Trabajo Fin de Grado
Grado en Ingeniería Electrónica Robótica y Mecatrónica

Implementación de controladores para convertidores de potencia en plataformas hardware tipo SoC mediante la herramienta HDL Coder

Autor: Ángel Márquez Rodríguez

Tutores: Sergio Vázquez Pérez y Eduardo Zafra Ratia

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

Sevilla, 2023

POWERUS
Power Electronics and Sustainable Energy Group Universidad de Sevilla

Trabajo Fin de Grado
Grado en Ingeniería Electrónica Robótica y Mecatrónica

Implementación de controladores para convertidores de potencia en plataformas hardware tipo SoC mediante la herramienta HDL Coder

Autor:
Ángel Márquez Rodríguez

Tutores:
Sergio Vázquez Pérez y Eduardo Zafra Ratia
Profesor Titular y PDI Universidad de Sevilla

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

Sevilla, 2023

Trabajo Fin de Grado: Implementación de controladores para convertidores de potencia en plataformas hardware tipo SoC mediante la herramienta HDL Coder

Autor: Ángel Márquez Rodríguez
Tutores: Sergio Vázquez Pérez y Eduardo Zafra Ratia

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

Primero, agradecer a mis tutores Eduardo Zafra y Sergio Vázquez. Por darme la oportunidad de hacer este proyecto con ellos y por sus enseñanzas, indicaciones y correcciones a lo largo del mismo.

También a todas las personas presentes en el laboratorio de electrónica de potencia que me han prestado su ayuda y conocimientos, particularmente Abraham Márquez por sus explicaciones en el diseño de PCBs y a Emilia Pérez por dejarme la OPAL para hacer las pruebas.

Después agradecer a mis padres Lola y Jose María que me han dado todo su apoyo y los medios durante todas mis etapas educativas. Sin ellos no sería quién soy y no hubiera llegado hasta aquí.

Agradecer a mi novia Miriam todos los buenos momentos y los ánimos que me ha dado a lo largo de esta etapa universitaria. Agradecer también a mis amigos y compañeros del grado, con los que he compartido todos los momentos duros que han ido apareciendo en el estudio. Y agradecer por último a mis amigos de siempre que tanto me han ayudado a despejarme cuando lo necesitaba.

Resumen

Este trabajo estudia el uso del add-on HDL Coder de MATLAB/Simulink para la programación de controladores para convertidores de potencia implementados en FPSoC. Como caso de estudio, se programará un chip Zynq-7000 de Xilinx que incluye la plataforma ZedBoard. Los objetivos del trabajo serán probar controladores implementados mediante las técnicas propuestas, comparar su funcionamiento y eficiencia en cuanto a recursos usados frente a otro controlador similar programado directamente en VHDL, y elaborar unas directrices que sirvan de guía para usar HDL Coder de manera rápida y eficiente.

Abstract

This work examines the use of the HDL Coder add-on for MATLAB/Simulink to design controllers for power converters implemented in an FPSoC. In this case of study, a Xilinx Zynq-7000 chip included in the ZedBoard platform will be used. The objectives of this project are to test controllers implemented using the proposed techniques, compare their performance and resource efficiency against a similar controller designed directly on VHDL, and develop guidelines that serve as a guide for using HDL Coder quickly and efficiently.

Índice Abreviado

Índice

Notación

ADC	Analog to Digital Converter
ARM	Advance RISC Machine (Arquitectura del procesador con la que se designa al mismo)
BRAM	Block RAM
CCS-MPC	Continous Control Set Model Prodictive Control
CPU	Central Processing Unit
CS	Chip-Select
DMIPS	Dhrystone Million Instructions Per Second
DSP	Digital Signal Processor
e	Error en corriente
ESA	Exhaustive Search Algorithm
FCS	Finite Control Set
FF	Flip Flop
FFT	Fast Fourier Transform
FPGA	Field Prgrammable Gate Array
FPSoC	Field Programmable System on Chip
FSM	Finite State Machine
$\bar{F}_{sw}$	Frecuencia de conmutación media
GPIO	General Porpouse I/O
HDL	Hardware Description Language
HLS	High Level Synthesis
I/O	Input Output
I_{abc}	Intensidades en coordenadas abc
$I_{\alpha\beta}$	Intensidades en coordenadas $\alpha\beta$
I_{dq}	Intensidades en coordenadas dq
I^*_{dq}	Referencia de corriente en ejes dq
IGBT	Insulated Gate Bipolar Transistor
L	Inductancia de la carga
LSB	Least Significative Bit
LUT	Look-Up Table
MIO	Multiplexed I/O
MPC	Model Predictive Control
PCB	Printed Circuit Board
PLL	Phase Locked Loop
PL	Programmable Logic
PS	Processing System
R	Resistencia de la carga
RAM	Random Access Memory
RTL	Register Transfer Logic
SCLK	Señal de reloj del AD7476A
SoC	System on Chip

S_{abc}	$[S_a, S_b, S_c]^T$ Señales de actuación digitales en abc
S_{dq}	$[S_d, S_q]^T$ Señales de actuación digitales en dq
THD	Total Harmonic Distortion
T_s	Tiempo de sampleo
VSI	Voltage Source Inverter
V_{dc}	Tensión del DC-Link
V_{abc}	Tensiones en coordenadas abc
$V_{\alpha\beta}$	Tensiones en coordenadas $\alpha\beta$
V_{dq}	Tensiones en coordenadas dq
V_{dq}^{ref}	Tensión de referencia en ejes dq
V_{abc}^{ref}	Tensión de referencia en ejes abc
w	Frecuencia angular
θ	Ángulo de referencia
λ	Factor de peso de la función de coste para el ajuste de la $\bar{F}_{sw}$

1 Introducción

1.1 Motivación

En 2020 se aprobó en el parlamento europeo el acuerdo de alcanzar la neutralidad climática en 2050 [1]. En dicho documento se propone la energía eléctrica como la forma de energía principal, dejando de lado los combustibles fósiles que hasta ahora han sido las dominantes. Para alcanzar estos objetivos será necesaria la electrificación de múltiples consumos como pueden ser el transporte o la calefacción así como el aumento de la producción de energía.

Además, en los últimos años, el mercado del autoconsumo con paneles solares ha crecido a un ritmo muy acelerado a causa de la subida en el precio de la luz [2]. Subida ocasionada por la creciente demanda de energía combinada con la cada vez mayor escasez de combustibles fósiles para satisfacer dicha demanda. Situación agravada recientemente por los conflictos bélicos actuales y los cortes en el suministro que han supuesto. Los combustibles fósiles se posicionan cada vez más como un recurso estratégico y escaso del cual Europa carece y depende ya que la energía que contienen es, en esencia, la capacidad de realizar trabajo. Europa habla ahora de soberanía energética, tener la capacidad de producir energía de fuentes alternativas como pueden ser las renovables o la nuclear es ahora una prioridad para mantenerse independientes.

Este panorama es el que pone a la electrónica de potencia en auge. Es una tecnología creciente de la que se espera que aporte las soluciones necesarias a la crisis energética y que ayude a cumplir los objetivos fijados para 2050. Los retos que se plantean para la electrónica de potencia son más diversos que nunca, plantas de grandes potencias, lineas de transporte en DC [3], aplicaciones embarcadas en vehículos [4] y la carga rápida de los mismos [5] [6] y la generación distribuida [7] son algunos ejemplos de las diferentes situaciones con requisitos especiales que se deben resolver.

Es aquí donde entran en juego las herramientas de desarrollo. Serán necesarias nuevas y diversas estrategias de control para las topologías que surjan y para mejorar los controladores ya existentes [8]. Para esto, los programas de diseño a alto nivel ayudan al diseñador a hacer desarrollos más rápidos con niveles de abstracción mayores. Reducir el time-to-market es una ventaja competitiva sobretodo con la gran cantidad de mercados en expansión en este ámbito, como pueden ser la instalación de paneles solares en residencias y edificios industriales o el sector del vehículo eléctrico. HDL Coder es un ejemplo de un add-on de Matlab/Simulink que pretende simplificar el diseño de circuitos digitales para su implementación en dispositivos FPGA.

1.2 Introducción a la electrónica de potencia

La electrónica de potencia es una rama de la ingeniería que se dedica al diseño de sistemas electrónicos para la gestión eficiente de la energía eléctrica. Uno de los problemas fundamentales de gestión que aborda es el de convertir unas formas de energía a otras. La energía eléctrica, puede encontrarse en múltiples formas, en distintos niveles de tensión en DC o en tensiones con distintas amplitudes y frecuencias en AC. Para realizar la conversión existen multitud de topologías de convertidores para los distintos casos. Parte del foco de este trabajo está sobre el diseño de controladores para dichos convertidores.

El funcionamiento de los convertidores está basado en dispositivos semiconductores especializados. Son diseñados por esta disciplina para tratar de minimizar las pérdidas y aumentar las tensiones y corrientes soportables, a fin de ser eficientes cuando se requiere gestionar sistemas de gran potencia. Existen diferentes dispositivos con características propias para cubrir todas las aplicaciones posibles. Entre estos dispositivos se

incluyen los diodos de potencia y tiristores, fundamentales para los rectificadores y rectificadores controlados, o los transistores de potencia como MOSFET o IGBT, usados para multitud de aplicaciones.

Por el objetivo de ser eficientes, todos los semiconductores de potencia buscan idealmente comportarse como un elemento capaz de impedir por completo el paso de corriente o poder dejar que pase sin que haya una caída de tensión. De este modo, idealmente las pérdidas son siempre nulas. Esta es la razón por la que los transistores de potencia se emplean siempre a modo de interruptor, haciendo que el control de estos convertidores sea a través de señales digitales.

El control a través de actuaciones con valores discretos es sólo uno de los retos a los que se enfrenta el diseño de controladores para convertidores. La red de transporte de energía eléctrica se ha construido extensivamente para funcionar en corriente alterna por la robustez y simplicidad del funcionamiento del transformador. Por la gran cantidad de conexiones a la red, los responsables de la misma exigen que se cumplan ciertos requisitos de distorsión de tensión para garantizar su calidad [9]. Además, cada vez el volúmen de energía que viene de fuentes renovables es mayor respecto a la generada por centrales térmicas con generadores síncronos. Como consecuencia, la red tiene menos inercia y se requiere de los controladores de los convertidores para contribuir a su estabilidad. Estos son algunos de los objetivos entorno a los se han ideado multitud de estrategias de control que regulen el funcionamiento de los convertidores [10].

Particularmente, el problema de obtener señales de salida con una baja distorsión armónica y reducir las pérdidas parecen tener soluciones opuestas. El cambio de estado del transistor o conmutación, lleva asociada unas pérdidas, debido a que en el transitorio, ni la tensión que cae en él ni la intensidad que pasa por él son nulas. Mientras que tener frecuencias de conmutación y sampleo altas consigue mejorar la calidad armónica, también aumenta las pérdidas. Para esto la electrónica de potencia plantea soluciones tanto desde el punto de vista del control como de hardware. FCS-MPC es una estrategia de control que se basa en la optimización de una función de coste con soluciones discretas. Planteando adecuadamente esta función, es posible ajustar esta relación de compromiso THD-eficiencia con un parámetro. La ventaja de este algoritmo es que la frecuencia de sampleo puede subirse para aumentar el ancho de banda del controlador, mejorando su comportamiento dinámico, manteniendo la frecuencia de conmutación. El aumento de la frecuencia de sampleo conlleva un acortamiento de los tiempos disponibles para el cálculo de las acciones de control. Esta es la razón de que buena parte de la literatura acerca de este algoritmo implemente sus propuestas sobre plataformas basadas en FPGA, por su potencial velocidad de cálculo. El desarrollo en HDL de circuitos digitales para FPGAs no es sencillo, por lo que en este trabajo se propone el uso de herramientas de alto nivel para tratar de facilitar el diseño. Por la parte de hardware, el uso de nuevos materiales para la fabricación de dispositivos permite aumentar la frecuencia de conmutación con un menor incremento en las pérdidas.

1.3 Estado del Arte

1.3.1 Ventajas de la abstracción

Siempre ha existido una relación de compromiso entre el nivel de detalle al que se diseña un proyecto y el tiempo de desarrollo. Trabajar con un nivel de abstracción bajo permite elaborar de manera muy precisa los elementos más pequeños del proyecto. Cuando se realizan proyectos más ambiciosos, la cantidad de elementos de bajo nivel presentes puede crecer de manera que el diseño de estos uno a uno sea inabordable en tiempo y esfuerzo de diseño. Por esta razón, cuando el progreso tecnológico permite el inicio de proyectos más grandes y ambiciosos, aparecen aplicaciones que agrupan los elementos de bajo nivel dentro de elementos de una jerarquía o nivel de abstracción superior. Estas aplicaciones generan los elementos de jerarquía inferior de manera automática a partir del diseño que haga el usuario del elemento superior, reduciendo así el esfuerzo de diseño a costa de la optimalidad del diseño de los elementos inferiores. Esto es la abstracción, extraer la información relevante del conjunto de elementos inferiores separándola de las particularidades propias de nivel inferior.

En el campo de los computadores, por ejemplo, se partió desde programar directamente en binario, hasta usar programas como Matlab. Todo esto pasando por las capas como código ensamblador, drivers y sistemas operativos que dan servicios de bajo nivel a las capas superiores para que el diseñador no tenga que definir todo bit a bit. Es un ejemplo exagerado para ilustrar cómo los programas de alto nivel se construyen sobre otros que van traduciendo las instrucciones del diseñador hasta el nivel más bajo.

Otra ventaja que ofrece la abstracción, es la posibilidad de no conocer lo que hay debajo. Para programar un microcontrolador no es necesario conocer nada del código ensamblador propio de este, ya que un programa traduce las instrucciones de un lenguaje de alto nivel como C al particular del micro. Eliminando así un

proceso de aprendizaje que alargaría los tiempos de diseño y que penalizaría el usar un hardware distinto en cada proyecto. Además, el no conocer lo que hay debajo ofrece la posibilidad de que no sea tan relevante. Si existen programas que traducen un código de alto nivel a uno de bajo nivel para un dispositivo concreto, el código de alto nivel será válido para todos los programas que sepan interpretarlo y traducirlo, siendo así independiente de las capas inferiores.

En el ámbito de este trabajo, el desarrollo de diseño de circuitos digitales ha ido de igual manera de diseñar directamente el circuito con transistores al código de descripción hardware. Este código es compatible con todas las FPGAs cuyos fabricantes den las herramientas necesarias para hacer la implementación. Componentes de software como IP Integrator de Vivado abstraen aún más de las puertas lógicas que componen el circuito.

1.3.2 High Level Synthesis

Breve historia

El término High Level Synthesis hace referencia a los programas que en los últimos años han aparecido para añadir una capa por encima al proceso de diseño de sistemas digitales [11]. Estos programas vienen a continuar con lo anteriormente mencionado sobre la necesidad de ser capaces de abarcar proyectos cada vez más grandes. En el mercado competitivo actual los tiempo de desarrollo son clave para la explotación de un producto y al tener cada vez una mayor demanda de dispositivos digitales, la demanda de mejores herramientas de desarrollo también crece.

Hoy día existen multitud de herramientas de este tipo con cierto recorrido en el mercado y cuyo uso es creciente. Unos ejemplos de empresas conocidas pueden ser VivadoHLS de Xilinx y CtoS de CADENCE [11]. Estas herramientas reciben como entrada un código C que traducen a un lenguaje de descripción hardware como VHDL o Verilog para que luego una herramienta sintetice este código en un circuito y después en un bitstream que cargarle a una FPGA.

Los primeros intentos de desarrollar este tipo de herramientas datan de proyectos académicos, entorno a 1970 en la Carnegie Mellon University, antes de que aparecieran a nivel industrial en los 90 [12]. Esta primera generación de herramientas HLS fracasasaron debido a que hacer diseños RTL en aquella época era todavía manejable y los diseñadores seguían aprendiendo a usar lenguajes HDL para generar síntesis RTL [13]. Entorno a la década de los 2000 apareció una nueva generación que se centró en generar código a partir del ya establecido C/C++. Esto supone un debate en la actualidad debido a que C es un lenguaje completamente centrado en sistemas secuenciales como son los microprocesadores y buena parte de la ventaja del uso de FPGAs es su potencialidad para realizar múltiples tareas en paralelo. Se cuestiona por tanto, si la mejor opción de código de input para los programas de HLS es un código C. Como respuesta a esto algunas herramientas modernas usan variantes de C e incorporan instrucciones adicionales más propias de un diseño hardware, algunos ejemplos son Handle-C o HardwareC. Otras alternativas que han surgido son las herramientas cuyo código de entrada es código MATLAB/Simulink como HDL Coder de la propia Mathworks o AccelDSP de Xilinx ya abandonada.

En el gráfico mostrado en la Figura 1.1 se representa la inversión en Programas de HLS durante esos años.

En la Figura 1.2 se puede ver una lista más completa de programas de HLS con sus lenguajes de entrada y más información relevante.

Con la compra de *AutoESL Desing Technologies*, una empresa emergente en la comunidad de HLS, por parte de Xilinx, una de las principales empresas relacionadas con las FPGAs, en 2011, el mercado de los programas de HLS se asentó y se plantea como una industria con muy buena proyección de futuro. Cada vez surgen más artículos innovadores en el uso de este tipo de herramientas y con logros cada vez más relevantes [15]. Estos logros se reparten por numerosos ámbitos debido a la flexibilidad de las FPGAs.

Casos prácticos actuales del uso de HLS

En este apartado se realizará un repaso de algunos artículos en los que se usan herramientas de HLS tanto fuera como dentro del campo de la electrónica de potencia.

El primer ejemplo usa una aproximación híbrida entre el RTL y el HLS para implementar algoritmos de redes neuronales (Deep Neural Networks) [16]. Sintetiza código C++ y OpenCL con un software de Altera de HLS para generar bloques de código tipo VHDL que luego integran junto con código de descripción hardware hecho a mano para hacer la implementación en una FPGA de Altera. Este es un buen ejemplo de un proyecto que con técnicas clásicas de diseño de RTL sería prácticamente inabarcable debido al coste humano en tiempo de diseño y al ritmo al que evolucionan este tipo de algoritmos hoy en día. Además, los profesionales dedicados al diseño de redes neuronales no suelen estar familiarizados con los lenguajes

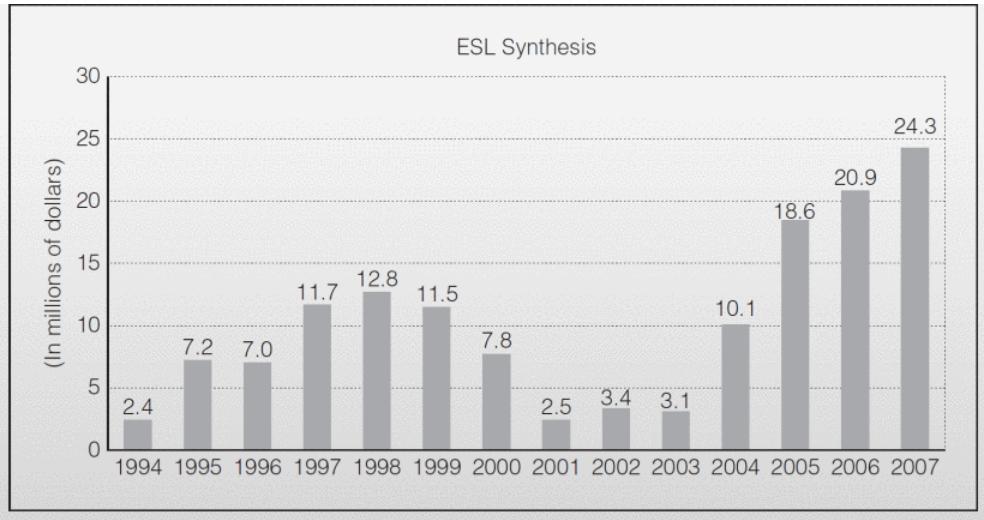

Figura 1.1 Gráfica de inversión en HLS [13].

de diseño hardware como VHDL o Verilog por lo que requieren de programas de HLS para abstraerse del circuito digital cuando afrontan este tipo de tareas.

En el ámbito de la secuenciación genómica, para acelerar el procesado de la información contenida en los genes, también han tenido éxito este tipo de herramientas y por las mismas razones [17].

En el uso de HDL Coder también existe bibliografía sobre la implementación de osciladores hiper-caóticos [18]. Estos sistemas son útiles para muchas aplicaciones de ciberseguridad donde es necesario generar números realmente aleatorios. En el mundo digital, la lógica binaria impone que todo esté representado por "1" y "0", haciendo que todo ocurra de manera determinista, por lo que para generar aleatoriamente un número se necesitan sistemas altamente complejos donde saber información del circuito no sea suficiente para determinar su evolución y por lo tanto determinar su salida. Gracias a las técnicas de HLS y en concreto HDL Coder ahora surgen trabajos sobre la implementación de osciladores hiper-caóticos en FPGA a partir de un esquema de Simulink.

En [19] se realiza un estudio más concreto de la efectividad de los algoritmos de HLS a la hora de generar RTL de manera tan eficiente como las técnicas de HDL. Para ello, crea un diseño que realiza una serie de cálculos, los implementa en una ZedBoard con un reloj de 100 MHz y compara resultados obtenidos con Vivado HLS y con Vivado HDL. Llega a la conclusión de que aunque las técnicas de HLS permiten diseñar de manera más sencilla se llegan a resultados menos eficientes en área. Sin embargo, admite que las herramientas de HLS permiten introducir comandos propios de la descripción hardware y que existen opciones para mejorar los resultados a costa de un mayor esfuerzo de diseño.

El artículo [12] realiza un estudio más sistemático en el que, a parte de realizar un repaso de la historia y avances en optimizaciones del HLS, comprueba como para ciertas tareas el uso de HLS puede incluso conllevar una mejora tanto en área usada como en tiempo de trabajo. Para realizar la comparación los autores definen una serie de operaciones matriciales y ponen a un diseñador de HDL experto contra otro que use AutoPilot HLS. En base a las métricas ofrecidas en el artículo se comprueba que en semanas de trabajo el diseñador que usa el HLS tarda menos en llegar a resultados comparables a los del HDL. Cuando las semanas de trabajo son comparables el diseñador que trabaja con AutoPilot logra llegar a un diseño que requiere menos recursos gracias a los algoritmos que aplica el HLS. A pesar de estos resultados alentadores, se muestran también deficiencias de estas herramientas. El proceso de verificación a alto nivel es más sencillo puesto que parte de la base de que si el código en C funciona el sistema electrónico funcionará también pero esto no es asegurable por la herramienta de HLS. En la práctica significa que un buen proceso de verificación se hace más complejo con HLS. La secuencialidad característica de un lenguaje como C no es la más apropiada cuando se pueden realizar tareas en paralelo. Aunque la optimización de los bucles de código si consigue realizar optimizaciones muy buenas, si las tareas son completamente distintas el HLS tiene dificultades para realizarlas en paralelo.

Pasando al ámbito de la electrónica de potencia, en [20] se aborda el diseño de un controlador del tipo FCS-MPC de horizonte largo para una topología multinivel aplicado a un motor de inducción. El artículo obtiene resultados de un controlador con horizonte 3 y con una frecuencia de conmutación media de 300 Hz. A pesar de la frecuencia tan baja a la que buscan conmutar, por ponerse en un supuesto de gestión de alta

Status	Compiler	Owner	License	Input	Output	Year	Domain	Test Bench	FP	FixP
In Use	Stratus HLS	Cadence Design Systems	Commercial	C/C++ SystemC	RTL	2015	All	Yes	Yes	Yes
	AUGH	TIMA Lab.	Academic	C subset	VHDL	2012	All	Yes	No	No
	eXCite	Y Explorations	Commercial	C	VHDL/Verilog	2001	All	Yes	No	Yes
	Bambu	PoliMi	Academic	C	VHDL/Verilog	2012	All	Yes	Yes	No
	Bluespec	BlueSpec Inc.	BSD-3	BSV	SystemVerilog	2007	All	No	No	No
	QCC	CacheQ Systems, Inc.	Commercial	C/C++/Fortan	Host Executable + FPGA Bit file (SystemVerilog is intermediate)	2018	All - multi-core and heterogeneous compute	Yes (C++)	Yes	Yes
	HDL Coder	MathWorks	Commercial	MATLAB, Simulink, Stateflow, Simscape	VHDL / Verilog	2003	Control Systems, Signal Processing, Wireless, Radar, Communications, Image and Computer Vision	Yes	Yes	Yes
	CyberWorkbench	NEC	Commercial	BDL, SystemC	VHDL/Verilog	2011	All	Cycle/ Formal	Yes	Yes
	Catapult	Mentor (Siemens business)	Commercial	C, C++, SystemC	VHDL/Verilog	2004	All	Yes	Yes	Yes
	DWARV	TU. Delft	Academic	C subset	VHDL	2012	All	Yes	Yes	Yes
	GAUT	U. Bretagne	Academic	C/C++	VHDL	2010	DSP	Yes	No	Yes
	Hastlayer	Lombiq Technologies	BSD-3	C#/C++/F#... (.NET)	VHDL	2015	.NET	Yes	Yes	Yes
	Instant SoC	FPGA Cores	Commercial	C/C++	VHDL/Verilog	2019	All	Yes	No	No
	Intel High Level Synthesis Compiler	Intel FPGA (Formerly Altera)	Commercial	C/C++	Verilog	2017	All	Yes	Yes	Yes
	LegUp HLS	LegUp Computing	Commercial	C/C++	Verilog	2015	All	Yes	Yes	Yes
	LegUp	U. Toronto	Academic	C	Verilog	2010	All	Yes	Yes	No
	MaxCompiler	Maxeler	Commercial	MaxJ	RTL	2010	DataFlow	No	Yes	No
	ROCCC	Jacquard Comp.	Commercial	C subset	VHDL	2010	Streaming	No	Yes	No
	Symphony C	Synopsys	Commercial	C/C++	VHDL/Verilog/ SystemC	2010	All	Yes	No	Yes
	VivadoHLS (formerly AutoPilot from AutoESL[13])	Xilinx	Commercial	C/C++/SystemC	VHDL/Verilog/ SystemC	2013	All	Yes	Yes	Yes
	Kiwi	U. Cambridge	Academic	C#	Verilog	2008	.NET	No	Yes	Yes
	gcc2verilog	U. Korea	Academic	C	Verilog	2011	All	No	No	No
	HercuLeS	Ajax Compilers	Commercial	C/NAC	VHDL	2012	All	Yes	Yes	Yes
	Shang	U. Illinois	Academic	C	Verilog	2013	All	Yes	?	?

Figura 1.2 Tabla herramientas HLS [14].

potencia, la frecuencia a la que muestrean es de 40 kHz. Para realizar todo el cálculo del horizonte largo en 25 μs aplican un "sphere decoding algorithm". Este algoritmo es empleado para no tener que evaluar todos los estados sino que evalúa sólo los estados internos a la esfera de la solución actual y reduce su radio en función de las evaluaciones. El diseño lo hacen a través de la aplicación Vitis HLS de Xilinx, una herramienta de HLS que toma código en C++ para crear el VHDL que implementan en una Zynq UltraScale+ 9EG. Para el código del ARM usan el add-on *Embedded Coder* de MATLAB. Además incluye en sus resultados el uso de recursos de la FPGA.

En [21] implementan de manera similar un FCS-MPC de horizonte 4 usando Vivado HLS pero para una Zynq UltraScale+ 7EV-2. Usa la misma frecuencia de muestreo, el mismo algoritmo para la búsqueda del

estado óptimo y frecuencias de conmutación de 300 Hz y 800 Hz. También usa una topología multinivel. En este artículo sin embargo obtienen los resultados mediante simulación con una FPGA-in-the-loop que se configura tanto para simular el convertidor como un motor de inducción. Los resultados también incluyen el uso de recursos de la FPGA.

En [22] se propone un algoritmo de tipo evolutivo-genético para realizar una reconfiguración dinámica de un campo de paneles fotovoltaicos. Esta estrategia pretende minimizar las pérdidas producidas por el sombreo parcial de un campo de paneles tratando de conectar en cada momento los paneles de modo que queden en puntos de funcionamiento lo más parejos posibles. De este modo se consigue un mejor funcionamiento de los algoritmos de búsqueda del punto de máxima potencia. Además se consigue proteger los paneles de puntos de funcionamiento que produzcan un deterioro y permite un mejor monitoreo y diagnóstico de problemas. En [23] este algoritmo es implementado sobre la Zynq-7000 de una ZedBoard usando Vivado HLS. Se consigue reducir los tiempos de cálculo aprovechando las capacidades de los SoC y se realiza una discusión sobre el uso de los recursos de hardware. Por último en [24] los mismos autores realizan optimizaciones en su algoritmo mediantes aceleradores de hardware implementados con HLS y realizan la implementación en una placa Zybo basada en la Zynq-7000.

El artículo [25] realiza una comparativa entre una implementación de un algoritmo de sincronización con la red hecha en HLS frente a otra hecha directamente en HDL. Usan Vivado HLS como herramienta para trabajar a alto nivel y sus resultados muestran que para este caso la implementación hecha en HLS superaba a la hecha con técnicas de bajo nivel excepto en el timming. Este punto no lo consideran una debilidad en el artículo dado que el HLS reducía de manera considerable el esfuerzo requerido. Este trabajo se desarrolla sobre una placa basada en la FPGA Spartan-3.

Con estos ejemplos se ve que el uso de herramientas de HLS es cada vez más común y previsiblemente lo será aún más en el futuro. No requerir un conocimiento avanzado sobre circuitos digitales y FPGAs para poder trabajar con ellas permitirá que muchas más personas puedan emplearlas en sus trabajos. Además los resultados obtenidos con HLS, a pesar de tener ciertas limitaciones, pueden estar a la altura de los realizados directamente en HDL especialmente si se tiene en cuenta el tiempo de diseño. Precisamente por esto último, los programas de HLS son un buena opción a la hora de realizar investigación y desarrollo de nuevas aplicaciones. Como ya se ha visto anteriormente, el uso de FPGAs en el campo de la electrónica de potencia está plenamente justificado, por lo que es de esperar que aparezcan más artículos que usen HLS. Con los artículos que se han mostrado en este apartado se ve que que el uso de estas heramientas en este campo está siendo exitoso.

1.4 Objetivos

Para el desarrollo del proyecto se harán varios diseños haciendo uso de la herramienta HDL Coder para después comparar los resultados obtenidos con los que se obtendrían haciendo el diseño directamente en VHDL con programas como Vivado. Hacer diseños de circuitos digitales con VHDL requiere conocer bien el lenguaje, tener cierta experiencia con él y tener un entendimiento profundo de cómo funcionan estos circuitos para poder plantear el diseño. Esto es un proceso largo y unos requisitos difíciles de adquirir rápidamente. Ahorrar estas necesidades antes de hacer el diseño hardware es parte de la intención de HDL Coder. Por esto, este proyecto también tratará de ser una guía básica del proceso de diseño en HDL Coder para aplanar en lo posible la curva de aprendizaje de esta herramienta.

De este modo los objetivos generales de este documento serán los siguientes:

1. Estudiar el uso del software para la generación de código
2. Diseñar un flujo de trabajo para trabajar en este entorno usando la combinación de herramientas de Mathworks que se considere más oportuna.
3. Implementar un control PI para la regulación de corrientes en una carga RL conectada a un VSI trifásico de dos niveles siguiendo la metodología propuesta.
4. Implementar un control FCS-MPC para el mismo sistema siguiendo también la misma metodología.
5. Comparar los resultados de estas implementaciones con los de otros controladores similares diseñados directamente en VHDL, tanto en desempeño como en consumo de recursos de la FPGA y de tiempo de cálculo.
6. Obtener conclusiones acerca de la eficiencia de la metodología y de la viabilidad de usar HDL Coder en el ámbito de los controladores de convertidores de potencia.

2 Fundamento matemático de los controladores propuestos

Este capítulo se dedica a explicar la base matemática sobre la que se diseñan los controladores propuestos para el estudio. Estos controladores son el control PI y el control FCS-MPC.

2.1 Sistema VSI y ecuaciones que rigen el sistema

Los controladores de este proyecto se diseñan para un convertidor DC/AC o "Voltage Source Inverter" (VSI) de tipo trifásico de 2 niveles. Este convertidor permite convertir energía eléctrica en forma de tensión en corriente continua a energía eléctrica en forma de tres tensiones de alterna. Su principio de funcionamiento es conectar cada una de las tres fases a la tensión de DC o a su tierra para generar tres corrientes senoidales desfasadas entre si. En la Figura 2.1 se muestra un esquemático del convertidor conectado a una carga trifásica RL. En ella se puede ver las definiciones de las tensiones y las corrientes que se van a usar para el modelado además de las señales de control para los dispositivos. R denota la resistencia de la carga y L su inductancia.

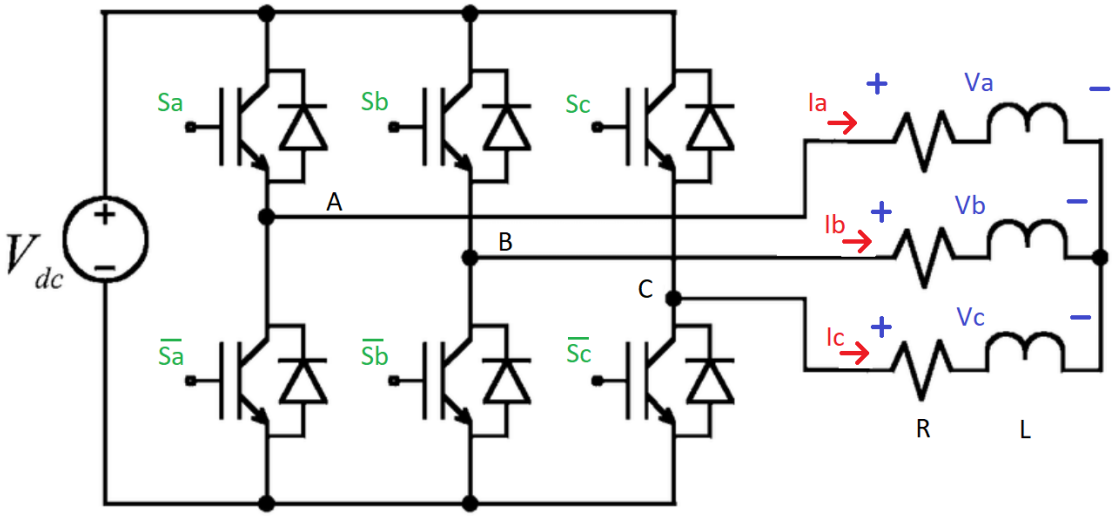

Figura 2.1 Esquemático del convertidor VSI de 2 niveles.

Con la magnitudes eléctricas definidas en el esquemático se define el vector $V_{abc} = [V_a, V_b, V_c]^T$ y el $I_{abc} = [I_a, I_b, I_c]^T$. La ecuación que rige el sistema queda como (2.1)

$$V_{abc} = \frac{dI_{abc}}{dt} L + R I_{abc} \tag{2.1}$$

Como se ve en la definición de las variables de control S_a,S_b,S_c cada fase tiene una señal asociada. Cada fase tiene dos dispositivos que se ponen en *on* o en *off* de manera opuesta, es decir, sus señales de control son la negada de la otra, para así asegurar que nunca se cortocircuita la tensión de DC. Los estados del convertidor quedan definidos por las señales de control, siendo el estado (1,1,1) el que tiene en *on* todos los interruptores superiores dejando las tensiones V_a,V_b,V_c a 0. Lo mismo ocurre con el (0,0,0) pero conectados a tierra. En el resto de los casos quedan dos fases de la carga en paralelo en serie con la tercera conectadas entre la tensión de DC y tierra. En la Figura 2.2 se representa el ejemplo del estado (1,0,0) el la que, tomando las referencias de tensión definidas en la figura anterior $V_a = 2 \times V_{dc}/3$, $V_b = -V_{dc}/3$, $V_c = -V_{dc}/3$. Quedan de este modo $2^3 = 8$ estados posibles que alternándolos se consigue que las tensiones a la salida del convertidor creen unas corrientes senoidales en la carga.

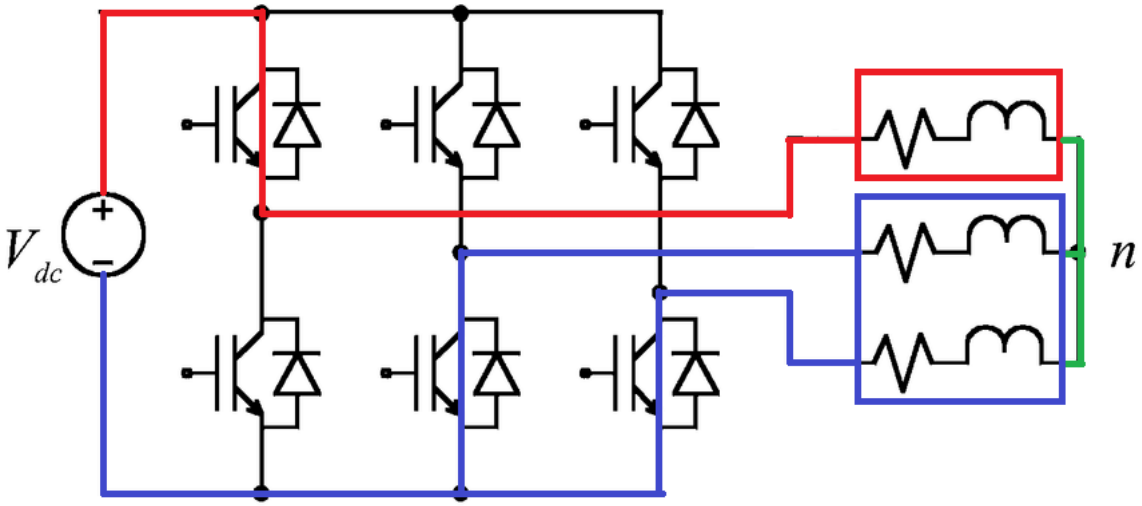

Figura 2.2 Esquemático del convertidor funcionando en el estado (1,0,0).

El objetivo del control será regular las corrientes que entran a la carga trifásica conectada al convertidor. Estas corrientes se deberán corresponder con las de un sistema trifásico, es decir, senoidales, con la misma amplitud y frecuencia y desfasadas 120º entre si, con la menor distorión armónica posible. Estas corrientes oscilarán a una frecuencia de 50 Hz definiendo θ como el ángulo de referencia que indica la fase y w la frecuencia angular que es igual a $50 \times 2\pi$.

2.2 Transformaciones de los ejes de referencia

Antes de entrar directamente con las dos estratégias de control propuestas conviene repasar antes las dos transformaciones matemáticas que se van a usar para realizar el control en unos ejes de coordenadas más convenientes. Estas son las transformadas de Clarke y de Park, ampliamente conocidas en el ámbito de la electrónica de potencia.

2.2.1 Transformación de Clarke

En un sistema trifásico de tres hilos se tienen tres fases en las cuáles sus corrientes suman 0 por ley de nudos de Kirchoff, ya que no existe un neutro que sirva de vía alternativa para la corriente. Esto es una relación lineal que implica que sólo son necesarias dos de las tres variables para definir el estado del sistema ya que la tercera variable vendrá dada por las otras dos. Partiendo de esta base, la transformación de Clarke permite pasar el vector I_{abc} a unos ejes $\alpha\beta$ mediante la expresión (2.2). Este vector gira entorno al origen de coordenadas a la velocidad angular de referencia. Esta transformación es útil porque extrae la información relevante de las magnitudes medidas y la plantea de una manera más sencilla en un sistema de orden reducido.

$$I_{\alpha\beta} = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} I_{abc} \tag{2.2}$$

2.2.2 Transformación de Park

Partiendo de los ejes $\alpha\beta$, añadiendo la información de la fase de sincronización para crear unos ejes síncronos dq, que giren a la frecuencia w, entorno al origen de los ejes $\alpha\beta$. Mediante la (2.3), se puede conseguir que las intensidades queden representadas como un vector fijo en ejes dq. En un caso real donde el equipo esté conectado a la red, la información sobre la fase se obtendría a partir de un PLL que reciba las medidas de la tensión de la red. De este modo la referencia de intensidad en ejes dq está directamente relacionada con las potencias activas y reactivas inyectadas a la red. En este trabajo, al tratarse de una carga *RL* sin conexión a la red, la generación de θ del sistema se hará como parte del control.

$$I_{dq} = \begin{bmatrix} cos(\theta) & sen(\theta) \\ -sen(\theta) & cos(\theta) \end{bmatrix} I_{\alpha\beta} \tag{2.3}$$

2.3 Control PI

El control Proporcional Integral tiene un principio de funcionamiento muy simple a la par que efectivo, lo que le ha llevado a ser unos de los métodos de control más usados. Su funcionamiento es tomar el error y su integral en el tiempo, multiplicarlos por unas constantes respectivamente (K_p y K_i) y la suma será la actuación que se aplique al sistema. Si las constantes se eligen correctamente, el sistema será estable y alcanzará la referencia sin error en régimen permanente. Por esto el PI se usa para seguir referencias constantes puesto que consigue errores en régimen permanente nulos, razón por la cuál es especialmente útil la transformada de Park.

Se define el error de intensidad e en (2.4) donde I_{dq} es la intensidad medida en ejes dq e I^*_{dq} es la referencia.

$$e = I_{dq} - I^*_{dq} \tag{2.4}$$

De este modo la actuación de control queda como se indica en (2.6) siendo el término $-JwLI_{dq}$ un término de desacoplo entre las dinámicas de las componentes del vector I_{dq} que viene de derivar la matriz de Park respecto al tiempo. Donde w es la derivada de θ respecto al tiempo y, en este caso, la tensión u_{dq} se corresponde con la tensión de cada fase al nodo negativo de la fuente de DC tras hacer las transformaciones correspondientes.

$$J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{2.5}$$

$$u_{dq} = -JwLI_{dq} + K_p e + K_i \int e dt \tag{2.6}$$

La implementación de PI se ha hecho a través de una ecuación en diferencias. Se parte de la función de transferencia en z del PI obtenida con el método de integración "Forward Euler" a la que se le ha añadido un retraso por el tiempo de cómputo (2.7), y se despeja u(z). El paso a ecuación en diferencias es directo (2.8). Con la ecuación en diferencias se acaba teniendo dos constantes obtenidas a partir de las K_p y K_i iniciales y T_s, que multiplican al error presente y pasado en lugar del error y su integral. De esta manera se ahorra tener que hacer el cálculo de la integral del error en la implementación, ya que este cálculo viene implícito en la ecuación en diferencias.

$$\frac{u(z)}{e(z)} = K_p \left(1 + K_i T_s \frac{1}{z-1}\right) z^{-1} \tag{2.7}$$

$$u_{k+1} = u_k + K_p e_k + K_p (K_i T_s - 1) e_{k-1} \tag{2.8}$$

Una vez calculado el término del PI por ecuación en diferencias se deberá sumar el término de desacoplo $-JwLI_{dq}$.

Con la actuación calculada en ejes dq se debe de realizar una transformación inversa para traerla de nuevo a los ejes abc usando las matrices traspuestas de la transformada de Park y de Clarke.

La actuación calculada son valores continuos de tensión que para traducirlos a señales booleanas de control para los dispositivos requiere un modulador PWM. Como los valores de tensión en el convertidor son discretos la técnica de Pulse Width Modulation establece que la tensión media por periodo sea igual a la tensión de la actuación.

Tabla 2.1 Definiciones de las variables de las ecuaciones (2.7) y (2.8).

Variable	Significado
z	Variable del dominio de la transformada z
$u(z)$	Actuación en variable z, se corresponde con la tensión V_{dq}
$e(z)$	Error e en variable z
K_p	Constante proporcional del PI
K_i	Constante integral del PI
T_s	Tiempo de muestreo
u_k	Actuación en el instante k
u_{k+1}	Actuación en el instante k+1
e_k	Error en el instante k
e_{k-1}	Error en el instante k-1

2.4 Control FCS-MPC

El control predictivo se basa en un modelo matemático del sistema que se pretende controlar. A partir de este modelo y de la información sobre el estado del sistema que se obtengan a través de las medidas se buscan las señales de actuación óptimas que llevarían al modelo al estado de referencia. Por esto, a diferencia del PI, es una estrategia de control sin memoria, que para cada periodo de muestreo calcula la actuación óptima sin tener en cuenta ni los estados del sistema pasados ni las actuaciones anteriores.

Existen multitud de tipos de control predictivo. El objetivo final de este proyecto es poder hacer una comparación entre la implementación de controladores con técnicas de bajo nivel y con el programa HDL Coder, un programa de HLS. Por esta razón se elige una tipología que ha sido implementada en otros trabajos en la misma plataforma pero programada en VHDL y de la que se tienen los resultados para poder hacer la comparación [26]. Esta tipología es FCS-MPC. Esta variante del control predictivo, a diferencia de la CCS-MPC trabaja directamente con una señal de control discreta que se corresponde con los posibles estados del convertidor de potencia y se salta la necesidad de usar un modulador. En el caso de un inversor trifásico de dos niveles, son los 8 descritos anteriormente. Más concretamente el algoritmo de control será de tipo Optimal Switching Vector (OSV-MPC) que decide en cada tiempo de muestreo que interruptores abrir o cerrar. De este modo no tiene una frecuencia de conmutación fija sino que decide en base a una función de coste cuando conmutar.

Para el modelado dinámico del sistema se tiene la siguiente ecuación, donde V_{dq} representa la tensión respecto al neutro de la carga V_{abc} a la que se le han realizado las transformaciones de Clarke y Park.

Que la función diferencial (2.9) escrita en forma de espacio de estados (formulación genérica en (2.10) y (2.11) queda como se ve en la ecuación (2.12).

$$V_{dq} = JwLI_{dq} + \frac{dI_{dq}}{dt}L + RI_{dq} \tag{2.9}$$

$$x_{k+1} = Ax_k + Bu_k \tag{2.10}$$

$$y_k = Cx_k + Du_k \tag{2.11}$$

Tabla 2.2 Significado de las variables de la descripción de espacio de estados genérica.

Variable	Significado
x	Estado
y	Salida
u	Actuación
A	Matriz de estados
B	Matriz de entrada
C	Matriz de salida
D	Matriz de transmisión directa

$$\frac{dI_{dq}}{dt} = -\begin{bmatrix} \frac{R}{L} & -w \\ w & \frac{R}{L} \end{bmatrix} I_{dq} + \begin{bmatrix} \frac{1}{L} & 0 \\ 0 & \frac{1}{L} \end{bmatrix} V_{dq} \tag{2.12}$$

Donde el estado del sistema son las intensidades, la actuación las tensiones, la matriz de estado A es la que multiplica a las intensidades y la matriz B la que multiplica a las tensiones. La matriz C es la matriz identidad por tomar el estado igual a la salida. La matriz D es cero.

Para aplicar este modelo continuo en el sistema digital, debe ser discretizado. Para esta tarea MATLAB dispone de una toolbox llamada "Control system toolbox" que incluye la función "ss()" para crear modelos continuos en espacio de estados y la función "c2d()" que permite discretizar esos modelos incluyendo el tiempo de muestreo [27]. Esta discretización se lleva acabo mediante un mantenedor de orden cero. Una vez se tiene el modelo discretizado, ya se tienen las matrices A y B que se implementan en la FPGA para hacer predicciones.

El control predictivo permite establecer más objetivos de control que el de simplemente alcanzar la referencia. La elección del estado óptimo se hace a través de una función de coste que puede incluir más términos a parte del término de error de seguimiento. De este modo se puede moldear un comportamiento para el control. En el caso del control para un VSI el término adicional que se incluye es uno asociado a las conmutaciones dado que estas tienen unas pérdidas asociadas. De este modo se puede ajustar un parámetro que limite las conmutaciones reduciendo la frecuencia de conmutación media $\bar{F}_{sw}$.

El método elegido para encontrar el estado óptimo es un método ESA (exaustive searching algorithm) que funciona calculando las funciones de coste para todas las actuaciones posibles y obteniendo el mínimo. La función de coste es una ponderación del error en corriente y del número de conmutaciones donde un parámetro λ ajusta el peso del segundo término (2.13) [28]. El número de conmutaciones es el número de fases que cambian de estado para pasar del estado actual al siguiente. Por ejemplo, para pasar del estado (1,1,0) al (1,0,1) corresponden 2 cambios.

$$coste_j = \left\| I_{dq}^{k+2} - I_{dq}^{*k+2} \right\|_2^2 + \lambda \left\| S_{abc,j}^{k+1} - S_{abc}^{k} \right\|_2^2 \tag{2.13}$$

3 Hardware

En esta capítulo se describe brevemente el hardware sobre el que se trabajará adicionalmente a la plataforma sobre la que se implementará el control.

3.1 Hardware del laboratorio

3.1.1 Zynq-7000 y ZedBoard

Como ya se ha mencionado anteriormente, la plataforma elegida para implementar los distintos controladores es una Zynq-7000 versión Z-7020 [29]. Esta FPSoC de Xilinx integra dos núcleos de ARM Cortex-A9 y una FPGA de la serie Artix-7 [30]. La combinación de procesadores con FPGAs da tanto una gran potencia de cálculo secuencial como paralelo. La Zynq-7000 queda dividida por tanto en 2 partes principales el Processing System (PS) y la Programmable Logic (PL). La PL es fundamentalmente la FPGA y la PS se compone de los núcleos de ARM y de los periféricos de comunicación, I/O y memoria RAM. Algunas de las características de estos componentes son:

Del ARM (PS)

- Dos ARM Cortex-A9 MPCores
- 2.5 DMIPS/MHz por CPU
- Hasta 866 MHz en cada CPU
- Soporta funcionamiento en un único procesador y en ambos tanto de manera simétrica como asimétrica
- Soporta el uso de punto flotante de precisión "single" y "double"

De la FPGA (PL)

- Familia Artix-7
- 53200 LUTs
- 106400 Flip-Flops
- 4.9 Mb repartidos en 140 Block RAMs de 36Kb. Hasta 72 bits de ancho. Dual Port. Configurable como block RAM de 18Kb.
- 220 DSP slices

Las LUTs y los Flip Flops de la FPGA se organizan entorno a Configurable Logic Blocks (CLB). En la familia Artix-7 estos CLBs se conforman cada uno por dos slices. Estos a su vez son los que están compuestos por LUTs y Flip Flops con los que se implementan las funcionalidades del circuito. Las LUTs son el componente que contiene la tabla de verdad de las funciones lógicas programadas y los Flip Flops son los biestables que se usan para configurar los registros. Además los slices contienen lo necesario para configurarlos de todas las maneras que se requieran y un bit de acarreo. Hay dos tipos de slices en esta familia, los SCLICEL, que solo disponen de recursos para hacer operaciones lógicas o aritméticas mientras que los SLICEM disponen ademas de bloques de memoria. En la Figura 3.1 se presenta el esquemático de este tipo

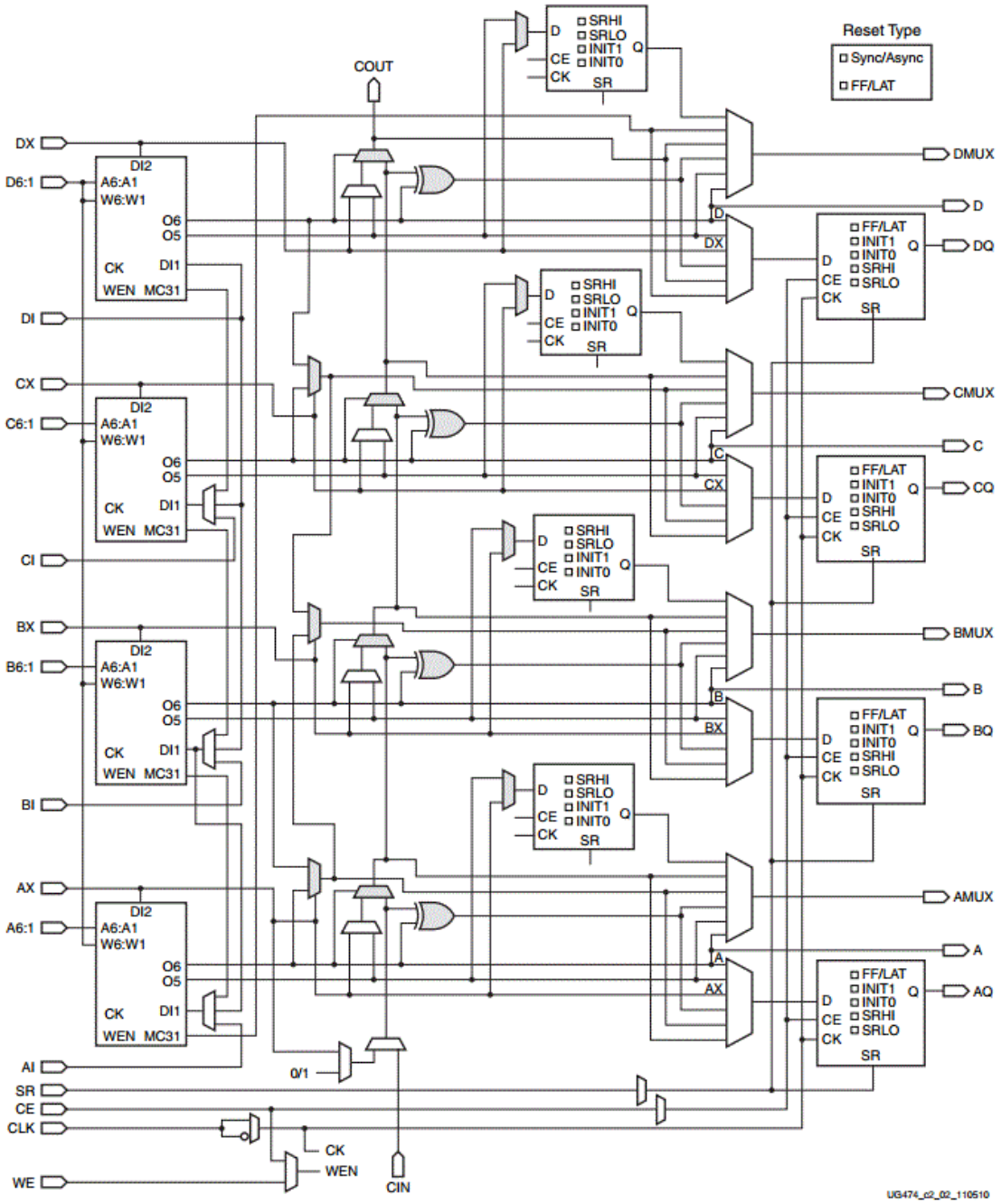

Figura 3.1 Esquemático de un SLICEM.

de slice para dar una visión de la estructura que tienen. La proporción de estos slices en la FPGA es de 2:1 para el SLICEL.

Los DSP slices son elementos específicos que contiene la Zynq para realizar cálculos aritméticos. Están especializados para realizar operaciones del tipo suma o multiplicación de manera más eficiente que a como se haría mediante CLBs. Así se ahorra una gran cantidad de recursos en forma de LUTs y flip flops. Incluyen multiplicadores de 18 x 25 bits con signo y sumadores de 48 bits entre otros elementos que agilizan las operaciones aritméticas. En la Figura 3.2 se observa el esquemático de uno de estos slices.

Además incluye un conjunto de periféricos (PS) entre los que destacan:

- Dos periféricos Ethernet MAC de velocidades 10/100/1000 que soportan el estándar IEEE 802.3 y 1588 rev 2.0.
- Dos controladores para una memoria externa SD/SDIO2.0/MMC3.31.

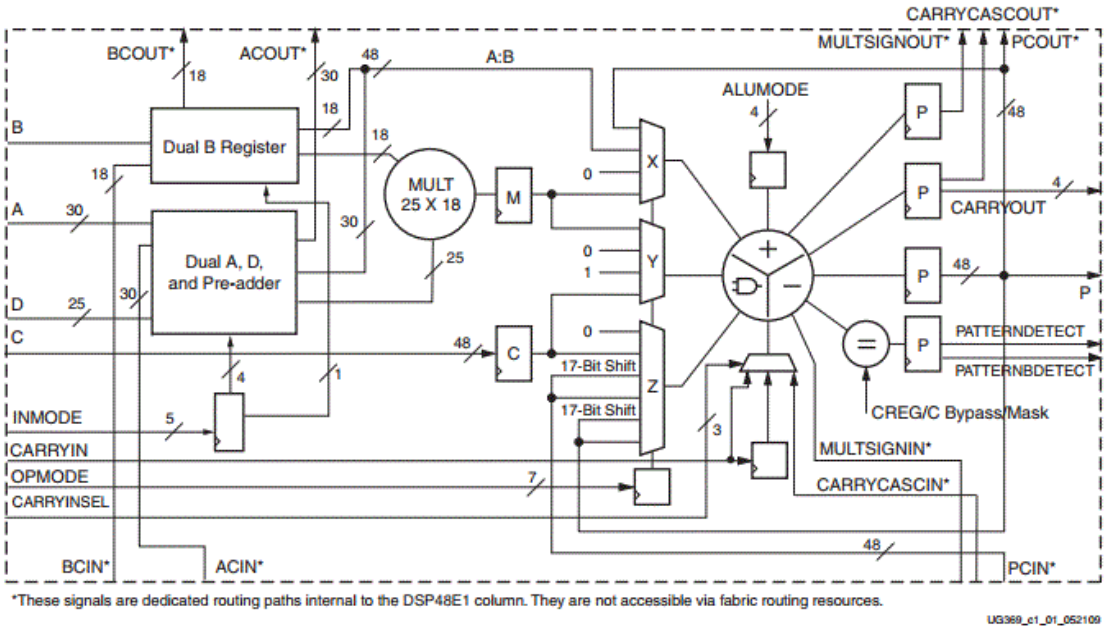

Figura 3.2 Esquemático de un DSP slice DSP48E1 presentes en la Zynq [31].

- Dos UART de alta velocidad (1 MHz). Full-dúplex y configurable para distintos protocolos y varias velocidades. Líneas de lectura y escritura independientes, cada una con una FIFO de 64 bytes.
- Hasta 54 pines dedicados para I/O (MIO).
- Dos ADC de 12 bits.

Respecto a la ZedBoard, (Zynq Evaluation & Development Board) es una placa de desarrollo para la Zynq-7000 fabricada por AVNET. Incluye multitud de interfaces para la comunicación. Estas características se ven resumidas en la Figura 3.3. De entre ellas, para este proyecto destacan.

- El puerto USB-UART conectado al periférico UART de la Zynq (PS) a través del que se realizará la programación de la placa.
- La tarjeta SD de 4 GB en la que se carga el sistema operativo proporcionado por el Support Package de Xilinx para que el ARM realize su operación de booteo de la placa.
- El reloj de 100 MHz que va directamente a la PL
- Los cuatro puertos Pmods que contienen los pines multiplexados para I/O (MIO) y que se usan en este proyecto para conectar las distintas placas externas. Estos puertos contienen 2x6 pines más dos de alimentación a 3.3 y dos de tierra.
- Los componentes asociados al GPIO de la PL. 8 leds, 8 interruptores y 5 botones configurables como salidas o entradas para la FPGA.

3.1.2 Convertidor de potencia y carga

El convertidor con el que se ha trabajado a lo largo del proyecto es un inversor trifásico de dos niveles basado en IGBTs modelo SKM100GB12T4, con un driver del fabricante Semikron que tiene recibe las señales de disparo por fibra óptica. El convertidor y la placa de drivers se muestran en la Figura 3.4. El equipo también posee un condensador de 1700 μF para su DC-Link con un sensor de tensión modelo LV25-P. Para aportar la tensión de DC en el laboratorio se dispone de una fuente de tensión regulable de American Reliance. La carga para las pruebas es una carga *RL* en estrella de 30 Ω y 20 mH, mostrada en la Figura 3.5 que puede conectarse en paralelo con una igual quedando como 15 Ω y 10 mH. Ambas configuraciones se usan en las pruebas para obtener resultados comparables a otros obtenidos en el mismo equipo en el laboratorio.

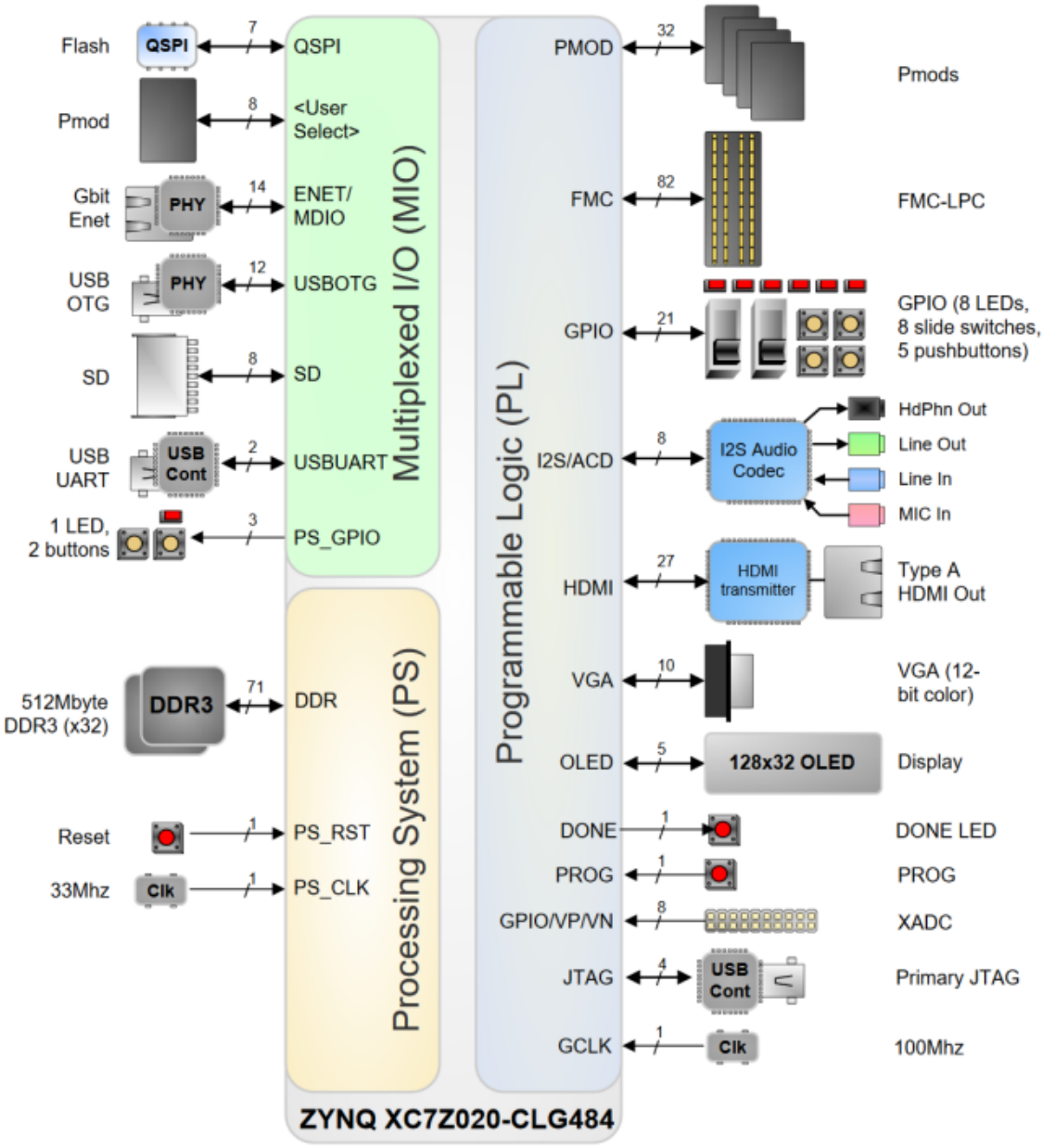

Figura 3.3 Esquema de las conexiones de la ZedBoard a la Zynq-7000 [32].

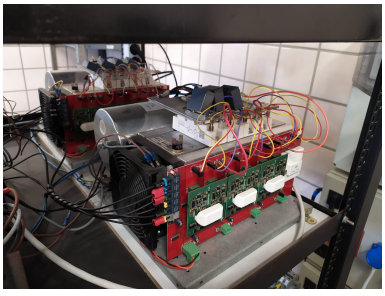

Figura 3.4 Convertidor del Laboratorio.

Figura 3.5 Carga *RL* del Laboratorio.

3.1.3 Sensores

Los sensores usados para medir las intensidades y la tensión de DC son el LA55-P [33] y el LV25-P [34] respectivamente del fabricante LEM. El LA55-P es un sensor de corriente por efecto Hall con un rango de medida de ± 50 amperios, con salida en corriente en el rango de $\pm 50mA$ y una alimentación a $\pm 15V$ Figura 3.6. El LV25-P es un transductor de tensión que por efecto Hall que requiere de una resistencia R1 para limitar la corriente que pasa por él a un máximo de 10 mA Figura 3.7.

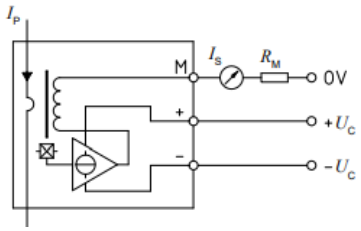

Figura 3.6 Esquema LA55-P.

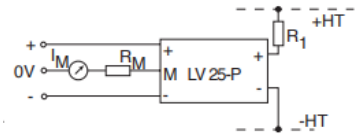

Figura 3.7 Esquema LV25-P.

3.1.4 Placa de disparos

Para enviar los disparos al driver del convertidor se usa fibra óptica por varios motivos. Ofrece inmunidad al ruido de conmutación que pueda llegar desde el convertidor y además protege frente a posibles derivaciones de corriente que podrían llegar al hardware de control o al usuario haciendo su manipulación más segura. La fibra óptica al ser de un material aislante da esta ventaja sobre un cable convencional. Por esto en el laboratorio se trabaja con equipos preparados para recibir este tipo de señales y se dispone de placas de adaptación como la que se ha usado para este proyecto. Esta placa convierte las salidas digitales que salen de la ZedBoard en el rango de tensión de 0 V a 3.3 V a señales luminosas que viajan a través de la fibra hasta el convertidor. Para esto eleva la tensión de 3.3 V a 5 V con un buffer con aislamiento ISO7240 que necesita una alimentación

externa de 5 V adicionalmente a la alimentación de la ZedBoard. Esta placa se ve en la Figura 3.12 de color rojo.

3.1.5 Placa para el acondicionamiento analógico de los sensores

Para acondicionar la señal en corriente proveniente de los sensores LEM, se emplea la placa *GTE M2C-M Analog Interface* disponible en el laboratorio y mostrada en la Figura 3.12 de color blanco. Esta placa dispone de 16 canales para entradas provenientes de los sensores, que se convierten a salidas en el rango de 0 V a 3 V para su posterior conversión. 8 de los 16 canales tienen un offset de 1,5 V y los demás no. Este offset es útil para cuando se trabaja con señales de alterna que pueden ser negativas o positivas dejando los canales sin offset para hacer medidas de continua como la de la tensión del DC-Link. Esta placa se basa en los amplificadores OPA4340 alimentados a 5V, además para conectar los sensores se usan conectores RJ-11 de 6 pines que además de servir para recibir la señal sirven para alimentar los sensores a -15 V y 15 V. De este modo la placa debe de estar alimentada por un lado de a 5 V y por otro a -15 V y a 15 V.

Esta placa recibe las corrientes de salida de los sensores LEM antes descritos y las convierte a tensión por medio de una resistencia. El valor de la resistencia está ajustado para que las corrientes máxima y mínima ($\pm$ 50 mA) se correspondan con las tensiones de entrada de la etapa de acondicionamiento $\pm$ 5 V. Esta etapa puede ser con o sin el offset según si el canal es para medidas de AC o no.

3.2 Hardware de diseño propio

3.2.1 Selección y diseño de placa para ADCs

Adicionalmente a aprender sobre el uso de HDL Coder y su posterior puesta a prueba se propuso encontrar una manera de hacer múltiples medidas simultáneamente para la plataforma ZedBoard. Se planteó como objetivo conseguir hacer medidas de todas las señales en 1 us. Esto a pesar de ser algo sobredimensionado para este proyecto sería útil en el futuro. Tener las medidas más rápido permite tener más tiempo para hacer los cálculos y así poder acortar los tiempos de muestreo y conmutación, algo posible gracias a las nuevas tecnologías en dispositivos de potencia. Estos dispositivos, llamados de "wide band gap" por tener una banda prohibida entre la banda de valencia y de conducción más ancha que los dispositivos de Si, pueden conmutar a mayor frecuencia ya que sus pérdidas crecen más lentamente con la frecuencia [35].

El uso del ADC interno de la Zynq-7000 se descartó por varias razones. La primera es que el Support Package de Xilinx para usar la ZedBoard no incluye herramientas para trabajar con estos ADC desde la interfaz de HDL Coder. La codificación del control de los ADC debería de hacerse desde Vivado, forzando a que haya que trabajar fuera de HDL Coder alejando así el foco del proyecto. Por otro lado, la cantidad de canales disponibles al exterior que el fabricante de la ZedBoard, AVNET, decidió sacar al exterior es reducido e imposibilita su uso sin tener que recurrir a multiplexaciones en el tiempo o tener que trabajar con otros ADCs externos.

Los requisitos de los que se partían a la hora de realizar el diseño eran que debía de tener todas las medidas listas en 1 us y que debería de tener una interfaz compatible con la ZedBoard que se pudiera configurar con HDL Coder. Desde HDL Coder sólo se pueden configurar las entradas digitales que vienen por los Pmods, llamados JA JB JC y JD, lo que deja solo 16 pines, 8 por Pmod, ya que el JA y JB los ocupa la placa de disparos.

Se barajaron múltiples opciones y se llegó a la conclusión de que la mejor estrategia para cumplir con las especificaciones era tener un conjunto de convertidores en paralelo que tuvieran salida serie. De este modo se podrían realizar todas las medidas a la vez y la ZedBoard las recibiría cada una de manera serie por uno de los pines del Pmod. La otra opción hubiera sido tener un solo convertidor que hiciera todas las medidas en muy poco tiempo y que diera tiempo a hacer todas las medidas dentro del tiempo objetivo. Esto se descartó porque requería sacar señales de la FPGA de muy alta frecuencia haciendo más complejo el diseño de la PCB donde se montaría y porque son significativamente más costosos y con encapsulados con los que sería más complicado trabajar en el laboratorio. Además estos convertidores suelen tener salida en paralelo por lo que hay que esperar a que acabe la conversión antes de enviar el dato, esto significa que no solo habría que hacer las medidas de una en una sino que entre una y otra se debe dar tiempo a enviar el dato.

El circuito integrado elegido es el AD7476A de Analog Instruments, un convertidor de 12 bits de resolución, salida serie, 1 MSPS y un encapsulado de 6 pines que permite hacer un diseño muy sencillo que se adapta perfectamente a las especificaciones [36]. Es un convertidor de búsqueda dicotómica que saca los resultados de la comparación a medida que hace la conversión haciendo la extracción de la información más rápida. En

la Figura 3.8 se puede ver el esquema temporal del puerto serie de este ADC. En el esquema se muestran las dos señales de control junto con la señal de salida. Como se puede ver, el flanco de bajada de la señal CS (Chip Select) es el que da el inicio a la conversión, y con cada ciclo del reloj de control (SCLK) a partir del cuarto tras el inicio de la conversión irán apareciendo a la salida los bits del más significativo al menos significativo. El CS deberá estar debidamente sincronizado con el SCLK para captar bien los bits de salida, teniendo en cuenta la capacidad de entrada de 5 nF del AD7476A. El reloj interno con el que funciona el chip es el proporcionado por la entrada de SCLK por lo que su frecuencia determinará el tiempo de conversión, a 50 MHz es cuando se alcanza la tasa de sampleo de 1 MSPS.

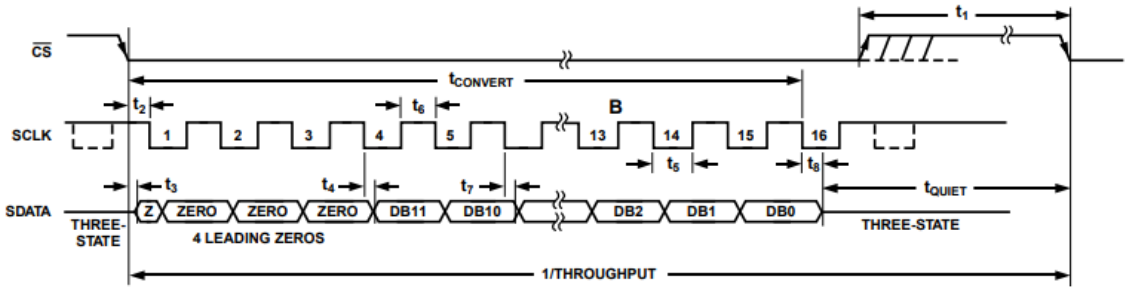

Figure 24. AD7476A Serial Interface Timing Diagram

Figura 3.8 Esquema temporal AD7476A.

La placa se diseñó para tener hasta 14 convertidores funcionando en paralelo controlados simplemente por 2 pines de los Pmod de la ZedBoard. Para capturar las señales digitales que salen de la PCB y para enviar las señales de control se diseñó un periférico que se implementa en la FPGA mediante el uso de HDL Coder. Siguiendo las instrucciones del datasheet se puso cerca del integrado una capacidad de 680 nF entre las pistas de VDD y GND. En la Figura 3.9 se ve el esquemático. Aparecen dos capacidades puesto que el diseño final se pusieron dos por si después era necesario aumentar el valor de la capacidad colocando otra en paralelo. El propósito de estos condensadores es absorber todo el ruido que pueda haber en la alimentación por lo que su funcionamiento no es crítico y conviene que pueda ser adaptable a los resultados. De este modo en el diseño final mostrado en la Figura 3.10 hay dos huecos para capacidades en paralelo.

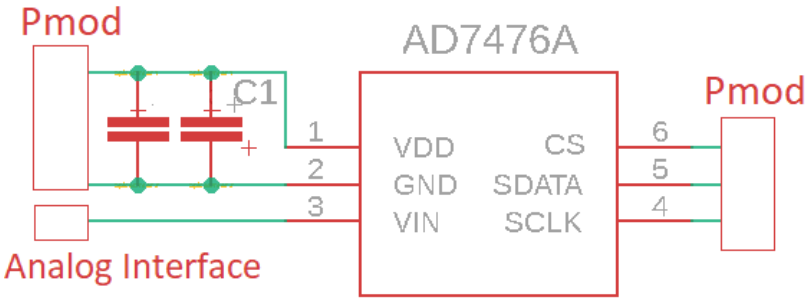

Figura 3.9 Esquemático del conexionado del AD7476A.

Posteriormente, se diseñó una segunda versión con 4 capas y más compacta que se envió a fabricar. Esta versión busca mejorar la las propiedades de la placa frente a ruido electromagnético.

El diseño se hizo procurando que las señales de control llegasen a todos los integrados con las distancias lo más parecidas posibles por lo que se eligieron los dos pines más cercanos al centro de la placa, señalados en la Figura 3.10. De ese modo, aunque es imposible que todas las pistas sean igual de largas se hacen lo más parejas posibles. El diseño final tiene las pistas principales en la capa superior, las dos capas intermedias se dejan para tierra y alimentación para ofrecer el mejor apantallamiento posible, y en la última capa se deja un plano de tierra y las pistas que distribuyen las señales de control ya que no pueden ir por otro lado. El único problema que se detectó en el diseño fue que no existe una adaptación de impedancias entre el pin de CS y todos los integrados por lo que en caso de montar los 14 chips simultáneamente la dinámica de la tensión en la pista de CS no es despreciable. Cada integrado aporta 5 nF de capacidad de entrada por lo que

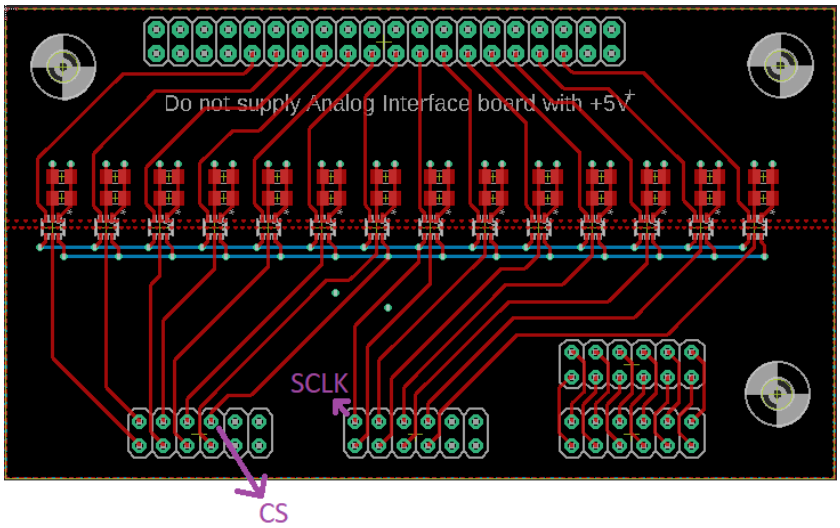

Figura 3.10 Diseño final de la placa en Eagle.

14 son 70 nF, que conjuntamente con una resistencia de 200 mΩ que hay entre el pin de la Zynq-7000 y el Pmod de la ZedBoard para proteger frente a cortocircuitos, crea una constante de tiempo de unos 14 ns que es significativa ante los 50 ns de periodo que puede alcanzar la señal de reloj SCLK. Todo esto es fácilmente solventable dando el flaco de bajada del CS un poco antes.

En la Figura 3.11 se ve con el osciloscopio como funciona el AD7476A. En verde se ve la señal de CS da comienzo a la conversión y como la salida del chip en amarillo pasa de estar en un estado de alta impedancia a un estado bajo. Contando los ciclos del reloj SCLK en azul se puede ver con qué bit se corresponde cada estado alto de la salida del ADC. Tras 16 ciclos de reloj la salida vuelve a un estado de alta impedancia tal y cómo se especificaba en el esquema de tiempo. Abajo en la imagen se ve que el tiempo de conversión cuando el reloj es de 20 MHz es de unos 800 ns ($16 \times 50ns$).

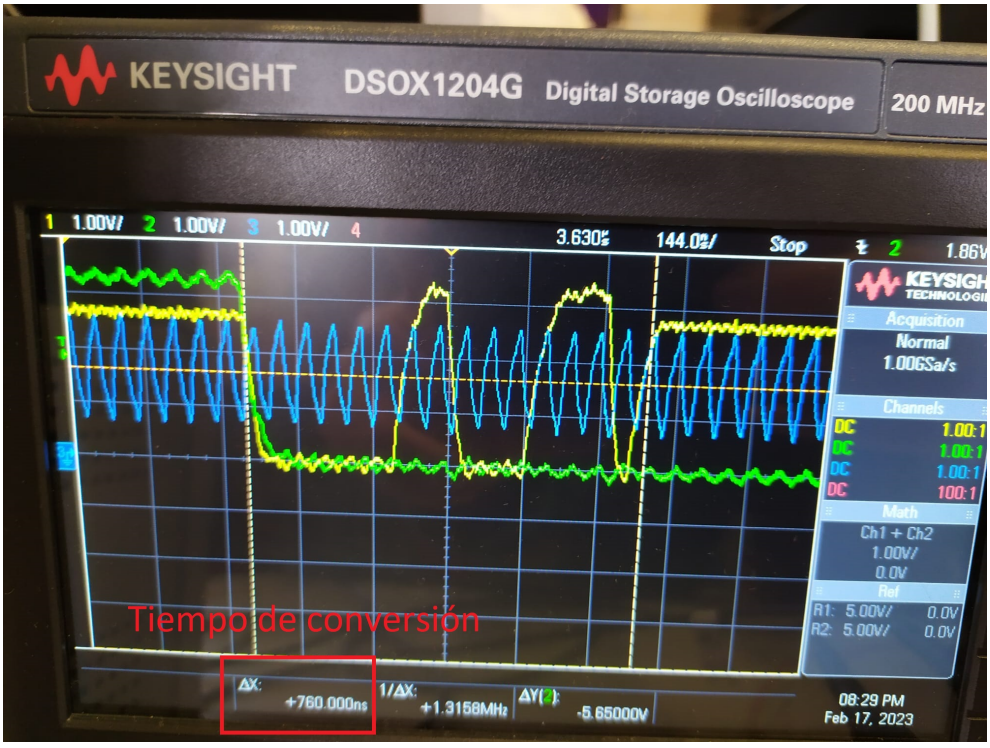

Figura 3.11 Medida de las señales del ADC en el osciloscopio.

3.3 Montaje completo

En la Figura 3.12 muestra una imagen con todas las placas para el desarrollo conectadas. De color blanco a la derecha se encuentra la *GTE M2C-M Analog Interface*, a través de los conectores de tipo RJ11 llegan las señales de los sensores y ambos cables para su alimentación. Esta placa requiere la alimentación de -15/15 V en el conector verde más a la derecha. La siguiente en el camino de la información, de color cobrizo, es el primer prototipo de la placa de ADCs basada en el AD7476A, que no requiere alimención puesto que la toma de 0/3.3V directamente de la ZedBoard. Esta última es la siguiente en aparecer siendo la más grande y en verde en la imagen. Ella contiene la Zynq y es donde se realiza tanto la conversión de la salida serie del ADC como donde se ejecuta todo el control y cualquier programa auxiliar. Además a través de su salida de Ethernet envía información al ordenador de desarrollo para su monitorización. Por último, arriba, en rojo, se encuentra la placa de disparos que requiere una alimentación de 0/5V y es la encargada de convertir los pulsos digitales que saca el control por los Pmods JA y JB de la ZedBoard, en señales luminosas que viajan a través del cable de fibra óptica hasta el convertidor, donde el driver de Semikron ya se encargará de pasarlas a sus correspondientes dispositivos.

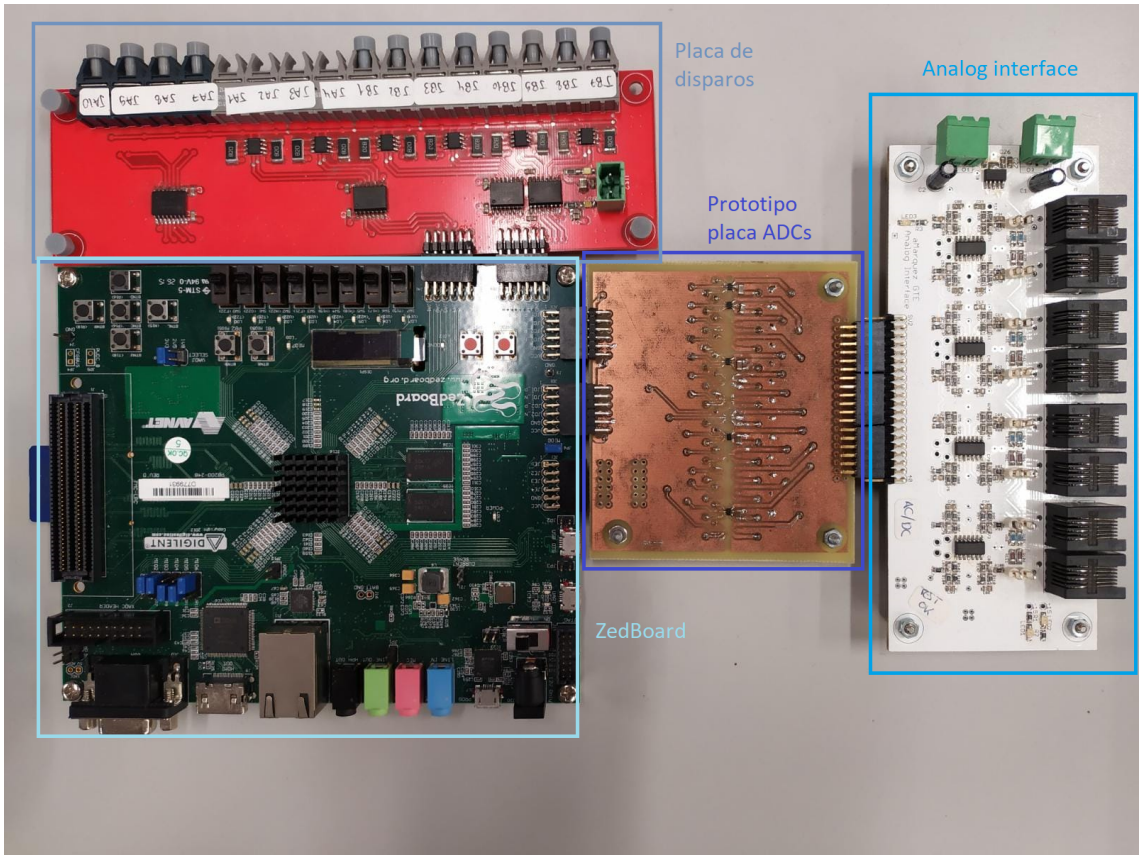

Figura 3.12 Montaje completo de la plataforma.

4 Flujo de trabajo e implementación del control PI

4.1 Instalaciones, versiones compatibles y creación del proyecto

Esta sección se dedicará a introducir los diferentes componentes de software usados para el desarrollo del proyecto, explicar para qué es necesario cada uno, su funcionamiento y cómo configurarlos para poder realizar el proyecto.

La versión de MATLAB instalada es la R2020a y la versión de Vivado 2019.1. La versión de Vivado usada es la System Edition y también se necesita la aplicación de System Generator que viene selecionada por defecto al hacer la instalación de Vivado. La selección de la versión de Vivado y de Matlab se ha hecho para garantizar la compatibilidad puesto que se requerirá de ambos programas funcionando conjuntamente para poder realizar el proceso de desarrollo completo [37]. Hay versiones más modernas de Vivado y MATLAB compatibles pero a 1 de Enero de 2022 Las versiones más modernas de Vivado dejaron de ser compatibles por lo que a fecha de inicio de este proyecto se optaron por las dos versiones dichas anteriormente [38]. Aunque se hayan elegido estas versiones de MATLAB y Vivado por su compatibiliad, la versión de System Generator no soporta oficialmente la versión de MATLAB pero modificando un archivo .xml se puede incluir. Para esto se siguen los pasos de la siguiente entrada del soporte de Matlab [39]. System Generator es una aplicación del conjunto de aplicaciones de Vivado con la cuál se usará para abrir MATLAB a partir de ahora. De esta manera, la configuración que necesita MATLAB para poder hacer llamadas a las aplicaciones de Vivado vendrá ya hecha, de lo contrario se deberá realizar manualmente introduciendo el path donde se tenga instalado Vivado con las función "hdlsetuptoolpath()" proporcionada por HDL Coder.

En la siguiente lista (Figura 4.1) se exponen los add-ons de MatLab (incluyendo HDL Coder) que se usaron para el desarrollo del proyecto y que son requeridos de alguna manera por HDL Coder.

- *HDL Coder, Embedded Coder, Matlab Coder y MinGw-64w C compiler* son los Add-ons dedicados a convertir código Matlab en código VHDL, código C, soportar esta generación de código y compilarlo. Además son los que ofrecen las herramientas para configurar y monitorizar el hardware y las herramientas enfocadas a asegurar la compatibilidad de los diseños en Simulink con la generación de código.
- *Simulink y Simulink Coder* son los necesarios para crear y generar código a partir de los esquemas de Simulink.
- *HDL Coder Support Package for Xilinx Zynq-7000 Platform, Embedded Coder Support Package for Xilinx Zynq-7000 Platform, Embedded Coder Interface to QEMU Emulator, Embedded Coder Support Package for Cortex-A ARM Processors* son todo paquetes para poder configurar opciones del hardware de Xilinx desde la interfaz de MATLAB/Simulink (por ejemplo, continene los ficheros necesarios para crear los ficheros de restricciones de la FPGA). Funcionan como librerías para las placas de desarrollo basadas en la Zynq-7000. Si se deseara programar una placa distinta existen paquetes de otros proveedores o la opción de configurar placas propias.
- *Fixed-Point Designer* es un Add-on que añade la posibilidad de crear variables con aritmética de punto fijo en MATLAB y añade la posibilidad de configurar este tipo de variables en multitud de bloques de

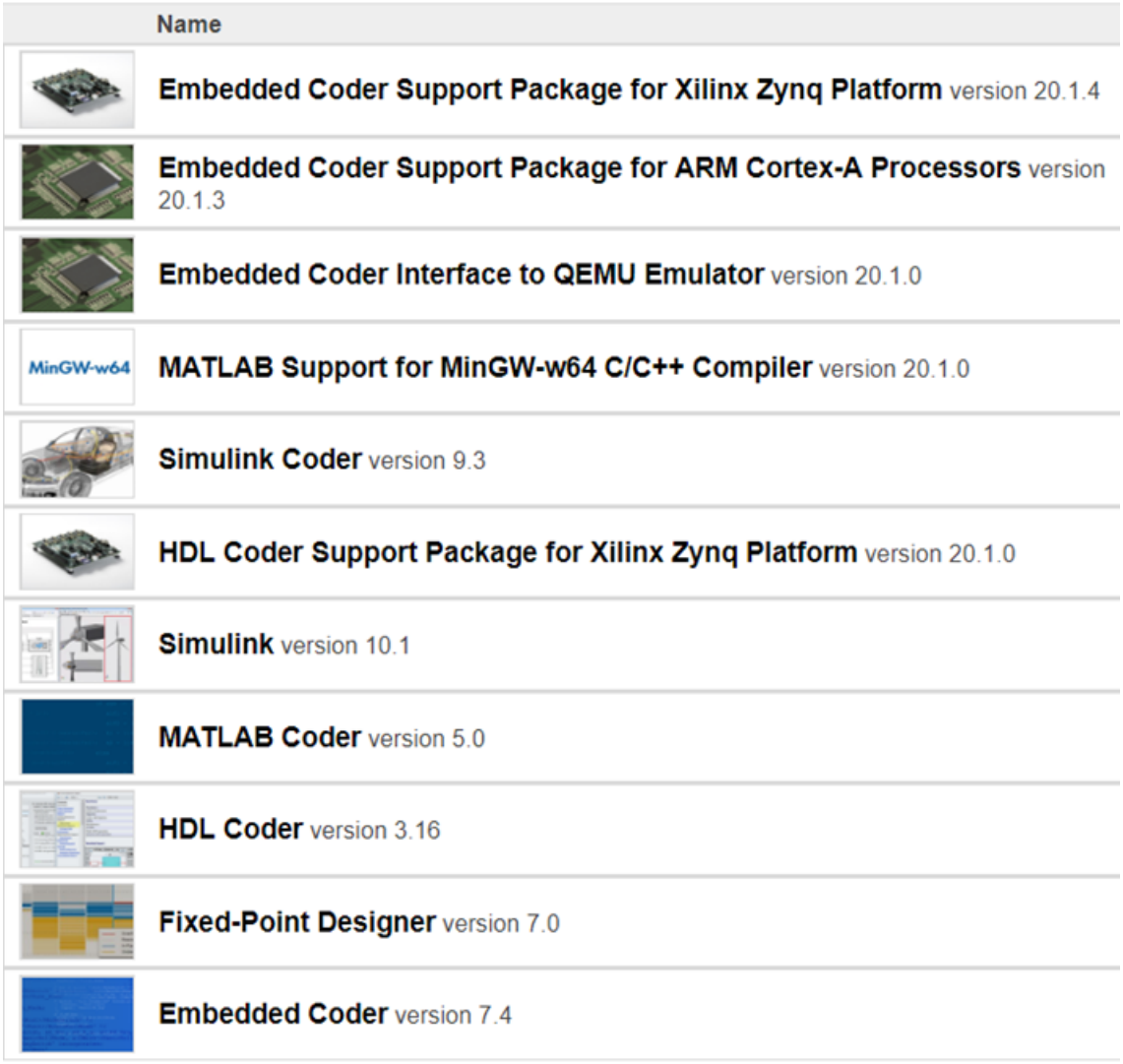

Figura 4.1 Lista de Add-ons requeridos para trabajar con HDL Coder.

Simulink. Además este add-on ofrece herramientas que permiten el análisis de rangos de las variables en punto fijo para hacer un mejor ajuste del número de bits y de la posición de la coma binaria, pero esto queda fuera del alcance de este proyecto.

Adicionalmente a la lista de software estrictamente requerido para trabajar con la ZedBoard desde MATLAB, para realizar simulaciones del sistema de electrónica de potencia se ha usado el toolbox de Simscape Electrical. Este componente permite verificar los algoritmos de control antes de probarlos en el convertidor real, teniendo la precaución de tener en cuenta posibles retrasos añadidos en algunos cálculos que no aparecen en la simulación de Simulink.

El *Embedded Coder Support Package for Xilinx Zynq-7000 Platform* requiere una configuración adicional que se hace mediante un wizard muy simple al que se accede desde la ventana principal de MATLAB *Manage Add-ons -> Embedded Coder Support Package for for Xilinx Zynq-7000 Platform -> Setup*. Este wizard proporciona los archivos que deberá llevar la tarjeta SD de la ZedBoard. Entre estos archivos se incluye el sistema operativo que ejecuta uno de los núcleos de ARM de la Zynq y configuración relativa a la comunicación entre el ordenador y la placa. En este proyecto se optó por configurar la comunicación para que fuese a través de un cable de ethernet, que conecta la ZedBoard y el PC de desarrollo, por simplicidad. Esta vía de comunicación permite al usuario ejecutar un archivo de Simulink que se generará de manera automática y a partir del cuál se podrán cambiar parámetros del controlador en linea y monitorizar las distintas variables configuradas como salida al ordenador.

HDL Coder requiere ciertas configuraciones para funcionar pero la mayoría aparecen durante el flujo de trabajo como errores y da la posibilidad al usuario de configurarlas en el momento por lo que se tratarán más adelante junto con el flujo de trabajo. La manera de trabajar con HDL Coder será con modelos de Simulink ya que es el método que permite hacer un diseño completo desde el software de Mathworks. De este modo, el primer paso para crear un proyecto es el mismo que para crear un proyecto en Simulink. Desde la opción de MALAB de Simulink, el usuario puede abrir un proyecto en blanco. En la pestaña de "Apps", dentro de Simulink, con el software anteriormente descrito instalado, aparece la opción de "HDL Coder", que pulsándola abre la interfaz para empezar a trabajar en el proyecto Figura 4.2. Trabajando de esta manera, se debe de elegir un subsistema del modelo de Simulink que será objetivo de la generación de código de HDL Coder. Todo lo que esté dentro de dicho subsistema será traducido a código VHDL o Verilog según se configure (VHDL en este proyecto).

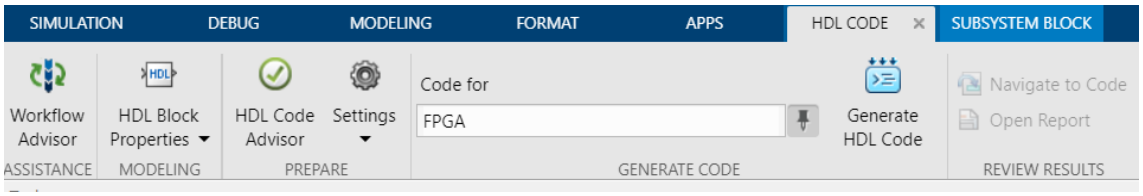

Figura 4.2 Ejemplo de configuración de HDL Coder para el subsistema FPGA.

4.2 Introducción al flujo de trabajo

Una vez se tenga la herramienta en funcionamiento, el primer paso será elaborar una metodología para trabajar con HDL Coder. El flujo de trabajo se aplicará en el diseño del control PI y del FCS-MPC. En esta sección se explicará la implementación del control PI conjuntamente con el flujo de trabajo propuesto para usarlo a modo de ejemplo. HDL Coder posee múltiples opciones para generar código a partir de diferentes elementos del entorno de MATLAB/Simulink, por lo que para tener una metodología eficiente, es clave encontrar la combinación de elementos apropiada. Esta metodología se basa en un modelo de Simulink que combina bloques propios de Simulink, con bloques de MATLAB Function con código, y con charts de Stateflow. Se opta por esta opción para así poder poner a prueba las diferentes capacidades de HDL Coder para generar código VHDL y para tratar de extraer lo mejor de cada funcionalidad de Simulink. Además, trabajar desde Simulink permite usar la herramienta "Workflow Advisor" que proporciona HDL Coder para hacer todos los pasos entorno a la generación de código de manera guiada una vez se tenga un diseño funcional. Este proceso se puede hacer también por línea de comandos pero resulta mucho más cómodo y ordenado seguir la herramienta de Simulink. Algunos métodos alternativos podrían ser hacer todo el diseño en código MATLAB y generar VHDL con los comandos de HDL Coder, usar exclusivamente elementos de Simulink y generar VHDL con el "Workflow Advisor" o generar fragmentos de código y juntarlos luego usando Vivado. Algunas de estas alternativas se probaron y se descartaron por presentar desventajas en cuanto a depuración de errores, velocidad de diseño o por requerir hacer código directamente en VHDL.

4.3 Esquema de tiempos

El primer paso para diseñar deberá ser plantear un esquema de tiempos para el circuito a implementar. Simulink ofrece la posibilidad de que distintas partes del esquema tengan distintos tiempos de actualización. La configuración de esta parte es con el parámetro "Sample time" de los bloques. Esto es útil para poder coordinar la parte de cálculo del control con los distintos periféricos que se requieran para controlar dispositivos externos que no necesariamente funcionen a la misma frecuencia. El código de VHDL que crea la herramienta tendrá una única señal de reloj. Al configurar distintas partes del esquema, con distintas frecuencias, hará que de manera automática se creen las señales de enable necesarias para que el circuito funcione como se espera. Por esta razón, la frecuencia de reloj elegida debe ser un submúltiplo entero de todas las frecuencias de sampleo del circuito, siendo esta una restricción a tener en cuenta desde el primer momento de la fase de diseño. Para facilitar este proceso es recomendable tener activado el "sample time display", para activarlo en el esquema de Simulink hacer (clic derecho sobre el fondo -> Sample time display -> colors) de esta manera aparecerá representado con colores el sample time de cada señal en el esquema de Simulink (ver Figura 4.3).

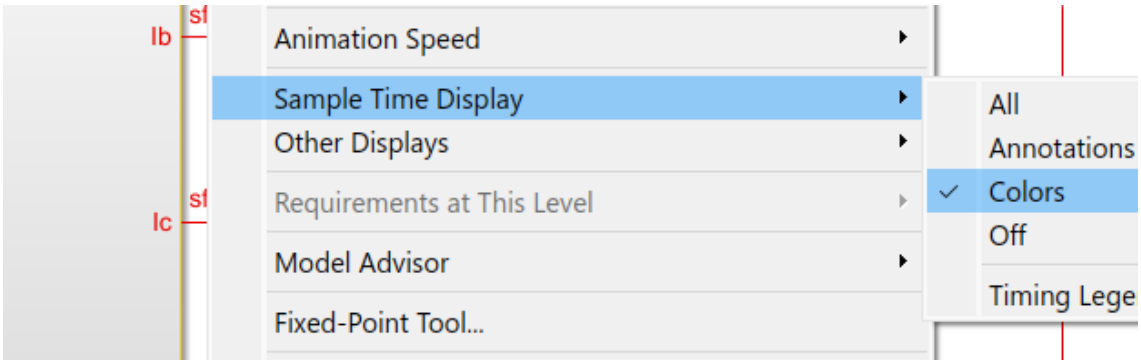

Figura 4.3 Configuración para mostrar con colores el sample time.

Este primer paso es previo a empezar a poner bloques en Simulink. Es necesario saber de antemano los tiempos de actualización que deberán tener las diferentes partes del circuito objetivo puesto que una mala construcción del mismo llevará a errores en etapas posteriores. Además los mensajes de error acerca de incoherencias en los tiempos de muestreo pueden ser un poco engañosos y proponer soluciones que solo llevan a más errores. Es importante entender que el "sample time" de una señal no significa que la señal cambie cada ese tiempo, sino que puede cambiar en un conjunto de instantes discretos separados por ese periodo en cuestión.

Comenzando con el ejemplo del control PI, es preciso determinar los diferentes subsistemas que lo compondrán para decidir cómo se estructura su esquema de tiempos. El diseño del control consta de un periférico para gestionar la comunicación con la placa de ADCs, un bloque de cálculo donde realizar todas las operaciones asociadas al control de corriente, un modulador de tipo PWM que gestione los disparos y una máquina de estados que dirija el funcionamiento del control.

Para el reloj principal del sistema se elige un reloj de 100 MHz. Esta frecuencia de reloj es de un valor alto en relación a otros diseños hechos en programas de HLS presentes en la bibliografía. La motivación tras esta decisión es el objetivo de probar los límites de esta herramienta y de hacer diseños que se puedan acercar más a diseños hechos "manualmente" con estrategias de más bajo nivel. 100 MHz es una frecuencia típica en el diseño con VHDL. La ZedBoard viene con un reloj dedicado a esa frecuencia y todos los periféricos son compatibles con ella. A esta frecuencia están configurados los tiempos de actualización del bloque de cálculo, el modulador PWM y la máquina de estados global. El bloque de cálculo debe de funcionar a esta frecuencia para realizar los cálculos lo más rápido posible, puesto que esta es la mayor frecuencia del sistema. El modulador PWM necesita realizar comparaciones a esta frecuencia, la explicación de esto se hará más adelante. La máquina de estados global debe de poder responder rápido ante un error por lo que no hay una ventaja en reducir su frecuencia.

El periférico de comunicación con la placa de ADCs funcionará con un tiempo de actualización mayor (30 ns) puesto que su funcionamiento viene determinado por el esquema de tiempos del AD7476A (Figura 3.8). Dentro de las librerías de HDL Coder no existe un bloque que permita generar una señal de reloj, ni ningún menú que permita asociar un reloj a alguno de los pines. La creación de relojes queda reservada al generador de código. Por esta razón la alternativa propuesta para generar la señal SCLK es un contador de 1 bit, que cada 30 ns, cambia la salida de 1 a 0 o viceversa. El periodo de la señal de reloj generada queda de 60 ns, aunque el periodo mínimo especificado por el datasheet es de 50 ns para la conversión más rápida. Este periodo es imposible de alcanzar en este diseño puesto que requeriría que el sample time del contador sea de 25 ns, algo imposible al no ser múltiplo entero del sample time del reloj del sistema 10 ns. En resumen se elige un sample time de 30 ns porque es el más grande que permite hacer la conversión lo más rápido posible.

Por último, existe una limitación de la frecuencia a la que los datos pueden ser enviados o recibidos por el ordenador de manera fiable, por lo que en base a las pruebas del artículo [40], se han elegido 200 μs como sample time de la transmisión de datos para la monitorización. Esto significa que el dato se enviará al ordenador una vez cada 200 μs. Será responsabilidad del diseñador que el dato sea correcto cuando se realice la transmisión.

En la Figura 4.4 se muestra el diagrama del diseño del PI y marcados con colores los distintos sample time. En rojo los 10 ns del reloj del sistema, en verde 30 ns y en azul 200 μs.

Adelantando contenido de la siguiente sección, cuando se conectan partes del circuito con distintos tiempos de actualización en Simulink se deben usar los bloques de "Rate transition block". Cuando el paso es de una

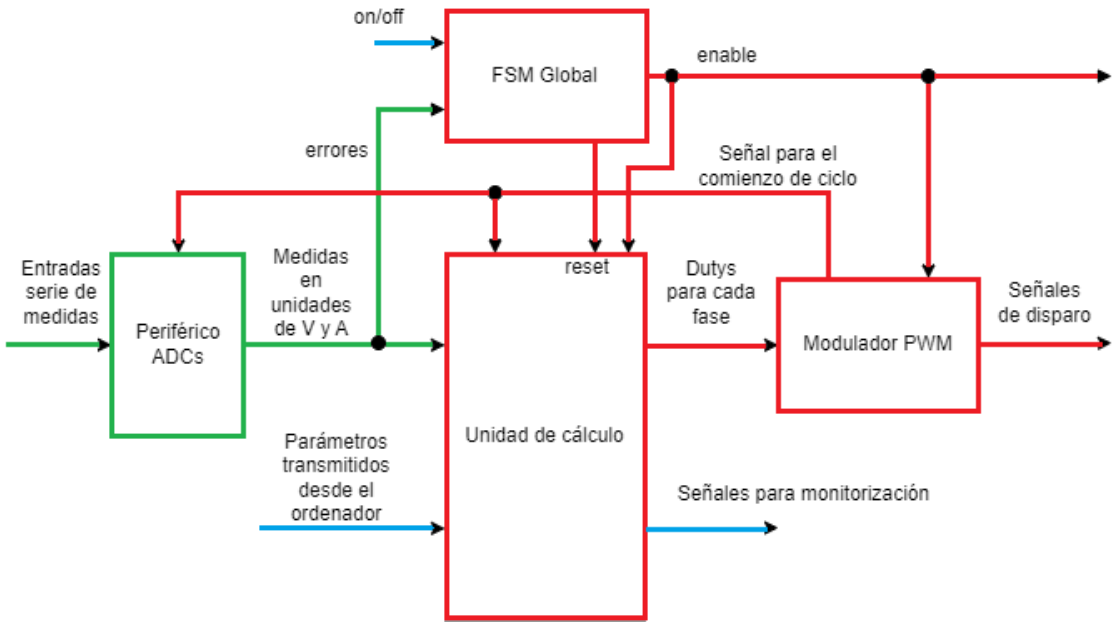

Figura 4.4 Diagrama del diseño para el PI.

frecuencia baja a una alta, este bloque acaba siendo un registro que hace un sobremuestreo, mientras que cuando el paso es de alta a baja frecuencia es un submuestreo.

4.4 Creación del esquema de Simulink

A continuación se crea el esquema de Simulink siguiendo el esquema de tiempo definido y pudiendo probar por simulación el comportamiento del algoritmo. Simulink ofrece una gran cantidad de alternativas para la implementación de un cierto algoritmo. La filosofía de diseño que se ha seguido en este trabajo ha sido usar:

- **Bloques de la librería básica de Simulink:** Para realizar rutado de señales (Mux, Demux y signal switches), para garantizar la integridad temporal de la información (Data Rate Transition, Delays), operaciones matemáticas básicas puntuales donde no merezca la pena implementar en bloques de código (Gain, Sum, product) y operaciones de lógica binaria. El entorno de Simulink sirve principalmente como un marco en el que gestionar el flujo de información de un bloque a otro de código y para asegurar la correcta temporización del circuito. Hacer estas tareas en un entorno visual simplifica el proceso, le aporta claridad y además, la división en subsistemas del circuito hace que la generación de código se fragmente en varios archivos y que luego la depuración de errores sea más sencilla.
- **Bloques "MATLAB Function":** Para implementar algoritmos matemáticos complejos. Hacer un gran número de operaciones con matrices con bloques de Simulink requeriría una inversión de tiempo grande al ser una tarea ciertamente tediosa debido a la gran cantidad de bloques en paralelo que habría que introducir. El lenguaje de MATLAB es un lenguaje preparado para trabajar con matrices de manera sencilla y un gran número de las funciones básicas de MATLAB son compatibles con la generación de código VHDL por lo que resulta lo más conveniente a pesar de las limitaciones que se tratarán en capítulos posteriores. Como con la división en subsistemas es recomendable no cargar todos los cálculos en un único bloque de código sino dividir el cálculo en varias funciones diferentes por razones de depuración.
- **Charts de Stateflow:** Para crear máquinas de estados. El diseño de FSM en VHDL pese a no tener que ser especialmente complejo, sí puede ser tedioso por tener que definir todos los casos posibles, separar correctamente la parte combinacional de la secuencial, etc. Con Stateflow el diseño de máquinas de estados se hace desde una interfaz gráfica, con una tabla para gestionar las variables y opciones para ajustar la temporización. Por todas estas razones y por la robustez a la hora de generar código se elige este método para el proyecto. A parte de que probar las distintas posibilidades es parte de los objetivos del proyecto.

- **Bloques de la librería de HDL Coder para Simulink:** Para implementar ciertos componentes propios de circuitos digitales el Add-on de HDL Coder añade una librería específica con objetivo de simplificar el proceso de diseño. Buena parte de los bloques de esta librería provienen directamente de los básicos de Simulink, para los que la herramienta asegura su compatibilidad. Además, se incluyen una variedad de bloques especialmente útiles y con configuraciones propias de diseño digital. Entre estos destaca el bloque HDL Counter, implementa un contador y tiene múltiples opciones de configuración como un reset, un límite para la cuenta, entrada de dirección etc. En definitiva son bloques que funcionan como los del IP Catalog de Vivado [41].

Este paso es el que contiene la mayor carga de trabajo puesto que es donde se hace el diseño en sí. El ejemplo del control PI es por tanto bastante extenso y se divide en los diseños de los cuatro bloques que lo componen (ver Figura 4.4). Además se hacen comentarios sobre cómo usar HDL Coder pero de manera específica para controladores de electrónica de potencia.

4.4.1 Periférico de control y comunicación con los ADCs

El diseño del periférico debe de cumplir con las siguientes funciones: generar el SCLK, generar la señal de CS sincronizada con el SCLK al comienzo de cada ciclo, capturar la salida serie del AD7476A, reconstruir el dato, mantenerlo estable a la salida hasta la próxima medida e interpretar el dato. En la Figura 4.5 se muestra el esquema con los bloques que implementan cada una de las funciones y su conexionado. El bloque azul (FSM) implementa la captación de la entrada serie, la reconstrucción y mantener la salida estable mediante una máquina de estados. El resto realizan la función que les da nombre.

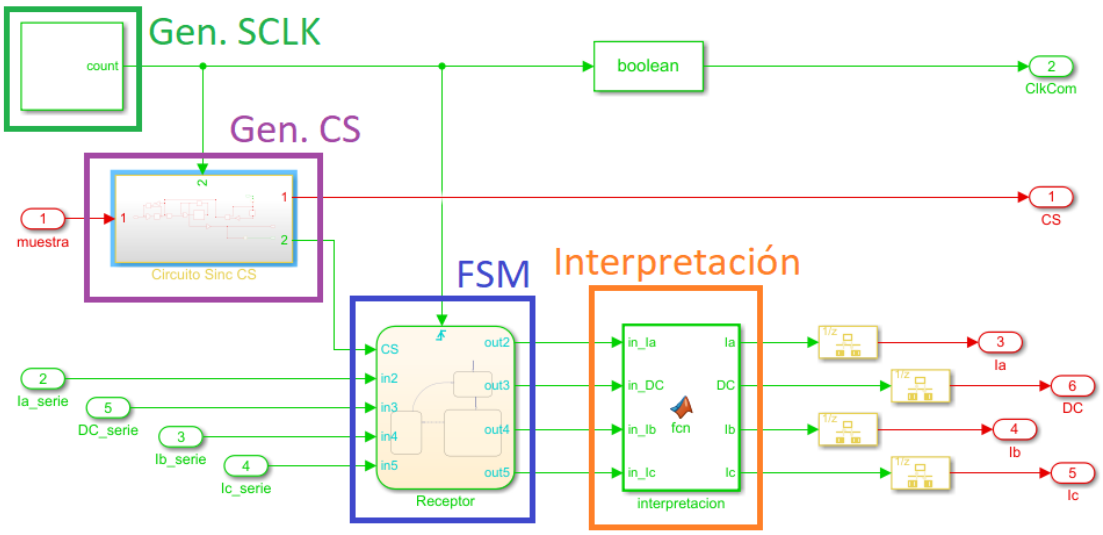

Figura 4.5 Esquema Periférico para ADCs.

Generación SCLK

SCLK es una señal de reloj, cuya generación va íntimamede ligada al esquema de tiempos del sistema. Por esta razón su generación se trató previamente en el apartado del esquema de tiempos. La implementación, como se ve en la Figura 4.5 es simplemente un bloque de HDL Counter configurado con un bit y el sample time del subsistema (30 ns), creando una señal de reloj de 60 ns de periodo.

Generación CS

La medida se inicia con el flanco de bajada del CS pero este no puede producirse inmediatamente tras la señal que se produce al inicio de cada periodo. El flanco de bajada debe aparecer en un momento concreto del ciclo del SCLK para garantizar una captura de la información correcta y de manera robusta. Como se muestra en la Figura 3.8 debe de darse más de 10 ns antes del flanco de SCLK. Esta sincronización se logra con un circuito digital simple montado a partir de bloques de "delay" a modo de biestables y con puertas lógicas. Tras esto un contador mantiene a 0 la señal CS durante 20 periodos y finalmente sube de nuevo a 1 para esperar al próximo ciclo. En la Figura 4.6 se muestra el circuito. En azul un detector de flanco de subida

de SCLK. En verde, un circuito que recibe y almacena la orden de realizar la medida en un biestable hasta el flanco de SCLK, tras esto envía una señal que activa el conjunto marcado en morado y resetea el biestable hasta el próximo ciclo. Este es un circuito que mantiene el CS a 0 el tiempo requerido por el esquema de tiempos del AD7476A mediante un contador. Este está configurado con un enable que lo mantiene parado mientras el valor del contador sea 0 y hasta que recibe la señal del grupo anterior. A partir de ese momento la salida del contador deja de ser cero y el comparador que realimenta al enable a través de la puerta OR mantiene el contador en marcha hasta que desborda y vuelve su salida a 0.

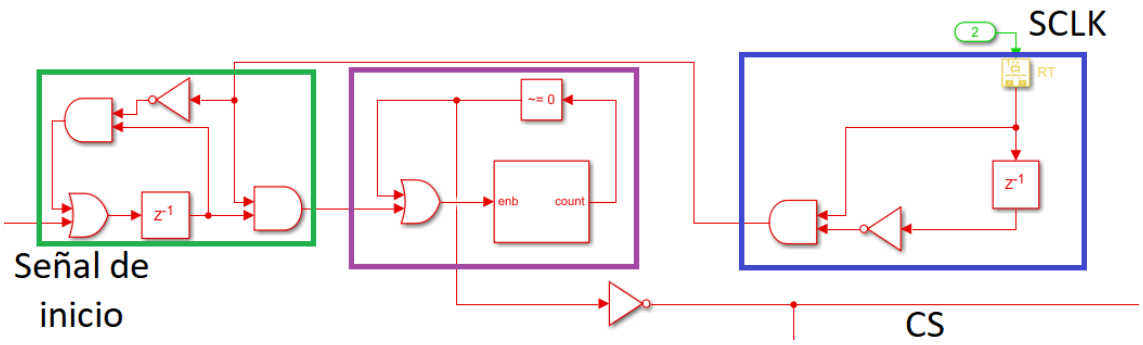

Figura 4.6 Circuito para la sincronización y generación de CS.

Captura de bit y reconstrucción

Con las SCLK y CS generadas es posible crear una máquina de estados que haga el proceso inverso al esquema de tiempos del AD7476A. Mediante un chart se implementa un estado de reposo, un estado en el que va almacenando y desplazando los bits entrantes en un registro de desplazamiento para cada canal, y un estado en el que actualiza el registro de salida. El chart está configurado para ejecutarse con cada flanco de subida del SCLK. El estado que captura los bits del protocolo serie tiene un contador que hace repetir el estado 16 veces antes de pasar al siguiente. La captura se hace en todos los canales en paralelo lo que hace el diseño escalable y simplemente añadiendo más registros se pueden añadir los canales necesarios. Los registros de desplazamiento se implementan mediante la una instrucción de MatLab que permite el desplazamiento de bits y otra que asigna el bit al primer biestable del registro. Los registros son de 16 bit aunque la resolución sea de 12 por simplicidad ya que por el esquema de tiempos se sabe que los 4 más significativos son 0. En la figura Figura 4.7 se ve la implementación con el chart, ahí aparece el estado "espera" que resetea los registros de desplazamiento, el estado "Lee" al que se pasa cuando se da el flanco de bajada de CS y que captura los bits de la trama serie, y por último el estado "saca valores" al que se pasa cuando ha acabado la conversión y que pone en el registro de salida los datos capturados. Con la subida de CS se vuelve al estado "espera". Se configuran 4 canales porque se toman 4 medidas simultáneamente, las 3 corrientes y la tensión del DC-Link.

Función de interpretación y calibración

Tras la obtención de la conversión de las señales de medida analógicas a binario de 12 bits es necesario realizar unos cálculos para recuperar la información en unidades de tensión o intensidad. Para definir esta función de interpretación se hace una calibración con toda la cadena de hardware para hacer las medidas, sensor, placa de acondicionamiento y ADCs. Para calibrar el equipo se introducen intensidades conocidas a través del sensor y se anotan los valores en binario resultantes creando una tabla. Repitiendo con el sensor de tensión del DC-Link se obtiene otra tabla. Con la función "polyfit()" de MATLAB se obtienen los coeficientes de un polinomio del grado deseado que más se aproxime al conjunto de puntos especificado. Con grado uno y los puntos de las tablas con las medidas tomadas se obtienen los coeficientes de una recta de ajuste del tipo $y = n + xm$. La salida en binario de la cadena de instrumentación, será el valor de x de la función y el valor real será el de la y resultante. La función de interpretación en el modelo de Simulink se implementa a través de un bloque MATLAB Function. Para este proyecto se ha usado la OPAL-4510 para depurar el control y el convertidor real para obtener resultados, por lo que se repite todo el proceso para ambos equipos.

En las Figura 4.8 y Figura 4.9 se pueden ver una gráficas comparativas con los datos de la calibración del equipo real y de la OPAL respectivamente frente a sus rectas de ajuste. Se aprecia un comportamiento muy lineal sobretodo en los sensores del equipo real. Para que se aprecie mejor en las rectas de las intensidades a la derecha se ha añadido un zoom, donde se observa que la recta se ajusta mejor en el equipo real que en la OPAL.

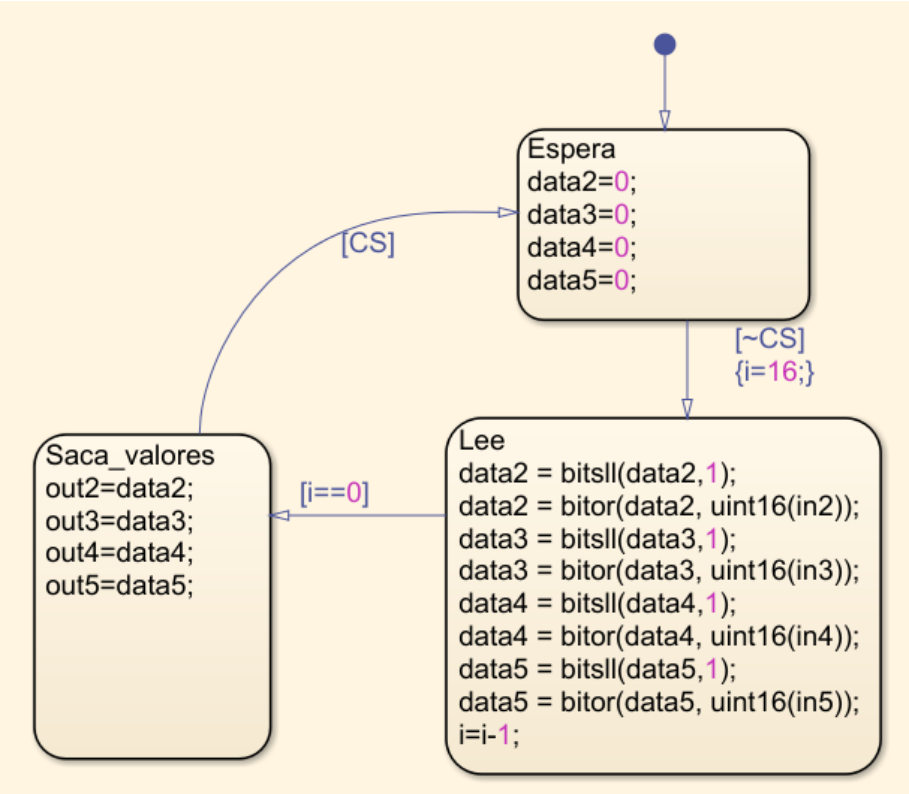

Figura 4.7 Configuración del chart para la máquina de estados del periférico. Las variables "in" son las entradas booleanas que vienen del ADC, las variables "data" son los registros de desplazamiento y las variables "out" son los registros de salida.

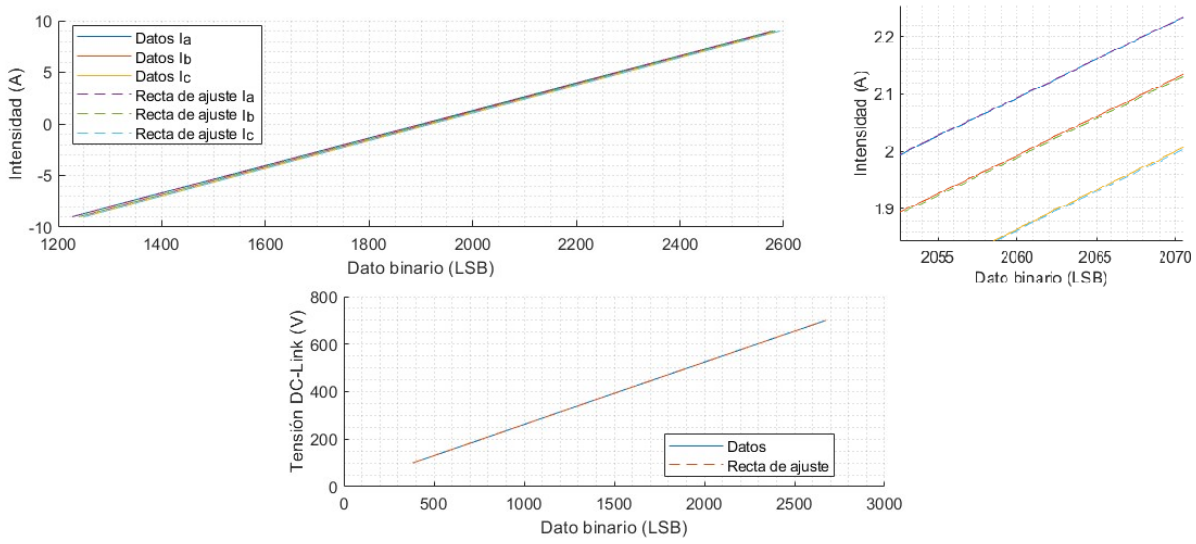

Figura 4.8 Calibración equipo real y rectas de ajuste.

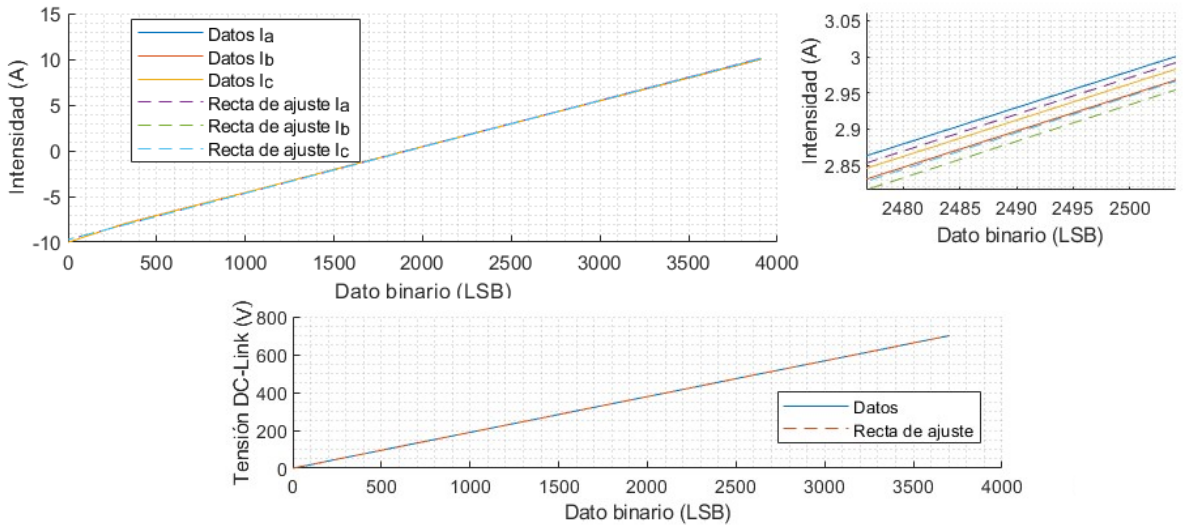

Figura 4.9 Calibración de la OPAL y rectas de ajuste.

4.4.2 Implementación del algoritmo

El algoritmo de control debe realizar las siguientes tareas a partir de los datos de las medidas:

- Generar una referencia de fase: puesto que el equipo va desconectado de la red, el controlador no requiere un PLL pero si un mecanismo con el que generar un ángulo θ que varíe acorde con la frecuencia deseada para las corrientes (50 Hz).
- Realizar las transformaciones de Clarke y de Park sobre el vector I_{abc} para trabajar en los ejes de coordenadas dq.
- Calcular mediante un PI en cada periodo de muestreo la actuación necesaria para el siguiente instante.
- Anti-transformar la variable de actuación V_{dq}^{ref} para pasarla de nuevo a ejes abc.
- Normalizar la variable V_{abc}^{ref} para obtener un duty que proporcionar al modulador.
- Asegurar que al inicio de cada periodo se carga al modulador el duty nuevo y que se mantiene estable durante todo el mismo.

Además de abordar todas estas tareas, el diseño busca paralelizar lo máximo posible los cálculos reduciendo el tiempo de cálculo para aprovechar la potencialidad de la FPGA.

Generación del ángulo de sincronización θ y la Matriz de Park

La generación de un ángulo θ y la matriz de Park se realiza mediante un contador, dos Look-up tables, una para el seno y otra para el coseno y un bloque de Matlab function para crear la matriz de manera cómoda. El contador está configurado con un sample time de 100 μs. El contador cuenta de manera ascendente de 0 a 199 completando el ciclo en 20 ms. Estos 200 puntos son el número de ciclos de la frecuencia de conmutación deseada (10kHz) que hay en un periodo de las señal de 50 Hz. De este modo, aunque no se crea un θ explícitamente, se crea una señal de tipo entero que cada una de sus posibles valores representa un valor de θ y se corresponde con un punto en las tablas de seno y coseno. La tabla se configura especificando los puntos contenidos y las entradas asociadas. En la Figura 4.10 se muestra como configurarla usando la función coseno y un vector con todos los posibles valores de θ.

Dentro de la librería de HDL coder existe un bloque específico para generar tablas para señales sinusoidales aprovechando la simetría de cuarto de onda pero, en este proyecto, el estrecho margen de tiempo que da el reloj del sistema hace imposible su utilización. El bloque en cuestión funciona comprobando en qué cuadrante se encuentra el ángulo, le resta el múltiplo de $\pi/2$ necesario para dejarlo en el primero y si estaba en en el segundo o el cuarto toma el complementario. Después entra a una tabla del cuarto de onda y toma la salida de signo positvo o negativo en función del cuadrante en el que estuviese inicialmente el ángulo. Probablemente HDL Coder implemente el VHDL de este código directamente a partir de un código predefinido mediante generics. Por esta razón el algoritmo del "Adaptative Pipeline", descrito en el siguiente apartado, no puede

entrar a colocar un registro entre todas estas operaciones que permita que se haga todo el proceso en varios ciclos de reloj distintos y que falle el build al intentar hacerlo todo en 10 ns.

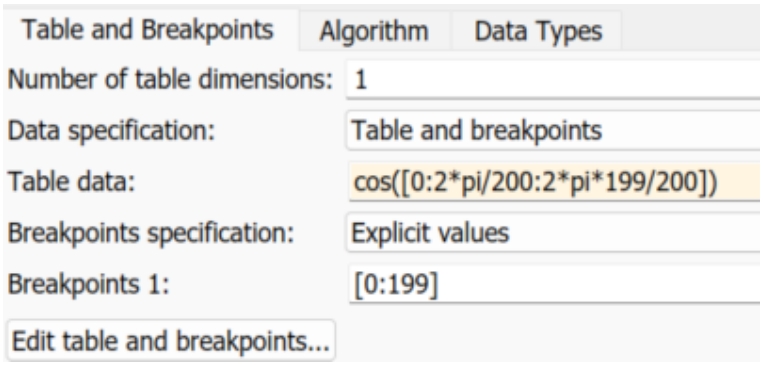

Figura 4.10 Configuración de la LUT para el coseno.

En la Figura 4.11 se observa la primera parte del subsistema de cálculo, la transformación a ejes dq. El contador de abajo a la izquierda es el mencionado en este subapartado y el bloque inmediatamente posterior el que contiene las LUTs. Es necesario un bloque de rate transition para pasar del sample time de 100 μs de contador para generar θ al de 10 ns del subsistema de cálculo. La matriz de Clarke se obtiene de un bloque "constant".

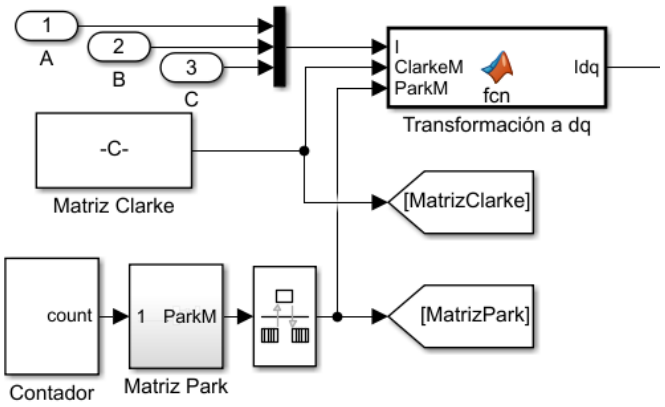

Figura 4.11 Esquema para la transformación de la intensidad medida a ejes dq y la generación del ángulo θ.

Transformaciones, anti-transformaciones e implementación del PI

Como se comentó al comienzo de esta sección, para realizar cálculos complejos se usan bloques de MATLAB function, por lo que se elige esta opción para realizar las transformaciones de los ejes, el cálculo del PI y las anti-transformaciones. Como excepción a esta regla, el proceso de normalizado de las tensiones de referencia se ha realizado por bloques por tener una parte hecha como look-up table.

Estos bloques de código se pueden programar de manera convencional, aunque conviene tener en cuenta que el objetivo final es un circuito digital.

La multiplicación de matrices con código MATLAB se hace mucho más apropiada que crear bloque a bloque los productos y sumas implicados en estas operaciones. Del mismo modo en este lenguaje existen multitud de funciones compatibles con la generación de código que justifican estas decisiones en el flujo de trabajo. Sin embargo el uso de bloques de código en el diseño tiene una desventaja en cómo aplica HDL Coder los algoritmos de optimización. Para la generación de código a partir de un esquema en Simulink, HDL Coder genera un archivo de Simulink intermedio donde aplica las optimizaciones que estime convenientes. Entre las optimizaciones más útiles destacan:

- "AdaptativePipelinig", que incluye registros, usando bloques de "unit delay", entre operaciones para así facilitar que se cumplen las restricciones de tiempo a la hora de hacer el build con Vivado (ver

Figura 4.12 a).

- "DistributedPipelining", que con el mismo objetivo que la opción anterior, distribuye los registros que se hayan colocado manualmente a lo largo del camino que recorre la información (ver Figura 4.12 b).
- "InputPipeline", que permite la colocación de varios registros a la entrada de un subsistema o bloque de código para que puedan ser distribuidos después por el "DistributedPipelining".
- "ConstrainedOutputPipeline", que permite crear un número fijo de registros a la salida del subsistema o bloque de código, no afectado por "DistributedPipelining", para garantizar que la salida ha pasado por cierto número de retrasos.

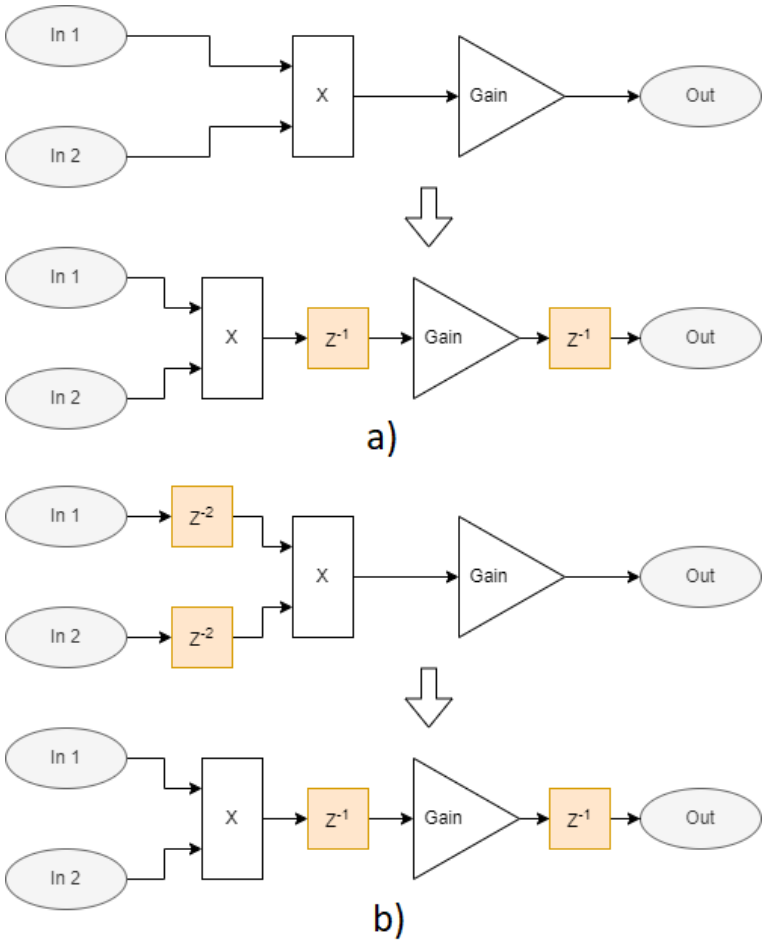

Figura 4.12 a) Ejemplo de la creación de delays por "Adaptative Pipelining" b) Ejemplo del desplazamientos de delays por "Distributed Pipelining" en un esquema con "Input Pipeline"=2.

La inserción de registros intermedios a las operaciones combinacionales es necesaria para asegurar que Vivado sea capaz de realizar la síntesis del circuito cumpliento con las restricciones de timing. Es decir, que entre registro y registro el tiempo de cálculo del circuito combinacional intermedio sea menor que el periodo del ciclo del reloj principal. Estas optimizaciones realizadas por HDL Coder permiten facilitar la tarea.

Cuando el código VHDL se genera a partir de un bloque de código MATLAB, dentro de un esquema de Simulink, HDL Coder puede realizar la conversión con dos arquitecturas (clic derecho sobre el bloque -> HDL Code -> HDL block properties -> architecture).

La opción por defecto es la "MATLAB function" que hace una traducción directa a VHDL a partir del código. A pesar de ser la opción por defecto, para este proyecto se le ha encontrado menos utilidad puesto que la ventaja que ofrece es sobretodo una mayor robustez en la generación de código. En esta opción, no se aplican optimizaciones útiles como la de "AdaptativePipeline" lo que hace que para su uso se deban de introducir registros manualmente por código entre las operaciones. El bloque que realiza las transformaciones de Clarke y de Park a partir de las matrices y del vector I_{abc} se diseñó inicialmente con esta arquitectura. En

él se usa la instrucción "coder.hdl.pipeline()" para introducir registros que hagan satisfacer las restricciones de tiempo. El código se muestra en la Figura 4.13.

La segunda arquitectura disponible es la de "MATLAB Datapath". Según los manuales de Matworks, a diferencia de la anterior, por este método se genera un esquema de Simulink a partir del código MATLAB, y sobre ese esquema de Simulink generado se aplican todas las optimizaciones disponibles. Esta ventaja es la que hace que el bloque que hace la anti-transformación, a pesar de realizar operaciones muy similares al de la transformación, requiere menos esfuerzo del diseñador al no requerir la instrucciones para colocar registros intermedios. Activando la opción de "AdaptativePipelining" este proceso se hace de manera automática. La desventaja de esta arquitectura para la generación es la compatibilidad, al pasar por un esquema de Simulink intermedio la conversión es menos robusta y se ofrecen menos compatibilidad con funciones de MATLAB a la hora de hacer un algoritmo. Las operaciones a realizar tanto en el paso a ejes dq como de vuelta a ejes abc son sencillas por lo que esta arquitectura es válida en ambos casos. Al comparar el caso anterior (Figura 4.13) con este (Figura 4.14) se ve claramente que es más conveniente usar "MATLAB Datapath" puesto que requiere menos tiempo y esfuerzo del diseñador para el mismo resultado. El caso de la transformación a ejes dq se deja hecha con "MATLAB Function" a modo de ejemplo.

```
function dq  = fcn(I, ClarkeM, ParkM)

aux= coder.hdl.pipeline(fi(ClarkeM'.*[I I],1,24,12));
ab=coder.hdl.pipeline([fi(sum(aux(:,1)),1,24,12);fi(sum(aux(:,2)),1,24,12)]);
aux2=coder.hdl.pipeline(fi(ParkM'.*[ab ab],1,24,12));
dq=coder.hdl.pipeline([fi(sum(aux2(:,1)),1,24,12);fi(sum(aux2(:,2)),1,24,12)]);
```

Figura 4.13 Código final con arquitectura MATLAB Function.

```
function Vabc  = fcn(u,ParkM,ClarkeM)

Vabc = fi(ClarkeM'*(ParkM'*u),1,17,6);
```

Figura 4.14 Código final con arquitectura MATLAB Datapath.

El último punto a comentar acerca del uso de bloques de código en el esquema de Simulink es el caso de que dichos bloques están dentro de bucles de realimentación del esquema de Simulink. Las optimizaciones de que ofrece HDL Coder no son compatibles con estos casos por lo que aún con la arquitectura de "MATLAB Datapath" será necesario introducir registros manualmente. Además, por la experiencia en este proyecto, si la arquitectura es "MATLAB Function" aunque se especifiquen explícitamente los registros con la instrucción necesaria luego no aparecerán en el código provocando un fallo de timing durante el build del bitstream. Este es el caso del bloque que implementa la funcionalidad del PI como ecuación en diferencias (código en Figura 4.15). En este caso existe una dependencia de valores pasados y por tanto se requiere un bloque que actúe como memoria y que al principio de cada periodo de muestreo, ponga a la entrada del bloque que realiza los cálculos nuevos valores pasados necesarios. Estos son los de la actuación y el error pasado. Este subsistema de memoria conformado por bloques de Unit Delay carga los datos con el pulso de inicio de periodo y tiene una entrada de reset que permite poner a 0 la memoria del PI.

```
function [ud,errd,uq,errq] = fcn(ref,P,eq2,dq,ud1,errd1,uq1,errq1)
p=dq(1);
errd=coder.hdl.pipeline(fi(ref-p,1,16,11));
ud=coder.hdl.pipeline(fi(coder.hdl.pipeline(errd*P)+coder.hdl.pipeline(errd1*eq2)+ud1,1,19,0));

q=dq(2);
errq=coder.hdl.pipeline(fi(-q,1,16,11));
uq=coder.hdl.pipeline(fi(coder.hdl.pipeline(errq*P)+coder.hdl.pipeline(errq1*eq2)+uq1,1,19,0));
```

Figura 4.15 Código con la ecuación en diferencias que implementa el PI.

Paralelamente al bloque del PI se usa una ganancia de valor wL y un par de sumadores para añadir la parte de desacoplo entre I_d e I_q.

En la Figura 4.16 se observa el esquema para el cálculo de la actuación en ejes dq a partir de la intensidad en estos mismos ejes.

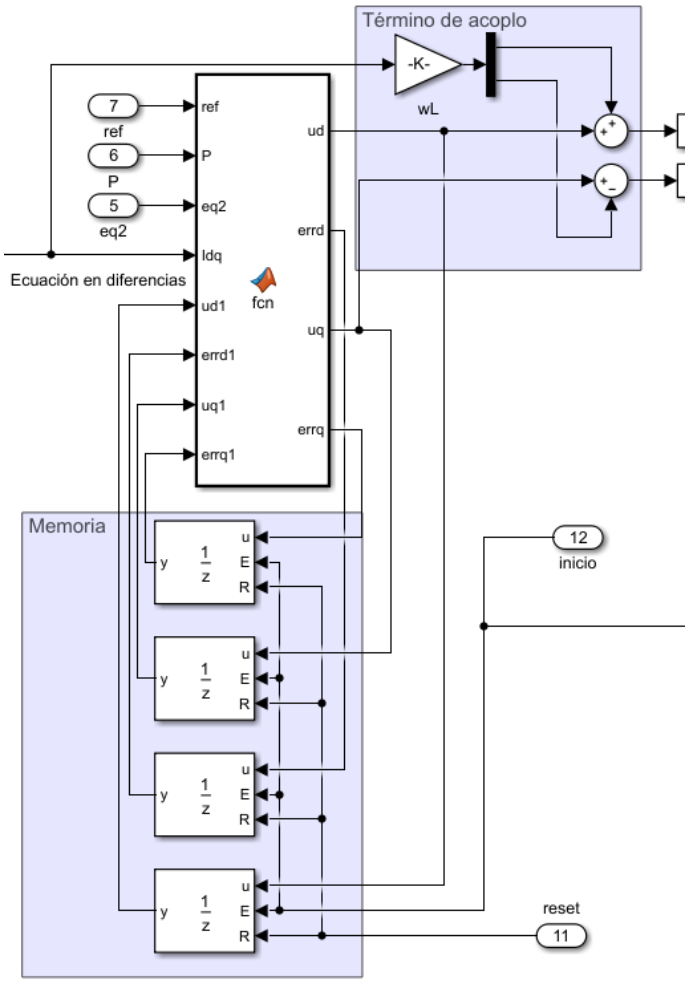

Figura 4.16 Esquema de Simulink que implementa la ecuación en diferencias (bloque MATLAB func), la memoria para la ecuación en diferencias y el término de desacoplo.

Normalizado

El normalizado de V_{abc}^{ref} para el modulador se hace dividiendo por la tensión medida del DC-Link y sumando 0.5. El modulador usado va de 0 a 1 y se debe de corresponder con las tensiones $\left(-\frac{V_{dc}}{2}\ ,\ \frac{V_{dc}}{2}\right)$. Por razones de restricción de timing no era posible hacer la división por lo que se tabularon los valores inversos de las tensiones y junto con un sumador se hace el normalizado (ver Figura 4.17).

Control del flujo de la información

La estrategia seguida para garantizar que las entradas se actualizan cuando deben y que las salidas del subsistema se mantienen estables durante todo el periodo de muestreo ha sido usar bloques de "Unit Delay Syncronous Enabled". Estos bloques se colocan tanto a la entrada como a la salida del subsistema a modo de registros y se trata todo lo que hay entre estos dos grupos de registros como si fuera un gran proceso combinacional en el sentido de que es un proceso sin memoria. Pasado cierto número de ciclos del reloj rápido las señales a la salida de este proceso serán estables y cuando llegue el inicio del siguiente ciclo estarán disponibles para cargarlas en los registros de salida. Este enfoque funciona porque la naturaleza del control y del tiempo de muestreo implica, que entre muestra y muestra, no hay entradas nuevas y sólo hay que actualizar la actuación una vez. El caso opuesto es el de la arquitectura pipeline, donde es necesario que todos los caminos entre la entrada y la salida tengan el mismo número de retrasos porque cada ciclo de reloj hay entradas nuevas y salidas nuevas. Para este tipo de arquitectura existe la opción de "Delay Balancing" que añade registros intermedios para asegurar que todos los caminos tengan el mismo numero de retrasos. En este proyecto esta opción está desactivada ya que la introducción de estos registros sólo aumentarían el consumo de recursos.

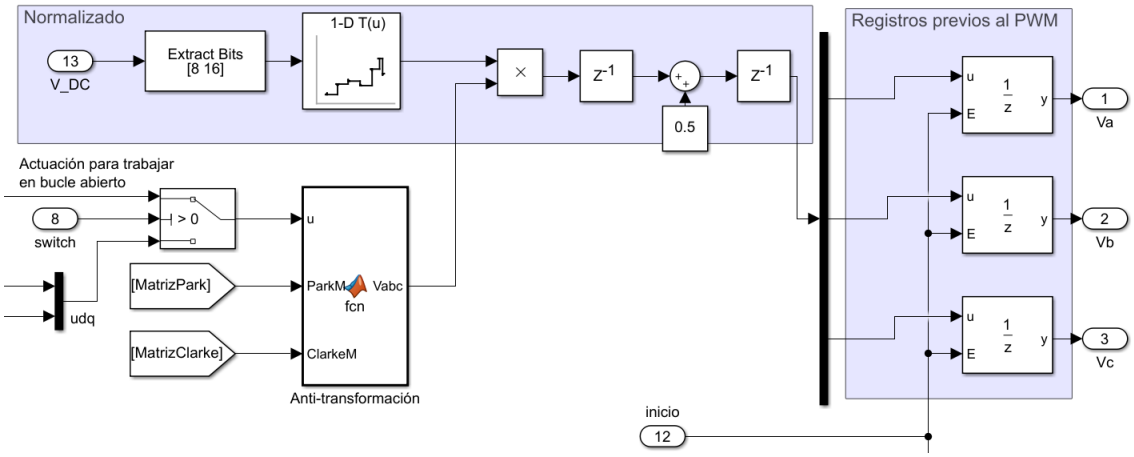

Figura 4.17 Bloques para la normalización, anti-transformación y control del instante en el que se vuelcan los datos al modulador.

4.4.3 Modulador PWM

La función del modulador es pasar de las actuaciones, que son valores continuos, a los pulsos digitales S_{abc}. Además, cumple funciones de temporización siendo el componente que ajusta la frecuencia de conmutación a la deseada (10 kHz) y el que envía la señal que activa el muestreo y da paso a un nuevo cálculo del control. La tensión de cada fase sólo puede tomar de manera instantánea o V_{dc} o 0 V. El objetivo es que la tensión media en cada periodo de conmutación sea la tensión de la actuación. En la Figura 4.18 se muestra esta explicación gráficamente. El modulador PWM consiste en comparar la actuación normalizada con una señal portadora de frecuencia superior a la frecuencia fundamental de las magnitudes eléctricas de sistema e igual a la frecuencia de conmutación deseada. Se elige como portadora una señal triangular por varias razones como que ofrece una respuesta con un mejor contenido armónico y que mejora la inmunidad al ruido dado que se realiza la medida en el punto intermedio entre dos conmutaciones [42].

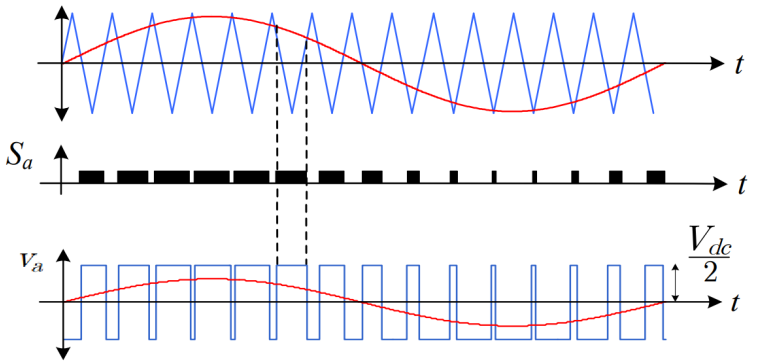

Figura 4.18 En la primera gráfica se representan los valores de la tensión de referencia normalizada para la fase A junto con los de la portadora. En la segunda se muestra el resultado de la comparación y por tanto el estado de la señal S_a. En la tercera V_a junto con la tensión de referencia sin normalizar, apreciándose cómo cuando la tensión de referencia es alta la tensión instantánea pasa más tiempo a $V_{dc}/2$ que a $-V_{dc}/2$.

Para generar la portadora se usa fundamentalmente el bloque de Simulink "HDL Counter" de la librería de HDL Coder. Este bloque contiene de manera configurable las funcionalidades básicas de un contador. En este caso se configura como un contador con la direccionalidad controlada por un biestable que se hace cambiar de estado cada vez que el contador sobrepasa ciertos límites. De este modo se consigue un bloque que cuente de arriba a abajo en el rango que se requiera. El número de niveles de la portadora viene dado por el ratio entre la frecuencia de conmutación y la frecuencia de reloj, $\frac{100MHz}{10kHz} = 10000$. La portadora es triangular para realizar las medidas lejos de los instantes de comutación (punto más bajo de la triangular),

quedando en 5000 pasos hacia arriba y hacia abajo. Esto da una resolución de portadora de algo más de 12 bits, por encima de 10 que sería el mínimo deseable. Esta cuenta debe de ser multiplicada por una ganancia que la deje en el rango [0 1] para así poder hacer la comparación. Esquema mostrado en la Figura 4.19.

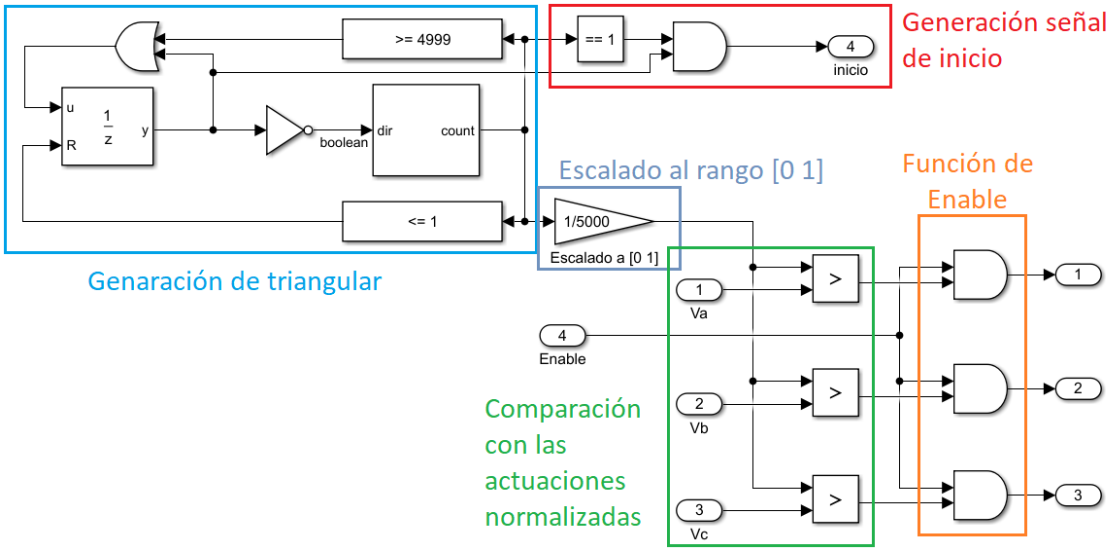

Figura 4.19 Esquema del modulador en Simulink.

La única funcionalidad añadida a este modulador es la creación de una señal que toma un valor alto sólo cuando la triangular está en bajada y además está en el instante anterior a llegar a 0 (ver Figura 4.20). Esta es la señal que comanda toda la temporización del diseño ya que es la que activa el proceso en el control del ADC y es la que habilita los registros de entrada y salida del bloque de control. De este modo todo el diseño funciona sincronizado con el modulador y comienza su ciclo cuando la triangular vale 0.

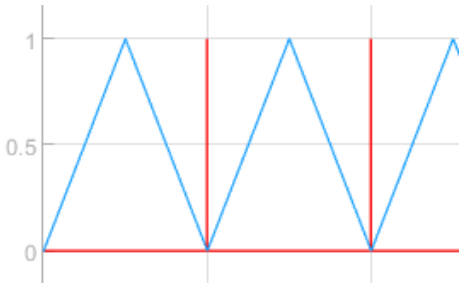

Figura 4.20 En azul la portadora triangular del modulador y en rojo la señal de inicio.

4.4.4 Máquina de estados para el control global

Para la gestión de errores y la señales de enable y de reset se implementa una máquina de estados que a partir de una señal de on/off enviada desde el ordenador permite operar el controlador. Para la máquina de estados se usa un chart muy sencillo (ver Figura 4.21). La señal de error viene de comparar las corrientes con unos límites que para este proyecto se han elegido arbitrariamente en 20 A. En una aplicación comercial se establecerían en base a criterios como el límite recomendado por el fabricante del IGBT. Para salir del error se debe accionar uno de los switches de la ZedBoard. Las señales de enable, reset y error salen por los pines de los leds de la ZedBoard para que visualmente se vea el estado en el que se encuentra el programa. La señal de reset va a la memoria del PI y la pone a 0 cuando el controlador está apagado o en estado de error, para así evitar que el PI siga integrando el error cuando no está actuando. Adicionalmente a esta máquina de estados existe una señal que pone a funcionar todo el circuito como un modulador en bucle abierto (OL Open Loop ver Figura 4.17) proporcionando V_{dq}^{ref} desde el ordenador.

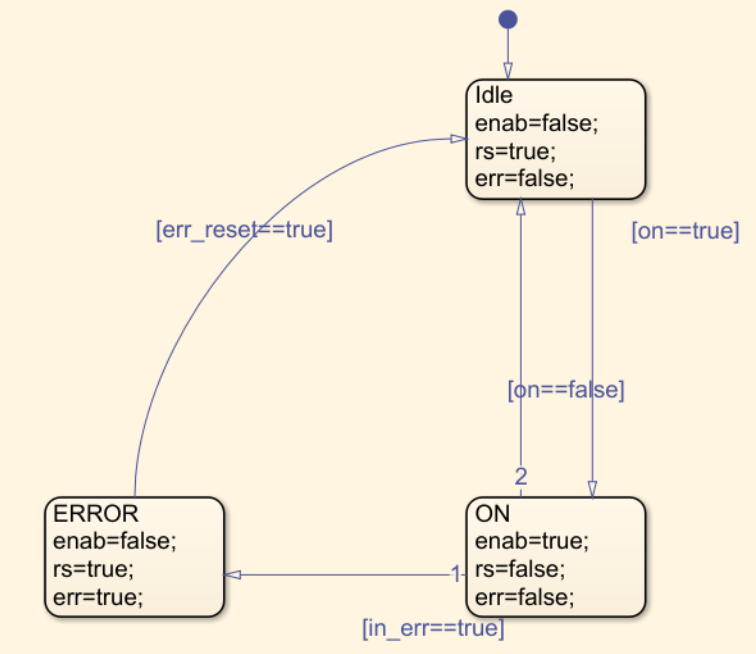

Figura 4.21 Diagrama de la máquina de estados del control global. Dispone de 3 modos, reposo, ON y ERROR y pasa de uno a otro mediante las señales de on, de error y una proveniente de una palanca de la ZedBoard, err_reset.

4.5 Paso a punto fijo

Tras tener un esquema de Simulink que al simularlo funcione, se deberán pasar las variables del esquema a punto fijo. Para implementar algoritmos en una FPGA usar variables con punto flotante resulta muy costoso en recursos, además, HDL Coder no soporta la implementación de variables en punto flotante sin librerías adicionales. MATLAB sólo trabaja con variables de tipo double y single de manera nativa, ambas de punto flotante, por lo que será necesario usar las herramientas del Add-on *Fixed-point Designer*.

Este Add-on necesario para trabajar con HDL Coder añade a la mayoría de bloques de Simulink el menú de configuración de la Figura 4.22. En este menú se da la posibilidad de especificar el número de bits de resolución, la posición de la coma binaria en caso de elegir este tipo de codificación (el único compatible con la generación de código), y si tiene o no un bit de signo. Dentro de los bloques de código sin embargo, el tipo de dato se especifica con la función "fi()". Esta función puede forzar a que el tipo de dato de salida de una función propia de MATLAB sea punto fijo en lugar de flotante, que es el tipo por defecto en la mayoría de funciones. MATLAB al hacer operaciones con variables en punto fijo calcula los resultados aumentando el número de bits para no perder resolución, para truncar el número de bits y especificar el tipo de variable de punto fijo resultante se deberá de usar de nuevo la función "fi()". La sintaxis de esta función tiene 4 argumentos, el primero es el dato que se quiere convertir a punto fijo, el segundo es un valor lógico que indica si la variable es con signo (1) o sin signo (0), el tercero es la longitud total del dato en bits y el último es la longitud de la parte de fracción. En la Figura 4.14 también se puede ver el uso de esta función. El primer argumento es el resultado de las operaciones que se quiere fijar a una lingitud en bits. El segundo argumento vale 1 por lo que el tipo de dato es con signo. El tercero es 17, lo que significan 17 bits de longitud y el último 6 lo que significa que el bit menos significativo vale 2^{-6}.

En la Figura 4.23 se describe la estructura de una variable en punto fijo, la longitud de la parte de fracción indica el valor del bit menos significativo y por lo tanto impone cuántos decimales de precisión podrá tener la variable. Además, para convertir el código de los bloques de MATLAB function resulta muy útil el menú de "Edit Data", al que se accede desde la pestaña "Editor" en la ventana de edición del código. En este menú se puede especificar el tipo de dato de todas las variables del código. Es recomendable fijar desde este menú el tipo de dato de las variables de entrada y dejar las variables internas tomar el dato heredado del código en el que se realizan los truncamientos con la función "fi()". Así se consigue que todos los tipos de datos queden bien definidos y poder cambiarlos en el futuro de manera sencilla. En la Figura 4.24 se muestra cómo acceder al menú y en la Figura 4.25 su apariencia y la configuración disponible para una variable.

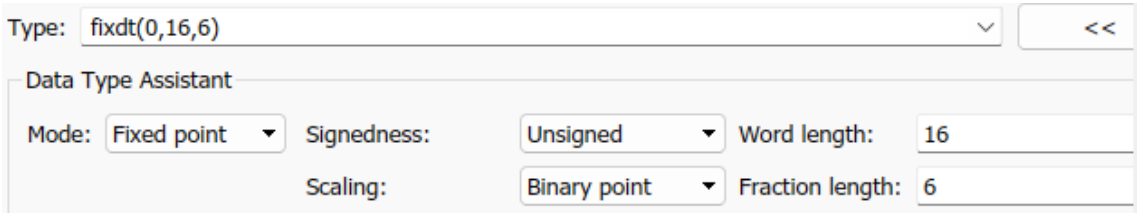

Figura 4.22 Menú para la configuración del tipo de dato.

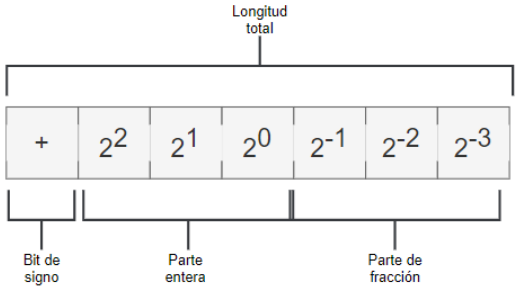

Figura 4.23 Significado de los distintos bits en variables de punto fijo.

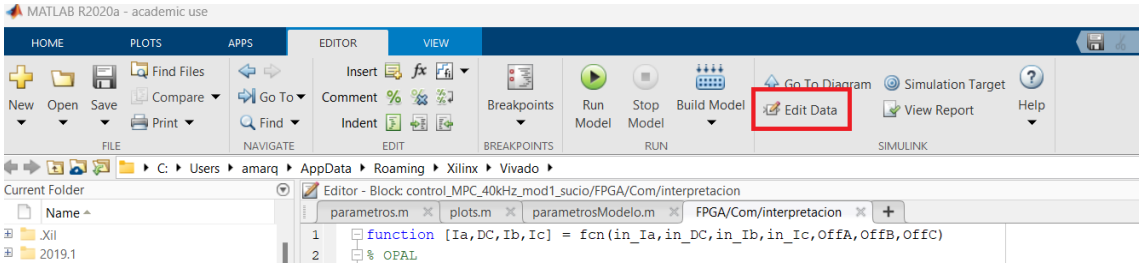

Figura 4.24 Localización del botón para acceder al menú para la configuración de las variables en un bloque de código.

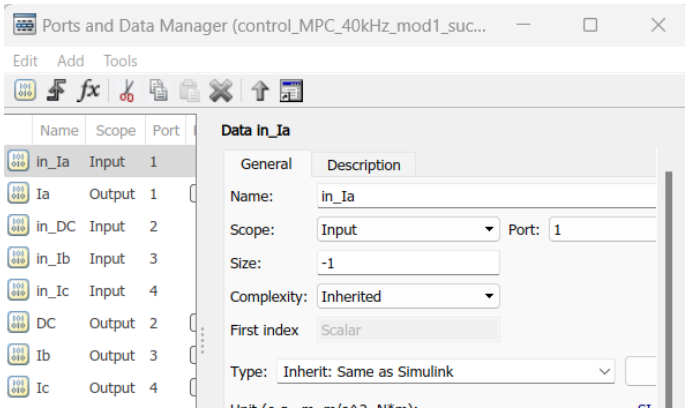

Figura 4.25 Menú para la configuración de las variables en un bloque de código.

La longitud total de bits impone la resolución, es decir, que tan grande es el rango total respecto al valor del LSB. El rango de la variable está impuesto por el bit de signo y por la diferencia entre el número de bits total y el de la parte de fracción. Es importante conocer este rango para poder asegurar que los valores de las variables no hacen overflow o underflow. Con el esquema pasado por completo a punto fijo, Simulink ofrece la posibilidad de simular de nuevo para observar los efectos de no tener variables de punto flotante. Es útil realizar estas comprobaciones para asegurar que los rangos de las variables son suficientes y para valorar los efectos de la resolución limitada en el algoritmo. Para seguir los tipos de datos es útil también tener activada la visualización de las mismas en el interfaz de Simulink (clic derecho -> other displays -> Port Data type).

En la Figura 4.3, en la que se muestra como configurar el sample time display, también se ve que debajo de esa opción aparece la de other displays para realizar la configuración descrita en este párrafo.

4.6 Workflow Advisor

A partir de este punto entra en juego la herramienta "Workflow Advisor". En la pestaña de HDL Coder en Simulink (Figura 4.2) a la izquierda aparece el icono para abrirla. Esta herramienta soporta varios enfoques para generar código y trabajar con él pero el elegido para este proyecto es el que permite hacer el desarrollo completo. Con esto se busca poder diseñar, implementar y probar el proyecto sin salir del entorno MATLAB/Simulink. La herramienta Workflow advisor funciona en base a una lista de pasos, cada uno con sus particularidades, que se ejecutan en orden. El flujo de trabajo de Workflow Advisor elegido para este proyecto consta de 4 grupos de tareas. Dentro de cada uno de estos grupos hay 4 pasos. La selección del flujo de trabajo elegido para el Workflow advisor se hace en el primer paso de todos, el 1.1 Set Target Device and Synthesis tool, mostrada su configuración en la Figura 4.26. El flujo de trabajo usado es el de "IP Core Generation" con la ZedBoard como placa objetivo y con Vivado como herramienta de síntesis. El resto de opciones quedan fuera del alcance. La apariencia y los pasos dentro del Workflow Advisor pueden cambiar en función del flujo de trabajo seleccionado.

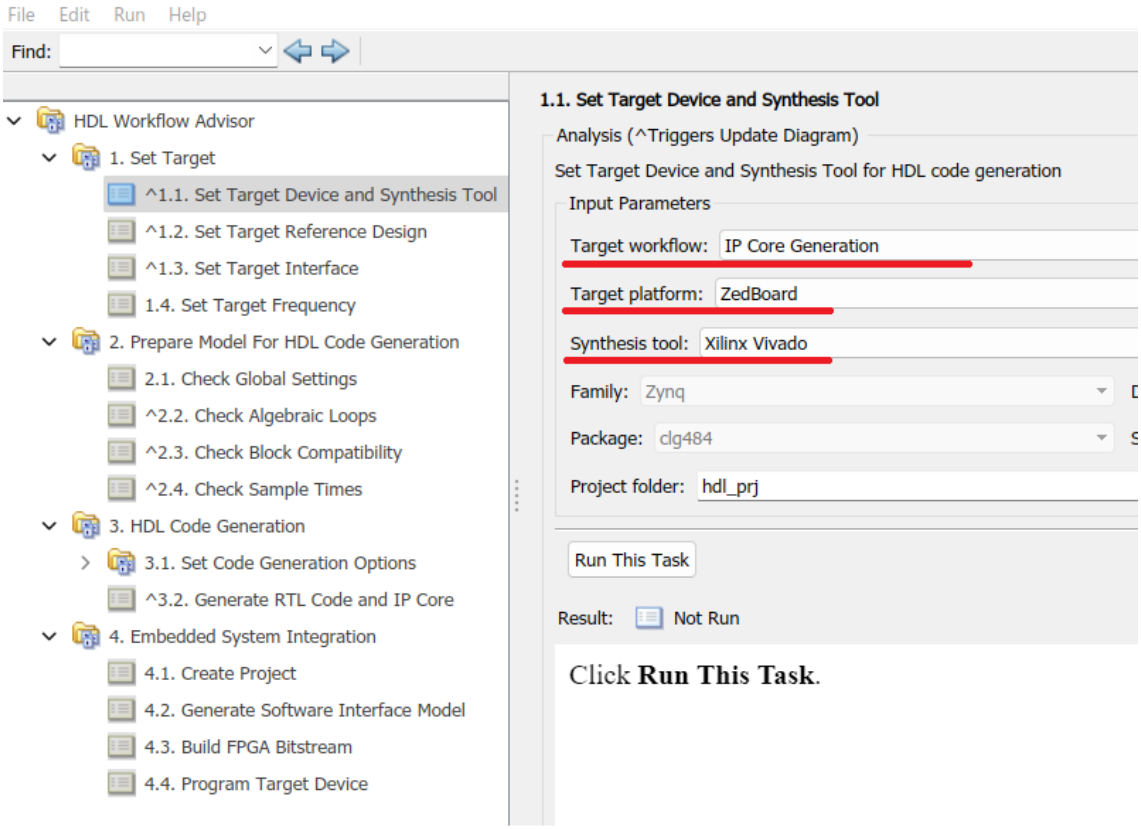

Figura 4.26 Menú de *Workflow Advisor* con el que se trabaja para generar el código.

Cada paso dentro de este menú permite hacer configuraciones sobre la generación de código o hace comprobaciones de compatibilidad y de configuración para asegurar que el proceso funciona correctamente. Este documento no entrará en detalle de cada opción pero si pretende dar una visión general de los pasos más relevantes para el diseñador, y aunque los errores más comunes se tratarán más adelante, se dará una indicación de la naturaleza del error que dan dichos pasos.

Los pasos del workflow elegido son los siguientes:

- ***1. Set Target***

- *1.1 Set Target Device and Synthesis tool*: A parte de la selección del flujo de trabajo, del hardware y de la herramienta de síntesis se selecciona la carpeta en la que se guardará el proyecto.
- *1.2 Set Target Reference Design*: Este paso comprueba que el esquema es coherente desde el punto de vista de Simulink y guarda un registro de la interfaz entrada-salida del subsistema objetivo, por lo que es importante volver a ejecutar este paso si se hace cualquier cambio en las variables de entrada-salida en Simulink, incluido cambiar el tipo de dato de punto fijo.
- *1.3 Set Target Interface Model*: Aquí se hace la asignación de las entradas y salidas a pines físicos de la FPGA o a los registros del protocolo AXI-Lite que conectan con el ARM y a través del cuál se puede enviar o recibir información desde el ordenador de desarrollo. Esta es la información que usará la herramienta para generar el fichero de restricciones (.xdc) en el futuro.
- *1.4 Set Target Frecuency*: Establece la frecuencia de reloj del sistema. La temporización del código viene dada sólo por este dato y por los ratios entre los tiempos de actualización establecidos en el esquema. HDL Coder no comprobará que esta frecuencia coincida con la establecida en el esquema de Simulink.
- ***2. Prepare model for HDL Generation:*** Todo este grupo hace comprobaciones sobre el esquema y la configuración de Simulink
- *2.1 Check Global Settings*: La primera vez que se ejecuta da errores sobre la configuración del propio HDL Coder con mensajes de error concretos que permiten su fácil solución. Una vez resueltos no reaparecen.
- *2.3 Check Block Compatibility*: Da error sobretodo cuando el esquema de tiempos no cumple con las directrices anteriores o cuando el paso a variables en punto fijo no se ha hecho correctamente. Será necesario seleccionar la opción de ignorar los warnings debido a que hay algunos que aparecen a modo de advertencia sobre un posible error. Será necesario tenerlos en cuenta pero habrá que ignorarlos para avanzar.
- ***3. HDL Code Generation:*** En este grupo se configuran opciones sobre la generación de código y se procede a la misma. Las configuraciones abarcan desde el lenguaje objetivo (VHDL por defecto) hasta opciones en la optimización del código generado, pasando por estándares de codificación, qué nombre dar a ciertas variables y demás opciones avanzadas. Para hacer alguna modificación sobre el código antes de sintetizar, implementar y cargarlo en la FPGA pero sin salir del flujo de trabajo propuesto se puede hacer entre este paso y el siguiente. Esto se hace accediendo a los ficheros del proyecto que luego usará Vivado (.vhd .xdc etc) y modificándolos. Un ejemplo donde es interesante es para configurar resistencias de pull-up o pull-down. La ZedBoard tiene esta opción, pero la librería aportada por Xilinx para este Add-on no, por lo que para poder configurarlas se deberá realizar modificando los ficheros .xdc generados tras este paso.
- ***4. Embedded System Generation***
- *4.1 Create Project*: Crea el proyecto de Vivado.
- *4.2 Generate Software Interface Model*: Crea automáticamente un modelo de Simulink para programar el ARM con la interfaz necesaria para transpasar información del ARM a la FPGA y así poder tomar datos y cambiar variables internas.
- *4.3 Build FPGA bitstream*: Llama a Vivado a través de una teminal para hacer todo el build.
- *4.4 Program Target Device*: Carga el bitstream en la FPGA si Vivado consigue completar el proceso sin un error en las restricciones de tiempo.

Si Vivado no consigue generar el bitstream en la terminal aparece un mensaje de error, que normalmente se referirá a un problema de slack negativo. La única manera de diagnosticar la causa de estos errores es abrir el proyecto de Vivado y examinar el timing report y el RTL. La solución a estos problemas es añadir registros entre procesos combinacionales del algoritmo. Esto puede hacerse con "Ditributed Pipeline" o "Adaptative pipeline". Para añadir los registros manualmente se usan bloques de Delay en Simulink y la orden coder.hdl.pipeline() en código MATLAB. Alternativamente a añadir retrasos que no todas las aplicaciones permiten, se puede disminuir la frecuencia de reloj pero ya esto es una decisión del diseñador.

Resumiendo este apartado, Workflow Advisor sirve de guía para el proceso final para la generación e implementación del código VHDL. Proporcionando configuraciones generales, detección de errores, la propia

generación del código y lo necesario para poder ponerlo en marcha en la plataforma escogida. Haciendo la comparación con programas para compilar código C y cargarlo en un microcontrolador se pueden ver muchas similitudes. El precompilador de estos y el código para el mismo realizan una configuración. En caso de errores en el código el compilador da mensajes de error. Si el código es correcto se compila a un binario y después muchos de estos progamas dan la opción de cargarlo en el micro y hacer debug si se tiene la placa de evaluación necesaria.

4.7 Monitorización de la FPGA

Si Vivado consigue generar el bitstream, el último paso es usar la ventana (mostrada en la Figura 4.27) con el modelo generado automáticamente por el "Workflow Advisor" para programar el ARM a través del Add-on *Embedded Coder*. Estableciendo el stop time a "inf", y pulsando la opción de "Monitor & tune" se carga el programa en el ARM y se comienza a poder monitorizar el funcionamiento del diseño. Desde esta ventana se puede cambiar en línea parámetros que se hayan definido como entradas a través del protocolo AXI-Lite en el apartado 1.3 del workflow y monitorizar las salidas que se hayan configurado de la misma forma (ver Figura 4.28). El flujo de trabajo propuesto usa sólo los núcleos de ARM como interfaz con el ordenador y el procesado de la información es implementado por completo en la FPGA. La creación automática de un modelo para que *Embedded Coder* lo convierta en código C y lo cargue en el ARM hace que esta parte del trabajo sea transparente para el diseñador, lo cual simplifica el proceso de diseño aunque restringiendo las posibilidades. El modelo generado se puede modificar para poder programar en el ARM funciones adicionales, pero trabajar con los esquemas generados automáticamente puede ser difícil de interpretar. Para programar el ARM habría que entrar dentro del bloque azul que aparece en la Figura 4.27 y ahí buscar las señales que se hayan configurado como salidas de la FPGA por protocolo AXI, realizar la programación en base a los métodos propuestos por "Embedded Coder" y volver a enviarlos a la FPGA por una entrada previamente configurada por protocolo AXI. Este proceso no se describe en este documento porque ha quedado fuera del estudio. Los dos puntos que se deben de tener en cuenta a la hora de trabajar con esta ventana son el tiempo de actualización y los tipos de las variables de los bloques externos al del ARM.

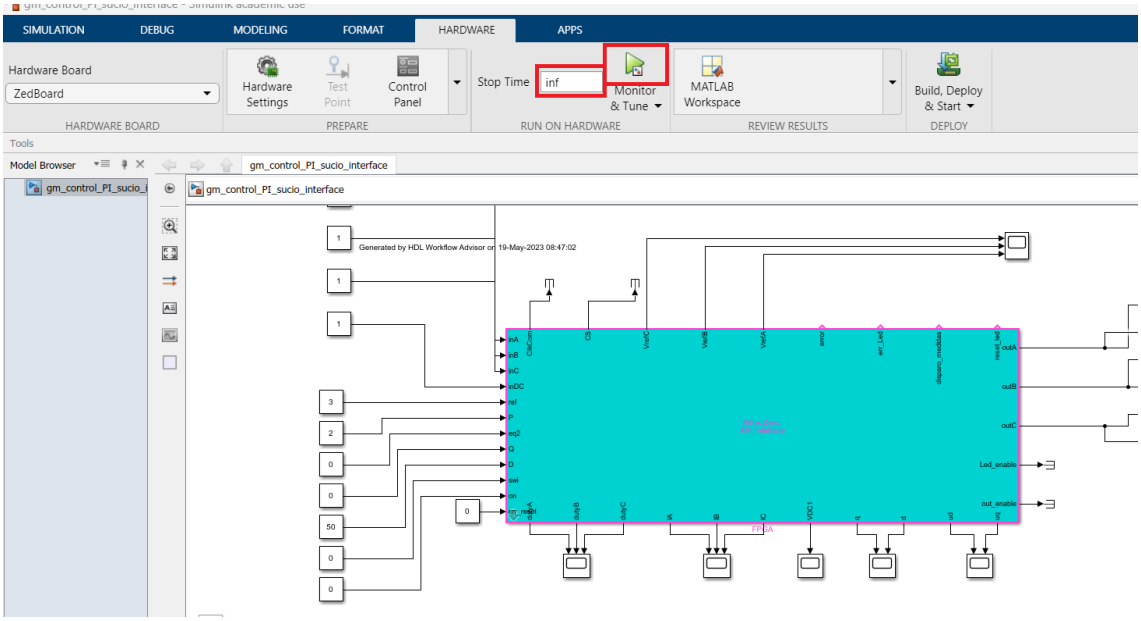

Figura 4.27 Imagen del aspecto general de la ventana con el modelo para el ARM con la que se hace el monitoreo.

El modelo para el ARM es, análogamente al diseño para HDL Coder, un subsistema (bloque azul que aparece en la Figura 4.27) en el que se incluye todo lo que se carga en el ARM (que se podría modificar). Todo lo que haya fuera de este subsistema lo usará Simulink para realizar cálculos con la información extraída ya en el ordenador. Puede ser útil para realizar medidas de evaluación del comportamiento general del control. Para que el intercambio de la información funcione, los bloques exteriores al de ARM deben de actualizarse

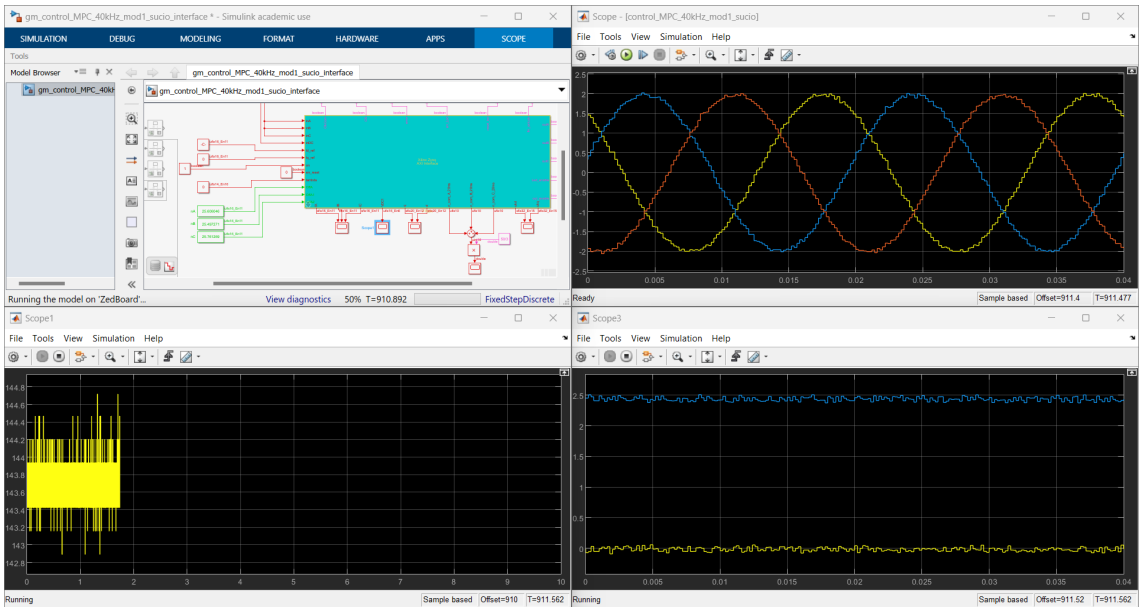

Figura 4.28 Captura de pantalla de la apariencia general durante el monitoreo a través de los scopes.

a una frecuencia múltiplo de la configurada en el diseño para la FPGA. Es importante hacer esta asignación de manera explícita (ver Figura 4.29), puesto que dejarlo en modo heredado hará que el nuevo esquema, que no posee los componentes que están dentro del subsistema que representaba la FPGA, tome erróneamente el tiempo de actualización. El programa que se generaría para el ARM no tendrá la temporización que deberá de tener y fallará al enviar y recibir información, sin dar un mensaje de error al usuario.

El paso de datos por el registro AXI (Interfaz entre el ARM y la FPGA) es por registros de 32 bit. El ARM no contempla la variedad de tipos de dato en punto fijo posibles, por lo que para pasar información al ordenador el ARM interpreta el dato como un uint32 y deja a Simulink la interpretación de ese dato. Por esta razón es tan importante el punto 1.2, y por esto se debe de volver a pasar cada vez que se haga alguna modificación en lo referente a entradas y salidas.

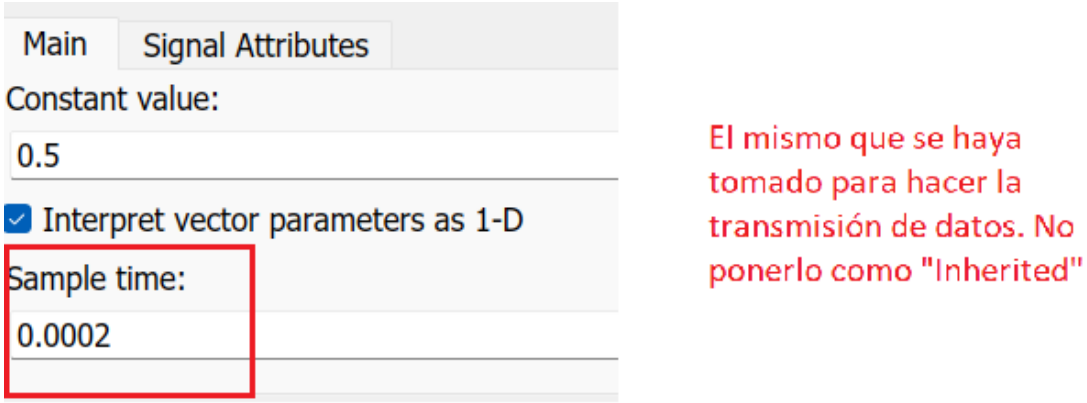

Figura 4.29 Configuración de un bloque de constante de uno de los parámetros de entrada del ARM.

5 Implementación del control FCS-MPC

5.1 Esquema de tiempo

El esquema de tiempo para este controlador queda prácticamente idéntico al del control PI con la salvedad de que en lugar de un modulador hay un timer para obtener la señal de inicio que se generaba antes en el propio modulador. En la Figura 5.1 se encuentra este esquema con el mismo código de colores que en la Figura 4.4.

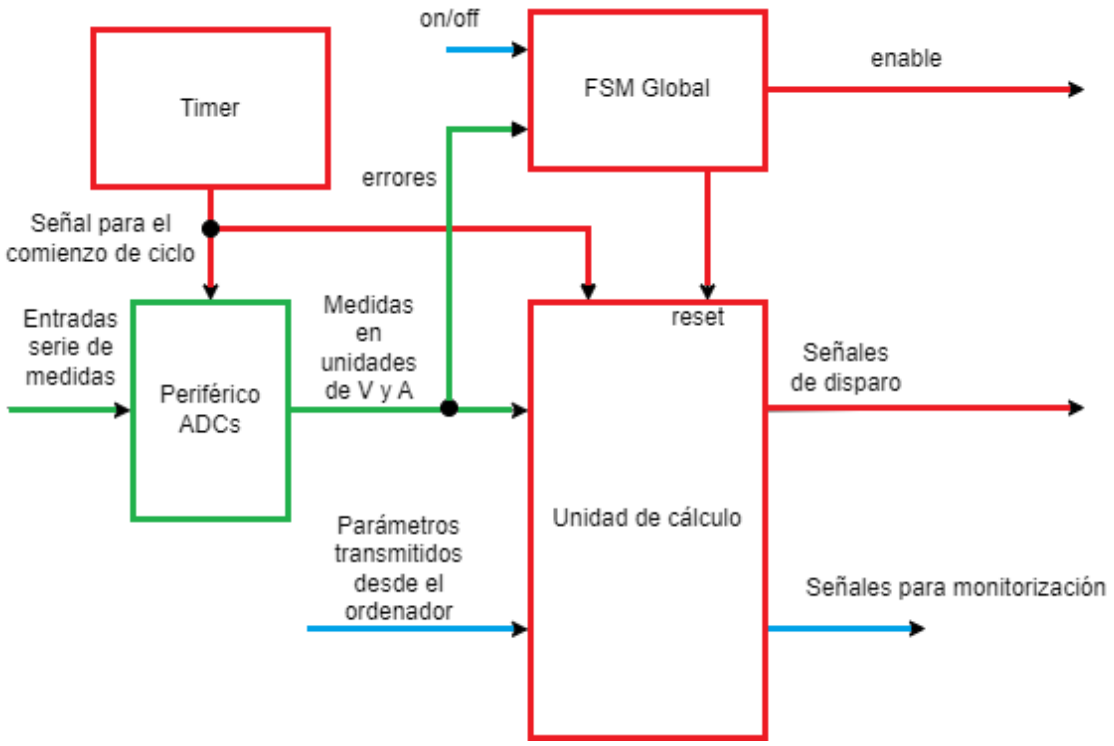

Figura 5.1 Diagrama de bloques del diseño del sistema basado en el algoritmo FCS-MPC.

5.2 Creación del esquema de Simulink

Para el control FCS-MPC se mantienen los diseños del periférico de control de los ADCs y la FSM que dirige el funcionamiento global. Esta estrategia de control no requiere un modulador. Se adaptará el modulador anterior para funcionar como un timer que cada intervalo de muestreo mande una señal que dé comienzo un nuevo cálculo del algoritmo de control. Se elige como frecuencia de muestreo 50 kHz, para poder comparar con los resultados obtenidos con otro controlador implementado por métodos convencionales. La única parte que se debe de diseñar es la parte del cálculo, que en este tipo de control tiene más peso. Por último, para

tomar medidas sobre el tiempo de cálculo del algoritmo de control y la frecuencia de conmutación se montan dos circuitos muy simples.

Dentro del subsistema de cálculo se mantiene la transformación a ejes dq de la intensidad medida I_{abc} y el único cambio a realizar en este apartado es ajustar la generación de la matriz de Park a la nueva frecuencia de muestreo. De modo que el contador con el que se indexaba en la tabla y el número total de puntos pasa a ser de 800 en lugar de 200. Adicionalmente, se guardan dos valores de la matriz de Park, el actual y el futuro, puesto que el valor futuro será necesario para hacer la predicción.

5.2.1 Algoritmo

Continuando con la estrategia de diseño se hace el bloque de código que realiza las predicciones del mismo modo que el anterior, tratando de paralelizar lo máximo posible los cálculos. El flujo de la información en el algoritmo queda como en la Figura 5.2. El horizonte de predicción es 1 y la decisión de qué estado imponer en el convertidor en cada tiempo de muestreo se toma en base a una función de coste. Para realizar la predicción en $k+2$ se hace primero una a $k+1$ a partir de las medidas en k y el estado del convertidor en k. Esta predicción difiere de la que se hizo en el periodo $k-1$ para el instante $k+1$ debido a que se hace a partir de las nuevas medidas tomadas y no a partir de una predicción. Para hacer las predicciones a $k+2$ se necesita obtener primero todas las posibles actuaciones en ejes dq para el instante $k+1$. Aquí entra la matriz de Park de $k+1$ que se multiplica por una matriz de constantes que contiene todos los vectores de las tensiones de salida del convertidor en ejes $\alpha\beta$ Tabla 5.1. Estos vectores se tienen precalculados por ser constantes. Todas las predicciones para $k+2$ se hacen en paralelo. La función de coste tiene en cuenta tanto el error al final del estado $k+2$ como el número de conmutaciones necesarias para cambiar desde el estado en k al siguiente. Un parámetro lambda permite balancear el peso que tiene en la función de coste las conmutaciones respecto al error instantáneo. Se obtiene de todos los costes el mínimo y se asigna a la salida por un lado, el vector con las tres señales de disparo que irán al convertidor en el siguiente ciclo y el indice y los valores en dq de la actuación elegida. El índice servirá para indexar la tabla de costes de conmutación, una tabla en la que cada fila se corresponde con un estado inicial y cada columna a un estado final, el valor de cada celda es el número de ramas que deben de cambiar de estado. Los valores de u_{k+1} en ejes dq serán el próximo u_k porque la matriz de Park en $k+1$ se corresponderá con el de k en el siguiente ciclo. De este modo se ahorra tener que volver a realizar la transformación. En la Figura 5.2 se muestra el diagrama de bloques del algoritmo y en la Figura 5.3 se ve el esquema de Simulink del subsistema de cálculo

Tabla 5.1 Tabla de los valores de tensión a la salida del convertidor por 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}
0	0,0,0	0	0	0	0	0
1	1,0,0	2/3	-1/3	-1/3	0.8164966	0
2	1,1,0	1/3	1/3	-2/3	0.4082483	0.70710678
3	0,1,0	-1/3	2/3	-1/3	-0.4082483	0.70710678
4	0,1,1	-2/3	1/3	1/3	-0.8164966	0
5	0,0,1	-1/3	-1/3	2/3	-0.4082483	-0.70710678
6	1,0,1	1/3	-2/3	1/3	0.4082483	-0.70710678
7	1,1,1	0	0	0	0	0

5.2.2 Bloques adicionales

Para estimar el tiempo de cálculo de manera experimental se monta un circuito que compara la salida del bloque de cálculo con el valor que tenía en el ciclo de reloj anterior. Por cada ciclo consecutivo en el que son iguales se suma 1 a un contador y cuando no son iguales se resetea. Cuanto el contador es mayor que 100 se pone a 1 un pin de la ZedBoard. De esta manera, se puede ver en el osciloscopio cuando la salida lleva 100 ciclos de 10 ns siendo estable. Sabiendo cuando se actualizan las variables de entrada y teniendo una idea previa del orden de magnitud del tiempo de cálculo, puede deducirse cuando ha finalizado el cálculo de manera concreta a partir de esta señal vista en el osciloscopio. En la Figura 5.4 se muestra el esquema que implementa esta funcionalidad. Puntualizar que la señal que entra es el vector S_{abc} por lo que al hacer la comparación se usan dos puertas ANDs para comprobar que todas las comparaciones tienen resultado lógico 1.

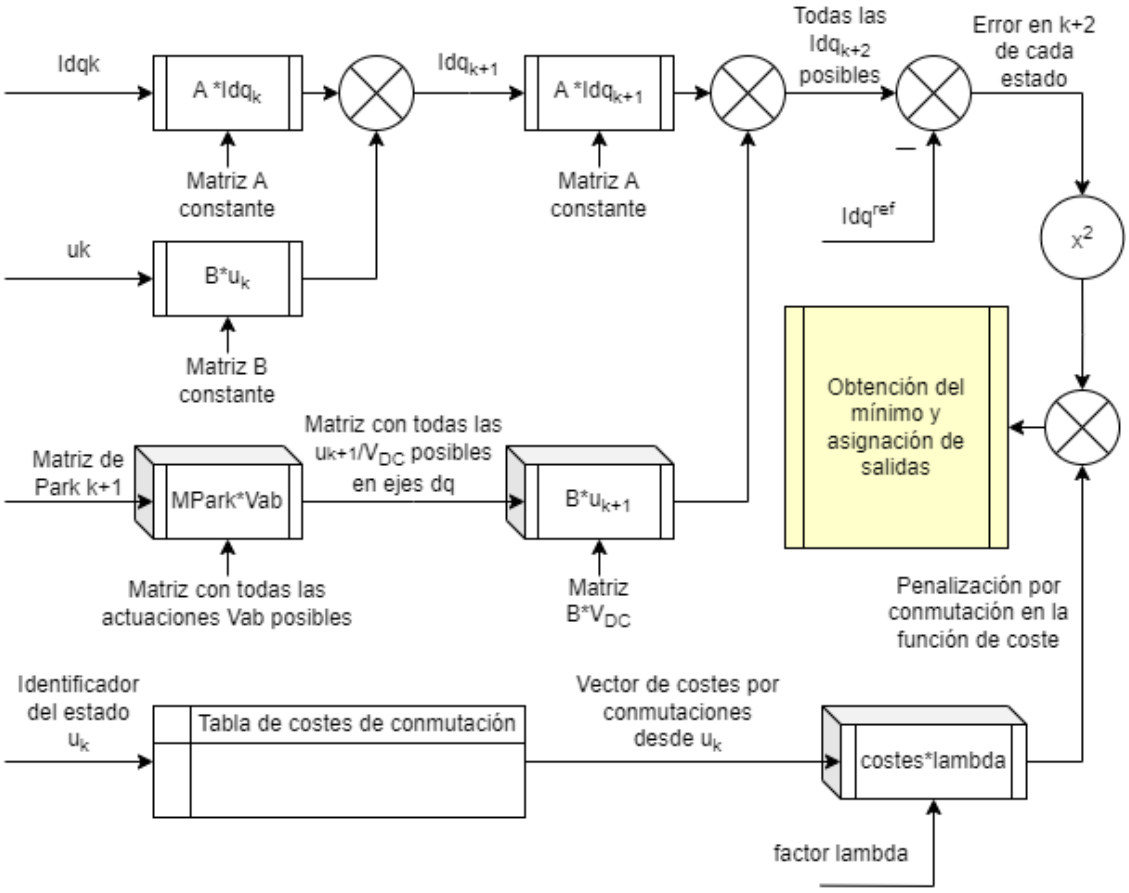

Figura 5.2 Diagrama de bloquees del algoritmo FCS-MPC.

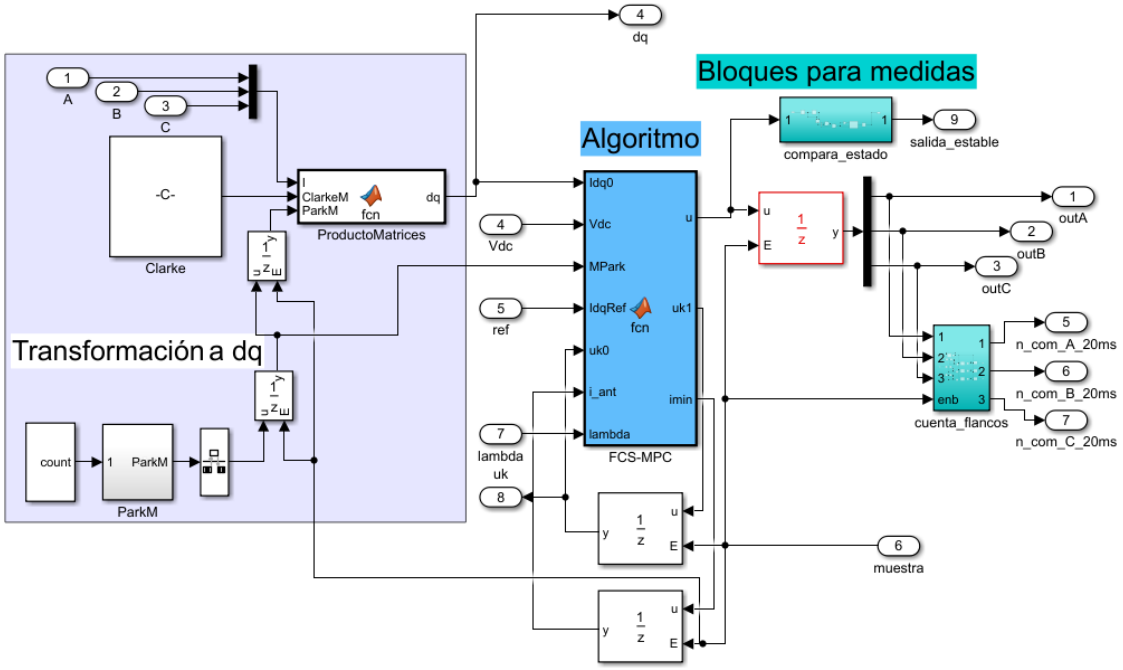

Figura 5.3 Esquema del bloque de cálculo en el FCS-MPC.

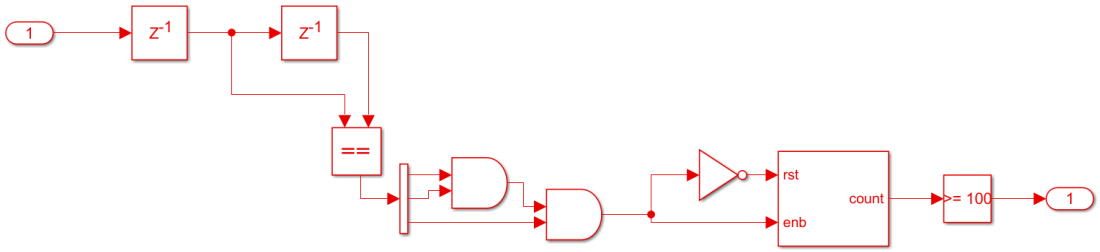

Figura 5.4 Esquema del bloque para la medición del tiempo de cálculo.

Para calcular la frecuencia media de conmutación se tiene un detector de flanco de subida en cada una de las señales que conforman el vector S_{abc}. Un contador cuenta el número de flancos y cada 20 ms se resetea y envía la cuenta al ordenador. Una vez allí, Simulink calcula la frecuencia de conmutación media por periodo. Se calcula como la suma de todas las conmutaciones entre las tres fases entre 3 (nº de conmutaciones medio por fase) y entre 20 ms (nº de conmutaciones medio por periodo y por fase). Ecuación (5.1) donde N_a,N_b,N_c se corresponden con el número de conmutaciones en un periodo de cada fase. En la Figura 5.5 está el esquema que implementa la función. Hay tres canales, uno por señal. Fundamentalmente cada canal es un contador que recibe una señal de enable proveniente de un detector de flanco. Cada 20 ms el contador que se ve arriba en la imagen envía una señal que carga los tres datos en los registros de salida y resetea los contadores para reiniciar el ciclo

$$\bar{F}_{sw} = \frac{N_a + N_b + N_c}{3 \times 20ms} \tag{5.1}$$

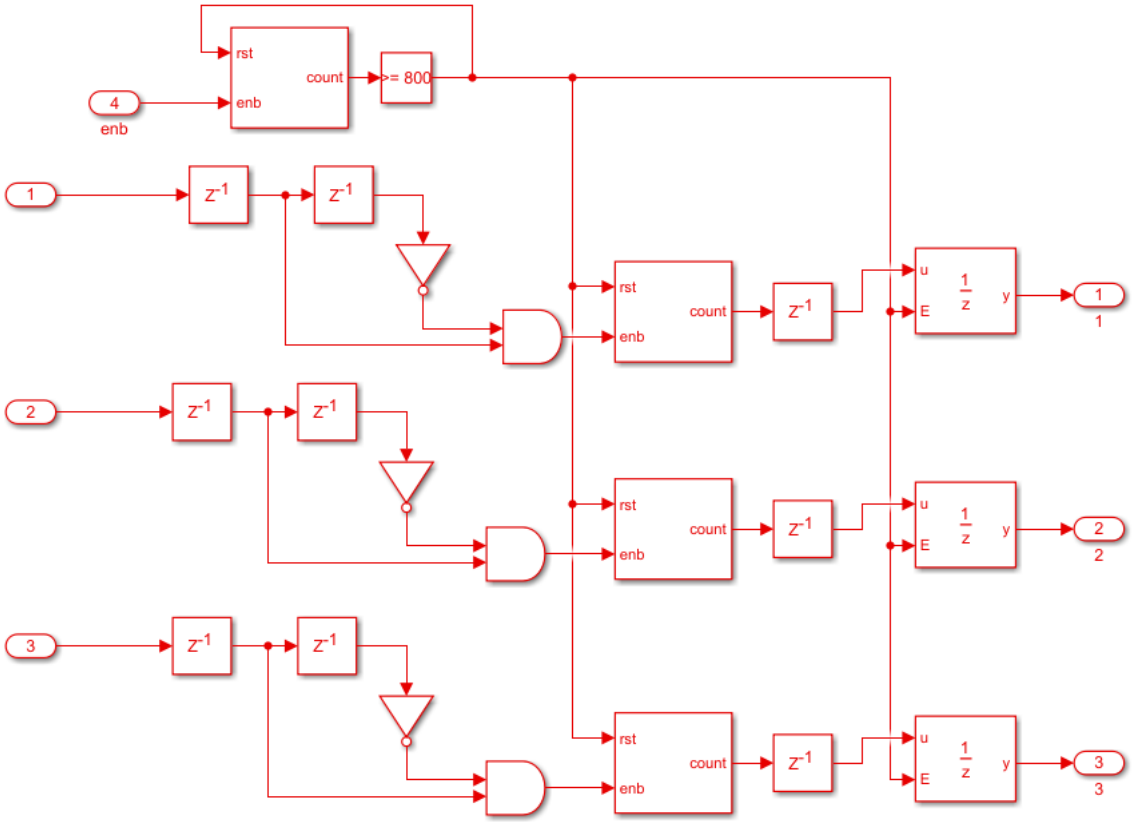

Figura 5.5 Esquema del bloque que cuenta los flancos de S_{abc}.

6 Resultados

Los resultados se analizarán tanto en tiempo con medidas obtenidas desde un osciloscopio, como en frecuencia realizando la FFT. Primero se mostarán unos experimentos con cambio de referencia que demuestren el funcionamiento de ambos controladores. Posteriormente se realizará la comparación con los datos proporcionados por el laboratorio de los controladores implementados en HDL. Además se expondrá el uso de recursos de la FPGA y los tiempos de cálculo del control MPC, para seguir con la comparación.

6.1 Validación de los controladores

Como se ve en la Figura 6.1, ambos controladores funcionan correctamente y alcanzan la referencia.

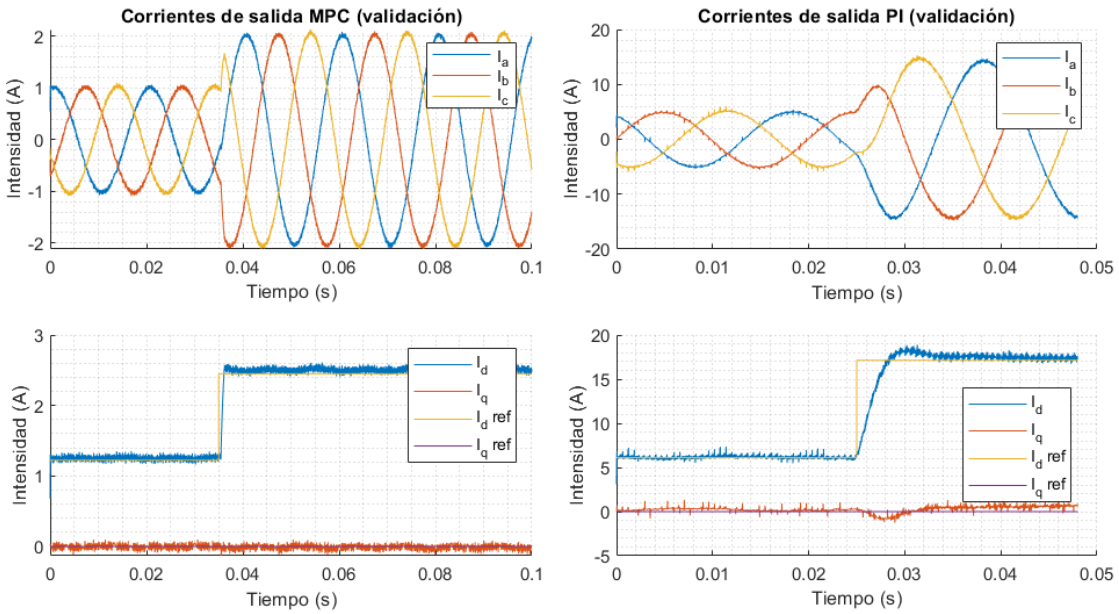

Figura 6.1 A la izquierda, experimento para el control FCS-MPC con V_{dc} =140 V, R=30 Ω, L=20 mH y un cambio de referencia de 1 a 2 A de amplitud. A la derecha, experimento para el control PI con V_{dc} =750 V, R=15 Ω, L=10 mH y un cambio de referencia de 5 a 14 A de amplitud.

6.2 Comparación

En la Tabla 6.1 se muestran las condiciones de estos experimentos. El cálculo del THD se hace como la media del THD de las tres fases calculado hasta 2.5 kHz.

Como se ve en las Figura 6.2 y Figura 6.3 los comportamientos en régimen permanente son muy similares, el control PI hecho en HLS tiene un poco menos contenido armónico que su contraparte en HDL, y en el

Tabla 6.1 Condiciones para los experimentos de validación y comparación.

Variable	Valor
Resistencia	15 Ω
Inductancia	10 mH
V_{dc}	750 V
I^*_{abc}	14 A

FCS-MPC es al revés. Para el FCS-MPC se ajustó en parámetro λ de modo que quedara a 5 kHz la $\bar{F}_{sw}$. La diferencia de THD en el FCS-MPC puede deberse a que aunque se ajustaran para dicha frecuencia media, en la práctica varía un poco en cada ciclo y en los experimentos hechos la del HLS tenia una frecuencia un poco inferior que la del HDL.

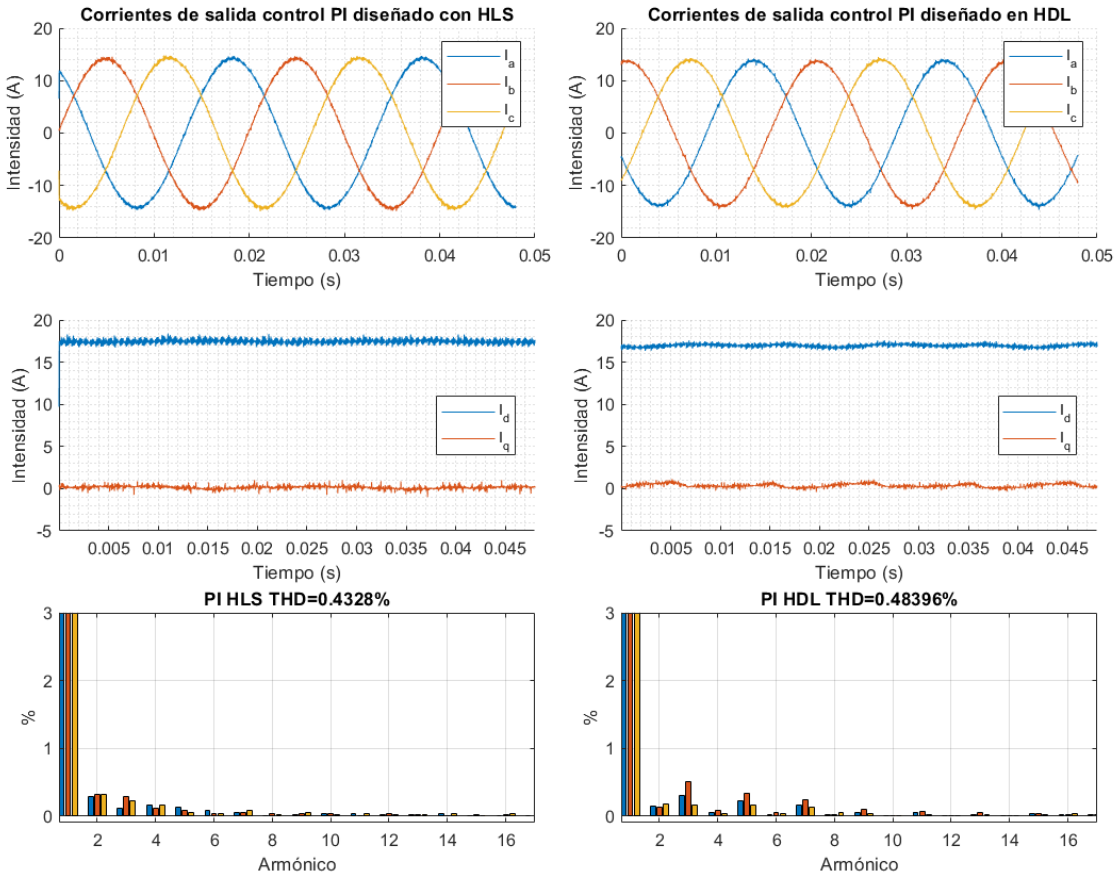

Figura 6.2 Comparación entre los controladores PI por HLS y por HDL.

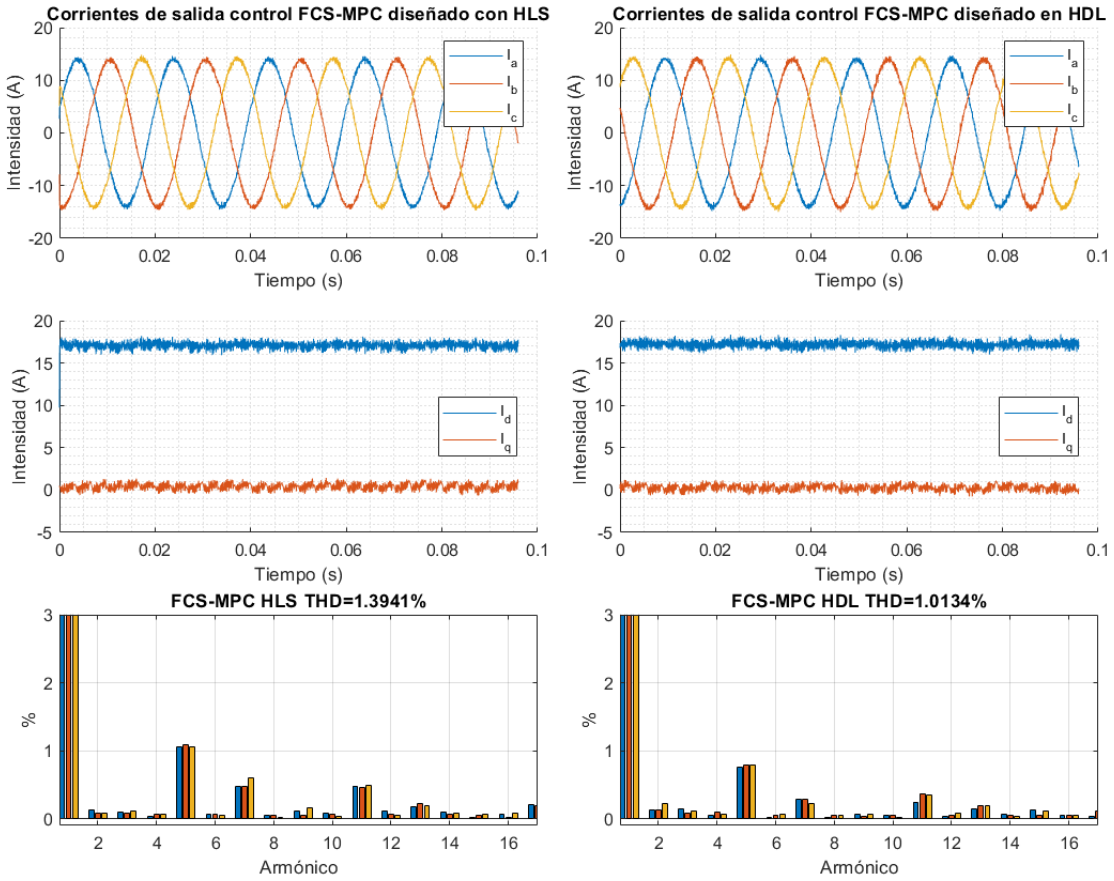

Figura 6.3 Comparación entre los controladores PI por HLS y por HDL.

6.3 Comparación en uso de recursos

Obteniendo desde Vivado los datos de uso se tiene la Tabla 6.2 en la que como "Resto" se incluyen todos los componentes que añade HDL Coder para apoyar al diseño como son los divisores de frecuencia y los componentes para hacer la gestión del paso de información del ARM a la FPGA a través del protocolo AXI. Para poder realizar la comparación se toman los datos del uso de recursos de los dos controladores implementados en HDL del artículo [26]. En dicho artículo se implementa el FCS-MPC en HDL usando Vivado y con dos estrategias diferentes. Una tratando de paralelizar los cálculos para reducir el tiempo requerido, como se hace en este proyecto, y la otra de manera secuencial para ahorrar área. Vivado ofrece la opción de que la implementación se haga sin usar bloques DSP de la FPGA. Todos los resultados se muestran en la Tabla 6.3 para realizar la comparación.

Tabla 6.2 Uso de recursos en FPGA FCS-MPC con HLS.

Bloque	LUTs	BRAM	Registros	DSP
Periférico ADCs	456	0	469	4
Timer	25	0	17	0
Cálculo	7572	1	6192	108
FSM global	2	0	2	0
Resto	1381	0	2572	0
Total	9436	1	9252	112
% del máximo	17.8 %	0.7 %	8.7 %	51 %

Tabla 6.3 Comparación uso de recursos en FPGA FCS-MPC con HLS y HDL.

Bloque	LUTs	BRAM	Registros	DSP
Total HLS	9436	1	9252	112
Total HDL paralelo	5158	2	3958	200
Total HDL paralelo sin DSPs	46805	2	4078	0
Total HDL secuencial	2562	2	3746	70
Total HDL secuencial sin DSPs	17262	2	3897	0

Aunque la comparación no es del todo directa debido a los bloques de DSP, si se observa que el diseño hecho con HDL Coder tiende a usar bastantes más registros que el hecho a bajo nivel. El uso de DSP es significativamente menor con HDL Coder lo que podría ser relevante en el uso de LUTs que es de nuevo muy favorable para el método clásico. Como se ve en las filas del diseño en HDL sin usar bloques DSP, estos son muy relevantes en el uso de LUTs. En general se aprecia que el diseño con HDL Coder a pesar de ser menos eficiente se mantiene en el orden de magnitud.

6.4 Parámetro λ y tiempo de cálculo

La última funcionalidad que queda por probar del control MPC es la regulación de la frecuencia media de conmutación $\bar{F}_{sw}$ con el factor λ. Para este apartado se realiza el experimento descrito en la Tabla 6.4 con los resultados mostrados en la Figura 6.4 junto con sus armónicos y las $\bar{F}_{sw}$ en la Tabla 6.5 .

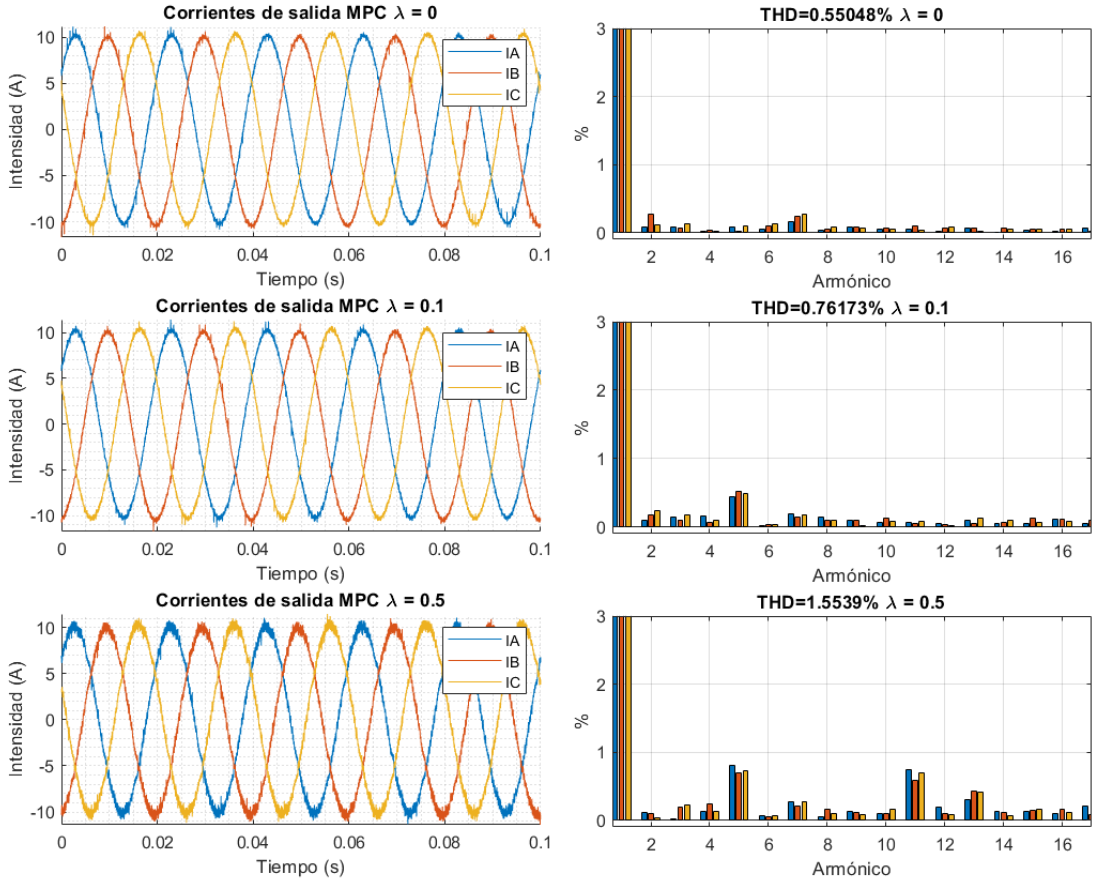

Figura 6.4 Corrientes de los experimentos para varios λ y sus espectros.

Tabla 6.4 Condiciones para el experimento para la regulación de $\bar{F}_{sw}$ mediante λ.

Variable	Valor
Resistencia	30 Ω
Inductancia	20 mH
V_{dc}	700 V
I^*_{abc}	10 A

Tabla 6.5 Efecto de λ en la frecuencia media de conmutación.

λ	$\bar{F}_{sw}$
0	9.1 kHz
0.1	6.8 kHZ
0.5	2.5 kHz

En cuanto al tiempo de cálculo, se obtuvo mediante la estrategia descrita en el apartado correspondiente de bloques adicionales del FCS-MPC. Con el osciloscopio se midieron 2.3 μs. El método que se implementó para esto fue poner un pin de la ZedBoard a 1 cuando la salida del bloque de cálculo llevase 100 ciclos de 10 ns (1 μs) siendo constante. Por esto hay que restar a los 2.3 μs este retraso y el tiempo de conversión del ADC, 960 ns. Quedando un tiempo de cálculo de unos 340 ns, lo que es 7 veces superior aproximadamente al tiempo del diseño con HDL de tipo paralelo (50 ns). Esto es causa de que la herramienta, a la hora de generar el código, es más propensa a usar registros de más para asegurar un funcionamiento estable. Esto explica la gran cantidad de registros usados en comparación con un diseño manual, lo que lleva directamente a un mayor número de retrasos que ralentizan el cálculo. En definitiva, gestionando el mismo volumen de información, si se usan más registros tiene que haber un mayor número de retrasos en el camino de la información. El diseño en HDL de manera secuencial tiene un tiempo de cómputo de 400 ns, ligeramente superior al de este proyecto. En la Figura 6.5 se observan dos señales vistas en el osciloscopio, la señal de fin de cálculo (verde) y la señal de CS del ADC (rosa) que sirve para tener una referencia del inicio del periodo y poder realizar la medida.

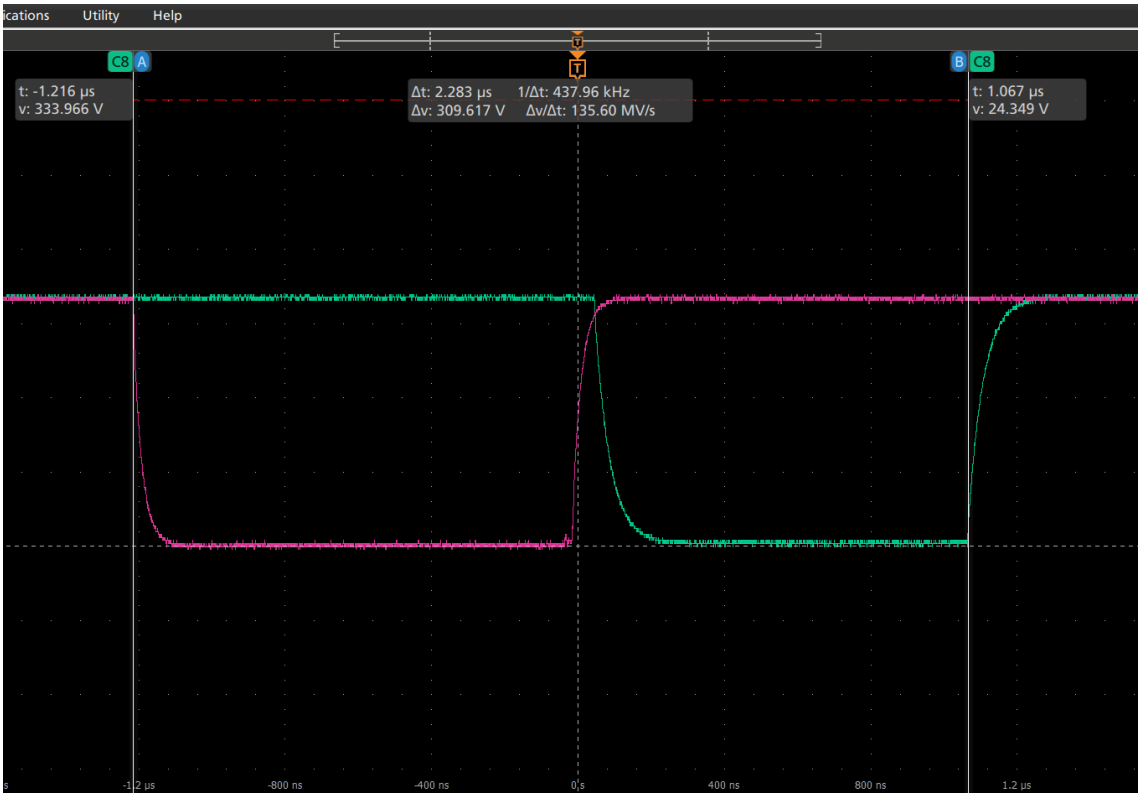

Figura 6.5 Imagen del osciloscopio con la señal de para la medida del tiempo de cálculo (verde) junto con la de CS (rosa) para tener la referencia de cuando empieza el ciclo. Medida tomada desde el flanco de bajada de CS (inicio del periodo) hasta el flanco de subida de la señal verde (fin del cálculo + 1 μs). Contando divisiones de 500 ns en el osciloscopio se tienen 2,3 μs.

7 Conclusiones y lineas de trabajo futuras

7.1 Conclusiones

A la vista de los resultados expuestos en el apartado anterior se puede concluir que, siguiendo la metodología de trabajo propuesta, es posible conseguir resultados casi idénticos en comportamiento a los que se obtendrían realizando el código manualmente. Por contra, los resultados no llegan a ser tan óptimos como un diseño a bajo nivel en cuanto a área y sobretodo tiempo de cálculo. Este hecho era algo que se esperaba antes de empezar el proyecto ya que precisamente el coste de usar herramientas de HLS son diseños menos óptimos y precisos. La ventaja del uso de HDL Coder radica en el tiempo ahorrado en el diseño. Para dar una perspectiva sobre el esfuerzo de diseño, todo el diseño del control MPC se realizó en un día y dos más para la verificación y depuración de errores.

El esfuerzo requerido para aprender a diseñar con HDL Coder puede ser alto debido a que los manuales del Add-on usan ejemplos muy simples y dan explicaciones escuetas sobre la muy amplia variedad de menús y opciones disponibles para el diseñador. Especialmente, al trabajar con un "Support Package" para incluir toda la información y las opciones de un hardware (ZedBoard) y un software (Vivado) de una empresa externa a Matworks (Xilinx) hace que los manuales den información escasa sobre muchas de las opciones disponibles. Por otro lado, HDL Coder está pensado para trabajar o bien con código o bien con bloques propios de Simulink en cuanto a lo que a los manuales respecta. La información acerca de la combinación de ambas opciones es escasa tanto en la bibliografía como en los manuales de Matworks. Aprovechar las opciones del código MATLAB para programar los cálculos y la interfaz gráfica de Simulink para gestionar el flujo de la información de un bloque a otro ha sido la intención en este proyecto y la base del diseño del flujo de trabajo. Con el flujo de trabajo propuesto, se espera que se pueda reducir en gran medida el tiempo requerido para aprender a trabajar con HDL Coder, aunque es cierto que existen muchas opciones dentro de los distintos Add-ons que han quedado fuera del estudio y que podrían ser convenientes y ofrecer mejoras a este.

El entorno de MATLAB/Simulink es bien conocido por alumnos que estén finalizando sus estudios de grado, por lo que una vez entendido el funcionamiento del flujo de trabajo les será sencillo aplicar sus conocimientos previos en este entorno para crear nuevos algoritmos. Poder diseñar una estrategia de control, simularla en Simulink con la toolbox "Simscape Electrical" y poder cargarla en la FPGA sin salir del mismo programa es una gran ventaja. Además, una vez se tiene un diseño funcional, realizar cambios o introducir bloques adicionales es muy sencillo y rápido. Es cierto que con un reloj de sistema alto y con la presencia de bucles en el esquema la inserción de registros intermedios en los cálculos para que se cumplan las restricciones de tiempo en el circuito puede ser un poco tedioso. Para aplicaciones sin bucles y con relojes lentos la herramienta probablemente tenga resultados muy buenos pero para el ámbito de este proyecto, la electrónica de potencia, estos elementos son inevitables. Aun así, con un poco de experiencia se puede preveer cuándo será necesario introducir estos registros.

Poniendo el foco en el uso de esta herramienta para el diseño de controladores de convertidores de potencia, se ha demostrado la viabilidad de la misma para su diseño e implementación. Con unos resultados que en combinación con el diseño de la nueva placa de ADCs hace posible todo el cálculo de un algoritmo FCS-MPC en unos 2 μs haciendo posible llegar hasta un muestreo de 500 kHz en un caso hipotético. Esto significa que HDL Coder, a pesar de no ser posible obtener los resultados más eficiente, tiene la potencialidad de servir como herramienta de prototipado rápido y el diseño de nuevas estrategias de control.

7.2 Lineas de trabajo futuras

En cuanto a una posible continuación de este proyecto se tienen dos vertientes, una más orientada al uso de la estrategia propuesta para hacer un diseño más completo y otra más centrada en explorar las posibilidades de HDL Coder y los add-ons entorno a él para tratar de hacer mejoras sobre la estrategia de diseño.

En cuanto a la primera vertiente los objetivos posibles son:

- Incrementar la frecuencia de muestreo del FCS-MPC y el horizonte de predicción que le saque partido a la buena velocidad de cálculo y a la velocidad de muestreo que ofrece la nueva placa de ADCs
- Montar una topología back-to-back que saque provecho a los canales de la placa de ADC y la capacidad de cálculo en paralelo de la FPGA. Además se probaría a hacer un diseño más complejo en cuanto a su esquema de tiempo por tener diferentes partes funcionando a diferentes frecuencias.
- Montar un convertidor con semiconductores de tecnologías modernas que permiten subir la frecuencia de conmutación manteniendo las pérdidas (Wide band gap). De nuevo se busca exprimir las posibilidades del uso de la ZedBoard en velocidad de calculo y de la placa de ADCs.

En cuanto a la vertiente más dedicada a continuar con el estudio de HDL Coder varios de los frentes que sigue dejando abierto este trabajo son:

- El uso del add-on de "fixed-point designer" para realizar un paso automatizado a variables de punto fijo. Este componente de software ofrece herramientas para realizar un paso automatizado a variables de punto fijo. Estas herramientas no se han podido acabar de explorar debido a que los manuales están enfocados a realizar tareas muy sencillas y que en la práctica resultaba más rápido configurar manualmente cada variable. Estas herramientas funcionan con la idea de determinar los rangos necesarios para las variables por medio de la simulación y así poder asignar una posición al punto binario de manera automática.
- El uso de la función fimth() que en principio debería de configurar cómo elije MATLAB el número de bits y la posición de la coma binaria en los resultados de operaciones con variables de punto fijo. Esto puede suponer un nuevo ahorro en el esfuerzo de diseño puesto que puede ahorrar tener que definir una a una las características de las variables intermedias. Especialmente en proyectos más grandes puede resultar en una ventaja.
- Generación de testbenches desde el propio HDL Coder. El proceso de trabajo con FPGA debe de conllevar una carga de trabajo de verificación importante. Realizar esta verificación vía testbench desde HDL Coder puede ser especialmente relevante, ya que la verificación a través de una simulación en Simulink puede obviar ciertos retrasos dentro de los bloques de código. En proyectos más extensos se espera poder hacer uso de esta herramienta para la verificación.
- Modificar el flujo de trabajo propuesto para poder incluir bloques de código VHDL externos y para poder generar bloques de código que después se usarán en un proyecto completo dentro del flujo normal. Esta parece la solución más directa para sortear las limitaciones que tiene HDL Coder a la hora de optimizar esquemas con bucles. Poder segmentar el proyecto en distintas partes combinado con la generación de testbenches ayudará a la verificación. Para esto la librería de HDL Coder tiene el bloque "Black Box" al que se le permite asignarle un fichero .vhd para generar después el proyecto.

Índice de Figuras

Índice de Tablas

Bibliografía

[1] "European green deal 2050," https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/european-green-deal_en, accessed: 2023-05-15.

[2] "El autoconsumo crece un 108% y hace descender la demanda eléctrica un 4,6%," https://cincodias.elpais.com/companias/2023-04-04/el-autoconsumo-crece-un-108-y-hace-descender-la-demanda-electrica-un-46.html, accessed: 2023-05-20.

[3] J. M. Kharade and N. G. Savagave, "A review of hvdc converter topologies," *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 6, no. 2, pp. 1822–1830, 2017.

[4] S. Ab-Ghani, H. Daniyal, N. Jaalam, N. H. Ramlan, and N. M. Saad, "Time-variant online auto-tuned pi controller using pso algorithm for high accuracy dual active bridge dc-dc converter," in *2022 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS)*. IEEE, 2022, pp. 36–41.

[5] M. Safayatullah, M. T. Elrais, S. Ghosh, R. Rezaii, and I. Batarseh, "A comprehensive review of power converter topologies and control methods for electric vehicle fast charging applications," *IEEE Access*, 2022.

[6] S. Rivera, S. Kouro, S. Vazquez, S. M. Goetz, R. Lizana, and E. Romero-Cadaval, "Electric vehicle charging infrastructure: From grid to battery," *IEEE Industrial Electronics Magazine*, vol. 15, no. 2, pp. 37–51, 2021.

[7] K. Sun, L. Zhang, Y. Xing, and J. M. Guerrero, "A distributed control strategy based on dc bus signaling for modular photovoltaic generation systems with battery energy storage," *IEEE Transactions on Power Electronics*, vol. 26, no. 10, pp. 3032–3045, 2011.

[8] N. Naghizadeh and S. S. Williamson, "A comprehensive review of power electronic converter topologies to integrate photovoltaics (pv), ac grid, and electric vehicles," in *2013 IEEE Transportation Electrification Conference and Expo (ITEC)*, 2013, pp. 1–6.

[9] "Disposición 2198 del boe núm. 42 de 2018 - red eléctrica española," https://www.ree.es/sites/default/files/01_ACTIVIDADES/Documentos/ProcedimientosOperacion/PO_resol_01feb2018.pdf, accessed: 2023-05-21.

[10] H. Wang, M. Liserre, and F. Blaabjerg, "Toward reliable power electronics: Challenges, design tools, and opportunities," *IEEE Industrial Electronics Magazine*, vol. 7, no. 2, pp. 17–26, 2013.

[11] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi *et al.*, "A survey and evaluation of fpga high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2015.

[12] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for fpgas: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

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

[14] "Wikipedia - high level synthesis," https://en.wikipedia.org/wiki/High-level_synthesis, accessed: 2023-05-17.

[15] J. Cong, J. Lau, G. Liu, S. Neuendorffer, P. Pan, K. Vissers, and Z. Zhang, "Fpga hls today: successes, challenges, and opportunities," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 4, pp. 1–42, 2022.

[16] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2017, pp. 152–159.

[17] L. Guo, J. Lau, Z. Ruan, P. Wei, and J. Cong, "Hardware acceleration of long read pairwise overlapping in genome sequencing: A race between fpga and gpu," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 127–135.

[18] T. Bonny, "Chaotic or hyper-chaotic oscillator? numerical solution, circuit design, matlab hdl-coder implementation, vhdl code, security analysis, and fpga realization," *Circuits, Systems, and Signal Processing*, vol. 40, no. 3, pp. 1061–1088, 2021.

[19] 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)*, 2017, pp. 171–174.

[20] E. Liegmann, P. Karamanakos, and R. Kennel, "Real-time implementation of long-horizon direct model predictive control on an embedded system," *IEEE Open Journal of Industry Applications*, vol. 3, pp. 1–12, 2022.

[21] S. A. Bin Khalid, E. Liegmann, P. Karamanakos, and R. Kennel, "High-level synthesis of a long horizon model predictive control algorithm for an fpga," in *PCIM Europe digital days 2020*, 2020, pp. 1–8 .

[22] P. L. Carotenuto, A. Della Cioppa, A. Marcelli, and G. Spagnuolo, "An evolutionary approach to the dynamical reconfiguration of photovoltaic fields," *Neurocomputing*, vol. 170, pp. 393–405, 2015.

[23] G. Petrone, F. Serra, G. Spagnuolo, and E. Monmasson, "Soc implementation of a photovoltaic reconfiguration algorithm by exploiting a hls-based architecture," *Mathematics and Computers in Simulation*, vol. 158, pp. 520–537, 2019.

[24] S. Ciaglia, E. Monmasson, G. Petrone, and G. Spagnuolo, "System-on-chip implementation of a pv dynamical reconfiguration algorithm," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, 2016, pp. 4826–4831.

[25] F. Sánchez, R. Mateos, E. 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*, 2013, pp. 2232–2237.

[26] E. Zafra Ratia, "Desarrollo en plataforma fpsoc de estrategias fcs-mpc para el control de convertidores de potencia," Trabajo Fin de Máster Inédito, 2019.

[27] "c2d - conversión de tiempo continuo-discreto - mathworks," https://es.mathworks.com/help/control/ref/lti.c2d.html, accessed: 2023-05-21.

[28] P. Karamanakos, T. Geyer, and R. Kennel, "On the choice of norm in finite control set model predictive control," *IEEE Transactions on Power Electronics*, vol. 33, no. 8, pp. 7105–7117, 2018.

[29] "Xilinx. zynq-7000 all programmable soc (product brief)," https://docs.xilinx.com/v/u/en-US/ds190-Zynq-7000-Overview, accessed: 2023-05-13.

[30] "Xilinx. 7 series fpgas configurable logic block user guide (ug474)," https://docs.xilinx.com/v/u/en-US/ug474_7Series_CLB, accessed: 2023-05-13.

[31] "User guide dsp48e1," https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1, accessed: 2023-05-15.

[32] "Zedboard users manual," http://zedboard.org/sites/default/files/documentations/ZedBoard, accessed: 2023-05-15.

[33] "Datasheet del sensor la55-p," https://www.farnell.com/datasheets/1449960.pdf, accessed: 2023-05-15.

[34] "Datasheet del sensor lv25-p," https://www.farnell.com/datasheets/1866272.pdf, accessed: 2023-05-15.

[35] R. S. Pengelly, S. M. Wood, J. W. Milligan, S. T. Sheppard, and W. L. Pribble, "A review of gan on sic high electron-mobility power transistors and mmics," *IEEE Transactions on Microwave Theory and Techniques*, vol. 60, no. 6, pp. 1764–1783, 2012.

[36] "Ad7476a datasheet and product info," https://www.analog.com/media/en/technical-documentation/data-sheets/AD7476A_7477A_7478A.pdf, accessed: 2023-05-21.

[37] "Mathworks. which versions of xilinx vivado are supported with which release of hdl coder?—matlab central," https://www.mathworks.com/matlabcentral/answers/518421-which-versions-of-xilinx-vivado-are-supported-with-which-release-of-hdl-coder, accessed: 2023-05-29.

[38] "Mathworks. hdl ip core generation using xilinx vivado fails as of january 1, 2022 —bug reports," https://es.mathworks.com/support/bugreports/2656440, accessed: 2023-05-29.

[39] "Mathworks. how to configure xilinx vivado 2017.2 system generator for matlab2017b?—matlab central," https://www.mathworks.com/matlabcentral/answers/359646-how-to-configure-xilinx-vivado-2017-2-system-generator-for-matlab2017b, accessed: 2023-05-29.

[40] L. Caseiro, D. Caires, and A. Mendes, "Prototyping power electronics systems with zynq-based boards using matlab/simulink—a complete methodology," *Electronics*, vol. 11, no. 7, p. 1130, 2022.

[41] "Vivado 2019.1 - designing with ip," https://www.xilinx.com/support/documentation-navigation/design-hubs/2019-1/dh0003-vivado-designing-with-ip-hub.html, accessed: 2023-05-30.

[42] D. C. Rus, N. S. Preda, I. I. Incze, M. Imecs, and C. Szabó, "Comparative analysis of pwm techniques: Simulation and dsp implementation," in *2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, vol. 3, 2010, pp. 1–6.