Trabajo Fin de Grado
Ingeniería de las Tecnologías Industriales

Desarrollo e implementación de un esquema de control embebido para un convertidor elevador DC/DC aplicado a sistemas fotovoltaicos en entorno Processor-in-the-Loop

Autor: Asunción Pérez Reina
Tutor: Sergio Vázquez Pérez

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2026

Trabajo Fin de Grado
Ingeniería de las Tecnologías Industriales

Desarrollo e implementación de un esquema de control embebido para un convertidor elevador DC/DC aplicado a sistemas fotovoltaicos en entorno Processor-in-the-Loop

Autor:
Asunción Pérez Reina

Tutor:
Sergio Vázquez Pérez
Catedrático de Universidad

Departamento de Ingenería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2026

Trabajo Fin de Grado: Desarrollo e implementación de un esquema de control embebido para un convertidor elevador DC/DC aplicado a sistemas fotovoltaicos en entorno Processor-in-the-Loop

Autor: Asunción Pérez Reina

Tutor: Sergio Vázquez Pérez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2026

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Agradecer a mi familia, en especial a mis padres, por su comprensión y apoyo incondicional. Gracias por confiar siempre en mí, incluso cuando yo dudaba de mis propias capacidades. Gracias por darme la libertad y el tiempo que necesitaba.

A mis compañeros, y ahora amigos, gracias por acompañarme en esta etapa. Ha sido una suerte poder coincidir con ustedes.

Finalmente, agradecer a mis profesores y, especialmente, a mi tutor, por la orientación, dedicación y paciencia durante la realización de este trabajo.

Asunción Pérez Reina

Resumen

Este trabajo aborda el desarrollo e implementación de un sistema de control para un convertidor elevador DC/DC aplicado a un sistema fotovoltaico, empleando una metodología de validación basada en la técnica Processor-in-the-Loop. El objetivo principal es analizar el comportamiento del control cuando este se ejecuta en un microcontrolador, concretamente en el DSP TMS320F28069M de la familia C2000 de Texas Instruments, específicamente diseñado para la ejecución eficiente de algoritmos de control en tiempo real, mientras la planta permanece en el entorno de simulación.

Para ello, se parte del modelado del generador fotovoltaico y del convertidor, sobre los que se diseña una estrategia de control en cascada. Esta se compone de un lazo externo asociado al seguimiento del punto de máxima potencia y de un lazo interno encargado de regular la corriente del convertidor. La implementación del control se realiza mediante generación automática de código y su posterior ejecución en el sistema embebido.

Asimismo, se integran en el entorno PIL los bloques necesarios para la interacción entre el modelo y el hardware de control, incluyendo la configuración de periféricos como el ADC y el PWM. Los resultados obtenidos permiten validar el funcionamiento de la estrategia planteada y ponen de manifiesto el interés del entorno PIL como herramienta para el desarrollo y verificación de sistemas de electrónica de potencia aplicados al ámbito fotovoltaico.

Abstract

This work presents the development and implementation of a control system for a DC/DC boost converter applied to a photovoltaic system, using a validation methodology based on the Processor-in-the-Loop technique. The main objective is to analyse the behaviour of the control when it is executed on a microcontroller, specifically on the DSP TMS320F28069M from the C2000 family by Texas Instruments, specifically designed for the efficient execution of real-time control algorithms, while the plant remains in the simulation environment.

To this end, the photovoltaic generator and the converter are modelled, on which a cascaded control strategy is designed. This consists of an outer loop associated with maximum power point tracking and an inner loop responsible for regulating the converter current. The control is implemented through automatic code generation and its subsequent execution on the embedded system.

Likewise, the necessary blocks are integrated into the PIL environment to enable interaction between the model and the control hardware, including the configuration of peripherals such as the ADC and PWM. The obtained results validate the operation of the proposed strategy and highlight the relevance of the PIL environment as a tool for the development and verification of power electronics systems applied to the photovoltaic field.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

A	Amperios
V	Voltios
Ω	Ohmios
W	Watios
DC	Corriente continua
AC	Corriente alterna
V	Tensión eléctrica
I	Corriente eléctrica
P	Potencia eléctrica
R	Resistencia
L	Inductancia
C	Capacitancia
V_{in}	Tensión de entrada del convertidor
V_{out}	Tensión de salida del convertidor
V_{pv}	Tensión de la planta FV
I_{pv}	Corriente de la planta FV
P_{pv}	Potencia de la planta FV
D	Ciclo de trabajo/duty
f_s	Frecuencia de conmutación
MPP	Punto de máxima potencia
MPPT	Maximum Power Point Tracking
PWM	Modulación por ancho de pulso
ADC	Convertidor analógico-digital
PIL	Processor-in-the-Loop
cos	Función coseno
sen	Función seno
tg	Función tangente

1 INTRODUCCIÓN

La transición hacia un modelo energético más sostenible constituye uno de los principales retos tecnológicos y medioambientales actuales. En este contexto, las fuentes de energía renovable han adquirido un papel estratégico, tanto por su contribución a la reducción de emisiones de dióxido de carbono como por su capacidad para disminuir la dependencia energética exterior mediante el aprovechamiento de recursos autóctonos como la radiación solar o el viento. Además, la evolución del sistema eléctrico está favoreciendo esquemas de generación más distribuidos, basados en múltiples fuentes de menor potencia próximas a los puntos de consumo, lo que permite reducir pérdidas de transporte y avanzar hacia una red más flexible y eficiente [1].

Dentro de este escenario, la energía solar fotovoltaica destaca como una de las tecnologías con mayor proyección, debido a su modularidad, escalabilidad y facilidad de integración en entornos de generación distribuida [1]. Su principio de funcionamiento se basa en el efecto fotovoltaico, es decir, en la conversión de la energía asociada a la radiación luminosa en energía eléctrica mediante materiales semiconductores [2]. El comportamiento de los generadores fotovoltaicos depende, sin embargo, de las condiciones de operación, especialmente de la irradiancia y de la temperatura, lo que hace necesario un adecuado acondicionamiento electrónico para garantizar un funcionamiento eficiente en cada instante [3].

En los sistemas fotovoltaicos conectados a red, la electrónica de potencia desempeña, por tanto, un papel fundamental. Por un lado, los convertidores permiten adaptar el punto de operación del generador para extraer la máxima potencia disponible; por otro, hacen posible la transferencia de esa energía hacia la red eléctrica con las condiciones adecuadas de tensión, corriente y sincronización. En este marco, el presente trabajo se centra en el estudio, modelado, control e implantación de un sistema fotovoltaico conectado a red, prestando especial atención al convertidor, a la estrategia de seguimiento del punto de máxima potencia y a la validación del control en un entorno próximo a la implementación embebida [4].

1.1. Contexto energético y papel de las energías renovables

En las últimas décadas, el sistema energético mundial ha experimentado una transformación progresiva impulsada por la necesidad de reducir el impacto ambiental asociado al uso intensivo de combustibles fósiles. Entre los principales factores que motivan este cambio destacan, por un lado, la preocupación por las emisiones de gases de efecto invernadero y su contribución al cambio climático y, por otro, la necesidad de disminuir la dependencia energética exterior mediante el aprovechamiento de recursos autóctonos. En este escenario, las energías renovables han adquirido una relevancia creciente como alternativa capaz de contribuir a un modelo energético más sostenible, diversificado y compatible con los objetivos de descarbonización a largo plazo.

Dentro de este proceso de transición energética, la electricidad está llamada a desempeñar un papel cada vez más importante como vector energético, lo que refuerza la necesidad de incrementar la participación de fuentes renovables en la generación eléctrica. En este sentido, tecnologías como la eólica y la solar fotovoltaica se han consolidado como opciones estratégicas debido a la disponibilidad de sus recursos primarios, su carácter inagotable a escala humana y su capacidad de integración en distintos niveles del sistema eléctrico. Además, el desarrollo de estas tecnologías no solo responde a criterios medioambientales, sino también a razones técnicas y económicas vinculadas a la seguridad de suministro, a la diversificación del mix energético y a la progresiva electrificación de la demanda [1].

De forma paralela, la evolución del sistema eléctrico está favoreciendo un cambio desde esquemas tradicionales de generación centralizada hacia modelos con una mayor presencia de recursos distribuidos. En este nuevo contexto, las energías renovables permiten plantear instalaciones de menor potencia próximas a los puntos de consumo, reduciendo pérdidas en el transporte de energía y aportando mayor flexibilidad a la red. Este marco resulta especialmente favorable para la expansión de la energía solar fotovoltaica, cuya modularidad y escalabilidad facilitan su integración tanto en aplicaciones de pequeña potencia como en instalaciones de mayor

entidad conectadas a red. Así, el estudio de los sistemas fotovoltaicos se sitúa dentro de una línea de desarrollo plenamente alineada con las tendencias actuales de transición energética y generación distribuida. Esta situación puede apreciarse en la Figura 1-1, donde se muestra la distribución del consumo mundial de energía primaria por fuentes energéticas.

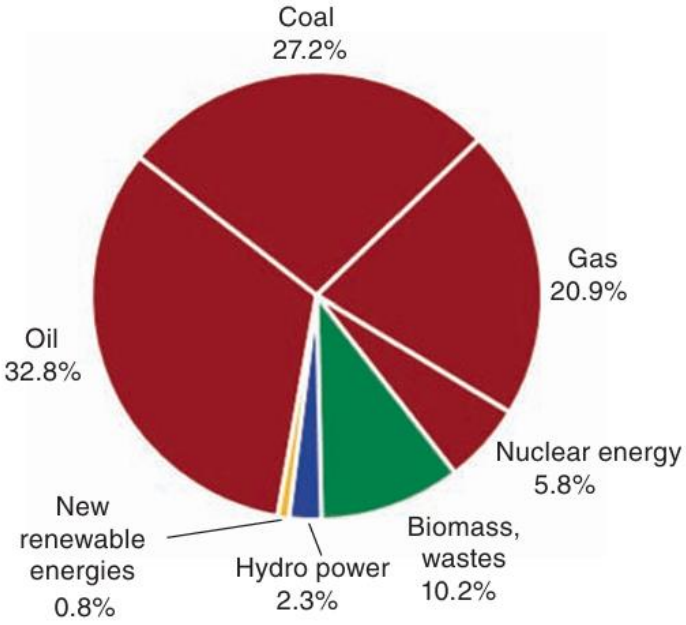

Figura 1-1. Distribución del consumo mundial de energía primaria por fuentes energéticas [5]

1.2. La energía solar fotovoltaica en el marco de la generación distribuida

Dentro del conjunto de tecnologías renovables, la energía solar fotovoltaica ocupa una posición especialmente relevante por su facilidad de integración en esquemas de generación distribuida. Frente al modelo convencional de generación centralizada, basado en grandes plantas alejadas de los puntos de consumo y en un flujo energético esencialmente unidireccional, la generación distribuida plantea la incorporación de múltiples unidades de menor potencia próximas a la demanda, con la consiguiente reducción de pérdidas de transporte y una mayor flexibilidad en la explotación de la red eléctrica. En este contexto, la tecnología fotovoltaica resulta particularmente adecuada, ya que permite desplegar sistemas de muy distinta escala, desde pequeñas instalaciones de autoconsumo hasta plantas de mayor potencia conectadas a red. La integración de este tipo de recursos en la red eléctrica se representa de forma esquemática en la Figura 1-2.

La idoneidad de la energía fotovoltaica para este tipo de aplicaciones se debe, en gran medida, a su carácter modular. La unidad básica de conversión es la célula fotovoltaica, que puede agruparse en módulos y paneles para adaptar la potencia instalada a las necesidades de cada aplicación. Esta estructura escalable facilita su implantación tanto en entornos urbanos como industriales, así como su integración progresiva en redes eléctricas con una presencia creciente de recursos energéticos descentralizados. De este modo, la generación fotovoltaica no solo contribuye a incrementar la participación de fuentes renovables en el mix energético, sino que también encaja de forma natural en la evolución hacia sistemas eléctricos más distribuidos y más próximos al punto de consumo.

Además, cuando la instalación fotovoltaica opera conectada a red, su integración requiere una etapa de conversión electrónica capaz de adaptar la energía generada en corriente continua a las condiciones de la red en corriente alterna. En este sentido, el inversor fotovoltaico actúa como el elemento de enlace entre el generador y la red, permitiendo la evacuación de potencia y asegurando un funcionamiento adecuado del sistema. Por ello, el estudio de la energía solar fotovoltaica en el ámbito de la generación distribuida no puede separarse del análisis de la electrónica de potencia asociada, aspecto que adquiere una importancia central en los sistemas conectados a red y que constituye una de las bases del presente trabajo.

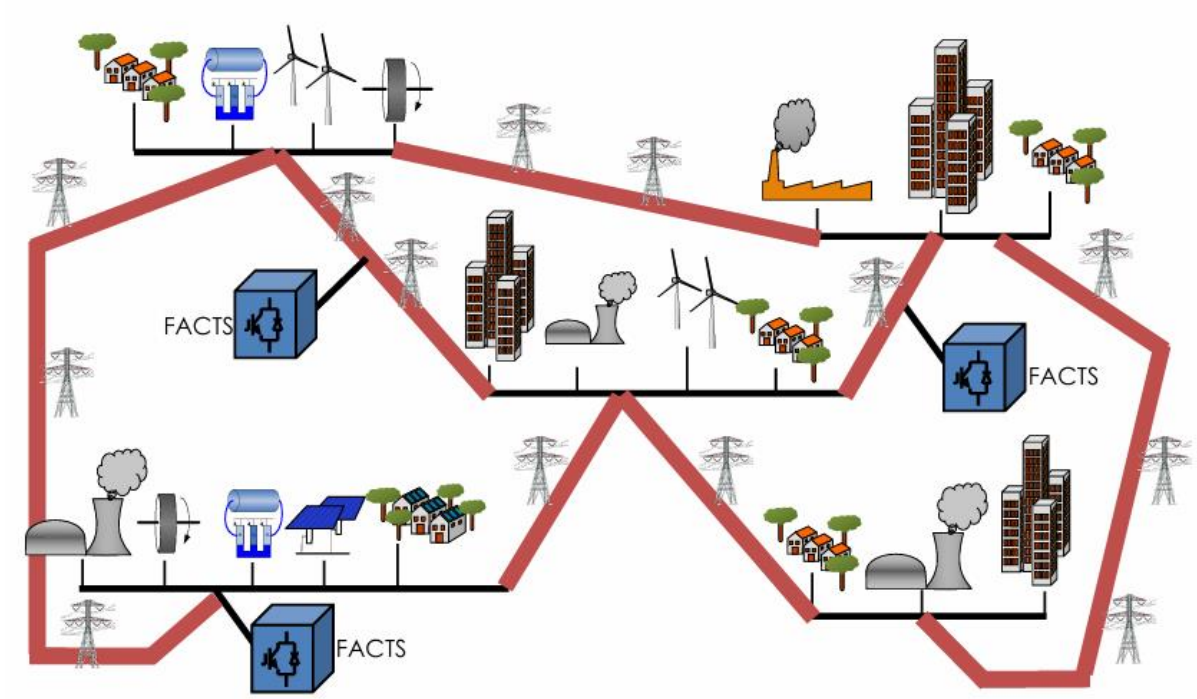

Figura 1-2. Integración de la energía solar fotovoltaica en esquemas de generación distribuida [6]

1.3. Fundamento de la conversión fotovoltaica

La conversión fotovoltaica se basa en la capacidad de determinados materiales semiconductores para transformar parte de la energía asociada a la radiación luminosa en energía eléctrica. Este fenómeno, conocido como efecto fotovoltaico, constituye el principio físico de funcionamiento de las células solares y permite obtener una diferencia de potencial cuando el material es iluminado [2].

Desde el punto de vista físico, cuando la radiación incide sobre el semiconductor, pueden producirse distintos fenómenos: una parte de los fotones puede atravesar el material, otra parte puede ser reflejada y otra puede ser absorbida. Solo esta última contribuye al proceso de conversión, y lo hace de manera efectiva cuando la energía del fotón es suficiente para promover electrones desde la banda de valencia hasta la banda de conducción, generando así pares electrón-hueco. La magnitud que describe la potencia radiante incidente por unidad de superficie es la irradiancia, normalmente expresada en W/m^2, y su valor influye directamente en la capacidad de generación de la célula fotovoltaica.

Para que esos portadores generados den lugar a una corriente útil, no basta con la absorción de radiación, sino que es necesaria además una estructura interna que cree un campo eléctrico capaz de separar las cargas. Esta función la cumple la unión p-n, formada por el contacto entre dos regiones de semiconductor dopadas de manera diferente. Bajo iluminación, el campo eléctrico asociado a dicha unión favorece la separación de electrones y huecos y orienta su desplazamiento hacia los contactos del dispositivo, permitiendo así el establecimiento de una corriente eléctrica en un circuito externo [5].

En consecuencia, el funcionamiento de una célula fotovoltaica puede entenderse como el resultado de dos requisitos fundamentales: por un lado, la existencia de una unión p-n que genere el campo eléctrico interno y, por otro, la incidencia de radiación con energía suficiente para producir pares electrón-hueco en el semiconductor. A partir de esta base física se desarrolla la célula fotovoltaica como unidad elemental de conversión, cuyo comportamiento eléctrico puede caracterizarse mediante sus curvas corriente-tensión y potencia-tensión. Este principio de funcionamiento se representa esquemáticamente en la Figura 1-3.

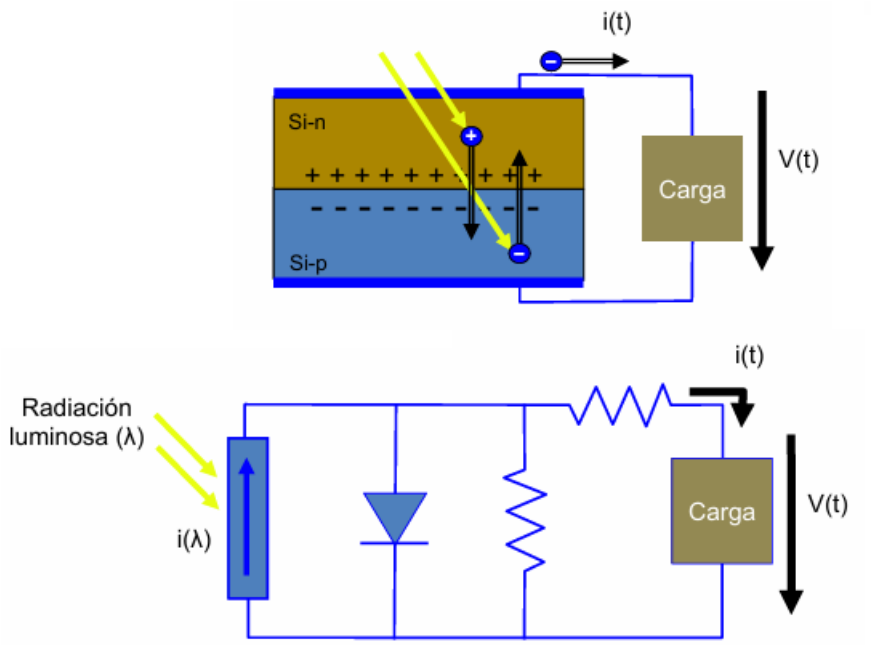

Figura 1-3. Principio de funcionamiento de una célula fotovoltaica basado en la conversión de la radiación luminosa en energía eléctrica [6]

1.4. Célula, módulo y características eléctricas del generador fotovoltaico

La célula fotovoltaica constituye la unidad elemental de conversión de energía solar en energía eléctrica. Sin embargo, debido a la reducida tensión y potencia que puede suministrar individualmente, en la práctica varias células se conectan eléctricamente para formar un módulo o panel fotovoltaico, capaz de proporcionar niveles de tensión y potencia adecuados para su utilización en aplicaciones reales. De este modo, el generador fotovoltaico se construye a partir de la asociación de múltiples células, manteniendo el mismo principio físico de funcionamiento, pero adaptando su capacidad energética a los requisitos del sistema. La relación entre la célula fotovoltaica como unidad elemental y el módulo como asociación de múltiples células se ilustra en la Figura 1-4.

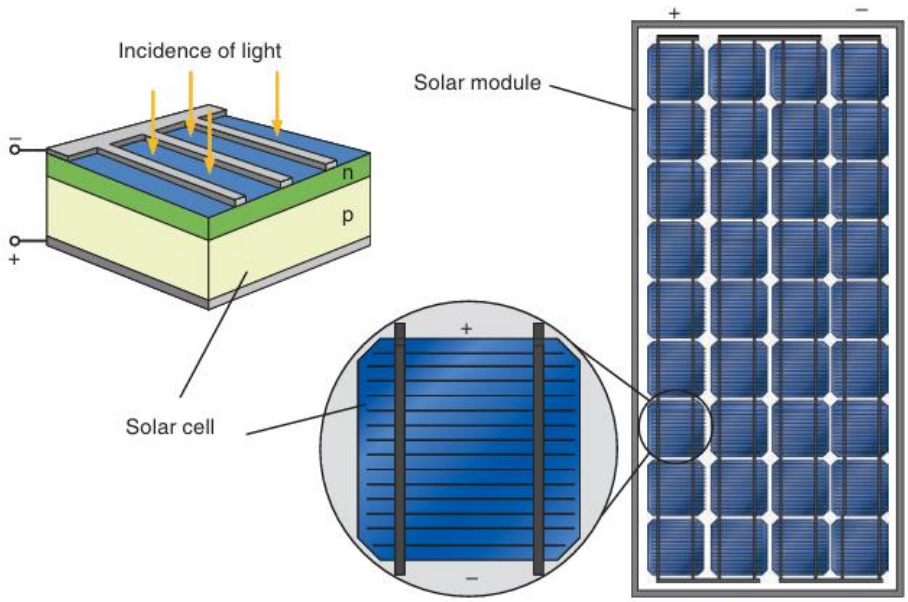

Figura 1-4. Célula fotovoltaica y módulo solar como unidades básicas de un sistema fotovoltaico [5]

Desde el punto de vista eléctrico, el comportamiento de una célula o de un panel fotovoltaico se representa habitualmente mediante su curva característica corriente-tensión (I-V). Esta curva permite identificar magnitudes fundamentales como la tensión de circuito abierto V_{OC}, la corriente de cortocircuito I_{SC}, la tensión en el punto de máxima potencia V_{MPP} y la corriente en el punto de máxima potencia I_{MPP}. A partir de ellas se define la potencia máxima P_{MPP}, que corresponde al punto de operación en el que el producto entre tensión y corriente alcanza su valor máximo. La identificación de estas magnitudes sobre las curvas características I-V y P-V se muestra en la Figura 1-5. La representación complementaria de la curva potencia-tensión (P-V) permite visualizar de forma más directa ese punto de máxima potencia, cuya localización depende de las condiciones de operación del generador.

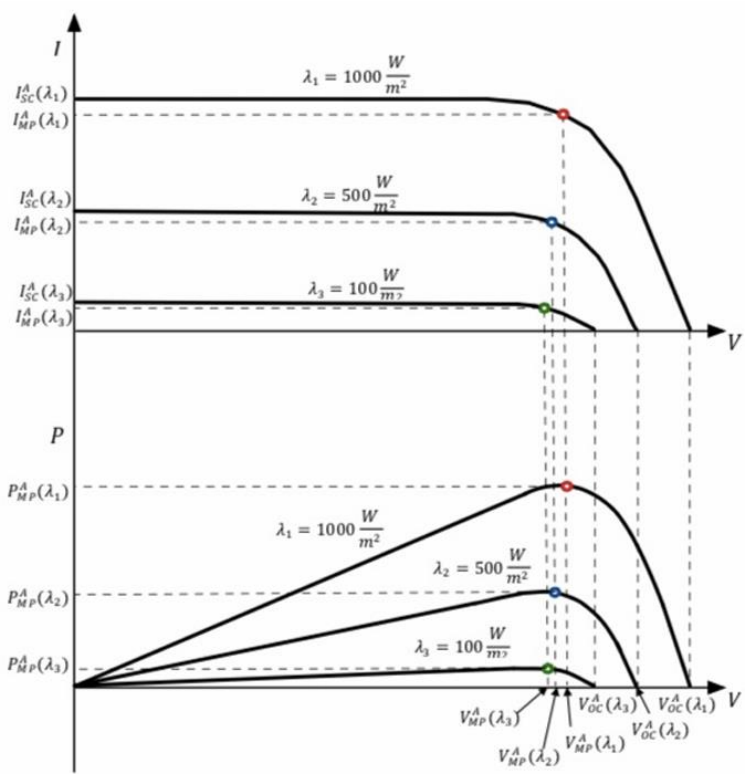

Figura 1-5. Curvas características I-V y P-V para distintos niveles de irradiancia a temperatura constante

Estas características no son constantes, sino que varían principalmente con la irradiancia y con la temperatura. En general, un aumento de la irradiancia incrementa la capacidad de generación del panel y eleva la potencia máxima disponible, mientras que la temperatura modifica la forma de las curvas características y suele provocar una reducción de la potencia máxima extraíble. Por tanto, el punto de funcionamiento óptimo del generador no permanece fijo, sino que se desplaza en función de las condiciones ambientales. Esta dependencia justifica la necesidad de emplear estrategias de seguimiento del punto de máxima potencia en los sistemas fotovoltaicos, especialmente cuando se pretende maximizar el aprovechamiento energético de la instalación [3].

Además de los parámetros anteriores, la calidad eléctrica de una célula o módulo puede evaluarse mediante el factor de forma, o fill factor FF, definido como el cociente entre la potencia máxima entregable y el producto $V_{OC}I_{SC}$. Este parámetro proporciona una medida de la "rectangularidad" de la curva I-V y, en consecuencia, de la aptitud del dispositivo para transformar la energía recibida en potencia eléctrica útil. Junto con el rendimiento de conversión, constituye uno de los indicadores más empleados para caracterizar el comportamiento de las tecnologías fotovoltaicas.

1.5. Principales tecnologías de células fotovoltaicas

Las tecnologías fotovoltaicas pueden clasificarse, de forma general, en función del material semiconductor empleado y del proceso de fabricación de la célula. Entre las tecnologías más extendidas en aplicaciones comerciales destacan las basadas en silicio cristalino, tanto monocristalino como policristalino, así como las tecnologías de capa fina, entre las que se encuentra el silicio amorfo. Cada una de ellas presenta características

diferenciadas en términos de rendimiento, coste y complejidad de fabricación, lo que condiciona su idoneidad para distintas aplicaciones [5].

Las células monocristalinas se fabrican a partir de silicio de elevada pureza, lo que da lugar a una estructura cristalina más uniforme y, en consecuencia, a mejores prestaciones eléctricas. De forma orientativa, pueden alcanzarse rendimientos del orden del 24% en laboratorio y del 15-18% en campo [6]. Además, suelen identificarse visualmente por sus tonalidades azules uniformes y por la presencia visible del conexionado metálico.

Por su parte, las células policristalinas también pertenecen a la familia del silicio cristalino, aunque presentan una estructura menos homogénea. Esto se traduce en rendimientos algo inferiores, situados típicamente en torno al 19-20% en laboratorio y al 12-14% en campo, si bien su proceso de fabricación resulta más sencillo [6]. Desde el punto de vista visual, se caracterizan por una superficie en la que pueden apreciarse distintos cristales y variaciones de tonalidad azul.

En el caso del silicio amorfo, representativo de las tecnologías de capa fina, el material semiconductor se deposita como una lámina delgada sobre un sustrato, habitualmente de vidrio o plástico. Esta solución permite configuraciones ligeras y superficies visualmente homogéneas, aunque con prestaciones inferiores a las de las tecnologías cristalinas. De manera orientativa, sus rendimientos se sitúan en torno al 16% en laboratorio y por debajo del 10% en campo [6].

En conjunto, las tecnologías de silicio cristalino han alcanzado una posición claramente dominante en el mercado fotovoltaico. De acuerdo con la distribución del mercado mundial de paneles PV en 2015 [6], representaban aproximadamente el 93% del mercado mundial, frente a una participación mucho menor de las tecnologías de capa fina; dentro del silicio cristalino, la tecnología monocristalina ocupaba la mayor cuota, con alrededor del 69%, seguida de la policristalina, con aproximadamente el 24%.

Asimismo, en los últimos años se han incorporado mejoras constructivas orientadas al aumento de eficiencia, como las configuraciones multi-busbar (MBB), half-cut (HC) o PERC [6], que permiten reducir pérdidas, mejorar el aprovechamiento de la radiación incidente y elevar el rendimiento del módulo fotovoltaico.

A modo de síntesis, la Tabla 1-1 resume las diferencias más relevantes entre las principales tecnologías fotovoltaicas en términos de rendimiento, características generales y proceso de fabricación.

Tecnología	Rendimiento en laboratorio	Rendimiento en campo	Características generales	Fabricación
Monocristalina	24 %	15-18 %	Presenta tonalidades azules uniformes y conexionado visible. Ofrece, en general, mayores prestaciones eléctricas.	Se obtiene a partir de silicio puro fundido y dopado con boro.
Policristalina	19-20 %	12-14 %	Se aprecian los cristales y varios tonos azules en la superficie. Su rendimiento es algo inferior al de la tecnología monocristalina.	Similar a la monocristalina, aunque con un proceso de fabricación más sencillo.
Amorfa	16 %	< 10 %	Presenta una coloración homogénea marrón y no se aprecian conexiones visibles.	Se deposita como una lámina delgada sobre un sustrato de vidrio o plástico.

Tabla 1-1. Comparación general entre las principales tecnologías de células fotovoltaicas

1.6. Convertidores electrónicos en sistemas fotovoltaicos conectados a red

En los sistemas fotovoltaicos conectados a red, la electrónica de potencia desempeña una función esencial, ya que el generador fotovoltaico entrega energía en corriente continua, mientras que la red eléctrica opera en corriente alterna. Por ello, resulta necesario incorporar convertidores capaces de adaptar la forma, el nivel y las condiciones de la energía generada para hacer posible su transferencia a la red en condiciones adecuadas de funcionamiento. En este contexto, el inversor fotovoltaico de conexión a red actúa como un convertidor DC/AC que enlaza la salida del campo de paneles con la red eléctrica, comportándose como una fuente de corriente sincronizada con la tensión de red. Además de evacuar la potencia generada, debe hacerlo cumpliendo los requisitos de operación exigidos por la red eléctrica. La estructura general de una instalación fotovoltaica conectada a red se muestra en la Figura 1-6.

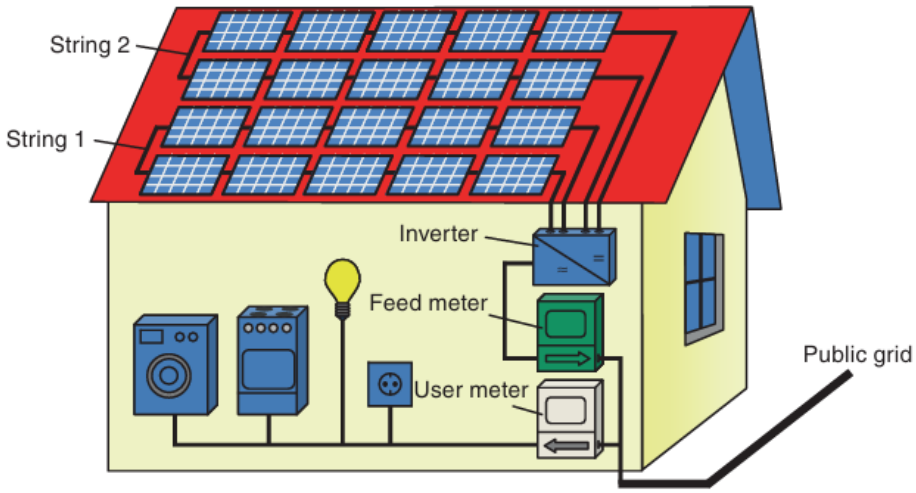

Figura 1-6. Estructura general de una instalación fotovoltaica conectada a red

La necesidad de estos convertidores no responde únicamente a la conversión DC/AC. Desde el punto de vista energético, el generador fotovoltaico presenta un punto de operación óptimo que depende de la irradiancia y de la temperatura. Si el módulo o el campo fotovoltaico se conecta directamente a la carga, su punto de funcionamiento no coincide necesariamente con el punto de máxima potencia, de modo que parte de la energía disponible no se aprovecha. La incorporación de un convertidor DC/DC permite desacoplar eléctricamente el generador de la etapa de salida y ajustar su tensión de funcionamiento para mantenerlo próximo al punto de máxima potencia en diferentes condiciones ambientales. Esta función resulta especialmente relevante en sistemas donde se persigue maximizar el rendimiento energético de la instalación. Esta idea se ilustra en la Figura 1-7, donde se observa cómo el convertidor DC/DC permite desplazar el punto de operación del generador hacia el punto de máxima potencia.

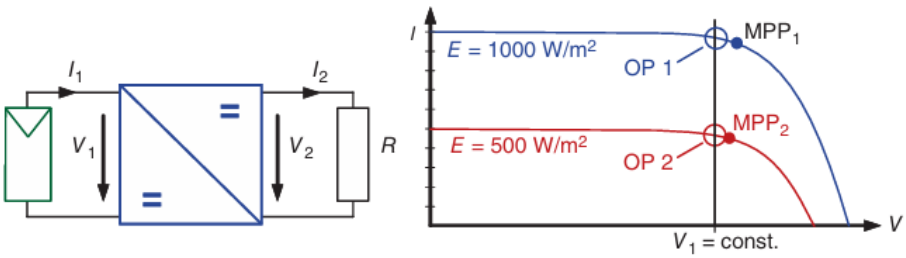

Figura 1-7. Influencia del convertidor DC/DC en el punto de operación del generador fotovoltaico y en el seguimiento del punto de máxima potencia

De forma general, las topologías de conversión empleadas en sistemas fotovoltaicos conectados a red pueden agruparse en dos grandes categorías: sistemas de una sola etapa, en los que el inversor DC/AC asume simultáneamente la adaptación a red y el control del punto de operación del generador, y sistemas de dos etapas, en los que un convertidor DC/DC previo se encarga del seguimiento del punto de máxima potencia y, en su caso, de la elevación del nivel de tensión, mientras que el inversor DC/AC realiza la inyección de corriente a

red. Asimismo, estas topologías pueden clasificarse según dispongan o no de aislamiento galvánico, así como por su nivel de potencia y configuración monofásica o trifásica. En términos generales, para potencias reducidas son habituales las configuraciones monofásicas, mientras que en niveles superiores de potencia predominan las soluciones trifásicas [5].

En particular, en las configuraciones con etapa DC/DC elevadora, dicho convertidor permite no solo imponer el punto de operación del generador fotovoltaico, sino también adaptar la tensión continua a un valor más adecuado para la etapa inversora posterior. A continuación, el inversor DC/AC, gobernado mediante técnicas de modulación, genera la corriente alterna de salida e incorpora elementos de filtrado para reducir el contenido armónico. De este modo, la arquitectura global del sistema combina la extracción eficiente de potencia del campo fotovoltaico con la adecuada transferencia de energía hacia la red eléctrica.

1.7. Alcance, objetivos y metodología del trabajo

A partir del marco descrito en los apartados anteriores, el presente trabajo se centra en el estudio de un sistema fotovoltaico conectado a red, prestando especial atención al modelado del generador fotovoltaico, al diseño del convertidor electrónico asociado y a la estrategia de control necesaria para garantizar una extracción eficiente de potencia y una adecuada transferencia de energía hacia la red eléctrica. Dentro de este contexto, el análisis se orienta al comportamiento del sistema en régimen dinámico y a la interacción entre la planta fotovoltaica, la etapa de conversión y los algoritmos de control implementados.

De forma más concreta, el trabajo aborda el desarrollo de un modelo del sistema en un entorno de simulación, el estudio de sus principales variables eléctricas y el diseño de los lazos de control asociados al convertidor. Asimismo, se considera el seguimiento del punto de máxima potencia como parte esencial de la operación del generador fotovoltaico, dado que las condiciones ambientales modifican continuamente su punto de funcionamiento óptimo. Todo ello permite disponer de una base coherente para analizar el comportamiento global del sistema y evaluar la respuesta obtenida ante distintas condiciones de operación.

Junto al desarrollo del modelo y del control, una parte relevante del trabajo consiste en trasladar la validación del sistema a un entorno de ejecución embebida más representativo. Con este propósito, se incorpora una etapa de verificación mediante un enfoque Processor-in-the-Loop (PIL) [8], en el que el algoritmo de control se ejecuta sobre el microcontrolador de destino mientras la planta permanece en simulación. Este procedimiento permite reducir la distancia entre la validación puramente software y la ejecución del control sobre la plataforma embebida seleccionada, al posibilitar la comprobación del código de control en un entorno más representativo de su funcionamiento.

Desde el punto de vista metodológico, el trabajo se estructura como una secuencia progresiva que parte del análisis teórico del sistema fotovoltaico, continúa con su modelado y con el diseño de la estrategia de control, y culmina con su validación en un entorno de simulación avanzada orientado a la ejecución embebida. De este modo, se combina el estudio de los fundamentos físicos y eléctricos del sistema con una aproximación práctica centrada en su implementación y verificación.

2 MODELO DEL SISTEMA

En este segundo capítulo se describe el modelo de la planta que se empleará en las simulaciones. En primer lugar, se ofrece una visión general del sistema y, posteriormente, se detalla el modelado de la planta fotovoltaica y del convertidor elevador. Asimismo, se presentan las simulaciones realizadas para verificar el correcto funcionamiento de los distintos modelos.

El propósito de este capítulo no es profundizar en la formulación de cada componente ni explicar el uso de la herramienta de simulación, sino presentar el modelo de partida que servirá como base para el desarrollo a partir de la simulación inicial.

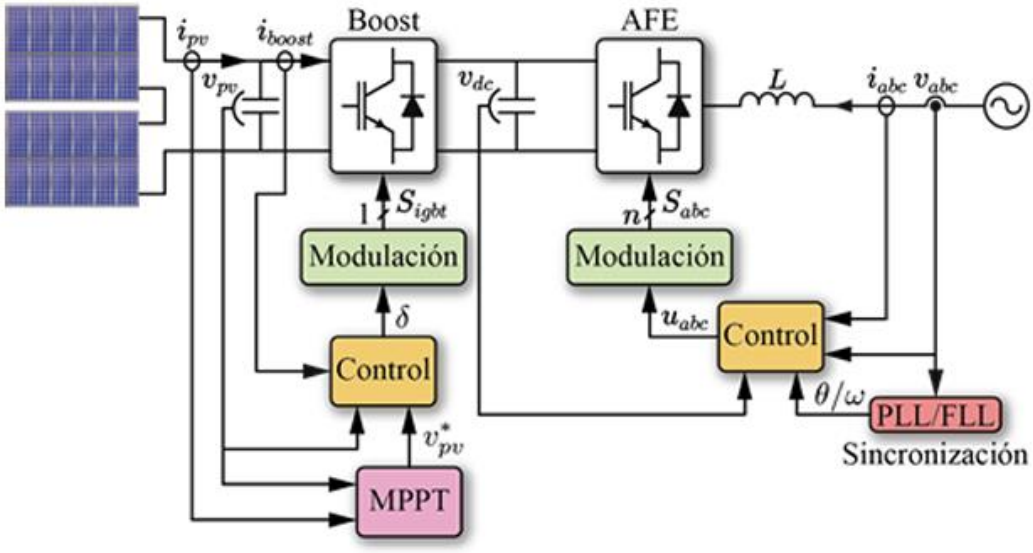

Figura 2-1. Esquema básico de un sistema fotovoltaico conectado a red y el diagrama de bloques para su control [14]

En la Figura 2-1 se muestra la arquitectura considerada para un sistema fotovoltaico conectado a red de doble etapa, junto con el diagrama de bloques de su control. El sistema está formado por un generador fotovoltaico que entrega energía en continua (v_{pv}, i_{pv}), un convertidor elevador DC/DC (Boost) y un inversor trifásico conectado a red (AFE). El convertidor Boost adapta el punto de operación del generador y eleva la tensión hacia el bus de continua, cuya tensión v_{dc} se estabiliza mediante el condensador del enlace DC, actuando como elemento de desacoplo energético entre la etapa DC/DC y la etapa DC/AC. Posteriormente, el inversor AFE convierte la energía del bus de continua en corrientes trifásicas i_{abc} que se inyectan a la red a través de una inductancia L, empleada como filtro y elemento de acoplamiento para reducir el rizado y facilitar el control de corriente.

Desde el punto de vista de control, el sistema se estructura en dos niveles. En un nivel superior se sitúa el algoritmo de seguimiento del punto de máxima potencia (MPPT), que a partir de las medidas del campo fotovoltaico determina la referencia de operación (v_{pv}^*) asociada a dicho punto, variable con las condiciones ambientales. Esta referencia se aplica al control del convertidor Boost, que ajusta su ciclo de trabajo δ y, mediante modulación PWM, genera la señal de disparo del interruptor de potencia. En paralelo, la etapa de conversión DC/AC dispone de su propio lazo de control y modulación: el inversor regula las corrientes inyectadas a la red a partir de v_{abc} e i_{abc}, utilizando un bloque de sincronización PLL/FLL que estima el ángulo θ y la frecuencia ω de la red. Con ello se garantiza la inyección de energía de forma sincronizada y con la calidad requerida, mientras el enlace DC permite compatibilizar la dinámica de extracción de potencia del generador FV con la evacuación de energía hacia la red.

2.1. Planta FV

La planta analizada, cuya configuración eléctrica se muestra en la Figura 2-2, está formada por un string de 15 paneles conectados en serie. Cada panel corresponde al módulo fotovoltaico Kyocera Solar KC200GT, constituido por 54 células conectadas en serie.

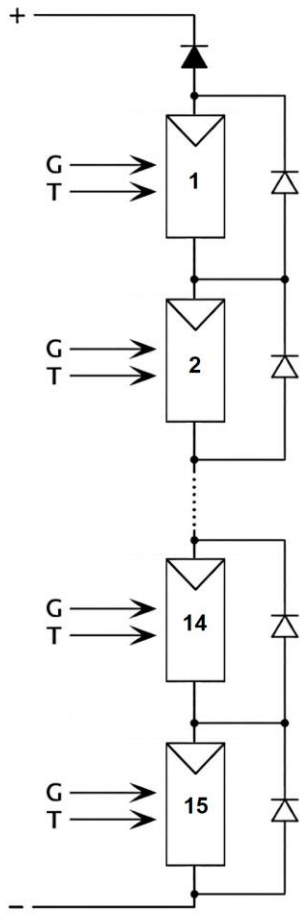

Figura 2-2. Planta FV

Al conectar los 15 paneles en serie, el string se comporta eléctricamente como una fuente de continua cuya tensión total resulta de la suma de las tensiones de cada panel, mientras que la corriente es la misma a través de todos ellos:

$$V_{string} = \sum_{k=1}^{15} V_k \, , I_{string} = I_1 = I_2 = \ldots = I_{15} \qquad (2-1)$$

donde V_k representa la tensión del panel k.

El hecho de que, en una conexión en serie, la corriente sea idéntica en todos los módulos hace que el string presente una elevada sensibilidad frente a desajustes entre paneles, tales como sombras parciales, suciedad, dispersión de parámetros, envejecimiento o diferencias de temperatura. Si un panel queda limitado y no puede suministrar la corriente impuesta por el resto del string, su punto de operación se desplaza y puede llegar a polarizarse en inversa, dando lugar a una tensión negativa.

Para evitar que un panel limitado se convierta en una carga y disipe potencia, con el consiguiente riesgo de aparición de hot-spots, se emplean diodos de bypass. Su papel es proporcionar un camino alternativo para la corriente del string cuando un panel no puede sostener la corriente circulante. En condiciones normales, el panel entrega tensión positiva y el diodo de bypass permanece polarizado en inversa, por lo que no conduce. Sin embargo, bajo condiciones severas de sombra o desajuste, el panel puede tender a entrar en inversa; cuando

dicha tensión inversa supera el umbral del diodo, este conduce y desvía la corriente alrededor del panel afectado.

Como consecuencia, el empleo de diodos de bypass produce un doble efecto. Por un lado, desde el punto de vista de la protección, limita la tensión inversa aplicada sobre el panel o sus células, reduciendo la disipación localizada de potencia y mitigando el riesgo de aparición de hot-spots. Por otro lado, desde el punto de vista operativo, permite que el string continúe conduciendo corriente, aunque con una reducción de tensión aproximadamente equivalente a la aportación del panel que ha quedado puenteado, además de la caída de tensión directa del propio diodo.

Bajo sombras parciales, la activación selectiva de diodos de bypass puede provocar una curva P-V con múltiples máximos locales, lo que influye en el comportamiento del MPPT. Asimismo, además de los diodos de bypass, se dispone de un diodo de bloqueo en serie con la hilera con el fin de evitar corrientes inversas desde el bus de continua hacia la planta en determinadas situaciones, como durante la noche o bajo condiciones de irradiancias muy bajas, cuando el generador no produce energía y podría llegar a absorberla, así como en presencia de otras fuentes conectadas al mismo bus de continua, tales como otras ramas fotovoltaicas, baterías o convertidores. La incorporación de este diodo impide que el string se comporte como una carga, reduciendo las pérdidas y evitando la aparición de polarización inversa no deseada en los módulos.

2.1.1 Modelo equivalente del módulo fotovoltaico

Para el modelado de cada módulo se ha utilizado como base uno de los bloques proporcionados por una demostración del software, adaptándolo y parametrizándolo, con el fin de emular el comportamiento eléctrico del módulo fotovoltaico Kyocera Solar KC200GT, cuya ficha técnica se puede consultar en [12]. En la representación adoptada, el subsistema equivalente se caracteriza por $N_s = 54$, correspondiente al número de células conectadas en serie, y $N_p = 5$, correspondiente al número de ramas equivalentes conectadas en paralelo consideradas en el modelo. El parámetro N_s interviene directamente en el cálculo de la tensión térmica equivalente, mientras que N_p permite escalar la corriente suministrada por el subsistema.

La estructura interna del subsistema fotovoltaico utilizado en PLECS se muestra en la Figura 2-3. Como puede observarse, el modelo está constituido por un bloque C-Script, una resistencia en serie R_s, una resistencia en derivación R_p y un filtro paso bajo de primer orden. Esta estructura reproduce el conocido modelo equivalente de un diodo, ampliamente utilizado para representar el comportamiento corriente-tensión de módulos fotovoltaicos.

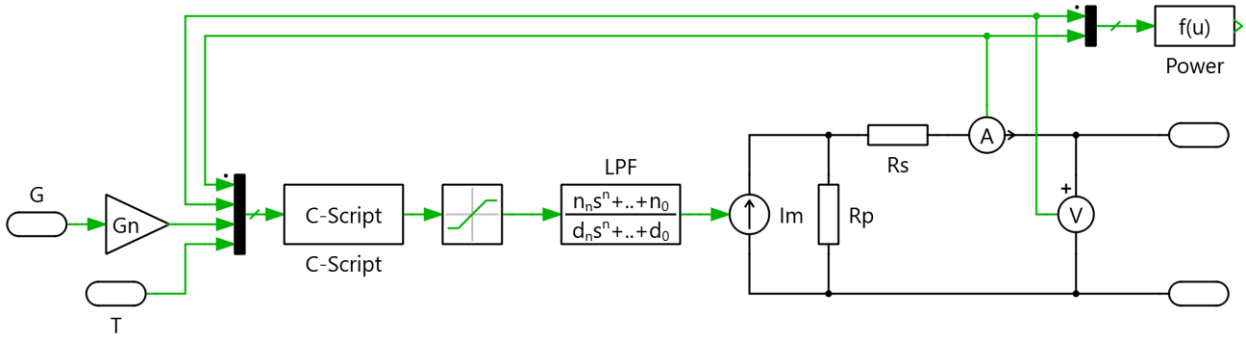

Figura 2-3. Esquema interno del modelo fotovoltaico empleado en PLECS

Desde el punto de vista físico, el circuito equivalente está formado por una fuente de corriente fotogenerada I_{ph}, que representa la corriente generada por la irradiancia incidente, un diodo equivalente, que modela el comportamiento de la unión PN de las células, una resistencia serie R_s, asociada a las pérdidas óhmicas internas, y una resistencia en derivación R_p, asociada a las corrientes de fuga internas del dispositivo.

A partir del circuito equivalente adoptado, la relación corriente-tensión del módulo puede expresarse como:

$$I = I_{ph} - I_0 \left[e^{\frac{V+R_s I}{aV_t}} - 1 \right] - \frac{V + R_s I}{R_p} \qquad (2-2)$$

donde I_0 es la corriente de saturación inversa del diodo, a es el factor de idealidad y V_t es la tensión térmica equivalente del conjunto de células conectadas en serie.

La ecuación $(2-2)$ se obtiene aplicando la ley de corrientes de Kirchhoff al circuito equivalente del módulo. En este planteamiento, la corriente suministrada por la fuente fotogenerada I_{ph} se reparte entre la corriente entregada a la salida, la corriente que circula por el diodo equivalente y la corriente de fuga asociada a la resistencia en derivación R_p.

En la implementación concreta utilizada en PLECS, la rama no lineal asociada al diodo se calcula dentro del bloque C-Script, mientras que la resistencia en derivación R_p se implementa como un elemento resistivo explícito conectado en paralelo. Por ello, la corriente interna calculada por la fuente de corriente controlada del modelo viene dada por:

$$I_m = I_{ph} - I_0 \left[e^{\frac{V+R_s I}{aV_t}} - 1 \right] \qquad (2-3)$$

siendo la rama resistiva R_p la encargada de representar externamente las corrientes de fuga del circuito equivalente.

A nivel interno, la irradiancia de entrada se introduce normalizada respecto a la irradiancia nominal G_n, es decir, en tanto por uno. Sea $g = G/G_n$ dicha irradiancia normalizada. De este modo, el modelo emplea internamente esta señal para escalar la corriente fotogenerada respecto a la condición nominal del módulo. La corriente interna calculada por el bloque, I_m, se obtiene mediante la expresión $(2-3)$. Este término representa la corriente generada por la fuente de corriente controlada interna del modelo fotovoltaico. Posteriormente, dicho valor se limita inferiormente a cero para evitar corrientes negativas y, a continuación, se somete a un filtro paso bajo de primer orden con el fin de suavizar la respuesta dinámica y mejorar la estabilidad numérica del modelo.

La corriente fotogenerada depende de la irradiancia incidente y de la temperatura de operación. En el modelo implementado, dicha corriente se expresa como:

$$I_{ph} = \left(I_{SC,n} + K_I \Delta T\right) g \qquad (2-4)$$

donde $I_{SC,n}$ es la corriente de cortocircuito nominal, K_I es el coeficiente térmico de corriente, g es la irradiancia normalizada y ΔT representa la desviación de temperatura respecto a la condición nominal.

De forma análoga, la corriente de saturación del diodo equivalente se calcula mediante:

$$I_0 = \frac{I_{SC,n} + K_I \Delta T}{e^{\frac{V_{OC,n}+K_V \Delta T}{aV_t}} - 1} \qquad (2-5)$$

donde $V_{OC,n}$ es la tensión de circuito abierto nominal y K_V es su coeficiente térmico.

La tensión térmica equivalente del conjunto de células en serie se define como:

$$V_t = \frac{N_s k (T + 273{,}15)}{q} \qquad (2-6)$$

siendo N_s el número de células conectadas en serie, k la constante de Boltzmann, q la carga elemental del electrón y T la temperatura de operación del módulo en grados Celsius. La tensión térmica representa la influencia de la temperatura sobre el comportamiento del diodo equivalente. Este término interviene en la rama exponencial del modelo y condiciona la forma de la curva I-V, especialmente en la región próxima a la tensión de circuito abierto. Además, su valor aumenta con la temperatura y con el número de células en serie consideradas en el modelo.

Conviene señalar que algunos parámetros del circuito equivalente, en particular la resistencia en serie R_s y la resistencia en derivación R_p, no suelen aparecer de forma explícita en la ficha técnica del fabricante. Por ello, estos parámetros deben determinarse previamente a partir de los datos nominales del módulo e introducirse posteriormente en la máscara del bloque. En consecuencia, R_s y R_p deben interpretarse como parámetros identificados del modelo equivalente, ajustados para reproducir la curva característica I-V del panel.

2.1.2 Parámetros del modelo

A continuación, se resumen los parámetros empleados en el modelo equivalente del módulo fotovoltaico:

- I_m: corriente interna generada por la fuente de corriente controlada del modelo fotovoltaico. Su valor no es constante, ya que depende de la irradiancia, la temperatura y del punto de operación eléctrico del módulo.
- I_{ph}: corriente fotogenerada asociada a la irradiancia incidente sobre el módulo. En el modelo empleado no se introduce como parámetro fijo, sino que se calcula en función de la irradiancia y de la temperatura mediante la ecuación $(2-4)$. En condiciones nominales:

$$I_{ph,n} \approx I_{SC,n} = 8{,}21\ A$$

- I_0: corriente de saturación inversa del diodo equivalente. En el modelo empleado no se introduce como parámetro independiente, sino que se calcula mediante la ecuación $(2-5)$.
- V: tensión en las bornas del módulo fotovoltaico. Se trata de una variable de simulación, por lo que no presenta un valor fijo único.
- I: corriente de salida en las bornas del módulo fotovoltaico. También es una variable de simulación.
- R_s: resistencia serie equivalente, que representa las pérdidas óhmicas internas debidas a contactos, conexiones y material semiconductor. El valor empleado es:

$$R_s = 0{,}068968\ \Omega$$

- R_p: resistencia en derivación del circuito equivalente, también denominada R_{sh} en la literatura, que representa las corrientes de fuga internas. El valor empleado es:

$$R_p = 30{,}13688\ \Omega$$

- a: factor de idealidad del diodo, que corrige la desviación respecto al comportamiento ideal de la unión PN. El valor empleado es:

$$a = 0{,}97734$$

- V_t: tensión térmica equivalente del conjunto de células en serie. En condiciones nominales:

$$V_{t,n} = \frac{54 \cdot k \cdot (25 + 273{,}15)}{q} \approx 1{,}387\ V$$

- N_s: número de células conectadas en serie considerado en el modelo equivalente del módulo. Este parámetro interviene en el cálculo de la tensión térmica y condiciona el comportamiento de la rama exponencial del diodo. Para el panel considerado:

$$N_s = 54$$

- N_p: número de ramas equivalentes conectadas en paralelo consideradas en el modelo. Este parámetro permite escalar la corriente suministrada por el subsistema y adaptar el modelo a la configuración eléctrica representada en la simulación.

$$N_p = 5$$

- k: constante de Boltzmann, cuyo valor es $1{,}3806503 \cdot 10^{-23}$ J/K.
- T: temperatura de operación del módulo fotovoltaico, expresada en ºC.
- T_n: temperatura nominal de referencia, 25 ºC.
- q: carga elemental del electrón, cuyo valor es $1{,}60217646 \cdot 10^{-19}$ C.
- G: irradiancia incidente sobre el módulo.
- G_n: irradiancia nominal de referencia:

$$G_n = 1000\ W/m^2$$

- $V_{OC,n}$: tensión de circuito abierto nominal del módulo:

$$V_{OC,n} = 32{,}9\ V$$

- $I_{SC,n}$: corriente de cortocircuito nominal del módulo:

$$I_{SC,n} = 8{,}21\ A$$

- K_V: coeficiente térmico de la tensión de circuito abierto:

$$K_V = -0{,}123\ V/K$$

- K_I: coeficiente térmico de la corriente de cortocircuito:

$$K_I = 0{,}00318\ A/K$$

A partir de esta estructura, el modelo puede parametrizarse para distintos módulos fotovoltaicos siempre que se disponga de los datos nominales y de los parámetros identificados del circuito equivalente. En el presente trabajo, dicha parametrización se ha realizado para reproducir el comportamiento del módulo Kyocera Solar KC200GT y su integración posterior en la planta fotovoltaica objeto de estudio.

2.2. Convertidor elevador

El convertidor elevador, o Boost, es un convertidor DC/DC no aislado ampliamente empleado como interfaz entre un generador fotovoltaico y un enlace de continua. Su función principal es elevar la tensión de entrada v_{pv} hasta una tensión superior en el bus DC v_{dc}, requerida por la etapa inversora conectada a red. Además de adaptar niveles de tensión, en sistemas fotovoltaicos el Boost actúa como elemento de control del punto de operación de la planta, permitiendo imponer el valor de v_{pv} que fija el algoritmo MPPT para extraer la máxima potencia.

La topología básica del Boost convencional está formada por un inductor L, un interruptor controlado (MOSFET o IGBT), un diodo hacia la salida y un condensador en el enlace DC que filtra el rizado y estabiliza v_{dc}.

En la Figura 2-4 se muestra el esquema del convertidor elevador convencional empleado como referencia para el análisis.

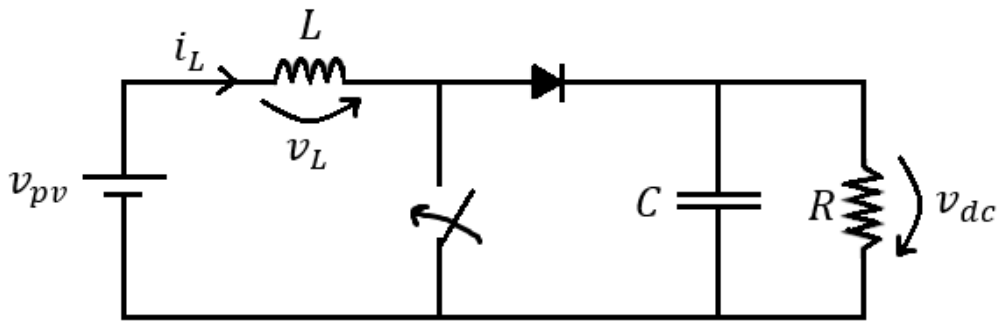

Figura 2-4. Esquema del convertidor elevador convencional

En modo de conducción continua (CCM), y bajo hipótesis ideales, su funcionamiento puede describirse mediante dos intervalos dentro del periodo de conmutación $T_s = 1/f_s$:

- Intervalo ON (interruptor cerrado, $0 \leq t < DT_s$)

  El diodo queda bloqueado y el inductor se conecta a la entrada, almacenando energía. La tensión en el inductor es aproximadamente:

$$v_L = v_{pv} \rightarrow \frac{di_L}{dt} = \frac{v_{pv}}{L}$$

- Intervalo OFF (interruptor abierto, $DT_s \leq t < T_s$)

  El diodo conduce y el inductor transfiere energía al enlace DC. En este intervalo se tiene:

$$v_L = v_{pv} - v_{dc} \rightarrow \frac{di_L}{dt} = \frac{v_{pv} - v_{dc}}{L}$$

En régimen permanente, la corriente del inductor al final de cada periodo de conmutación coincide con la del inicio. En consecuencia, el valor medio de la tensión en el inductor a lo largo de un periodo es nulo, a $\overline{v_L} = 0$. Así, se deduce:

$$D v_{pv} + (1 - D)\left(v_{pv} - v_{dc}\right) = 0 \rightarrow v_{dc} = \frac{v_{pv}}{1 - D}$$

siendo D el ciclo de trabajo ($0 < D < 1$). Esta relación muestra que, al aumentar D, la tensión de salida puede ser significativamente mayor que la de entrada.

Asimismo, una aproximación útil del rizado de corriente del inductor en CCM es:

$$\Delta i_L \approx \frac{v_{pv} D}{L f_s}$$

lo que evidencia la influencia de L y de la frecuencia de conmutación f_s sobre el rizado y, por tanto, sobre la calidad de la corriente absorbida del generador FV.

Aunque el análisis anterior se ha presentado sobre el convertidor Boost convencional por su claridad expositiva, la etapa DC/DC implementada en el modelo corresponde a un Boost síncrono. Esta variante sustituye el diodo por un interruptor controlado, manteniendo, bajo hipótesis ideales en CCM, la misma relación tensión/ciclo de trabajo deducida anteriormente.

En la Figura 2-5 se muestra el esquema del convertidor elevador síncrono empleado. La rama de conmutación se implementa mediante dos semiconductores (Top y Bottom) en configuración tipo half-bridge, cada uno con su diodo antiparalelo. En comparación con el Boost convencional, el elemento equivalente al diodo se reemplaza por un interruptor activo, lo que puede reducir las pérdidas de conducción.

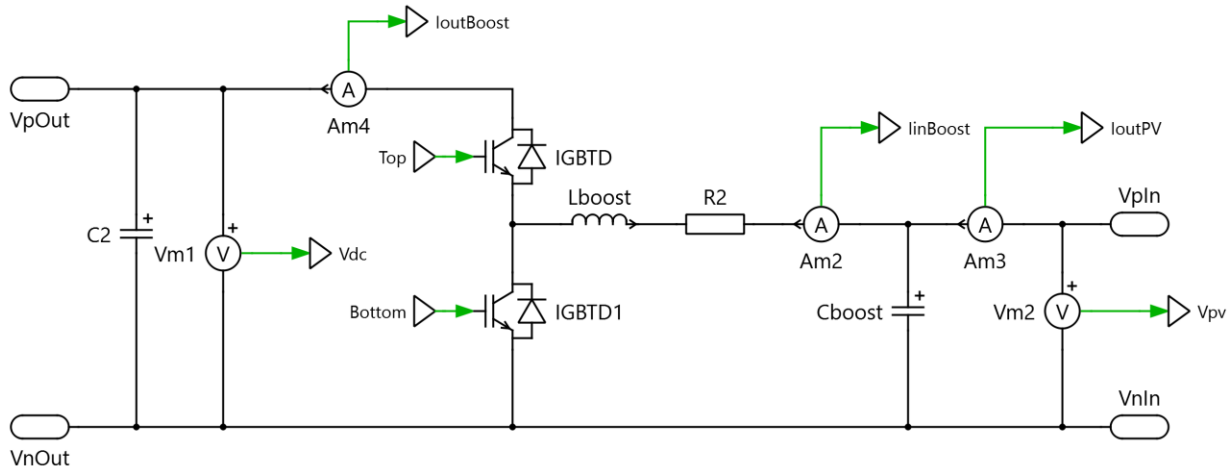

Figura 2-5. Esquema del convertidor elevador síncrono implementado en el modelo

Desde el punto de vista funcional, el Boost síncrono reproduce los dos intervalos del Boost convencional: durante el intervalo equivalente a ON, el interruptor inferior (Bottom) conduce y el inductor almacena energía; durante el intervalo equivalente a OFF, el interruptor superior (Top) conduce y el inductor transfiere energía al enlace DC. Por tanto, bajo hipótesis ideales y en CCM, las expresiones obtenidas anteriormente para v_L en cada intervalo y la relación $v_{dc} = \frac{v_{pv}}{1-D}$ se mantienen, ya que dependen de la periodicidad de la corriente del inductor y del valor medio de su tensión en un periodo.

La implementación síncrona requiere modulación complementaria entre Top y Bottom e inclusión de tiempo muerto (dead-time) para evitar conducción simultánea de ambos dispositivos (shoot-through).

Finalmente, en la estrategia de puesta en marcha del sistema se adopta una habilitación secuencial de las etapas. En el modelo, la tensión del enlace DC, v_{dc}, es regulada por el control del inversor (Enabled dq-Control), mientras que la etapa Boost se habilita posteriormente. Dado que el convertidor síncrono incorpora diodos antiparalelos, durante el arranque pueden existir trayectorias pasivas de conducción determinadas por las tensiones instantáneas del enlace DC y del lado FV. Este aspecto se ha considerado en la definición de las condiciones iniciales y en la interpretación del comportamiento transitorio del sistema.

2.3. Transformadas de Clarke y Park para el control de la etapa inversora

Dado que en la implementación del sistema se emplean bloques de transformación abc-dq y dq-abc en la etapa inversora, resulta conveniente presentar previamente las transformaciones de Clarke y Park, con el fin de facilitar la comprensión del modelo y del esquema de control adoptado. En sistemas trifásicos conectados a red, las tensiones y corrientes se expresan de forma natural en coordenadas de fase abc. Sin embargo, para el análisis del sistema y el diseño del control, resulta más conveniente trabajar en sistemas de referencia transformados.

La transformación de Clarke permite representar un sistema trifásico de tres hilos en un plano bidimensional $\alpha\beta$. Esta transformación resulta adecuada porque, en este tipo de sistemas, las tres componentes no son linealmente independientes, por lo que puede trabajarse en dos dimensiones sin pérdida de información relevante. En este trabajo se adopta su formulación invariante en potencia, ya que permite preservar la equivalencia de potencia entre las variables expresadas en coordenadas abc y aquellas representadas en el sistema ortogonal $\alpha\beta$. Esta propiedad resulta especialmente útil en el análisis y en el control de convertidores trifásicos conectados a red.

Bajo la hipótesis de un sistema equilibrado y sin componente homopolar, la transformación de Clarke puede escribirse como:

$$\begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix}$$

donde x_a, x_b y x_c representan una magnitud trifásica genérica, ya sea tensión o corriente.

Sobre esta representación estacionaria, la transformación de Park permite pasar al sistema de referencia síncrono dq. Su interés principal radica en que facilita el diseño de la estrategia de control, ya que transforma las variables sinusoidales en magnitudes continuas cuando el sistema de referencia gira sincronizado con la red. Asimismo, al alinear dicho sistema con el vector de tensión, este pasa a presentar una única componente dominante, lo que simplifica notablemente el planteamiento del control.

La transformación de Park se expresa como:

$$\begin{bmatrix} x_d \\ x_q \end{bmatrix} = \begin{bmatrix} cos\ \theta & sen\ \theta \\ -sen\ \theta & cos\ \theta \end{bmatrix} \begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix}$$

donde θ es el ángulo instantáneo del sistema de referencia síncrono.

Para aplicar esta transformación es necesario disponer de un bloque de sincronización que proporcione el ángulo θ o la frecuencia de la red. En convertidores conectados a red, esta función se realiza habitualmente mediante un sistema PLL o FLL, encargado de seguir la fase y la frecuencia de la tensión de red y garantizar así una referencia adecuada para la transformación.

Una vez calculadas las variables en ejes dq, el diseño del control resulta más sencillo, ya que estas componentes pueden tratarse como magnitudes continuas en régimen estacionario. Posteriormente, para obtener las referencias trifásicas necesarias para la modulación del inversor, se aplican las transformaciones inversas de Park y Clarke, recuperando así las variables en coordenadas abc.

En el sistema desarrollado en este trabajo, estas transformaciones se emplean en la etapa inversora para convertir las magnitudes trifásicas medidas a coordenadas dq, facilitar el diseño del control y generar posteriormente las referencias trifásicas necesarias para la modulación. De este modo, la introducción de estas transformaciones proporciona la base necesaria para interpretar los bloques de control empleados posteriormente en la implementación del modelo.

2.4. Caracterización de la planta fotovoltaica mediante curvas I-V y P-V

La caracterización eléctrica de la planta fotovoltaica se realiza a partir del análisis de las curvas I-V y P-V, ya que estas permiten describir la relación entre las principales magnitudes eléctricas del sistema e identificar su

punto de máxima potencia. En las condiciones consideradas, correspondientes a una temperatura de 25 ºC y una irradiancia de 1000 W/m^2, los parámetros más representativos de la planta son una tensión de circuito abierto (V_{oc}) de 493,5 V, una corriente de cortocircuito (I_{sc}) de 41,05 A, una tensión en el punto de máxima potencia (V_{mpp}) de 394,5 V, una corriente en dicho punto (I_{mpp}) de 38,05 A y una potencia máxima (P_{mpp}) de 15,01 kW. El análisis de estas magnitudes resulta fundamental para evaluar el comportamiento de la instalación y determinar su zona óptima de funcionamiento.

Como se observa en la Figura 2-6, correspondiente a la curva I-V de la planta fotovoltaica, la corriente suministrada se mantiene prácticamente constante para valores bajos y medios de tensión. Sin embargo, al aproximarse a la región próxima a la tensión de circuito abierto, la corriente experimenta una caída pronunciada hasta anularse en V_{oc}. Asimismo, la intersección de la curva con el eje de corriente permite identificar la corriente de cortocircuito. Este comportamiento es característico de los generadores fotovoltaicos y permite analizar la capacidad de entrega de corriente de la planta en función de la tensión de operación.

Por su parte, tal y como se muestra en la Figura 2-7, la curva P-V presenta un incremento progresivo de la potencia con la tensión hasta alcanzar un máximo, correspondiente al punto de máxima potencia. En este punto, situado en torno a V_{mpp} = 394,5 V, la planta entrega su mayor potencia útil, aproximadamente de P_{mpp} = 15,01 kW. A partir de dicho valor, la potencia disminuye progresivamente mientras la tensión sigue aumentando, hasta anularse al alcanzarse la tensión de circuito abierto. Esta curva permite identificar de forma clara la región de operación de máximo aprovechamiento energético, lo que resulta especialmente relevante para el estudio e implementación de estrategias de seguimiento del punto de máxima potencia.

En conjunto, ambas curvas permiten identificar el punto de funcionamiento óptimo de la planta fotovoltaica y constituyen la base para el posterior desarrollo de la estrategia MPPT implementada en el sistema.

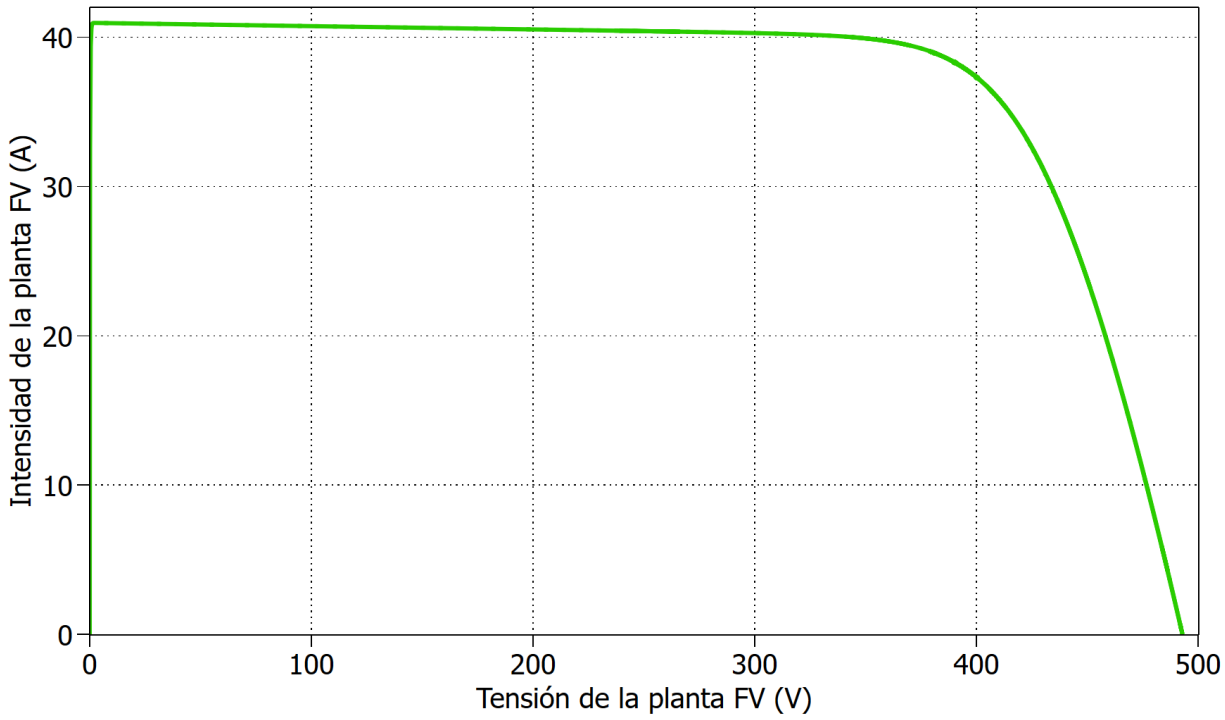

Figura 2-6. Curva característica corriente-tensión (I-V) de la planta FV para 25ºC y 1000W/m^2

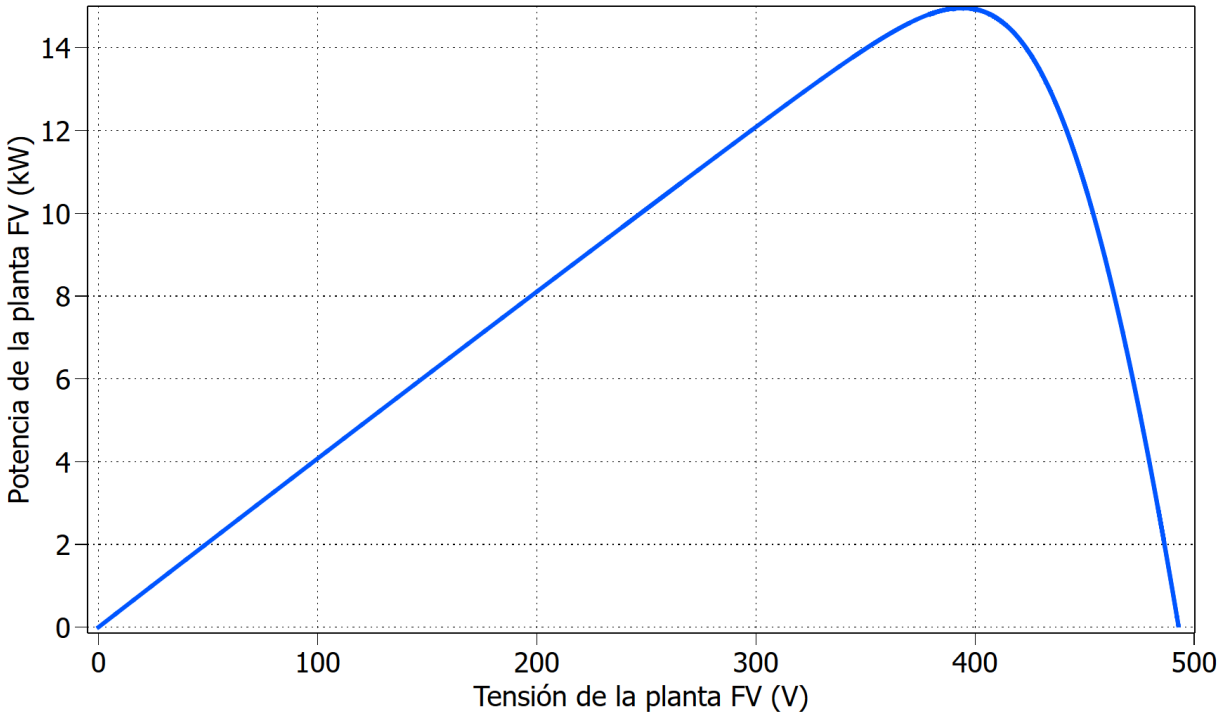

Figura 2-7. Curva característica potencia-tensión (P-V) de la planta FV para 25ºC y 1000W/m^2

3 ALGORITMOS MPPT

El rendimiento de un sistema fotovoltaico está intrínsecamente ligado a las condiciones ambientales, presentando un comportamiento eléctrico fuertemente no lineal. La potencia de salida de un módulo fotovoltaico varía continuamente, principalmente, en función de dos perturbaciones externas: la irradiancia solar incidente y la temperatura de operación de la célula. Debido a esta variabilidad, existe un único punto de operación en la curva característica Corriente-Voltaje (I-V) y Potencia-Voltaje (P-V) en el cual el panel entrega la máxima potencia posible. Este punto se denomina Punto de Máxima Potencia (MPP, Maximum Power Point).

Para maximizar la eficiencia del sistema y garantizar la extracción óptima de energía, es imprescindible la implementación de algoritmos de Seguimiento del Punto de Máxima Potencia (MPPT, Maximum Power Point Tracking). Un algoritmo MPPT puede entenderse como un controlador electrónico que, a partir de las medidas de tensión y corriente del generador (V_{pv} e I_{pv}), estima la potencia $P_{pv} = V_{pv}I_{pv}$ y ajusta dinámicamente la consigna de operación del arreglo fotovoltaico. En función de la arquitectura, esta actuación puede integrarse en la etapa DC/DC o en el inversor.

En la configuración considerada en este trabajo, basada en una arquitectura de doble etapa (convertidor DC/DC elevador tipo boost síncrono e inversor conectado a red), el MPPT actúa sobre el convertidor DC/DC, ya que es el encargado de imponer el punto de operación del generador FV. En particular, el algoritmo MPPT genera una referencia de tensión V_{pv}^{ref}, que es seguida por los lazos internos del convertidor mediante el ajuste del PWM, de modo que $V_{pv} \rightarrow V_{pv}^{ref}$. Con ello se desplaza el punto de funcionamiento del generador hacia el MPP y se maximiza la potencia extraída del campo fotovoltaico para las condiciones ambientales existentes, reduciendo la energía no aprovechada asociada a operar fuera del punto óptimo.

En este capítulo se describe uno de los métodos MPPT más utilizados: Perturbación y Observación. Este es seleccionado por su simplicidad y su integración directa con la estructura de control del convertidor elevador. Se detalla su principio de funcionamiento y la lógica de actualización de la referencia V_{pv}^{ref} empleada en el modelo.

3.1. Método de Perturbación y Observación (P&O)

El método Perturbación y Observación es uno de los algoritmos MPPT más utilizados debido a su simplicidad y bajo coste computacional. Su principio consiste en perturbar periódicamente la referencia de tensión del generador fotovoltaico, V_{pv}^{ref}, y observar el efecto de dicha perturbación sobre la potencia extraída. El algoritmo se basa exclusivamente en magnitudes eléctricas medidas en bornes del generador, sin requerir la estimación de irradiancia ni temperatura.

En el instante discreto k se miden la tensión y la corriente del campo FV, $V_{pv,k}$ e $I_{pv,k}$, y se calcula la potencia:

$$P_k = V_{pv,k} I_{pv,k}$$

A partir de muestras consecutivas se definen los incrementos:

$$\Delta P_k = P_k - P_{k-1}, \qquad \Delta V_{pv,k} = V_{pv,k} - V_{pv,k-1}.$$

La lógica del algoritmo establece:

- Si $\Delta P_k > 0$: la perturbación aplicada ha acercado el punto de operación al MPP, por lo que se mantiene el sentido de la perturbación en V_{pv}^{ref}.
- Si $\Delta P_k < 0$: la perturbación ha alejado el punto de operación del MPP, por lo que se invierte el sentido de la perturbación.

De acuerdo con la formulación utilizada en el diagrama de flujo de P&O, el cual se puede ver en la Figura 3-1, la referencia se actualiza mediante un paso Δ_k, constante en magnitud:

$$V_{pv,k}^{ref} = V_{pv,k-1}^{ref} \pm \Delta_k,$$

donde el signo se determina a partir de ΔP_k y $\Delta V_{pv,k}$. La Tabla 3-1 resume esta decisión en términos del incremento aplicado Δ_k.

- Si $\Delta V_{pv,k} > 0$ y $\Delta P_k > 0 \rightarrow \Delta_k > 0$, incrementar V_{pv}^{ref}.
- Si $\Delta V_{pv,k} > 0$ y $\Delta P_k < 0 \rightarrow \Delta_k < 0$, decrementar V_{pv}^{ref}.
- Si $\Delta V_{pv,k} < 0$ y $\Delta P_k > 0 \rightarrow \Delta_k < 0$, decrementar V_{pv}^{ref}.
- Si $\Delta V_{pv,k} < 0$ y $\Delta P_k < 0 \rightarrow \Delta_k > 0$, incrementar V_{pv}^{ref}.

$\Delta V_{pv,k}$	ΔP_k	Δ_k
> 0	> 0	> 0
> 0	< 0	< 0
< 0	> 0	< 0
< 0	< 0	> 0

Tabla 3-1. Regla de decisión del algoritmo P&O

En la implementación, además de actualizar V_{pv}^{ref} se almacenan las variables del instante anterior ($P_{k-1} \leftarrow P_k, V_{pv,k-1} \leftarrow V_{pv,k}$) para el cálculo de incrementos en el siguiente periodo de muestreo, tal y como refleja el bloque inicial del diagrama de flujo (Figura 3-1).

Debido a la perturbación discreta, P&O produce una oscilación inherente alrededor del MPP, cuyo nivel depende del tamaño de paso Δ_k. Un paso grande acelera la convergencia, pero incrementa el rizado de potencia y la operación alrededor del óptimo; un paso pequeño reduce el rizado, pero empeora la respuesta dinámica frente a cambios ambientales.

Por último, ante la presencia de variaciones rápidas de irradiancia, el incremento de potencia (ΔP_k) puede estar influido por el propio cambio ambiental y no únicamente por la perturbación aplicada. En esos casos, el algoritmo puede tomar decisiones temporales no óptimas, motivo por el que en aplicaciones prácticas se recurre a filtrado, umbrales o estrategias de paso adaptativo cuando se requiere mayor robustez [16].

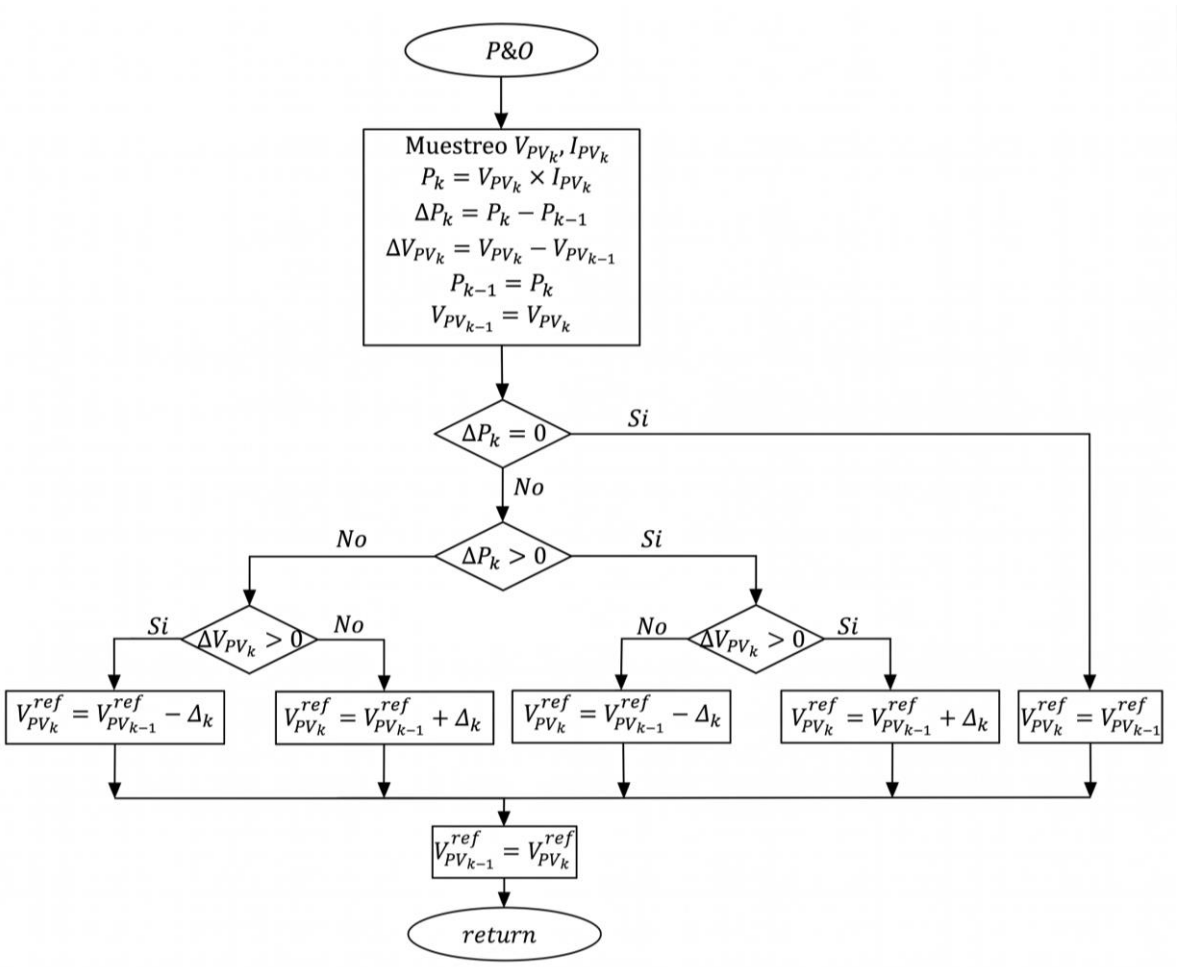

Figura 3-1. Diagrama de flujo del método Perturbación y Observación [6]

4 DISEÑO DEL CONTROL DEL CONVERTIDOR

En este capítulo se desarrolla el diseño del control del convertidor elevador síncrono encargado de imponer el punto de operación del generador fotovoltaico. Dado que el algoritmo MPPT proporciona una referencia de tensión V_{pv}^{ref} pero no actúa directamente sobre la planta, se adopta una estructura de control en cascada con dos lazos: un lazo externo de regulación de tensión FV, que transforma el error entre V_{pv} y V_{pv}^{ref} en una consigna de potencia P^{ref}, y un lazo interno de corriente, más rápido, que garantiza el seguimiento de la referencia de corriente asociada I_{pv}^{ref} y genera la señal de modulación aplicada al PWM.

Esta arquitectura permite separar escalas temporales y asegurar estabilidad: el lazo interno compensa la dinámica rápida del convertidor, mientras que el lazo externo ajusta de forma más lenta el punto de funcionamiento del generador. En los apartados siguientes se presentan las ecuaciones empleadas, la implementación discreta de los controladores PI y los parámetros seleccionados para cada lazo.

4.1. Lazo externo de control

El MPPT define la consigna de operación del generador fotovoltaico, proporcionando la referencia V_{pv}^{ref} correspondiente al entorno del punto de máxima potencia. Sin embargo, el MPPT no actúa directamente sobre la planta; por ello, se implementa un lazo externo de control cuya misión es asegurar el seguimiento dinámico de la referencia de tensión. En concreto, este lazo regula el punto de operación del generador ajustando la potencia demandada al convertidor, de manera que la tensión medida V_{pv} evolucione hacia V_{pv}^{ref}.

En el instante discreto k, el lazo utiliza la medida $V_{pv}[k]$ y la referencia $V_{pv}^{ref}[k]$. En la implementación, el error de tensión se define en forma energética:

$$e_v[k] = \frac{1}{2}\left(\left(V_{pv}^{ref}[k]\right)^2 - \left(V_{pv}[k]\right)^2\right) \qquad (4-1)$$

y se procesa mediante un controlador PI discreto, cuya salida se interpreta como potencia de referencia. Con el fin de garantizar realimentación negativa, se adopta una convención de signo tal que un déficit de tensión ($V_{pv} < V_{pv}^{ref}$) se traduzca en una reducción de la potencia extraída del generador. Por ello, se introduce un cambio de signo, de modo que:

$$P^{ref}[k] = -u_v[k] \qquad (4-2)$$

Esta potencia de referencia se utiliza posteriormente para generar la consigna del lazo interno de corriente, encargado de la actuación rápida sobre el modulador PWM.

Aunque un controlador PI discreto puede expresarse en forma incremental, en este trabajo se adopta una realización en estado integral explícito. En esta formulación, la acción integral del controlador se almacena como una variable de estado discreta y se actualiza en cada periodo de muestreo mediante una discretización Forward Euler. Así, para el lazo externo se tiene:

$$I_v[k] = I_v[k-1] + T_s e_v[k] \qquad (4-3)$$

$$u_v[k] = K_{p,v} e_v[k] + K_{i,v} I_v[k] \qquad (4-4)$$

donde $I_v[k]$ representa el estado integral del controlador, T_s es el periodo de muestreo y $K_{p,v}$ y $K_{i,v}$ son las ganancias proporcional e integral, respectivamente. Esta formulación es matemáticamente equivalente a la forma incremental habitual, pero permite describir de manera más directa la actualización de las variables internas del controlador.

En el modelo se emplean $K_{p,v} = 1$ y $K_{i,v} = 10$. Estos valores se seleccionan de manera que el lazo externo presente una dinámica más lenta que el lazo interno de corriente, manteniendo la separación de escalas propia

del control en cascada y evitando interacciones entre la regulación de corriente y el seguimiento de la referencia de tensión.

4.2. Lazo interno de control

El lazo interno tiene como objetivo proporcionar una dinámica rápida al convertidor, regulando la corriente del lado fotovoltaico para seguir una referencia generada por el lazo externo. En régimen de conducción continua, la corriente medida I_{pv} es representativa de la corriente del inductor del convertidor, por lo que su regulación permite controlar el intercambio instantáneo de energía entre generador FV y el bus DC. En una estructura en cascada, este lazo se diseña con un ancho de banda significativamente mayor que el del lazo externo, garantizando estabilidad y desacoplo dinámico.

La salida del lazo externo se define como una potencia de referencia $P^{ref}[k]$. A partir de la relación entre potencia, tensión y corriente, se obtiene la referencia de corriente:

$$I_{pv}^{ref}[k] = \frac{P^{ref}[k]}{V_{pv}[k]} \tag{4-5}$$

El error de corriente se define como:

$$e_i[k] = I_{pv}[k] - I_{pv}^{ref}[k] \tag{4-6}$$

De forma análoga al lazo externo, se emplea un controlador PI discreto en estado integral explícito. En consecuencia, la acción integral se actualiza en cada instante de muestreo y la salida del controlador se calcula a partir del error actual y del valor acumulado de dicha integral:

$$I_i[k] = I_i[k-1] + T_s e_i[k] \tag{4-7}$$

$$u_i[k] = K_{p,i} e_i[k] + K_{i,i} I_i[k] \tag{4-8}$$

donde $I_i[k]$ es el estado integral del controlador de corriente, y $K_{p,i}$ y $K_{i,i}$ son las ganancias proporcional e integral del lazo interno. En el modelo se emplean $K_{p,i} = 10$ y $K_{i,i} = 500$. Estos valores se han seleccionado para que el lazo interno de corriente presente una dinámica claramente más rápida que el lazo externo de tensión, asegurando la separación de escalas propia del control en cascada.

En la implementación, la salida del PI de corriente se combina con un término de feedforward basado en la tensión medida, construyendo la referencia de modulación:

$$RefMod[k] = V_{pv}[k] + u_i[k] \tag{4-9}$$

El término $V_{pv}[k]$ aporta una compensación directa, mientras que $u_i[k]$ introduce la corrección necesaria para forzar el seguimiento de la referencia de corriente generada a partir de $P^{ref}[k]$. Finalmente, esta señal se emplea para generar la acción de control aplicada al convertidor mediante el modulador PWM.

5 IMPLEMENTACIÓN DEL SISTEMA EN PLECS

Una vez desarrollados el modelo del sistema fotovoltaico, la estrategia de seguimiento del punto de máxima potencia y el diseño de los lazos de control del convertidor, se procede a su implantación en el entorno de simulación PLECS. Esta etapa permite integrar en una misma plataforma tanto la planta de potencia como los algoritmos de control, constituyendo una base de simulación coherente para el análisis dinámico del sistema.

A diferencia de los capítulos anteriores, centrados en la descripción individual de los distintos subsistemas y en el desarrollo teórico de las leyes de control, en este capítulo se aborda su integración dentro de un esquema global de simulación. Para ello, en primer lugar, se presenta el modelo base implementado en PLECS, en el que se dispone la arquitectura completa del sistema fotovoltaico conectado a red. Posteriormente, se describe la incorporación de la estrategia de control mediante bloques C-Script, que permiten implementar de forma explícita los reguladores y facilitar su posterior adaptación a entornos de validación más próximos al sistema real.

5.1. Modelo base del sistema en PLECS

Una vez definidos en los capítulos anteriores el modelo del generador fotovoltaico, la topología de conversión empleada y la estrategia general de control, se aborda la implantación del sistema en el entorno de simulación PLECS. Como primera etapa, se desarrolla un modelo base que integra en un mismo esquema la planta fotovoltaica, el convertidor elevador, el enlace de continua, el inversor trifásico y los bloques auxiliares necesarios para la medida de variables, el tratamiento de señales y la generación de disparos. Esta implementación permite analizar de forma conjunta el comportamiento dinámico del sistema y comprobar la coherencia de su funcionamiento antes de abordar desarrollos posteriores.

5.1.1 Estructura general del modelo implementado

La Figura 5-1 muestra el esquema general del modelo base implementado en PLECS. En él se distinguen claramente dos etapas principales. Por un lado, la etapa fotovoltaica con convertidor elevador, encargada de extraer la potencia generada por la planta y adaptar su nivel de tensión al requerido por el enlace de continua. Por otro, la etapa de conversión DC/AC, formada por un inversor trifásico de dos niveles y su correspondiente interfaz de conexión a red. Sobre esta estructura energética se integran, además, los bloques de transformación de coordenadas, control y modulación necesarios para el funcionamiento del sistema.

La planta fotovoltaica se sitúa en la parte derecha del esquema general y alimenta directamente al subsistema Boost Converter. Tal como se observa en la Figura 5-2, este subsistema incorpora la etapa de potencia del convertidor elevador síncrono, compuesta por dos interruptores controlados, la inductancia de elevación y los elementos pasivos asociados al filtrado y al almacenamiento de energía. Asimismo, se incluyen los bloques de medida de las principales variables eléctricas, entre ellas la tensión de entrada del convertidor, la corriente suministrada por la planta, la corriente de la bobina y la tensión de salida.

Además de la etapa de potencia, el subsistema integra los bloques funcionales necesarios para la supervisión, el control y la actuación del convertidor. En este sentido, se dispone de un bloque de control, denominado Enabled boost-Control, que procesa las magnitudes medidas y genera la referencia de modulación del convertidor. Dicha referencia se aplica posteriormente al bloque PWM, encargado de transformarla en los pulsos de disparo de los interruptores superior e inferior de la etapa elevadora. Asimismo, la incorporación de una señal de habilitación permite fijar explícitamente el instante de activación tanto del control como de la modulación, lo que facilita el análisis del arranque y evita transitorios no deseados durante la inicialización del sistema. Por tanto, en el modelo base, el bloque PWM forma parte de la propia arquitectura funcional del subsistema Boost Converter, actuando como interfaz entre la señal de modulación generada por el control y los dispositivos de conmutación del convertidor.

La parte izquierda del modelo corresponde a la etapa inversora conectada a red. En ella se dispone de una fuente trifásica que representa la red, el filtro de conexión y el convertidor trifásico de dos niveles. El enlace entre esta etapa y el convertidor elevador se realiza a través del bus de continua, cuya tensión constituye una magnitud fundamental para el funcionamiento conjunto del sistema. Para el control del inversor se emplean bloques de transformación de coordenadas abc-dq y dq-abc, asociados a las transformaciones de Clarke y Park, que permiten expresar las variables trifásicas en un sistema de referencia más adecuado para el diseño del control. Asimismo, se incorpora el bloque Enabled dq-Control, encargado de generar las referencias necesarias para la modulación.

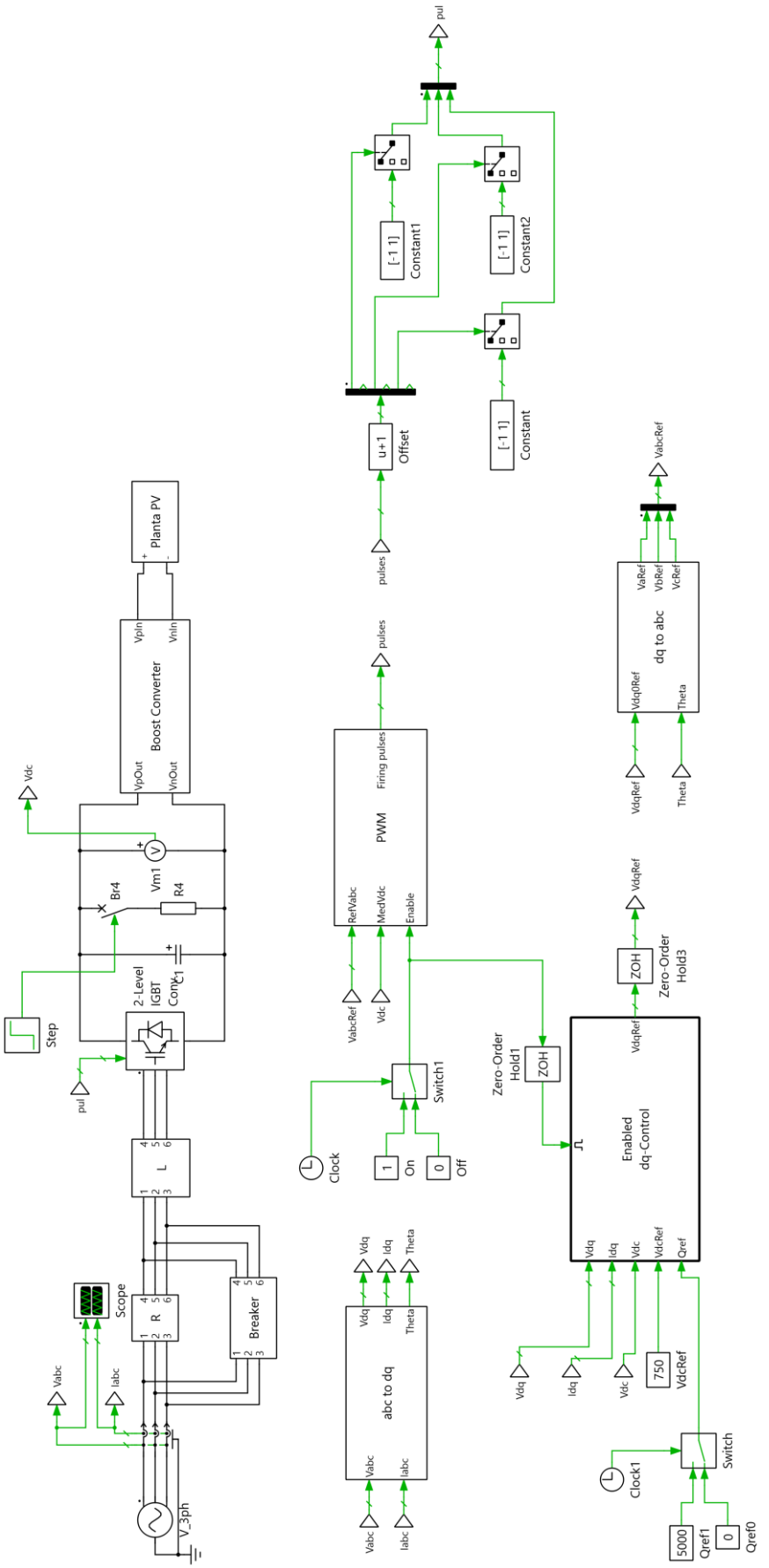

Figura 5-1. Esquema general del modelo del sistema implementado en PLECS

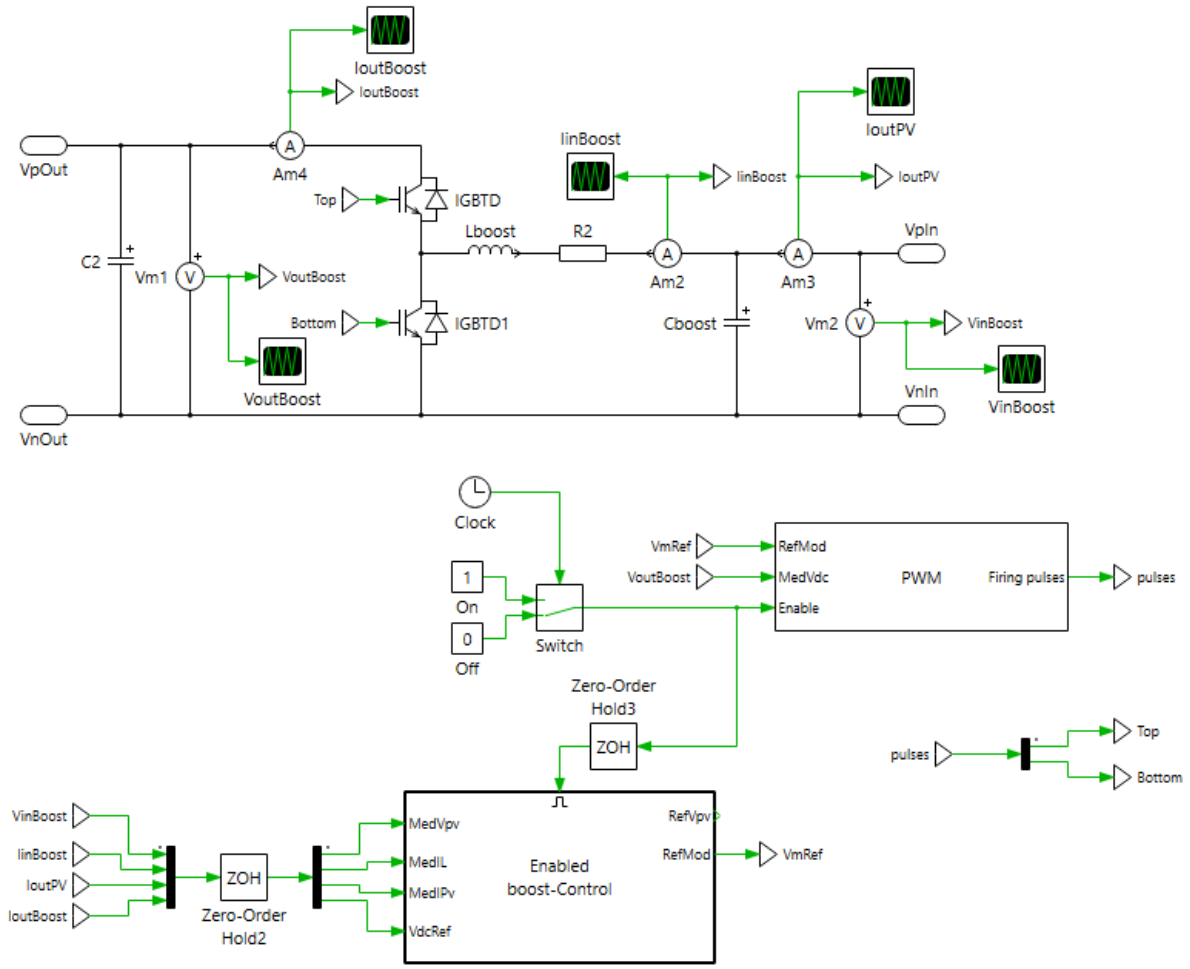

Figura 5-2. Subsistema del convertidor elevador implementado en PLECS y bloques funcionales asociados a su control y modulación

En conjunto, el modelo base implementado en PLECS no solo reproduce la topología de potencia del sistema fotovoltaico conectado a red, sino también la estructura funcional necesaria para su supervisión y control. La organización modular adoptada facilita la identificación de cada subsistema y permite disponer de una base de simulación adecuada para el análisis de la respuesta dinámica del sistema.

Una vez descrita la estructura general del modelo y los principales bloques que lo componen, resulta necesario verificar su comportamiento mediante simulación. Para ello, en el apartado siguiente se analizan las principales variables eléctricas de la etapa fotovoltaica y del convertidor elevador, con objeto de comprobar la coherencia de la respuesta obtenida tanto durante el arranque como en régimen permanente.

5.1.2 Validación inicial mediante simulación

Con objeto de validar el comportamiento del modelo base implementado en PLECS, se realiza una simulación temporal de 2 segundos en la que se analizan las principales magnitudes eléctricas de la etapa fotovoltaica y del convertidor elevador. En las figuras siguientes se recoge la evolución temporal de dichas variables, mientras que la Figura 5-8 muestra la comparación entre la tensión del generador fotovoltaico V_{pv} y la referencia $RefVpv$ calculada por el bloque MPPT. Conviene señalar que, en este modelo, la tensión de entrada del convertidor elevador $V_{inBoost}$ coincide con la tensión en bornas del generador fotovoltaico V_{pv}. Asimismo, el bloque MPPT se encuentra integrado dentro de la estructura interna del bloque Enabled boost-Control, cuya organización funcional se detalla posteriormente en la Figura 5-9.

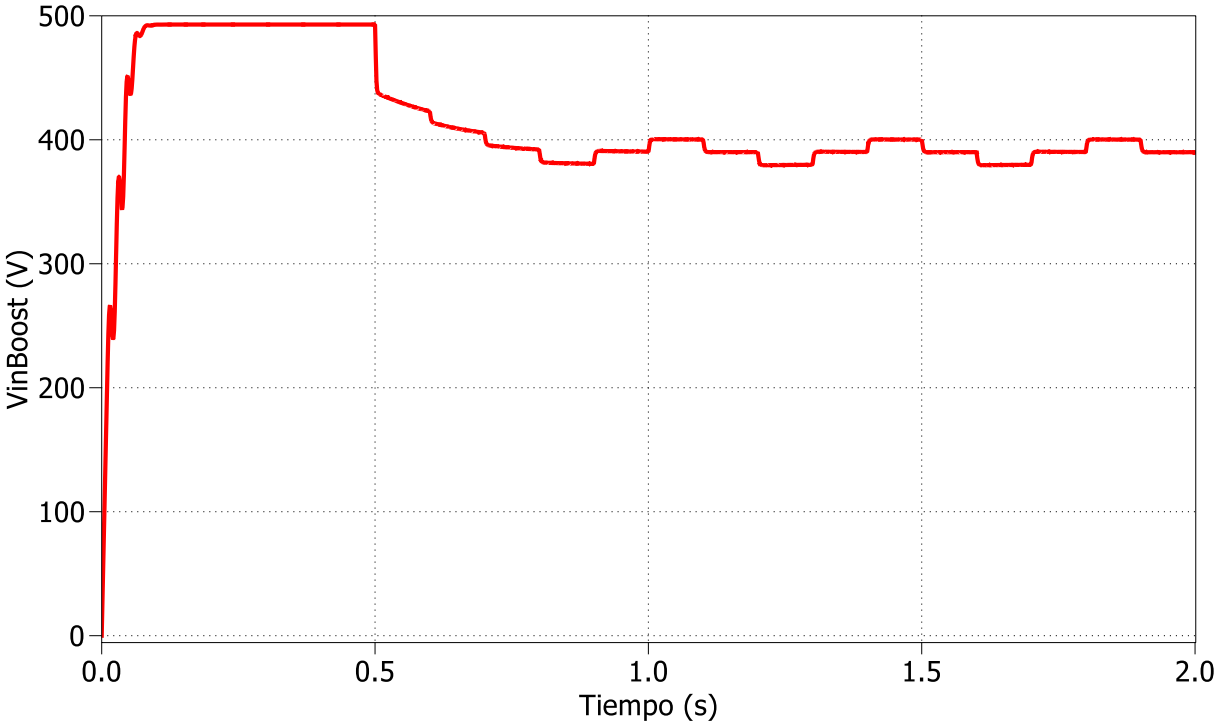

Figura 5-3. Evolución temporal de la tensión de entrada del convertidor

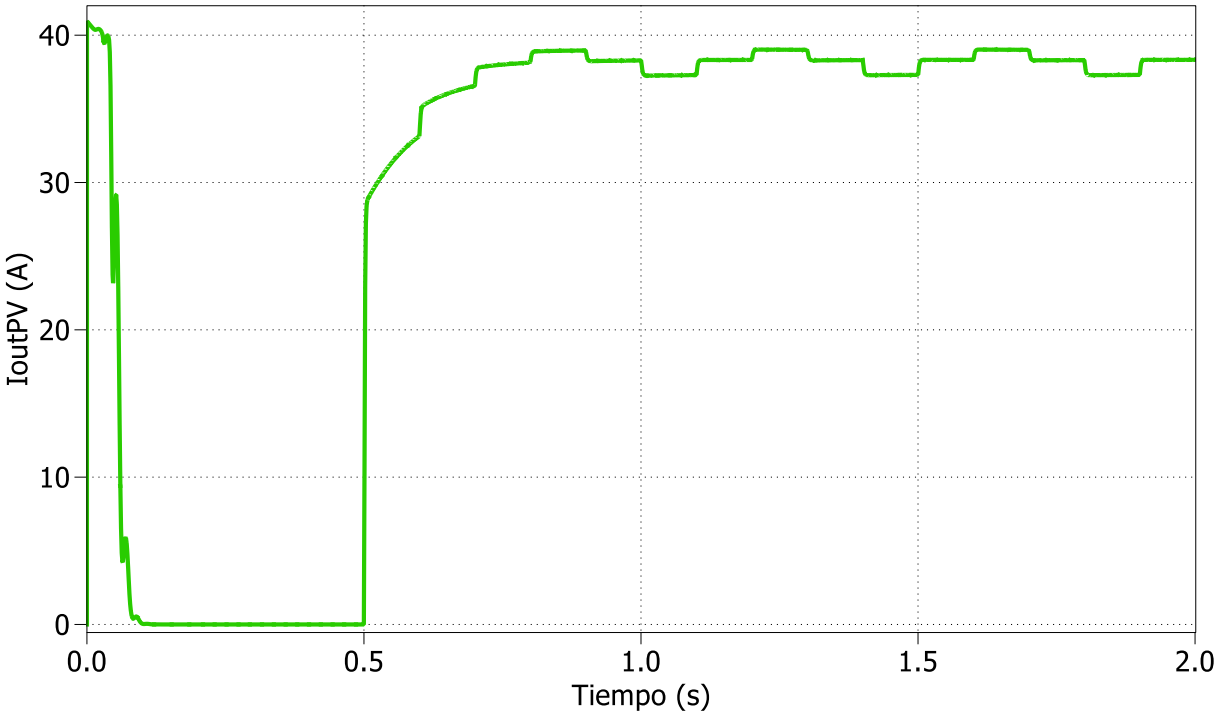

Figura 5-4. Evolución temporal de la corriente suministrada por la planta FV

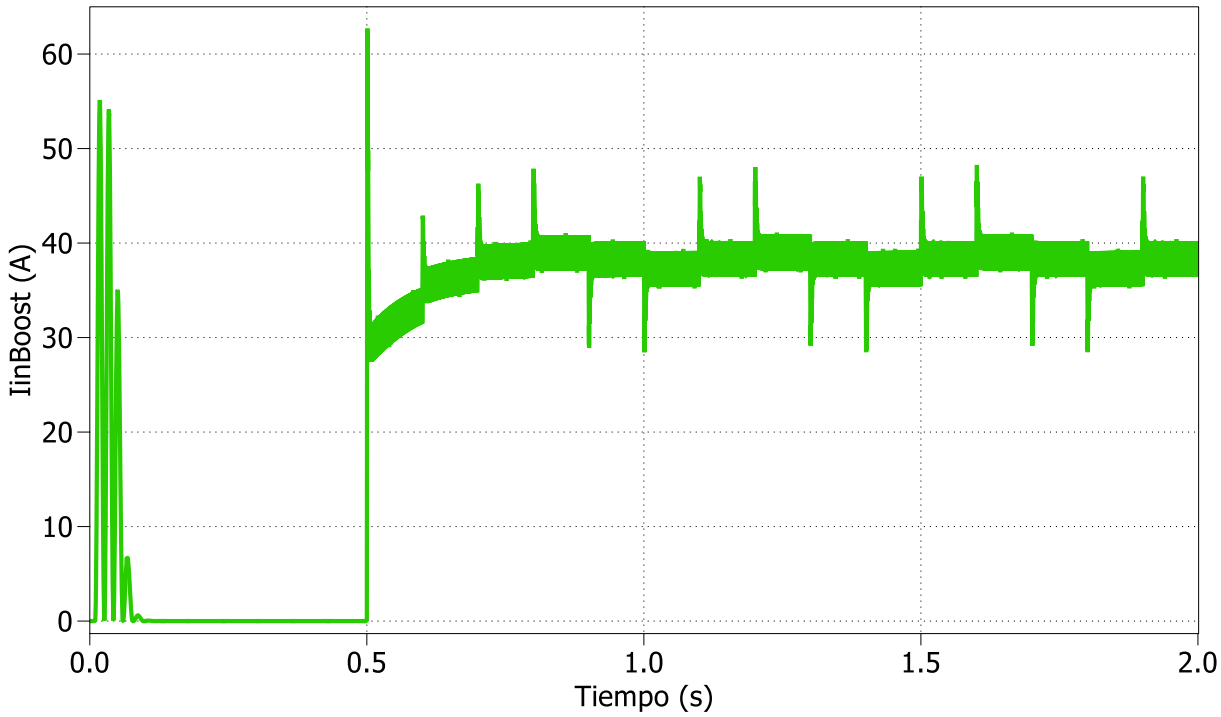

Figura 5-5. Evolución temporal de la corriente de la bobina del convertidor

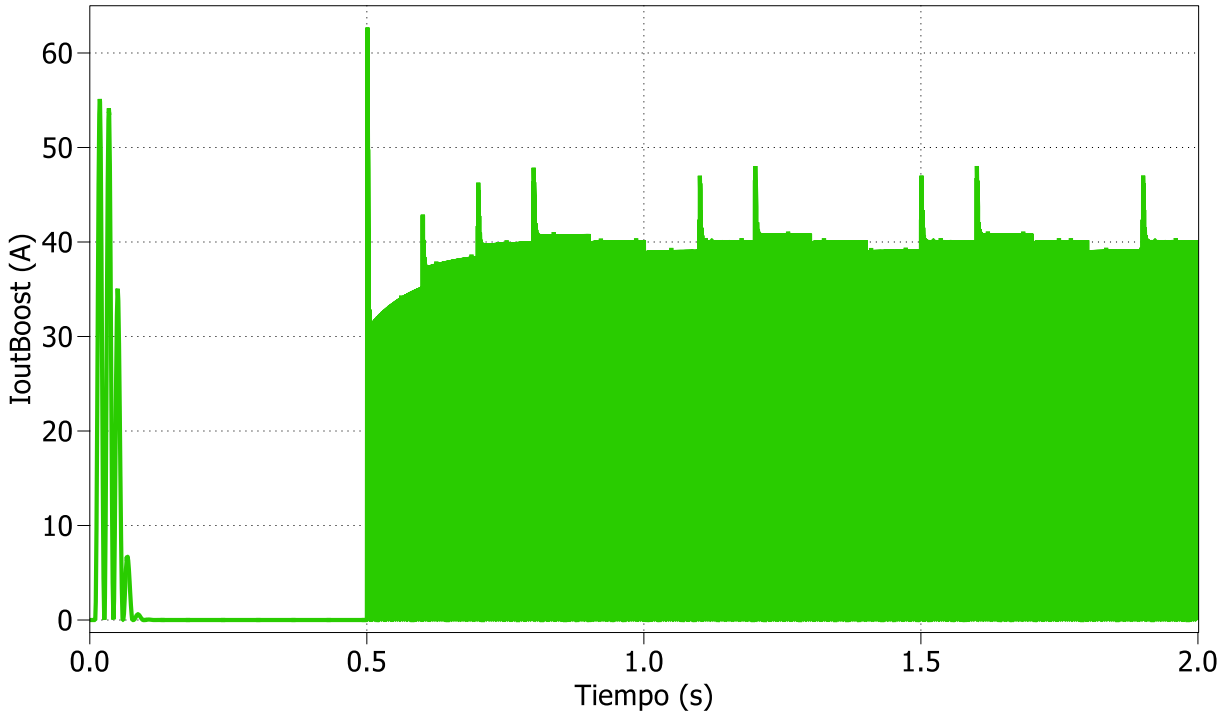

Figura 5-6. Evolución temporal de la corriente de salida del convertidor

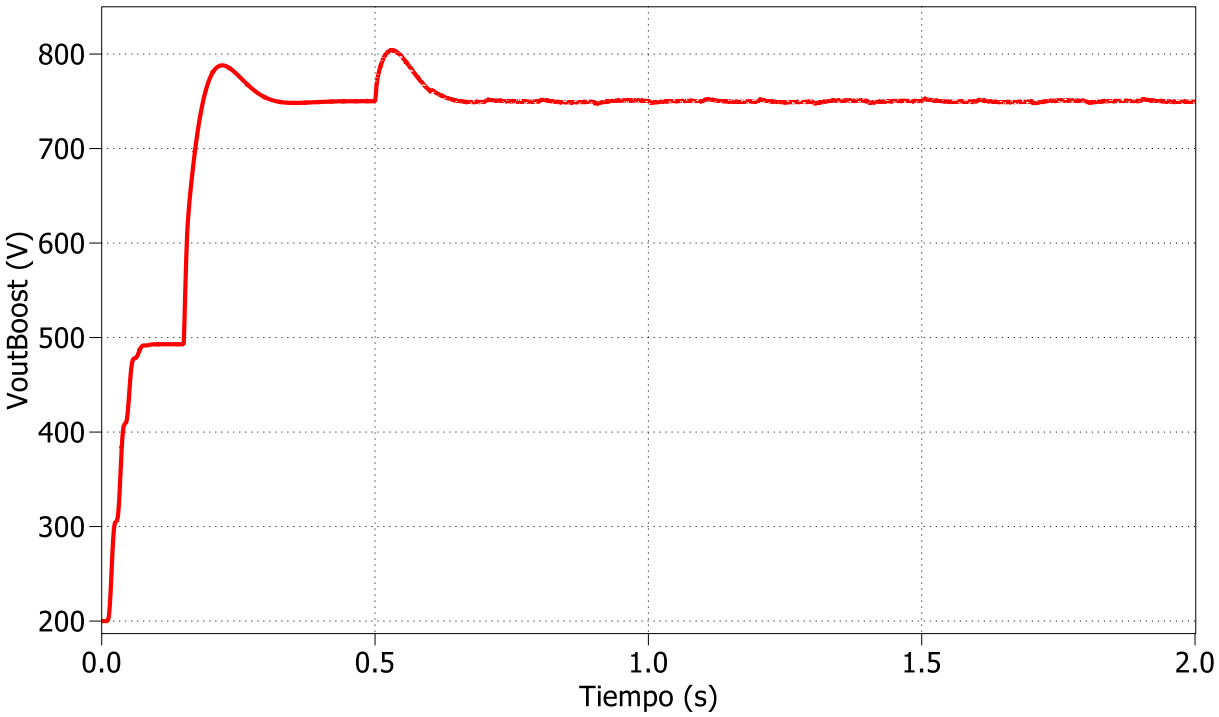

Figura 5-7. Evolución temporal de la tensión de salida del convertidor

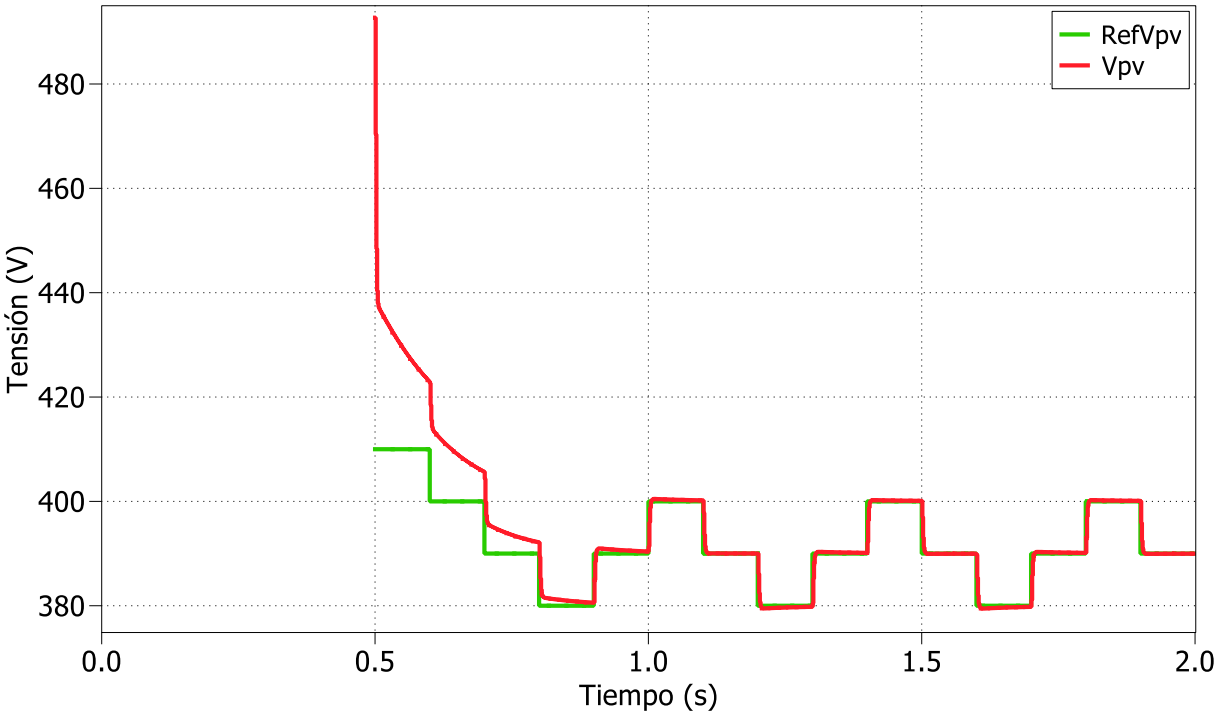

Figura 5-8. Evolución temporal de la tensión de la planta FV y de la referencia de tensión proporcionada por el algoritmo MPPT

En la respuesta temporal obtenida se distinguen claramente dos intervalos de funcionamiento. Durante el primero, correspondiente al arranque del sistema, la tensión del generador aumenta rápidamente hasta situarse próxima a la tensión en circuito abierto, mientras que la corriente suministrada por la planta desciende prácticamente a cero. Este comportamiento resulta coherente con una situación inicial en la que el generador fotovoltaico todavía no entrega potencia significativa al convertidor.

A partir del instante de habilitación del control, situado en t = 0,5 s, se observa un cambio claro en el punto de

operación del sistema. Tal como se aprecia en la Figura 5-3, la tensión de entrada del convertidor desciende desde valores próximos a circuito abierto hasta estabilizarse en el entorno de 380-400 V. Al mismo tiempo, según se muestra en la Figura 5-4, la corriente entregada por la planta aumenta progresivamente hasta alcanzar valores cercanos a 38-39 A. Esta evolución indica que, una vez activado el control, el generador pasa de una condición prácticamente sin carga a un régimen de extracción de potencia compatible con la operación en las proximidades del punto de máxima potencia.

Este comportamiento puede analizarse con mayor detalle mediante la comparación entre la tensión medida del generador y la referencia calculada por el algoritmo MPPT, mostrada en la Figura 5-8. Tras la habilitación del sistema, la referencia $RefVpv$ presenta variaciones escalonadas asociadas al proceso de búsqueda del punto de máxima potencia, mientras que la tensión real V_{pv} sigue dicha referencia con un pequeño transitorio debido a la dinámica del convertidor y de los lazos de control. En el intervalo inicial posterior a la activación, la tensión medida parte de un valor próximo a circuito abierto y converge progresivamente hacia la referencia generada por el MPPT. Una vez alcanzado el régimen permanente, ambas magnitudes permanecen próximas entre sí, lo que confirma que el sistema de control es capaz de desplazar el punto de operación del generador siguiendo la consigna establecida por el algoritmo de seguimiento.

Asimismo, las pequeñas oscilaciones periódicas que se observan en $RefVpv$ y en la tensión real del generador son coherentes con el funcionamiento del método MPPT implementado. En efecto, la introducción deliberada de pequeñas perturbaciones sobre la referencia permite explorar el entorno del punto de máxima potencia y mantener el generador operando en sus proximidades. Por tanto, estas variaciones no deben interpretarse como una anomalía del sistema, sino como una consecuencia inherente de la estrategia de seguimiento adoptada.

La corriente de la bobina, representada en la Figura 5-5, presenta el comportamiento característico del convertidor elevador en conmutación. Tras la habilitación del sistema aparece un transitorio inicial de mayor amplitud, seguido de un régimen estacionario con rizado superpuesto asociado al proceso de conmutación de los interruptores. La corriente de salida del convertidor, mostrada en la Figura 5-6, presenta una evolución similar en valor medio, aunque su representación aparece más densamente poblada debido al propio efecto de la conmutación y al punto de medida considerado. En ambos casos, la respuesta obtenida resulta coherente con la dinámica esperada de la etapa DC/DC.

Por su parte, la tensión de salida del convertidor, mostrada en la Figura 5-7, experimenta un incremento rápido durante el arranque y presenta un sobreimpulso inicial antes de estabilizarse en torno a 750 V, valor consistente con la referencia impuesta al enlace de continua. Tras la activación del control, se aprecia un nuevo transitorio de menor duración, a partir del cual la tensión permanece regulada alrededor de dicho valor con pequeñas oscilaciones de baja amplitud. Este resultado confirma el correcto acoplamiento entre la planta fotovoltaica, el convertidor elevador y el bus de continua del sistema.

En conjunto, los resultados obtenidos validan el modelo base desarrollado en PLECS, ya que muestran una respuesta dinámica coherente tanto en el arranque como en régimen permanente. La evolución de las variables medidas confirma que la planta fotovoltaica, la etapa elevadora y el enlace de continua interactúan adecuadamente dentro del entorno de simulación, constituyendo así una base sólida para la posterior implementación detallada de los bloques de control.

5.2. Implementación de los lazos de control del convertidor mediante bloques C-Script

Una vez definido el modelo base del sistema en PLECS, el siguiente paso consiste en implementar mediante bloques C-Script los lazos de control asociados al convertidor elevador. Esta modificación no altera la estructura general del subsistema descrita en el apartado anterior, ya que la etapa de potencia, el bloque PWM y la lógica de habilitación se mantienen sin cambios. La actuación se centra, por tanto, en sustituir la implementación previa de los lazos de control por una formulación programada explícitamente, más flexible y más próxima a una futura ejecución embebida.

En este contexto, el uso de bloques C-Script permite trasladar al entorno de simulación las ecuaciones de control de forma directa, definiendo explícitamente las variables de entrada y salida, la actualización de estados internos y la secuencia de cálculo de cada controlador. Sobre esta base, en los apartados siguientes se describe la

implementación del lazo externo y del lazo interno del convertidor, así como la validación de la respuesta obtenida tras esta modificación.

5.2.1 Consideraciones generales de implementación

La estructura de control del convertidor elevador se mantiene organizada en torno a tres funciones principales: la obtención de la referencia de operación del generador fotovoltaico mediante el algoritmo MPPT, el lazo externo de control, encargado de generar una consigna intermedia, y el lazo interno de control, responsable de producir la señal de modulación aplicada al bloque PWM. La novedad introducida en esta etapa reside en que los dos lazos de control se implementan ahora mediante bloques C-Script.

Desde el punto de vista funcional, las variables medidas del convertidor y de la planta fotovoltaica se muestrean y se entregan al bloque de control, que procesa dichas señales y calcula la acción de control correspondiente. La salida final de esta estructura es la señal de modulación aplicada al bloque PWM descrito en el apartado 5.1.1. Por tanto, la modificación introducida no afecta al papel del bloque de modulación dentro del modelo, sino al modo en que se obtiene la referencia que lo alimenta.

Antes de abordar la implementación específica de los lazos de control mediante bloques C-Script, resulta conveniente presentar la estructura funcional general del bloque Enabled boost-Control. Como se observa en la Figura 5-9, dicho bloque se organiza entorno al algoritmo MPPT, el lazo externo de control y el lazo interno de control, estableciéndose entre ellos el flujo de señales necesario para generar la referencia de modulación aplicada posteriormente al bloque PWM. Sobre esta organización general se introduce posteriormente la implementación mediante C-Script de los lazos externo e interno, manteniéndose inalterada la estructura funcional del subsistema de control.

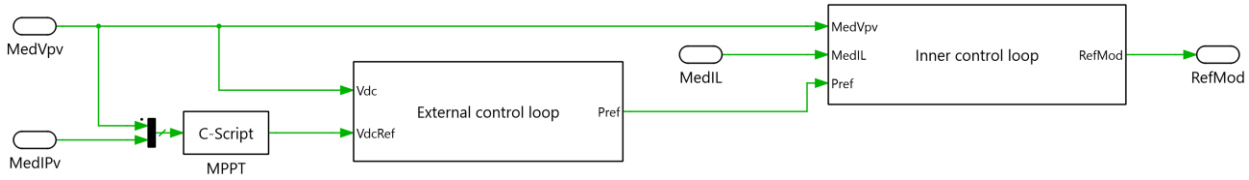

Figura 5-9. Estructura funcional del bloque Enabled boost-Control implementado en PLECS

La implementación mediante C-Script presenta además una ventaja metodológica importante, ya que permite concentrar en un único bloque la lógica de cálculo del controlador, la actualización de las variables internas y la gestión de sus entradas y salidas. De este modo, la estructura del modelo resulta más compacta y trazable, y se establece una correspondencia más directa entre el diseño teórico desarrollado en el capítulo 1 y su implantación efectiva en el entorno de simulación.

5.2.2 Implementación del lazo externo mediante C-Script

El lazo externo tiene como finalidad desplazar el punto de operación del generador fotovoltaico hacia la referencia de tensión calculada por el algoritmo MPPT. Para ello, el bloque implementado en C-Script recibe como entradas la tensión medida del generador y la referencia de tensión proporcionada por el bloque de seguimiento del punto de máxima potencia.

A partir de estas señales se calcula el error de control, que constituye la entrada del controlador PI diseñado en el capítulo 1. La acción proporcional y la acción integral se evalúan dentro del propio código, actualizando la variable de estado asociada al integrador y generando como salida la consigna intermedia empleada posteriormente por el lazo interno. De este modo, el bloque C-Script reproduce de forma explícita la secuencia de cálculo del controlador, evitando la necesidad de descomponer su implementación en múltiples bloques elementales.

La adopción de esta solución permite, además, mantener en una única estructura la lógica de evaluación del error, el cálculo de la acción de control y la actualización de estados internos. Esto simplifica la organización del modelo y facilita la correspondencia entre la formulación analítica del controlador y su realización práctica en PLECS. En consecuencia, el lazo externo implementado mediante C-Script conserva la misma finalidad funcional que en el modelo previo, pero con una estructura más compacta y más próxima a una futura implementación sobre procesador.

5.2.3 Implementación del lazo interno mediante C-Script

El lazo interno se encarga de regular la dinámica rápida del convertidor elevador a partir de la consigna generada por el lazo externo. En el esquema adoptado, esta etapa recibe la referencia intermedia procedente del lazo externo y las magnitudes medidas necesarias para construir la variable de error asociada a la corriente del convertidor.

De acuerdo con la estrategia de control desarrollada previamente, a partir de la salida del lazo externo se obtiene la referencia utilizada por el controlador interno. En particular, la referencia de corriente se calcula a partir de la expresión discreta $I_{pv}^{ref}[k] = \frac{P^{ref}[k]}{V_{pv}[k]}$. En esta operación se incorpora una protección frente a valores reducidos de la tensión $V_{pv}[k]$, con el fin de evitar problemas numéricos asociados a la división y garantizar un cálculo robusto de la consigna, especialmente durante el arranque o en condiciones transitorias. Esta referencia se compara con la magnitud medida correspondiente, generando el error que alimenta al controlador PI del lazo interno. Al igual que en el caso anterior, el controlador se implementa mediante un bloque C-Script, en el que se evalúan de forma explícita la acción proporcional, la acción integral y la actualización del estado interno del controlador.

La salida de este bloque constituye la señal de modulación que se aplica al bloque PWM del convertidor. Por tanto, el lazo interno representa la última etapa de cálculo dentro de la estructura de control, ya que traduce la consigna generada por el lazo externo en una señal apta para gobernar la conmutación de la etapa de potencia.

La implementación mediante C-Script resulta especialmente adecuada en este nivel, dado que permite recoger en un único bloque la secuencia completa de cálculo del controlador y mantener una organización clara de las variables implicadas. Además, al tratarse del lazo más rápido de la estructura de control, esta formulación facilita el seguimiento del flujo de señales y la validación de la respuesta dinámica del convertidor dentro del entorno PLECS.

5.2.4 Validación de la implementación mediante simulación

Una vez implementados el lazo externo y el lazo interno mediante bloques C-Script, se verifica que esta modificación no altera el comportamiento global del sistema previamente validado con el modelo base. Para ello, se representan nuevamente las principales magnitudes eléctricas analizadas en el apartado 5.1.2, ahora obtenidas con el modelo en el que ambos controladores PI se encuentran implementados mediante esta estrategia.

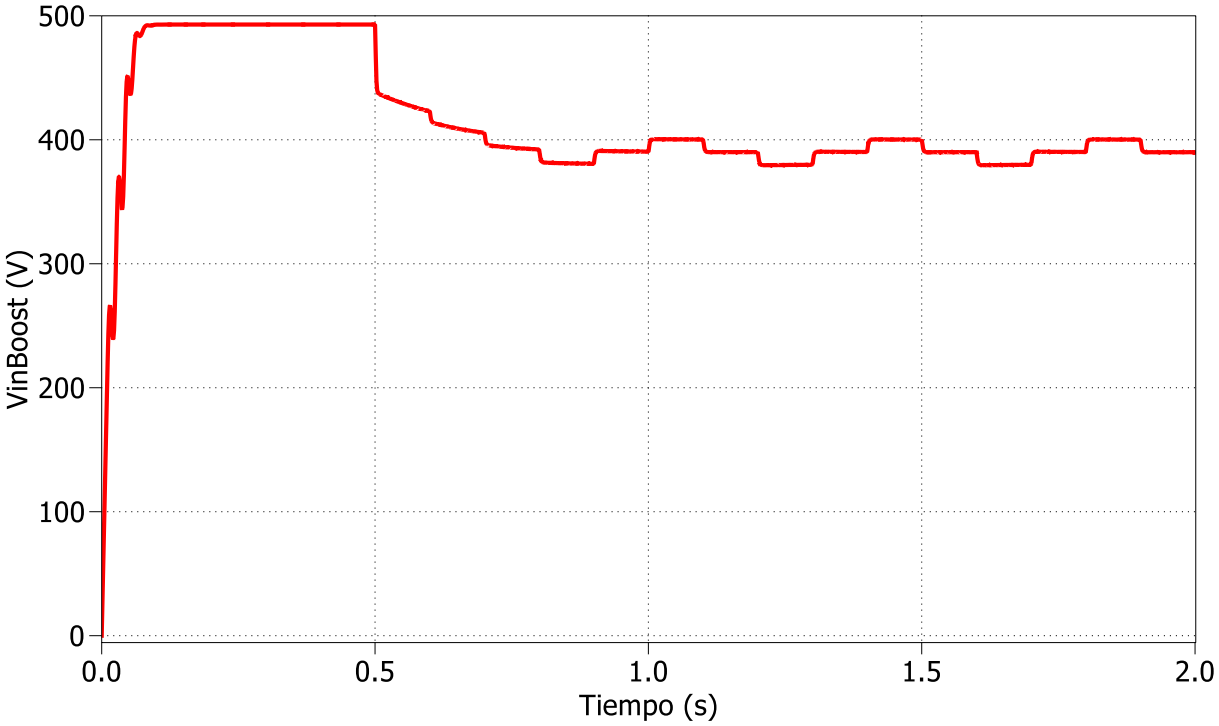

Figura 5-10. Evolución temporal de la tensión de entrada del convertidor obtenida con la implementación en C-Script de los lazos de control

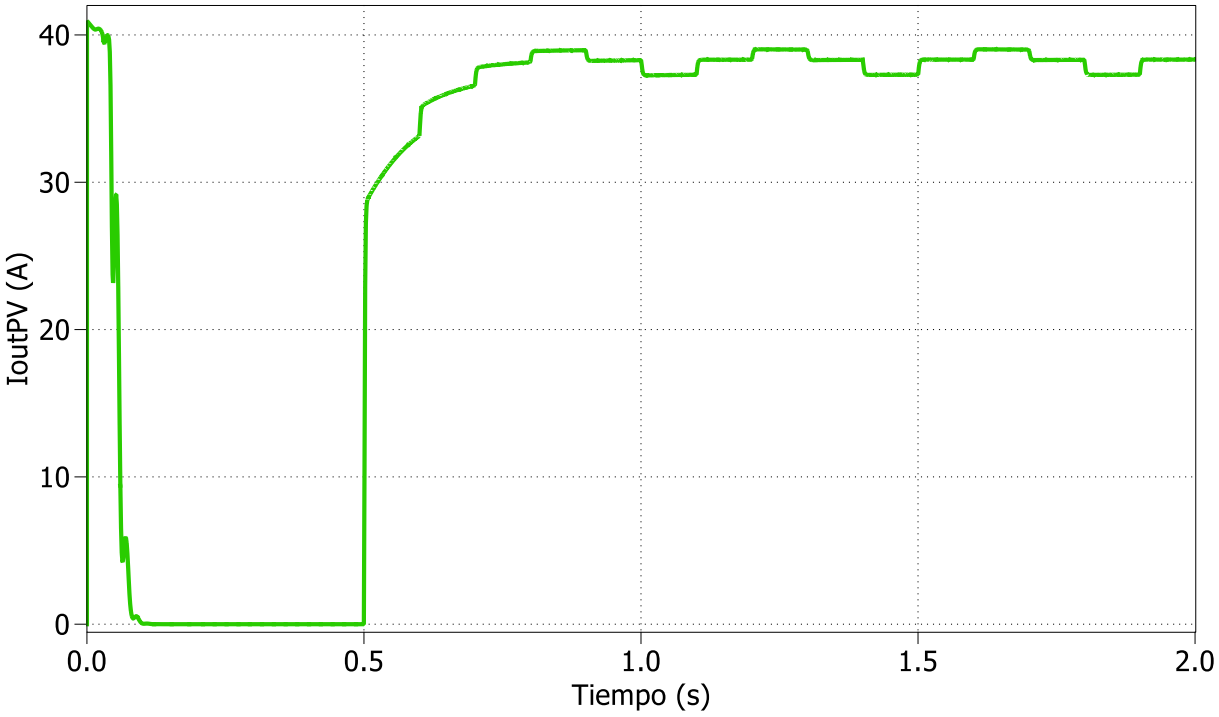

Figura 5-11. Evolución temporal de la corriente suministrada por la planta FV obtenida con la implementación en C-Script de los lazos de control

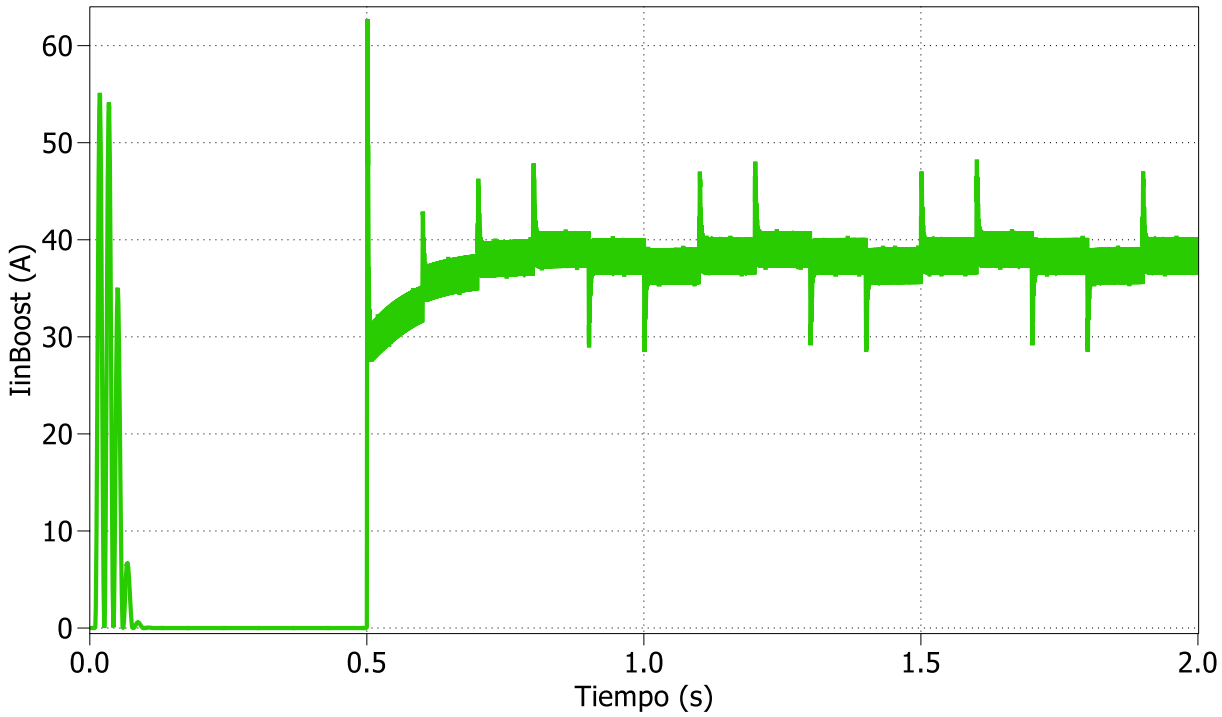

Figura 5-12. Evolución temporal de la corriente de la bobina del convertidor obtenida con la implementación en C-Script de los lazos de control

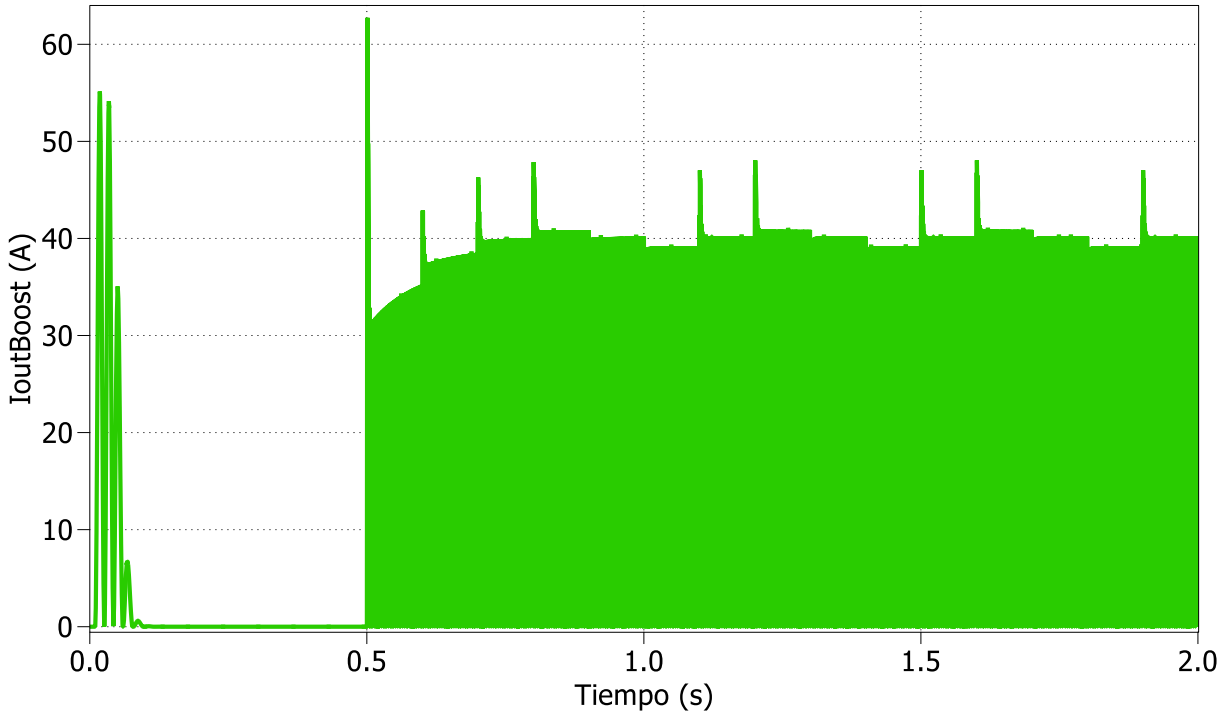

Figura 5-13. Evolución temporal de la corriente de salida del convertidor obtenida con la implementación en C-Script de los lazos de control

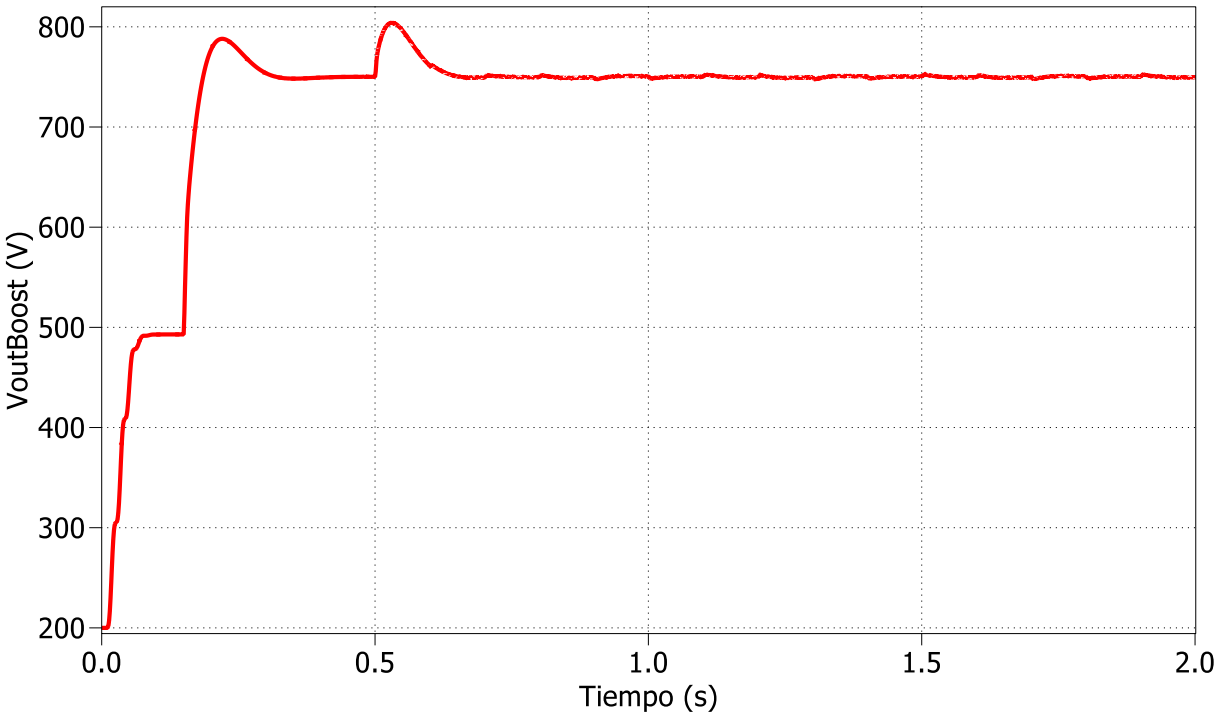

Figura 5-14. Evolución temporal de la tensión de salida del convertidor obtenida con la implementación en C-Script de los lazos de control

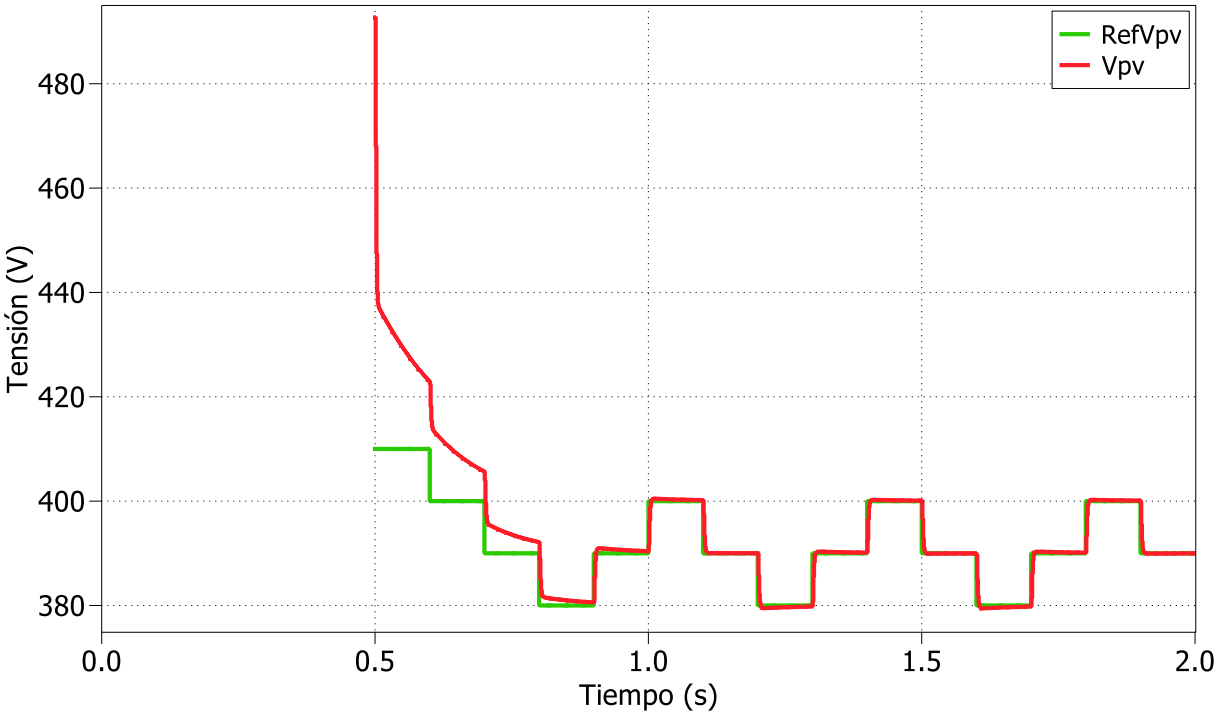

Figura 5-15. Evolución temporal de la tensión de la planta FV y su referencia proporcionada por el MPPT obtenida con la implementación en C-Script de los lazos de control

El análisis de los resultados muestra que la nueva implementación mantiene la evolución esperada de las variables principales del convertidor. En particular, la tensión del generador fotovoltaico sigue adecuadamente la referencia calculada por el algoritmo MPPT, mientras que la corriente de la planta, la corriente de la bobina y la tensión de salida del convertidor presentan una respuesta coherente con la observada previamente.

Asimismo, no se aprecian desviaciones significativas ni en el transitorio ni en régimen permanente que puedan

asociarse al cambio de implementación de los lazos de control. Por tanto, la sustitución de los bloques previos por una formulación basada en C-Script conserva la coherencia funcional del modelo y valida esta nueva realización del control en el entorno PLECS.

Una vez validada en simulación la implementación de los lazos de control mediante bloques C-Script, el siguiente paso consiste en trasladar esta estrategia a un esquema de verificación Processor-in-the-Loop, con objeto de evaluar su comportamiento en condiciones de ejecución más próximas a las de la plataforma embebida objetivo.

6 PROCESSOR IN THE LOOP (PIL)

Una vez validada en simulación la estrategia de control implementada en PLECS, el siguiente paso consiste en trasladar dicha estrategia a un entorno de verificación más próximo a su ejecución sobre la plataforma embebida objetivo. Para ello, en este capítulo se adopta la metodología Processor-in-the-Loop (PIL), en la que la planta permanece simulada en PLECS mientras que el algoritmo de control se ejecuta sobre el microcontrolador objetivo, en este caso el TI C2000 F28069M.

Este enfoque permite aumentar el realismo de la validación sin necesidad de recurrir a un entorno de implementación completo, al incorporar en el análisis aspectos propios de la ejecución embebida. En los apartados siguientes se describe el fundamento del entorno PIL, su integración en la plataforma empleada, los bloques utilizados para el intercambio de señales y los resultados obtenidos.

6.1. Fundamento y esquema de funcionamiento del entorno PIL

En el desarrollo de controladores para electrónica de potencia, una primera validación suele realizarse íntegramente en simulación, de modo que el algoritmo de control se ejecuta dentro del propio entorno y actúa sobre un modelo matemático de la planta. Este planteamiento resulta adecuado para verificar la lógica general del control y comprobar la coherencia de las ecuaciones implementadas. Sin embargo, tiende a idealizar determinados aspectos que, en una implementación real, condicionan de forma directa la respuesta del sistema, tales como la representación numérica, las saturaciones, los retardos de cálculo o la temporización impuesta por interrupciones y periféricos.

Con objeto de reducir esta diferencia entre simulación e implementación embebida, en este trabajo se recurre a una metodología Processor-in-the-Loop (PIL). En este esquema, la planta continúa ejecutándose en PLECS, mientras que el algoritmo de control se genera, compila y ejecuta sobre el microcontrolador objetivo, que en este caso corresponde a un TI C2000 F28069M. De este modo, el lazo de control se cierra mediante el intercambio de señales entre el entorno de simulación y el microcontrolador: PLECS proporciona al código de control variables equivalentes a medidas eléctricas, y el microcontrolador devuelve la acción de control calculada para su aplicación sobre el modelo del convertidor.

La utilización de esta estrategia permite validar el mismo código de control que posteriormente se pretende emplear en la implementación final, manteniendo al mismo tiempo un entorno de ensayo controlado, repetible y seguro. Así, la metodología PIL introduce una capa adicional de realismo respecto a la simulación convencional, ya que permite tener en cuenta efectos asociados a la ejecución sobre el microcontrolador sin necesidad de abandonar el entorno de simulación.

En el caso particular de este trabajo, la planta simulada en PLECS incluye el generador fotovoltaico, el convertidor elevador y los elementos asociados a su operación. Por su parte, el microcontrolador ejecuta el algoritmo de control completo, incluyendo el MPPT, los lazos PI y la lógica auxiliar necesaria para gobernar el convertidor. La interacción entre ambos dominios se realiza mediante un conjunto de variables de intercambio gestionadas por el bloque PIL, que actúa como interfaz entre la simulación y el microcontrolador. En este esquema se distinguen, por un lado, las señales impuestas desde PLECS al microcontrolador, que representan las medidas del sistema, y, por otro, las magnitudes calculadas por el código de control, que se devuelven al simulador para gobernar la planta.

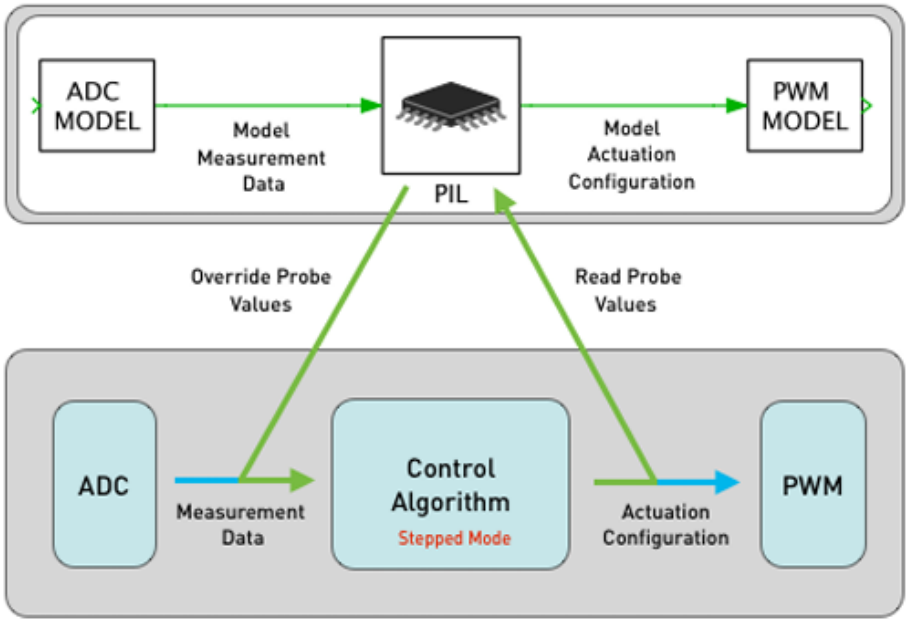

Figura 6-1. Esquema del funcionamiento del PIL [8]

La Figura 6-1 muestra de forma esquemática el funcionamiento de este entorno. En ella se observa que las magnitudes equivalentes a las medidas del sistema se transfieren desde PLECS hacia el microcontrolador, mientras que la acción del control calculada por el código de control se devuelve al simulador para excitar el modelo PWM y la etapa de potencia. De esta forma, la validación no se limita a comprobar la formulación teórica del algoritmo, sino que permite analizar también su integración efectiva con el microcontrolador y con la secuencia temporal de ejecución asociada al sistema embebido. Asimismo, este esquema implica una coordinación temporal entre la simulación y la ejecución del código de control, de modo que las magnitudes intercambiadas entre ambos dominios correspondan al mismo instante discreto de operación.

A partir de este esquema general, en los apartados siguientes se describe con mayor detalle la integración del entorno PIL en la plataforma empleada, los bloques utilizados para el intercambio de señales y la configuración adoptada para la validación del control sobre el microcontrolador objetivo.

6.2. Integración del entorno PIL en la plataforma empleada

La implementación del entorno PIL se realiza empleando como plataforma de ejecución un microcontrolador TI C2000 F28069M, integrado en la tarjeta LAUNCHXL-F28069M, mientras que la planta y el resto del entorno de simulación se mantienen en PLECS. De este modo, el sistema queda dividido en dos dominios claramente diferenciados: por una parte, el modelo de la planta, que continúa ejecutándose en el entorno de simulación, y por otra, el código de control, que se genera, compila y ejecuta sobre el microcontrolador objetivo.

Para llevar a cabo esta integración, el flujo de trabajo se apoya en la generación automática de código desde PLECS y en su compilación mediante Code Composer Studio (CCS), entorno de desarrollo utilizado para la plataforma TI C2000. A través de este procedimiento, el algoritmo de control validado previamente en simulación se traslada a una estructura de ejecución embebida, permitiendo verificar su funcionamiento sobre el microcontrolador sin modificar la planta simulada.

Desde el punto de vista funcional, la integración del entorno PIL requiere establecer un intercambio ordenado de señales entre ambos dominios. Así, PLECS proporciona al microcontrolador las magnitudes necesarias para la ejecución del control, equivalentes a las variables medidas del sistema, mientras que el microcontrolador devuelve al entorno de simulación la acción de control calculada. Esta acción se emplea posteriormente para gobernar el modelo del convertidor y cerrar el lazo de control dentro del esquema PIL.

Antes de su envío al microcontrolador, las magnitudes eléctricas calculadas en PLECS se adaptan al rango de trabajo del convertidor analógico-digital. Para ello, las señales de tensión y corriente se escalan de forma que resulten compatibles con la entrada del ADC. Posteriormente, dentro del entorno embebido, el bloque ADC

aplica el reescalado inverso para recuperar las magnitudes físicas empleadas por el algoritmo de control. De este modo, el intercambio entre simulación y microcontrolador respeta el rango de operación del periférico sin alterar la coherencia de las variables utilizadas en el control. La descripción detallada de esta etapa de adaptación de medidas se presenta en el apartado 6.3.4, dedicado al bloque ADC A.

La Figura 6-2 muestra el esquema general de la implementación PIL desarrollada en este trabajo. En ella se aprecia cómo las señales calculadas en PLECS se introducen en el microcontrolador mediante los bloques de intercambio correspondientes, mientras que las variables generadas por el código de control se recuperan para su aplicación sobre la etapa de potencia simulada. Asimismo, se aprecia la integración de los bloques asociados a adquisición y actuación, lo que permite reproducir de forma coherente el flujo de señales entre la planta simulada y el sistema embebido.

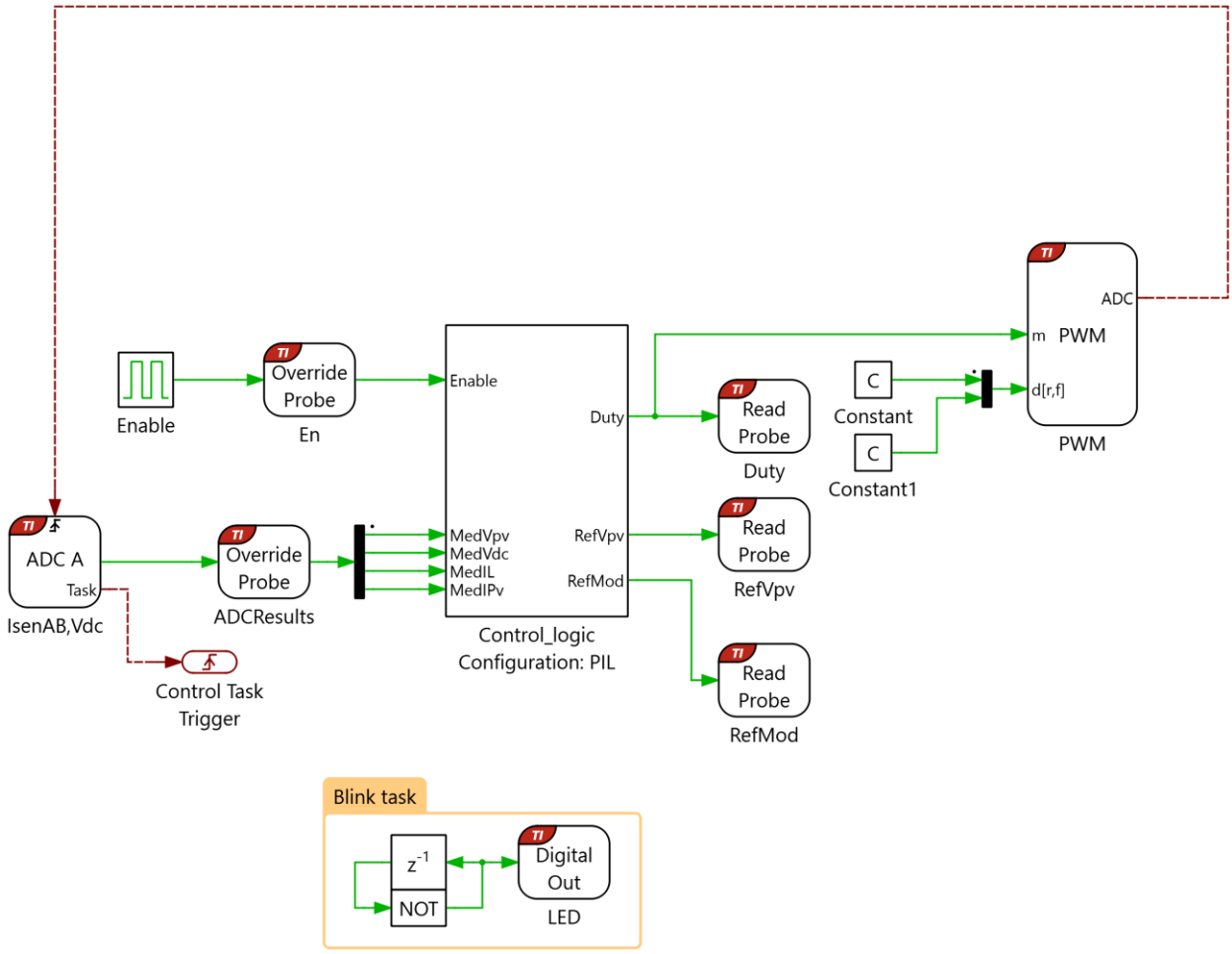

Figura 6-2. Esquema de la implementación PIL desarrollada

En la configuración adoptada en este trabajo, las señales transferidas desde PLECS al microcontrolador corresponden a las magnitudes medidas necesarias para el control del convertidor, concretamente $MedVpv$, $MedIPv$, $MedIL$, $MedVdc$ y la señal de habilitación $Enable$. Como variables de salida se recuperan, entre otras, la referencia de tensión $RefVpv$, la referencia de modulación $RefMod$ y la señal de actuación $Duty$, empleada para gobernar la etapa PWM simulada.

Adicionalmente, se incluye una tarea auxiliar de señalización, utilizada con fines de supervisión del funcionamiento del microcontrolador durante la ejecución del entorno PIL.

Esta organización permite validar no solo el diseño del algoritmo, sino también su correcta integración con la plataforma de ejecución seleccionada. Asimismo, constituye una etapa especialmente útil dentro del proceso de desarrollo, ya que permite comprobar el comportamiento del código de control en condiciones más próximas a las de una implementación embebida que las de una simulación convencional.

6.2.1 Configuración del entorno de generación de código

Además del esquema de intercambio de señales descrito en el apartado anterior, la implementación del entorno PIL requiere la configuración del subsistema de control mediante las opciones de Coder Options de PLECS. En este trabajo, dicha configuración se realiza para la familia objetivo TI2806x, seleccionando en particular el chip TMS320F28069. Esta elección resulta coherente con la plataforma hardware empleada y determina tanto las opciones de generación de código como la configuración de los periféricos y del modo de ejecución externo.

En la pestaña General se adopta una representación en coma flotante de tipo float y se fija la opción Use internal oscillator, con una frecuencia de reloj del sistema de 90 MHz. Asimismo, se selecciona la política Enforce exact value para la tolerancia del paso de discretización, con objeto de mantener la coherencia entre el periodo de muestreo definido en el modelo y la frecuencia efectiva de ejecución del control sobre el microcontrolador. Esta decisión resulta especialmente relevante, ya que la precisión temporal del sistema embebido viene limitada por la frecuencia SYSCLK, de la que dependen tanto la temporización de las tareas como la generación de las señales PWM.

En cuanto al proceso de compilación, se selecciona como Build type la opción Generate code into CCS project. De este modo, PLECS genera el código del subsistema de control dentro del proyecto previamente importado en Code Composer Studio (CCS), concretamente en el directorio correspondiente a la carpeta cg. Esta estrategia permite compilar, depurar y programar el microcontrolador desde CCS, aprovechando sus herramientas de desarrollo y depuración. Además, el propio manual indica que, al emplear esta modalidad, el código generado debe dirigirse a la carpeta cg del proyecto CCS importado.

Dentro de la pestaña Target, se mantiene la referencia del ADC A en Internal 3,3 V, en concordancia con el rango de operación habitual del periférico. Por su parte, en la configuración del External Mode se selecciona la opción Serial, con un Target buffer size de 1000 y empleando la interfaz SCI_DSP. Para dicha comunicación se configuran los pines GPIO [28, 29], correspondientes a recepción y transmisión, respectivamente. Esta elección coincide con la configuración recomendada para la tarjeta LAUNCHXL-F28069, donde la comunicación en modo externo se establece precisamente a través de ese par de pines.

Desde el punto de vista temporal, en la pestaña Scheduling se selecciona el modo multi-tasking. En esta configuración se define una Base task, asociada a la ejecución del control, con un periodo de muestreo $T_{s,controller} = 0{,}0001\ s$, y una tarea auxiliar Blink task con periodo de 0,5 s. De este modo, la tarea base se ejecuta cada 100 µs, mientras que la tarea auxiliar se reserva para funciones de señalización de menor prioridad. Según el manual, en modo multi-tarea la Base task posee la prioridad más alta, mientras que las tareas adicionales deben definirse con periodos que sean múltiplos enteros del periodo base. Esta organización resulta adecuada para preservar la ejecución determinista del control y separar de ella tareas no críticas temporalmente.

En conjunto, esta configuración permite mantener la coherencia entre el modelo desarrollado en PLECS, el código generado automáticamente y la plataforma de ejecución empleada. Sobre esta base se construye el entorno PIL utilizado en este trabajo, en el que el código de control se ejecuta sobre el microcontrolador TI C2000 F28069M mientras la planta permanece simulada en PLECS.

6.3. Bloques empleados en la implementación PIL

En este apartado se describen los principales bloques empleados para integrar el entorno Processor-in-the-Loop dentro del modelo desarrollado en PLECS. Estos bloques permiten establecer la comunicación entre la simulación y el microcontrolador, redirigir señales entre ambos dominios y reproducir, dentro del entorno PIL, el comportamiento asociado a los periféricos de adquisición y actuación del sistema de control.

Desde el punto de vista funcional, la implementación PIL requiere, por una parte, un bloque capaz de ejecutar el código de control sobre el microcontrolador objetivo y, por otra, un conjunto de bloques auxiliares que permitan intercambiar señales entre dicho código y la planta simulada. En este contexto, adquieren especial relevancia el bloque PIL, los bloques Override Probe y Read Probe, así como los bloques asociados a los periféricos, tales como el ADC y el PWM.

La utilización conjunta de estos elementos permite reproducir de forma coherente la interacción entre las variables medidas del sistema, la ejecución del algoritmo de control y la generación de las señales de actuación.

De este modo, el esquema PIL no se limita a ejecutar el código sobre el microcontrolador, sino que reproduce también la lógica de intercambio de información necesaria para cerrar el lazo de control con la planta simulada.

A continuación, se describe la función de cada uno de estos bloques y la configuración adoptada en el presente trabajo para su integración dentro del entorno PIL.

6.3.1 Bloque PIL

El bloque PIL constituye el elemento central del esquema de validación, ya que es el encargado de enlazar el entorno de simulación con la ejecución del código de control sobre el microcontrolador objetivo. Su función consiste en transferir al microcontrolador las señales procedentes de la planta simulada y devolver a PLECS las variables de salida calculadas por el algoritmo de control. De este modo, el bloque actúa como interfaz principal entre ambos dominios.

En el esquema desarrollado, el bloque PIL recibe como entradas las magnitudes eléctricas necesarias para la ejecución del control, equivalentes a las variables medidas del sistema. Estas señales son enviadas al microcontrolador, donde se ejecuta el código correspondiente al MPPT, a los lazos PI y a la lógica auxiliar del convertidor. Como salida, el bloque devuelve al entorno de simulación la acción de control calculada, que posteriormente se aplica sobre la etapa de modulación y sobre la planta simulada.

La Figura 6-3 muestra el bloque PIL empleado en este trabajo y las señales principales intercambiadas entre PLECS y el microcontrolador. Como puede observarse, este bloque no representa únicamente un canal de comunicación, sino la frontera funcional entre la planta simulada y el código de control ejecutado sobre el hardware objetivo.

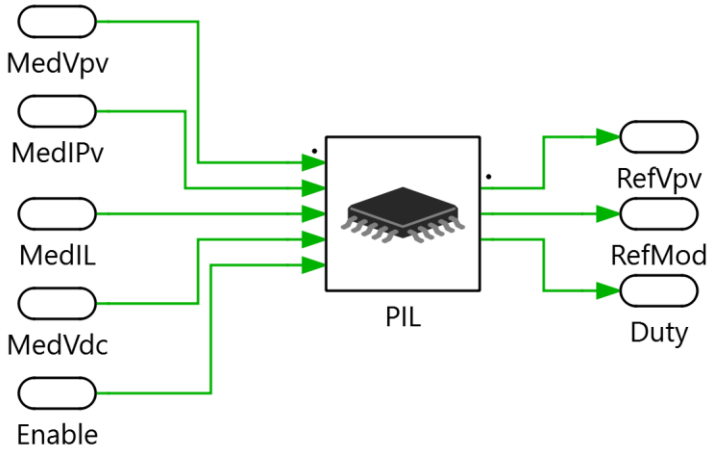

Figura 6-3. Bloque PIL empleado para el intercambio de señales entre PLECS y el microcontrolador

En cuanto a su configuración, el bloque se asocia al flujo de generación de código y compilación empleado para la plataforma TI C2000 F28069M, de manera que las variables definidas en PLECS quedan vinculadas con las señales correspondientes del código de control. Gracias a ello, el entorno PIL permite verificar el comportamiento del mismo código que posteriormente se pretende utilizar en la implementación embebida final, manteniendo al mismo tiempo la planta dentro de un entorno de simulación controlado.

6.3.2 Bloque Override Probe

El bloque Override Probe se utiliza para imponer desde PLECS el valor de determinadas variables que, en una ejecución convencional del código, procederían de las entradas de medida o de señales externas del sistema. Su función dentro del entorno PIL consiste, por tanto, en sustituir lecturas reales por valores calculados en la simulación, manteniendo sin cambios la estructura general del código de control.

En el esquema adoptado, este bloque se emplea para introducir en el microcontrolador las magnitudes equivalentes a las medidas de tensión y corriente calculadas por la planta simulada, así como la señal de habilitación del sistema. De este modo, el código embebido recibe internamente variables análogas a las que tendría disponibles durante su ejecución sobre el microcontrolador, aunque en este caso son generadas por el

entorno de simulación.

La Figura 6-4 muestra el bloque Override Probe utilizado en el entorno PIL. Su incorporación resulta esencial para mantener la coherencia entre el modelo de planta ejecutado en PLECS y el código de control que se ejecuta sobre el microcontrolador, ya que permite alimentar dicho código con señales simuladas sin modificar su estructura interna.

Figura 6-4. Bloque Override Probe

En cuanto a su configuración, este bloque se vincula a las variables del código que deben ser sobreescritas por los valores procedentes de PLECS. Así, el Override Probe actúa como punto de entrada de las magnitudes simuladas dentro del entorno embebido, permitiendo reproducir de forma fiel la interacción entre la planta y el sistema de control.

6.3.3 Bloque Read Probe

El bloque Read Probe permite recuperar en PLECS variables generadas por el código de control que se ejecuta en el microcontrolador durante la simulación PIL. Su función complementa a la del Override Probe, ya que, mientras este último introduce señales en el entorno embebido, el Read Probe extrae de él las variables de salida necesarias para cerrar el lazo de control o para supervisar el funcionamiento interno del algoritmo.

En la implementación desarrollada, este bloque se utiliza para leer variables como la referencia de tensión $RefVpv$, la referencia de modulación $RefMod$ y la señal de actuación $Duty$. Estas magnitudes se devuelven al modelo de PLECS, donde se emplean para aplicar la acción de control a la etapa de potencia simulada y para observar el comportamiento interno del algoritmo ejecutado sobre el microcontrolador.

La Figura 6-5 muestra el bloque Read Probe empleado en este trabajo. Gracias a este mecanismo, el entorno de simulación puede acceder a variables calculadas por el código de control sin necesidad de alterar su estructura funcional, lo que facilita tanto la validación del comportamiento del sistema como la supervisión de magnitudes internas relevantes.

Figura 6-5. Bloque Read Probe

Desde el punto de vista de su configuración, el bloque se asocia a las variables del código que deben ser exportadas hacia PLECS. Esto permite recuperar no solo la acción de control final, sino también variables internas útiles para el análisis del funcionamiento del algoritmo durante la validación PIL.

6.3.4 ADC A

El bloque ADC A representa el periférico de conversión analógico-digital empleado por el microcontrolador para adquirir las señales de medida utilizadas por el control. En una implementación embebida real, este periférico recibe tensiones adaptadas al rango de operación del ADC y suministra al algoritmo de control las medidas adquiridas correspondientes. Dentro del entorno PIL, su presencia permite reproducir de forma más fiel la cadena de adquisición del sistema, incorporando tanto el comportamiento del periférico como la secuencia temporal con la que se actualizan las medidas. De acuerdo con el manual de PLECS, la salida del bloque representa la tensión medida en el pin del ADC y puede expresarse mediante una relación lineal del tipo $y = Input \cdot Scale + Offset$. Asimismo, cuando se define un vector de canales, las conversiones se realizan secuencialmente en el orden especificado.

En este trabajo, las magnitudes calculadas en PLECS no se entregan directamente al algoritmo de control con su valor físico original, sino que se adaptan previamente al rango de trabajo del ADC mediante una etapa de escalado. Posteriormente, una vez atravesado el bloque ADC, se aplica el reescalado inverso para recuperar las magnitudes físicas utilizadas por el control. De este modo, el intercambio entre la planta simulada y el microcontrolador respeta el rango de operación del periférico sin alterar la coherencia de las variables empleadas en el algoritmo. La Figura 6-6 muestra el bloque ADC A utilizado en la implementación PIL, mientras que la Figura 6-7 presenta la etapa de adaptación de medidas empleada para escalar las variables procedentes de la planta simulada. Esta doble operación de escalado y reescalado permite reproducir con mayor fidelidad el comportamiento de la cadena de medida del sistema embebido.

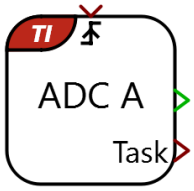

Figura 6-6. Bloque ADC A

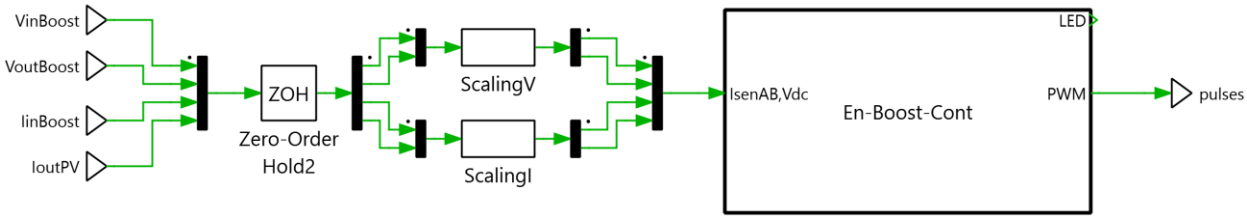

Figura 6-7. Etapa de adaptación de las señales medidas al rango de trabajo del ADC

Dado que el ADC del microcontrolador opera en un intervalo de entrada comprendido entre 0 y 3,3 V, resulta necesario adaptar previamente las señales de tensión y corriente del convertidor a dicho rango. Para ello, cada magnitud física se escala tomando como referencia su valor máximo esperado, de forma que el valor máximo de la magnitud física quede asociado al límite superior del rango de conversión del ADC. Así, el factor de adaptación se define como el cociente entre la tensión máxima admisible por el ADC y el valor máximo de la magnitud que se desea medir. De este modo, cuando la variable alcanza su valor máximo, la señal acondicionada a la entrada del ADC toma el valor de 3,3 V.

De forma general, para una magnitud de entrada x, el valor adaptado al rango del ADC puede expresarse como:

$$x_{ADC} = \frac{3{,}3}{x_{max}} x$$

donde x_{max} representa el valor máximo de la magnitud física considerada. En consecuencia, el término $\frac{3{,}3}{x_{max}}$ actúa como la ganancia de escalado necesaria para convertir el rango físico de la variable en el rango eléctrico admisible por el ADC, manteniendo una relación lineal entre ambos. De forma equivalente, esta adaptación puede escribirse como:

$$x_{ADC} = K_x x$$

donde K_x es la ganancia de adaptación correspondiente a cada variable. Posteriormente, dentro del entorno embebido, se aplica la operación inversa para reconstruir la magnitud física a partir del valor adquirido por el ADC.

En cuanto a su configuración, el bloque se ha definido con la unidad ADC A y con la fuente de disparo Show trigger port, de modo que el inicio de la conversión depende de una señal externa de disparo. En la implementación desarrollada, dicha señal de trigger procede del módulo PWM, lo que permite sincronizar la adquisición con la lógica temporal de conmutación y control del convertidor. Este aspecto se aborda con mayor detalle en el apartado dedicado a la configuración del PWM. Además, el tiempo de adquisición se ha fijado al valor mínimo permitido (Set to minimal value), con objeto de reducir el tiempo dedicado a la conversión de cada

canal. Según el manual, el parámetro de disparo permite seleccionar entre un inicio de conversión automático o externo y, cuando se emplean varios canales, estos se convierten secuencialmente siguiendo el orden del vector definido.

La configuración concreta del bloque empleada en este trabajo se muestra en la Figura 6-8, donde pueden observarse los parámetros principales asociados a la adquisición y al reescalado de las señales medidas en el entorno PIL.

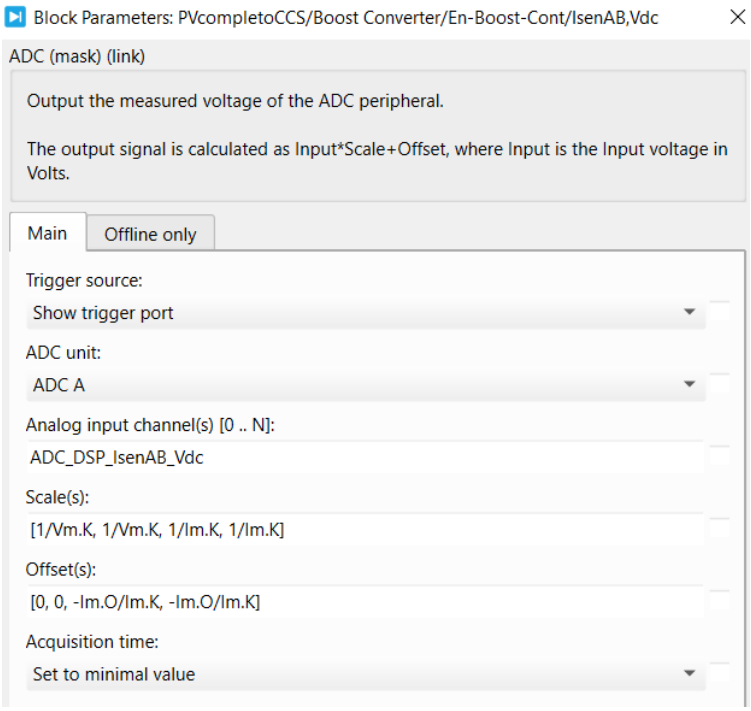

Figura 6-8. Configuración del bloque ADC A empleada en el entorno PIL

El vector de canales configurado en el modelo es:

$$ADC\_DSP\_IsenAB\_Vdc = [3, 3+8, 5+8, 2+8]$$

En el caso del microcontrolador TI28069, los canales pertenecientes al grupo ADCINBx se introducen también bajo la unidad ADC A, desplazando su índice en 8 posiciones. Por ello, la configuración anterior corresponde, en orden de muestreo, a los canales ADCINA3, ADCINB3, ADCINB5 y ADCINB2.

La adaptación inversa aplicada a la salida del bloque ADC se implementa mediante los parámetros Scale(s) y Offset(s), definidos como:

$$Scale(s) = [1/Vm.K, 1/Vm.K, 1/Im.K, 1/Im.K]$$

$$Offset(s) = [0,0, -Im.O/Im.K, -Im.O/Im.K]$$

siendo:

$$Vm.K = \frac{3{,}3}{805}, \qquad Im.K = \frac{3{,}3}{65}, \qquad Im.O = \frac{3{,}3}{2}$$

De acuerdo con los factores de escala y offset definidos, las dos primeras señales se reconstruyen como magnitudes de tensión sin término de offset, mientras que las dos últimas se obtienen como magnitudes de corriente con compensación del desplazamiento introducido en el acondicionamiento analógico. En consecuencia, las salidas asociadas a cada canal pueden expresarse como:

$$y_1 = \frac{1}{Vm.K} u_{ADC,1}, \qquad y_2 = \frac{1}{Vm.K} u_{ADC,2}$$

$$y_3 = \frac{1}{Im.K} u_{ADC,3} - \frac{Im.O}{Im.K}, \qquad y_4 = \frac{1}{Im.K} u_{ADC,4} - \frac{Im.O}{Im.K}$$

donde $u_{ADC,i}$ representa la tensión, en voltios, presente en cada entrada del convertidor. De este modo, el bloque aplica el reescalado necesario para recuperar las magnitudes físicas a partir de las señales previamente adaptadas al rango del ADC. En particular, las señales de corriente incorporan un término de compensación asociado al offset introducido en la etapa de acondicionamiento, correspondiente a la media escala del convertidor. Esto indica que las señales de corriente llegan al ADC con un desplazamiento de continua situado en torno a la media escala del convertidor, esto es, 1,65 V, lo cual resulta habitual cuando se desea medir corrientes bipolares mediante un ADC unipolar de 0 a 3,3 V.

En definitiva, el bloque ADC A no solo actúa como punto de entrada de las señales medidas al entorno embebido, sino que también reproduce el proceso de adquisición secuencial y el reescalado necesario para recuperar las magnitudes físicas empleadas por el algoritmo de control. Así, su utilización contribuye a mantener la correspondencia entre las variables simuladas en planta y las realmente procesadas por el microcontrolador durante la ejecución PIL.

6.3.5 PWM

El bloque PWM representa el periférico encargado de generar la señal de actuación aplicada al convertidor. En el entorno PIL, este bloque permite reproducir la lógica de modulación del sistema embebido, de manera que la acción de control calculada por el microcontrolador se traduzca en una señal equivalente a la utilizada para gobernar la etapa de potencia. Así, su inclusión no solo permite representar la señal de control final, sino también reproducir de forma más fiel la secuencia temporal de actuación del sistema embebido.

En la implementación desarrollada, la variable $Duty$ calculada por el código de control se recupera mediante un bloque Read Probe y se aplica a la entrada de modulación del bloque PWM del entorno PIL. A partir de esta señal, el bloque genera la modulación correspondiente y la integra dentro del esquema general de validación. De este modo, el lazo de control se cierra manteniendo una secuencia de actuación coherente con la estructura de una implementación real.

La Figura 6-9 muestra el bloque PWM utilizado en este trabajo. Su incorporación dentro del entorno PIL resulta especialmente relevante, ya que permite representar no solo la señal de control final, sino también su asociación con la lógica temporal del sistema embebido y con el proceso de adquisición del ADC. En este sentido, la sincronización entre medida, cálculo y actuación resulta esencial en sistemas de electrónica de potencia, ya que el instante en que se actualiza la modulación condiciona directamente el comportamiento dinámico del convertidor. En la arquitectura de PLECS para C2000, la sincronización entre adquisición ADC, ejecución del control y actuación PWM constituye un aspecto clave del lazo de control digital.

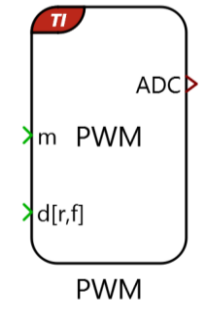

Figura 6-9. Bloque PWM

En cuanto a su configuración, el bloque se ha definido con el generador $PWM\_DSP\_PWM = [4]$, por lo que la modulación se implementa sobre el recurso ePWM4 del microcontrolador. La frecuencia de conmutación se ha fijado en $f_{sw} = 10$ kHz, mientras que el tipo de portadora seleccionado es Symmetrical (Carrier counts up/down), es decir, una portadora triangular simétrica con conteo ascendente y descendente. Además, el parámetro Phase(s) se ha establecido en 0, por lo que no se introduce desfase respecto a la referencia de sincronización. El ajuste Frequency tolerance = Enforce exact value impone que la frecuencia solicitada se respete exactamente, evitando aproximaciones derivadas de la resolución finita del reloj del sistema, y que el parámetro Frequency tolerance define el comportamiento cuando la frecuencia deseada no puede alcanzarse de forma exacta.

En la pestaña Output, el bloque se ha configurado en modo Complementary outputs, de forma que genera un par de señales complementarias asociadas a los canales A y B del periférico PWM. Este modo resulta adecuado para la etapa de potencia implementada, ya que permite accionar de forma coordinada los dos interruptores del convertidor. Asimismo, la opción Dead time variation se ha habilitado, por lo que el tiempo muerto no se fija internamente como un valor constante, sino que se introduce externamente a través de la entrada $d[r, f]$. De acuerdo con el manual, esta entrada permite especificar los retardos de flanco ascendente y descendente en segundos. En la configuración adoptada se ha fijado además un Minimum dead time de $1x10^{-6}$ s, lo que establece un valor mínimo de seguridad cuando se emplea tiempo muerto variable. La polaridad se ha definido como Active state is logic '1', la secuencia como Positive y el puerto de habilitación adicional permanece oculto (Enable port = Hide). El manual indica que el modo Complementary outputs hace operar los canales A y B de forma complementaria con tiempo muerto, y que la secuencia positiva produce un patrón PWM que comienza con el estado activo.

Las opciones de alta resolución se encuentran deshabilitadas, por lo que no se emplean ajustes finos adicionales sobre el ciclo de trabajo, el periodo o el tiempo muerto. Por su parte, en la pestaña Shadowing, la carga de los registros de comparación y de los retardos variables se ha configurado en Underflow and Overflow. Esta elección resulta coherente con la lógica temporal del bloque, ya que permite actualizar la modulación en los instantes extremos de la portadora y evitar inconsistencias durante la conmutación. De hecho, el manual señala que el paquete de soporte actualiza por defecto el registro del ciclo de trabajo del ePWM en eventos underflow y overflow para prevenir corrupción de datos.

En la pestaña Protection, se ha seleccionado la opción Allow control by protection block, lo que permite que un bloque externo de protección, si está presente en el esquema, pueda actuar sobre las salidas PWM. En la pestaña Sync, la sincronización se ha configurado como Self, con las salidas de sincronización deshabilitadas. Según el manual, cuando se selecciona la auto-sincronización, el primer recurso PWM asignado debe presentar fase nula, condición que en este caso se satisface al haberse fijado Phase(s) = 0.

Especialmente relevante en esta implementación es la configuración de la pestaña Events, donde se ha definido ADC trigger = Overflow y Task trigger = Disabled. Esto significa que el bloque PWM se utiliza para generar la señal que inicia la conversión del ADC en el instante overflow de la portadora. De este modo, el muestreo de las variables eléctricas queda sincronizado con la modulación del convertidor. El manual indica que los ADC triggers configuran el inicio de conversión del ADC y que dicho inicio puede ser generado por una portadora PWM; además, todos los canales asociados a la unidad ADC se convierten secuencialmente cuando se activa dicho trigger. Por tanto, en este modelo, el bloque PWM actúa como referencia temporal del proceso de adquisición descrito en el apartado anterior, estableciendo la relación entre conmutación, muestreo y ejecución del control digital.

Finalmente, en la pestaña Offline only se ha mantenido la opción PWM clock = Determine automatically, de manera que la frecuencia de reloj empleada en la simulación offline se determina automáticamente a partir de la configuración general del microcontrolador objetivo. En conjunto, la configuración adoptada permite reproducir en el entorno PIL el comportamiento del periférico ePWM utilizado en la implementación embebida, tanto en la generación de pulsos complementarios como en la temporización del disparo del ADC.

6.4. Coordinación temporal y secuencia de ejecución en el entorno PIL

Un aspecto clave en la validación Processor-in-the-Loop es la correcta coordinación temporal entre la simulación de la planta y la ejecución del código de control sobre el microcontrolador. Aunque la planta permanece modelada en PLECS, el algoritmo de control se ejecuta siguiendo una estructura temporal análoga a la de una implementación embebida real. Por ello, la validez del entorno PIL no depende únicamente del intercambio de señales entre ambos dominios, sino también de que dicho intercambio se produzca de forma sincronizada con el muestreo, la ejecución del control y la actualización de la señal de actuación.

En la configuración adoptada en este trabajo, la ejecución del control se organiza alrededor de una tarea base con periodo de muestreo $T_{s,controller} = 0{,}0001\ s$, tal como se indicó en el apartado 6.2.1. Esta tarea constituye la referencia temporal principal del sistema de control y determina la cadencia con la que se adquieren las medidas, se actualizan las variables internas del algoritmo y se calcula la nueva acción de control. De este modo, el entorno PIL conserva la estructura discreta del controlador diseñada previamente, pero trasladando su

ejecución al microcontrolador objetivo.

Desde el punto de vista funcional, la secuencia temporal del entorno PIL puede describirse del siguiente modo. En primer lugar, la planta simulada en PLECS genera las magnitudes equivalentes a las medidas del sistema. Estas señales se adaptan al rango de trabajo del ADC y se introducen en el microcontrolador mediante los bloques de intercambio correspondientes. A continuación, el periférico ADC realiza la adquisición de dichas variables, y una vez disponibles las conversiones, se habilita la ejecución de la tarea de control. En esta etapa, el microcontrolador procesa las medidas recibidas, ejecuta el algoritmo MPPT, actualiza los lazos PI y calcula la señal de actuación correspondiente.

Una vez calculada la acción de control, las variables de salida se recuperan desde el microcontrolador mediante los bloques Read Probe y se transfieren nuevamente a PLECS. Entre ellas destaca la señal $Duty$, que se aplica al bloque PWM para gobernar la etapa de potencia simulada. De este modo, el lazo de control queda cerrado manteniendo una secuencia coherente de adquisición, cálculo y actuación, equivalente a la que tendría lugar en una implementación física del sistema.

Esta organización da lugar a un comportamiento que puede considerarse de pseudo-tiempo real desde el punto de vista funcional. En efecto, el algoritmo de control se ejecuta con la misma lógica temporal que tendría en el microcontrolador real, respetando la secuencia de muestreo, cálculo y actualización de la modulación. Sin embargo, el avance del tiempo continúa gobernado por el simulador, de manera que la planta no necesita ejecutarse estrictamente en tiempo real. Aun así, la estructura temporal del control sí resulta representativa de la implementación embebida, lo que permite evaluar con mayor fidelidad la interacción entre el código de control y los periféricos asociados.

Además, conviene señalar que la actualización de la señal aplicada al PWM no se produce de manera instantánea tras el cálculo del control, sino de acuerdo con la lógica temporal propia del periférico. En consecuencia, entre la adquisición de las medidas y la aplicación efectiva de la nueva acción de control puede existir un retardo discreto asociado a la secuencia de ejecución del sistema embebido. Lejos de constituir una limitación, este comportamiento forma parte del realismo adicional que aporta la metodología PIL respecto a una simulación puramente numérica.

En conjunto, la coordinación temporal adoptada en el entorno PIL permite reproducir de forma coherente la interacción entre planta simulada, la adquisición de medidas, la ejecución del control y la generación de la señal de actuación. Esto resulta fundamental para que la validación no se limite a la comprobación funcional del algoritmo, sino que incluya también los efectos asociados a su integración temporal sobre el microcontrolador objetivo.

Sobre esta base temporal y funcional, en el apartado siguiente se analizan los resultados obtenidos con el entorno PIL, con objeto de comprobar que la ejecución del control sobre el microcontrolador mantiene el comportamiento esperado del sistema.

6.5. Resultados de la validación PIL

Una vez definido el entorno Processor-in-the-Loop e integrados los bloques necesarios para el intercambio de señales entre PLECS y el microcontrolador, se procede a la validación del sistema bajo esta nueva configuración. En esta etapa, el objetivo no es volver a analizar en detalle la dinámica del convertidor ya estudiada en los apartados anteriores, sino comprobar que la ejecución del código de control sobre el microcontrolador objetivo mantiene el comportamiento previsto del sistema dentro del esquema PIL.

Para ello, se representan algunas de las variables más significativas de la etapa fotovoltaica y del convertidor elevador, así como la señal de actuación generada por el microcontrolador. En particular, se analiza la evolución de la tensión del generador fotovoltaico y de su referencia, la tensión de salida del convertidor, la corriente de la bobina y la señal Duty calculada por el control. El análisis conjunto de estas magnitudes permite verificar tanto la correcta interacción entre PLECS y el microcontrolador como la validez funcional del control en condiciones representativas de una implementación embebida.

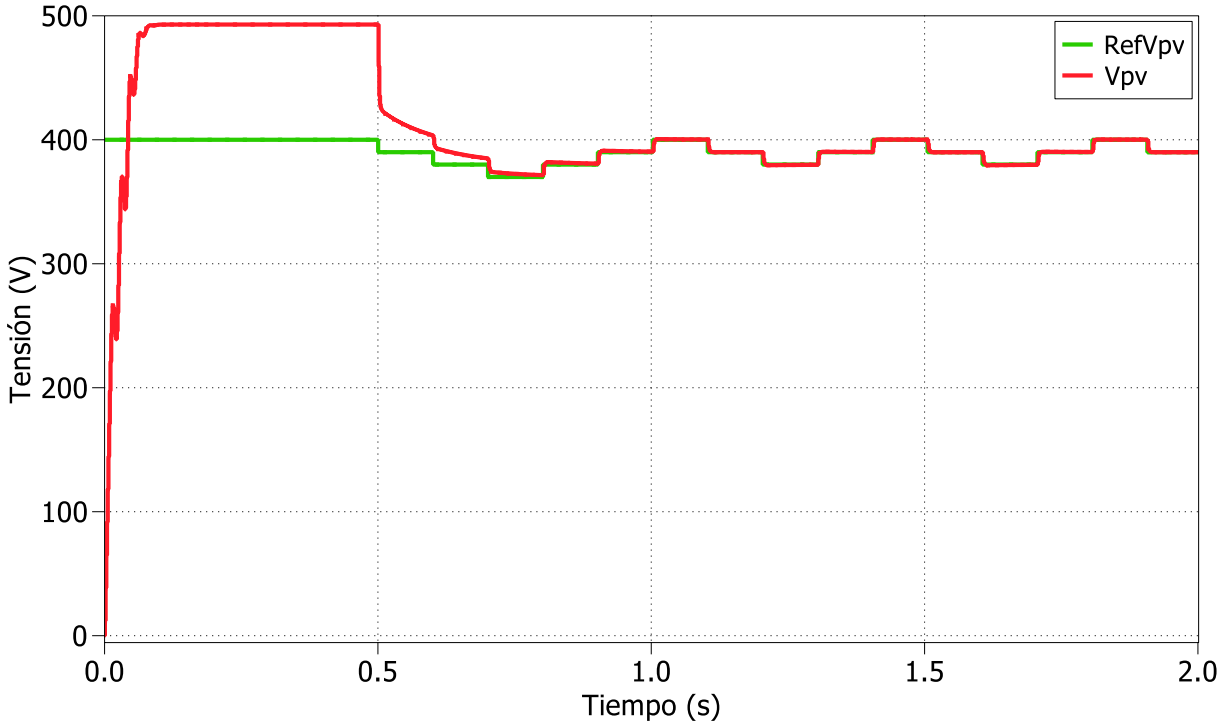

Figura 6-10. Evolución temporal de la tensión del generador FV y de su referencia en el entorno PIL

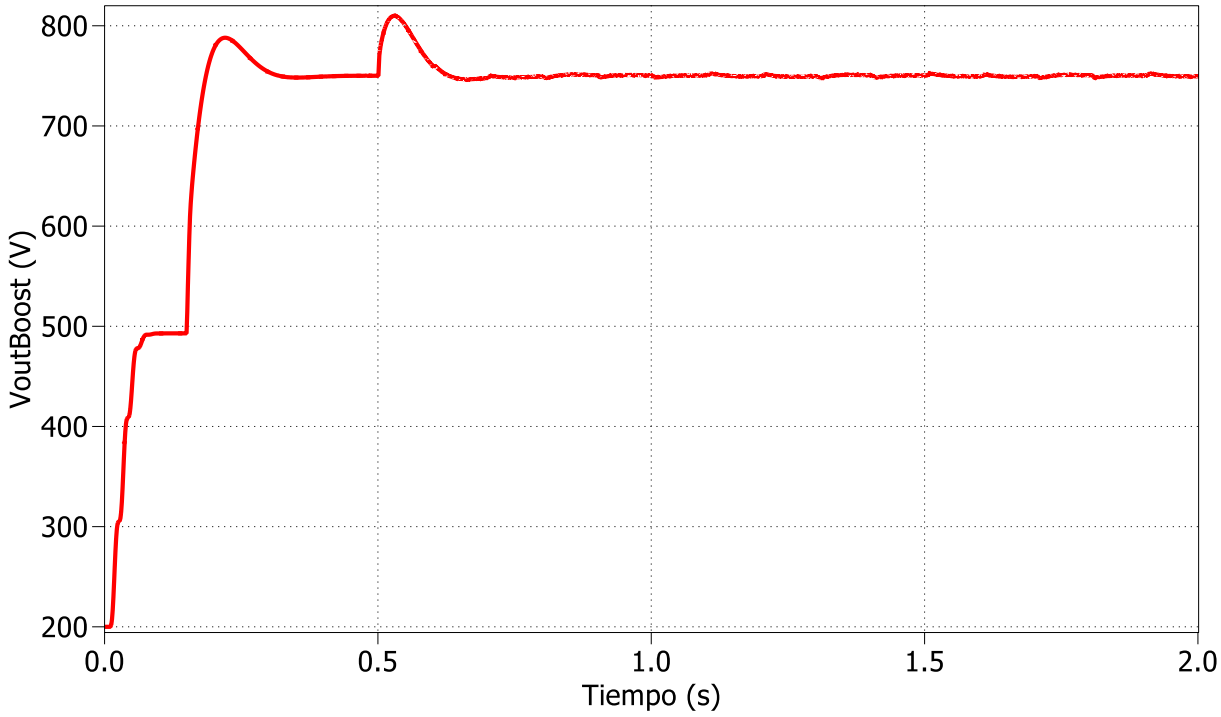

Figura 6-11. Evolución temporal de la tensión de salida del convertidor en el entorno PIL

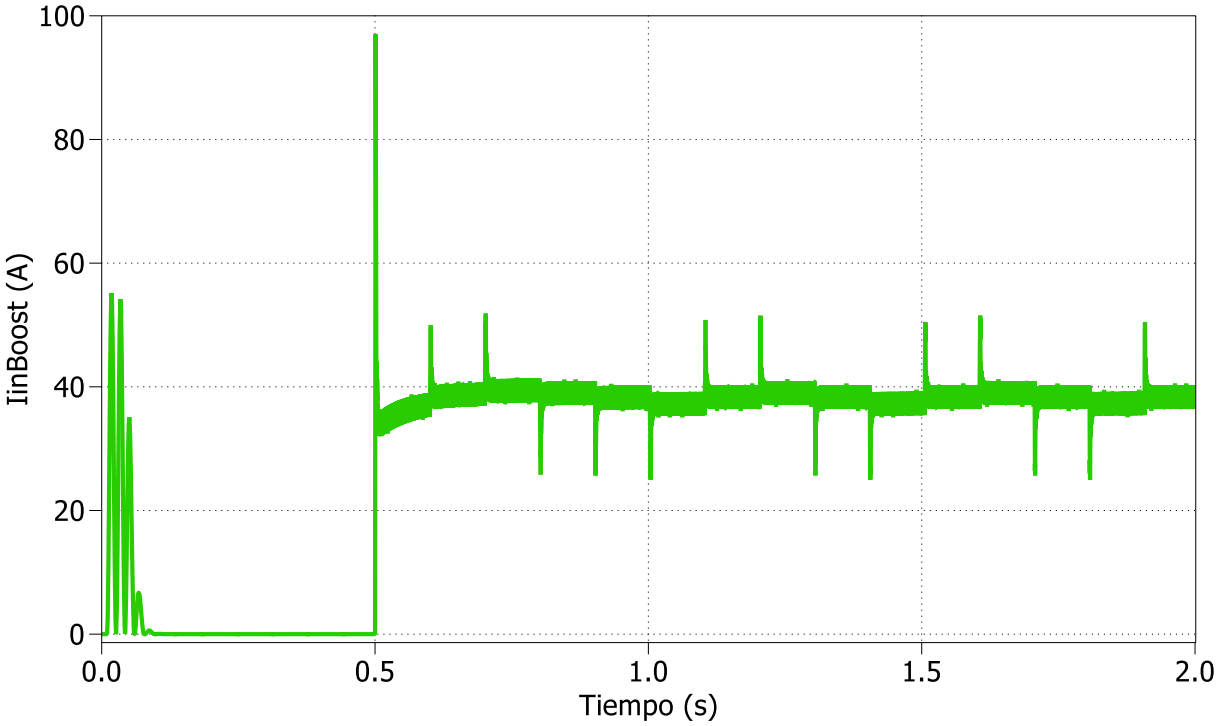

Figura 6-12. Evolución temporal de la corriente de la bobina del convertidor en el entorno PIL

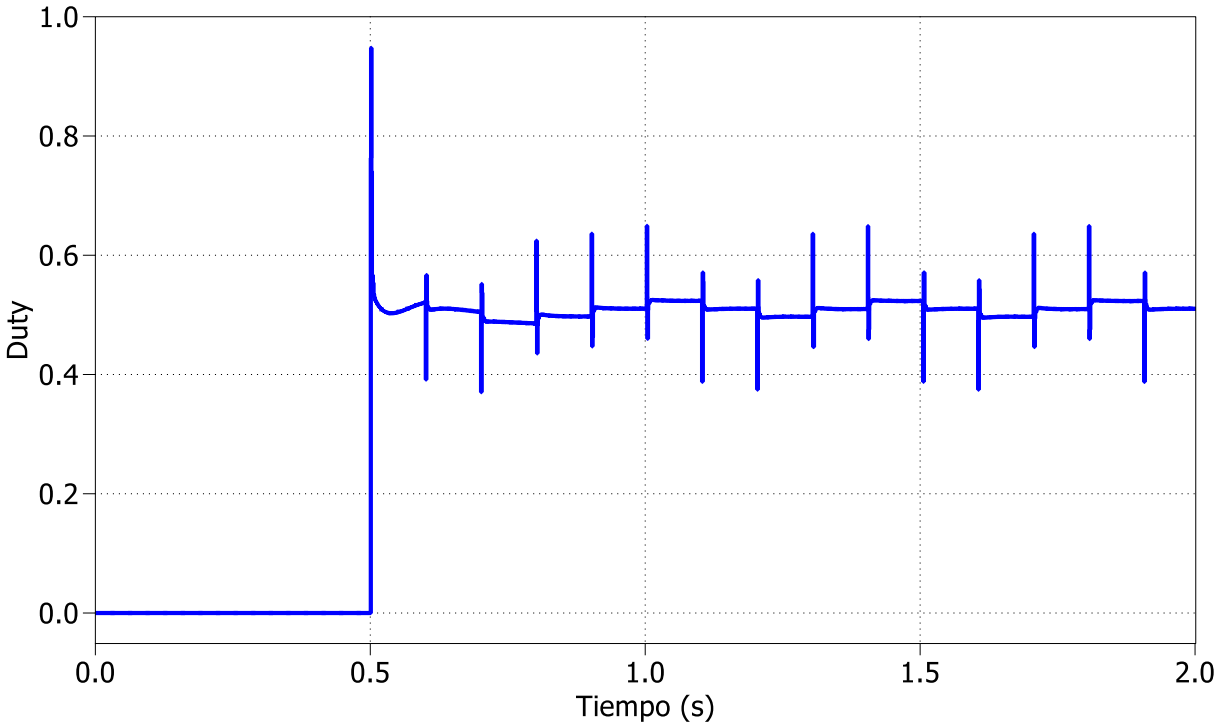

Figura 6-13. Evolución temporal de la señal de actuación generada por el microcontrolador en el entorno PIL

La comparación entre la tensión del generador fotovoltaico V_{pv} y la referencia $RefVpv$ muestra que, una vez habilitado el control, la tensión de la planta abandona la condicion próxima a circuito abierto y converge hacia la referencia calculada por el algoritmo MPPT. A partir de ese instante, $RefVpv$ presenta variaciones escalonadas asociadas al proceso de búsqueda del punto de máxima potencia, mientras que V_{pv} sigue dicha consigna con una dinámica coherente y sin errores apreciables de seguimiento en régimen permanente.

Por su parte, la evolución de la tensión de salida del convertidor confirma que la respuesta global del sistema se

mantiene dentro del comportamiento esperado, sin apreciarse desviaciones relevantes atribuibles al traslado del control al microcontrolador. Del mismo modo, la corriente de la bobina presenta una dinámica compatible con la operación de la etapa elevadora, lo que indica que la acción de control calculada se aplica correctamente sobre la planta simulada.

Finalmente, la señal $Duty$ permite comprobar de forma directa la actuación generada por el microcontrolador dentro del entorno PIL. Como puede observarse en la Figura 6-13, esta señal permanece nula hasta la habilitación del control y, a partir de ese instante, adopta valores coherentes con el funcionamiento del convertidor elevador, estabilizándose en torno a un valor medio próximo a 0,5 con variaciones asociadas a la dinámica del cotnrol y al ajuste del punto de operación.

Desde el punto de vista temporal, la respuesta observada resulta compatible con la secuencia de ejecución descrita en el apartado anterior, incluyendo la adquisición de medidas, el cálculo del control y la actualización de la señal de actuación. Por tanto, la validación PIL no solo confirma el correcto funcionamiento del algoritmo, sino también su integración efectiva con la plataforma embebida y con la lógica temporal asociada a los periféricos utilizados.

En conjunto, los resultados obtenidos permiten concluir que la metodología PIL reproduce adecuadamente el comportamiento funcional del sistema, manteniendo la coherencia entre la planta simulada y el código de control ejecutado sobre el microcontrolador. De este modo, la validación realizada confirma que el control implementado puede ejecutarse sobre la plataforma objetivo sin introducir desviaciones relevantes respecto al comportamiento obtenido previamente en simulación.

7 CONCLUSIONES Y TRABAJOS FUTUROS

El presente trabajo ha permitido desarrollar e implementar una estrategia de control para un convertidor DC/DC elevador aplicado a un sistema fotovoltaico, abarcando desde el modelado del generador hasta el diseño de los lazos de control y su validación en un entorno Processor-in-the-Loop (PIL) mediante un sistema embebido. De este modo, se ha seguido una metodología que integra el análisis en simulación con la verificación de la ejecución del control sobre procesador, constituyendo una etapa intermedia clave dentro del proceso de desarrollo.

Los resultados obtenidos ponen de manifiesto que la estructura de control planteada resulta adecuada para la aplicación considerada, permitiendo un funcionamiento coherente del convertidor dentro del sistema fotovoltaico. Asimismo, el uso del entorno PIL ha permitido incorporar al desarrollo aspectos propios de la implementación digital, como la discretización del control, el escalado de señales y la interacción con periféricos como el ADC y el PWM. En este sentido, el trabajo no solo valida el diseño del control, sino que también demuestra su correcta integración en una plataforma embebida.

Por otra parte, el desarrollo realizado abre la puerta a futuras ampliaciones dentro de esta misma línea de trabajo. En particular, resultaría de interés profundizar en el análisis del comportamiento del sistema bajo diferentes condiciones de operación, como variaciones de irradiancia y temperatura, así como estudiar con mayor detalle la influencia de parámetros asociados a la implementación digital, tales como el tiempo de muestreo, la cuantificación o las limitaciones de los periféricos. Asimismo, podría ampliarse el estudio mediante la comparación de distintas estrategias de seguimiento del punto de máxima potencia o la evaluación de técnicas de control más avanzadas que permitan mejorar la robustez y la respuesta dinámica del sistema.

En conjunto, el trabajo desarrollado confirma la validez de la metodología empleada para el diseño y validación de estrategias de control en sistemas fotovoltaicos, evidenciando el potencial del entorno PIL como herramienta eficaz para aproximar el diseño teórico a su implementación real en aplicaciones de electrónica de potencia.

ANEXO A: MPPT y lazos de control en PLECS

Code declarations

```
#include <math.h>

// -------- Parámetros --------
static double Kp_v, Ki_v;      // lazo externo (tensión PV -> Pref)
static double Kp_i, Ki_i;      // lazo interno (corriente -> RefMod)
static double Ts, Vpv_eps;

static double IncVref, Nciclos, VpvRef_init;

// -------- Entradas --------
#define Vpv       InputSignal(0,0)  // MedVpv
#define Ipv_mppt  InputSignal(0,1)  // MedIPV (para MPPT)
#define IL        InputSignal(0,2)  // MedIL (para lazo interno)

// -------- Estados discretos (k-1) --------
#define I_v_1        DiscState(0)   // integral PI externo
#define I_i_1        DiscState(1)   // integral PI interno
#define Vpv_ant_1    DiscState(2)
#define P_ant_1      DiscState(3)
#define VpvRef_ant_1 DiscState(4)
#define contador_1   DiscState(5)

// -------- Next states (commit en update) --------
static double I_v_n, I_i_n;
static double Vpv_ant_n, P_ant_n, VpvRef_ant_n, contador_n;
```

Start function code

```
Kp_v        = ParamRealData(0,0);
Ki_v        = ParamRealData(1,0);
Kp_i        = ParamRealData(2,0);
Ki_i        = ParamRealData(3,0);
Ts          = ParamRealData(4,0);
Vpv_eps     = ParamRealData(5,0);

IncVref     = ParamRealData(6,0);
Nciclos     = ParamRealData(7,0);
VpvRef_init = ParamRealData(8,0);

// Inicialización
I_v_1        = 0.0;
I_i_1        = 0.0;

Vpv_ant_1    = Vpv;
P_ant_1      = Vpv * Ipv_mppt;
VpvRef_ant_1 = VpvRef_init;

// Para ejecutar MPPT en la primera iteración
contador_1   = Nciclos;

// next = current
I_v_n        = I_v_1;
I_i_n        = I_i_1;
Vpv_ant_n    = Vpv_ant_1;
P_ant_n      = P_ant_1;
VpvRef_ant_n = VpvRef_ant_1;
contador_n   = contador_1;
```

Output function code

```
double v = Vpv;

// ======================================================
// 1) MPPT P&O: (Vpv, Ipv_mppt) -> RefVpv
// ======================================================
double VpvRef;

if (contador_1 >= Nciclos) {
  contador_n = 1.0;

  double P  = v * Ipv_mppt;
  double DP = P - P_ant_1;
  double DV = v - Vpv_ant_1;

  if (DP == 0.0) {
    VpvRef = VpvRef_ant_1;
  } else if (DP > 0.0) {
    if (DV > 0.0) VpvRef = VpvRef_ant_1 + IncVref;
    else          VpvRef = VpvRef_ant_1 - IncVref;
  } else { // DP < 0
    if (DV > 0.0) VpvRef = VpvRef_ant_1 - IncVref;
    else          VpvRef = VpvRef_ant_1 + IncVref;
  }

  // next-states MPPT
  Vpv_ant_n    = v;
  P_ant_n      = P;
  VpvRef_ant_n = VpvRef;

} else {
  VpvRef = VpvRef_ant_1;
  contador_n = contador_1 + 1.0;

  // mantener memorias
  Vpv_ant_n    = Vpv_ant_1;
  P_ant_n      = P_ant_1;
  VpvRef_ant_n = VpvRef_ant_1;
}

// ======================================================
// 2) Lazo externo: (VpvRef, Vpv) -> Pref
//    e_v = 0.5*(VpvRef^2 - Vpv^2)
//    Pref = -PI(e_v)
// ======================================================
double e_v = 0.5*(VpvRef*VpvRef - v*v);

double I_v = I_v_1 + Ts*e_v;          // Forward Euler
double u_v = Kp_v*e_v + Ki_v*I_v;
double Pref = -u_v;

I_v_n = I_v;

// ======================================================
// 3) Lazo interno: (Pref, Vpv, IL) -> RefMod
//    Iref = Pref / Vpv
//    e_i  = IL - Iref
//    RefMod = Vpv + PI(e_i)
// ======================================================
double Iref = 0.0;
if (fabs(v) >= Vpv_eps) {
  Iref = Pref / v;
} else {
  Iref = 0.0;
}

double e_i = IL - Iref;

double I_i = I_i_1 + Ts*e_i;          // Forward Euler
double u_i = Kp_i*e_i + Ki_i*I_i;

I_i_n = I_i;

// -------- Salidas --------
OutputSignal(0,0) = VpvRef;     // RefVpv
OutputSignal(0,1) = v + u_i;    // RefMod
```

Update function code

```
I_v_1        = I_v_n;
I_i_1        = I_i_n;

Vpv_ant_1    = Vpv_ant_n;
P_ant_1      = P_ant_n;
VpvRef_ant_1 = VpvRef_ant_n;
contador_1   = contador_n;
```

ANEXO B: Archivos CCS

En_Boost_Cont_main.c

```
/*
   Copyright (c) 2019-2021 by Plexim GmbH
   All rights reserved.

   A free license is granted to anyone to use this software for any legal
   non safety-critical purpose, including commercial applications, provided
   that:
   1) IT IS NOT USED TO DIRECTLY OR INDIRECTLY COMPETE WITH PLEXIM, and
   2) THIS COPYRIGHT NOTICE IS PRESERVED in its entirety.

   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
   OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
   FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE
   AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
   LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM,
   OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE
   SOFTWARE.
 */

// clang-format off: |>G_SUBS<|
#include "includes.h"

#include "plx_dispatcher.h"

#ifndef _FLASH

#endif

extern volatile unsigned int IFR;
extern volatile unsigned int IER;

// function prototypes
extern void MemCopy(Uint16 *SourceAddr, Uint16 *SourceEndAddr, Uint16
*DestAddr);

// linker addresses, needed to copy code from flash to ram
extern Uint16 RamfuncsLoadStart, RamfuncsLoadEnd, RamfuncsRunStart;

void En_Boost_Cont_initialize();

#define TSP_VER 0x0202
#ifdef TSP_VER
#if (TSP_VER != THIS_TSP_VER)
```

```
#error TSP Version mismatch.
#endif
#endif

void main(void)
{
#ifdef _FLASH
    MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
#endif

    // disable all interrupts
    DINT;
    IER = 0x0000;
    IFR = 0x0000;


    En_Boost_Cont_initialize();

    // enable interrupts
    EINT;   // global
    ERTM;   // real-time

    // go!
    DISPR_start(); // will not return

    PLX_ASSERT(0); // should never get here
}
// clang-format on
```

En_Boost_Cont_0.c

```
/*
 * C-Script file for: En-Boost-Cont/Control_logic/Codegen/C-Script
 * Generated with   : PLECS 5.0.1
 * Generated on     : 18 Mar 2026 16:06:12
 */
typedef float real_t;
#define REAL_MAX FLT_MAX
#define REAL_MIN FLT_MIN
#define REAL_EPSILON FLT_EPSILON
#include <math.h>

// -------- Parámetros --------
static double Kp_v, Ki_v;      // lazo externo (tensión PV -> Pref)
static double Kp_i, Ki_i;      // lazo interno (corriente -> RefMod)
static double Ts, Vpv_eps;

static double IncVref, Nciclos, VpvRef_init;

// deadbands MPPT
static double P_eps, V_eps;

// límites físicos
static double VpvRef_min, VpvRef_max;
static double Pmax, IL_max, RefMod_max;

// divisor de control interno
static double Nctrl;

// -------- Entradas --------
#define Vpv       InputSignal(0,0)   // MedVpv
#define Ipv_mppt  InputSignal(0,1)   // MedIPV (para MPPT)
```

```
#define IL        InputSignal(0,2)  // MedIL (para lazo interno)
#define Vdc       InputSignal(0,3)  // MedVdc (bus / VoutBoost)
#define Enable    InputSignal(0,4)  // (0 o 1)

// -------- Estados discretos (k-1) --------
#define I_v_1        DiscState(0)   // integral PI externo
#define I_i_1        DiscState(1)   // integral PI interno
#define Vpv_ant_1    DiscState(2)
#define P_ant_1      DiscState(3)
#define VpvRef_ant_1 DiscState(4)
#define contador_1   DiscState(5)

#define subcnt_1     DiscState(6)   // cuenta 0..Nctrl-1
#define duty_prev_1  DiscState(7)   // último duty aplicado (0..1)

// -------- Next states --------
static double I_v_n, I_i_n;
static double Vpv_ant_n, P_ant_n, VpvRef_ant_n, contador_n;
static double subcnt_n, duty_prev_n;

// -------- Helpers --------
static double sat01(double x)
{
   if (x < 0.0)
      return 0.0;
   if (x > 1.0)
      return 1.0;
   return x;
}

static double sat(double x, double lo, double hi)
{
   if (x < lo)
      return lo;
   if (x > hi)
      return hi;
   return x;
}
struct CScriptStruct
{
   int numInputTerminals;
   int numOutputTerminals;
   int* numInputSignals;
   int* numOutputSignals;
   int numContStates;
   int numDiscStates;
   int numZCSignals;
   int numSampleTimes;
   int numParameters;
   int isMajorTimeStep;
   int isRefineStep;
   float time;
   const float ***inputs;
   float ***outputs;
   float *contStates;
   float *contDerivs;
   float *discStates;
   float *zCSignals;
   const int *paramNumDims;
   const int **paramDims;
   const int *paramNumElements;
   const float **paramRealData;
```

```
    const char **paramStringData;
    const char * const *sampleHits;
    const float *sampleTimePeriods;
    const float *sampleTimeOffsets;
    const float * const *inputSampleTimePeriods;
    const float * const *inputSampleTimeOffsets;
    float *nextSampleHit;
    const char** errorStatus;
    const char** warningStatus;
    const char** rangeErrorMsg;
    int* rangeErrorLine;
    void (*writeCustomStateDouble)(void*, float);
    float (*readCustomStateDouble)(void*);
    void (*writeCustomStateInt)(void*, int);
    void (*writeCustomStateData)(void*, const void*, int);
    void (*readCustomStateData)(void*, void*, int);
};
#define NumInputs cScriptStruct->numInputSignals[0]
#define NumOutputs cScriptStruct->numOutputSignals[0]
#define Input(signalIdx) (*cScriptStruct->inputs[0][signalIdx])
#define Output(signalIdx) (*cScriptStruct->outputs[0][signalIdx])
#define NumInputTerminals cScriptStruct->numInputTerminals
#define NumOutputTerminals cScriptStruct->numOutputTerminals
#define NumInputSignals(terminalIdx) cScriptStruct->numInputSignals[ \
      terminalIdx]
#define NumOutputSignals(terminalIdx) cScriptStruct->numOutputSignals[ \
      terminalIdx]
#define InputSignal(terminalIdx, \
                    signalIdx) (*cScriptStruct-
>inputs[terminalIdx][signalIdx])
#define OutputSignal(terminalIdx, \
                     signalIdx) (*cScriptStruct->outputs[terminalIdx][ \
                                   signalIdx])
#define NumContStates cScriptStruct->numContStates
#define NumDiscStates cScriptStruct->numDiscStates
#define NumZCSignals cScriptStruct->numZCSignals
#define NumParameters cScriptStruct->numParameters
#define NumSampleTimes cScriptStruct->numSampleTimes
#define IsMajorStep cScriptStruct->isMajorTimeStep
#define IsRefineStep cScriptStruct->isRefineStep
#define CurrentTime cScriptStruct->time
#define NextSampleHit (*cScriptStruct->nextSampleHit)
#define SetErrorMessage(string) { *cScriptStruct->errorStatus=(string); }
#define SetWarningMessage(string)
#define ContState(idx) cScriptStruct->contStates[idx]
#define ContDeriv(idx) cScriptStruct->contDerivs[idx]
#define DiscState(idx) cScriptStruct->discStates[idx]
#define ZCSignal(idx) cScriptStruct->zCSignals[idx]
#define IsSampleHit(idx) (*cScriptStruct->sampleHits[idx])
#define SampleTimePeriod(idx) cScriptStruct->sampleTimePeriods[idx]
#define SampleTimeOffset(idx) cScriptStruct->sampleTimeOffsets[idx]
#define InputSampleTimePeriod(terminalIdx, \
                              signalIdx) cScriptStruct-
>inputSampleTimePeriods \
   [terminalIdx][signalIdx]
#define InputSampleTimeOffset(terminalIdx, \
                              signalIdx) cScriptStruct-
>inputSampleTimeOffsets \
   [terminalIdx][signalIdx]
#define ParamNumDims(idx) cScriptStruct->paramNumDims[idx]
#define ParamDim(pIdx, dIdx) cScriptStruct->paramDims[pIdx][dIdx]
#define ParamRealData(pIdx, dIdx) cScriptStruct->paramRealData[pIdx][dIdx]
```

```
#define ParamStringData(pIdx) cScriptStruct->paramStringData[pIdx]

void En_Boost_Cont_0_cScriptStart(const struct CScriptStruct *cScriptStruct)
{
   Kp_v        = ParamRealData(0,0);
   Ki_v        = ParamRealData(1,0);
   Kp_i        = ParamRealData(2,0);
   Ki_i        = ParamRealData(3,0);
   Ts          = ParamRealData(4,0);
   Vpv_eps     = ParamRealData(5,0);

   IncVref     = ParamRealData(6,0);
   Nciclos     = ParamRealData(7,0);
   VpvRef_init = ParamRealData(8,0);

   // deadbands MPPT
   P_eps       = ParamRealData(9,0);
   V_eps       = ParamRealData(10,0);

   // límites físicos
   VpvRef_min  = ParamRealData(11,0);
   VpvRef_max  = ParamRealData(12,0);
   Pmax        = ParamRealData(13,0);
   IL_max      = ParamRealData(14,0);
   RefMod_max  = ParamRealData(15,0);

   Nctrl       = ParamRealData(16,0);
   if (Nctrl < 1.0)
      Nctrl = 1.0;

   // Inicialización
   I_v_1       = 0.0;
   I_i_1       = 0.0;

   Vpv_ant_1   = Vpv;
   P_ant_1     = Vpv * Ipv_mppt;
   VpvRef_ant_1 = VpvRef_init;

   // Ejecuta MPPT en la primera iteración
   contador_1  = Nciclos;

   subcnt_1    = 0.0;
   duty_prev_1 = 0.0;

   // next = current
   I_v_n       = I_v_1;
   I_i_n       = I_i_1;
   Vpv_ant_n   = Vpv_ant_1;
   P_ant_n     = P_ant_1;
   VpvRef_ant_n = VpvRef_ant_1;
   contador_n  = contador_1;

   subcnt_n    = subcnt_1;
   duty_prev_n = duty_prev_1;
}

void En_Boost_Cont_0_cScriptOutput(const struct CScriptStruct *cScriptStruct)
{
    double v = Vpv;

    // ======================================================
    // ENABLE gating (apagado seguro + reset)
```

```
    // ======================================================
    if (Enable < 0.5) {

      // Salidas seguras (sin conmutación)
      OutputSignal(0,0) = VpvRef_init;
      OutputSignal(0,1) = 0.0;          // RefMod
      OutputSignal(0,2) = 0.0;          // Duty

      // Reset de estados para cuando vuelvas a habilitar
      I_v_n = 0.0;
      I_i_n = 0.0;

      Vpv_ant_n    = v;
      P_ant_n      = v * Ipv_mppt;
      VpvRef_ant_n = VpvRef_init;

      // Para que el MPPT se ejecute en el primer ciclo al habilitar
      contador_n   = Nciclos;

      // -------- Reset divisor interno y duty retenido --------
      subcnt_n     = 0.0;
      duty_prev_n  = 0.0;

      return;
    }

    // ======================================================
    // EJECUTAR CONTROL SOLO CADA Nctrl LLAMADAS (mantener duty entre medias)
    // ======================================================

    // Si aún no toca calcular, devolvemos duty anterior y congelamos
integradores/MPPT
    if (subcnt_1 < (Nctrl - 1.0)) {

      // avanza subcontador
      subcnt_n = subcnt_1 + 1.0;

      // mantiene duty
      duty_prev_n = duty_prev_1;
      OutputSignal(0,2) = duty_prev_1;

      // congela integradores
      I_v_n = I_v_1;
      I_i_n = I_i_1;

      // congela memorias MPPT
      Vpv_ant_n    = Vpv_ant_1;
      P_ant_n      = P_ant_1;
      VpvRef_ant_n = VpvRef_ant_1;

      contador_n   = contador_1 + 1.0;

      return;
    }

    // Si llegamos aquí: toca calcular control completo
    subcnt_n = 0.0;

    // Ts efectivo (porque actualizo cada Nctrl ciclos)
    double Ts_eff = Ts * Nctrl;

    // ======================================================
```

```
    // 1) MPPT P&O: (Vpv, Ipv_mppt) -> RefVpv
    // ======================================================
    double VpvRef;

    if (contador_1 >= Nciclos) {
      // resetea a 0 para periodo exacto Nciclos
      contador_n = 0.0;

      double P  = v * Ipv_mppt;
      double DP = P - P_ant_1;
      double DV = v - Vpv_ant_1;

      // deadband (evita nerviosismo con ruido)
      if (fabs(DP) < P_eps || fabs(DV) < V_eps) {
        VpvRef = VpvRef_ant_1;
      } else if (DP > 0.0) {
        VpvRef = (DV > 0.0) ? (VpvRef_ant_1 + IncVref) : (VpvRef_ant_1 -
IncVref);
      } else { // DP < 0
        VpvRef = (DV > 0.0) ? (VpvRef_ant_1 - IncVref) : (VpvRef_ant_1 +
IncVref);
      }

      // saturación física VpvRef
      VpvRef = sat(VpvRef, VpvRef_min, VpvRef_max);

      // next-states MPPT
      Vpv_ant_n    = v;
      P_ant_n      = P;
      VpvRef_ant_n = VpvRef;

    } else {
      VpvRef = VpvRef_ant_1;

      contador_n = contador_1 + 1.0;

      // mantener memorias
      Vpv_ant_n    = Vpv_ant_1;
      P_ant_n      = P_ant_1;
      VpvRef_ant_n = VpvRef_ant_1;
    }

    // ======================================================
    // 2) Lazo externo: (VpvRef, Vpv) -> Pref
    //    e_v = 0.5*(VpvRef^2 - Vpv^2)
    //    Pref = -PI(e_v)
    // ======================================================
    double e_v = 0.5*(VpvRef*VpvRef - v*v);

    double I_v = I_v_1 + Ts_eff*e_v;          // Forward Euler (Ts efectivo)
    double u_v = Kp_v*e_v + Ki_v*I_v;
    double Pref = -u_v;

    // Saturación de Pref
    Pref = sat(Pref, 0.0, Pmax);

    // Anti-windup básico externo
    int sat_hi_v = (Pref >= Pmax);
    int sat_lo_v = (Pref <= 0.0);
    int windup_v = ( (sat_hi_v && (e_v < 0.0)) || (sat_lo_v && (e_v > 0.0))
);
```

```
    if (windup_v) {
      I_v_n = I_v_1;   // congela integral
    } else {
      I_v_n = I_v;
    }

    // ======================================================
    // 3) Lazo interno: (Pref, Vpv, IL) -> RefMod
    //    Iref = Pref / Vpv
    //    e_i  = IL - Iref
    //    RefMod = Vpv + PI(e_i)
    // ======================================================
    double Iref = 0.0;
    if (fabs(v) >= Vpv_eps) {
      Iref = Pref / v;
    } else {
      Iref = 0.0;
    }

    // Clamp de Iref
    Iref = sat(Iref, 0.0, IL_max);

    double e_i = IL - Iref;

    // Integra provisional (luego aplicamos anti-windup por saturación de
duty)
    double I_i = I_i_1 + Ts_eff*e_i;          // Forward Euler (Ts efectivo)
    double u_i = Kp_i*e_i + Ki_i*I_i;

    // RefMod provisional
    double RefMod = v + u_i;

    // Clamp RefMod
    RefMod = sat(RefMod, 0.0, RefMod_max);

    // ======================================================
    // 4) Duty para PWM: duty = sat(RefMod / MedVdc, 0..1)
    // ======================================================
    double duty_unsat = 0.0;
    if (fabs(Vdc) >= 1e-9) {
      duty_unsat = RefMod / Vdc;
    } else {
      duty_unsat = 0.0;
    }
    double duty = sat01(duty_unsat);

    // Anti-windup interno (conditional integration)
    int sat_hi_i = (duty >= 1.0 - 1e-12);
    int sat_lo_i = (duty <= 0.0 + 1e-12);

    // Con e_i = IL - Iref, si e_i > 0 suele aumentar u_i -> aumenta duty.
    // Ajusta el signo si en tu planta ocurre lo contrario.
    int windup_i = ( (sat_hi_i && (e_i > 0.0)) || (sat_lo_i && (e_i < 0.0))
);

    if (windup_i) {
      I_i_n = I_i_1;   // congela integral
      // Recalcula u_i y RefMod con integral congelada (coherencia)
      u_i   = Kp_i*e_i + Ki_i*I_i_1;
      RefMod = sat(v + u_i, 0.0, RefMod_max);

      // Recalcula duty final
```

```
      if (fabs(Vdc) >= 1e-9) duty = sat01(RefMod / Vdc);
      else duty = 0.0;
    } else {
      I_i_n = I_i;
    }

    // -------- Guardar duty para los ciclos "skip" --------
    duty_prev_n = duty;

    // -------- Salidas --------
    OutputSignal(0,0) = VpvRef; // RefVpv
    OutputSignal(0,1) = RefMod; // RefMod
    OutputSignal(0,2) = duty;  // Duty
}

void En_Boost_Cont_0_cScriptUpdate(const struct CScriptStruct *cScriptStruct)
{
   I_v_1        = I_v_n;
   I_i_1        = I_i_n;

   Vpv_ant_1    = Vpv_ant_n;
   P_ant_1      = P_ant_n;
   VpvRef_ant_1 = VpvRef_ant_n;
   contador_1   = contador_n;

   subcnt_1    = subcnt_n;
   duty_prev_1 = duty_prev_n;
}

void En_Boost_Cont_0_cScriptDerivative(
                                       const struct CScriptStruct
*cScriptStruct)
{
}

void En_Boost_Cont_0_cScriptTerminate(
                                      const struct CScriptStruct
*cScriptStruct)
{
}
```

En_Boost_Cont.c

```
/*
 * Implementation file for: PVcompletoCCS2/Boost Converter/En-Boost-Cont
 * Generated with         : PLECS 5.0.1
 *                          TI2806x 2.2.2
 * Generated on           : 18 Mar 2026 16:06:12
 */
#include "En_Boost_Cont.h"
#ifndef PLECS_HEADER_En_Boost_Cont_h_
#error The wrong header file "En_Boost_Cont.h" was included. Please check
#error your include path to see whether this file name conflicts with the
#error name of another header file.
#endif /* PLECS_HEADER_En_Boost_Cont_h_ */
#if defined(__GNUC__) && (__GNUC__ > 4)
#   define _ALIGNMENT 16
#   define _RESTRICT __restrict
#   define _ALIGN __attribute__((aligned(_ALIGNMENT)))
#   if defined(__clang__)
#      if __has_builtin(__builtin_assume_aligned)
#         define _ASSUME_ALIGNED(a) __builtin_assume_aligned(a, _ALIGNMENT)
```

```
#      else
#         define _ASSUME_ALIGNED(a) a
#      endif
#   else
#      define _ASSUME_ALIGNED(a) __builtin_assume_aligned(a, _ALIGNMENT)
#   endif
#else
#   ifndef _RESTRICT
#      define _RESTRICT
#   endif
#   ifndef _ALIGN
#      define _ALIGN
#   endif
#   ifndef _ASSUME_ALIGNED
#      define _ASSUME_ALIGNED(a) a
#   endif
#endif
#include <stdint.h>
#include <stdbool.h>
#include <math.h>
#include <string.h>
#include "plx_hal.h"
#include <stdlib.h>
#define PLECSRunTimeError(msg) En_Boost_Cont_errorStatus = msg
struct CScriptStruct
{
   int numInputTerminals;
   int numOutputTerminals;
   int* numInputSignals;
   int* numOutputSignals;
   int numContStates;
   int numDiscStates;
   int numZCSignals;
   int numSampleTimes;
   int numParameters;
   int isMajorTimeStep;
   int isRefineStep;
   float time;
   const float ***inputs;
   float ***outputs;
   float *contStates;
   float *contDerivs;
   float *discStates;
   float *zCSignals;
   const int *paramNumDims;
   const int **paramDims;
   const int *paramNumElements;
   const float **paramRealData;
   const char **paramStringData;
   const char * const *sampleHits;
   const float *sampleTimePeriods;
   const float *sampleTimeOffsets;
   const float * const *inputSampleTimePeriods;
   const float * const *inputSampleTimeOffsets;
   float *nextSampleHit;
   const char** errorStatus;
   const char** warningStatus;
   const char** rangeErrorMsg;
   int* rangeErrorLine;
   void (*writeCustomStateDouble)(void*, float);
   float (*readCustomStateDouble)(void*);
   void (*writeCustomStateInt)(void*, int);
```

```
   void (*writeCustomStateData)(void*, const void*, int);
   void (*readCustomStateData)(void*, void*, int);
};
static struct CScriptStruct En_Boost_Cont_cScriptStruct[1];
static char En_Boost_Cont_isMajorStep = '\001';
static const uint32_t En_Boost_Cont_subTaskPeriod[1]= {
   /* [0.5, 0], Base task */
   5000
};
static uint32_t En_Boost_Cont_subTaskTick[1];
static char En_Boost_Cont_subTaskHit[1];
#define En_Boost_Cont_UNCONNECTED 0
static uint32_t En_Boost_Cont_D_uint32_t[1];
void En_Boost_Cont_0_cScriptStart(const struct CScriptStruct *cScriptStruct);
void En_Boost_Cont_0_cScriptOutput(const struct CScriptStruct
*cScriptStruct);
void En_Boost_Cont_0_cScriptUpdate(const struct CScriptStruct
*cScriptStruct);
void En_Boost_Cont_0_cScriptDerivative(
                                       const struct CScriptStruct
*cScriptStruct);
void En_Boost_Cont_0_cScriptTerminate(
                                     const struct CScriptStruct
*cScriptStruct);
static uint32_t En_Boost_Cont_tickLo[2];
static int32_t En_Boost_Cont_tickHi[2];
En_Boost_Cont_BlockOutputs En_Boost_Cont_B;
En_Boost_Cont_ModelStates En_Boost_Cont_X _ALIGN;
#if defined(EXTERNAL_MODE) && EXTERNAL_MODE
const float * const En_Boost_Cont_ExtModeSignals[] = {
   &En_Boost_Cont_B.ADCResults[2],
   &En_Boost_Cont_B.ADCResults[3],
   &En_Boost_Cont_B.C_Script[2],
   &En_Boost_Cont_B.C_Script[1],
   &En_Boost_Cont_B.C_Script[0],
   &En_Boost_Cont_B.ADCResults[0]
};
#endif /* defined(EXTERNAL_MODE) */
const char * En_Boost_Cont_errorStatus;
const float En_Boost_Cont_sampleTime[2][2] = {
   /* Task "Base task" */
   {0.0001f, 0.f},
   /* Task "Blink task" */
   {0.5f, 0.f}
};
const char * const En_Boost_Cont_checksum =
   "f6518eae4abefa50f877659798170c9bd55843d1";
/* Target declarations */
// tag step function to allow special linking
#pragma CODE_SECTION(En_Boost_Cont_step, "step")
extern void En_Boost_Cont_initHal();
/* Override Probe : 'En-Boost-Cont/ADCResults' */
En_Boost_ContProbes_t En_Boost_Cont_probes;

void En_Boost_Cont_initialize(void)
{
   En_Boost_Cont_tickHi[0] = 0;
   En_Boost_Cont_tickLo[0] = 0;
   En_Boost_Cont_tickHi[1] = 0;
   En_Boost_Cont_tickLo[1] = 0;
   /* Initialize sub-task tick counters */
   En_Boost_Cont_subTaskTick[0] = 0; /* Base task, [0.5, 0] */
```

```
   memset(&En_Boost_Cont_X, 0, sizeof(En_Boost_Cont_X));

   /* Target pre-initialization */
   En_Boost_Cont_initHal();


   /* Initialization for Pulse Generator : 'En-Boost-Cont/Enable' */
   En_Boost_Cont_D_uint32_t[0] = 19;

   /* Initialization for C-Script : 'En-Boost-Cont/Control_logic/Codegen/C-
Script' */
   {
      static int numInputSignals[] = {
         5
      };
      static const float* inputPtrs[] = {
         &En_Boost_Cont_B.ADCResults[0], &En_Boost_Cont_B.ADCResults[3],
         &En_Boost_Cont_B.ADCResults[2], &En_Boost_Cont_B.ADCResults[1],
         &En_Boost_Cont_B.En
      };
      static const float** inputs[] = {
         &inputPtrs[0]
      };
      static int numOutputSignals[] = {
         3
      };
      static float* outputPtrs[] = {
         &En_Boost_Cont_B.C_Script[0], &En_Boost_Cont_B.C_Script[1],
         &En_Boost_Cont_B.C_Script[2]
      };
      static float** outputs[] = {
         &outputPtrs[0]
      };
      static float nextSampleHit;
      static const char * sampleHits[] = {
         &En_Boost_Cont_isMajorStep
      };
      static float sampleTimePeriods[] = {
         0.0001
      };
      static float sampleTimeOffsets[] = {
         0
      };
      static float terminalSampleTimePeriods[] = {
         0.f, 0.f, 0.f, 0.f, 0.f
      };
      static float terminalSampleTimeOffsets[] = {
         -1.f, -1.f, -1.f, -1.f, -1.f
      };
      static const float* terminalSampleTimePeriodPtrs[] = {
         &terminalSampleTimePeriods[0]
      };
      static const float* terminalSampleTimeOffsetPtrs[] = {
         &terminalSampleTimeOffsets[0]
      };
      static const char* errorStatus;
      static const char* warningStatus;
      static const char* rangeErrorMsg;
      static const float paramData[] = {
         1.f,10.f,10.f,500.f,0.0001f,1e-06f,10.f,1000.f,400.f,0.f,0.f,330.f,
         430.f,1000000.f,1000000.f,1000000.f,5.f
      };
```

```
        static const float* paramDataPtr[] = {
           paramData+0,paramData+1,paramData+2,paramData+3,paramData+4,
           paramData+5,paramData+6,paramData+7,paramData+8,paramData+9,
           paramData+10,paramData+11,paramData+12,paramData+13,paramData+14,
           paramData+15,paramData+16
        };
        static const char paramStringData[] =
           "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0";
        static const char* paramStringDataPtr[] = {
           paramStringData+0,paramStringData+1,paramStringData+2,
           paramStringData+3,paramStringData+4,paramStringData+5,
           paramStringData+6,paramStringData+7,paramStringData+8,
           paramStringData+9,paramStringData+10,paramStringData+11,
           paramStringData+12,paramStringData+13,paramStringData+14,
           paramStringData+15,paramStringData+16
        };
        static const int paramDim[] = {
           1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
        };
        static const int* paramDimPtr[] = {
           paramDim+0,paramDim+2,paramDim+4,paramDim+6,paramDim+8,paramDim+10,

paramDim+12,paramDim+14,paramDim+16,paramDim+18,paramDim+20,paramDim+
           22,paramDim+24,paramDim+26,paramDim+28,paramDim+30,paramDim+32
        };
        static const int paramNumElements[] = {
           1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
        };
        static const int paramNumDims[] = {
           2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
        };
        errorStatus = NULL;
        warningStatus = NULL;
        rangeErrorMsg = NULL;
        En_Boost_Cont_cScriptStruct[0].isMajorTimeStep = 1;
        En_Boost_Cont_cScriptStruct[0].isRefineStep = 0;
        En_Boost_Cont_cScriptStruct[0].numInputTerminals = 1;
        En_Boost_Cont_cScriptStruct[0].numInputSignals = numInputSignals;
        En_Boost_Cont_cScriptStruct[0].inputs = inputs;
        En_Boost_Cont_cScriptStruct[0].numOutputTerminals = 1;
        En_Boost_Cont_cScriptStruct[0].numOutputSignals = numOutputSignals;
        En_Boost_Cont_cScriptStruct[0].outputs = outputs;
        En_Boost_Cont_cScriptStruct[0].numContStates = 0;
        En_Boost_Cont_cScriptStruct[0].contStates = NULL;
        En_Boost_Cont_cScriptStruct[0].contDerivs = NULL;
        En_Boost_Cont_cScriptStruct[0].numDiscStates = 8;
        En_Boost_Cont_cScriptStruct[0].discStates =
           &En_Boost_Cont_X.C_Script[0];
        En_Boost_Cont_cScriptStruct[0].numZCSignals = 0;
        En_Boost_Cont_cScriptStruct[0].numParameters = 17;
        En_Boost_Cont_cScriptStruct[0].paramNumDims = paramNumDims;
        En_Boost_Cont_cScriptStruct[0].paramDims = paramDimPtr;
        En_Boost_Cont_cScriptStruct[0].paramNumElements = paramNumElements;
        En_Boost_Cont_cScriptStruct[0].paramRealData = paramDataPtr;
        En_Boost_Cont_cScriptStruct[0].paramStringData = paramStringDataPtr;
        En_Boost_Cont_cScriptStruct[0].numSampleTimes = 1;
        En_Boost_Cont_cScriptStruct[0].sampleTimePeriods = sampleTimePeriods;
        En_Boost_Cont_cScriptStruct[0].sampleTimeOffsets = sampleTimeOffsets;
        En_Boost_Cont_cScriptStruct[0].inputSampleTimePeriods =
           terminalSampleTimePeriodPtrs;
        En_Boost_Cont_cScriptStruct[0].inputSampleTimeOffsets =
           terminalSampleTimeOffsetPtrs;
```

```
      En_Boost_Cont_cScriptStruct[0].sampleHits = sampleHits;
      En_Boost_Cont_cScriptStruct[0].nextSampleHit = &nextSampleHit;
      En_Boost_Cont_cScriptStruct[0].errorStatus = &errorStatus;
      En_Boost_Cont_cScriptStruct[0].warningStatus = &warningStatus;
      En_Boost_Cont_cScriptStruct[0].rangeErrorMsg = &rangeErrorMsg;
      En_Boost_Cont_0_cScriptStart(&En_Boost_Cont_cScriptStruct[0]);
      if (*En_Boost_Cont_cScriptStruct[0].errorStatus)
         En_Boost_Cont_errorStatus =
            *En_Boost_Cont_cScriptStruct[0].errorStatus;
   }

   /* Initialization for Delay : 'En-Boost-Cont/Delay2' */
   En_Boost_Cont_X.Delay2 = false;
}

void En_Boost_Cont_step(int task_id)
{
   if (En_Boost_Cont_errorStatus)
   {
      return;
   }
   switch(task_id)
   {
   case 0: /* Task "Base task" */
   {
      {
         size_t i;
         for (i = 0; i < 1; ++i)
         {
            En_Boost_Cont_subTaskHit[i] = (En_Boost_Cont_subTaskTick[i] ==
0);
         }
      }

      /* Override Probe : 'En-Boost-Cont/ADCResults' */
      SET_OPROBE(En_Boost_Cont_probes.ADCResults_1, (PLXHAL_ADC_getIn(0,
0)));
      SET_OPROBE(En_Boost_Cont_probes.ADCResults_2, (PLXHAL_ADC_getIn(0,
1)));
      SET_OPROBE(En_Boost_Cont_probes.ADCResults_3, (PLXHAL_ADC_getIn(0,
2)));
      SET_OPROBE(En_Boost_Cont_probes.ADCResults_4, (PLXHAL_ADC_getIn(0,
3)));
      En_Boost_Cont_B.ADCResults[0] = En_Boost_Cont_probes.ADCResults_1;
      En_Boost_Cont_B.ADCResults[1] = En_Boost_Cont_probes.ADCResults_2;
      En_Boost_Cont_B.ADCResults[2] = En_Boost_Cont_probes.ADCResults_3;
      En_Boost_Cont_B.ADCResults[3] = En_Boost_Cont_probes.ADCResults_4;
      if (En_Boost_Cont_subTaskHit[0])
      {
         /* Pulse Generator : 'En-Boost-Cont/Enable' */
         En_Boost_Cont_B.Enable = En_Boost_Cont_D_uint32_t[0] <
                                  10 ? 1.f : 0.f;
      }
      /* Override Probe : 'En-Boost-Cont/En' */
      SET_OPROBE(En_Boost_Cont_probes.En_1, En_Boost_Cont_B.Enable);
      En_Boost_Cont_B.En = En_Boost_Cont_probes.En_1;

      /* C-Script : 'En-Boost-Cont/Control_logic/Codegen/C-Script' */
      En_Boost_Cont_0_cScriptOutput(&En_Boost_Cont_cScriptStruct[0]);
      if (*En_Boost_Cont_cScriptStruct[0].errorStatus)
         En_Boost_Cont_errorStatus =
            *En_Boost_Cont_cScriptStruct[0].errorStatus;
```

```
      /* Read Probe : 'En-Boost-Cont/Duty' */
      En_Boost_Cont_probes.Duty_1 = En_Boost_Cont_B.C_Script[2];

      /* Read Probe : 'En-Boost-Cont/RefVpv' */
      En_Boost_Cont_probes.RefVpv_1 = En_Boost_Cont_B.C_Script[0];

      /* Read Probe : 'En-Boost-Cont/RefMod' */
      En_Boost_Cont_probes.RefMod_1 = En_Boost_Cont_B.C_Script[1];
      /* PWM : 'En-Boost-Cont/PWM' */

      PLXHAL_PWM_setDuty(0, En_Boost_Cont_B.C_Script[2]);
      if (En_Boost_Cont_errorStatus)
      {
         return;
      }
      if (En_Boost_Cont_subTaskHit[0])
      {
         /* Update for Pulse Generator : 'En-Boost-Cont/Enable' */
         En_Boost_Cont_D_uint32_t[0] += 1;
         if (En_Boost_Cont_D_uint32_t[0] > 19)
         {
            En_Boost_Cont_D_uint32_t[0] = 0;
         }
      }
      /* Update for C-Script : 'En-Boost-Cont/Control_logic/Codegen/C-Script'
*/
      En_Boost_Cont_0_cScriptUpdate(&En_Boost_Cont_cScriptStruct[0]);
      if (*En_Boost_Cont_cScriptStruct[0].errorStatus)
         En_Boost_Cont_errorStatus =
            *En_Boost_Cont_cScriptStruct[0].errorStatus;

      /* Update for PWM : 'En-Boost-Cont/PWM' */
      PLXHAL_PWM_enableAllOutputs();

      /* Increment sub-task tick counters */
      {
         size_t i;
         for (i = 0; i < 1; ++i)
         {
            En_Boost_Cont_subTaskTick[i]++;
            if (En_Boost_Cont_subTaskTick[i] >=
                En_Boost_Cont_subTaskPeriod[i])
            {
               En_Boost_Cont_subTaskTick[i] = 0;
            }
         }
      }
      break;
   }

   case 1:
/* Task "Blink task" */
   {

      /* Delay : 'En-Boost-Cont/Delay2' */
      En_Boost_Cont_B.Delay2 = En_Boost_Cont_X.Delay2;

      /* Logical Operator : 'En-Boost-Cont/Logical\nOperator2' */
      En_Boost_Cont_B.LogicalOperator2 = !En_Boost_Cont_B.Delay2;
      /* Digital Out : 'En-Boost-Cont/LED' */
      PLXHAL_DIO_set(0, En_Boost_Cont_B.LogicalOperator2);
      if (En_Boost_Cont_errorStatus)
```

```
      {
         return;
      }

      /* Update for Delay : 'En-Boost-Cont/Delay2' */
      En_Boost_Cont_X.Delay2 = En_Boost_Cont_B.LogicalOperator2;
      break;
   }
   }
}

void En_Boost_Cont_terminate(void)
{
   /* Target termination */
#pragma diag_default 552

   /* Termination for C-Script : 'En-Boost-Cont/Control_logic/Codegen/C-
Script' */
   En_Boost_Cont_0_cScriptTerminate(&En_Boost_Cont_cScriptStruct[0]);
   if (*En_Boost_Cont_cScriptStruct[0].errorStatus)
      En_Boost_Cont_errorStatus =
*En_Boost_Cont_cScriptStruct[0].errorStatus;

}
```

En_Boost_Cont_hal.c

```
/*
 * Hardware configuration file for: TI2806x
 * Generated with                 : PLECS 5.0.1
 * Generated on                   : Wed Mar 18 16:06:12 2026
 */

/* HAL Includes */
#include "En_Boost_Cont.h"
#include "plx_hal.h"
#include "plx_dispatcher.h"
#include "pil.h"
#include "plx_serial.h"
#include "string.h"
#include "plx_sci.h"
#include "plx_ain.h"
#include "plx_pwm.h"
#include "math.h"
#include "plx_dio.h"


/* HAL Declarations */
void DeviceInit(Uint16 clock_source, Uint16 pllDiv);
void InitFlash();
void DSP28x_usDelay(long LoopCount);

// Clock configuration
#define SYSCLK_HZ 90000000L
#define LSPCLK_HZ (SYSCLK_HZ / 4l)
#define PLL_DIV 18
#define PLL_DIVSEL 2
#define PLL_SRC 0
#define PLX_DELAY_US(A)  DSP28x_usDelay( \
                                      ((((long double) A * \
                                         1000.0L) / \
                                        11.111111L) - 9.0L) / 5.0L)
```

```
PIL_Obj_t PilObj;
PIL_Handle_t PilHandle = 0;
// external mode helper symbols
PIL_CONFIG_DEF(uint32_t, ExtMode_targetFloat_Size,
               sizeof(En_Boost_Cont_FloatType));
PIL_CONFIG_DEF(uint32_t, ExtMode_targetPointer_Size,
               sizeof(En_Boost_Cont_FloatType*));
PIL_CONFIG_DEF(uint32_t, ExtMode_sampleTime_Ptr,
               (uint32_t)&En_Boost_Cont_sampleTime);
PIL_CONFIG_DEF(uint32_t, ExtMode_checksum_Ptr,
               (uint32_t)&En_Boost_Cont_checksum);
      #if defined(En_Boost_Cont_NumTunableParameters) && \
   (En_Boost_Cont_NumTunableParameters > 0)
PIL_CONFIG_DEF(uint32_t, ExtMode_P_Ptr, (uint32_t)&En_Boost_Cont_P);
PIL_CONFIG_DEF(uint32_t, ExtMode_P_Size,
               (uint32_t)En_Boost_Cont_NumTunableParameters);
      #endif
      #if defined(En_Boost_Cont_NumExtModeSignals) && \
   (En_Boost_Cont_NumExtModeSignals > 0)
PIL_CONFIG_DEF(uint32_t, ExtMode_ExtModeSignals_Ptr,
               (uint32_t)&En_Boost_Cont_ExtModeSignals[0]);
PIL_CONFIG_DEF(uint32_t, ExtMode_ExtModeSignals_Size,
               (uint32_t)En_Boost_Cont_NumExtModeSignals);
      #endif
      #define CODE_GUID {0x09, 0x3c, 0x97, 0x5e, 0xdb, 0x2d, 0xb8, 0x40};

PIL_CONST_DEF(unsigned char, Guid[], CODE_GUID);
PIL_CONST_DEF(unsigned char, CompiledDate[], "03/18/2026 04:06 PM");
PIL_CONST_DEF(unsigned char, CompiledBy[], "PLECS Coder");
PIL_CONST_DEF(uint16_t, FrameworkVersion, PIL_FRAMEWORK_VERSION);
PIL_CONST_DEF(char, FirmwareDescription[], "TIC2000 Project (CPU0)");
PIL_CONST_DEF(uint16_t, StationAddress, 0);
PIL_CONST_DEF(uint32_t, BaudRate, 115200);
static void SerialPoll(PIL_Handle_t aHandle)
{
   PLXHAL_Serial_handleBreak(0);

   while(PLXHAL_Serial_rxIsReady(0))
   {
      // assuming that there will be a "break" when FIFO is empty
      PIL_SERIAL_IN(aHandle, (int16_t) PLXHAL_Serial_getChar(0));
   }
   int16_t ch;
   if(!PLXHAL_Serial_txIsBusy(0))
   {
      if(PIL_SERIAL_OUT(aHandle, &ch))
      {
         PLXHAL_Serial_putChar(0, ch);
      }
   }
}
#pragma DATA_SECTION(ScopeBuffer, "scope")
uint16_t ScopeBuffer[1024] /*__attribute__((aligned(16)))*/;
extern void PIL_setAndConfigScopeBuffer(PIL_Handle_t aPilHandle,
                                        uint16_t* aBufPtr, uint16_t aBufSize,
                                        uint16_t aMaxTraceWidthInWords);
extern const char * const En_Boost_Cont_checksum;
PLX_Serial_Obj_t SerialObjs[1];
PLX_Serial_Handle_t SerialHandles[1];
uint16_t PLXHAL_Serial_getChar(int16_t aChannel)
{
   return PLX_Serial_getChar(SerialHandles[aChannel]);
```

```
}

void PLXHAL_Serial_putChar(int16_t aChannel, uint16_t data)
{
   PLX_Serial_putChar(SerialHandles[aChannel], data);
}

void PLXHAL_Serial_putStr(int16_t aChannel, uint16_t *str)
{
   for (int i = 0; i < strlen((char *)str); i++)
   {
      PLXHAL_Serial_putChar(aChannel, str[i]);
   }
}

bool PLXHAL_Serial_rxIsReady(int16_t aChannel)
{
   return PLX_Serial_rxReady(SerialHandles[aChannel]);
}

bool PLXHAL_Serial_txIsBusy(int16_t aChannel)
{
   return PLX_Serial_txIsBusy(SerialHandles[aChannel]);
}

void PLXHAL_Serial_handleBreak(int16_t aChannel)
{
   if(PLX_Serial_breakOccurred(SerialHandles[aChannel]))
   {
      PLX_Serial_reset(SerialHandles[aChannel]);
   }
}
PLX_SCI_Obj_t SciObjs[1];
PLX_SCI_Handle_t SciHandles[1];
PLX_AIN_Handle_t AdcHandles[1];
PLX_AIN_Obj_t AdcObj[1];

float PLXHAL_ADC_getIn(uint16_t aHandle, uint16_t aChannel)
{
   return PLX_AIN_getInF(AdcHandles[aHandle], aChannel);
}
extern bool EpwmForceDisable;
PIL_SYMBOL_DEF(En_Boost_Cont_probes_Duty_1, 0, 1.0, "");
PIL_SYMBOL_DEF(En_Boost_Cont_probes_RefMod_1, 0, 1.0, "");
PIL_SYMBOL_DEF(En_Boost_Cont_probes_RefVpv_1, 0, 1.0, "");
PIL_SYMBOL_DEF(En_Boost_Cont_probes_ADCResults_1, 0, 1.0, "");
PIL_SYMBOL_DEF(En_Boost_Cont_probes_ADCResults_2, 0, 1.0, "");
PIL_SYMBOL_DEF(En_Boost_Cont_probes_ADCResults_3, 0, 1.0, "");
PIL_SYMBOL_DEF(En_Boost_Cont_probes_ADCResults_4, 0, 1.0, "");
PIL_SYMBOL_DEF(En_Boost_Cont_probes_En_1, 0, 1.0, "");

extern En_Boost_ContProbes_t En_Boost_Cont_probes;

void PilCallback(PIL_Handle_t aPilHandle, PIL_CtrlCallbackReq_t aCallbackReq)
{
   switch(aCallbackReq)
   {
   case  PIL_CLBK_ENTER_NORMAL_OPERATION_REQ:
      // allow power
      PIL_inhibitPilSimulation(aPilHandle);
      EpwmForceDisable = false;
      return;
```

```
    case PIL_CLBK_LEAVE_NORMAL_OPERATION_REQ:
        // disable power
        EpwmForceDisable = true;
        PIL_allowPilSimulation(aPilHandle);
        return;

    case PIL_CLBK_PREINIT_SIMULATION:
        return;

    case PIL_CLBK_INITIALIZE_SIMULATION:
        En_Boost_Cont_probes.Duty_1 = 0;
        En_Boost_Cont_probes.RefMod_1 = 0;
        En_Boost_Cont_probes.RefVpv_1 = 0;
        En_Boost_Cont_initialize();
        return;

    case PIL_CLBK_TERMINATE_SIMULATION:
        return;

    case PIL_CLBK_STOP_TIMERS:
        // stopping relevant timers
        CpuTimer0Regs.TCR.bit.TSS = 1;
        CpuTimer1Regs.TCR.bit.TSS = 1;
        PLX_PWM_disableAllClocks();

        return;

    case PIL_CLBK_START_TIMERS:
        // starting relevant timers
        CpuTimer0Regs.TCR.bit.TSS = 0;
        CpuTimer1Regs.TCR.bit.TSS = 0;
        PLX_PWM_enableAllClocks();

        return;
    }
}
extern PLX_PWM_Handle_t EpwmHandles[];
// PWM Enable Functions
void PLXHAL_PWM_setToPassive(uint16_t aChannel)
{
    PLX_PWM_setOutToPassive(EpwmHandles[aChannel]);
}

void PLXHAL_PWM_setToOperational(uint16_t aChannel)
{
    PLX_PWM_setOutToOperational(EpwmHandles[aChannel]);
}

void PLXHAL_PWM_prepareSetOutToXTransition(uint16_t aHandle)
{
    PLX_PWM_prepareSetOutToXTransition(EpwmHandles[aHandle]);
}

// PWM Set Duty Cycle Function
        #pragma CODE_SECTION(PLXHAL_PWM_setDuty, "ramfuncs")
void PLXHAL_PWM_setDuty(uint16_t aHandle, float aDuty)
{
    PLX_PWM_setPwmDuty(EpwmHandles[aHandle], aDuty);
}

// PWM Set Dead Time Functions
```

```
        #pragma CODE_SECTION(PLXHAL_PWM_setRisingDelay, "ramfuncs")
void PLXHAL_PWM_setRisingDelay(uint16_t aChannel, float delay)
{
   PLX_PWM_setDeadTimeCountsRising(EpwmHandles[aChannel],
                                   delay * 9.000000e+07f);
}

        #pragma CODE_SECTION(PLXHAL_PWM_setFallingDelay, "ramfuncs")
void PLXHAL_PWM_setFallingDelay(uint16_t aChannel, float delay)
{
   PLX_PWM_setDeadTimeCountsFalling(EpwmHandles[aChannel],
                                    delay * 9.000000e+07f);
}

// PWM Set Sequence Functions
void PLXHAL_PWM_setSequence(uint16_t aChannel, uint16_t aSequence)
{
   PLX_PWM_setSequence(EpwmHandles[aChannel], aSequence);
}

void PLXHAL_PWM_setSequenceAq(uint16_t aChannel, uint32_t aSequenceAq)
{
   PLX_PWM_setSequenceAq(EpwmHandles[aChannel], aSequenceAq);
}

void PLXHAL_PWM_enableOutputSwap(uint16_t aChannel, bool aEnableSwap)
{
   PLX_PWM_enableOutputSwap(EpwmHandles[aChannel], aEnableSwap);
}
    #pragma CODE_SECTION(PLXHAL_PWM_setDutyAB, "ramfuncs")
void PLXHAL_PWM_setDutyAB(uint16_t aHandle, float aDutyA, float aDutyB)
{
   PLX_PWM_setPwmDutyA(EpwmHandles[aHandle], aDutyA);
   PLX_PWM_setPwmDutyB(EpwmHandles[aHandle], aDutyB);
}

void PLXHAL_PWM_setOverflowSocDelay(uint16_t aHandle, float aPhase)
{
   PLX_PWM_setPwmDutyCGreaterThanZero(EpwmHandles[aHandle], 1.0f - aPhase);
}

void PLXHAL_PWM_setUnderflowSocDelay(uint16_t aHandle, float aPhase)
{
   PLX_PWM_setPwmDutyCSmallerThanFull(EpwmHandles[aHandle], aPhase);
}
PLX_PWM_Handle_t EpwmHandles[1];
PLX_PWM_Obj_t EpwmObj[1];
bool EpwmForceDisable = false;
void PLXHAL_PWM_enableAllOutputs()
{
   if(!EpwmForceDisable)
   {
      PLX_PWM_enableOut(EpwmHandles[0]);
   }
}
PLX_DIO_Handle_t DoutHandles[1];
PLX_DIO_Obj_t DoutObj[1];

void PLXHAL_DIO_set(uint16_t aHandle, bool aVal)
{
   PLX_DIO_set(DoutHandles[aHandle], aVal);
}
```

```
void PIL_setErrorMessage(PIL_Handle_t aPilHandle, const char* aMessage);
extern const char * En_Boost_Cont_errorStatus;
extern PIL_Handle_t PilHandle;
DISPR_TaskObj_t TaskObj[2];
extern void En_Boost_Cont_step(int task_id);
extern void En_Boost_Cont_enableTasksInterrupt();
extern void En_Boost_Cont_syncTimers();
static void Tasks(bool aInit, void * const aParam)
{
   if(En_Boost_Cont_errorStatus)
   {
      PIL_setErrorMessage(PilHandle, En_Boost_Cont_errorStatus);
      return;
   }
   if(aInit)
   {
      En_Boost_Cont_enableTasksInterrupt();
   }
   else
   {
      En_Boost_Cont_step(*(int *)aParam);
   }
}
interrupt void En_Boost_Cont_baseTaskInterrupt(void)
{
   AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;                  // clear ADCINT1
flag reinitialize for next SOC
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;  // acknowledge interrupt to PIE
(All ADCS in group 1)
   IER |= M_INT1;

   DISPR_dispatch();
}


/* Interrupt Enable Code */
void En_Boost_Cont_enableTasksInterrupt(void)
{
   IER |= M_INT1;

}

/* Timer Synchronization Code */
void En_Boost_Cont_syncTimers(void)
{
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;        // start all the timers
synced

}

/* Background task */
void En_Boost_Cont_background(void)
{

}

/* HAL Initialization Code */
static bool HalInitialized = false;
void En_Boost_Cont_initHal()
{

   if(HalInitialized == true)
```

```
    {
        return;
    }
    HalInitialized = true;

    // Pre init code
    DeviceInit(PLL_SRC, PLL_DIV);
    InitFlash();
    // set cpu timers to same clock as ePWM
    CpuTimer0Regs.TPR.all = 0;
    CpuTimer1Regs.TPR.all = 0;
    CpuTimer2Regs.TPR.all = 0;
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;  // stop all the TB clocks
    EDIS;
    PilHandle = PIL_init(&PilObj, sizeof(PilObj));
    PIL_setGuid(PilHandle, PIL_GUID_PTR);
    PIL_setChecksum(PilHandle, En_Boost_Cont_checksum);
    PIL_setAndConfigScopeBuffer(PilHandle, (uint16_t *)&ScopeBuffer, 1024,
12);
    PIL_setNodeAddress(PilHandle, PIL_D_StationAddress);
    PIL_setSerialComCallback(PilHandle, (PIL_CommCallbackPtr_t)SerialPoll);
    {
        for(int i = 0; i < 1; i++)
        {
            SerialHandles[i] =
                PLX_Serial_init(&SerialObjs[i], sizeof(SerialObjs[i]));
        }
    }
    {
        for(int i = 0; i < 1; i++)
        {
            SciHandles[i] =
                PLX_SCI_init(&SciObjs[i], sizeof(SciObjs[i]));
        }
    }
    {
        PLX_AIN_sinit(90000000);

        for (int i = 0; i < 1; i++)
        {
            AdcHandles[i] = PLX_AIN_init(&AdcObj[i], sizeof(AdcObj[i]));
        }
    }
    // configure ADC A
    {
        PLX_AIN_AdcParams_t params;
        PLX_AIN_setDefaultAdcParams(&params, 3.300000, 0);
        params.ADCCTL1.bit.ADCREFSEL = 0;
        PLX_AIN_configure(AdcHandles[0], (PLX_AIN_Unit_t)0, &params);
    }
    // configure SOC0 of ADC-A to measure ADCIN3
    {
        PLX_AIN_ChannelParams_t params;
        PLX_AIN_setDefaultChannelParams(&params);
        params.scale =  2.439393939e+02f;
        params.offset = 0.000000000e+00f;
        // set SOC trigger to PWM4
        params.ADCSOCxCTL.bit.TRIGSEL = 11;
        params.ADCSOCxCTL.bit.ACQPS = 6;
        PLX_AIN_setupChannel(AdcHandles[0], 0, 3, &params);
    }
```

```
// configure SOC1 of ADC-A to measure ADCIN11
{
   PLX_AIN_ChannelParams_t params;
   PLX_AIN_setDefaultChannelParams(&params);
   params.scale =  2.439393939e+02f;
   params.offset = 0.000000000e+00f;
   // set SOC trigger to PWM4
   params.ADCSOCxCTL.bit.TRIGSEL = 11;
   params.ADCSOCxCTL.bit.ACQPS = 6;
   PLX_AIN_setupChannel(AdcHandles[0], 1, 11, &params);
}
// configure SOC2 of ADC-A to measure ADCIN13
{
   PLX_AIN_ChannelParams_t params;
   PLX_AIN_setDefaultChannelParams(&params);
   params.scale =  1.969696970e+01f;
   params.offset = -3.250000000e+01f;
   // set SOC trigger to PWM4
   params.ADCSOCxCTL.bit.TRIGSEL = 11;
   params.ADCSOCxCTL.bit.ACQPS = 6;
   PLX_AIN_setupChannel(AdcHandles[0], 2, 13, &params);
}
// configure SOC3 of ADC-A to measure ADCIN10
{
   PLX_AIN_ChannelParams_t params;
   PLX_AIN_setDefaultChannelParams(&params);
   params.scale =  1.969696970e+01f;
   params.offset = -3.250000000e+01f;
   // set SOC trigger to PWM4
   params.ADCSOCxCTL.bit.TRIGSEL = 11;
   params.ADCSOCxCTL.bit.ACQPS = 6;
   PLX_AIN_setupChannel(AdcHandles[0], 3, 10, &params);
}
{
   INIT_OPROBE(En_Boost_Cont_probes.ADCResults_1);
   INIT_OPROBE(En_Boost_Cont_probes.ADCResults_2);
   INIT_OPROBE(En_Boost_Cont_probes.ADCResults_3);
   INIT_OPROBE(En_Boost_Cont_probes.ADCResults_4);
   INIT_OPROBE(En_Boost_Cont_probes.En_1);
}
PIL_setCtrlCallback(PilHandle, (PIL_CtrlCallbackPtr_t)PilCallback);
PIL_requestNormalMode(PilHandle);
PLX_PWM_sinit();
for (int i = 0; i < 1; i++)
{
   EpwmHandles[i] = PLX_PWM_init(&EpwmObj[i], sizeof(EpwmObj[i]));
}
// configure PWM4 at 10000.0 Hz in triangle mode
// soc='p'
{
   PLX_PWM_Params_t params;
   PLX_PWM_setDefaultParams(&params);
   params.outMode = PLX_PWM_OUTPUT_MODE_DUAL;
   params.reg.TBPRD = 4500;
   params.reg.TBCTL.bit.CTRMODE = 2;
   params.reg.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO_PRD;
   // active state is high
   params.reg.DBCTL.bit.POLSEL = 2; // DB_ACTV_HIC
   // enable deadtime insertion
   params.reg.DBCTL.bit.OUT_MODE = 3; // DB_FULL_ENABLE
   params.reg.DBCTL.bit.IN_MODE = 0; // DBA_ALL
   // TZ settings
```

```
      params.reg.TZSEL.bit.CBC1 = 0;
      params.reg.TZSEL.bit.OSHT1 = 0;
      params.reg.TZSEL.bit.CBC2 = 0;
      params.reg.TZSEL.bit.OSHT2 = 0;
      params.reg.TZSEL.bit.CBC3 = 0;
      params.reg.TZSEL.bit.OSHT3 = 0;
      PLX_PWM_configure(EpwmHandles[0], 4, &params);
      // configure deadtime to 1.000000e-06/1.000000e-06 seconds
(rising/falling)
      PLX_PWM_setDeadTimeCounts(EpwmHandles[0], 90.000000, 90.000000);
      // PWM sequence starting with active state
      PLX_PWM_setSequence(EpwmHandles[0], 1);
      EPwm4Regs.ETSEL.bit.SOCASEL = 2;
      EPwm4Regs.ETPS.bit.SOCAPRD = 1;
      EPwm4Regs.ETSEL.bit.SOCAEN = 1;
   }
   PLX_DIO_sinit();
   for (int i = 0; i < 1; i++)
   {
      DoutHandles[i] = PLX_DIO_init(&DoutObj[i], sizeof(DoutObj[i]));
   }
   {
      PLX_DIO_OutputProperties_t props = {
         0
      };
      props.type = PLX_DIO_PUSHPULL;
      props.enableInvert = false;
      PLX_DIO_configureOut(DoutHandles[0], 34,  &props);
   }
   DISPR_sinit();
   DISPR_configure((uint32_t)(9000), PilHandle, &TaskObj[0],
                   sizeof(TaskObj)/sizeof(DISPR_TaskObj_t));
   DISPR_registerIdleTask(&En_Boost_Cont_background);
   DISPR_registerSyncCallback(&En_Boost_Cont_syncTimers);
   DISPR_setPowerupDelay(10);
   {
      static int taskId = 0;
      // Task 0 at 1.000000e+04 Hz
      DISPR_registerTask(0, &Tasks, 9000L, (void *)&taskId);
   }
   {
      static int taskId = 1;
      // Task 1 at 2.000000e+00 Hz
      DISPR_registerTask(1, &Tasks, 45000000L, (void *)&taskId);
   }
   EALLOW;
   AdcRegs.INTSEL1N2.bit.INT1CONT = 0;  // disable ADCINT1 Continuous mode
   AdcRegs.INTSEL1N2.bit.INT1SEL = 3;  // setup EOC3 to trigger ADCINT1
   AdcRegs.INTSEL1N2.bit.INT1E = 1;  // enable ADCINT1
   AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1;  // ADCINT1 trips after AdcResults
latch
   EDIS;
   EALLOW;
   PieVectTable.ADCINT1 = En_Boost_Cont_baseTaskInterrupt;
   PieCtrlRegs.PIEIER1.bit.INTx1 = 1;
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
   EDIS;


   // Post init code (for modules that depend on other modules)
   PLX_SCI_setInterface(SerialHandles[0], SciHandles[0]);
   PLX_Serial_configViaPinset(SerialHandles[0], 0, 22500000);
```

```
    PLX_Serial_setupPort(SerialHandles[0], 115200);

    // If applicable, boot secondary core(s)
}
```

BIBLIOGRAFÍA

[1] INTERNATIONAL ENERGY AGENCY, *Renewables 2023: Analysis and forecast to 2028.* IEA, 2024.

[2] GREEN, Martin A. *Solar Cells: Operating Principles, Technology, and System Applications*. Prentice-Hall, 1982.

[3] VILLALVA, Marcelo Gradella, GAZOLI, Jonas Rafael y FILHO, Ernesto Ruppert. *Comprehensive approach to modeling and simulation of photovoltaic arrays*. IEEE Transactions on Power Electronics, 2009.

[4] BLAABJERG, Frede, TEODORESCU, Remus, LISERRE, Marco y TIMBUS, Adrian V. *Overview of control and grid synchronization for distributed power generation systems*. IEEE Transactions on Industrial Electronics, 2006.

[5] MERTENS, Konrad. *Photovoltaics: fundamentals, technology, and practice*. John Wiley & Sons, 2018.

[6] SERGIO VÁZQUEZ PÉREZ, *Apuntes de Integración de Energías Renovables*. Departamento de Ingeniería Electrónica de la Universidad de Sevilla, 2022.

[7] PLEXIM GmbH, *PLECS User Manual*, 2024.

[8] PLEXIM GmbH, *PLECS Processor-in-the-Loop (PIL) User Manual*, 2022.

[9] PLEXIM GmbH, *PLECS TI C2000 Target Support User Manual*, 2024.

[10] PLEXIM GmbH, *Simple PIL Model: Embedded Code Generation Demo Model*, 2022.

[11] PLEXIM GmbH, *Introduction to the C-Script Block*, Tutorial Version 1.0.

[12] KYOCERA Corporation, *KC200GT High Efficiency Multicrystal Photovoltaic Module*.

[13] TEXAS INSTRUMENTS, *TMS320F28379D Microcontroller Datasheet*.

[14] S. VÁZQUEZ PÉREZ y A. MÁRQUEZ ALCAIDE, *Práctica 3: Modelado y simulación de un sistema fotovoltaico conectado a la red elétrica*. Departamento de Ingeniería Electrónica de la Universidad de Sevilla, 2022.

[15] S. VÁZQUEZ PÉREZ, J. I. LEÓN GALVÁN y L. GARCÍA FRANQUELO, *Apuntes de Electrónica de Potencia*. Departamento de Ingeniería Electrónica de la Universidad de Sevilla, 2022.

[16] MICROCHIP TECHNOLOGY INC., *Practical Guide to Implementing Solar Panel MPPT Algorithms*, Application Note AN1521 (DS00001521A), 2013.