representar la potencia en enteros.

En el caso de las divisiones, la solución más óptima es utilizar desplazamientos de bits a la derecha siempre que sea posible (potencias de 2). Si no, utilizar el inverso del divisor y multiplicar. Si el divisor es una constante, esto es trivial. Si no lo es, lo mejor es tabular el inverso.

Bloque de BRAM para senos y cosenos

De cara a obtener la representación en ejes dq de las magnitudes eléctricas, es necesario calcular el seno y el coseno de la fase para poder transformar a ejes síncronos. Un posible método para conseguir esto es utilizar algún algoritmo numérico como el CORDIC. Xilinx ofrece un "IP core" que implementa este algoritmo para una serie de funciones trascendentes, entre las que se encuentran el seno y el coseno [73]. El problema de optar por esta solución es que no es óptima en velocidad, ya que introduce cierto "delay" hasta que se consigue el resultado a la salida del bloque. La opción por la que se ha optado es la de tabular la función seno y a partir de ahí obtener también el valor del coseno conociendo la propiedad trigonométrica:

$$cos(\theta) = sin(\theta + \frac{\pi}{2}) \tag{5.5}$$

A la hora de confeccionar la tabla, la primera decisión a tomar es el número de puntos y la precisión de los mismos. Como se están usando valores de 16 bits, se decide que los puntos de la tabla sean igualmente enteros con signo de 16 bits. Como se tratan de valores que en el cuerpo de los números reales están entre -1 y 1, se utiliza la escala a enteros comentada en el anterior apartado para las multiplicaciones. De esta forma, al valor máximo le correspondería el número 2^{15} (realmente sería $2^{15}-1$ que sería el número entero positivo máximo representable en 16 bits) y al mínimo -2^{15}. Cuando sea necesario multiplicar por un seno o un coseno, se realiza la multiplicación de la misma forma en la que se expuso en el anterior apartado.

En cuanto al número de puntos que debe de contener la tabla, se decidió mediante simulaciones que fuera de 4096 puntos (2^{12}). Para ello, se hicieron una serie de simulaciones en MATLAB con el modelo y el mismo tipo de control y se eligió el mínimo número para el cual no existía una degradación de los resultados obtenidos muy notable respecto a la utilización de las operaciones seno y coseno de la librería "math.h".

Conocida la representación en escala de enteros de los valores del seno y el coseno, es igualmente importante decidir como se va a trabajar con el ángulo. Es necesario escalar el valor en radianes (rad) de éste en una representación de enteros, para ello, se utilizarán los radianes binarios (brad), mediante la siguiente conversión:

$$\theta(rad) = \frac{2\pi}{2^N}\,\theta(brad) \tag{5.6}$$

donde N es el ancho de bits con el que se representará el ángulo en binario. Para obtener el ángulo en enteros, simplemente se despeja:

$$\theta(brad) = \frac{2^N}{2\pi}\,\theta(rad) \tag{5.7}$$

Esta manera de representar el ángulo en enteros tiene dos grandes ventajas. Por un lado, cuando ocurre un "overflow" del valor binario el comportamiento es idéntico a lo que ocurre con los ángulos en radianes. Es decir, se da una vuelta y se sigue por el ángulo restante tras traspasar el valor de 2π. De esta manera, 3π y π son efectivamente, el mismo ángulo. En el caso de la representación en brads ocurre lo mismo al desbordar el valor máximo de 2^N (equivalente a 2π). Lo mismo ocurre para ángulos negativos. En el caso de los radianes, el ángulo $-\pi$ es idéntico al ángulo 3π. En la representación en brads, donde los números enteros negativos son el complemento a 2 del positivo, ocurre lo mismo. Si se calcula el valor correspondiente a $-\pi$ en brads, éste sería -2^{N-1}. Si se halla su representación binaria en complemento a 2 para un ancho de bits igual a N, el número binario resultante será exactamente igual que el número binario positivo de N bits interpretado como "unsigned". Es decir, el mismo número binario será $-\pi$ si se interpreta como "signed" ó 3π si se hace lo propio como "unsigned". En la práctica, estas propiedades permiten que no haya que preocuparse por qué sucede cuando el ángulo sobrepasa el valor máximo o por si el valor se interpreta como número con signo o sin signo.

Resta decidir el número de bits con el que se representará el ángulo (N). No necesariamente tiene que ser este valor igual al número de puntos de la tabla, ya que se puede trabajar con una mayor precisión para el ángulo y hacer un desplazamiento de bits hacia la derecha de $N-12$ bits para consultar en la posición correspondiente de la tabla. Realmente, para esta aplicación podría ser suficiente una representación de $N = 16$ bits, ya que el ángulo es simplemente un valor que se inicializa a 0 y se va incrementando una cantidad en cada intervalo de sampling que depende de la frecuencia de sampling y la frecuencia que se quiere imponer en las magnitudes eléctricas (50 Hz en este caso). Sin embargo, para aplicaciones donde es necesario un PLL para sincronizarse a la fase de la red, este ángulo es producto de un control PI, por lo que mayor precisión es requerida. Como se han hecho otras aplicaciones para la medida de las tensiones de red y

sus transformadas de Clarke y Park y es posible en el futuro hacer aplicaciones de control con conexión a red, de cara a la compatibilidad, se decidió utilizar una representación de 32 bits y mantener esa representación en esta aplicación.

Establecida la base aritmética con la que se abordarán los ángulos, senos y cosenos, se procederá a describir cómo se codifica esta metodología en VHDL. Lo ideal es que toda la información que se quiera tabular en la FPGA se guarde en los bloques RAM que incluye la serie 7 de FPGAs de Xilinx y que fueron introducidos en el tercer capítulo de este proyecto. El motivo de ello es optimizar los recursos, de manera que no se gasten recursos CLBs al inferir una "distributed RAM" que reste recursos al resto del diseño. Lo mejor será utilizar recursos dedicados al almacenamiento de datos. Para que la herramienta infiera un bloque RAM para guardar los datos es necesario codificar el bloque VHDL siguiendo las recomendaciones reflejadas en [51].

Retomando las características de los bloques RAM en esta familia de FPGAs, se tratan de bloques "dual port" (hasta dos puertos de salida simultáneos) y de capacidad máxima 36 Kb. Como la tabla que se ha ideado tiene 4096 posiciones de anchura de bits igual a 16, la capacidad necesaria es de aproximadamente 64 Kb. Esto quiere decir que como mínimo el diseño más eficiente consumirá 2 bloques RAM. Adicionalmente, como son bloques "dual port", se puede tener una salida para el seno y otra desplazada $\frac{\pi}{2}$ valores para el coseno. En el caso de que se requieran más salidas, sería necesario duplicar los bloques de RAM consumidos. Esto es importante ya que para la arquitectura propuesta es necesario tanto el seno y el coseno de la fase actual $(k+0)$ como para el siguiente intervalo de sampling, en $k+1$. Una opción que se podría considerar sería a partir de la misma tabla sacar cuatro salidas, pero esto duplicaría el consumo de recursos de bloques RAM. La solución ideal es sólo consultar el valor en el horizonte más lejano que es requerido y el resto de valores guardarlos mediante registros. Por este motivo, en el diagrama de la Figura 5.12, se observa que el ángulo que entra al bloque de memoria es el ángulo en el instante $k+1$, ya que se calcula éste y el $k+0$ se consigue a través de un retraso digital de un paso con el valor que en el anterior intervalo de sampling era el $k+1$.

De esta manera, el bloque VHDL se codifica con una entrada que sea el ángulo en brads de 32 bits y las dos salidas de 16 bits que son el seno y el coseno extraídos de la tabla. Para la tabla se define el tipo "memory_type" que será un "array" de 4096 enteros en el rango de -32768 a 32767 (16 bits). Para los puntos se define una señal de este tipo que se inicializa con todos los valores del seno a tabular. Para leer la tabla, se definen sendos índices que serán enteros en el rango de 0 a 4095. De manera concurrente, estos índices se calculan a través del ángulo mediante desplazamiento de bits, sumando $\frac{\pi}{2}$ en brads para el índice del coseno. Por último, se hace una conversión a tipo "integer" para consultar la posición de la tabla. En el proceso síncrono, se asignan la salidas correspondientes, con conversión a std_logic_vector.

Para asegurar que la herramienta de síntesis infiera un bloque de RAM, es conveniente definir en la arquitectura los atributos:

Código 5.8 Atributos para bloque RAM.

```
attribute ram_style : string;
attribute ram_style of sinl : signal is "block";
```

Bloque de transformadas

Este bloque se encarga de realizar las transformadas de Clarke y de Park de las magnitudes eléctricas. Tiene como entradas las medidas que salen de la memoria compartida y el flag de aviso "Flag1" que notifica la disponibilidad en memoria de las mismas. También recibe como entrada el parámetro "deltatheta" que es la cantidad a incrementar el ángulo en cada intervalo. A la salida se obtienen las magnitudes eléctricas transformadas a ejes dq junto con una señal de dato válido que se propaga a través de los bloques junto con el resto de cálculos. Adicionalmente, es el encargado de guardar el valor actual del seno y el coseno introduciendo el retraso de un paso. Para mayor claridad, se ofrece un diagrama de bloques con las operaciones que realiza el bloque de transformadas en la Figura 5.17.

Como se puede observar, el bloque transformadas esta compuesto por varios sub-bloques que se encargan cada uno de realizar una serie de cálculos determinados. Se procede a analizar cada uno de ellos:

- **Señales**. Este bloque se encarga de sincronizar las señales justo cuando se produce un flanco en el "Flag1". Para ello la señal "Flag1" se hace pasar por un bloque de detección de flanco de subida en el top level, de manera que la señal que llega al bloque transformadas ("Flag") está activa sólo un ciclo de

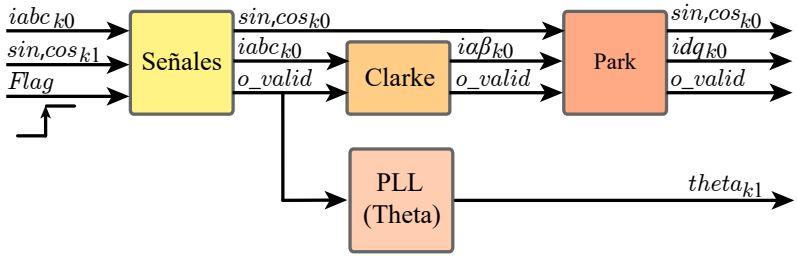

Figura 5.17 Diagrama de bloques de la transformada a ejes dq.

reloj. El bloque señales se trata entonces de un biestable que asigna las salidas cuando en un flanco de reloj, la entrada "Flag" está a 1.

Adicionalmente, se encarga de retrasar el valor del seno y del coseno. Para ello, recibe la salida de la tabla, que se corresponde con los valores en $k+1$ del intervalo anterior. En el nuevo intervalo, cuando se produce un flanco de "Flag1", la salida de la tabla todavía no se ha actualizado, por lo que se toman los valores y se guardan en los registros correspondientes a $k+0$

- **PLL (theta)**. Este bloque es el encargado de generar el ángulo. En aplicaciones donde sea necesario un PLL, en él se implementará un control PI cuyo objetivo de control es minimizar y llevar a 0 la componente "q" de la tensión en ejes síncronos. En este caso, como no es necesario sincronizarse con la red, se puede imponer la fase, por lo que simplemente se tiene una señal que se resetea a 0 con un reset global del sistema y que se incrementa una cantidad "deltatheta" cuando el sistema no está en reposo y se produce un flanco de "o_valid" (es "Flag" con un ciclo de retraso, proveniente del bloque "Señales"). Por este motivo, el ángulo que produce es realmente el ángulo en $k+1$.

- **Clarke**. Este bloque se encarga de llevar a cabo la transformación de las tensiones e intensidades en ejes naturales abc (representación en tres dimensiones) a ejes estacionarios $\alpha\beta$ (representación en el plano). En particular, se realiza la transformación de Clarke "Power Invariant". Matemáticamente esta transformación se realiza según la ecuación:

$$\begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} \tag{5.8}$$

Como se puede observar, todas las operaciones se pueden descomponer en sumas/restas y multiplicaciones que se pueden resolver en VHDL siguiendo la guía ofrecida en el subapartado donde se trata la aritmética de punto fijo, por lo que no se volverá a incidir en ellas.

- **Park**. Este bloque se encarga de llevar a cabo la transformación de las tensiones e intensidades en ejes estacionarios $\alpha\beta$ a ejes síncronos dq. Básicamente, esto permite trabajar con variables que numéricamente son cantidades de continua. Se implementa a través de la siguiente ecuación:

$$\begin{bmatrix} v_d \\ v_q \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} \tag{5.9}$$

Desde el punto de vista de la codificación VHDL este bloque es idéntico al anterior. A su salida ofrece la señal "o_valid" del bloque de transformadas, que es producto de la propagación del flanco de "Flag1" a través de los sucesivos bloques. Para asegurar que los cálculos están listos, esta señal se pasa por un bloque de detección de flanco de bajada, de manera que al bloque de control que toma las medidas transformadas llega el aviso con un "delay" adicional que asegura que los cálculos de las transformadas están listos. Cuando la señal del flanco de bajada está activa ("sflagfall"), se puede proceder a realizar los cálculos del algoritmo predictivo.

Predicción a k+1

Este bloque se encarga de realizar la primera predicción en base al modelo matemático del sistema. Conociendo el estado aplicado al comienzo del algoritmo, se puede conocer gracias a este modelo qué valores tomarán

las distintas variables al término del mismo, es decir, en el instante $k+1$. Este bloque necesitará entonces los valores de los componentes de las matrices A y B del modelo discretizado, que se pueden obtener mediante el script de Matlab del Código 2.1. Estos valores se pasan a través de la interfaz gráfica y llegan a la FPGA a través de la memoria compartida. Para el caso de este modelo, se tratan de matrices 2x2, por lo que hay ocho parámetros en total.

Como se pudo ver en el segundo capítulo de este proyecto, también es necesario conocer la tensión generada por el inversor, ya que es la entrada del sistema en representación en espacio de estados (Ecuación 2.3). Ésta no se mide directamente sino que se calcula a través de la medida de tensión del dc-link y se calcula a través del estado actual mediante los valores de la Tabla 2.2.

En VHDL, esto se hace mediante un bloque "Vinversor" dentro del bloque de Predicción a $k+1$. Básicamente, se toma el estado actual, que proviene del bloque "Disparos". Conocido el estado, con un proceso combinacional se implementa una estructura "IF...ELSE" que halla la tensión del inversor en ejes estacionarios $\alpha\beta$ normalizada respecto a v_{dc}. Para hallar la tensión real, se multiplica mediante una asignación concurrente por el valor de tensión del dc-link. Tras este cálculo, sólo resta transformar esta tensión a dq de igual manera que en el bloque de "Park".

Tras el cálculo de la tensión generada por el inversor en ejes dq, se puede proceder al cálculo del modelo predictivo. Éste se compone de una serie de multiplicaciones y sumas que se codifican en el bloque "MPC" de manera similar a lo visto para otros cálculos. Se hace mediante asignaciones concurrentes en la arquitectura. Igualmente, la señal de dato válido se propaga hasta la salida del bloque y se hace pasar por otro bloque de detección de flanco para avisar al bloque de predicción en $k+2$ de la disponibilidad de los cálculos

5.4.3 Programación de la parte hardware III. Alternativas para la predicción a k+2

Este apartado se dedicará a la descripción de las dos alternativas consideradas para la implementación de la predicción a $k+2$. En este puntom el sistema ya dispone de la predicción en el instante $k+1$, que se ha calculado a través de la información del sistema conocida en $k+0$. Ahora es necesario llevar a cabo un algoritmo de tipo "ESA" que compute todas las posibles predicciones del sistema a $k+2$. Para ello es necesario comprobar cada uno de los estados posibles del sistema, calculando la predicción correspondiente y una función de coste que quedó definida en la Ecuación 2.9. Por simplicidad, no se va a considerar la penalización de las conmutaciones. Conociendo los distintos costes que supondría aplicar cada uno de los posibles estado, se elige el estado que ocasione el menor de ellos, y que atendiendo a la formulación de la función de coste, será el que minimice la distancia entre la intensidad de salida y su referencia.

La necesidad de realizar todos estos cálculos es lo que genera el alto coste computacional de este tipo de algoritmos, que crece de forma exponencial con las dimensiones del sistema y el número de horizontes de predicción que se quieran implementar. Por ello, su implementación no es trivial y puede dar lugar a distintos tipos de posibles soluciones. En particular, en este Trabajo Fin de Máster se explorarán dos de ellas. Una de ellas será una implementación donde las distintas predicciones se hagan en paralelo, mientras que en la otra, las predicciones se harán de manera secuencial. Es de esperar que entre ambas haya diferencias importantes en tiempo de cálculo y consumo de recursos, lo cual dará lugar a un análisis que se expondrá en el siguiente capítulo.

Metodología en paralelo

Mediante esta implementación, las distintas predicciones a $k+2$ se hacen todas mediante bloques en paralelo. Como se puede observar en la Figura 5.18 lo que se hace es replicar la estructura del bloque de predicción a $k+1$, tantas veces como estados posibles. En este caso se tienen ocho estados, de los cuales el estado "000" y el "111" deben de ofrecer el mismo resultado.

Desde el punto de vista de la codificación VHDL, se define como componente un bloque de predicción a $k+2$ que es idéntico al de predicción en $k+1$ salvo los cálculos que se pueden sacar fuera porque no dependen de cada estado posible. De esta forma, se define un top level en el que se instancian ocho de estos componentes, cada uno con idénticas entradas salvo el estado. Los cálculos del modelo predictivo que dependen de las corrientes de salida, se efectúan de manera concurrente en este top, ya que no dependen del estado y sus resultados se ofrecen como entradas a los ocho bloques. Los que dependen de la tensión del inversor se realizan de manera idéntica al caso de la predicción en $k+1$. Para cada uno de estos bloques, se define un nuevo sub-bloque que se encarga de calcular el coste asociado a su estado. Para ello, se codifica la ecuación Ecuación 2.9 de manera similar a como se han abordado este tipo de operaciones en el resto de bloques.

La salida del top level para predicciones a $k+2$ son el coste calculado por cada uno de los bloques. Cada uno de ellos definidos como números de 32 bits. Estos costes van a la entrada de un bloque comparador

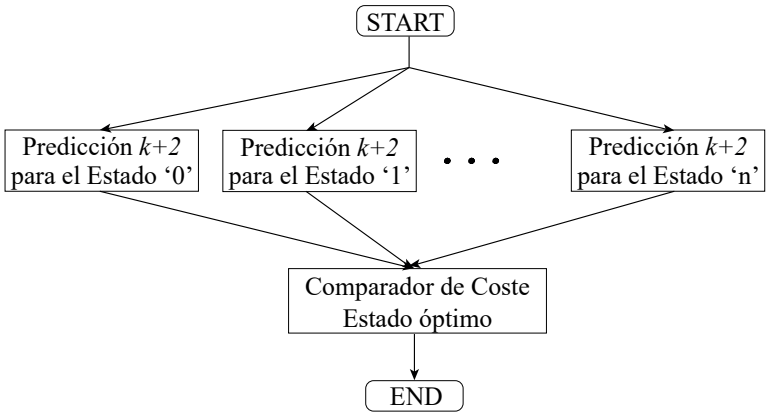

Figura 5.18 Diagrama de bloques de la implementación en paralelo de la predicción a $k+2$.

que es necesario implementar para esta metodología en paralelo. Básicamente lo que hace este bloque es comparar mediante un proceso combinacional cada uno de ellos y seleccionar el coste menor. El estado correspondiente al coste seleccionado será el estado óptimo y se envía al bloque de "Disparos" para que se aplique al final del intervalo de control actual.

Una de las particularidades de esta metodología es que no es necesario colocar un biestable al final de la predicción en $k+1$ que sincronice las salidas, ya que todos los cálculos se pueden realizar de manera combinacional, propagándose hasta la selección del estado óptimo, que es guardado en el bloque "Disparos". Al evitar la espera para sincronizarse con la señal de reloj a la salida de cada bloque se puede reducir el tiempo de cálculo.

Por otro lado, pese a que esta metodología pueda alcanzar los resultados más rápidos, es de esperar un mayor consumo de recursos al paralelizar los cálculos.

Metodología secuencial

En oposición a la metodología anteriormente descrita, ahora se optará por un diseño completamente secuencial para predicción $k+2$. Mediante este método, se define una segunda máquina de estados que se encarga de hacer los cálculos de cada estado de manera secuencial. Un diagrama de bloques de las operaciones que realiza esta máquina de estados se ofrece en la Figura 5.19.

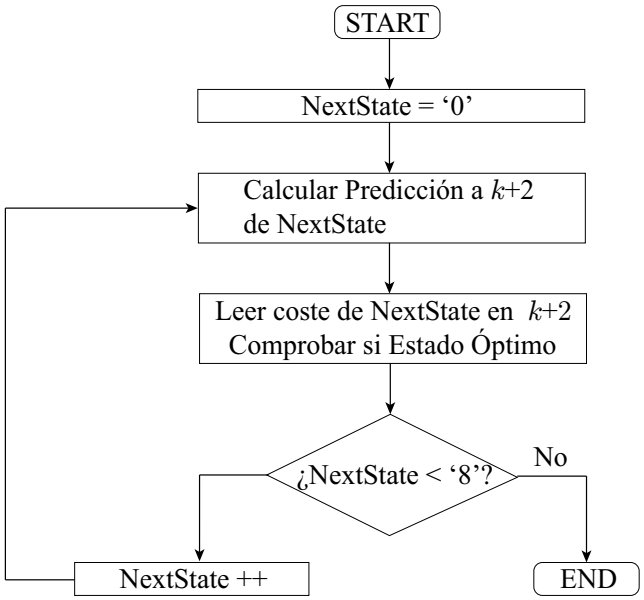

Figura 5.19 Diagrama de bloques de la implementación secuencial de la predicción a $k+2$.

Como se puede observar, ésta se encuentra formada por cuatro estados: "Reposo", "Cálculo", "Lectura" y "Update".

Por defecto, la FSM estará en el estado de "Reposo", a la espera de que se complete la predicción a $k+1$. Este hecho se notifica mediante el flag de aviso de que la predicción $k+1$ ha finalizado. En este caso es importante asegurar que todos los cálculos están listos al mismo tiempo cuando empieza a trabajar la FSM de la predicción a $k+2$. Por este motivo, ahora sí se define un biestable a la salida del bloque $k+1$ que sincroniza todas las salidas con la señal de reloj, incluida la señal de dato válido. En el estado de reposo, se aprovecha para resetear el valor del costo máximo al valor tope alcanzable con 32 bits y para dejar el primer estado a calcular como el estado "000".

El estado de "Cálculo" se alcanza si se produce un flanco de bajada en el flag. Éste es realmente un estado transitorio donde se espera un ciclo de reloj a que la predicciones a $k+2$ del estado impuesto desde la FSM estén disponibles. Para ello, se define un único bloque de predicción a $k+2$ que es idéntico al bloque de predicción a $k+1$, con la salvedad de que incluye el cálculo del coste en su top level mediante operaciones concurrentes. Éste recibe el estado a calcular desde la FSM y devuelve el coste, listo en el siguiente ciclo de reloj.

En el siguiente estado, "Lectura", se lee el coste calculado y se compara con el valor del costo máximo actual. Si es menor, se machaca el valor del coste máximo con el coste calculado y se guarda el estado actual como óptimo. En cualquier caso, se actualiza el estado sumando una unidad para calcular el siguiente estado.

Por último, el estado "Update" comprueba si el estado actualizado ha desbordado, volviendo de nuevo al estado "000". Este hecho significa que todos los estados han sido ya comprobados, y por lo tanto, ya se tiene el estado óptimo. Se puede volver al estado de reposo. En caso contrario, se volverá al estado de "Cálculo" y se sigue repitiendo el proceso, comprobando cada estado hasta el último.

Como se puede deducir, ya no es necesario implementar un bloque comparador de 8 entradas ya que sólo es necesario comparar dos valores en cada pasada de la FSM. En general, es de esperar que el consumo en recursos de esta solución sea mucho menor, pero a costa de un tiempo de cálculo mayor.

5.4.4 Programación de la parte hardware IV. Top Level y constraints

En este punto, se ha completado la descripción de los bloques que conforman el diseño propuesto para la FPGA. Resta definir un top level en el que todos estos bloques se definan como componentes y se cableen mediante señales en su arquitectura. Igualmente, será necesario definir la entidad con los puertos de entrada y salida de todo el sistema. Éstas se tratan de todos los botones físicos (reset, marcha/paro y despeje de error), señales de salida hacia el convertidor (A,B,C, Enable y Chopper), entradas del convertidor (Error), LEDs y señal de reloj. Para este caso, no se utiliza el Chopper, por lo que esta salida se mantiene a "0" en todo momento.

Por último, todas estas señales deben de estar correctamente asociadas a su pin físico en el archivo de constraints. Adicionalmente, se puede definir el nivel de tensión al que operarán estos pines y el pulldown o pullup para las entradas. Se adjunta el archivo de constraints, ya que es la forma más intuitiva de poder describir la distribución de pines de entrada y salida utilizada:

Código 5.9 Archivo de constraints.

```
set_property PACKAGE_PIN Y9 [get_ports {clk_100Mhz}];
    # "clk_100Mhz" - Senal de reloj
create_clock -name global_clk -period 10 ns [get_ports {clk_100Mhz}];

set_property PACKAGE_PIN P16 [get_ports {rst}];
    # "BTNC" - Boton de reset
set_property PACKAGE_PIN N15 [get_ports {switch}];
    # "BTNL" - Boton de marcha/paro
set_property PACKAGE_PIN R18 [get_ports {errorboton}];
    # "BTNR" - Boton de despeje de error
set_property PACKAGE_PIN AA8 [get_ports {error}];
    # "JA10" - Señal de error convertidor (PMODA)


set_property PACKAGE_PIN T22 [get_ports {ledC1}];
    # "LD0" - LED visualizacion disparo rama C
```

```
set_property PACKAGE_PIN T21 [get_ports {ledB1}];
    # "LD1" - LED visualizacion disparo rama B
set_property PACKAGE_PIN U22 [get_ports {ledA1}];
    # "LD2" - LED visualizacion disparo rama A

set_property PACKAGE_PIN V9 [get_ports {C1}];
    # "JB9" - Señal de disparo rama C (PMODB)
set_property PACKAGE_PIN W10 [get_ports {B1}];
    # "JB8" - Señal de disparo rama B (PMODB)
set_property PACKAGE_PIN V12 [get_ports {A1}];
    # "JB7" - Señal de disparo rama A (PMODB)
set_property PACKAGE_PIN W8 [get_ports {chopper}];
    # "JB4" - Señal de disparo chopper (PMODB)

set_property PACKAGE_PIN U14 [get_ports {led_enable}];
    # "LD7" - LED visualizacion Enable
set_property PACKAGE_PIN V8 [get_ports {enable}];
    # "JB10" - Señal de Enable convertidor (PMODB)

set_property PACKAGE_PIN U19 [get_ports {led_error}];
    # "LD6" - LED visualizacion Error
set_property PACKAGE_PIN W22 [get_ports {led_desp}];
    # "LD5" - LED visualizacion Despejar error
set_property PACKAGE_PIN V22 [get_ports {Pinout}];
    # "LD4" - LED para medir tiempos de cálculo

set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 13]];
set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 33]];
set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 34]];
set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 35]];
set_property PULLDOWN true [get_ports {rst}];
set_property PULLDOWN true [get_ports {switch}];
set_property PULLDOWN true [get_ports {errorboton}];
set_property PULLDOWN true [get_ports {error}];
```

Con todo esto, el diseño hardware está completado, por lo que se puede proceder a la generación del "bitstream" que se utiliza como base para el diseño de las aplicaciones software.

5.4.5 Programación de la aplicación baremetal

Este apartado se dedicará a la programación en lenguaje C de la aplicación que ejecutará el ARM1. Todos los pasos a la hora de crear el proyecto de aplicación y configurar parámetros como el "linker script" son los ya comentados en los primeros apartados de este capítulo, por lo que este apartado estará centrado únicamente en explicar la codificación del mismo.

Función principal "main"

La manera en la que se ha estructurado esta aplicación es con una función principal "main" que se encarga de configurar e inicializar todos los periféricos y variables utilizadas. Para ello, se utilizan las definiciones de la librería "xparameters.h", que incluye las definiciones más importantes del proyecto. La lista de periféricos configurados con sus respectivas librerías son:

- **Caché L1 de la Memoria OCM**. Necesario deshabilitarla para el modo AMP mediante la función "Xil_SetTlbAttributes(0xFFFF0000,0x04de2)". Esta función está en la librería "xil_mmu.h".
- **GPIO MIO**. La GPIO asociada al MIO se inicializa mediante las funciones de la librería "xgpiops.h". Es importante diferenciar esta librería de la "xgpio.h". La primera sirve para configurar las GPIO MIO y la segunda para las GPIO definidas a través de EMIO. Las funciones y la manera de configurarlas no es la misma. En este caso, es necesario definir una variable del tipo "XGpioPs" que se inicializa

mediante las funciones "GPIO_Config" y "XGpioPs_CfgInitialize". Una vez esta variable contiene toda la información referente al diseño hardware, es necesario configurar cada pin mediante las funciones:

  - **XGpioPs_SetDirectionPin**. Se utiliza para configurar el pin dado como entrada o como salida digital.
  - **XGpioPs_SetOutputEnablePin**. Habilita la salida del pin digital.

  Las funciones **"XGpioPs_ReadPin"** y **"XGpioPs_WritePin"** permiten la lectura/escritura de las GPIOs configuradas como tal.

- **SPI**. La inicialización del SPI sigue la misma estructura que la de las GPIO, definiendo una variable "XSpiPs" y utilizando las funciones análogas de la librería "xspips.h". Para configurar el SPI para funcionar en los parámetros requeridos, se utilizan las funciones:
  - **XSpiPs_SetOptions**. Se utiliza para configurar el SPI de la Zynq-7000 como maestro. Además, se configura el modo "FORCE_SSELECT". Esto se debe a que el controlador SPI presente en el Zynq-7000 está pensado para transacciones de 8 bits, por lo que en modo automático, cortará la comunicación deshabilitando el Chip Select a partir del octavo bit. Para que esto no ocurra, es necesario definir este modo que permite utilizar una longitud de trama más larga. La desventaja de este método es que la deshabilitación del Chip Select no la hace el controlador hardware, si no que llega la señal por software, añadiendo un delay adicional entre el final de la trama y la deshabilitación de la señal CS.
  - **XSpiPs_SetClkPrescaler**. La señal de reloj del SPI que se configuró en el hardware, se puede preescalar con un divisor potencia de 2. En este caso se elige el valor de 8 para una frecuencia de reloj cercana a los 20 MHz que permite el ADC.
  - **XSpiPs_SetSlaveSelect**. Elige el esclavo con el que se comunicará el SPI. Como en este caso sólo hay uno, se predefine al esclavo 0.
- **Interrupción**. De nuevo, la configuración de este periférico sigue una estructura similar, mediante la definición de una variable "XScuGic" y el uso de las funciones de la librería "xscugic.h", correspondiente al "General Interrupt Controller" (GIC) del Zynq-7000. También se utiliza la librería "xil_exception.h" para configurar la rutina de interrupción asociada.
  - **XScuGic_Connect**. Se utiliza para conectar el ID correspondiente de la interrupción PL-PS configurada, es decir, la interrupción privada "Core1_nIRQ", con la función de interrupción (handler) que se ejecutará. La ID es la número 31, también disponible mediante el "#define" "XPS_IRQ_INT_ID".
  - **XScuGic_Enable**. Habilita en la GIC el puerto de interrupción correspondiente a la interrupción definida.
  - **Xil_ExceptionInit**. Habilita el manejo de excepciones del procesador ARM.
  - **Xil_ExceptionRegisterHandler**. Asocia el manejador de excepciones del GIC (parte del Zynq-7000, como si fuera un periférico) al procesador ARM.
  - **Xil_ExceptionEnable**. Activa el manejo de excepciones del ARM.

Una vez este proceso de configuración inicial ha finalizado, el "main" entra en un bucle "while(1)" que se encarga de comprobar si la aplicación de Linux ha actualizado el valor de algún parámetro en la memoria compartida. Ambas aplicaciones ejecutadas en sendos núcleos, comparten información a través de la On-Chip Memory (OCM). Concretamente, se utiliza el sector 3 de esta memoria, con direcciones en el rango de "0xFFFF0000" a "0xFFFFFFFF". Mientras que para la compartición de memoria entre FPGA y ARM1 la opción es utilizar la RAM instalada en la Zedboard en el controlador DDR del Zynq-7000 (en el rango de direcciones del periférico definido en la parte hardware), la opción de la OCM (dentro del integrado) es más rápida y tiene menos latencia que la memoria RAM externa y por lo tanto su uso es preferible para compartir información entre los dos núcleos ARM. Para las lecturas y escrituras en estas posiciones de memoria de la OCM, la metodología seguida es la definición de una serie de punteros que apuntan cada uno a la dirección correspondiente.

En caso de detectar el cambio de algún parámetro en la memoria, este se actualiza internamente y en caso de ser un parámetro utilizado por la FPGA, se escribe en el registro correspondiente de la RAM mediante las funciones de lectura/escrituras existentes en el driver generado para el periférico creado. Concretamente,

éstas se encontrarán en la librería "ADCreg_100.h", que debe de aparecer en el BSP creado para la aplicación. Si no, será necesario actualizar los repositorios del SDK, añadiendo la carpeta del proyecto IP core del periférico creado.

Rutina de Interrupción

Cuando llega una interrupción desde la FPGA al puerto configurado, el ARM empieza a ejecutar la rutina de interrupción asociada en la configuración del GIC. En este caso, la rutina de interrupción se encarga de ejecutar las tareas críticas de gestionar las medidas del ADC y escribir en la RAM los valores de estas medidas para que la FPGA pueda leerlas. Escribe también el Flag1 cuando todas estas medidas están disponibles en memoria. Finalmente, se aprovecha la rutina de interrupción para escribir en la OCM las tablas de datos con las variables que se representan en la interfaz gráfica. Esto incluye la gestión de las peticiones de "trigger". Por lo tanto, se divide la rutina de interrupción en tres secciones:

- **Gestión del protocolo SPI para comunicación con el ADC**. Se utiliza la función de la librería "xspips.h" **XSpiPs_PolledTransfer**. Ésta recibe el puntero a la variable de la SPI definida en la configuración y los dos punteros a los sendos búffer de lectura y escritura. El último argumento es el número de bytes de la transacción, que en este caso es de 2. Como está configurado el modo "FORCE_SSELECT", el CS se deshabilita al término del segundo byte de la trama.
  - **WriteBuffer**. El búffer de escritura se trata de la trama de entrada que el ADC leerá por su puerto SDI. Contiene el número binario de 16 bits con la configuración del ADC y el canal a seleccionar para la siguiente lectura (su resultado se obtiene en la segunda lectura posterior), tal y como se recogió en la Tabla 4.1. Se configura el modo manual con rango de 2.5 V en todas las transacciones.
  - **ReadBuffer**. El búffer de lectura es en el que se recibe la información que el ADC manda a través de su puerto SDO, contiene el resultado de la conversión pedida dos transacciones antes. Éste se trata de un array de dos componentes "unsigned" de 8 bits. Por lo que para obtener el dato completo es necesario sumar ambos componentes en un entero de 16 bits con el desplazamiento binario de 8 bits para los MSB (y multiplicando por una máscara que quite el número de canal del dato numérico).

  Estas operaciones se hacen para los cuatro canales que se quieren medir. En el caso de la medida de tensión de continua se corrige también su escala para expresarla en la escala de las intensidades.
- **Escritura en la RAM**. Con todos los valores de las medidas expresados en la misma escala y el offset corregido, estos se escriben en los registros correspondientes del periférico creado en la implementación hardware. Se utiliza la función generada en su driver **"ADCREG_100_mWriteReg"**. Ésta recibe la dirección base del periférico y el offset respecto de ésta del registro en el que se quiere escribir. El último argumento se trata del entero con signo de 32 bits a escribir en ese registro. Se proporcionan dos medidas de 16 bits en cada uno, con una de ellas desplazada 16 bits a la izquierda. La última escritura se trata del "Flag1".
- **Relleno de las tablas**. En cuanto al envío de datos para la interfaz gráfica, se han provisto 12 tablas de 600 puntos. Estas tablas se definen mediante su dirección base en memoria y se van escribiendo incrementando en cada pasada el índice multiplicado por 4 hasta que están completas. El índice se multiplica por cuatro, ya que cada dirección apunta a un byte de memoria. Los distintos datos se transforman a su representación real y se hace "typecast" a flotante (4 bytes) antes de escribirlos en la tabla. De esta manera, a la interfaz ya llegan los datos como flotantes. Entre estos datos, algunos de ellos provienen de las medidas y otros son cálculos internos de la FPGA cableados a la RAM, por lo que se leen antes mediante la función **"ADCREG_100_mReadReg"**.

  Hay que tener en cuenta que no todos los datos se escriben en las tablas, ya que la resolución está limitada a 600 puntos. Si según la frecuencia de muestreo elegida y el número de ciclos que se quiere representar en la interfaz gráfica, el número de puntos real supera al existente en las tablas, se realiza un submuestreo. Esto quiere decir que se salta un determinado número de muestras que se calcula conociendo el número de puntos disponibles y la frecuencia de muestreo elegida (división de ésta entre 50 Hz y multiplicando por el número de ciclos a representar). Dividiendo este resultado entre los 600 puntos, se puede conocer el número de puntos que hay que saltarse.

  Cuando las tablas se completan, si no hay petición de envío de datos, éstas se siguen rellenando desde el principio. Si se ha pedido un envío de datos, las tablas se congelan en este punto y no se siguen

rellenando puntos. Se envía a la aplicación de Linux una señal a través de la OCM de que las tablas están rellenas y se puede proceder a enviar las tablas de la OCM a la interfaz. Cuando todos los datos se han enviado a la interfaz con éxito, la aplicación de Linux vuelve a enviar una petición de desbloquear las tablas y éstas pueden seguir rellenándose desde 0 de nuevo.

Otro caso particular sucede cuando desde la interfaz se ha pedido un trigger de alguna señal. Cuando esto sucede, se notifica a través de la OCM y el ARM1 empieza desde ese momento a comprobar si la señal que corresponda supera el límite de trigger definido en la interfaz. Mientras no suceda, la tabla sigue rellenándose normalmente. Cuando se cumple la condición del trigger, es necesario mover las tablas, de manera que el dato más reciente quede centrado en el punto definido desde la interfaz. Hecha esta operación, las tablas se siguen rellenando hasta su límite. En este punto, las tablas se dejan bloqueadas hasta que sean leídas completamente.

5.4.6 Programación de la aplicación de Linux

Este apartado se dedicará a la programación en lenguaje C de la aplicación que ejecutará el sistema operativo Linux en el ARM0. El desarrollo de esta aplicación se ha basado en la desarrollada en [69]. Básicamente, se trata de una aplicación servidor que define un socket con protocolo TCP/IP para conectarse a un cliente, que en este caso será una aplicación codificada en Java y ejecutada en un PC. Para establecer esta conexión, es necesario conocer la dirección IP asignada a la Zedboard por el router al que se conecta y utilizarla al establecer la conexión desde la GUI.

En esta aplicación, se han de definir los correspondientes punteros apuntando a la misma dirección que los definidos en la aplicación del ARM1. De esta manera, todas tablas de datos, parámetros y señales de aviso que se utilicen estarán en la misma dirección de la OCM y ambos núcleos podrán compartir información sin problemas.

Cuando la creación del socket se completa, esta aplicación entra en un bucle "while(1)" donde queda en espera a que llegue una petición desde la interfaz gráfica por el socket. Cuando esto sucede, se lee el paquete de datos llegado y se guarda en una estructura que incluye un header con el tipo de petición y un array de flotantes de tamaño "SIZE_DATA" (en este caso, 600 puntos). Mediante una estructura "switch...case", se decide la acción a tomar según el valor del header del paquete de datos. De esta manera, si se trata de un envío de parámetros, se machacan los parámetros correspondientes en memoria y se muestran por pantalla a través de la consola conectada al Linux para avisar del cambio. Igualmente, si llega alguna notificación de congelar tabla o de trigger, se activa la señal correspondiente para avisar de la petición del cliente al ARM1. La petición "ASKTRIGGER", se realiza periódicamente una vez pedido un trigger desde la interfaz, para preguntar si el trigger se ha producido y los datos están listos. En caso de llegar una petición de envío de datos, se lee la tabla correspondiente y se rellena el paquete de datos con los valores en flotante. Por último, se envían a través del socket.

5.5 Interfaz gráfica

En este punto, la programación de la plataforma Zedboard queda completamente descrita. Resta introducir la interfaz gráfica utilizada para la representación de la información y la interacción con el sistema. Ésta interfaz está programada en Java, y se basa en un diseño de Abraham Márquez Alcaide que se readaptó para el Trabajo Fin de Grado [69]. En este caso, se ha ampliado el diseño, readaptándolo a las necesidades de este proyecto e introduciendo nuevas funcionalidades como la posibilidad de pedir un trigger o la de escribir los datos recibidos en un fichero de texto para poder tratarlos en un software como MATLAB.

Desde el punto de vista de la codificación en Java, la interfaz se basa en la librería "JFreeChart" para la elaboración de gráficas. Básicamente, la interfaz se divide en los paneles norte, este, oeste y centro, en los que se van colocando objetos de Java con los que el usuario puede interactuar. En la Figura 5.20 se puede observar la distribución de la misma, que se procederá a comentar:

- **Panel norte para el envío de comandos**. Este panel implementa los botones de interacción básica con el sistema.
  - **Start, Reset, RmvError**. Son los tres botones análogos a la botonera física para interactuar con la FSM del sistema implementada en la FPGA. Los comandos se propagan con sendas escrituras en memoria hasta la FPGA.
  - **Read,Write**. Para la lectura y escritura de parámetros.

  - **Trigger**. Envía una petición de Trigger, enviando los parámetros asociados al mismo (panel "Limits").
  - **IP, Puerto**. Permite introducir la dirección IP asignada a la Zedboard para la correcta conexión del socket. El puerto queda predefinido en ambas aplicaciones
  - **Pestaña controller**. Permite tener varias pestañas para comunicarse con distintos sistemas. En este caso no se utiliza porque solo hay uno.
  - **Connect**. Establece la conexión a través del socket.
  - **Receive**. Pide el envío de un sólo paquete de datos.
  - **Continuous**. Pide el envío cíclico de paquetes de datos mediante un timer, hasta que se deseleccione.

- **Panel ADC**. Permite la escritura de los parámetros de calibración del ADC.
- **Panel Limits**. Configura los parámetros del Trigger. Se puede configurar el valor para el que se dispara el trigger y en qué punto de la gráfica se quiere representar el instante del salto. Éste punto se introduce como un porcentaje del eje X.
- **Panel Parameters**. Configura los parámetros generales del sistema. Entre ellos se encuentran los ciclos a representar en la gráfica (determina el submuestreo en el ARM1), la frecuencia de muestreo del sistema, y parámetros del control como las referencias, los valores de las componentes de las matrices del modelo discretizado y el parámetro lambda de penalización de las conmutaciones.
- **Panel Controller**. Permite elegir qué canales se van a mostrar en la gráfica cuando se solicite un envío de datos. Se configuran ocho de ellos en la pestaña "Mag" y cuatro de ellos en la pestaña "Aux" (correspondientes a las doce tablas definidas en la implementación en la Zynq-7000). Los cuadros de la derecha permiten configurar ganancias y offsets internos de la interfaz para la representación de los datos (no se envían al ADC).
- **Panel Oscilloscope**. Éste es el panel central donde se representan las medidas. El eje de abscisas se corresponde con el tiempo en milisegundos y el eje de ordenadas se corresponde con el valor en números reales de las magnitudes representadas. La gráfica se autoescala en cada refresco y también tiene capacidades de Zoom manual.

En la Figura 5.21 se adjunta una captura de la interfaz en funcionamiento, junto con los datos recepcionados que se han exportado y representado en MATLAB. En esta figura se aprecian cinco señales, la corrientes en ejes *abc* y en ejes *dq*. Se ha pedido un trigger en la señal 0, que corresponde a la intensidad en la fase A con valor de 1.9 A y centrado en la mitad de la gráfica. De esta manera, se puede observar un transitorio del sistema.

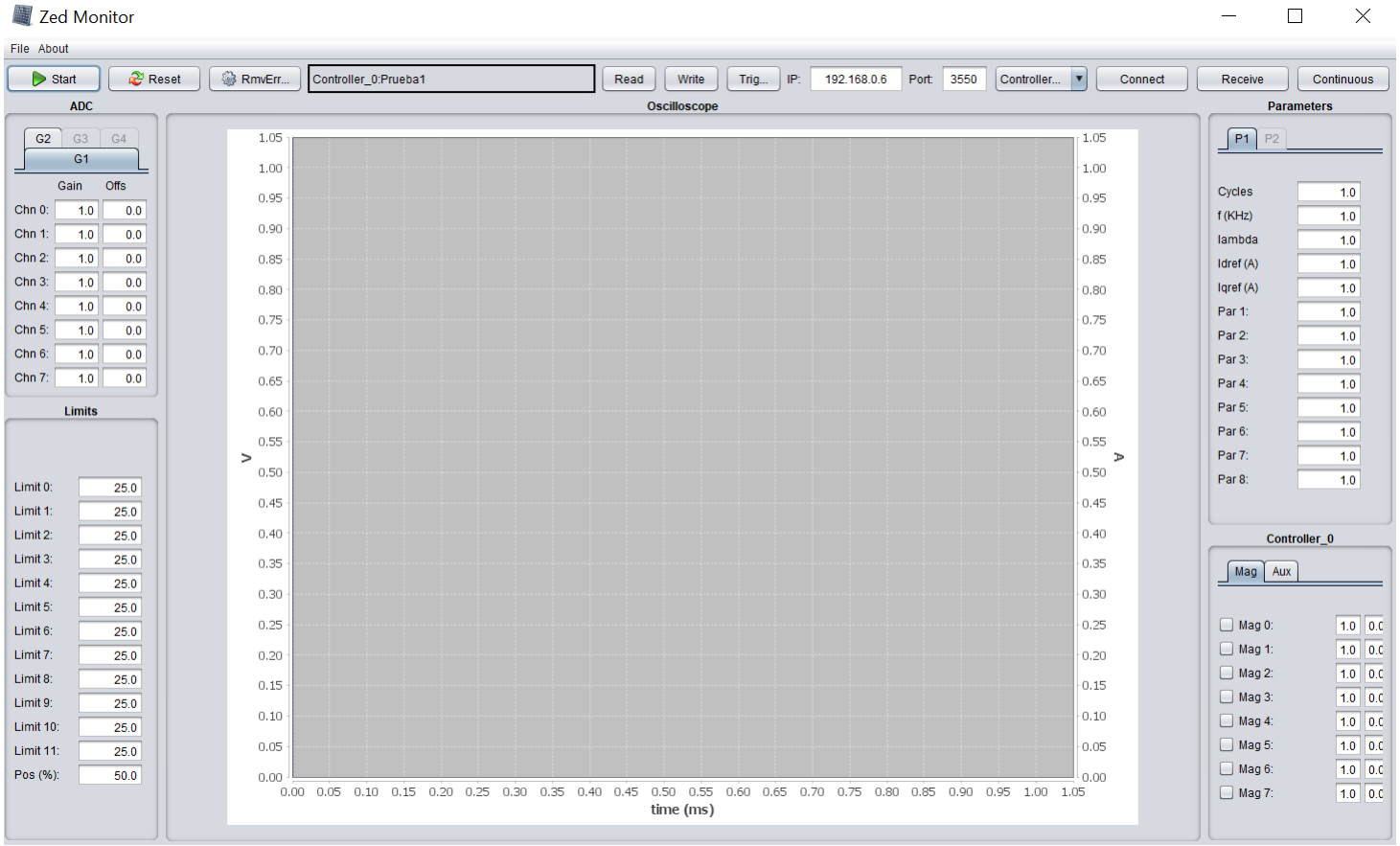

Figura 5.20 Interfaz de usuario diseñada en Java.

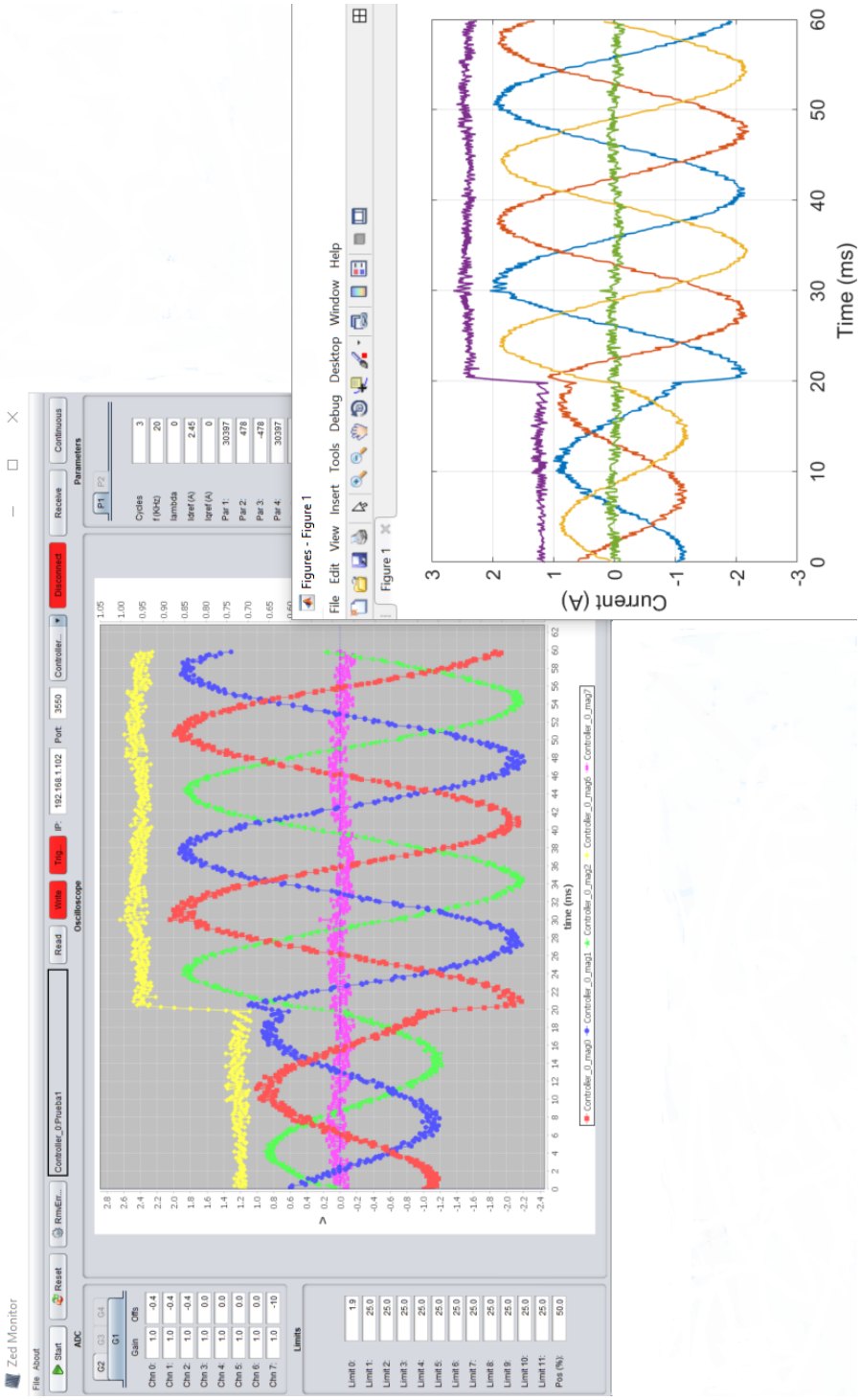

Figura 5.21 Interfaz de usuario en funcionamiento.

6 Resultados experimentales y análisis de los resultados

En capítulo se detallará la validación experimental de la arquitectura propuesta para la plataforma de control Zynq-7000. Se comprobará la viabilidad de la misma mediante experimentos en el laboratorio con el equipo real y se mostrarán los resultados obtenidos. Así mismo, se llevará a cabo un análisis del diseño desde el punto de vista de tiempo de computación y de recursos de FPGA consumidos, dando lugar a una breve discusión en torno a las dos posibles alternativas de diseño y a las ventajas y desventajas asociadas a cada una de ellas. De cara a poder contrastar la utilidad y la importancia de los resultados obtenidos, se comparará el tiempo de ejecución obtenido con esta plataforma con lo que se puede obtener en una plataforma de control más convencional de tipo DSP.

6.1 Resultados experimentales

Este apartado se centrará en las pruebas de laboratorio realizadas, detallando el equipo disponible y el montaje del mismo. Por último, se expondrán los resultados obtenidos en éstas.

6.1.1 Equipos de laboratorio

Lo primero de todo será montar la plataforma de control Zedboard con todas sus placas de expansión, tal y como se aprecia en la (ver Figura 6.1).

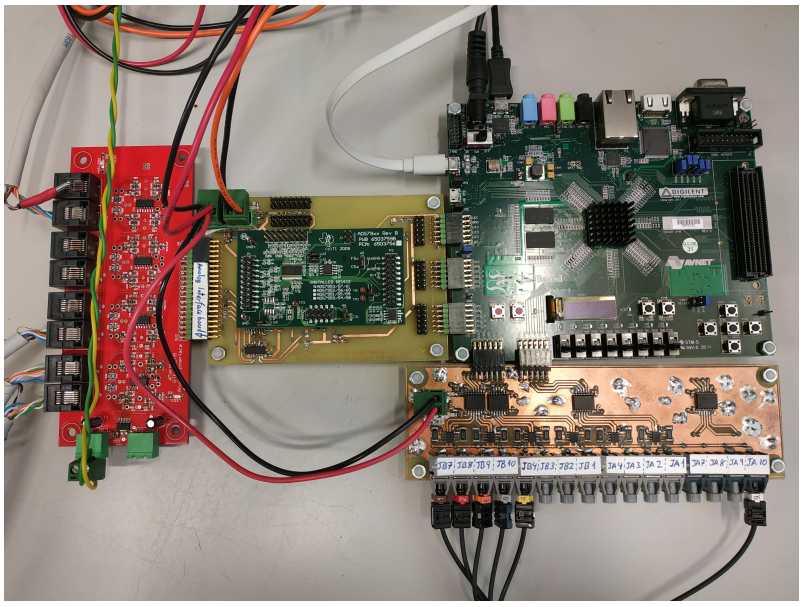

Figura 6.1 Zedboard con placas de expansión fabricadas en el laboratorio.

Todas las placas que se conectan a la Zedboard han sido diseñadas y montadas en el laboratorio. Las placas que se conectan a la izquierda de la Zedboard, por los puertos PMOD JE,JC y JD, son las correspondientes al ADC y placa roja de adaptación de sensores que fueron introducidas en el capítulo cuarto de este documento, donde se repasó la funcionalidad de ambas. Como se puede observar, a la entrada de la placa de adaptación de medidas, se disponen los cables con conectores RJ11 de 6 pines que conectan esta placa con los sensores. Por un lado, el cable superior, proviene del sensor de tensión instalado en el dc-link del inversor (ver Figura 4.1) y se conecta a los canales de continua de la placa roja. Por otro lado, los tres cables inferiores provienen de una placa con tres sensores de corriente por los que se pasan los tres cables de la manguera trifásica que conecta el inversor con la carga *RL*. Se conectan a los canales de alterna de la placa roja para medir las corrientes de salida del inversor.

En cuanto a la carga que se utilizará, se dispone en el laboratorio de una carga trifásica *RL* reconfigurable, que se muestra en la Figura 6.2. Básicamente, esta carga está formada por resistencias de 60 Ω y bobinas de 20 *mH* conectadas en paralelo que mediante los contactores del panel principal se pueden reconectar de diferentes formas. De esta manera, de solo conectar un set de cargas, se tendrían 60 Ω y 20 *mH*, mientras que de conectar todas las cargas se tendrían 15 Ω y 10 *mH*. Esta carga se puede conectar a un convertidor mediante una manguera trifásica, que tiene incorporados los sensores de corriente.

Figura 6.2 Carga *RL* trifásica del laboratorio.

Por otro lado, la otra placa conectada a la Zedboard, en su parte inferior por los puertos PMOD JA y JB, se trata de la placa de conversión de señales eléctricas a fibra óptica. Básicamente, esta placa toma todos los pines digitales de estos puertos y los transforma a señal óptica que puede ser interpretada por el driver del convertidor. En este caso se tienen como salidas las señales de disparo de las tres ramas (la negada la saca el driver del inversor), la del chopper (no se utiliza en esta aplicación) y la señal de Enable del convertidor que habilita las señales de disparo. Por último, se tiene una entrada de error proveniente del driver del convertidor.

Para el dc-link, el inversor consta de conectores para aplicar una tensión de continua, para lo cual se utilizará una fuente de tensión de 1500 *W* que puede proporcionar hasta 600 *V* y 2.5 *A* de continua.

Por último, se cuenta con un analizador de red "Fluke 434 Series II" que mediante las correspondientes sondas de corriente permite muestrear las señales generadas y hacer análisis de armónicos.

6.1.2 Resultados de los experimentos

Para los experimentos que se mostrarán a continuación, se han utilizado los parámetros que se exponen en la Tabla 6.1.

Tabla 6.1 Variables y parámetros del sistema.

Variable	Descripción
Frecuencia de Muestreo F_{sw}	40 kHz
Frecuencia de las corrientes de referencia F	50 Hz
Inductancia de la carga L	20 mH
Resistencia de la carga R	30 Ω
Tensión del dc-link v_{dc}	140 V
Parámetro limitador de conmutaciones λ_1	0

Establecidos los parámetros bajo los que se desarrollará el experimento, lo primero de todo será comprobar la viabilidad del del sistema. Es decir, que las corrientes se regulan al valor establecido en la referencia y que el control se comporta adecuadamente tanto en el régimen transitorio como en el permanente. Además, ambas alternativas propuestas para la implementación FPGA, tanto el desarrollo paralelo como el secuencial deben de tener el mismo comportamiento desde el punto de vista del control. Se ofrecen las gráficas para ambas implementaciones en la Figura 6.3 y la Figura 6.4.En estas gráficas, se muestran las corrientes en coordenadas abc y dq junto con las referencias en dq. Se puede observar un transitorio de 1 A a 2 A de la amplitud de la corriente en ejes abc en el punto temporal $t = 40$ ms. Las referencias en dq son de 0 para la componente en q y salto de 1.21 A a 2.45 A para la componente d.

Adicionalmente, mediante el analizador de red Fluke, se puede medir el contenido espectral de las corrientes hasta el armónico número 50 (correspondiente a 2500 Hz). Este análisis se adjunta en la Figura 6.5. Una peculiaridad del control de tipo FCS-MPC implementado es que aunque la frecuencia de muestreo sea de 40 kHz, la frecuencia a la que conmutan realmente los dispositivos de potencia no es igual a ésta, si no que es una frecuencia variable que oscila entre 5 y 7 veces menor que la frecuencia de muestreo. Esto se debe simplemente a que según el cálculo de la función de coste, se puede elegir mantener el mismo estado varios intervalos de control consecutivos. Por ello, una comparación directa del THD con otras tipologías de control no es trivial debido al factor crucial que la frecuencia de conmutación representa para este parámetro. Aún así, para poder obtener una mejor idea de por donde van los resultados, se adjunta una gráfica en la Figura 6.6 del contenido armónico obtenido con un control de tipo PI con el mismo convertidor y mismos parámetros eléctricos, con una frecuencia de conmutación igual a 10 kHz. En este tipo de control, la frecuencia de conmutación es fija e igual a la de sampleo.

En cuanto, al régimen transitorio, por norma general el control FCS-MPC tiene una respuesta dinámica mucho más rápida que el PI. Aunque este último posee una mayor robustez ante cambios en la carga debido a que no la incluye en su formulación. Se puede, sin embargo, introducir este término en la formulación del PI, haciéndolo más parecido en cuanto a respuesta dinámica al control FCS-MPC, pero perdiendo la ventaja de la robustez del control. Estas diferencias se pueden apreciar en la Figura 6.7. Por otro lado, FCS-MPC presenta la ventaja adicional de poder definir múltiples objetivos de control de manera sencilla. Adicionalmente, se puede observar como ante un cambio que sólo se produce en la referencia de corriente en su componente d, el control FCS-MPC no ve prácticamente afectada la regulación de la componente q, mientras que en el control de tipo PI existe un mayor acoplo entre ambas.

Por otro lado, también se obtiene en este experimento el tiempo de cálculo del algoritmo de control para las dos alternativas de diseño para el algoritmo de control en FPGA propuestas. Esto se hace mediante un pin digital que se pone a '1'cuando el algoritmo comienza sus cálculos y se devuelve a '0'cuando finaliza. Se adjunta una captura del osciloscipio con el que se muestrea esta señal digital en la Figura 6.8. Como se puede apreciar, el tiempo de cálculo se reduce notablemente en la implementación en paralelo frente a la implementación secuencial.

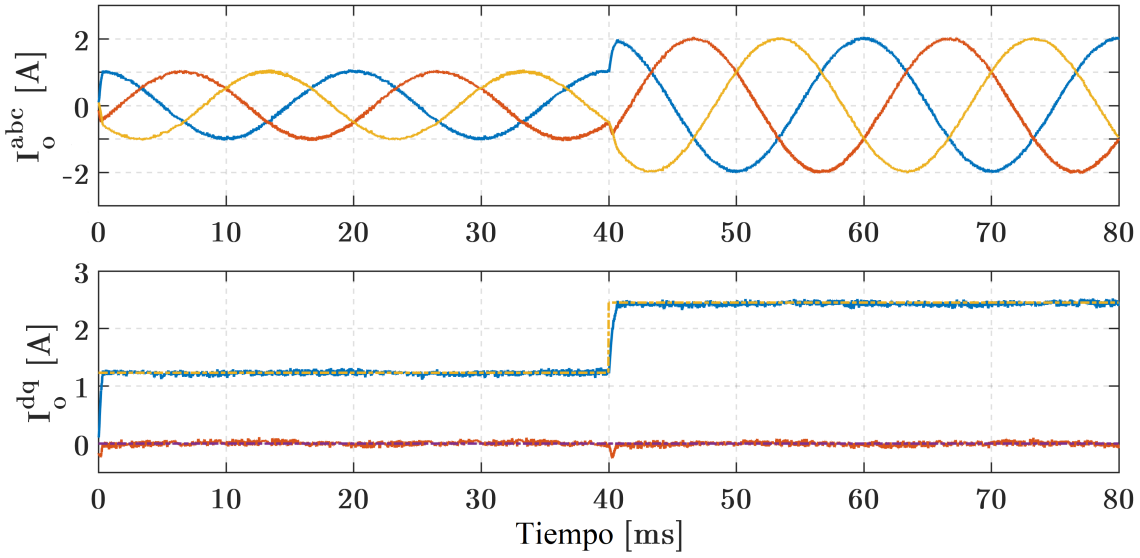

Figura 6.3 Resultados experimentales para la implementación en paralelo.

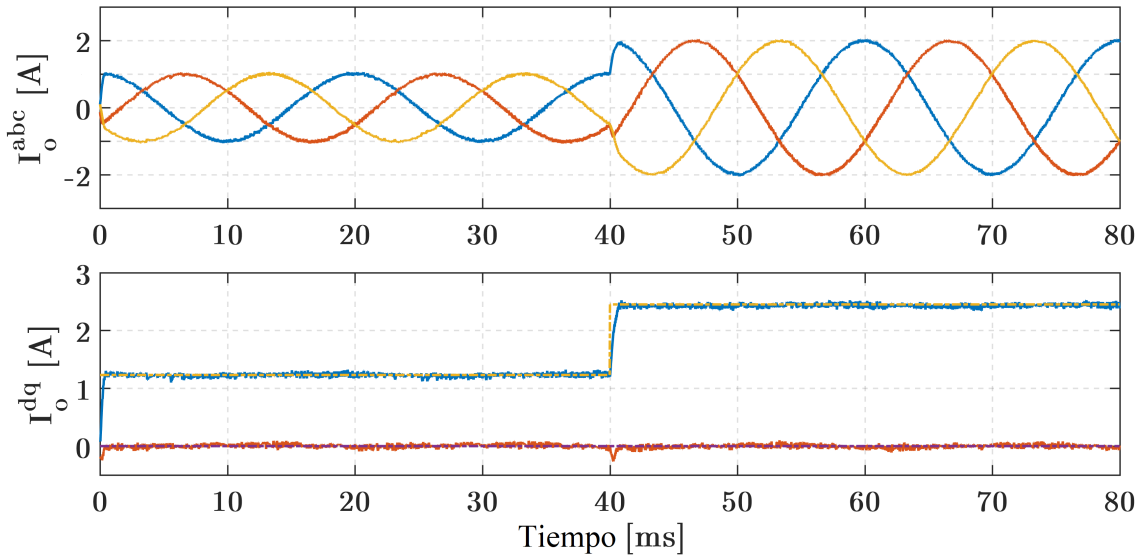

Figura 6.4 Resultados experimentales para la implementación secuencial.

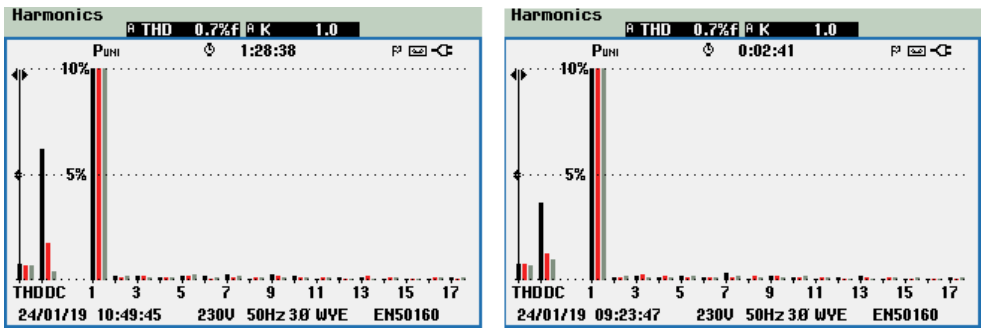

Figura 6.5 Análisis espectral para implementación parelela (izquierda) y secuencial (derecha). $THD = 0.7\,\%$ en ambos casos.

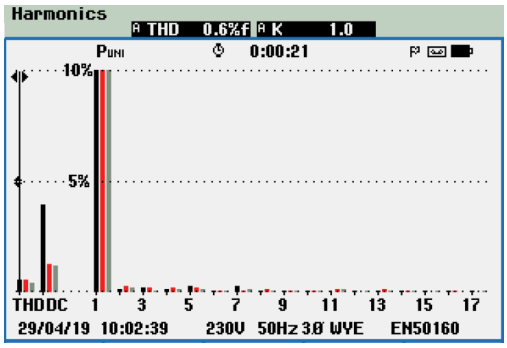

Figura 6.6 Análisis espectral para un control tipo PI conmutando a 10 kHz. $THD = 0.6\,\%$.

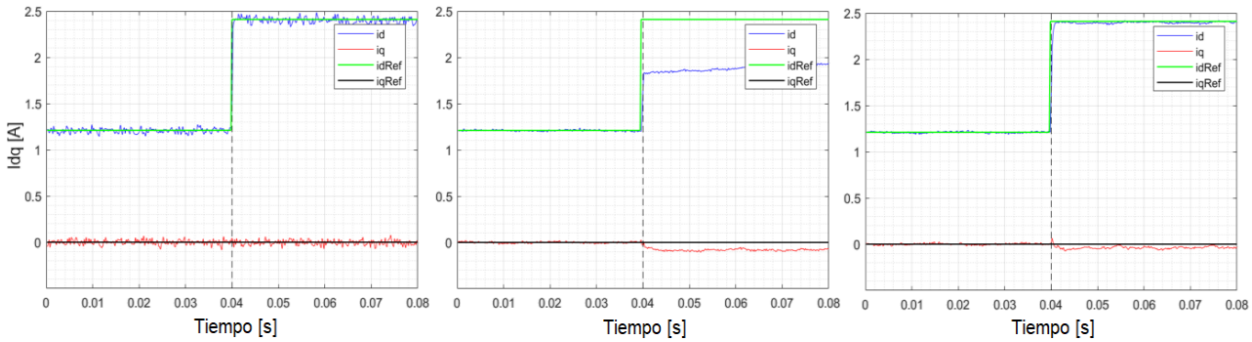

Figura 6.7 Comparación en régimen transitorio entre tipologías de control para la regulación de una referencia de corriente en ejes *dq* en un inversor trifásico de dos niveles. A la izquierda el control FCS-MPC. En el centro control PI sin término *RL* y a la derecha control PI con término *RL*.

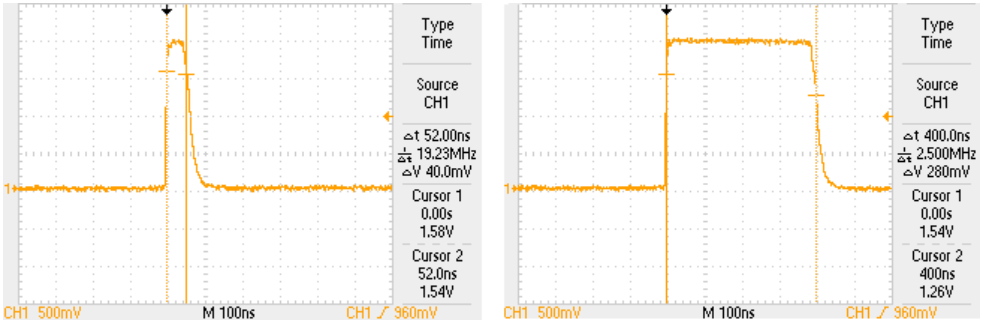

Figura 6.8 Tiempo de cálculo para la implementación paralela (izquierda) y secuencial (derecha): 50 *ns* para el diseño paralelo y 400 *ns* para el secuencial.

6.2 Análisis de consumo de recursos y tiempo de cálculo

Por último, este apartado se dedicará a llevar a cabo un análisis final de las ventajas e inconvenientes que presentan las dos alternativas de diseño propuestas a la hora de ser implementadas en una FPGA. En particular, se centrará el contenido de este apartado en analizar cuánto ocupan en área cada diseño y cuánto consumen de recursos en la FPGA. Para ello, se hará referencia a los conceptos introducidos en el tercer capítulo de este Trabajo Fin de Máster, en lo referente a los distintos recursos presentes en la FPGA Artix-7 que incluye la Zynq-7000. La disponibilidad de distintos tipos de recursos para afrontar las diferentes operaciones en una implementación, da al programador cierta flexibilidad para decidir en qué proporción utilizar estos recursos según las necesidades de espacio y velocidad. Esto se puede hacer manualmente, desde el propio diseño VHDL, mediante una codificación correcta del mismo que ayude a la herramienta a inferir el tipo de recursos que debe. También mediante el uso de atributos que fuercen a la herramienta de síntesis a ello. Adicionalmente, el software Vivado permite retocar ciertos parámetros del proceso de síntesis en su pestaña de "Settings".

En una primera instancia, se ha optado por no modificar ninguno de estos parámetros y utilizar las opciones por defecto de síntesis que ofrece Vivado. De esta manera, una vez efectuada la síntesis y la implementación de las dos alternativas de diseño propuestas, se puede acceder al reporte de utilización jerárquico de la implementación, que da una lista del consumo de recursos de cada bloque. El reporte del diseño paralelo se encuentra en la Tabla 6.2 y el del secuencial en la Tabla 6.3. Para el bloque de predicción en $k+2$, que es la gran diferencia entre ambos, este bloque incluye el consumo de recursos de los bloques auxiliares que han sido necesarios definir en cada caso. Por ejemplo, en el caso del diseño paralelo hubo que codificar un bloque comparador extra de ocho entradas que no está en el caso del secuencial, por lo que su consumo se computa dentro del apartado predicción a $k+2$. En el caso del secuencial, ocurre lo propio con la máquina de estados auxiliar que hubo que definir para recorrer los distintos estados. Bajo el término "Miscelánea" se incluyen el resto de bloques secundarios no incluidos bajo el resto de apartados principales, como son los bloques de detección de flanco o de rebotes. También es necesario señalar que dentro del apartado "ARM wrapper" se incluye todo el diagrama de bloques, incluyendo el periférico creado. Parte del consumo de este bloque se debe a diseños propietarios de Xilinx que se generan automáticamente al instanciar ciertos bloques como el del bloque procesador. En las tablas, el tipo de recurso "Registros" se refiere a biestables de tipo "Flip-Flop", inferidos en los procesos síncronos.

Tabla 6.2 Consumo de recursos jerárquico para diseño paralelo.

Bloque	LUT	Registros	BRAM	Bloques DSP
"ARM wrapper"	2032	3390	0	0
Contador	90	48	0	0
Transformación dq	321	161	0	14
BRAM (sin,cos)	5	1	2	0
Predicción $k+1$	70	33	0	26
Predicción $k+2$	2618	257	0	160
FSM	6	6	0	0
Disparos	2	9	0	0
Miscelánea	14	53	0	0
Total	5158	3958	2	200
(%)	9.70%	3.70%	1.43%	90.91%

Tabla 6.3 Consumo de recursos jerárquico para diseño secuencial.

Bloque	LUT	Registros	BRAM	Bloques DSP
"ARM wrapper"	2035	3390	0	0
Contador	88	48	0	0
Transformación dq	161	161	0	16
BRAM (sin,cos)	5	1	2	0
Predicción $k+1$	70	33	0	26
Predicción $k+2$	181	45	0	28
FSM	6	6	0	0
Disparos	2	9	0	0
Miscelánea	14	53	0	0
Total	2562	3746	2	70
(%)	4.82%	3.51%	1.43%	31.82%

Como era de esperar, la implementación secuencial se caracteriza por un consumo de recursos en la FPGA bastante menor. Esto es a cambio de aumentar el tiempo de cálculo requerido, tal y como se pudo observar en los resultados experimentales (Figura 6.8). Esta reducción de recursos permite poder implementar el diseño tal y como esta planteado en una FPGA de menor tamaño y coste más reducido. Adicionalmente, existe más espacio disponible para aumentar la complejidad computacional del diseño, ya sea mediante el control de un sistema con un mayor número de estados o un mayor número de horizontes de predicción. Por otro lado, el mayor consumo de área en el diseño paralelo se traduce también en un aumento de la potencia consumida por el circuito estimada por Vivado y de la temperatura del chip, tal y como se aprecia en la Tabla 6.6.

Otro hecho interesante que puede apreciarse en los resultados de implementación es el alto consumo de bloques DSP, de los que hay 220 en la FPGA. Esto se debe a que por defecto, el sintetizador infiere estos bloques para realizar operaciones de multiplicación, multiplicación+adición/resta o multiplicación+acumulación [51]. Este tipo de operaciones aritméticas está muy presente en el diseño planteado, tanto para las transformadas como para el propio algoritmo de control, tal y como se pudo ver en el quinto capítulo de este proyecto. Por tanto, es normal que la herramienta de síntesis esté utilizando estos recursos de manera preferente. Lo que ocurre es que según el uso de estos recursos DSP se acerca a su límite, el sintetizador empieza a utilizar recursos de tipo LUT para satisfacer algunas de las operaciones planteadas. Esto hace que la comparación entre ambos diseños no sea del todo directa, al no haber una métrica clara de a cuantos bloques LUT equivale un bloque DSP.

Por este motivo, y porque no todas las FPGAs del mercado cuentan con recursos de tipo DSP, se va a proponer una nueva implementación, para la cual se fuerza al sintetizador a sólo inferir bloques LUT para realizar todas las operaciones. Además, se fuerza la no utilización de bloques DSP añadiendo el atributo "USE_DSP", de manera similar a como se hizo para forzar la síntesis de un bloque RAM, pero esta vez dando el valor "NO" al atributo para que no se infieran bloques DSP [51]. Forzando la no utilización de bloques DSP, se puede obtener una medida del consumo de área para ambos diseños que es más fácil de comparar y de extrapolar a otras FPGAs del mercado. Los resultados de esta segunda implementación se pueden encontrar en la Tabla 6.4 y en la Tabla 6.5. Para el proceso de síntesis, en este caso, se ha aplicado el preset de "High Area Optimization" en las "Settings" de Vivado, para incentivar el ahorro de área en la síntesis.

Tabla 6.4 Consumo de recursos jerárquico para diseño paralelo sin bloques DSP.

Bloque	LUT	Registros	BRAM	Bloques DSP
"ARM wrapper"	1994	3510	0	0
Contador	83	48	0	0
Transformación dq	3099	161	0	0
BRAM (sin,cos)	5	1	2	0
Predicción $k+1$	5747	33	0	0
Predicción $k+2$	35859	257	0	0
FSM	5	6	0	0
Disparos	2	9	0	0
Miscelánea	11	53	0	0
Total	46805	4078	2	0
(%)	87.95%	3.82%	1.43%	0%

Tabla 6.5 Consumo de recursos jerárquico para diseño secuencial sin bloques DSP.

Bloque	LUT	Registros	BRAM	Bloques DSP
"ARM wrapper"	2000	3510	0	0
Contador	87	48	0	0
Transformación dq	3100	161	0	0
BRAM (sin,cos)	5	1	2	0
Predicción $k+1$	5740	33	0	0
Predicción $k+2$	6309	76	0	0
FSM	5	6	0	0
Disparos	2	9	0	0
Miscelánea	14	53	0	0
Total	17262	3897	2	0
(%)	32.45%	3.65%	1.43%	0%

Finalmente, para poner en perspectiva el valor de los resultados conseguidos en esta plataforma, en lo que respecta a los tiempos computacionales alcanzados, el mismo algoritmo que se ha implementado en la FPGA se ha implementado en una plataforma DSP, cuyo uso en este tipo de aplicaciones está mucho más extendido. En concreto, se ha utilizado el DSP TMS320F28335 de la familia C2000 Delfino de Texas

Instruments. Este DSP se trata de una plataforma muy común, utilizada para implementar el control de múltiples equipos de potencia en el Grupo de Tecnología Electrónica, y de la que también se puede encontrar mucha documentación y ejemplos orientados a este tipo de aplicaciones de control.

De esta manera, se ha codificado una aplicación en lenguaje C que se ejecuta en esta plataforma con una velocidad de reloj de 150 MHz. Compilando la aplicación con las directivas de optimización de velocidad de cálculo al máximo en las opciones del compilador, se pudo alcanzar un tiempo de cálculo del algoritmo de 12 μs. Esto supone ser una 30 veces más lento que el diseño secuencial propuesto para la FPGA y 240 veces para el caso del diseño en paralelo. Esta información se recoge en la Tabla 6.6 junto con las estimaciones de consumo y temperaturas para los diseños en la Zynq-7000 estimados por Vivado.

Tabla 6.6 Tiempos de cálculo para los métodos propuestos.

Magnitud	Paralelo	Secuencial	TMS320F28335
Tiempo de computación [ns]	50	400	12000
Consumo estimado [W]	1.713	1.704	N/A
Temperatura estimada [ºC]	44.8	44.7	N/A

Con estos datos como referencia, se puede afirmar que con una plataforma convencional de tipo DSP como es el TMS320F28335 la tasa de sampleo alcanzable está mucho más limitada por su capacidad de cálculo. Sólo con el tiempo que tarda en calcularse el algoritmo, una frecuencia de muestreo de 100 kHz, que podría ser una frecuencia de conmutación muy común para los dispositivos de tipo "Wide Bandgap", ya no es alcanzable. Por otro lado, la implementación FPGA propuesta podría llegar sobradamente a esa frecuencia. Todo esto es para un diseño hardware que utiliza una señal de reloj de 100 MHz, por lo que si se utilizará un reloj de 150 MHz como en el DSP, los resultados serían aún mejorables en el caso de la FPGA, sin alterar la arquitectura propuesta.

7 Conclusiones y Trabajo Futuro

En este Trabajo Fin de Máster, se ha detallado el diseño de una arquitectura HW/SW para una plataforma FPSoC moderna como es el AP-SoC Zynq-7000 de Xilinx. En concreto, el objetivo de este proyecto era demostrar la viabilidad y las ventajas que una plataforma como ésta puede conllevar en su utilización para aplicaciones de electrónica de potencia. Para ello, se ha formulado un control de tipo FCS-MPC para la regulación de las corrientes generadas por un inversor trifásico de dos niveles.

El primer paso que se ha seguido en este documento ha sido precisamente formular el problema desde el punto de vista matemático, describiendo las ecuaciones que componen el algoritmo de control y que describen el modelo del sistema. Se han detallado algunas de las ventajas que posee este tipo de control y se han referenciado algunos trabajos de investigación recientes que demuestran el interés investigador existente en la actualidad por el control predictivo en el campo de la electrónica de potencia.

Posteriormente, se ha procedido a describir todos y cada uno de los componentes de la plataforma de control. Se han detallado las características de la placa Zedboard y el chip Zynq-7000 que implementa y se ha hecho lo propio para el ADC externo utilizado y para las placas auxiliares que se han requerido para interconectar el hardware.

Descrito el hardware a utilizar, y tras un breve repaso de las herramientas software que ofrece Xilinx para programar sus dispositivos y el establecimiento de ciertos conceptos que se utilizan de forma recursiva a lo largo del documento, se ha podido profundizar en la arquitectura propuesta. En particular, se ha expuesto un codiseño HW/SW donde la FPGA y los núcleos ARM que incluye el Zynq-7000 trabajan en conjunción repartiéndose el peso de las tareas a realizar por la plataforma y compartiendo información a través de la memoria compartida. Tras establecer la arquitectura básica, se ha podido abordar la programación de cada uno de los elementos que componen el sistema.

Desde el punto de vista de la FPGA, se ha abordado todo el diseño hardware, con los bloques que lo componen y la codificación en lenguaje VHDL de los mismos. En concreto, se han descrito alguna de las dificultades y particularidades que se pueden encontrar a la hora de trabajar con este tipo de dispositivos, en contraposición a un sistema microprocesador, y se han expuesto las metodologías con las que se han resuelto. Esto incluye dos alternativas de diseño que se han propuesto para la implementación del algoritmo cuyas ventajas y desventajas se han comparado y analizado.

En cuanto a los dos núcleos procesadores que incluye el FPSoC, se ha descrito la compilación y puesta en marcha del SO Linux que ejecutará uno de ellos. Se ha introducido el concepto de "Asymmetric Multiprocessing" (AMP) y se han establecido las guías para que el sistema pueda ejecutar, basándose en esta configuración, una aplicación baremetal totalmente independiente en el segundo núcleo con la que puede compartir información. Así mismo se ha descrito la programación de sendas aplicaciones a ser ejecutadas por ambos procesadores. Para la monitorización del sistema, se ha establecido una conexión TCP/IP entre la tarjeta Zedboard y un PC que ejecuta una interfaz gráfica diseñada en Java.

De esta manera, se ha podido desarrollar y establecer la base de un sistema completo de control y monitorización de un convertidor de potencia. Su funcionamiento y viabilidad han sido contrastados experimentalmente en laboratorio y los resultados se han expuesto en este documento. Esto ha dado lugar a diversos análisis y comparaciones de bastante interés. Por un lado, se ha podido comprobar el funcionamiento del control FCS-MPC implementado en este proyecto y comparar los resultados obtenidos en el mismo convertidor de potencia con un control de tipo PI. Se ha podido comparar el comportamiento de ambos tanto en régimen permanente como en régimen transitorio, pudiendo establecer qué características diferencian ambos tipos de control con una base experimental.

Por otro lado, se han comparado a nivel de tiempo de computación y de consumo de recursos los dos diseños propuestos para la implementación FPGA. Se ha podido comprobar como la metodología en paralelo permite alcanzar los mejores tiempos de computación, pero a costa de un consumo de recursos mayor. De esta manera, este diseño podría ser el más apropiado en FPGAs de gran tamaño donde el objetivo es alcanzar la mayor velocidad posible. En el caso de que las restricciones de área aumenten, ya sea porque se utiliza una FPGA de menor coste o porque el tamaño del modelo o el número de horizontes de predicción aumentan más allá de las capacidades de área de la misma, es posible que el diseño secuencial se convierta en una solución más apropiada. La implementación en secuencial permite acudir a FPGAs de menor tamaño y coste, manteniendo un tiempo de cálculo reducido cuando se compara con los resultados obtenidos con una solución convencional basada en plataformas DSP.

A la luz de todos los resultados obtenidos, se puede afirmar que migrar a plataformas de control basadas en dispositivos FPSoC para aplicaciones de electrónica de potencia, y en concreto, la arquitectura propuesta en este Trabajo Fin de Máster pueden resultar de mucho interés para el ámbito académico e industrial. Especialmente, con la adopción de algoritmos de control cada vez más complejos y la llegada de los nuevos dispositivos semiconductores "Wide Bandgap" que permiten alcanzar frecuencias de conmutación mucho más altas que los tradicionales conteniendo las pérdidas energéticas. De esta forma, toda la complejidad de cálculo que estas tendencias suponen, puede ser abordada por las nuevas plataformas FPSoC mientras que con las plataformas más convencionales no.

En definitiva, el trabajo desarrollado para este Trabajo Fin de Máster ha permitido establecer un marco base para la utilización de una plataforma nueva cuya viabilidad y rendimiento han sido contrastados experimentalmente. Esto abre la puerta al desarrollo de una gran cantidad de trabajos futuros, ya que la puesta en marcha desde cero de una plataforma es siempre el paso más complejo. Una vez establecida esta arquitectura base, y gracias a la flexibilidad inherente de la plataforma, las posibilidades de ampliación son prácticamente ilimitadas. Algunas que se pueden citar:

- **Afinar el "trade-off" área-velocidad**. En este proyecto se han desarrollado dos implementaciones en lo que refiere al algoritmo de control totalmente contrapuestas. Es de esperar que al aumentar el tamaño del algoritmo, porque aumente el tamaño del sistema, o porque se incremente el horizonte de predicción, las diferencias entre ambos aumenten. Esto quiere decir que puede ser conveniente como trabajo futuro explorar soluciones intermedias que combinen la implementación de parte de las predicciones en paralelo y otra parte en secuencial, de manera que se puedan obtener diseños que permitan optimizar a la vez los parámetros de consumo de recursos y velocidad.

- **ADC externo**. El ADC utilizado en este proyecto permite una tasa de sampleo de como mucho 1 MSPS. En la práctica, tal y como se pudo exponer en el capítulo quinto de este proyecto, esta tasa ni siquiera se alcanza debido a particularidades del periférico SPI de la Zynq-7000. Midiendo los tiempos de lectura/escritura del ADC se obtiene que para cada medida realizada, se están gastando cerca de 2 microsegundos, estando la tasa de sampleo real en torno a los 500 kSPS. Esto se traduce en que el ADC es ahora mismo el verdadero cuelo de botella del sistema y uno de los puntos claros de mejora. Posibles soluciones pasan por explorar otros ADCs externos en el mercado que permitan tasas más altas de sampleo o pasar a otra plataforma, como la Pynq Z1. Ésta se basa en el mismo integrado exacto que la Zedboard, pero tiene accesibles un mayor número de pines del ADC, por lo que se podría trasladar la arquitectura propuesta de una a otra, simplemente pasando a realizar las medidas con el ADC de la Zynq-7000, que tiene una tasa de sampleo de 1 MSPs.

- **Prueba en otros equipos**. En la actualidad, se está trabajando en la puesta en marcha en el laboratorio de una UPS para su control mediante FCS-MPC. Este equipo resulta básicamente de colocar un filtro LC a la salida del inversor, lo que se traduce en un aumento del tamaño del modelo. En particular, con este sistema se llevó un artículo de congreso al CPE-POWERENG 2019 que estaba centrado en aspectos del control y del diseño del observador para las corrientes de salida, pero que sólo contenía resultados de simulación. La idea es validar estos resultados experimentales, por lo que se está trabajando actualmente en su puesta en marcha con un control en una plataforma DSP. Un paso interesante podría ser controlar este equipo con la Zedboard.

- **Montar equipo "Wide Bandgap"**. Podría resultar de mucho interés poder contrastar la arquitectura propuesta en un convertidor de potencia basado en esta nueva tecnología de semiconductores. Para ello, sería necesario montar el equipo, que no se dispone en estos momentos en el laboratorio.

Índice de Figuras

1.1 Convertidor DC/AC [10] 1
1.2 Esquema básico de control y modulación de un convertidor DC/AC [12] 2
1.3 Bandas de un material semiconductor [18] 3

2.1 Circuito de un inversor trifásico de dos niveles [39] 6
2.2 Circuito de un inversor trifásico de dos niveles con carga RL. Los valores de tensión se miden respecto al neutro (N) de la carga 6

3.1 Kit de desarrollo Zedboard, del fabricante Avnet [35] 11
3.2 Diagrama de bloques de SoC Zynq-7000 12
3.3 Características Zynq-7000 (PS). El integrado correspondiente a la Zedboard es el Z-7020 14
3.4 Características Zynq-7000 (PL). El integrado correspondiente a la Zedboard es el Z-7020 16
3.5 Ejemplo de un CLB básico en una FPGA [48] 16
3.6 CLB en FPGA serie 7 de Xilinx [49] 17
3.7 "SLICEL" de la serie 7 de FPGAs de Xilinx [49] 18
3.8 "SLICEM" de la serie 7 de FPGAs de Xilinx [49] 19
3.9 Bloque "DSP48E1" de la serie 7 de FPGAs de Xilinx [50] 19
3.10 Kit de evaluación Zedboard [53] 20
3.11 Diagrama de bloques de componentes en placa de desarrollo Zedboard [35] 22
3.12 Distribución de pines del integrado Zynq-7000 en bancos [35] 23

4.1 Convertidor trifásico de dos niveles en laboratorio 26
4.2 Esquema sensor LV 25-P [56] 26
4.3 Esquema sensor LA 55-NP [57] 27
4.4 Placa de adaptación de medidas 27
4.5 Esquemático circuito adaptación 28
4.6 Diagrama de bloques funcional del ADS 7953 [58] 29
4.7 Diagrama temporal del proceso de conversión [58] 30
4.8 Diagrama temporal detallado de la interfaz digital [58] 30
4.9 Diagrama temporal detallado de la secuencia de canales en modo manual [58] 31
4.10 Kit de desarrollo ADS7953EVM-PDK con placa de interconexión para Zedboard y etapa de adaptación 32
4.11 Señal senoidal de 50 Hz muestreada con el ADC 34

5.1 Creación de proyecto en Vivado. Elección de placa 36
5.2 Ventana principal de Vivado (Versión 2018.1) 36
5.3 "Block Design" con bloque "ZYNQ7 Processing System" 38
5.4 Ventana principal del Xilinx SDK 39
5.5 Configuraciones por defecto del "Board Support Package" 40
5.6 Creación de un proyecto de aplicación 41
5.7 Diagrama de flujos con la división de tareas propuesta 42
5.8 Archivos en la raíz de la tarjeta SD 43
5.9 Ventana de creación del BOOT.bin 44

5.10 Archivos dts para creación del Device Tree 45
5.11 Interacción desde terminal con el SO ejecutándose en la Zedboard 49
5.12 Diagrama de bloques de la arquitectura propuesta 49
5.13 Rango de direcciones asignado al periférico creado 51
5.14 Diseño de bloques para la parte PS 53
5.15 Diagrama de la máquina de estados 54
5.16 Señal de error generada en el driver del inversor 54
5.17 Diagrama de bloques de la transformada a ejes dq 60
5.18 Diagrama de bloques de la implementación en paralelo de la predicción a $k+2$ 62
5.19 Diagrama de bloques de la implementación secuencial de la predicción a $k+2$ 62
5.20 Interfaz de usuario diseñada en Java 69
5.21 Interfaz de usuario en funcionamiento 70

6.1 Zedboard con placas de expansión fabricadas en el laboratorio 71
6.2 Carga RL trifásica del laboratorio 72
6.3 Resultados experimentales para la implementación en paralelo 74
6.4 Resultados experimentales para la implementación secuencial 74
6.5 Análisis espectral para implementación parelela (izquierda) y secuencial (derecha). $THD = 0.7\,\%$ en ambos casos 74
6.6 Análisis espectral para un control tipo PI conmutando a 10 kHz. $THD = 0.6\,\%$ 75
6.7 Comparación en régimen transitorio entre tipologías de control para la regulación de una referencia de corriente en ejes dq en un inversor trifásico de dos niveles. A la izquierda el control FCS-MPC. En el centro control PI sin término RL y a la derecha control PI con término RL 75
6.8 Tiempo de cálculo para la implementación paralela (izquierda) y secuencial (derecha): $50\ ns$ para el diseño paralelo y $400\ ns$ para el secuencial 75

Índice de Tablas

2.1	Variables y parámetros del sistema	7
2.2	Valores de tensión de salida del convertidor en función del estado	7
4.1	Estructura de la trama de entrada SDI para el modo manual [58]	31
4.2	Configuración jumpers placa ADS7953EVM-PDK [59]	32
4.3	Pines Alimentación de la placa ADS7953EVM-PDK [59]	33
4.4	Pinout interfaz Analógica de la placa ADS7953EVM-PDK[59]	33
4.5	Pinout interfaz Digital de la placa ADS7953EVM-PDK [59]	33
5.1	Configuración de jumpers en Zedboard. El resto de jumpers quedan al aire	48
6.1	Variables y parámetros del sistema	73
6.2	Consumo de recursos jerárquico para diseño paralelo	76
6.3	Consumo de recursos jerárquico para diseño secuencial	76
6.4	Consumo de recursos jerárquico para diseño paralelo sin bloques DSP	77
6.5	Consumo de recursos jerárquico para diseño secuencial sin bloques DSP	77
6.6	Tiempos de cálculo para los métodos propuestos	78

Índice de Códigos

2.1	Discretización de un modelo de espacio de estados en MATLAB	8
5.1	Archivo BIF	43
5.2	Compilación DeviceTree	45
5.3	Eliminar esta sección del archivo pcw.dtsi	45
5.4	Archivo system-top.dtsi	46
5.5	Eliminar esta sección del archivo zynq-7000.dtsi	46
5.6	Sección "slcr" del archivo zynq-7000.dtsi	47
5.7	Código de ini1.sh	47
5.8	Atributos para bloque RAM	59
5.9	Archivo de constraints	63

Bibliografía

[1] J. M. Carrasco, L. G. Franquelo, J. T. Bialasiewicz, E. Galvan, R. C. PortilloGuisado, M. A. M. Prats, J. I. Leon, and N. Moreno-Alfonso, "Power-electronic systems for the grid integration of renewable energy sources- a survey," *IEEE Transactions on Industrial Electronics*, vol. 53, no. 4, pp. 1002–1016, June 2006.

[2] S. Ou and H. Hsiao, "Analysis and design of a novel single-stage switching power supply with half-bridge topology," *IEEE Transactions on Power Electronics*, vol. 26, no. 11, pp. 3230–3241, Nov 2011.

[3] M. P. Kazmierkowski, L. G. Franquelo, J. Rodriguez, M. A. Perez, and J. I. Leon, "High-performance motor drives," *IEEE Industrial Electronics Magazine*, vol. 5, no. 3, pp. 6–26, Sep. 2011.

[4] G. Vivek and J. Biswas, "Study on hybrid SVPWM sequences for two level VSIs," in *2017 IEEE International Conference on Industrial Technology (ICIT)*, March 2017, pp. 219–224.

[5] R. K. Varma, W. Litzenberger, A. Ostadi, and S. Auddy, "Bibliography of facts: 2005-2006 part i ieee working group report," in *2007 IEEE Power Engineering Society General Meeting*, June 2007, pp. 1–8.

[6] K. W. E. Cheng, "Recent development on electric vehicles," in *2009 3rd International Conference on Power Electronics Systems and Applications (PESA)*, May 2009, pp. 1–5.

[7] G. Wang, G. Konstantinou, C. D. Townsend, J. Pou, S. Vazquez, G. D. Demetriades, and V. G. Agelidis, "A review of power electronics for grid connection of utility-scale battery energy storage systems," *IEEE Transactions on Sustainable Energy*, vol. 7, no. 4, pp. 1778–1790, Oct 2016.

[8] K. Rahbar, C. C. Chai, and R. Zhang, "Energy cooperation optimization in microgrids with renewable energy integration," *IEEE Transactions on Smart Grid*, vol. 9, no. 2, pp. 1482–1493, March 2018.

[9] J. Zhang, "Power electronics in future electrical power grids," in *2013 4th IEEE International Symposium on Power Electronics for Distributed Generation Systems (PEDG)*, July 2013, pp. 1–3.

[10] L. G. Franquelo and J. I. Leon, *Lecture: Multilevel Converters Topologies for Industrial Applications*. Universidad de Sevilla, 2017.

[11] J. I. Leon, S. Kouro, L. G. Franquelo, J. Rodriguez, and B. Wu, "The essential role and the continuous evolution of modulation techniques for voltage-source inverters in the past, present, and future power electronics," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 5, pp. 2688–2701, May 2016.

[12] L. G. Franquelo and J. I. Leon, *Lecture: Modulation Techniques for Multilevel Converters*. Universidad de Sevilla, 2017.

[13] S. Vazquez, J. Rodriguez, M. Rivera, L. G. Franquelo, and M. Norambuena, "Model predictive control for power converters and drives: Advances and trends," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 2, pp. 935–947, Feb 2017.

[14] P. Cortes, G. Ortiz, J. I. Yuz, J. Rodriguez, S. Vazquez, and L. G. Franquelo, "Model predictive control of an inverter with output *lc* filter for ups applications," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 6, pp. 1875–1883, June 2009.

[15] N. He, M. Chen, J. Wu, N. Zhu, and D. X. Ge, "20 kw zero-voltage-switching sic-mosfet grid inverter with 300 khz switching frequecny," *IEEE Transactions on Power Electronics*, pp. 1–1, 2018.

[16] J. Millán, P. Godignon, X. Perpiñà, A. Pérez-Tomás, and J. Rebollo, "A survey of wide bandgap power semiconductor devices," *IEEE Transactions on Power Electronics*, vol. 29, no. 5, pp. 2155–2163, May 2014.

[17] A. S. Abdelrahman, Z. Erdem, Y. Attia, and M. Z. Youssef, "Wide bandgap devices in electric vehicle converters: A performance survey," *Canadian Journal of Electrical and Computer Engineering*, vol. 41, no. 1, pp. 45–54, winter 2018.

[18] Wikipedia - Estructura de bandas de un semiconductor. https://es.wikipedia.org/wiki/Banda_prohibida#/media/File:Semiconductor_band_structure_(lots_of_bands_2).svg. [Online - Acceso Marzo 2019].

[19] W. Qian, X. Zhang, F. Jin, H. Bai, D. Lu, and B. Cheng, "Using high-control-bandwidth fpga and sic inverters to enhance high-frequency injection sensorless control in interior permanent magnet synchronous machine," *IEEE Access*, vol. 6, pp. 42 454–42 466, 2018.

[20] Infineon - Wide Bandgap Semiconductors (SiC/GaN). https://www.infineon.com/cms/en/product/power/wide-band-gap-semiconductors-sic-gan/. [Online - Acceso Marzo 2019].

[21] E. Monmasson and M. N. Cirstea, "Fpga design methodology for industrial control systems a review," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1824–1842, Aug 2007.

[22] E. Monmasson, L. Idkhajine, and M. W. Naouar, "Fpga-based controllers," *IEEE Industrial Electronics Magazine*, vol. 5, no. 1, pp. 14–26, March 2011.

[23] M. Naouar, E. Monmasson, A. A. Naassani, I. Slama-Belkhodja, and N. Patin, "Fpga-based current controllers for ac machine drives: A review," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1907–1925, Aug 2007.

[24] O. Gulbudak and E. Santi, "Fpga-based model predictive controller for direct matrix converter," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 7, pp. 4560–4570, July 2016.

[25] E. T. Mekonnen, J. Katcha, and M. Parker, "An fpga-based digital control development method for power electronics," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, Oct 2012, pp. 222–226.

[26] D. Navarro, O. Lucia, L. A. Barragan, I. Urriza, and O. Jiménez, "High-level synthesis for accelerating the fpga implementation of computationally demanding control algorithms for power converters," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1371–1379, Aug 2013.

[27] R. Velazquez, O. Lucia, D. Navarro, L. A. Barragan, J. I. Artigas, and C. Sagues, "Design of an fpga-based full-state feedback controller using high level synthesis tools," in *2014 IEEE 15th Workshop on Control and Modeling for Power Electronics (COMPEL)*, June 2014, pp. 1–6.

[28] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," *IEEE Design Test of Computers*, vol. 26, no. 4, pp. 18–25, July 2009.

[29] A. Stanciu and C. Gerigan, "Comparison between implementations efficiency of hls and hdl using operations over galois fields," in *2017 IEEE 23rd International Symposium for Design and Technology in Electronic Packaging (SIITME)*, Oct 2017, pp. 171–174.

[30] M. Pelcat, C. Bourrasset, L. Maggiani, and F. Berry, "Design productivity of a high level synthesis compiler versus hdl," in *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, July 2016, pp. 140–147.

[31] F. M. Sánchez, R. Mateos, E. J. Bueno, J. Mingo, and I. Sanz, "Comparative of hls and hdl implementations of a grid synchronization algorithm," in *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, Nov 2013, pp. 2232–2237.

[32] R. F. Molanes, J. J. Rodriguez-Andina, and J. Farina, "Performance characterization and design guidelines for efficient processor-fpga communication in cyclone v fpsocs," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 5, pp. 4368–4377, May 2018.

[33] I. Bahri, L. Idkhajine, E. Monmasson, and M. E. A. Benkhelifa, "Hardware/software codesign guidelines for system on chip fpga-based sensorless ac drive applications," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2165–2176, Nov 2013.

[34] Zynq-7000 SoC Data Sheet: Overview, howpublished = "https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf", note = "[online - acceso marzo 2019]".

[35] Zedboard Hardware User's Guide. http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf. [Online - Acceso Marzo 2019].

[36] S. Vazquez, J. I. Leon, L. G. Franquelo, J. Rodriguez, H. A. Young, A. Marquez, and P. Zanchetta, "Model predictive control: A review of its applications in power electronics," *IEEE Industrial Electronics Magazine*, vol. 8, no. 1, pp. 16–31, March 2014.

[37] R. Mendez, D. Sbarbaro, and J. Espinoza, "High dynamic and static performance FCS-MPC strategy for static power converters," in *2016 IEEE Energy Conversion Congress and Exposition (ECCE)*, Sep. 2016, pp. 1–7.

[38] B. Stellato, T. Geyer, and P. J. Goulart, "High-speed finite control set model predictive control for power electronics," *IEEE Transactions on Power Electronics*, vol. 32, no. 5, pp. 4007–4020, May 2017.

[39] G. T. Electrónica, *Apuntes Electrónica de Potencia II: Técnicas de Modulación Básicas*. Universidad de Sevilla, 2011.

[40] M. Comanescu, "Influence of the discretization method on the integration accuracy of observers with continuous feedback," in *2011 IEEE International Symposium on Industrial Electronics*, June 2011, pp. 625–630.

[41] Mathworks - c2d. https://es.mathworks.com/help/control/ref/c2d.html. [Online - Acceso Marzo 2019].

[42] P. Cortes, J. Rodriguez, C. Silva, and A. Flores, "Delay compensation in model predictive current control of a three-phase inverter," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 2, pp. 1323–1325, Feb 2012.

[43] Hardwarebee - List of FPGA companies. http://hardwarebee.com/list-fpga-companies/. [Online - Acceso Abril 2019].

[44] Zynq-7000 all programmable SoC (product brief). https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf. [Online - Acceso Marzo 2019].

[45] AMBA Specification. https://www.arm.com/products/silicon-ip-system/embedded-system-design/amba-specifications. [Online - Acceso Marzo 2019].

[46] Zynq-7000 SoC: Technical reference manual. UG585. https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf. [Online - Acceso Abril 2019].

[47] MIO and EMIO Configuration for Zynq-7000. https://www.xilinx.com/video/soc/mio-emio-configuration-zynq-7000.html. [Online - Acceso Abril 2019].

[48] Wikipedia - Logic block. https://en.wikipedia.org/wiki/Logic_block. [Online - Acceso Abril 2019].

[49] 7 series FPGAs configurable logic block: User guide. UG474. https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf. [Online - Acceso Abril 2019].

[50] 7 series FPGAs dsp48e1: User guide. UG479. https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf. [Online - Acceso Abril 2019].

[51] Vivado design suite user guide: Synthesis. UG901. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug901-vivado-synthesis.pdf. [Online - Acceso Abril 2019].

[52] 7 series FPGAs memory resources: User guide. UG473. https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf. [Online - Acceso Abril 2019].

[53] Zedboard. http://zedboard.org/product/zedboard. [Online - Acceso Abril 2019].

[54] Zedboard Schematic. http://zedboard.org/sites/default/files/documentations/ZedBoard_RevD.2_Schematic_130516.pdf. [Online - Acceso Abril 2019].

[55] Zedboard master XDC file. https://github.com/Digilent/Zedboard/blob/master/Resources/XDC/zedboard_master.xdc. [Online - Acceso Abril 2019].

[56] Datasheet - Voltage transducer LV 25-P. http://www.farnell.com/datasheets/1866272.pdf. [Online - Acceso Abril 2019].

[57] Datasheet - Current transducer LA 55-P. http://www.farnell.com/datasheets/1449960.pdf. [Online - Acceso Abril 2019].

[58] ADS 7953 Datasheet. http://www.ti.com/lit/ds/symlink/ads7953.pdf. [Online - Acceso Mayo 2019].

[59] ADS 7953EVM-PDK Datasheet. http://www.ti.com/lit/ds/symlink/ads7953.pdf. [Online - Acceso Mayo 2019].

[60] Introduction to SPI Interface. https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html. [Online - Acceso Mayo 2019].

[61] Wikipedia - Serial peripheral interface. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface#Pros_and_cons. [Online - Acceso Mayo 2019].

[62] Datasheet SMP04 - CMOS Quad Sample and Hold amplifier. https://www.analog.com/media/en/technical-documentation/data-sheets/SMP04.pdf. [Online - Acceso Mayo 2019].

[63] Vivado design suite. https://www.xilinx.com/products/design-tools/vivado.html. [Online - Acceso Mayo 2019].

[64] Petalinux tools. https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html. [Online - Acceso Marzo 2019].

[65] Xilinx Wiki - Prebuilt Linux images. https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842316/Zynq+Releases. [Online - Acceso Mayo 2019].

[66] Simple AMP running linux and bare-metal system on both Zynq SoC processors. https://www.xilinx.com/support/documentation/application_notes/xapp1078-amp-linux-bare-metal.pdf. [Online - Acceso Marzo 2019].

[67] Xilinx Wiki. https://xilinx-wiki.atlassian.net/wiki/spaces/A/overview. [Online - Acceso Marzo 2019].

[68] Xilinx Wiki - U-boot images. https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842374/U-Boot+Images. [Online - Acceso Mayo 2019].

[69] A. Ruiz Martínez, "Sistema de medida remota de señales eléctricas en tiempo real." *Universidad de Sevilla. Grado en Ingeniería de Tecnologías de Telecomunicación.*, 2018.

[70] Xilinx Wiki - Build device tree blob. https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842279/Build+Device+Tree+Blob. [Online - Acceso Mayo 2019].

[71] MicroZed Chronicles - Creating a PL peripheral. https://forums.xilinx.com/t5/Xcell-Daily-Blog-Archived/The-Zynq-PS-PL-Part-One-Adam-Taylor-s-MicroZed-Chronicles-Part/ba-p/418935. [Online - Acceso Mayo 2019].

[72] Floating point package user's guide . https://reup.dmcs.pl/wiki/images/4/45/Float_ug.pdf. [Online - Acceso Mayo 2019].

[73] CORDIC v6.0 - LogiCORE IP Product guide. https://www.xilinx.com/support/documentation/ip_documentation/cordic/v6_0/pg105-cordic.pdf. [Online - Acceso Mayo 2019].