Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Implementación del sistema de control de un convertidor de potencia usando la técnica PIL para un sistema eléctrico aislado basado en generación solar fotovoltaica

Autor: José Luis Romero Arias
Tutor: Juan Manuel Carrasco Solís
Cotutor: Eduardo Galván Díez

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

Sevilla, 2020

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Implementación del sistema de control de un convertidor de potencia usando la técnica PIL para un sistema eléctrico aislado basado en generación solar fotovoltaica

Autor:

José Luis Romero Arias

Tutor:

Juan Manuel Carrasco Solís

Catedrático

Cotutor:

Eduardo Galván Díez

Catedrático

Dpto. Ingeniería Electrónica

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Carrera: Implementación del sistema de control de un convertidor de potencia usando la técnica PIL para un sistema eléctrico aislado basado en generación solar fotovoltaica

Autor: José Luis Romero Arias

Tutor: Juan Manuel Carrasco Solís

Cotutor: Eduardo Galván Díez

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis maestros

A las buenas personas que me han aportado

Agradecimientos

La realización y entrega de este proyecto supone, sin duda, un alivio enorme y el final de mi etapa como estudiante en la Universidad de Sevilla. Una etapa marcada por épocas muy duras, pero también de crecimiento personal.

Han sido varias las personas que me han aportado de manera positiva a lo largo de este camino, y me gustaría nombrar en especial a:

Mi familia, en concreto a mis padres, Maribel y Javier, y a mi tía-abuela Fulgencia, por tratar de transmitirme todo su apoyo y ánimo, desde siempre. Por hacerme entender el valor de la constancia y el esfuerzo. A mis hermanas, Ana Mengjie, Pablo y Manuel por ofrecerme su cariño y los medios que hayan hecho falta para hacer más llevadero el camino.

Lucía, por la paciencia infinita, por conseguir mejorar los días con solo una video llamada y relativizar las dificultades.

Mis amigos, por estar cuando lo he necesitado y contagiar su risa tan fácilmente.

Luisa, por ayudarme a acercarme al camino de la felicidad en los malos tiempos.

Juan Manuel y Eduardo, tutores de este trabajo, por ofrecerme un tema de proyecto que se adaptaba a mis preferencias y por su ayuda para poder finalizarlo.

José Luis Romero Arias

Sevilla, 2020

Resumen

En el presente documento se analiza una nueva metodología de diseño para la implementación y ajuste del sistema de control de unas aplicaciones de electrónica de potencia basada en la técnica de *Processor In the Loop* (PIL). En concreto, en este proyecto se trata de modelar y simular sistemas complejos de plantas de generación eléctrica basada en energías renovables mediante paquetes de simulación como MATLAB – Simulink, permitiendo que la implementación de la estrategia de control se realice en la plataforma hardware real donde se llevaría a cabo la aplicación final. Esto permite poder diseñar, desarrollar y ajustar el sistema de control real sin necesidad de disponer del sistema de la aplicación final.

En este Trabajo Fin de Grado se propone una simulación de un sistema de generación eléctrica aislada, basado en energía fotovoltaica con batería. A este sistema, se le ha aplicado la técnica de simulación *Processor In the Loop* (PIL), la cual consiste mayormente en implementar un algoritmo de control en una placa de desarrollo o sistema microprocesador embebido.

Se ha propuesto el algoritmo de control que implementa el Seguimiento del Punto de Máxima Potencia (MPPT) de una planta de generación fotovoltaica implementado en un convertidor de potencia DC/DC elevador. Este algoritmo MPPT propuesto implementa la estrategia de *Perturbación y Observación* (P&O), lo que nos permitirá garantizar la máxima captura de potencia generada por el sistema de generación fotovoltaico.

Para la simulación, la planta de nuestro sistema se ejecutará en el software Simulink (Matlab), mientras que el algoritmo de control lo hará en la placa de desarrollo con el microprocesador *TMS320F28379D* de la familia *C2000 Delfino* del fabricante Texas Instruments.

Abstract

In the present document a new methodology of design for the implementation and adaptation of the control system of some power electronic applications based on the *Processor In the Loop* (PIL) technique has been analysed. Specifically, this project studies modeling and simulating complex systems of power generation plants based on renewable energies using simulation packages such us, MATLAB – Simulink, allowing that the control approach implementation runs in the real hardware platform where the final application is carried out. This allows the design, development and adaption of the real control system without the need of the final application system.

This Final Degree Thesis proposes a simulation of an isolated power generation system, based on photovoltaic energy with battery storage. In this system, *Processor In the Loop* (PIL) simulation approach has been applied, which is comprised by implementing a control algorithm in a development board or a embedded microprocessor system.

It has been proposed the control algorithm that implements the Maximun Power Point Tracking (MPPT) of a photovoltaic power plant implemented in a power converter DC/DC boost. This algorithm MPPT applies the *Perturb and Observe* (P&O) approach, wihch will allow to guarantee the maximun capture of the power generate by the photovoltaic generation system.

For he simulation, our system plant will be run in the software Simulink (Matlab), whereas the control algorithm will be run in the development board with the microprocessor *TMS320F28379D* of *C2000 Delfino* by TI manufacturer.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Lista de abreviaturas y siglas

ADC	Analogic to Digital Converter, convertidor de analógico a digital
EDI	Entorno de Desarrollo Integrado
CCS	Code Composer Studio
COM	Communication port, puerto de comunicación
CPU	Central Processing Unit, unidad central de procesamiento
DC	Duty Cicle, ciclo de trabajo
DO	Digital Output, salida digital
DSP	Digital Signal Processor, procesador digital de señal
GEI	Gases de Efecto Invernadero
IGBT	Insulated Gate Bipolar transistor, transistor bipolar de puerta aislada
kW	Kilowatts, kilovatios
LED	Light-Emitting Diode, diodo emisor de luz
m	metro
MPP	Maximun Power Point, punto de máxima potencia
MPPT	Maximun Power Point Tracker, seguimiento del punto demáxima potencia
PI	Proportional-Integral (controller), controlador proporcional e integrador
PID	Proportional-Integral-Derivative (controller), controlador proporcional, integrador y derivativo
PIL	Processor In the Loop
P&O	Perturb and Observe, perturbación y observación
PWM	Pulse-Width Modulation, modulación por ancho de pulso
TLC	Target Language Compiler
USB	Universal Serial Bus
W	Watt, vatio

1 INTRODUCCIÓN

Hay dos cosas infinitas: el universo y la estupidez humana. Y del universo no estoy seguro.

- Albert Einstein -

Nos encontramos en un contexto que podemos entender como un punto de inflexión de gran importancia en el que sin duda tenemos que tomar parte, obrando de la mejor manera que sepamos. Este contexto de crisis climática provocado por la actividad humana es un hecho y consenso total en la comunidad científica [1]. Entre otras de las causas, se encuentran la producción, el transporte y el consumo insostenible de los combustibles fósiles.

Vemos evidencias a lo largo del planeta de la grave situación a la que nos enfrentamos tales como, la subida del nivel del mar, sequías y deshielo de los casquetes polares. Además, en estos últimos años ha aumentado el número y la intensidad de los episodios climáticos extremos.

En las Conferencias de las Naciones Unidas sobre Cambio Climático se han conseguido importantes acuerdos para intentar frenar esta situación de crisis climática, tales como son el protocolo de Kioto (COP3, 1997) o el acuerdo de París (COP21, 2015). En la última edición celebrada en Madrid (COP25, 2019), se ha hecho incapié en la importancia de conseguir ciertos hitos fundamentales que se han recogido en las anteriores ediciones como:

- Evitar que el incremento de la temperatura media global supere los 2ºC.
- Equilibrar las emisiones y las absorciones de los gases de efecto invernadero (GEI).
- Sentar las bases para una transición hacia modelos de desarrollo bajos en emisiones. [2]

Sobre un contexto algo más técnico, sabemos que gracias al avance en la tecnología se ha conseguido un aumento en la eficiencia, abaratamiento de los costes de los materiales y, además, parece que las políticas sobre el ámbito de las energías renovables también están acompañando (p. ej. con la eliminación del "impuesto al sol"). Dados este conjunto de sucesos, se está asentando una situación muy favorable para atajar el problema desde un ámbito ingenieril.

En el amplio abanico de posibilidades que encontramos en un proyecto de ingeniería, en este documento en concreto trabajaremos sobre la etapa de verificación y validación del código de control y sus funcionalidades. Trataremos con la técnica de simulación PIL (Processor In the Loop), en dicha técnica se implementa el código de nuestro bloque de control en un procesador embebido. Es decir, se ha verificado el funcionamiento de un sistema de control real, con el hardware y software final de la aplicación. La ventaja de este tipo de metodología es que nos permite identificar si el procesador es capaz de ejecutar la lógica de control desarrollada. Si ocurre algún fallo, podremos volver al código para intentar solucionarlo.

Esto ha sido una breve puesta en contexto y las razones principales para la elección del tema sobre el que trata el presente trabajo.

1.1 Definición de Processor In the Loop (PIL)

El PIL es una técnica de test que nos permite implementar y verificar el funcionamiento de un sistema de control real, con el hardware y software final de la aplicación. Es decir, mientras por un lado el modelo de la planta se ejecuta en un software de simulación, el control propuesto para la aplicación lo hace en un microcontrolador.

El software de simulación sobre el que trabajaremos es Simulink (Matlab), que es un software muy utilizado en diferentes campos de la ingeniería. El sistema de desarrollo que se nos asigna es el launchPad LAUNCHXL-F28379D (TMS320F28379D) C2000 Delfino de Texas Instruments, ya que es el microcontrolador del que dispone el departamento de Ingeniería Electrónica de la Universidad de Sevilla. Esta placa de desarrollo permite la técnica Processor In the Loop (PIL).

A continuación, y con ayuda de la **Figura 1-1**, veremos de manera simplificada cómo quedaría un escenario en el que empleamos la técnica PIL. En principio tenemos un bucle compuesto por una planta "plant (software)" que queremos controlar y su respectivo bloque de control "controller (software)". Ambos bloques se han ejecutado en nuestro software de simulación. Por otro lado, al introducir la técnica PIL, incorporamos el bloque "PIL (target)" en nuestro escenario. Este último bloque hace las veces de controlador, pero en vez de ejecutarse en el software de simulación lo hace en nuestro microcontrolador. Debido a que el bloque PIL tiene la misma función que el bloque de control situado en el software de simulación, decidimos colocarlos en paralelo. En última instancia, se trata de observar la fidelidad con la que el control implementado en el hardware "PIL (target)" es capaz de reproducir el comportamiento del bloque de control implementado originalmente en el software de simulación "controller (software)".

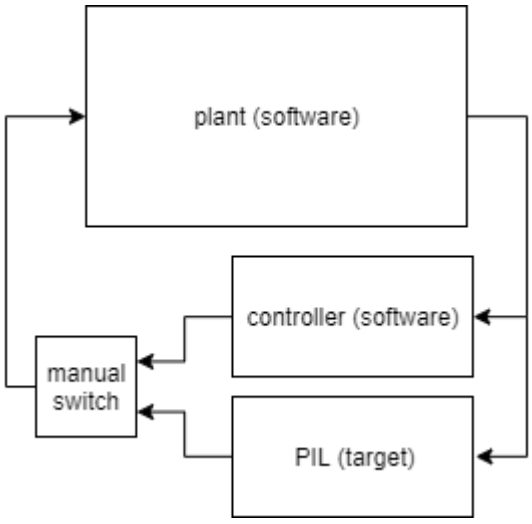


Figura 1-1. Esquema simplificado de la técnica PIL.

Como se puede observar, los datos de salida de la planta pasan por ambos bloques de control. Por lo tanto, es recomendable colocar a la salida de ambos bloques un interruptor manual para poder decidir con qué implementación se desea simular el sistema completo.

2 OBJETIVOS

Los objetivos planteados para el desarrollo de este trabajo fin de grado fueron los siguientes:

- Comprender el funcionamiento del modelo original que se nos asignaba para el software de simulación Simulink (Matlab). Se ha prestando especial interés al bloque de control del convertidor de potencia elevador DC/DC, sobre el cuál se realiza la técnica PIL.

- Aprender a implementar la técnica PIL. Lo cual implica, por un lado, la instalación y configuración del software de simulación a usar y, por otro lado, la incorporación del hardware a la simulación.

- Analizar los límites de la técnica PIL en la familia DSP de TI F28379D. Se proponen como métodos: ver hasta dónde es capaz de ser programada la placa de desarrollo hardware e interactuar con los periféricos de entrada/salida de la misma placa.

Los dos primeros objetivos se han desarrollado con éxito. Sin embargo, con el tercero hemos tenido la imposibilidad de finalizarlo completamente debido a la dependencia del uso de los laboratorios, los cuales fueron cerrados con motivo de una causa de fuerza mayor como fue la pandemia del COVID-19.

3 INTRODUCCIÓN AL SISTEMA FOTOVOLTAICO AISLADO CON BATERÍA

Los sistemas de generación eléctrica basado en energía fotovoltaica con batería "son muy utilizados en aplicaciones donde no existe acometida eléctrica y es necesario alimentar cargas de tensión alternas senoidales o cargas en tensión continua. Típicamente se utilizan para alimentación de sistemas informáticos o de telecomunicación instalados de forma remota." [3]. En la **Figura 3-1**, se muestra un esquema donde se observan las partes principales que componen el sistema fotovoltaico que tratamos en este proyecto.

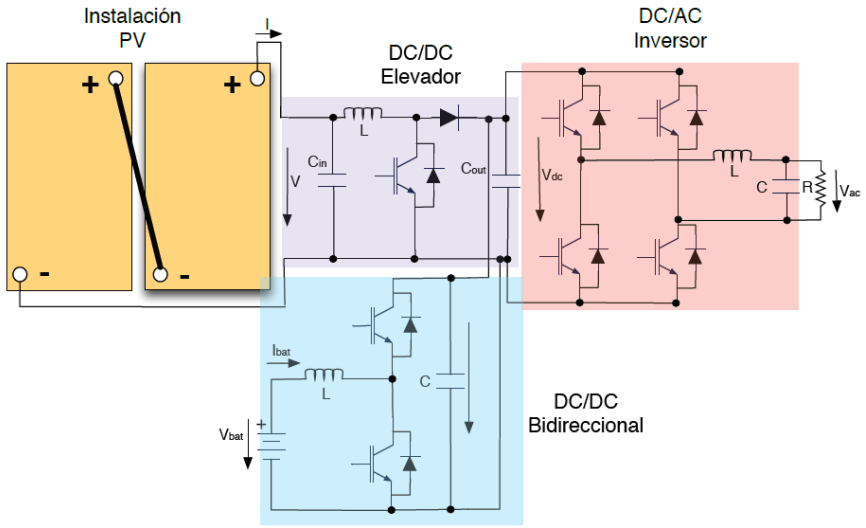


Figura 3-1. Esquema de un sistema de generación eléctrica basado en energía fotovoltaica con batería.

El sistema se puede dividir en cuatro bloques principales. El conjunto de paneles fotovoltaicos, el convertidor de potencia DC/DC elevador, el convertidor de potencia DC/DC bidireccional y el convertidor de potencia DC/AC inversor. Cabe mencionar que la batería estará conectada al convertidor bidireccional. Además, a continuación del bloque convertidor inversor, irá colocado un filtro. Para finalizar, y a continuación del filtro, se encuentra la carga resistiva aislada, la cual pretendemos alimentar con este sistema fotovoltaico.

3.1 Paneles fotovoltaicos

Los paneles fotovoltaicos son el primer bloque del sistema y uno de los más importantes, pues son los encargados de transformar la energía proveniente del sol en energía eléctrica. La energía que producen los paneles se obtiene mediante la captación de los fotones emitidos por la radiación solar, directa e indirecta, y la producción de pares electrón-hueco, que contribuyen a la corriente de salida del panel. Por otro lado, los paneles están compuestos por un conjunto de células fotovoltaicas. Son estas últimas las encargadas de la generación eléctrica mediante el efecto fotovoltaico.

3.1.1 Efecto fotovoltaico

La conversión de la energía solar en electricidad es posible gracias a las propiedades de materiales semiconductores como el silicio. Los paneles fotovoltaicos y, por tanto, sus respectivas celdas, están fabricadas con obleas de silicio. En el efecto en cuestión, tenemos una celda conectada a una carga y los rayos del sol inciden sobre la celda, se produce entonces una diferencia de potencial en los extremos de la carga y circula una corriente por ella.

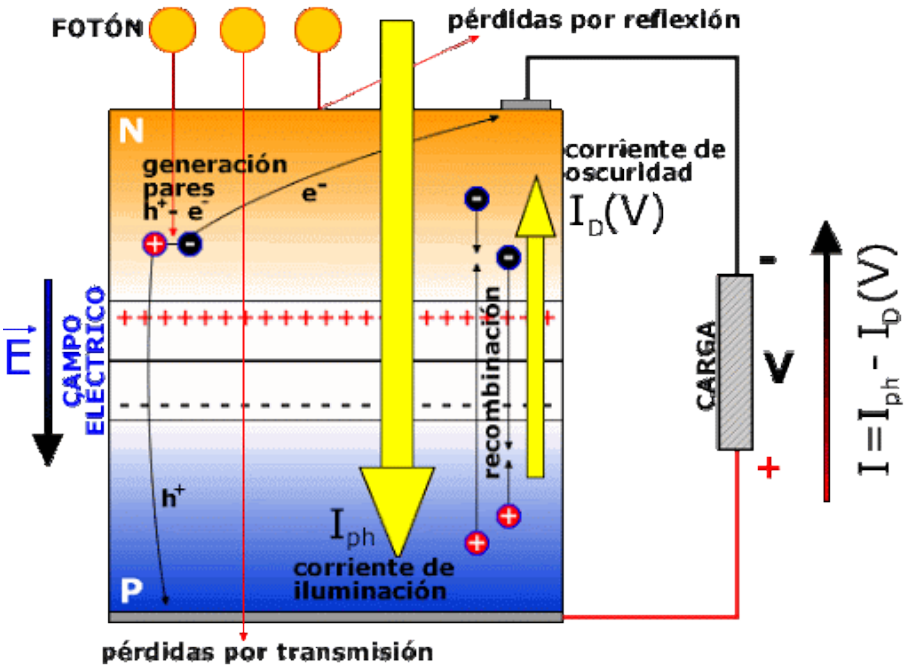


Figura 3-2. Funcionamiento de una célula fotovoltaica.

En la oblea, cada átomo de silicio comparte sus cuatro electrones de valencia con los átomos vecinos más próximos. Sin embargo, el material no es buen conductor porque los electrones están bien sujetos al átomo. Es entonces cuando se pasa al dopaje de la oblea. La parte superior de la oblea se dopa con un elemento que tenga cinco electrones en la capa de valencia (p.ej. fósforo), dicha capa se llamará *semiconductor tipo n*. Por otra parte, la parte inferior de la oblea se dopa con un material con tres electrones en la capa de valencia (p.ej. boro), esta capa resultará ser *semiconductor tipo p*.

Podemos concluir que la zona n dispone de electrones libres y la zona p dispone de huecos ("carga" positiva), ambos disponibles para la conducción. Además, la agitación térmica del material hace que, de forma espontánea, algunos electrones del lado n se difundan hacia el lado p, ocupando huecos de ese lado y dejando otros tantos huecos en el lado n. Debido a este desplazamiento de electrones, dejan parte del lado p con cierta carga negativa neta. Por otro lado, el lado n queda con otra carga positiva. Esta zona cargada es la unión de ambos semiconductores, se llama región de deplexión.

Las cargas netas que se encuentran en la zona de deplexión generan un campo eléctrico. Dicho campo, genera a su vez una tensión entre ambos lados de la zona de deplexión. La luz solar está compuesta por unas partículas elementales llamadas fotones. Cuando un fotón que penetra en la celda, alcanza la zona de deplexión, puede transmitirle su energía a un electrón extrayendo el electrón del átomo y generando un hueco. El campo eléctrico que existe en la zona de deplexión, impulsa el electrón hacia arriba. Por su parte, el hueco, que se comporta como una partícula positiva, es impulsado hacia abajo.

Si, como se observa en la **Figura 3-2**, soldamos un contacto eléctrico en la parte superior de la cara n y otro en la parte inferior de la cara p y, además, unimos ambos contactos con un cable, entonces tendremos un circuito eléctrico. Muchos electrones haciendo lo mismo constituyen la corriente fotovoltaica. [4] [5]

"Un circuito equivalente simplificado de la célula fotovoltaica, consiste en una fuente de intensidad en paralelo con un diodo". En la siguiente figura se muestra la tendencia de la intensidad tanto de Lafuente de intensidad como del diodo. La intensidad de dicha fuente "es la generada internamente en la célula solar y es proporcional a la intensidad de radiación" [6]. Además, el circuito equivalente cuenta con dos resistencias, una en serie y otra en paralelo a la fuente de tensión y al diodo anteriormente comentados.

La *corriente de iluminación* (I_{ph}), intensidad de la fuente de intensidad, es aquella que se debe a la generación de portadores que produce la iluminación. Por otro lado, la *corriente de oscuridad* (I_D (V)), intensidad del diodo, es debida a la recombinación de portadores que produce el voltaje externo necesario para poder entregar energía a la carga. Finalmente, la intensidad que recorre la carga será el resultado de la primera menos la segunda, como se observa en la **Figura 3-3**. Donde *e* es la carga del electrón, *k* es la constante de Boltzmann, T_C es la temperatura a la que opera la célula solar en ºC e I_0 es la corriente inversa de saturación.

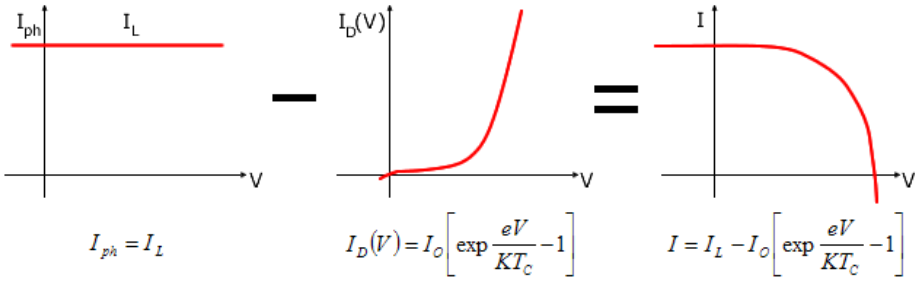


Figura 3-3. Característica I–V de la célula fotovoltaica.

3.2 Convertidor DC/DC elevador

La tensión que se obtiene a la salida de los paneles generalmente no tiene el valor deseado con el cual trabaja la cargar. Es por ello, que esta etapa, el convertidor de potencia se va a encargar de elevar la tensión de salida de la etapa con respecto a la tensión de la entrada. Además, en esta misma etapa se emplea el algoritmo de control Seguimiento del Punto Máximo de Potencia (MPPT) basado en la estrategia Perturbación y Observación (P&O).

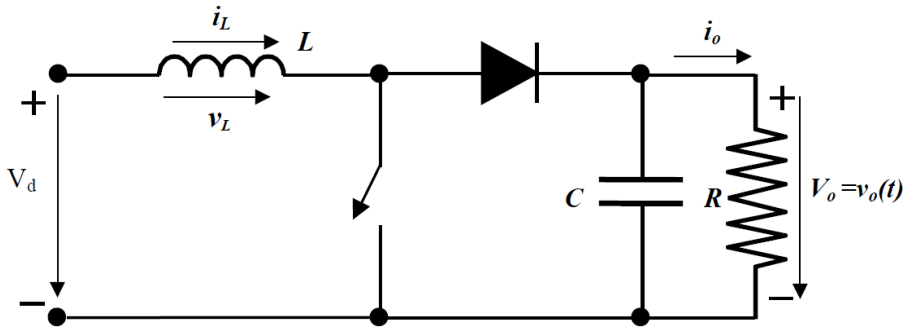


Figura 3-4. Circuito del convertidor de potencia DC/DC elevador.

El circuito se compone, de izquierda a derecha en la imagen anterior, de una tensión constante a la entrada (V_d), una bobina, un transistor de potencia del transistor bipolar de puerta aislada (IGBT) que hace las veces de interruptor, un diodo, una capacidad y una resistencia con su respectiva tensión de salida (V_o). En términos del escenario que nos ocupa en este proyecto y observando la **Figura 3-1**, se concluye que V_d se corresponde con la tensión entregada por el conjunto de módulos fotovoltaicos y V_o con la tensión que se entrega al bus de

tensión continua. Más adelante, en la sección **5.1**, se explica con algo más de precisión el comportamiento del IGBT como interruptor.

El IGBT, tiene una alta impedancia de entrada en la puerta y necesita muy poca energía para conmutarlo entre los modos de saturación y corte. Por lo tanto, el resultado será dos circuitos distintos dependiendo de si el IGBT se encuentra en modo de conducción, es decir, se comporta como interruptor cerrado, o por el contrario se encuentra en modo de no conducción, que actúa como si fuera un interruptor abierto.

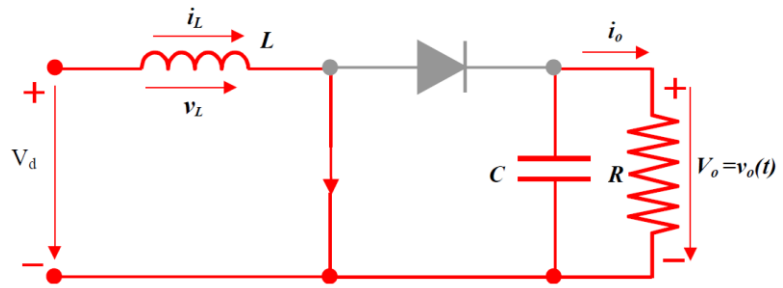


Figura 3-5. Convertidor DC/DC elevador con interruptor cerrado (conducción).

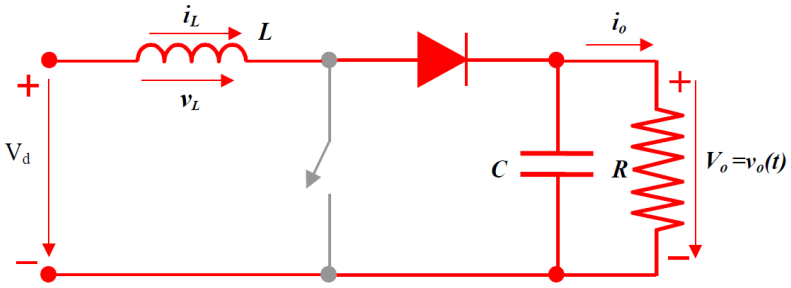


Figura 3-6. Convertidor DC/DC elevador con interruptor abierto (no conducción).

3.2.1 Algoritmo de control MPPT con estrategia P&O

Como se ha comentado anteriormente, parte del algoritmo de control propuesto para el convertidor DC/DC elevador va a ser el de MPPT con estrategia P&O. Las variables de entrada de este algoritmo van a ser la tensión (V_{pv}) y la intensidad (I_{pv}) a la salida de los paneles y la variable de salida del algoritmo es la tensión de referencia (V_{pvRef}) del sistema fotovoltaico para alcanzar el MPP. Finalmente, a la salida de la parte de control se obtiene una señal que va a controlar el disparo que ataca directamente a la puerta del IGBT del convertidor.

El objetivo del algoritmo MPPT es mantener en funcionamiento el sistema entorno al punto de máxima potencia (Maximun Power Point en inglés, representado con MPP) controlando las variables de tensión e intensidad para que lleguen a sus valores máximos (VM, IM). De esta manera, se consigue optimizar la captura de energía. Existen varias estrategias para implementar el algoritmo MPPT, en este proyecto se utilizará la conocida como P&O.

En la **Figura 3-7** se muestra una gráfica de ejemplo que nos servirá para visualizar el funcionamiento de la estrategia P&O. En dicha figura se representan las curvas de potencia y tensión de un sistema fotovoltaico, eje de ordenadas y abscisas respectivamente. Como referencia, tenemos el valor de tensión (V_{MPP}) para el cual se alcanza el valor de máxima potencia (P_{max}). De manera simplificada, tenemos dos posibles maneras de aproximarnos al valor V_{MPP}. Por un lado, se dice que nos aproximamos por la izquierda al valor V_{MPP} cuando el incremento de potencia y de tensión, de un instante determinando al instante justamente anterior, es positivo

para ambas variables. Por otro lado, nos aproximamos por la derecha a V_{MPP} cuando el incremento de potencia y tensión es positivo y negativo respectivamente.

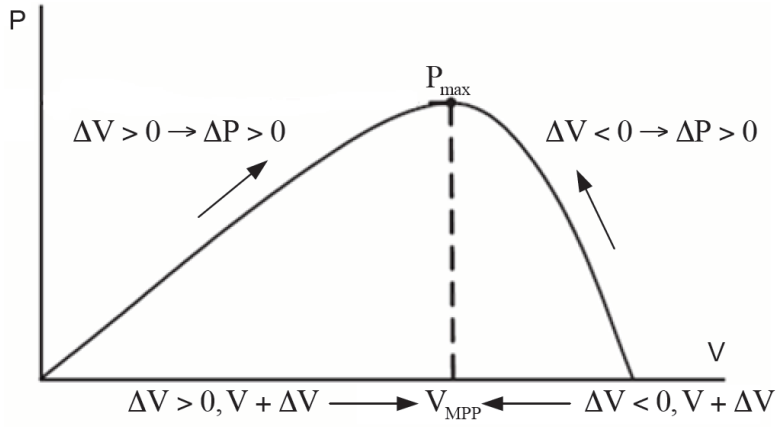


Figura 3-7. Curva P-V de ejemplo de un sistema fotovoltaico.

En la **Figura 3-8** se observa el diagrama de flujo con el que funciona la estrategia P&O, donde *k* es el instante actual y *k-1* es el instante justamente anterior. En dicha estrategia, se procederá a incrementar o decrementar la tensión de referencia dependiendo de si el incremento de la potencia y tensión con respecto al instante justamente anterior es positivo o negativo.

Entrando más en detalle sobre la estrategia P&O, inicialmente se muestran la tensión y la intensidad a la salida del sistema fotovoltaico para posteriormente calcular la potencia. La potencia recientemente calculada se compara con aquella del instante justamente interior, $P_k - P_{k-1}$. Si resulta que la diferencia es igual a cero, acaba la ejecución del algoritmo para ese instante de tiempo *k* y se pasa al siguiente instante. En caso de que la diferencia no sea cero, se comprueba si ese incremento de potencia ($P_k - P_{k-1}$) es positivo o negativo.

A. Para el caso de que el incremento de potencia sea positivo, se comprueba si el incremento de tensión del instante actual con respecto al instante justamente anterior $V_k - V_{k-1}$, es positivo o negativo. Si el incremento de tensión es positivo, V_{pvRef} se incrementa una pequeña cantidad y si el incremento de tensión es negativo, V_{pvRef} se decrementa una pequeña cantidad.

B. Para el caso de que el incremento de potencia sea negativo, se comprueba si el incremento de tensión del instante actual con respecto al instante justamente anterior $V_k - V_{k-1}$, es positivo o negativo. Si el incremento de tensión es positivo, V_{pvRef} se decrementa una pequeña cantidad y si el incremento de tensión es negativo, V_{pvRef} se incrementa una pequeña cantidad.

Para ambos casos anteriores, el algoritmo termina y como resultado, da a la salida la variable V_{pvRef}. Seguidamente el algoritmo se vuelve a ejecutar operando con las variables correspondientes para el siguiente instante de tiempo.

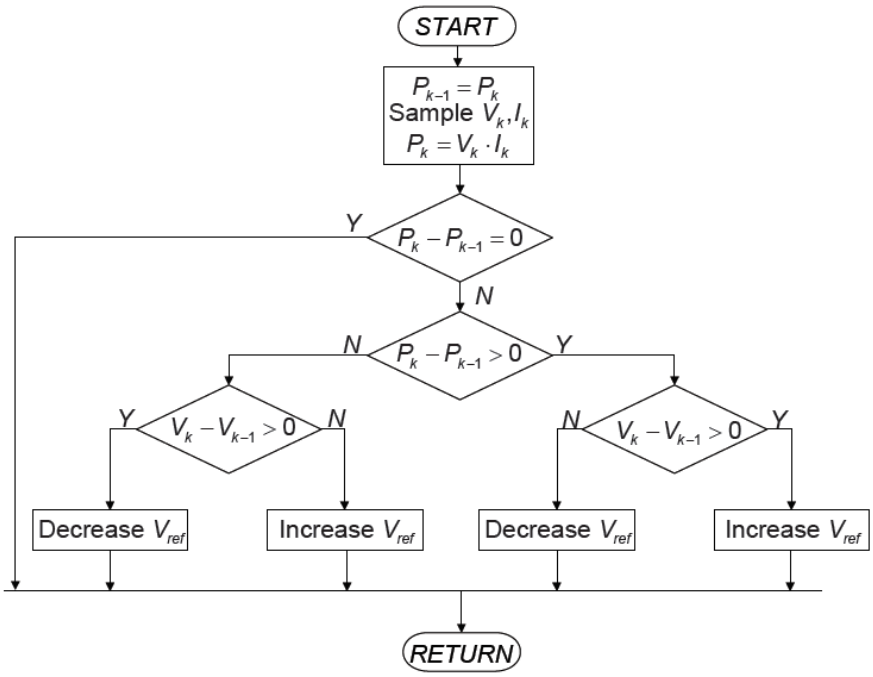


Figura 3-8. Estrategia P&O.

Además del algoritmo MPPT mencionado anteriormente, el sistema de control final para un convertidor DC/DC elevador puede llegar a ser algo más complejo. Para el escenario definitivo de este proyecto, sistema fotovoltaico aislado con batería, se propone un sistema de control cuyo conjunto de operaciones obtienen D. Dicha variable, que hace las veces de señal de control (para más detalles, ver sección **5.1**), se emplea a su vez para obtener la señal PWM, señal que controlará directamente el funcionamiento del transistor IGBT.

Las razones para realizar esas operaciones son las siguientes. Las tensiones a la salida de los paneles (V) y a la salida del algoritmo MPPT (V_{ref}) se elevan al cuadrado porque se pretende controlar el error de la energía del condensador a la salida de los paneles. Como resultado del error calculado anteriormente, lo que obtenemos es proporcional a la potencia de referencia (P_{ref}). Posteriormente se divide por la tensión a la salida de los paneles (V) para obtener la intensidad de referencia (I_{ref}), puesto que, al final, el convertidor DC/DC elevador controla dicha intensidad de referencia. Luego ese incremento de tensión nos dará la tensión PWM que dividiendo por la tensión dada por los paneles (V) obtenemos el D.

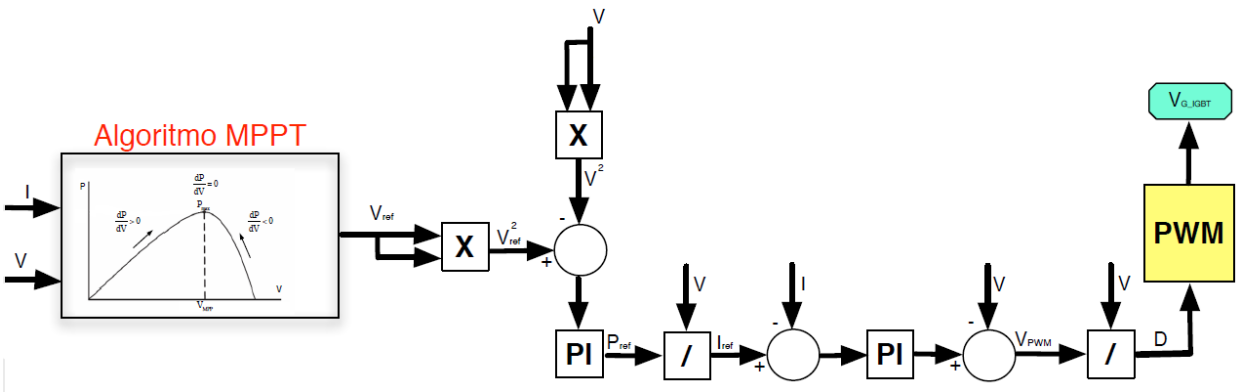


Figura 3-9. Algoritmo MPPT + el resto del sistema de control.

El sistema de control que se observa en la **Figura 3-9** pertenece al convertidor DC/DC elevador. La mayor parte de este sistema de control se ha implementado en el hardware en la sección **5.3**.

3.3 Convertidor DC/DC bidireccional y batería

El convertidor bidireccional tiene la capacidad de transmitir la energía en ambos sentidos. En la **Figura 3-10** se observa una batería conectada al sistema de generación eléctrica fotovoltaico que se desea cargar. Sin embargo, también se pretende que la misma batería suministre energía a la carga en ausencia de energía eléctrica generada por los paneles. Por el lado izquierdo del convertidor, se encuentra la batería. Se trata del lado de menor tensión, y por el lado derecho, se encuentra la salida del convertidor que conecta con el bus de continua, el cual es el lado de mayor tensión.

Además, este convertidor se encarga de mantener un valor constante deseado en el bus de continua. La batería se carga o se descarga dependiendo de si la energía captada por los paneles fotovoltaicos es suficiente para alimentar la carga aislada o no, respectivamente. Por esta razón, el carácter bidireccional es imprescindible.

El modelo está compuesto por una bobina encargada de almacenar la energía, dos condensadores (aunque uno no aparezca en la **Figura 3-10**, se sitúa justo en el lado derecho de la batería) cuyas funciones son filtrar las corrientes y mantener constantes las tensiones. También, el modelo consta de dos transistores IGBT empleados como interruptores, estos son los encargados de permitir el flujo de energía en uno u otro sentido. El modelo elegido para la formación del convertidor DC/DC es una fusión de un convertidor buck (reductor) y un convertidor boost (elevador). Las ventajas de esta topología es la simplicidad y la facilidad para acoplarse al convertidor DC/AD inversor.

Con esto anterior, "se busca que se permita el flujo de energía del lado de mayor tensión al de menor tensión cuando se desea cargar la batería, funcionando como buck (reductor). De igual modo se busca también permitir el flujo de energía del lado de menor tensión al de mayor tensión cuando se desea descargar energía a la red, funcionando como boost (elevador)." En la **Figura 3-10** se observa el esquema del convertidor bidireccional. [3]

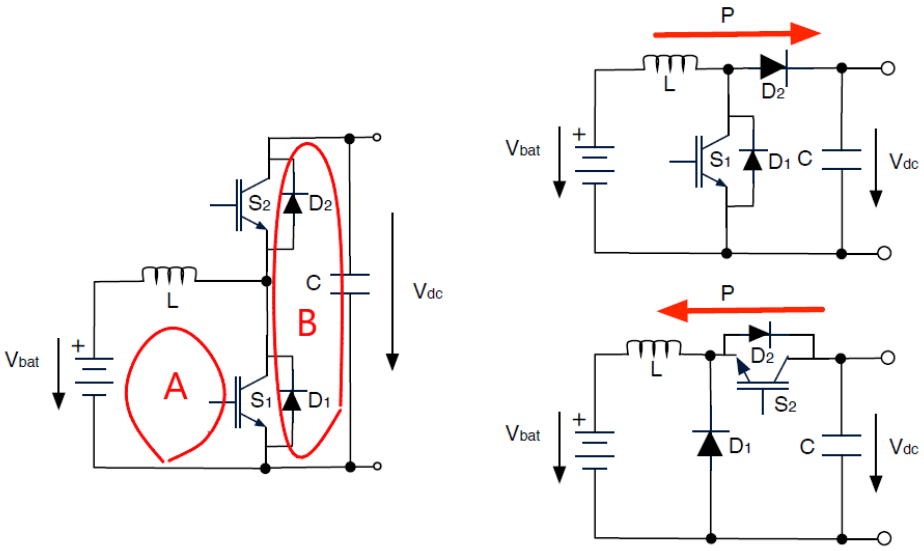


Figura 3-10. Convertidor DC/DC bidireccional basado en topología elevadora y reductora. Diferenciación entre la malla A y la B.

Para una mayor claridad y brevedad en la descripción del funcionamiento del convertidor DC/DC bidireccional vamos a denominar al transistor IGBT inferior como interruptor "S_1" y al transistor IGBT superior como interruptor "S_2". "D_1" y "D_2" serán sus respectivos diodos, aquellos que se sitúan en paralelo al propio IGBT.

1. Funcionamiento como boost (elevador).
    a. S_2 permanece abierto. Con S_1 cerrado, la corriente circula desde la batería a través de la bobina, atravesando el interruptor S_1, cerrando el circuito a través de la batería. En esta etapa, la bobina almacena energía en el campo magnético que se crea al pasar la corriente eléctrica por ella.
    b. Cuando se abre S_1, se interrumpe el paso de corriente por la bobina. Esta reacciona liberando energía almacenada y esto hace aumentar la tensión a la malla A (imagen 3-10) que ahora es mayor que la tensión en la malla B (imagen 3-10). Por esta razón la corriente gira hacia la derecha pasando por el diodo D_2.
    c. Al cerrar S_1 de nuevo, vuelve a cargarse la bobina, por lo que se repite el proceso y volvemos a situarnos en el apartado 1.a.

2. Funcionamiento como buck (reductor).
    a. S_1 permanece abierto. Con S_2 cerrado, la intensidad circula de la fuente de tensión a través del interruptor S_2, atravesando la bobina, la cual se carga de energía como en el apartado anterior 1.a. Luego se cierra el circuito a través de la batería.
    b. Cuando se abre S_2, la Fuente de tensión ya no puede enviar corriente a la batería. Por otro lado, se libera la energía acumulada en la bobina enviando corriente la batería y se cierra el circuito a través del diodo D_1.
    c. Al cerrar S_2 de nuevo, vuelve a cargarse la bobina, por lo que se repite el proceso y volvemos a situarnos en el apartado 2.a.

3.4 Convertidor DC/AC inversor

Por último, el bloque del modelo situado en la parte derecha de la **Figura 3-1**, se observa el convertidor DC/AC inversor cuya función es convertir la tensión continua que le llega por el bus de continua (lado izquierdo) en una tensión senoidal para entregársela a la carga (lado derecho). A continuación, se presenta una figura en la que se muestra un circuito tipo de este tipo de convertidor. Cabe destacar que la topología empleada es la de Puente completo monofásico, esto es así porque está preparado para ceder la energía a una carga aislada que necesite una sola fase.

El convertidor consta principalmente de cuatro transistores IGBT, con sus respectivos diodos, que hace las veces de interruptor. En la siguiente figura se distinguen dos puntos principales marcados de color rojo, "A" y "B" los cuales hacen referencia a los transistores situados en la parte izquierda y derecho el convertidor respectivamente.

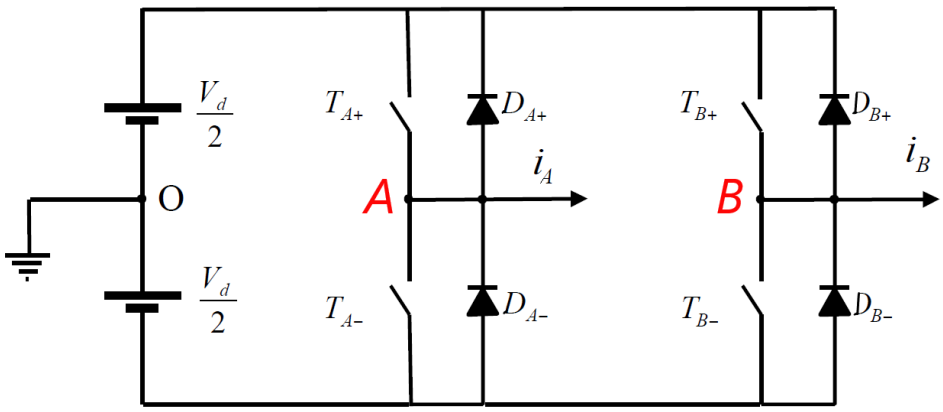


Figura 3-11. Convertidor DC/AC inversor basado en topología de puente completo monofásico.

Los puntos *A* y *B* serán los puntos de conexión (salida del convertidor DC/AC) inversor, positivo y negativo respectivamente, con el siguiente bloque del sistema fotovoltaico aislado. En la **Figura 3-1**, se observa que este bloque siguiente se trata de un filtro compuesto por una bobina y una capacidad. Finalmente, justo después de este filtro, se encuentra la carga aislada a la cual queremos alimentar con este sistema de generación eléctrica fotovoltaica.

En la **Figura 3-12** se va a suponer que una carga genérica "Z" se conecta a los puntos *A* y *B*. Además, en el lado A se va a imponer que solo uno de sus correspondientes interruptores, T_{A+} o T_{A-}, puede estar cerrado en un instante de tiempo determinado. La razón de esta imposición es que en caso de que se cierren los dos interruptores de un lado a la vez, se generaría un cortocircuito. Para el lado B se puede aplicar el mismo razonamiento comentado en este párrafo.

En la siguiente figura, además de mostrar la carga genérica "Z", se va a simplificar la representación de los interruptores, puesto que, teniendo en cuenta la imposición del párrafo anterior, cada par de interruptores (un par a cada lado) va a tener solamente dos posiciones posibles. Por tanto, vamos a redefinir los interruptores *T* de la imagen anterior con los interruptores *S* de la siguiente imagen, que quedarían de la siguiente manera:

- $S_A = 0$ cuando T_{A+} está abierto y T_{A-} cerrado.
- $S_A = 1$ cuando T_{A+} está cerrado y T_{A-} abierto.
- $S_B = 0$ cuando T_{B+} está abierto y T_{B-} cerrado.
- $S_B = 1$ cuando T_{B+} está cerrado y T_{B-} abierto.

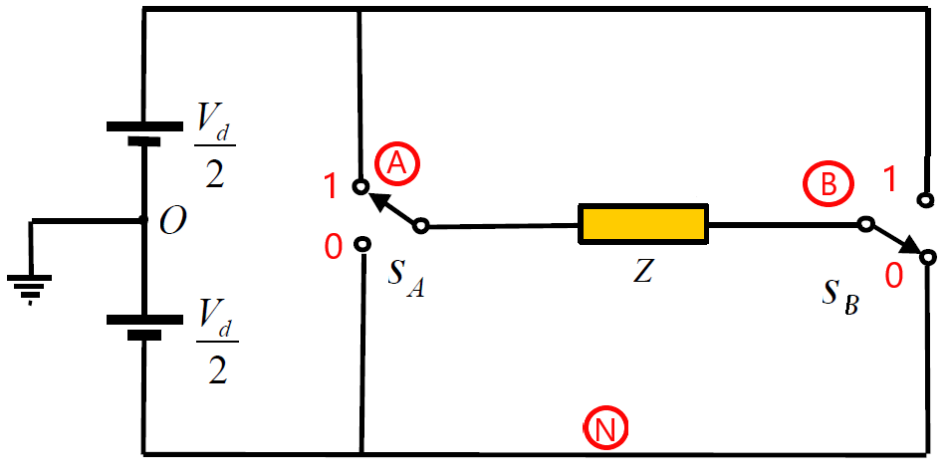


Figura 3-12. Circuito equivalente del convertidor DC/AC inversor basado en topología de puente completo monofásico.

Por tanto, para las distintas posiciones de los interruptores *S* nos quedan cuatro posibles estados, mostrados en la **Tabla 3–1**, los cuales darán su propio valor de tensión en la carga.

Tabla 3–1 Estados de los interruptores de un convertidor inversor monofásico basado en topología de puente completo.

Estado	Interruptor S_A	Interruptor S_B	Tensión en la carga (V_z)
0	0	0	0
1	0	1	$-V_d$
2	1	0	V_d
3	1	1	0

3.4.1 Modulación PWM unipolar para topología de puente completo

La modulación PWM unipolar es la escogida para este convertidor DC/AC inversor. En la **Tabla 3–2** se muestra la comparación de la señal de control y la triangular para generar la señal PWM. La relación entre ambas señales (1ª columna de la tabla) hace que se posicionen los interruptores de una manera concreta (2ª columna de la tabla) que, a su vez, causa una determinada tensión en bornes de la carga (4ª columna en la tabla). [7]

Tabla 3–2 Estados de los interruptores de un convertidor inversor monofásico para la modulación PWM unipolar.

Relación entre la señal de control y la triangular	Posición de los interruptores	Tensión V_{AN}	Tensión V_{BN}	Tensión en la carga ($V_z = V_{AN}-V_{AN}$)
$V_{control}>V_{tri}$	$S_A=1$ y $S_B=0$	V_d	0	V_d
$V_{control}<V_{tri}$	$S_A=0$ y $S_B=1$	0	V_d	$-V_d$
$(-V_{control})>V_{tri}$	$S_A=1$ y $S_B=1$	V_d	V_d	0
$(-V_{control})<V_{tri}$	$S_A=0$ y $S_B=0$	0	0	0

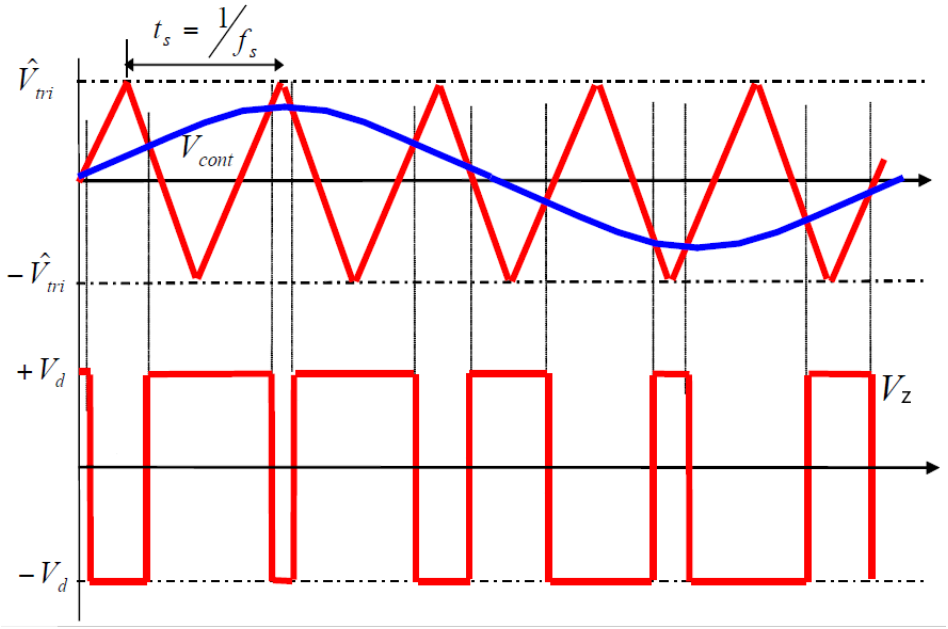


Figura 3-13. Generación de la señal PWM mediante la comparativa entre la señal de control (azul) y la señal triangular.

En la **Figura 3-13** se observa la generación de la señal PWM, que atacará directamente en el terminal de puerta de los transistores IGBT. Donde:

- f_s: frecuencia de modulación (frecuencia de la señal triangular que es constante)
- $\hat{V}_{tri}$: máximo valor de la señal triangular (constante)

4 GUÍA PARA LA CONFIGURACIÓN Y EL USO DEL SOFTWARE DE SIMULACIÓN

Uno de los pasos principales para poder abordar un proyecto que realice la técnica PIL y que, por tanto, incluya cualquier microcontrolador de la familia C2000 del fabricante Texas Instruments es obtener el software con el que trabajaremos. Además, posteriormente debe configurar adecuadamente para que se ajuste a nuestro microcontrolador en concreto. A continuación, veremos una guía detallada paso a paso para poder llevar a cabo la configuración con éxito. Hablaremos de versiones, paquetes, ejemplos, etc.

Es muy recomendable seguir el orden en el que se tiene que instalar cada software, o al menos que el paquete de instalación de Matlab sea el último puesto que, como veremos en una sección posterior, se requerirá tener software ya instalado para poder finalizar la instalación del tercero. En primer lugar, habla del **Code Composer Studio** (CCS), en segundo, de **ControlSUITE**, y, por último, de un **paquete de apoyo para hardware** de Matlab.

Cabe destacar que esta guía no solo sirve para el microcontrolador en concreto con el que se trabaja a lo largo de este proyecto (TMS320F28379D), sino que además es escalable sobre todo para aquellos microcontroladores de la misma familia y sentará las bases para la simulación con la técnica PIL en el entorno software de simulación de Simulink (Matlab).

4.1 Code Composer Studio (CCS)

Code Composer Studio (CCS) es un entorno de desarrollo integrado (EDI) compatible con un catálogo de microcontroladores de Texas Instruments y procesadores embebidos, en este catálogo se incluye por supuesto la familia C2000 del microcontrolador que usaremos durante este trabajo fin de grado. Este software consta de herramientas que se podrán usar para el desarrollo y la depuración de aplicaciones embebidas. Además, entre las características que destacan incluye un optimizador de código, un editor de código fuente, un entorno para construir un proyecto y un depurador.

A continuación, se facilita el enlace de la página oficial de Texas Instruments, que nos lleva directamente la sección del CCS [8] que nos aportará más información acerca de nuestra familia del microprocesador, un apartado de descarga e información adicional tales como guías de usuario, información técnica, canal de la plataforma YouTube, etc. Incluye alguna guía de instalación que puede servir de gran ayuda.

Para este proyecto se ha trabajado con la **versión 6.2.0** del software. Una vez instalada dicha versión, procedemos a la configuración del propio software la cual nos permitirá trabajar de una manera correcta y eficiente durante el proyecto.

Cabe recordar que nuestro modelo es el launchPad LAUNCHXL-F28379D. Es cierto que, aunque la instalación vaya enfocada a un solo dispositivo en concreto, la configuración se puede extrapolar a otros dispositivos. Para esta configuración específica nos podemos ayudar tanto de la página oficial de Texas Instrumentes como de los manuales incluidos en la propia caja del kit de desarrollo.

Abrimos el programa CCS y desde la interfaz principal hacemos clic en ***View > Target Configurations***. En estas próximas imágenes iremos viendo cómo configurar el software para nuestro LaunchPad en concreto.

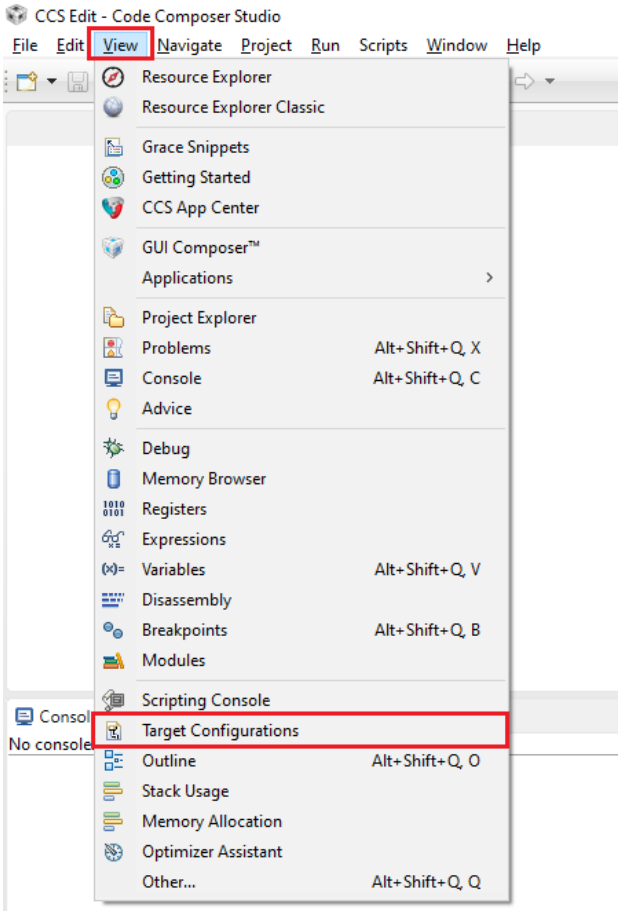


Figura 4-1. "Target Configurations".

Situados en la pestaña ***Target Configurations***, hacemos clic en ***New Target Configurations***.

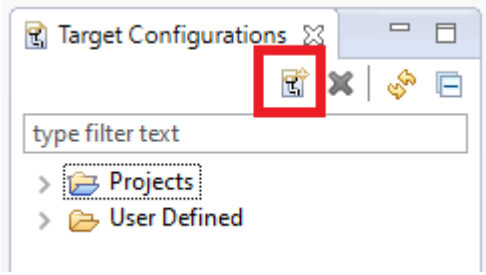


Figura 4-2. "New Target Configuration".

Ahora hay que darle un nuevo nombre a nuestro archivo de configuración y marcar la opción ***Use share location***, esta última es la localización de nuestro archivo de configuración. En caso de que se vaya a usar más de una configuración es recomendable ser específico con el nombre que le damos al nuevo fichero para que, a la hora de trabajar con ellos, sea mucho más intuitivo. En nuestro caso, usamos el nombre ***f28379D_target.ccml***.

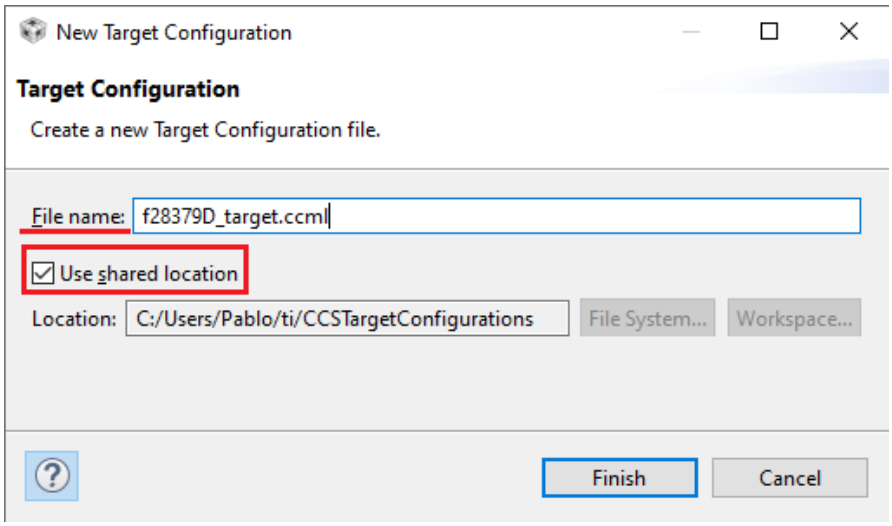


Figura 4-3. "Create a new Target Configuration file".

A continuación, elegimos el tipo de conexión y el modelo del dispositivo y guardamos. Para asegurarnos que estamos indicando el **depurador** correcto nos basta con buscar la guía de usuario de nuestro modelo de hardware en la página web oficial de Texas Instruments. [9]

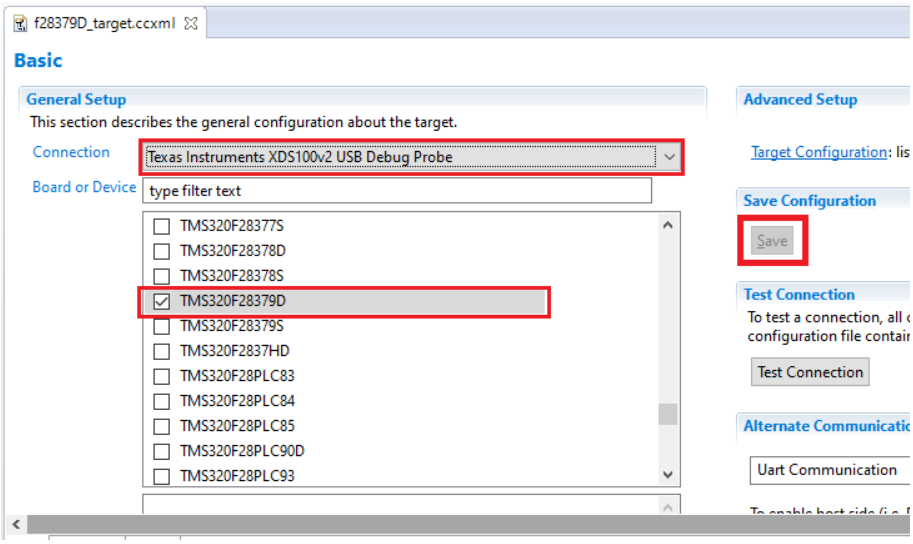


Figura 4-4. "General Setup".

En la pestaña ***Target Configurations*** hacemos clic derecho sobre la configuración que hemos creado anteriormente y elegimos ***Launch Selected Configuration***. Para este paso, es preciso que posteriormente hayamos conectado la placa al ordenador mediante el adaptador que viene dentro de su caja.

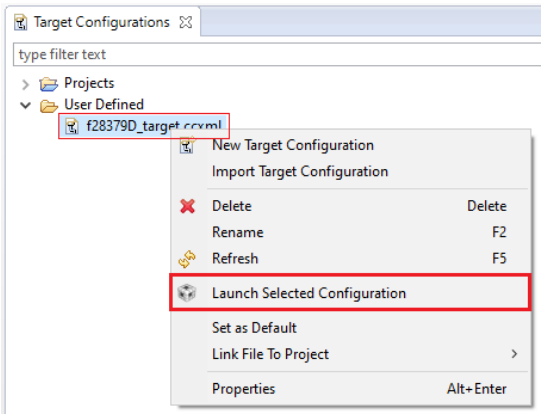


Figura 4-5. “Launch Selected Configuration”.

En la pestaña ***Debug***, hacemos clic en ***CPU1*** y posteriormente hacemos lo propio en conectar el dispositivo.

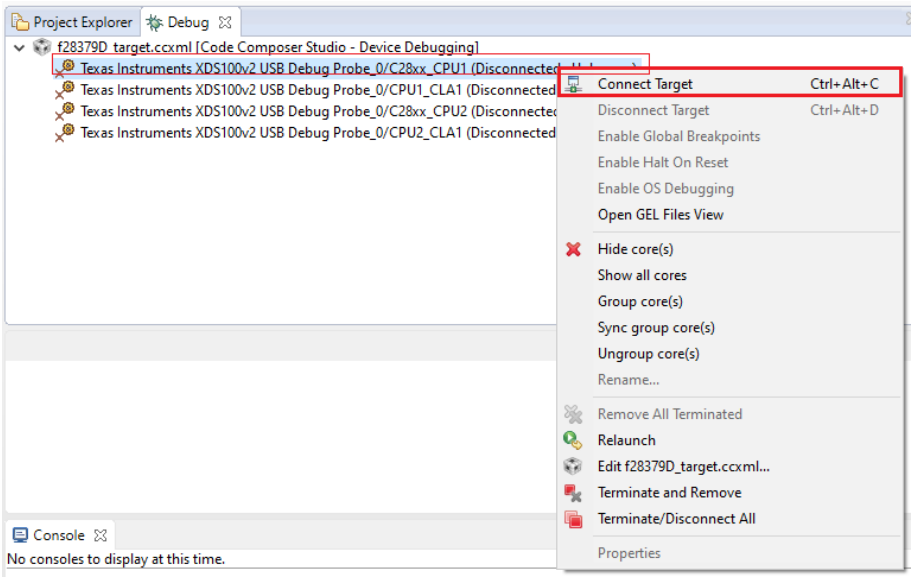


Figura 4-6. “Debug > Connect Target”.

Llegados a este punto, debería aparecer la pestaña **disassembly**, indicando que la conexión entre nuestro microcontrolador y el CCS ha sido un éxito.

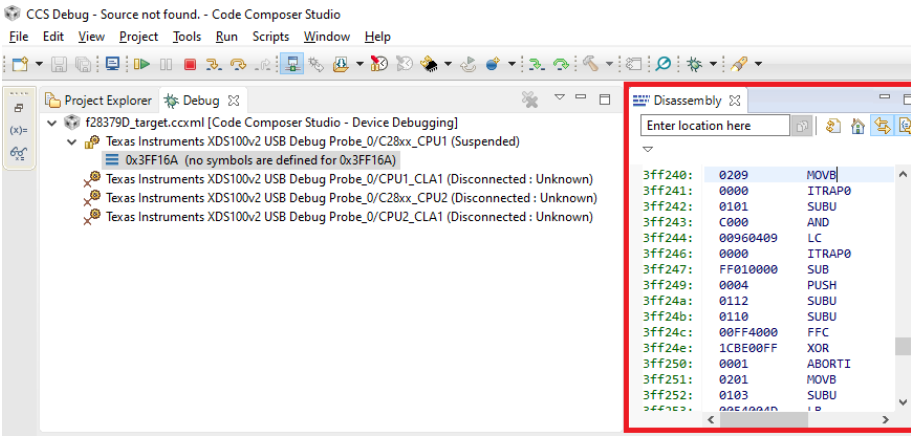
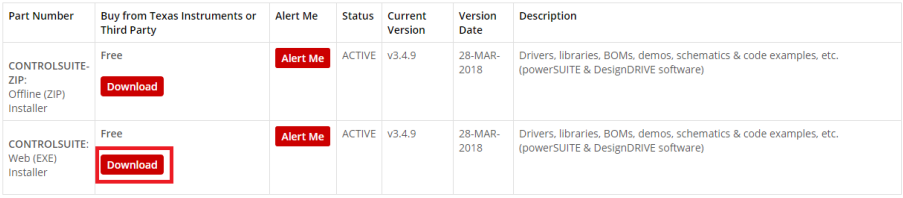


Figura 4-7. “Disassembly”.

4.2 ControlSUITE

Se trata de un software y una herramienta de desarrollo para los microcontroladores de la familia C2000 de Texas Instruments. A través de este software podemos obtener información que nos puede ser de utilidad como manuales, guías de usuario para el dispositivo, documentación del firmware, librerías de apoyo y ejemplos ya creados con el fin de verificar el buen funcionamiento de nuestro microcontrolador.

Realizaremos la descarga del software ControlSUITE desde la página web oficial de Texas Instruments [10]. Por supuesto, será necesario crearse una cuenta o iniciar sesión en la web de Texas Instruments. En el caso de que hubiera algún problema o duda a la hora de hacer la instalación o cómo usar el software en sí, más abajo en la página web proporcionada, aparecen algunas notas sobre la instalación. Además, contamos con una sección, hacia el final de la página, de tópicos que nos podrán ayudar con la iniciación.



Part Number	Buy from Texas Instruments or Third Party	Alert Me	Status	Current Version	Version Date	Description
CONTROLSUITE-ZIP: Offline (ZIP) Installer	Free Download	Alert Me	ACTIVE	v3.4.9	28-MAR-2018	Drivers, libraries, BOMs, demos, schematics & code examples, etc. (powerSUITE & DesignDRIVE software)
CONTROLSUITE: Web (EXE) Installer	Free Download	Alert Me	ACTIVE	v3.4.9	28-MAR-2018	Drivers, libraries, BOMs, demos, schematics & code examples, etc. (powerSUITE & DesignDRIVE software)

Figura 4-8. Descarga del software ControlSUITE.

Una vez instalado el software, podríamos seguir con el proyecto avanzando a la siguiente sección. Sin embargo, recomiendo encarecidamente tener en cuentas las próximas páginas, pues podría ser de interés para desarrollar futuros proyectos en el entorno del software CCS. De algún modo, podremos ser capaces de apreciar la versatilidad de lo que queda de apartado por dos motivos. Por un lado, podremos comprobar el correcto funcionamiento del entorno del CCS y de nuestra placa de desarrollo. Por otro lado, podremos darle un uso prácticos al software ControlSUITE, además de permitirnos intalar el software que nos queda pendiente, el cual se aborda en el próximo capítulo.

Para ello, para completar este apartado, haremos **uso de uno de los ejemplos que nos proporciona ControlSUITE**. A continuación, os mostraremos cómo descargarnos en nuestro dispositivo de Texas Instruments el proyecto que viene dado como ejemplo llamado **Blinky_dc_cpu01**, que encontraremos en la ruta siguiente: **C:\ti\controlSUITE\device_support\F2837xD\v100\F2837xD_examples_Cpu1**.

La principal razón de usar un ejemplo de estas características, parpadeo de un LED, es que podemos realizar la descarga de un proyecto a nuestra placa de desarrollo y comprobar su buen funcionamiento. Todo esto, sin contar con ningún elemento adicional como podrían ser un módulo boost, cables o una fuente de alimentación independiente entre otros. Simplemente necesitaremos un ordenador con el correspondiente CCS y ControlSUITE instalado, nuestra placa de desarrollo y el cable que viene con ella para la alimentación y la comunicación a través del ordenador.

Una vez realizado con éxito el apartado anterior de CSS, abrimos el software, nos situamos en la pestaña **CSS Edit**, hacemos clic derecho sobre **Project** y acto seguido hacemos lo propio sobre **Import CSS Project**.

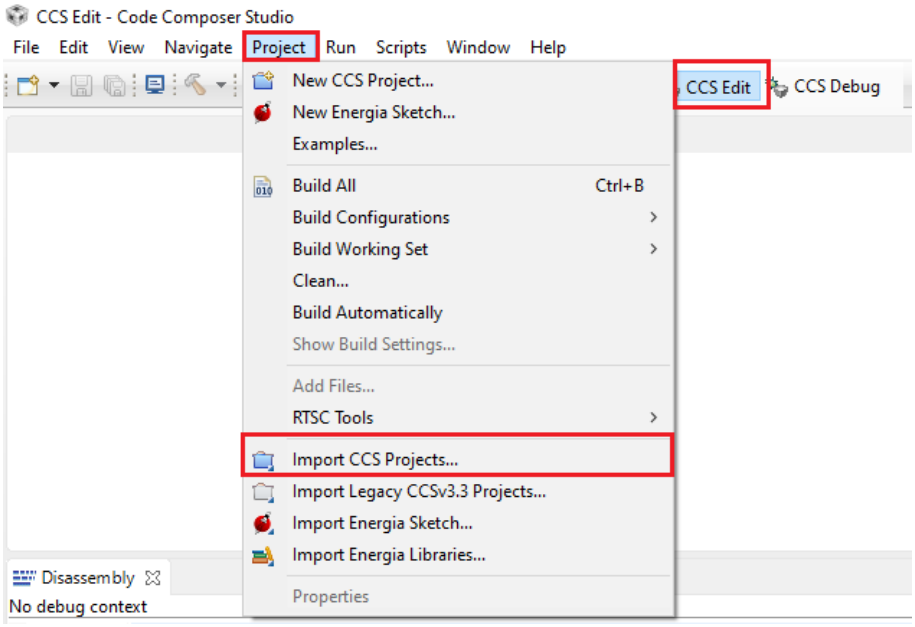


Figura 4-9. Importar proyecto de ControlSUITE a CCS.

En la siguiente ventana emergente que nos aparezca, debemos indicar la siguiente ruta **C:\ti\controlSUITE\device_support\F2837xD\v100\F2837xD_examples_Cpu1**, es decir, la ruta en la que se encuentra el proyecto **Blinky_dc_cpu01** facilitado por la herramienta ControlSUITE.

Conforme vamos avanzando en la ruta en la que se encuentra localizado el ejemplo, podremos observar el amplio catálogo de placas de desarrollo que cubren los proyectos proporcionados por ControlSUITE. Los modelos compatibles para estos proyectos de apoyo son: c2834x, f28m35x, f28m36x, f2802x, f2802x0, f2803x, f2805x, f2806x, f2807x, f2823x, f2833x, **f2837xD**, f2837xS y f28004x.

Una vez en el interior del directorio correspondiente a la placa que nos interese, podremos visualizar distintas versiones tanto del software como de los archivos de los proyectos. Además, cuando escogemos una versión en concreto, tendremos acceso a distintos directorios donde se encuentran archivos comunes, archivos de cabecera, archivos preparados para trabajar con una sola unidad central de procesamiento (CPU) y ejemplos que están diseñados para usar las dos CPUs de las que disponen algunos modelos de los mencionados en el párrafo anterior.

En el ejemplo que nos compete en este caso para el launchPad LAUNCHXL-F28379D usaremos la versión 100 del archivo para esta demostración. Además, el ejemplo está diseñado para trabajar con la CPU1 solamente, de las dos que dispone.

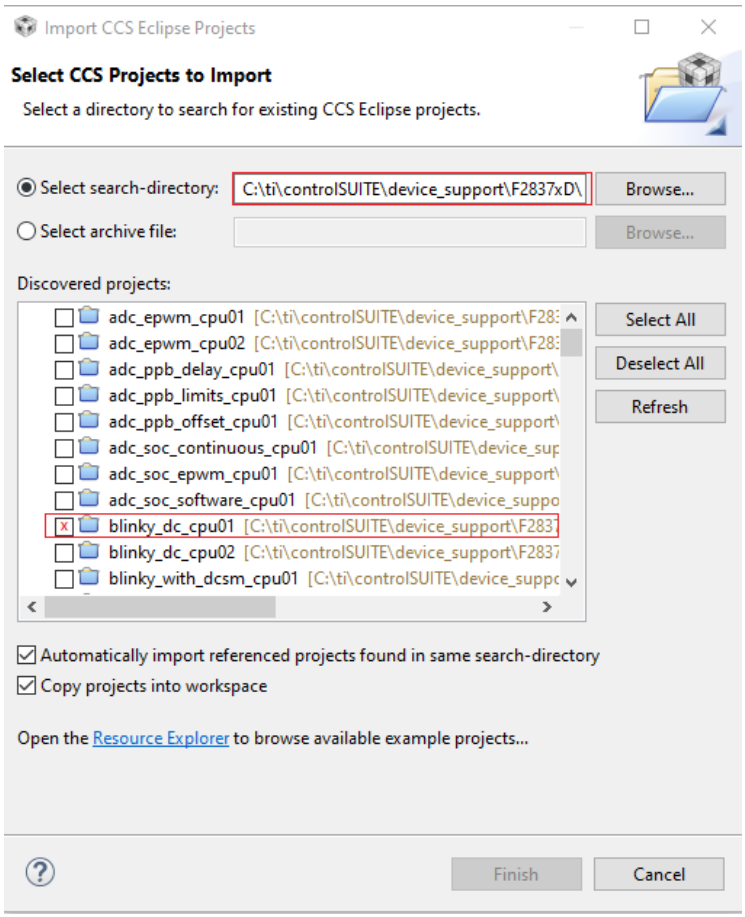


Figura 4-10. Selección de proyecto a importar.

Acto seguido, nos aparecerá el proyecto seleccionado como ejemplo en el explorador de proyectos del CCS. Existe la posibilidad de cargar el proyecto tanto en la memoria RAM como en la memoria flash. Como por defecto aparece en la RAM, mostraré como cambiarlo de la memoria RAM a la memoria flash. Este apartado depende de las preferencias del usuario y de la aplicación en concreto que quiera desarrollar. Al final de esta sección también se comenta otro modo de poder elegir entre memoria ambos tipos de memoria, RAM y flash.

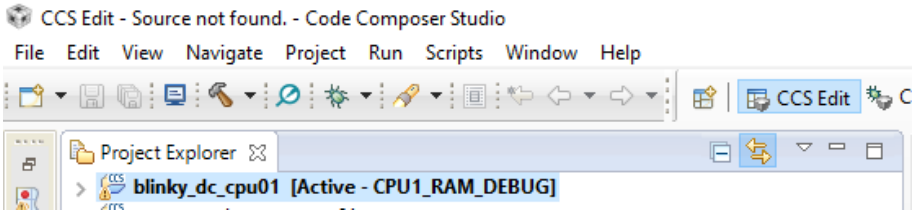


Figura 4-11. Proyectos importados en el explorador de proyectos.

A continuación, hacemos clic derecho sobre el proyecto sobre el proyecto en concreto con el que estamos trabajando, en este caso, **Blinky_dc_cpu01**. Se desplegará una ventana de opciones y seleccionamos ***Build Configurations > Set Active > CPU1_FLASH_DEBUG***. Tras está acción, ya estará configurado para que utilice la memoria flash en vez de la RAM.

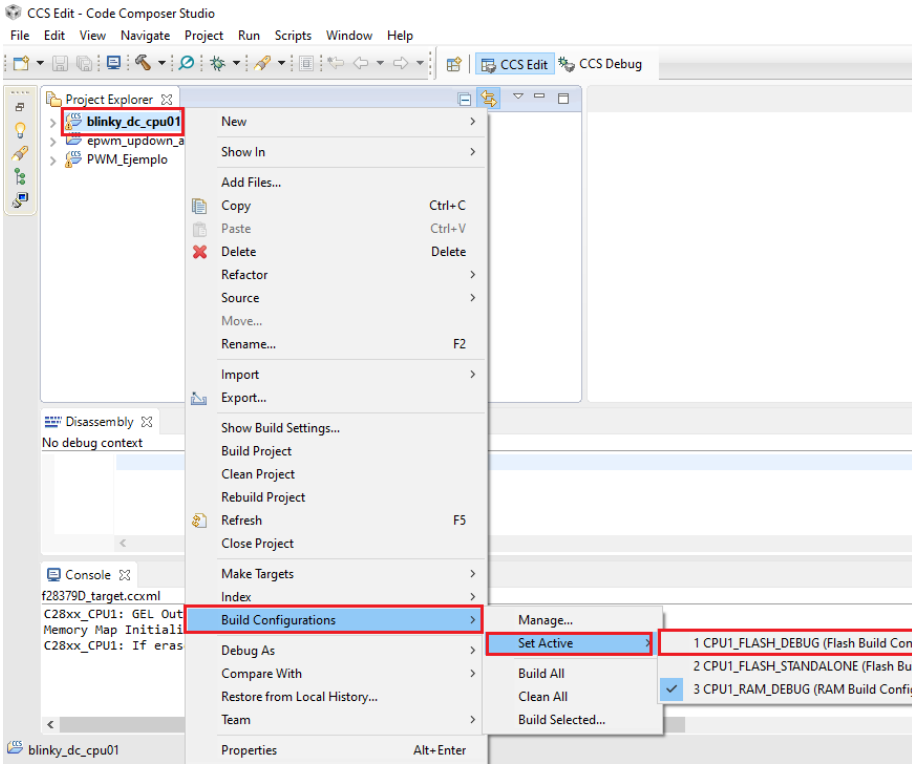


Figura 4-12. Selección de memoria flash.

En el siguiente paso, vincularemos nuestro fichero de configuración **f28379D_target.ccml** a nuestro proyecto de ejemplo **Blinky_dc_cpu01.**

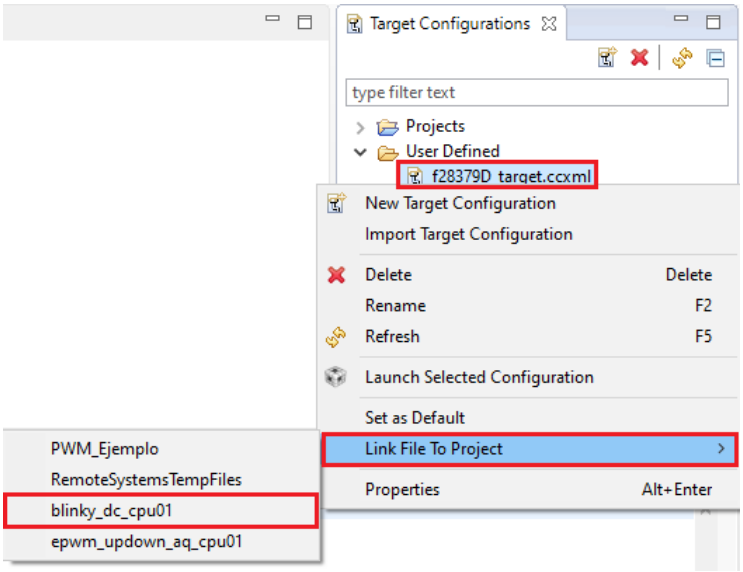


Figura 4-13. Vinculación del fichero de un configuración con un proyecto.

Ahora veremos en la **Figura 4-14** y **Figura 4-15** cómo nos ocupamos de **construir** y, cuando termine de completarse esta primera acción, **depurar** el proyecto respectivamente.

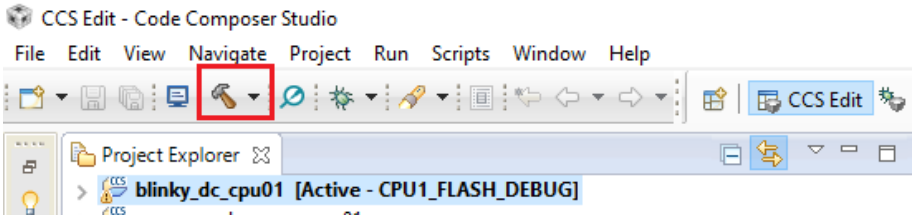


Figura 4-14. Construir proyecto.

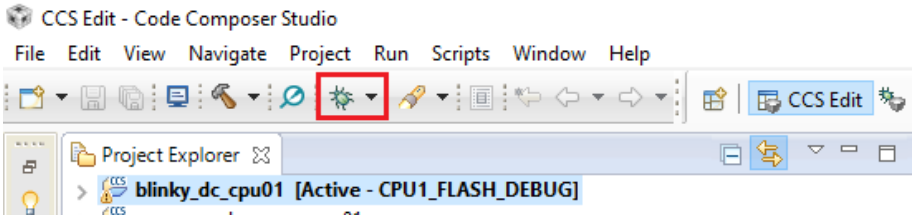


Figura 4-15. Depurar proyecto.

Para finalizar la puesta en marcha de este ejemplo, nos queda un paso más. Para ello, debemos situarnos en la pestaña ***Debug*** y seleccionamos el depurador para la ***CPU1*** y posteriormente hacer clic en ***Connect Target***. Una vez seguidos los pasos para este ejemplo, deberíamos poder observar parpadear el LED de color azul de nuestro microcontrolador.

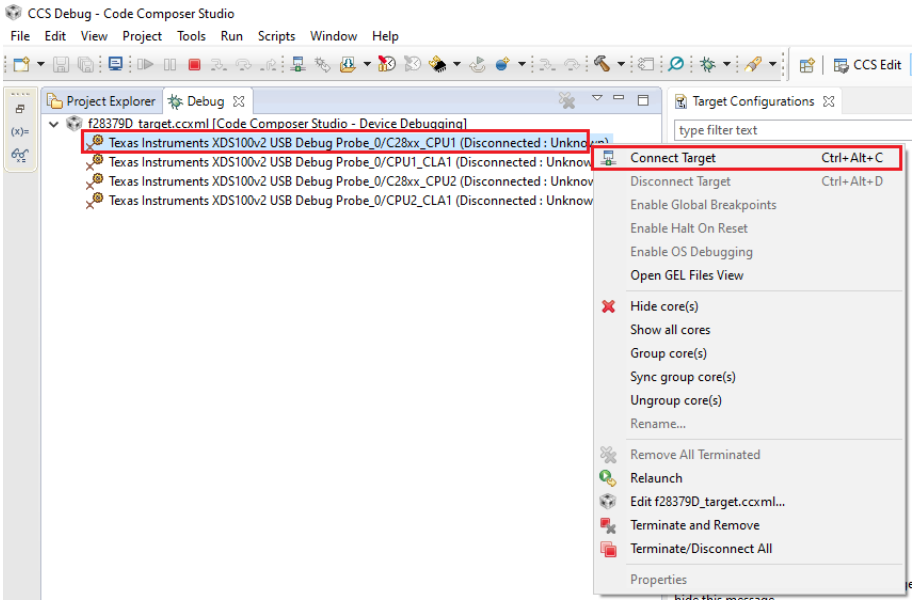


Figura 4-16. Establecimiento de la conexión con el hardware.

Cabe destacar la importancia que cobra la versión del compilador que estemos usando. Pues podría dar lugar a incompatibilidades y por tanto no podríamos realizar, en este caso, el ejemplo que hemos comentado. Por ello vamos a mostrar dónde encontrar esta configuración para poder adaptar la versión al proyecto en concreto con el que estemos trabajando. Para acceder a esta configuración, debemos situarnos en el ***explorador de proyectos***, hacer clic derecho sobre nuestro ejemplo y posteriormente en ***propiedades***.

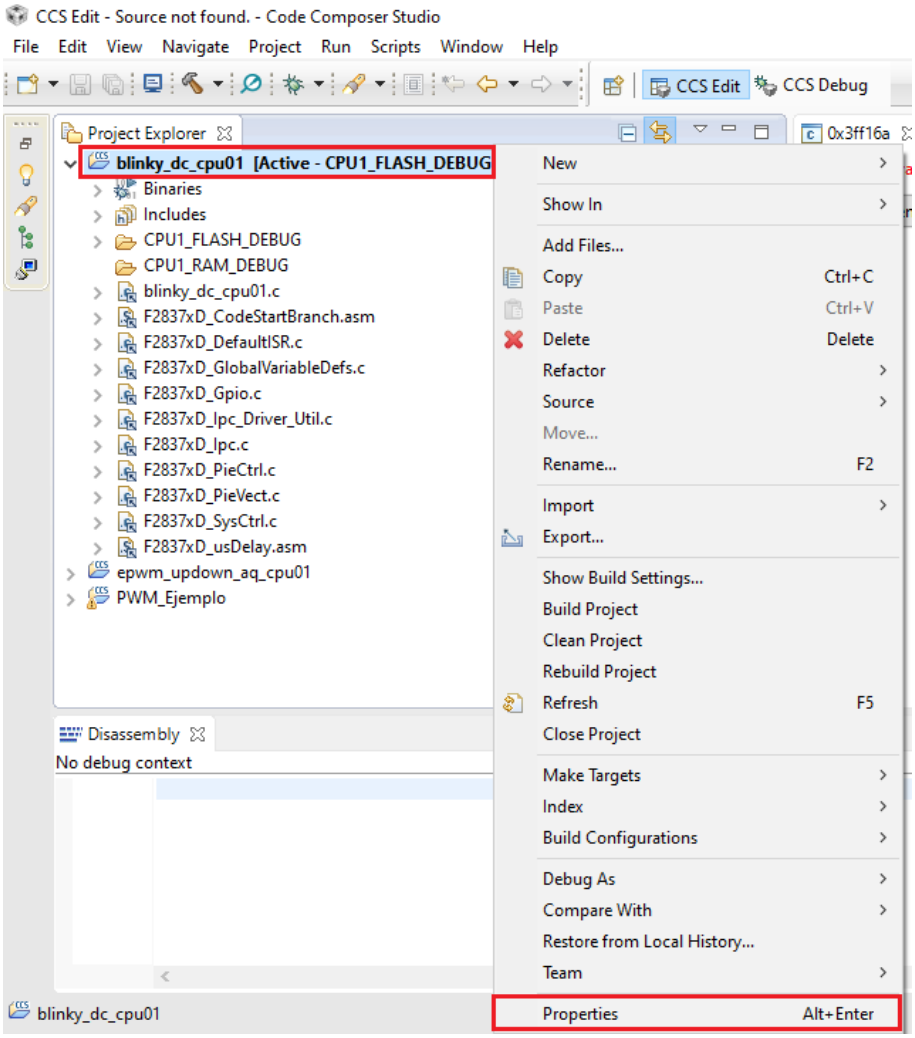


Figura 4-17. Propiedades del proyecto.

Una vez que se abre la ventana de propiedades, en la pestaña general, seremos capaces de seleccionar el modelo exacto de nuestra placa dentro de la familia C2000. Por supuesto, la versión del compilador. Y otros campos como la elección de la memoria flash o RAM y personalizar el fichero enlazador (linker).

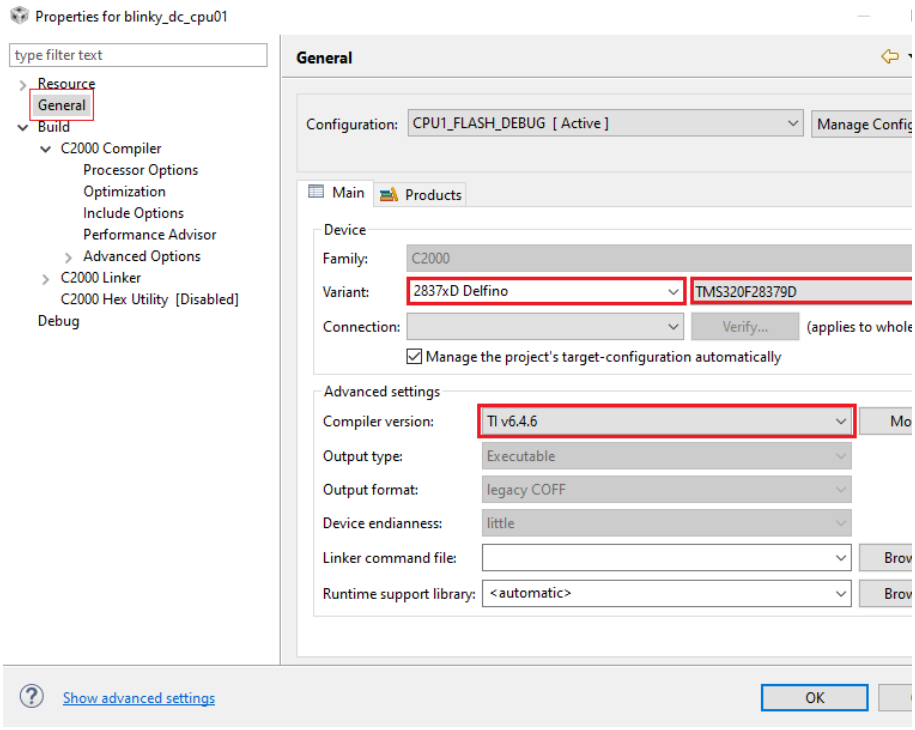


Figura 4-18. Propiedades del proyecto > General.

Varios de los ejemplos proporcionados por la herramienta ControlSUITE están pensado para trabajar con una versión del compilador un poco desfasada, a pesar de contar con actualizaciones recientes. Entonces, decidimos utilizar una versión algo más antigua del compilador, la **versión 6.4.6**. Porque, por un lado, se ajuste el número de la versión y, por otro lado, el número de la versión ha sido directamente recomendado por los estudiantes Eduardo y Guillermo, los cuales han estado trabajando estrechamente con el departamento de electrónica de la universidad con placas de desarrollo de una familia muy cercanas a la f28379D.

Ambos, además, tenían como tutor al profesor Sergio, autor del documento **PIL Tutorial** [11] en el cual nos hemos basado para configurar el software CCS y Matlab a medida para nuestro modelo de placa de desarrollo en concreto.

4.3 Hardware Support Package (Matlab)

Matlab va a ser nuestro software principal de trabajo y, dentro de este, el entorno de programación visual Simulink. Obligatoriamente, necesitaremos algún paquete de apoyo adicional para los microcontroladores de la familia C2000 del fabricante Texas Instruments. Por otro lado, el codificador embebido que nos aporta este paquete de apoyo, genera código C optimizado para nuestro procesador digital de señal (del inglés, DSP) de la serie C2000 y posteriormente descargarlo en nuestra placa de desarrollo del fabricante Texas Instruments.

En este caso vamos a contar con la versión de **Matlab R2018a**. Para proceder a descargar los paquetes de ayuda para microprocesadores de la serie C2000 de Texas Instruments debemos abrir el entorno e programación Matlab, y dirigirnos a la pestaña ***HOME*** en la sección ***ENVIRONMENT***; desde allí accederemos a ***Add-Ons > Get Hardware Support Packages***. Dentro de ***Add-Ons*** se incluye una amplia variedad de recursos tales como productos, aplicaciones, paquetes de apoyo, etc.

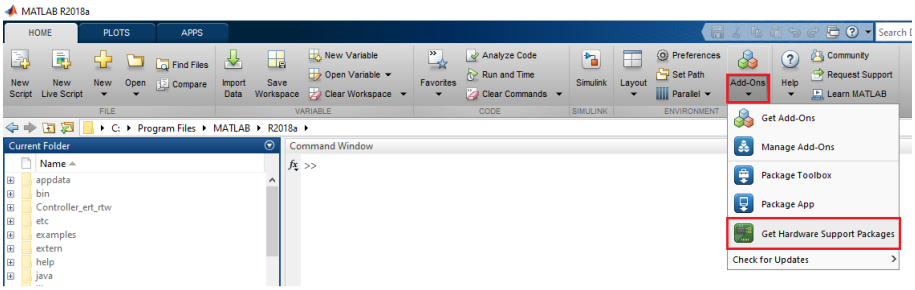


Figura 4-19. Buscando paquete de apoyo hardware en Matlab.

A continuación, buscamos el paquete de apoyo que encaja con nuestro microprocesador y hacemos clic ***Embedded Coder Support Package for Texas Instruments C2000 Processors***, luego hacemos clic en ***Install*** y una vez instalado seleccionamos ***Setup Now*** para pasar a los pasos necesarios para la configuración del paquete.

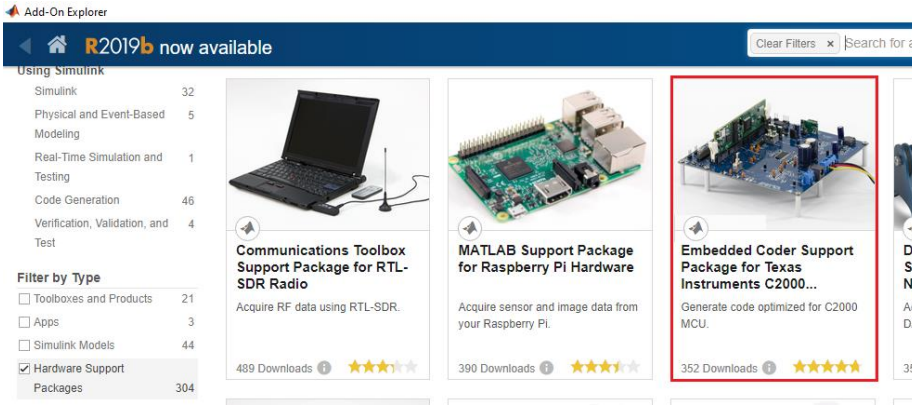


Figura 4-20. Obteniendo paquete de apoyo hardware en Matlab.

A continuación, para la configuración completa del paquete de apoyo hardware que acabamos de descargar e instalar desde Matlab, van a aparecer ventanas emergentes. Una buena manera para explicar dichas ventanas (de qué trata y qué se hace en cada una) es mediante una lista, donde cada punto de la lista se corresponde con una nueva ventana emergente. En total, son siete las ventanas que deben aparecer. Cada vez que se haya completado una ventana, haremos clic en ***Next*** para pasar a la siguiente y, por tanto, al siguiente punto de la lista:

- En esta primera ventana seleccionamos la familia de microprocesadores con la cual queremos trabajar, en nuestro caso es ***Texas Instruments C2000 (Embedded Coder)***.
- En la segunda, elegimos la versión del CCS y para este proyecto nos hemos decantado por la opción ***Texas Instruments Code Composer Studio v6 C2000 (Embedded Coder)***
- En este tercer paso, nos requerirá que instalemos software de terceras partes como son el ***CCS*** y el ***ControlSUIT***, ambos de Texas Instruments. Si hemos seguido el orden de este capítulo ya deberíamos tenerlos descargados por lo tanto solo habría que avanzar a la siguiente ventana. De no ser así, siguiendo el proyecto en las dos secciones anteriores podemos obtener de dónde o cómo configurar el software en cuestión.
- Cuarta ventana, ahora toca validar que la instalación del software CCS está bien realizada indicando la ruta de instalación, ***C:\Program Files\ccsv6***
- En la quinta, validamos que la instalación del software ControlSUITE está bien realizada indicando la ruta de instalación, ***C:\ti\controlSUITE***
- En la sexta, confirmamos que el directorio de instalación y el número de versión del software de terceras partes listados en la propia ventana son correctos.
- En este punto hemos finalizado los pasos de configuración del paquete de apoyo hardware. Tenemos la opción de ver algunos ejemplos que nos proporciona este paquete marcando la pestaña que hay en esta ventana actual o directamente podemos hacer clic en ***Finish*** para finalizar por completo este proceso de configuración.

4.3.1 Ejemplo proporcionado por Matlab para la simulación con la técnica PIL

Parece que ya disponemos del software imprescindible para la realización de una simulación en Simulink con la técnica PIL. Sin embargo, queda uno de los aspectos clave para el desarrollo de este proyecto. Se trata de la configuración obligatoria del entorno de Simulink sin la cual no vamos a ser capaces de realizar la simulación de nuestro modelo definitivo con éxito. Una vez completada esta configuración, seremos capaces no solo de comprobar el correcto funcionamiento de los paquetes instalados de la placa, sino que además nos introduciremos de lleno en uno de los conceptos clave de este proyecto, la simulación con la técnica PIL.

Como ya hemos mencionado anteriormente, disponemos de la placa de desarrollo **launchPad LAUNCHXL-F28379D** de la familia **C2000** de Texas Instruments. Es por ello, que para poder probar la técnica PIL en el software Simulink haremos **uso de un ejemplo proporcionado por Matlab**. Para ello necesitaremos abrir el programa Matlab y situarnos en la ventana de comandos, acto seguido, introduciremos por teclado el nombre de ejemplo **c2000_pil_block** y presionamos la tecla “Intro”.

Acto seguido, se nos abrirá automáticamente una ventada del entorno de Simulink con el ejemplo mencionado. También tendremos accesible el ejemplo a navegando hasta el directorio donde lo instalamos, la ruta es: **C:\ProgramData\MATLAB\SupportPackages\R2018a\toolbox\target\supportpackages\tic2000\tic2000examples**. Esta ruta solo será alcanzable una vez sean instalados los paquetes de apoyo de Matlab para nuestra familia de placa de desarrollo.

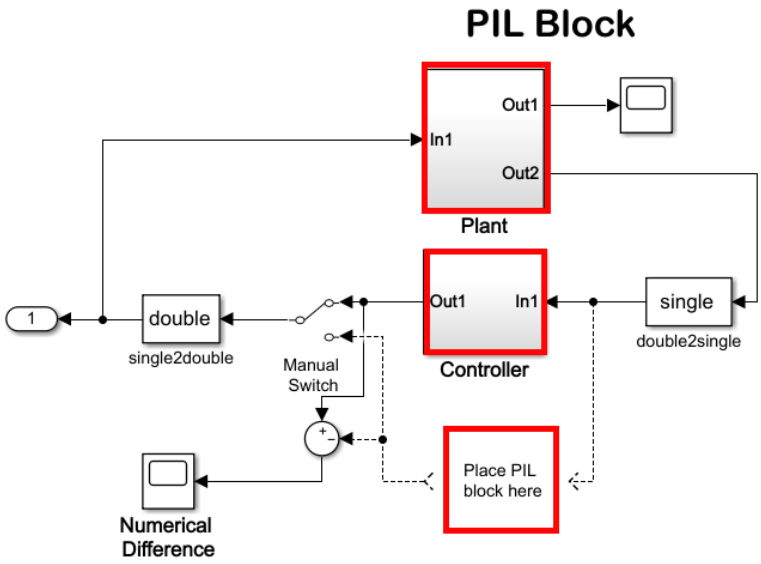


Figura 4-21. Esquema de bloques de la técnica PIL en Simulink.

El ejemplo consta de tres partes bien diferenciadas, son mencionadas en orden descendientes en la imagen anterior. En primer lugar, observamos el bloque planta; este nombre es comúnmente utilizado para referirnos al modelo matemático que queremos controlar. En este caso, la planta está compuesta por un integrador y es cierto que quizás no es del todo necesario saber este dato para comprobar el buen funcionamiento del bloque PIL, sin embargo, nos ayudará a comprender el comportamiento de la curva dada por la salida 1 de dicho bloque.

En segundo lugar, nos encontramos con el bloque del controlador. Como su propio nombre indica, se encargará de controlarla señal deseada que, en este caso, es la señal de salida del bloque planta. Este bloque está compuesto por una serie de bloques de ganancia y unidades de retrasos (muestrea y mantiene con una muestra de periodo de retraso).

En tercer lugar, podemos ver el hueco donde deberíamos colocar el bloque PIL que hará las veces de controlador con la diferencia de que las operaciones que realizará el bloque PIL no serán simuladas en el entorno Simulink, sino que serán directamente ejecutadas en la placa de desarrollo de Texas Instruments. La obtención del bloque PIL la comentaremos unos párrafos más adelante.

También debemos tener en cuenta que a la salida el bloque controlador y del PIL nos encontraremos un conmutador manual que nos permitirá ver la tendencia de sus señales de salida. Además, se hace la diferencia de dichas señales y deberá ser cero, confirmando que el bloque PIL es capaz de replicar el comportamiento del controlador.

Como elemento clave a destacar, queda por clarificar los bloques de conversión del tipo de dato que se observa la entrada y salida de ambos bloques encargado de realizar la función de control. Es obligatorio convertir los datos a la entrada de la parte de control a un tipo **single** porque el tipo **double** no es soportado en la ejecución con el bloque PIL. Es más, si probamos con un tipo de dato equivocado, no advertirá que el tamaño del dato usado por el ordenador (8 bytes) y la placa de desarrollo (4 bytes) no casan. Obviamente, necesitaremos un bloque de conversión a la salida de la parte de control para que la planta pueda funcionar con otro tipo de dato si así se quisiese o necesitase.

Una vez realizada la introducción pertinente, vamos a pasar a comentar la configuración necesaria para ajustarla a nuestro caso en particular, el **lanunchPad LAUNCHXL-F28379D**. Para una mejor visión global de los cambios que se deben realizar vamos a dividir la preparación del ejemplo en dos partes, **por un lado**, la configuración de algunos de los parámetros a los cuales podremos acceder desde la barra de herramienta situada en la parte superior del software. **Por otro lado**, se hará la pertinente modificación de los bloques del escenario, posteriormente se descargará el código en la placa de desarrollo y luego se ejecutará el código de la parte de control del ejemplo en la placa.

1. Configuración en la ventana de los parámetros del modelo. Podemos acceder a dicha ventana haciendo clic en el icono con forma de engranaje que se encuentra marcado en rojo en la siguiente imagen.

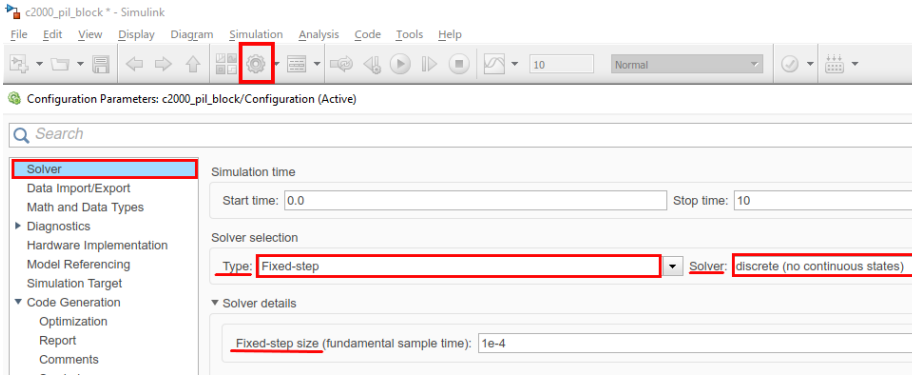


Figura 4-22. Ventana para la configuración de los parámetros del modelo: "Solver".

a. Una vez abierta la ventada de parámetros, en la pestaña ***Solver*** debemos seleccionar ***Solver selection*** como ***Fixed-step*** y ***discrete (no continuous states)***. Además, tenemos que fijar el Tiempo de muestreo fundamental con un valor adecuado, para este ejemplo 1e-4.

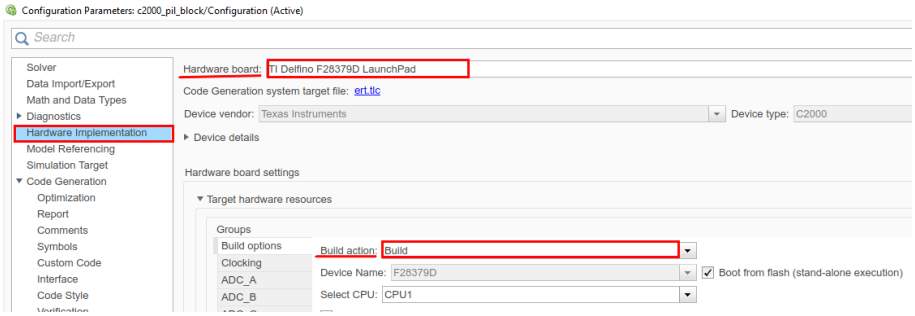


Figura 4-23. Ventana para la configuración de los parámetros del modelo: "Hardware Implementation".

b. En la pestaña ***Diagnostics > Hardware Implementation*** nos aparece la opción de seleccionar nuestra placa ***TI Delfino F28379D LaunchPad***. También, en la sección ***Hardware board settings > Target hardware resources > Build options*** debemos marcar ***Build action*** como ***Build***.

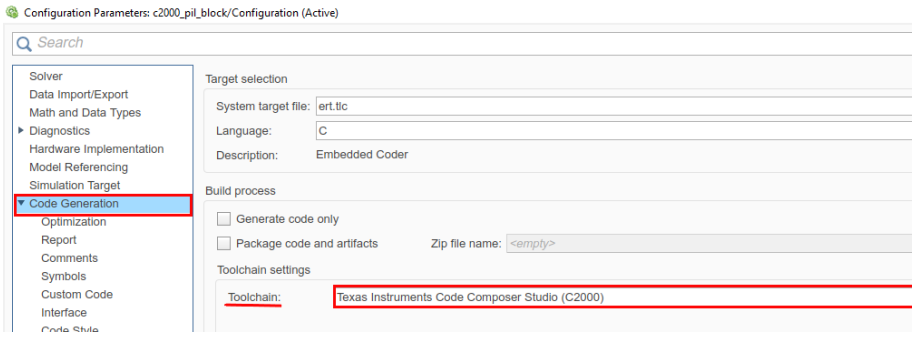


Figura 4-24. Ventana para la configuración de los parámetros del modelo: "Code Generation".

c. En la pestaña ***Code Generation*** en el apartado ***Toolchain settings*** marcamos ***Toolchain*** como ***Texas Instruments Code Composer Studio (C2000)***.

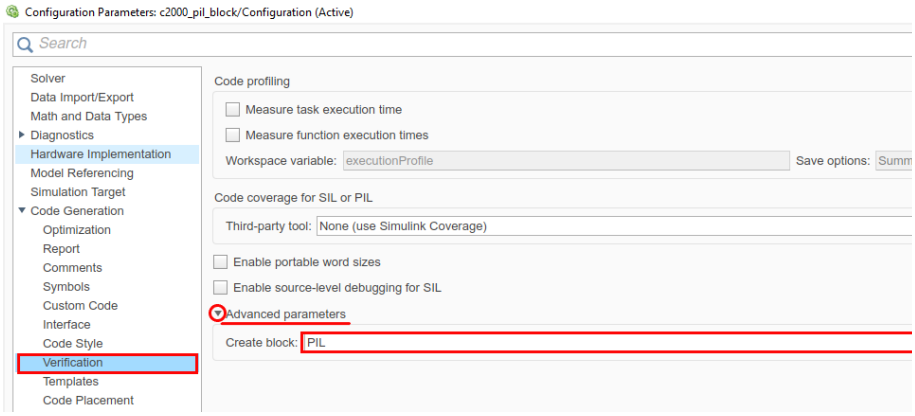


Figura 4-25. Ventana para la configuración de los parámetros del modelo: "Code Generation > Verification".

d. En la pestaña ***Code Generation > Verification*** en la sección ***Advance parameters*** seleccionamos ***create block*** como ***PIL***.

2. El resto de modificaciones en el escenario, como sustitución de algún bloque y modificación del tipo de dato. Es recomendable tener en mente la Figura 4-21, pues nos permitirá situarnos gráficamente en el ejemplo en los próximos párrafos e imágenes.

a. A continuación, entramos en el bloque planta, y sustituimos el integrador original por un integrador de tiempo discreto. Es decir, el escenario debe tener un carácter discreto, para que tenga coherencia con la configuración de los parámetros del modelo que hemos seleccionado anteriormente.

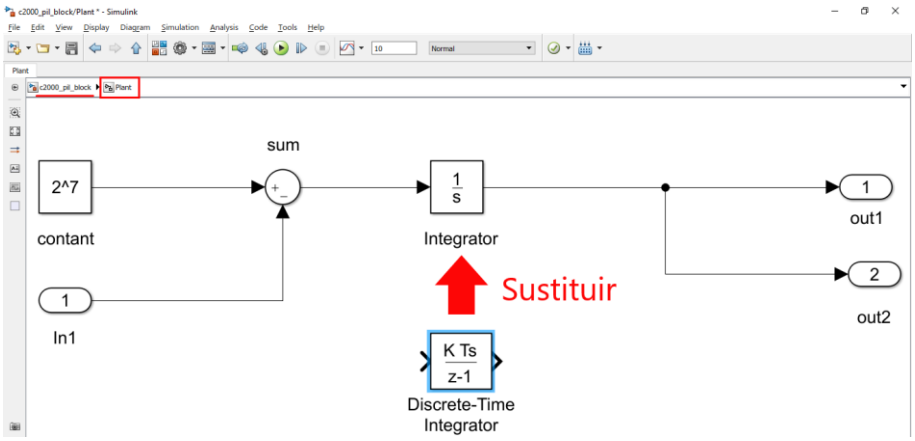


Figura 4-26. Sustitución de bloque integrador en tiempo continuo por discreto.

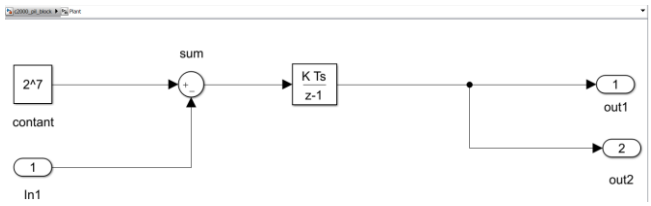


Figura 4-27. Bloque integrador en tiempo discreto.

b. Necesidad de convertir el tipo de dato a ***single*** a la entrada del controlador, pues solo de esta manera podremos implementar la técnica PIL. Justo después del controlador, es necesario convertir los datos al tipo de dato con los que vayamos a trabajar en la planta. En la Figura 4-21 se pueden observar ambos bloques de conversión de tipo de dato, antes y después del bloque controlador. En este ejemplo ya viene colocado el bloque de conversión por defecto. Además, el directorio de trabajo de Matlab debe ser uno distinto a la ruta donde se instaló.

c. Una vez en no situamos de vuelta en el escenario principal del ejemplo, hacemos clic derecho sobre el bloque ***Controller***, elegir ***C/C++ Code > Deploy this Subsystem to Hardware***. Después, clic en ***Build***.

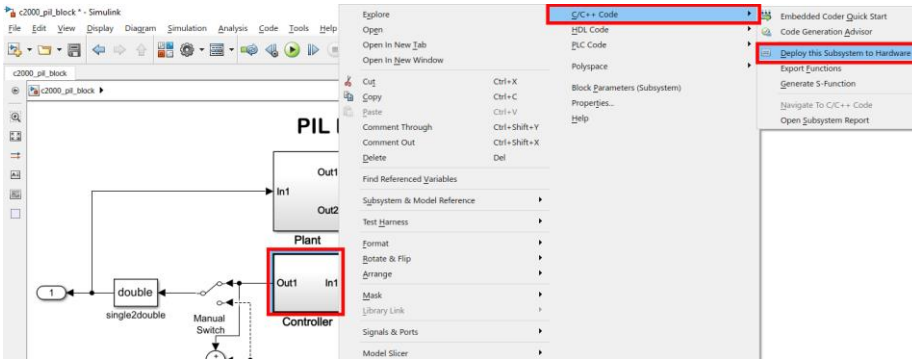


Figura 4-28. Implementar subsistema seleccionado en el hardware.

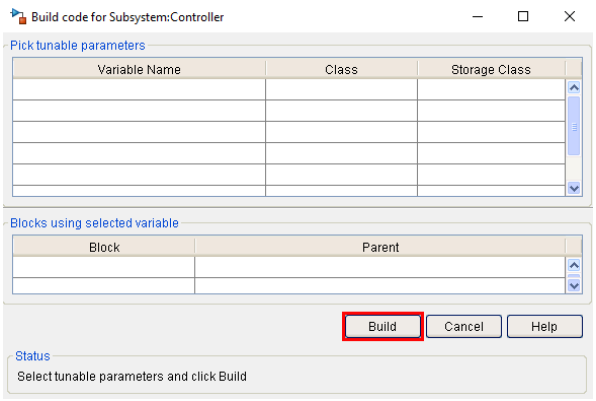


Figura 4-29. Implementar subsistema seleccionado en el hardware.

d. Aparecerá un bloque PIL, en una ventana emergente, el cual copiaremos y posteriormente lo pegaremos en nuestro escenario en paralelo a nuestro bloque de control. Justo en ***Place PIL block here***, el hueco habilitado para dicho bloque en el ejemplo.

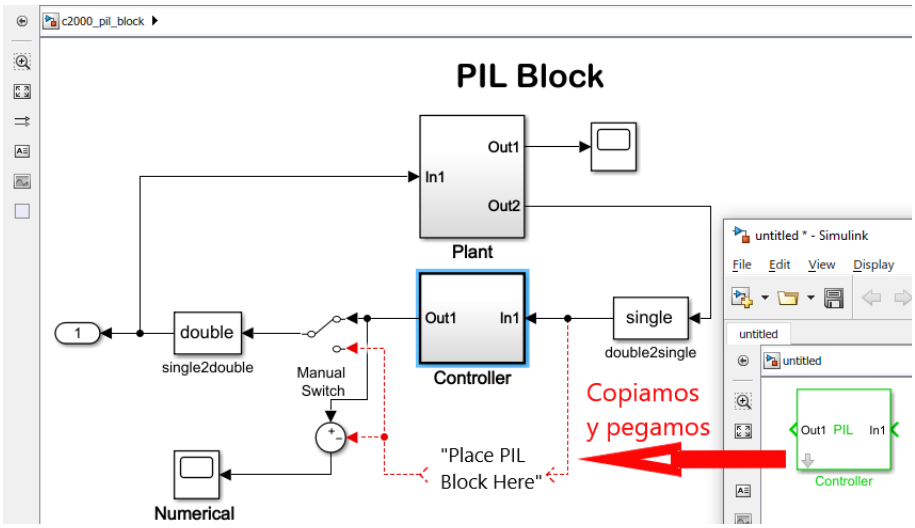


Figura 4-30. Añadimos el bloque PIL a nuestro escenario.

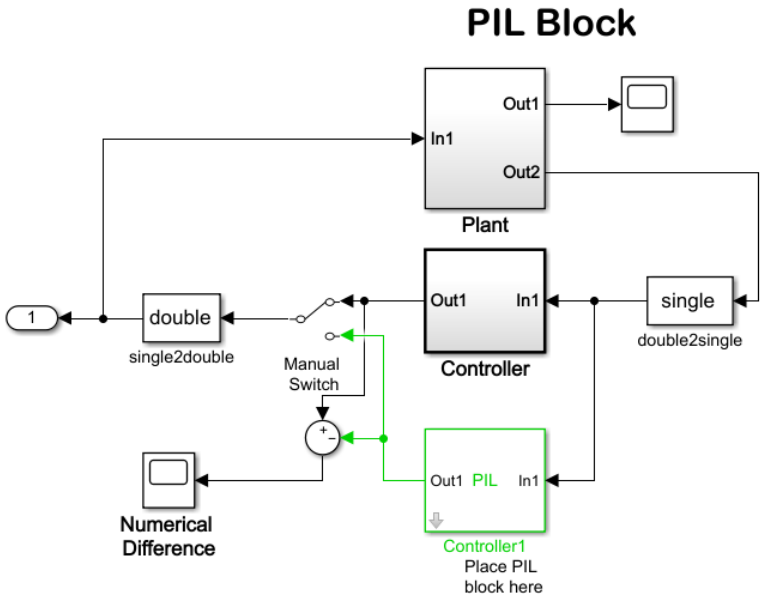


Figura 4-31. Bloque PIL añadido en paralelo al controlador.

Así debería quedar el escenario al final de estos pasos. Como último detalle, debemos configurar algunos parámetros que permitan la ejecución PIL sobre comunicación serie. En concreto, toca hablar de tres parámetros como son el puerto COM, el "baud rate", es decir, seleccionar el número de bits por segundo que queremos en la comunicación PIL, y habilitar PIL sobre comunicación serie. Para ello, debemos introducir los siguientes comandos en la ventana de comandos de Matlab.

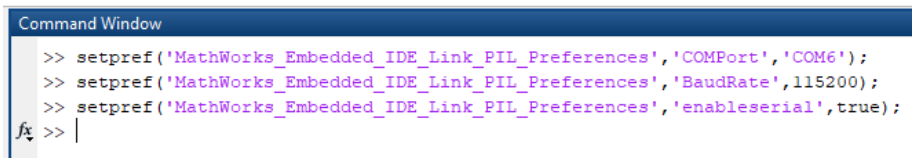


Figura 4-32. Comandos para ejecutar PIL sobre comunicación serie.

Recomendamos que esta clase de configuraciones se unifiquen en un solo archivo (script) para que, con el simple hecho de ejecutar dicho archivo, se actualicen nuestras preferencias para la simulación en concreto que vayamos a realizar. Esto nos permitirá ahorrar tiempo a la hora de establecer las configuraciones y tener menos fallos pues, una vez que confirmamos que una configuración está bien establecida, no tendremos que volver a modificarla. A continuación, las preferencias:

- setpref ('MathWorks_Embedded_IDE_Link_PIL_Preferences', 'COMPort', 'COM6');
- setpref ('MathWorks_Embedded_IDE_Link_PIL_Preferences', 'BaudRate', 115200);
- setpref ('MathWorks_Embedded_IDE_Link_PIL_Preferences', 'enableserial', true);

Con respecto a la primera línea de preferencias, necesitamos conocer el puerto COM para poder reflejarlo como parámetro. Es decir, debemos saber el nombre del puerto al cual conectamos la placa de desarrollo. Para ello, hacemos clic con el botón derecho del ratón sobre el icono de Windows (edición 10) y a continuación hacemos clic en ***Administrador de dispositivos***. Una vez aquí, hacemos clic sobre la sección ***Puertos (COM y LPT)*** y entonces seremos capaces de reconocer en qué puerto serie USB estamos conectados.

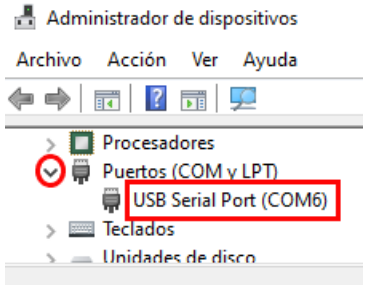


Figura 4-33. Puerto en el que conectamos el hardware.

La segunda línea de comando será para fijar el número de bits por segundos que se va a transferir en la comunicación PIL. La tercera línea, habilitaré PIL sobre la comunicación serie.
Una vez aclaradas estos últimos detalles, procedemos a ejecutar el ejemplo proporcionado por Matlab **c2000_pil_block**. Para ello tendremos que hacer clic en el botón verde circular que aparecerá en la ventana del ejemplo de Simulink.

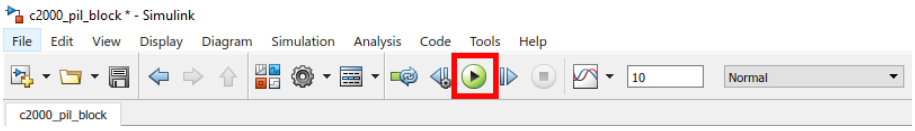


Figura 4-34. Ejecutar la simulación en Simulink.

Después de unos segundos, el escenario habrá simulado completamente y estaremos listo para sacar conclusiones y ver la tendencia de las curvas. Además, merece la pena prestar atención a las diferentes curvas de resultados que obtenemos al usar el bloque controlador o el bloque PIL. Lo que esperamos es que la tendencia de la señal de salida del bloque planta sea la más parecida posible tanto usando el bloque controlador como el bloque PIL.

En la **Figura 4-35** la señal de salida del bloque planta habiendo realimentado con el bloque controlador (curva roja) y el bloque PIL (curva azul) al mismo tiempo. Para ellos hemos tenido que modificar el escenario original un poco. Hemos duplicado el bloque planta, por lo que nos quedarían dos bloques de esta naturaleza. El siguiente paso es conectar a unos de los bloques planta la salida del controlador y al otro bloque planta la salida del bloque PIL. De esta manera seremos capaces de comparar ambas curvas. Además, añadiremos un bloque adicional que unifique las salidas de los bloques planta en una sola gráfica.

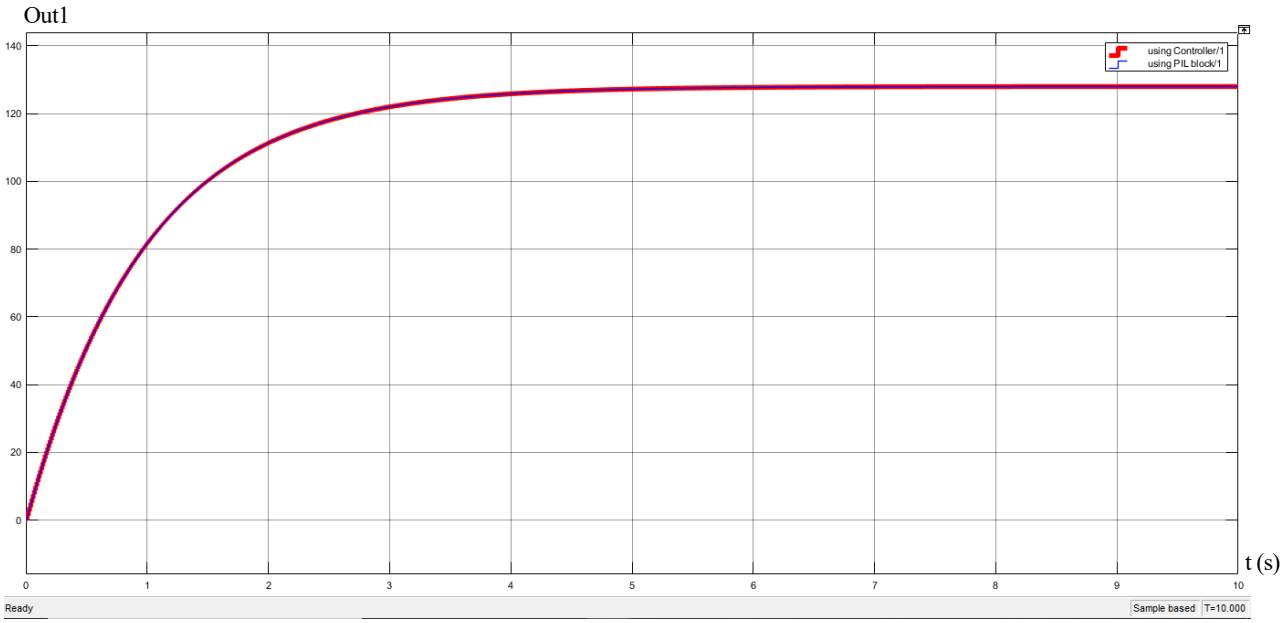


Figura 4-35. Variable de salida de la planta (Out1) que aparece en el modelo Simulink de la figura 4-31: usando el controlador (roja) y usando el bloque PIL (azul).

Podemos observar que es muy complicado diferenciar ambas gráficas pues gracias a la fidelidad de la técnica PIL, ambas curvas quedan superpuestas. Demostrando de esta manera que la técnica PIL replica con bastante exactitud el comportamiento del bloque controlador. Podemos observar la gráfica diferencia que encontramos en el ejemplo para ver la diferencia entre la respuesta dada por el bloque controlador y el bloque PIL. Podemos observar que la diferencia es cero con bastante precisión.

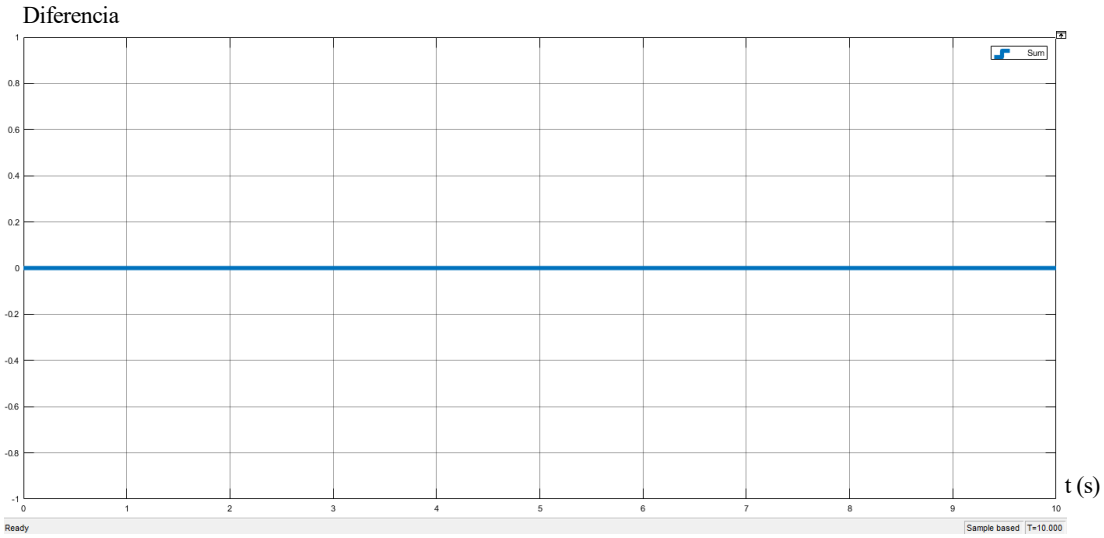


Figura 4-36. Señal "Diferencia" entre la salida del bloque controlador y el bloque PIL, señal "Diferencia" disponible en el scope "Numerical Difference" de la figura 4-31

5 IMPLEMENTACIÓN HARDWARE: CONVERTIDORES DE POTENCIA PARA SIMULACIÓN PIL

En este capítulo vamos a describir los distintos pasos para llevar acabo la implementación de algoritmos de control de circuitos de potencia en nuestra placa Hardware de la familia C2000 del fabricante TI.

En primer lugar, mencionaremos dos conceptos claves como son el de transistor bipolar de puerta aislada (**IGBT** en inglés) y la modulación por ancho de pulso (**PWM** en inglés). Por otro lado, trataremos de implementar la parte de control de una fuente de alimentación en nuestra placa de desarrollo. Por último, veremos lo pasos a seguir para implementar el controlador de un sistema de generación eléctrica basado en energía fotovoltaica con batería en el hardware.

5.1 Transistor bipolar de puerta aislada y Modulación por ancho de pulso

Uno de los elementos claves en los convertidores de potencia que nos ocupan es el IGBT. Dicho transistor, nos interesa usarlo en dos modos de funcionamiento, saturación y corte. En el primero, el transistor se comporta como un interruptor cerrador, es decir, dejando pasar la corriente del circuito, y en el segundo como un interruptor abierto.

La idea es controlar cada cuanto conmuta de un estado de funcionamiento al otro, para poder conseguir el comportamiento deseado en el circuito. Para definir este criterio de transición entre estados, se usará la señal PWM.

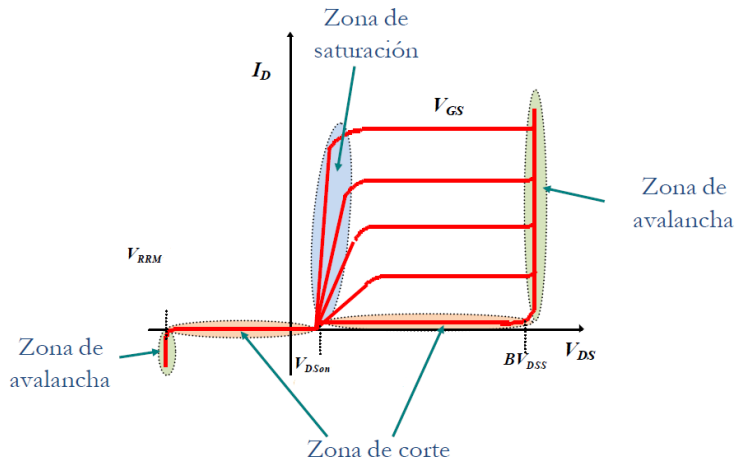


Figura 5-1. Curva característica estática de un transistor IGBT de canal n.

La técnica de la modulación PWM sirve en general para transmitir información. En este caso concreto se emplea para indicar cuándo se quiere que el transistor IGBT se comporte como un interruptor cerrado (conducción) o un interruptor abierto (no conducción). La información comparando una señal triangular con nuestra señal de control y como resultado obtendremos la señal PWM. Esta señal PWM será la que atacará la puerta de los transistores IGBT, gobernando el comportamiento del circuito.

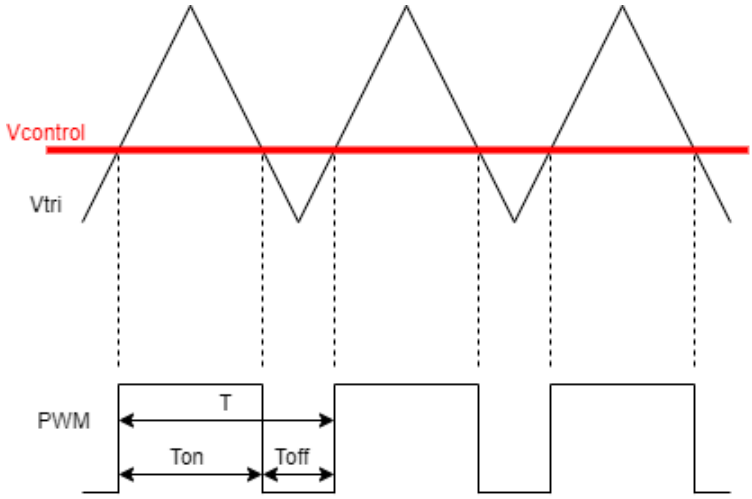


Figura 5-2. Funcionamiento de la modulación por ancho de pulso (PWM) .

En la modulación PWM se dispone de tres señales, la señal de control y la triangular son comparadas para obtener la señal PMW. La señal triangular siempre será constante (en amplitud y en frecuencia) y simétrica, por tanto, será la posición relativa entre la señal de control y la triangular la que vaya definiendo la anchura de pulso de nuestra señal PWM. En la Figura 5-2 se observa que la señal triangular en simétrica, sin embargo, en el escenario definitivo que se trata más adelante en el proyecto no todas las señales serán así. De hecho, en el convertidor DC/DC elevador se empleará una señal diente de sierra. Si en un momento determinado el valor de la señal triangular es mayor al correspondiente de la señal de control, la señal PWM se define con un valor alto (conducción). En caso contrario, la señal PWM se define con un bajo (no conducción). Se pueden definir los siguientes tiempos:

- El periodo de la señal PWM (T).
- El tiempo en el que conduce el transistor, es decir, en el que la señal que ataca a la puerta del transistor se encuentra con un valor lógico uno (T_{on}).
- El tiempo en el que no conduce el transistor, es decir, en el que la señal que ataca a la puerta del transistor se encuentra con un valor lógico cero (T_{off}).

Es esta señal PWM, que actúa en la puerta de los transistores IGTB, la que va a marcar el comportamiento de los mismos. Y relacionando con el párrafo anterior, concluimos que, con un valor lógico uno (Ton) de la señal PWM el IGBT de forma ideal se comportará como un cortocircuito y con un valor lógico cero (T_{off}) el transistor se comportará como un circuito abierto.

Este párrafo anterior nos permite explicar también en qué consiste el ciclo de trabajo (**Duty Cycle** en inglés, representada por la letra D). El D se define como el porcentaje en el que la señal PWM se encuentra con valor uno lógico. El control de esta variable D nos permitirá conseguir nuestros objetivos.

$$D = \frac{Ton}{T} \tag{5–1}$$

5.2 Ejemplo de implementación: Fuente de alimentación

En esta sección se incluirá el estudio de un convertidor DC/DC elevador funcionando como fuente de alimentación. Es cierto que no se trata del escenario completo pensado para este proyecto, sin embargo, este ejemplo se propuso como paso previo antes de abordar el definitivo. Este escenario se le asigna al alumno con la intención de que este se familiarice con un entorno de electrónica de potencia. Esto, sumado al manejo de la técnica PIL aprendido en los capítulos anteriores dejará al alumno preparado para afrontar en la siguiente sección el sistema de generación eléctrica aislada basado en energía fotovoltaica con batería y con aplicación del algoritmo MPPT.

El escenario que nos ocupa en esta sección, consta de un convertidor DC/DC elevador cuyo objetivo es entregar a la carga el doble de tensión de la que aporta la fuente de DC a la entrada del convertidor. En concreto, se trata de proporcionar la salida del convertidor una tensión V_o=10V, mayor que aquella que proporcionamos la fuente de DC a la entrada del convertidor una tensión de V_d=5V. Una de las claves en este ejemplo es el **IGBT** que, junto al lazo de realimentación, gobernará el comportamiento del circuito.

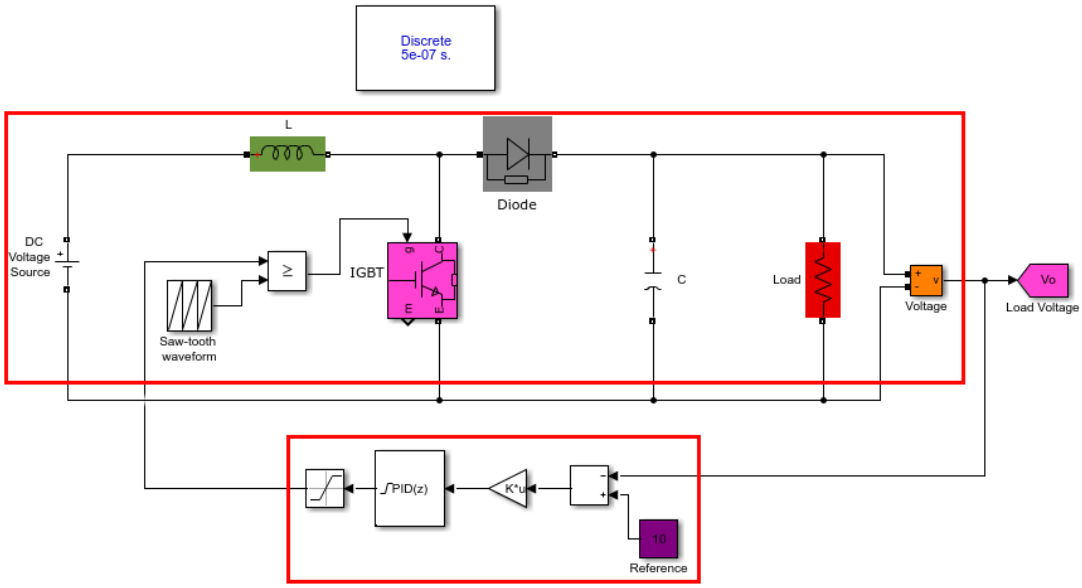


Figura 5-3. Convertidor elevador en fuente de alimentación .

Este escenario es del que partimos y podemos distinguir dos partes bien diferenciadas en la imagen anterior. Por un lado, tenemos la planta de nuestro sistema, es decir, aquello que queremos controlar. Por otro lado, observamos el lazo de realimentación que se encargará de controlar el **D** con el que se disparará el transistor, y por tanto influir sobre el comportamiento de nuestro circuito.

Teniendo como guía la subsección **4.3.1**, y haciendo uso del propio escenario **Figura 4-31**, llegaremos a simular mediante la técnica PIL el escenario que nos compete en esta sección: fuente de alimentación. Para ello, en primer lugar, necesitamos adaptar el escenario de dicha subsección para que sea coherente el nombre de los distintos bloques (planta y controlador) con lo que se encuentra en su interior, además de la funcionalidad que desempeñan.

La razón por la cual vamos a reutilizar un escenario de una subsección anterior es clara, a medida que vayamos avanzando en este proyecto nos iremos encontrando más dificultades a la hora de implementar el controlador con la técnica PIL. Es por ello, que vemos innecesario tener que añadir la dificultad de configurar el escenario para poder realizar una simulación PIL desde cero. Además, sería eficiente configurar de nuevo el escenario por el simple hecho de que al menos para este proyecto se van a tratar tres escenarios distintos con la técnica PIL.

Como se ha mencionado anteriormente, existen dos partes diferenciadas: planta y controlador. A continuación, presentamos aquellas partes del circuito que irán incluidos en cada bloque. En el bloque planta introduciremos

la parte del circuito que se muestra en la imagen inferior izquierda de la **Figura 5-4**, mientras que en el bloque controlador se incluye el lazo de realimentación que se observa en la imagen inferior derecha.

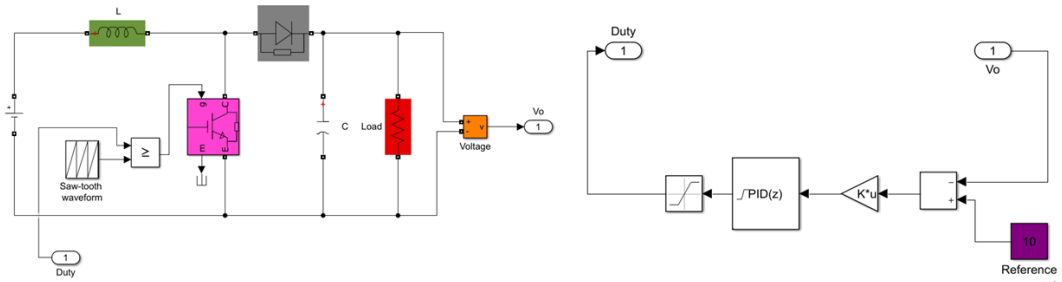


Figura 5-4. Planta y controlador de la fuente de alimentación, respectivamente .

Ya hemos designado a cada bloque aquella parte del circuito que nos interesa. Aunque recomendamos encarecidamente la reutilización del escenario representado en la **Figura 4-31** y revisar la subsección **4.3.1** antes de avanzar, pues es un escenario cuyo único fin es usar la técnica PIL, indicaremos igualmente a modo de resumen una serie de modificaciones y configuraciones necesarias para emplear la técnica PIL. La **primera**, la configuración de algunos apartados en la ventada de parámetros del modelo. La **segunda**, pasar en los bloques que lo requieran del dominio del tiempo continuo al discreto. La **tercera**, debemos asegurar que cada bloque dentro del controlador va a heredar el tipo de dato single del bloque anterior, sino, debemos forzar dicho tipo de dato.

1. En este primer apartado del que nos ocuparemos será la ventana de parámetros del modelo, mostrada como un icono de una tuerca en la barra de herramientas de Simulink en nuestro escenario. Indicaremos las pestañas y los campos que hay que modificar a modo de lista pues esta configuración se ha detallado con imágenes en un apartado anterior del proyecto.
    a. En la pestaña *Solver*:
        i. Solver Section
            1. *Type* como *Fixed-step*
            2. *Solver* como *discrete (no continuous states)*
        ii. *Solver details*: *Fixed-step size (fundamental simple time)* como *0.5e-6* (este parámetro es modificable y elegimos en el bloque "powergui" de nuestro escenario el mismo tiempo de muestreo)
    b. En la pestaña *Diagnostics* > *Hardware Implementation*:
        i. *Hardware board* como *TI Delfino F28379D LaunchPad*
        ii. Hardware boardsettings > *Target hardware resources* > *Build options* >*Build action* como *Build*
    c. En la pestaña *Code Generation*:
        i. Toolchain settings > *Toolchain* como *Texas Instruments Code Composer Studio (C2000)*
    d. En la pestaña *Code Generation* > *Verification*:
        i. *Advance parameters* > *Create Block* como *PIL*

2. En este segundo apartado trataremos de cambiar aquellos bloques que funcionen en el dominio del tiempo continuo para que hagan lo propio en el dominio del tiempo discreto. Para este caso, solamente tendremos que modificar un bloque y es el controlador PID que está situado en el bloque de control. Para modificar dicha característica basta con hacer doble clic en el bloque del controlador ***Proporcional-Integral-Derivativo (PID)*** y en la sección de ***Time domain*** pasamos de ***Continuous-time*** a ***Discrete-time***.

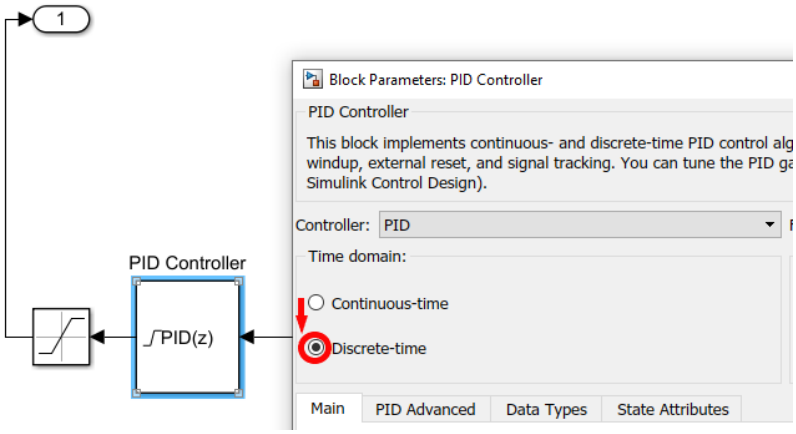


Figura 5-5. Controlador PID: Paso del dominio del tiempo continuo a discreto.

3. El tercer apartado nos aseguraremos de que todos los tipos de datos dentro del bloque controlador son de tipo single para que luego, a la hora de construir y ejecutar el escenario con nuestro bloque PIL no tengamos problemas y nos avise Matlab de que hay algo que no está bien configurado. El mensaje que se muestra a continuación puede ser un ejemplo de mensaje de advertencia por parte de Matlab. Tiene que quedar claro que si dentro del bloque controlador hay bloques cuyo tipo de dato son distintos a single, no podremos seguir avanzando con la construcción y ejecución de nuestro proyecto.

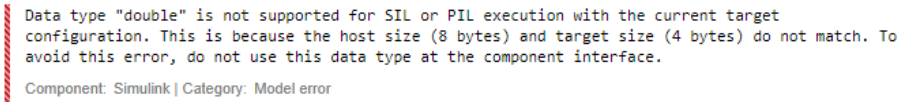


Figura 5-6. Error al intentar simular con tipo de dato “double”.

Para realizar el cambio de tipo de dato debemos hacer doble clic sobre todos los componentes que se encuentren en el interior de nuestro bloque controlador. En nuestro caso se encuentran diferentes bloques tales como una constante, un bloque suma, una ganancia un controlador PID y una saturación. Una vez que hemos hecho doble clic sobre el subcomponente en concreto debemos irnos a la pestaña ***signal Attributes*** y seleccionar el parámetro ***Output data type*** como ***single***. Es decir, forzamos que a la salida de cada bloque dentro de nuestro controlador sea de tipo **single**. En el caso concreto del controlador PID, debemos ir a la pestaña ***Data Type***.

En esta próxima imagen se observa dónde se cambia exactamente el tipo de dato para cada uno de los componentes que forman parte de la parte de control

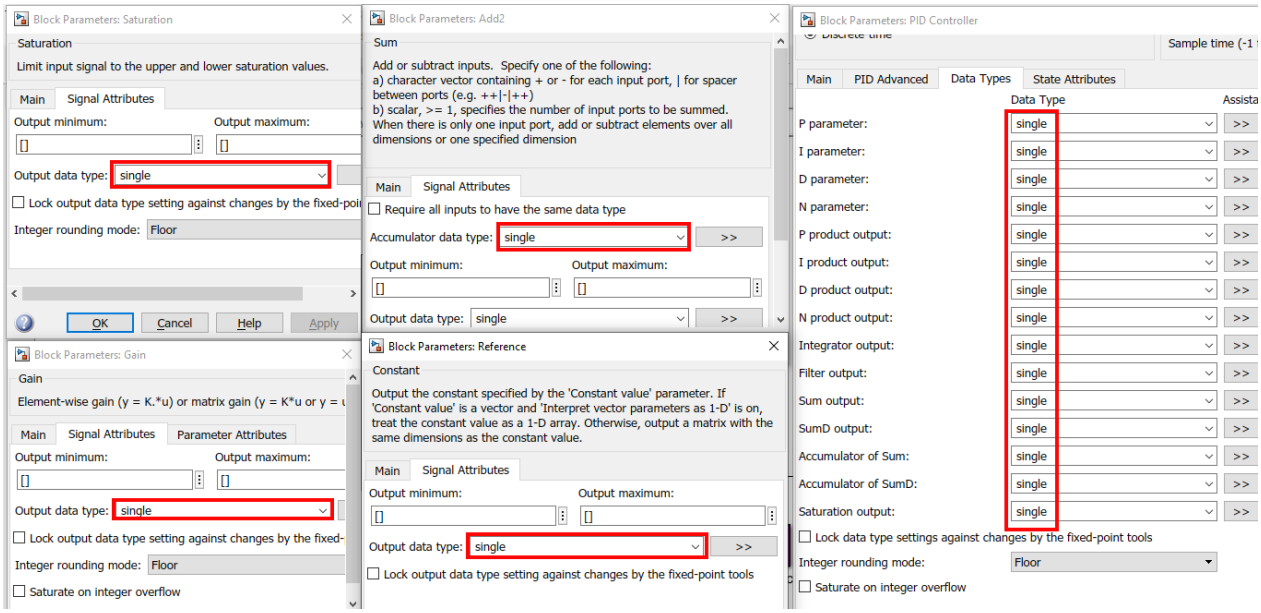


Figura 5-7. Todos los bloques del controlador con tipo de dato “single”.

Una vez que hayamos terminado estos tres pasos clave para la preparación del sistema de la técnica PIL, debemos situarnos de vuelta en el escenario principal. Seguidamente hacemos clic derecho sobre el subsistema ***Controller***, nuestro controlador, debemos seleccionar ***C/C++ Code > Deploy this Subsystem to Hardware***. Después, aparecerá un cuadro de diálogo donde tenemos que hacer clic en ***Build***.

Aparecerá un bloque PIL, en una ventana emergente, el cual copiaremos y posteriormente lo pegaremos en nuestro escenario en paralelo a nuestro bloque de control. Justo como en la **Figura 4-30**, donde se observa que el bloque PIL queda en paralelo al bloque controlador. La disposición de los bloques planta, controlador y PIL se aprecia en la **Figura 4-31**. A la hora de ejecutar, elegimos un tiempo de simulación de 0.5 segundos, que es el tiempo necesario para poder ver cómo la tendencia de la tensión de salida llega a estabilizarse en régimen permanente, y un modo de simulación normal. Entonces, ejecutamos el modelo haciendo clic en el icono circular de color verde de la siguiente imagen.

Figura 5-8. Tiempo y modo de simulación elegidos.

Como valor añadido a la simulación, vamos a incluir en una sola gráfica dos curvas de la tensión a la salida del convertidor DC/DC elevador la primera de ellas es usando el bloque controlador original (curva roja) y la segunda cuando usamos bloque PIL (curva azul), que hace las veces de controlador.

En la siguiente imagen se observa que con el bloque PIL se sigue la tendencia esperada de la curva de tensión a la salida del elevador. Recordemos que en la **Figura 4-35** también se obtuvo un comportamiento similar. La diferencia en la tensión salida de nuestro circuito empleando ambos bloques de control es aproximadamente cero, del orden que pudimos observar en la **Figura 4-36** al realizar el ejemplo del PIL proporcionado por Matlab – Simulink.

Por tanto, se puede llegar a la conclusión de que el hardware no solo ha soportado el código del controlador, sino que además la técnica de simulación PIL ha sido perfectamente capaz de replicar el comportamiento del controlador original.

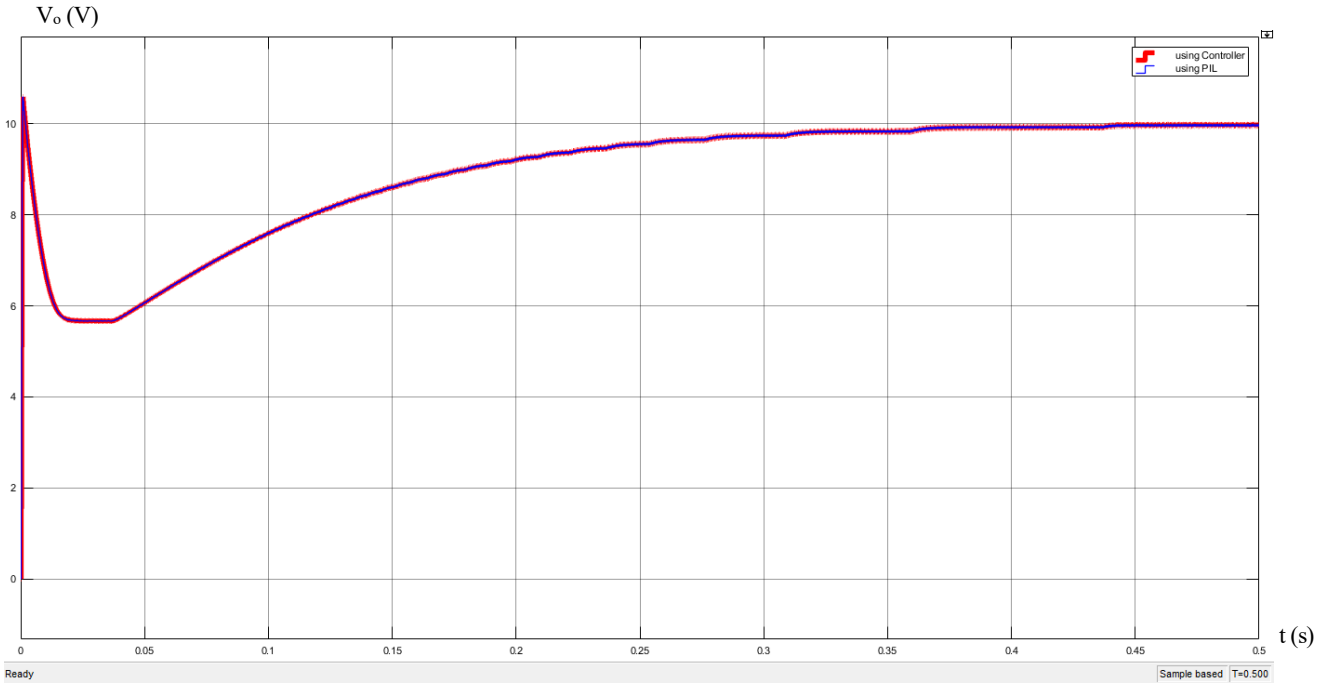


Figura 5-9. Tensión de salida del elevador (V_o) que aparece en el modelo Simulink de la figura 5-3: usando el controlador (roja) y usando el bloque PIL (azul).

Cabe destacar que por primera vez hemos sido capaces de usar la técnica de simulación PIL en un circuito de electrónica de potencia básico. A partir de ahora, seremos capaces de abordar escenarios más complejos.

5.3 Escenario completo: Sistema fotovoltaico aislado con batería (MPPT)

En esta sección vamos a presentar el sistema de generación eléctrica aislado, basado en energía fotovoltaica aislada con batería. En primer lugar, se muestra la aplicación original completa en el software de simulación Matlab – Simulink. Entonces se puede relacionar los distintos bloques que componen este sistema con aquel de introducción explicado en el capítulo **3**.

En segundo lugar, se procede a simular el sistema para saber de dónde se parte, es decir, cómo evolucionan ciertas variables del sistema. De este modo, al final de la sección será posible comparar las gráficas obtenidas tras la implementación de la técnica PIL y las gráficas originales para sacar conclusiones. En tercer lugar, se implementará la técnica PIL que conlleva que el código del controlador se ejecute en la placa de desarrollo. Por supuesto para conseguir dicha implementación será necesario adaptar la arquitectura original tal y como se ha visto en capítulos y secciones anteriores.

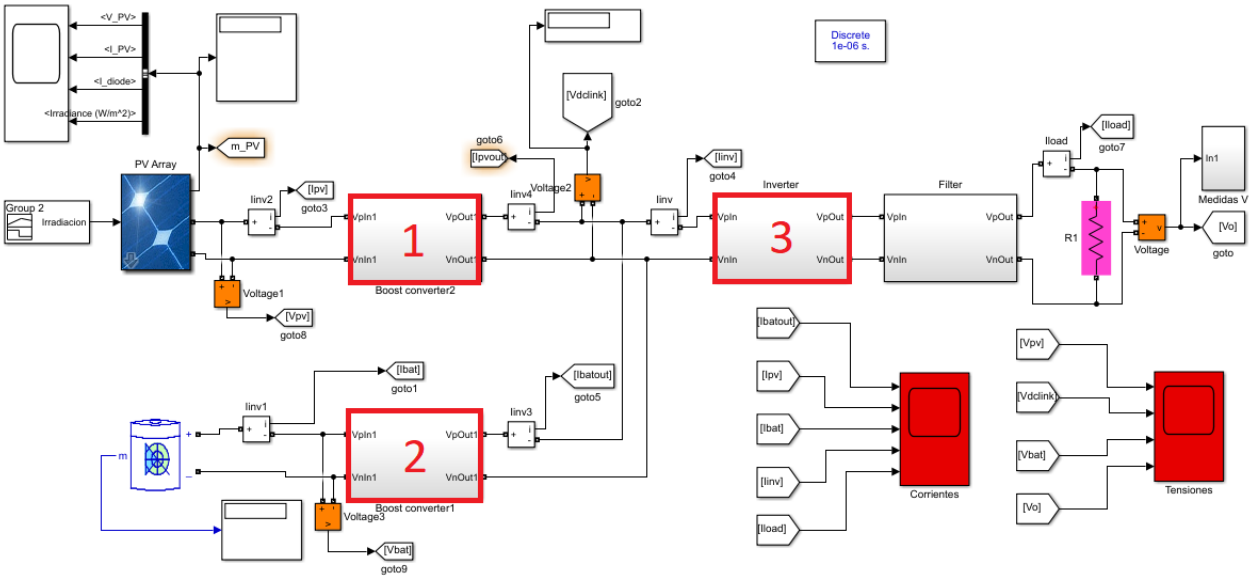


Figura 5-10. Sistema de generación eléctrica aislado basado en energía fotovoltaica con batería.

El escenario está compuesto principalmente por un modelo de placas solares, con parámetro de placas reales, en este caso se tratan de las placas situadas en el techo del edificio de la escuela técnica superior de ingeniería. A continuación, podemos observar tres convertidores de potencia: (1) el bloque DC/DC elevador, (2) el bloque DC/DC bidireccional (reductor-elevador) y (3) el bloque DC/AC inversor. Cabe mencionar que dentro del bloque elevador se ejecuta el algoritmo MPPT. Además, encontraremos un filtro y una carga a la que se le entregará la potencia generada. Una batería, que se encuentra conectada a un inversor (2), se cargará si la potencia entregada por las placas satisface la potencia requerida por la carga. En otro caso, la batería se descargará para aportar potencia a la carga.

El comportamiento de las señales originales de tensión a la salida de los paneles, la intensidad a la salida de los paneles y la irradiancia (la original, pues más adelante se modificará) la veremos en la siguiente imagen. Además, está incluida en la tercera posición de la próxima imagen la intensidad del diodo de paneles. El bloque del cual obtenemos dichas gráficas es el bloque “scope” que se encuentra situado en la parte superior izquierda de la figura anterior.

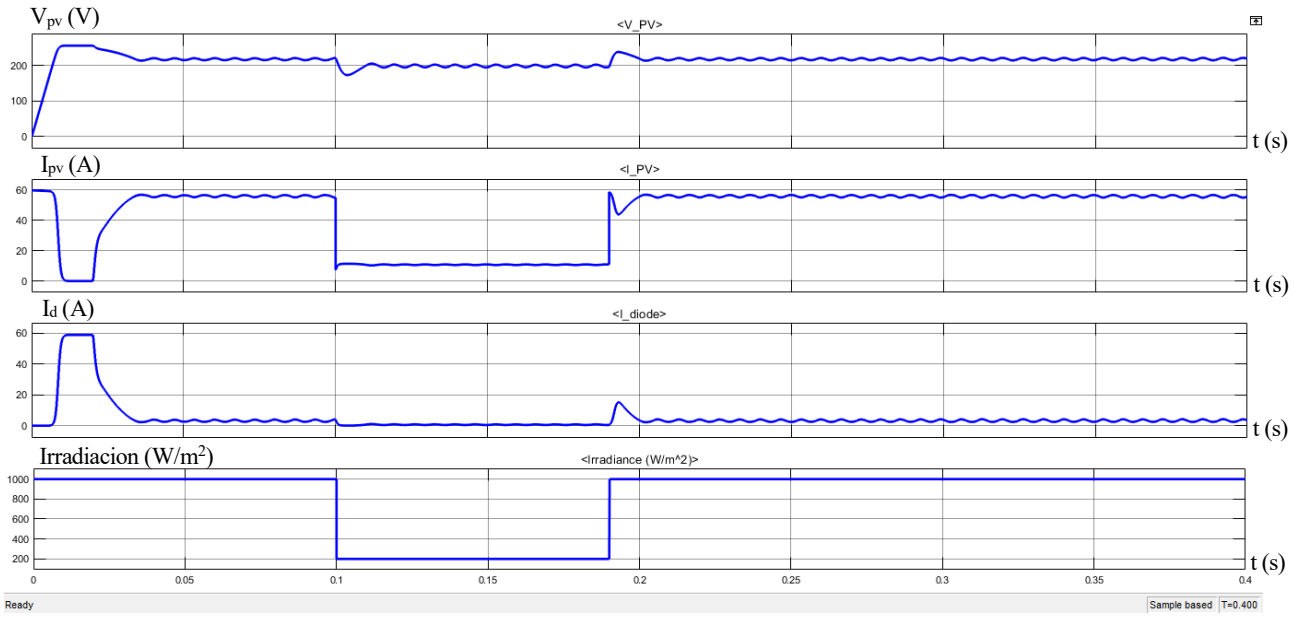


Figura 5-11. Escenario completo, las variables aparecen en el modelo Simulink de la figura 5-10: Tesión (V_{pv}) e intensidad (I_{pv}) a la salida del sistema de paneles, la intensidad del diodo (I_d) del modelo equivalente a los paneles solares y la irradiancia (Irradiacion).

El modelo del módulo comercial del que disponemos en la simulación es el **SPR-305-WHT** del fabricante **Sun Power**, y como hemos comentado anteriormente, los parámetros seleccionados son sacados de unos módulos fotovoltaicos reales. En la **Figura 5-12** se presentan las curvas características de tensión, intensidad y potencia de un módulo fotovoltaico.

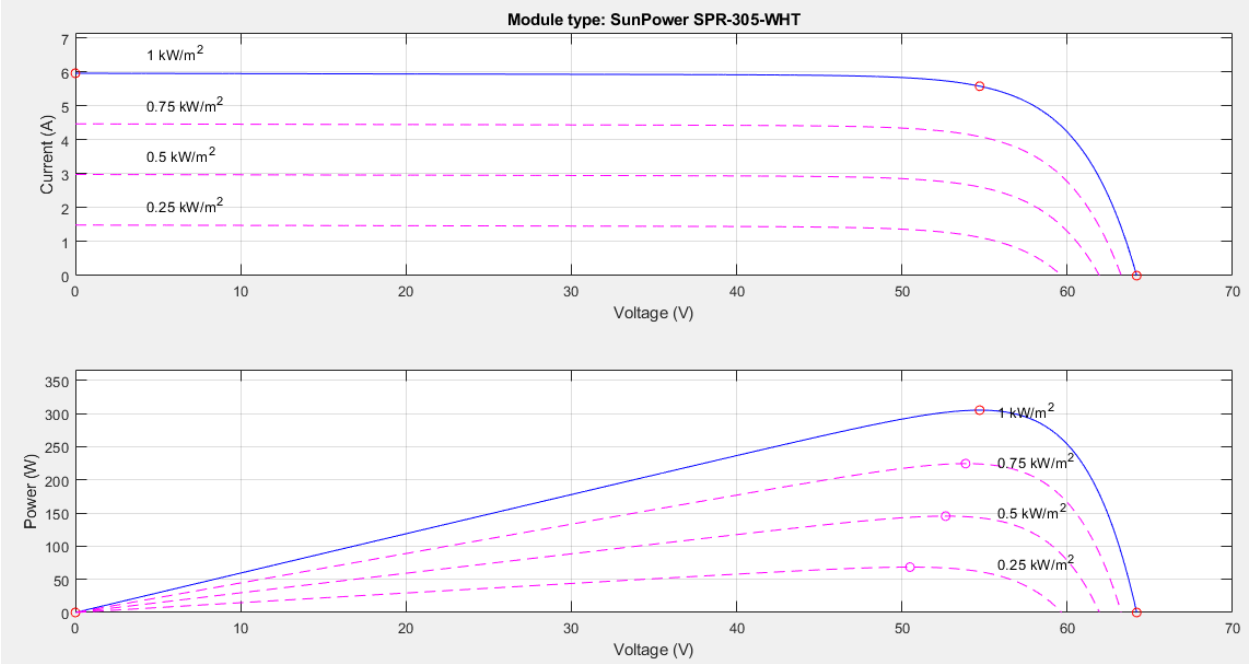


Figura 5-12. Curvas de salida del módulo fotovoltaico comercial SPR-305-WHT de Sun Power: intensidad, tensión y potencia.

Estas variables del módulo **SPR-305-WHT** no se pueden relacionar con ninguna variable del modelo de Simulink, dado que el sistema fotovoltaico del modelo está definido como un conjunto de módulos agrupados en serie y en hileras.

Sin embargo, generalmente, como en nuestro caso, se necesitará más de un módulo fotovoltaico ya que la potencia que requiere la carga será más elevada que aquella que nos proporciona un módulo por sí solo. Para esta aplicación en concreto, se configurará el número de paneles de tal manera que el conjunto de paneles entregue una potencia pico aproximada de **12** kW para una irradiación de **1000** W/m^2. Ajustando el número, se concluye que son necesarios **4** módulos en serie y **10** series en paralelo (hileras). Con los 4 módulos en serie se busca obtener una tensión en torno a los 220 V que es entregada al convertidor DC/DC elevador.

En la **Figura 5-13** se representan las curvas características tales como intensidad, tensión y potencia para este conjunto de paneles que formarán el sistema de generación eléctrica. Las curvas se representan para distintos valores de irradiación. Cabe mencionar que de la siguiente gráfica se pueden obtener algún dato interesante como por ejemplo la tensión de circuito abierto (**V_{oc}**), la cual nos será útil más adelante; así como también la intensidad de cortocircuito.

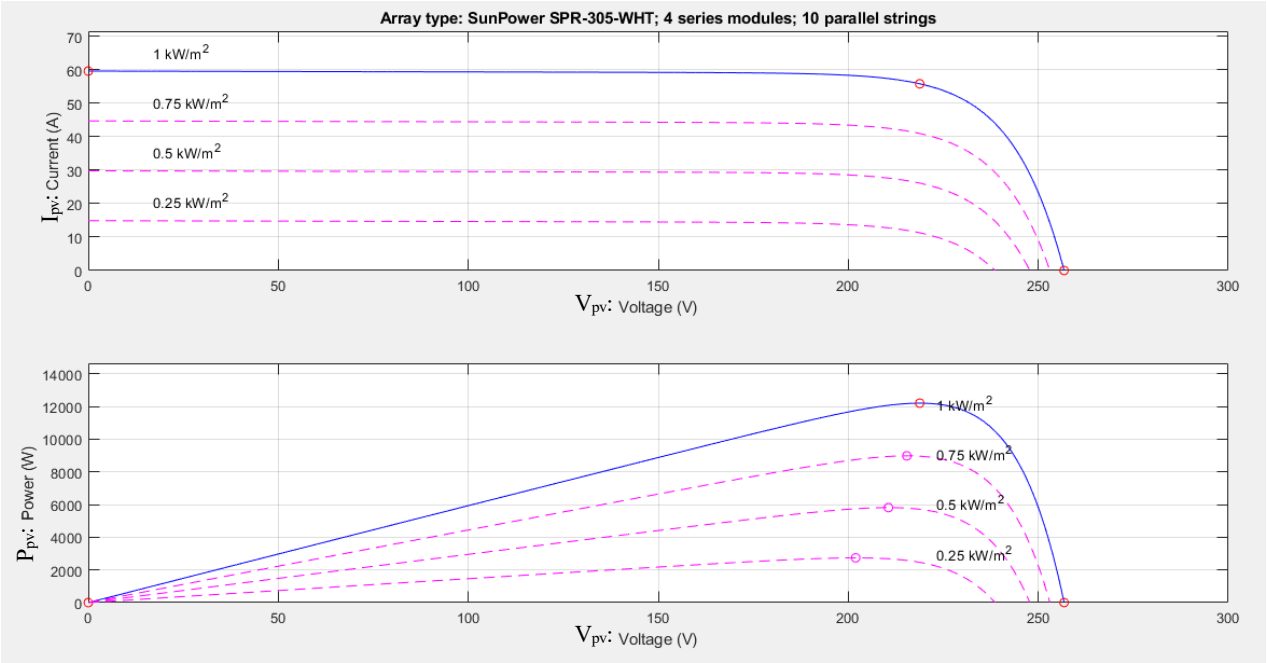


Figura 5-13. Curvas de salida del conjunto de paneles fotovoltaicos SPR-305-WHT de Sun Power: intensidad (I_{pv}), tensión (V_{pv}) y potencia (P_{pv}= I_{pv}* V_{pv}) a la salida del conjunto de paneles.

En este caso, las variables comentadas en la imagen anterior si se pueden relacionar directamente con aquellas variables que se aprecian en el modelo de Simulink de la **Figura 5-10**.

El objetivo que nos ocupa en las siguientes páginas, será el de adaptar este escenario a uno que nos permita hacer una simulación con la técnica PIL. En concreto, realizaremos dicha técnica para la parte de control del convertidor elevador sea la ejecutada en nuestra placa de desarrollo. Como ya hemos comentado ante, esta parte de control tiene integrado el algoritmo MPPT el cual consigue el funcionamiento con el máximo punto de potencia posible con la irradiación disponible.

No solo se comentará la adaptación de dicho escenario, sino que se argumentará la toma de decisiones y se expondrá las dificultades encontradas a lo largo de este proceso. Así como algunas recomendaciones que además de este ejemplo en concreto, pueden ser útiles para el desarrollo de otros proyectos en Simulink. Por ello, sin más dilación, pasamos a la configuración del escenario y nos apoyaremos haciendo referencias a apartados anteriores del proyecto, puesto que algunas acciones son muy similares, si no son las mismas.

Para inicializar algunos parámetros del escenario nos ayudaremos del fichero **parameter.m**, que es adjuntado en el **Anexo A.** Ejecutaremos dicho fichero una vez abramos el escenario de carácter renovable, es decir, el escenario con el que trabajaremos en este apartado.

Aunque se recomienda la reutilización del escenario representado en la **Figura 4-31** y revisar la subsección **4.3.1** antes de avanzar, pues es un escenario cuyo único fin es usar la técnica PIL, indicaremos igualmente a modo de resumen una serie de modificaciones y configuraciones necesarias para emplear la técnica PIL. La **primera**, la configuración de algunos apartados en la ventada de parámetros del modelo. La **segunda**, pasar en los bloques que lo requieran del dominio del tiempo continuo al discreto. La **tercera**, debemos asegurar que cada bloque dentro del controlador va a heredar el tipo de dato single del bloque anterior, sino, debemos forzar dicho tipo de dato.

1. En este primer apartado del que nos ocuparemos será la ventana de parámetros del modelo, mostrada como un icono de una tuerca en la barra de herramientas de Simulink en nuestro escenario. Indicaremos las pestañas y los campos que hay que modificar a modo de lista pues esta configuración se ha detallado con imágenes en un apartado anterior del proyecto.
    a. En la pestaña *Solver*:
        i. *Solver Section*
            1. *Type* como *Fixed-step*
            2. *Solver* como *discrete (no continuous states)*
        ii. *Solver details*: *Fixed-step size (fundamental simple time)* como *1e-6* (este parámetro es modificable y elegimos en el bloque "powergui" de nuestro escenario el mismo tiempo de muestreo)
    b. En la pestaña *Diagnostics* > *Hardware Implementation*:
        i. *Hardware board* como *TI Delfino F28379D LaunchPad*
        ii. *Hardware boardsettings* > *Target hardware resources* > *Build options* >*Build action* como *Build*
    c. En la pestaña *Code Generation*:
        i. *Toolchain settings* > *Toolchain* como *Texas Instruments Code Composer Studio (C2000)*
    d. En la pestaña *Code Generation* > *Verification*:
        i. *Advance parameters* > *Create Block* como *PIL*

2. En este segundo apartado trataremos de cambiar aquellos bloques que funcionen en el dominio del tiempo continuo para que hagan lo propio en el dominio del tiempo discreto. Para este caso, solamente tendremos que modificar tres bloques y son los controladores PI. Para modificar dicha característica basta con hacer doble clic en el bloque del controlador ***Proporcional-Integral (PI)*** y en la sección de ***Time domain*** pasamos de ***Continuous-time*** a ***Discrete-time***. Los distintos controladores PI se encuentran en:
    a. Convertidor DC/DC elevador > *boost-control*:
        i. *External control loop*
        ii. *Inner control loop*
    b. Convertidor DC/DC bidireccional > *PWM*

3. Este tercer apartado es idéntico al que se encuentra en la página **54.**

En la **Figura 5-14** se observa la curva de irradiación original, la cual decidimos modificarla ligeramente para poder prolongar un poco el comportamiento transitorio en el cambio del valor de irradiación. Esta modificación se realiza ensanchando (0.04s) el intervalo en el cual la irradiación se mantiene al mínimo valor (200 W/m²).

Con esta modificación se pretende y se conseguirá apreciar como la curva de la tensión de referencia (salida del algoritmo MPPT) es capaz de seguir la tensión de salida de los paneles fotovoltaicos. Finalmente, la caída del valor a 200 W/m² se realiza en 0.1s mientras que la vuelta al valor de 1000 W/m² tendrá lugar en los 0.19s. Estos valores de tiempo son meramente orientativos y su valor no ha sido especialmente elegidos por un motivo en concreto, más que el dar tiempo suficiente al algoritmo para que sea capaz de hacer el correcto seguimiento del punto máximo de potencia durante el transitorio en los cambios de irradiación.

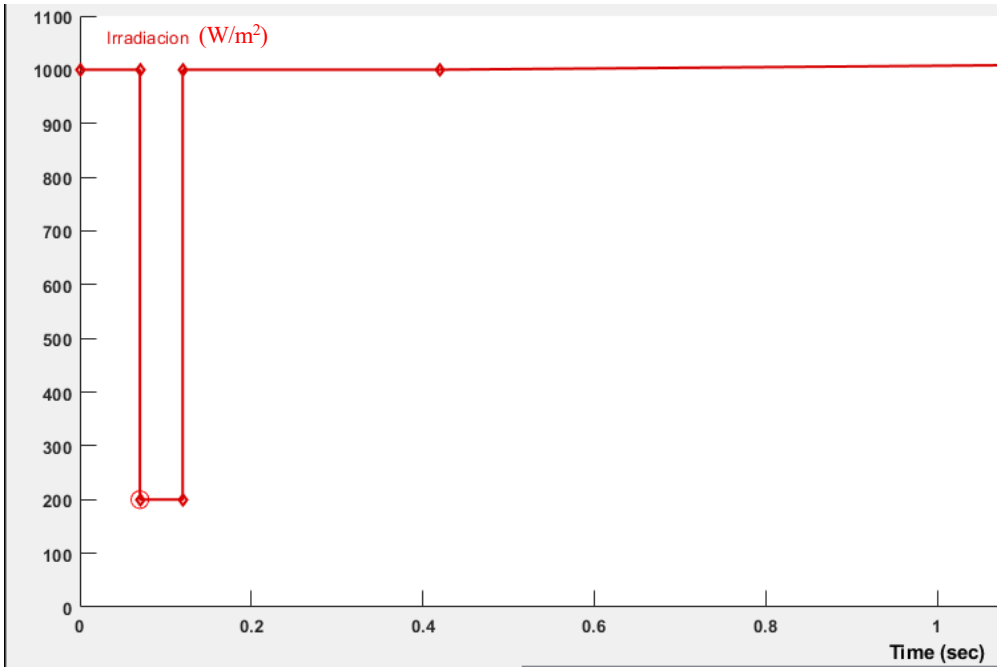


Figura 5-14. Señal de irradiación original, entrada de los paneles fotovoltaicos.

Simulando el escenario en modo discreto nos percatamos de que existe un aviso de advertencia por parte de Matlab. En dicho mensaje nos indica que no es posible simular, ya que el modelo aún contiene algún bloque continuo y anteriormente habíamos seleccionado un solucionador de tiempo discreto.

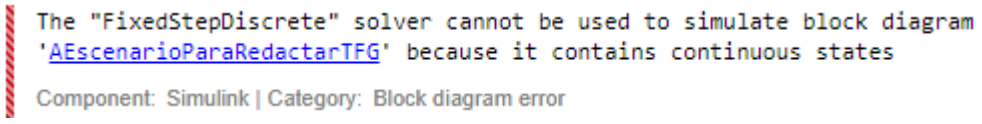


Figura 5-15. Error de simulación debido al uso de bloques trabajando en el dominio del tiempo continuo.

Para solventar este problema indagamos en los distintos subsistemas del escenario y hallamos que hay algunos bloques encargadas de hacer medidas en algunas de las señales (análisis de Fourier, por ejemplo) en las que se emplean integradores en el dominio del tiempo continuo. Estas medidas se realizan sobre algunas señales de tensión e intensidad y para no tener que modificar tantos integradores a su equivalente en el dominio del

tiempo discreto, decidimos eliminarlos para que no se tengan en cuenta a la hora de realizar la simulación. Ahora debería poder simularse el escenario en tiempo discreto.

Una vez que se nos permite simular el escenario en tiempo discreto procedemos introducirnos de lleno en el subsistema más ambicioso e interesante pues incluye el algoritmo MPPT, se trata del subsistema señalado con el número 1 en la **Figura 5-10**, y es el convertidor elevador. Dentro de este último podemos encontrar tres partes bien diferenciadas, como se muestra en la figura **Figura 5-16**. Por un lado, tenemos el propio circuito elevador, por otro lado, y remarcado en rojo, la parte de control del elevador. Y, por último, tenemos el bloque encargado de generar la señal PWM. En este trabajo nos centraremos en el segundo bloque, el bloque encargado de controlar el convertidor elevador.

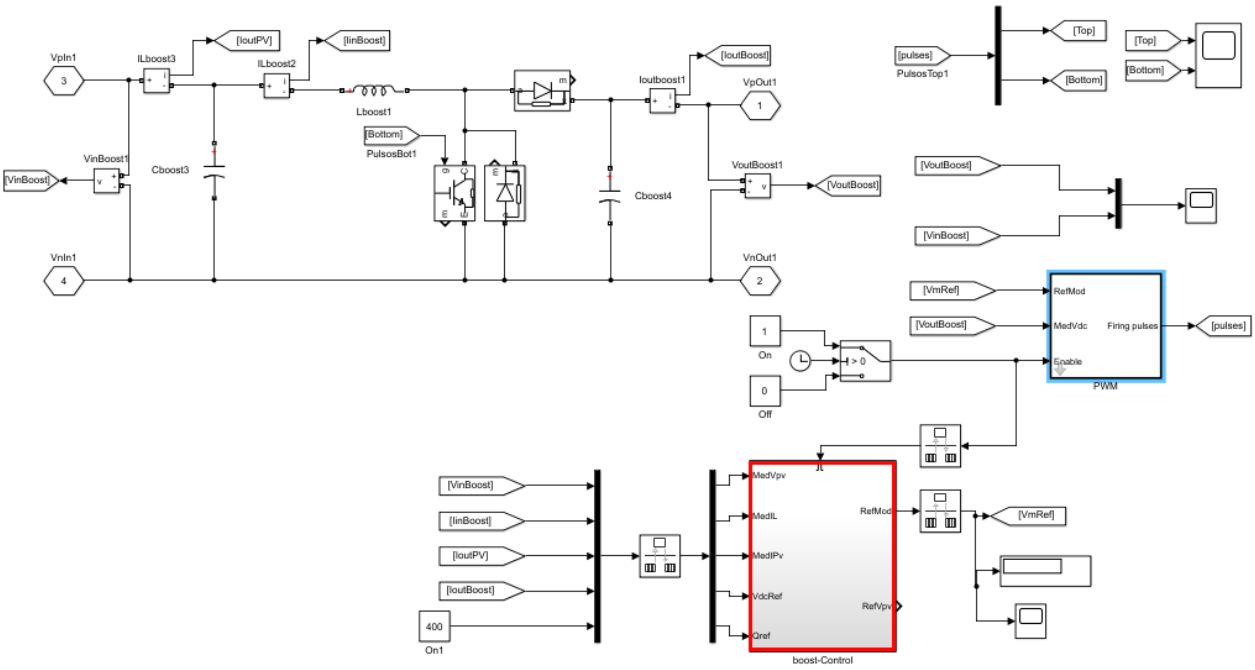


Figura 5-16. Convertidor de potencia DC/DC elevador.

El bloque controlador es un subsistema habilitador o "enabled" (marcado en rojo en la **Figura 5-16**) que consta de tres tipos de puertos, como el de entrada, el puerto de entrada habilitador, por el que entra la señal de control, y el puerto de salida. La señal habilitadora puede ser un escalar o un vector. El funcionamiento del bloque responde a la siguiente lógica: tanto en el caso en el que la señal habilitadora sea mayor que cero (caso escalar) como en el caso en el que cualquiera de sus elementos sea mayor que cero (caso vector), el subsistema se ejecuta. En otro caso, no se ejecutarán las acciones que se encuentren dentro del subsistema.

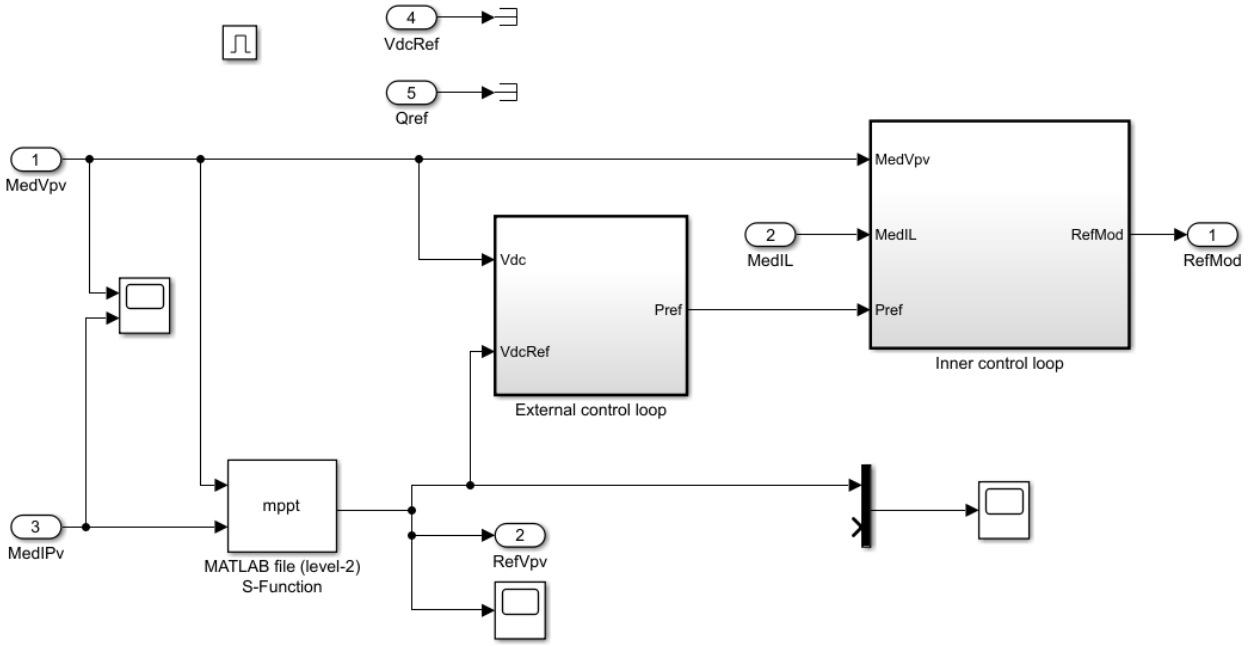


Figura 5-17. Sistema de control (*boost-control*) del convertidor DC/DC elevador.

A modo esquematizado, la **Figura 5-18** intenta aclarar las operaciones que llevarán los bloques que conformar el controlador del convertidor elevador. El objetivo que nos proponemos es tratar de simular el escenario con la técnica PIL. Esto implica que tendremos que implementar tanto el algoritmo MPPT como las operaciones siguientes en nuestra placa de desarrollo.

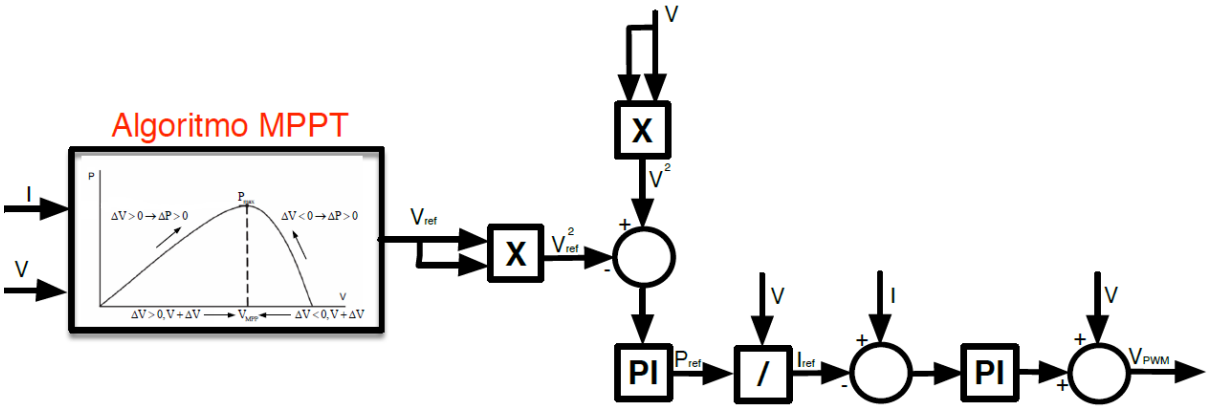


Figura 5-18. Esquema del sistema de control (*boost-control*) del convertidor DC/DC elevador.

Decidimos ir incluyendo los bloques que conforman el controlador del elevador de uno en uno al bloque PIL. Esta forma de proceder nos parece la más acertada pues si se detecta cualquier error podremos saber exactamente cuál es el bloque que está propiciando el fallo. De este modo, conseguimos reducir el tiempo dedicado buscar una solución al problema. El primer bloque que pasará a formar parte del bloque PIL será el bloque encargado de realizar el algoritmo MPPT. Este algoritmo está implementado en un bloque S-function de nivel 2 y su código original será mostrado en el **Anexo B**.

5.3.1 Implementación del algoritmo MPPT

Para ello, preparamos el escenario introduciendo el bloque MPPT en un subsistema, posteriormente añadimos un conversor de tipo single a la entrada del subsistema, por que como hemos visto anteriormente, es obligatorio usar el tipo de dato single dentro del bloque PIL, y luego, dicho subsistema lo implementamos en la placa de desarrollo. Antes de implementar el subsistema, el escenario del controlador del elevador debería quedar tal que así.

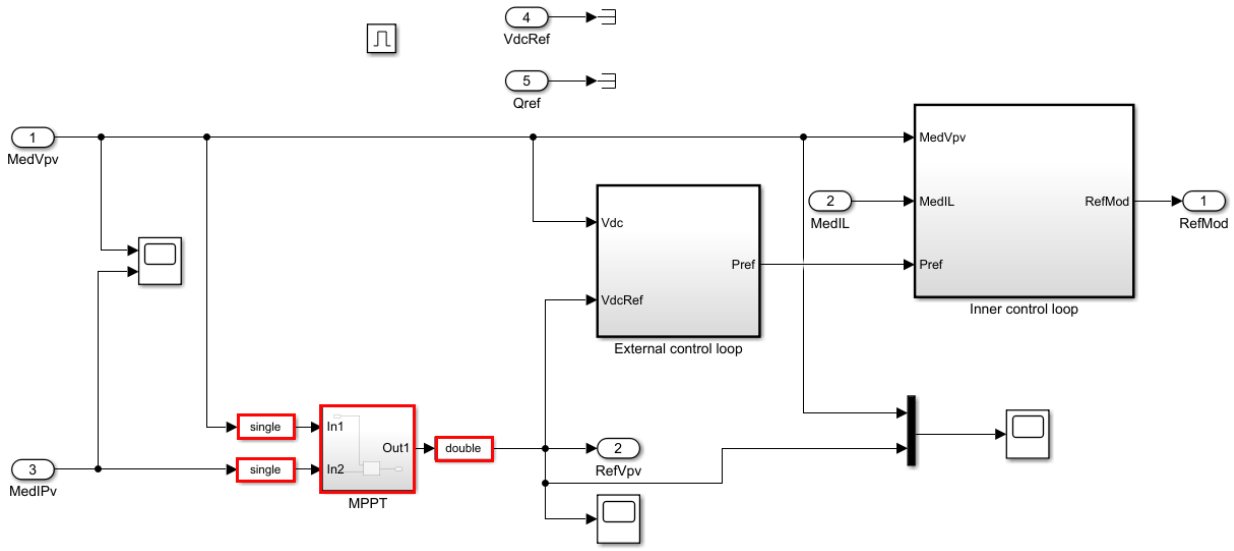


Figura 5-19. Sistema de control (*boost-control*): preparación del bloque MPPT marcado para PIL.

Sin embargo, antes de proceder a implementar el bloque del subsistema que incluye el algoritmo MPPT debemos asegurarnos que los datos dentro del algoritmo sean de tipo single, para que tenga coherencia con el tipo de dato con el que se trabaja dentro del bloque PIL. Para ello, modificamos el código original del algoritmo **mppt.m** y lo adaptamos a una versión con los datos tipo single **mppt_single.m**, este último archivo, se encuentran en el **Anexo C**. Si no realizamos este último cambio, no seremos capaces de simular el escenario porque debido a la incoherencia del tipo de dato dentro del controlador Matlab no nos lo permitirá y nos aparecerá un error. En la **Figura 5-20** se muestra la sustitución del bloque mppt por el bloque mppt_single.

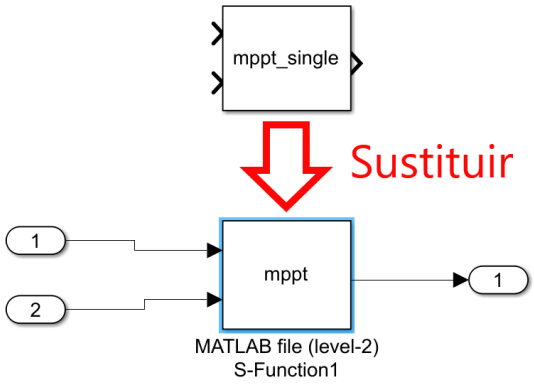


Figura 5-20. Sistema de control (*boost-control*) del convertidor DC/DC elevador.

Hay dos maneras de desplegar un bloque en el hardware. La primera, es hacer clic derecho sobre el subsistema que queremos implementar y posteriormente hacer clic en "C/C++ Code > Deploy this Subsystem to Hardware". La segunda, se trata de hacer clic sobre el bloque a implementar, posteriormente hacer clic en el icono que podrán observar en la siguiente imagen y luego hacer clic en "Deploy selected Subsystem to

Hardware". Podremos usar la segunda estrategia solo en caso de que dispongamos de dicho icono en la barra de herramientas.

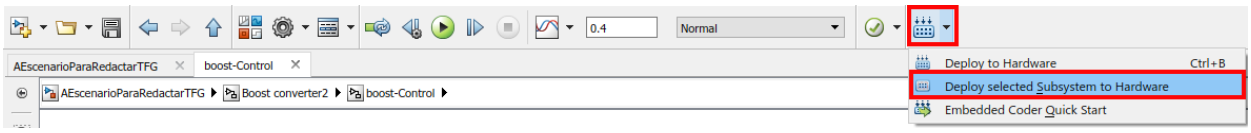


Figura 5-21. Implementar subsistema en el hardware.

Justo a continuación, debería aparecernos una ventana emergente con el bloque PIL. Como se ha visto en la **Figura 4-30**, ahora deberíamos copiar el bloque PIL y pegarlo en nuestro escenario, justo en paralelo con la parte de control que queremos implementar en nuestra placa de desarrollo que de momento es solo el algoritmo MPPT. Es recomendable colocar un interruptor manual para poder aplicar a la simulación la señal obtenida por el controlador en el software o la señal obtenida por el bloque PIL. Acto seguido, pulsamos el botón verde en la barra de herramientas para ejecutar la simulación.

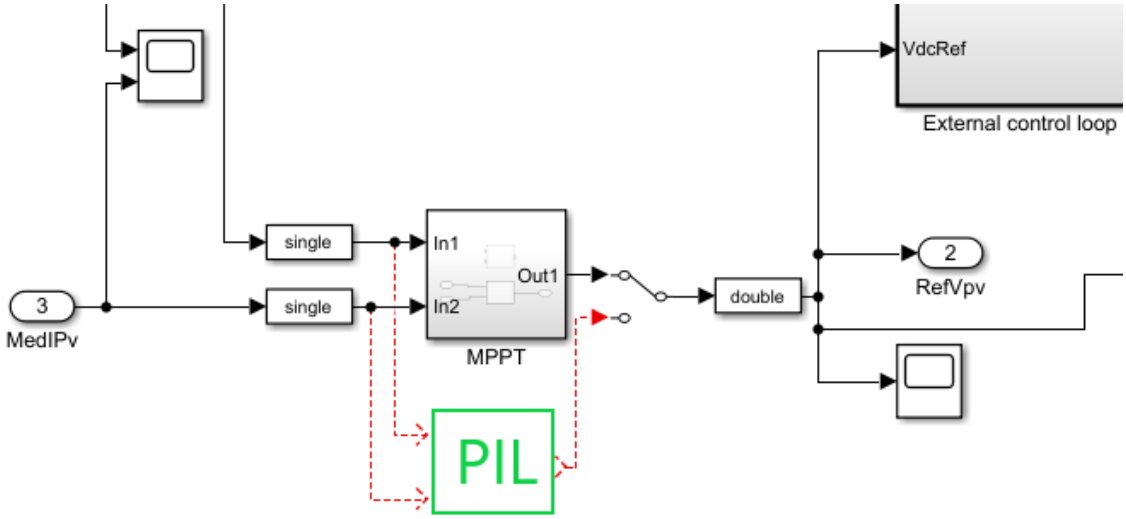


Figura 5-22. Sistema de control (*boost-control*) del convertidor DC/DC elevador: copia y pega del bloque PIL.

Entonces, el modelo comienza a ejecutarse, sin embargo, nos aparece un mensaje de error de Matlab sobre el bloque habilitador, se muestra en la próxima figura. Recordemos que en este bloque se encuentra todos los bloques que controlan el elevador, bloques como el algoritmo MPPT, el lazo de control externo y el lazo de control interno, todos ellos encargados de proporcionar la tensión PWM. Este mensaje de error nos advierte de que el **código generado para la verificación no permite** llamadas a bloques de ejecución condicional, como el bloque "enabled" en este caso. Como solución para poder evitar este error, nos instan a sacar el bloque controlador de un contexto condicional, es decir, que no podremos aplicar la técnica PIL dentro del bloque habilitador (bloque condicional en general).

```
*** Loading the program to the target...
*** Program is running.

*** Disconnecting from target...

*** Terminating debug session...

*** LOAD & RUN DONE.
```

```
An error occurred while running the simulation and the simulation was terminated
Caused by:
  • The block's enable callback has been invoked. This is not allowed because the generated code
    under test does not support enable semantics. To avoid this error, remove the block from a
    conditional execution context, for example, an enabled subsystem.
Component: Simulink | Category: Model error
```

Figura 5-23. Error al intentar implementar la técnica PIL dentro de un bloque de ejecución condicional.

Nos encontramos ante una de las tomas de decisiones del proyecto. Dado que no es viable seguir con la implementación de la técnica PIL del controlador del elevador puesto que la naturaleza del bloque "enabled" (bloque de ejecución condicional) no lo permiten, decidimos eliminar el comportamiento del bloque habilitador.

Para ello necesitaremos, a priori, un par de pasos. Por un lado, dado que no podemos realizar la técnica PIL dentro del bloque habilitador, vamos a ir desplazando los bloques que componen la parte de control (algoritmo MPPT, lazo de control externo y lazo de control interno) fuera del bloque habilitador. Por otro lado, y por tener coherencia con la primera acción, vamos a modificar un parámetro del archivo **parameters.m** a valor cero, en concreto, se trata de **TenablePV**. Dicho parámetro inicializa la ejecución del algoritmo MPPT, pero ahora que se saca el bloque MPPT del bloque "enabled" no tiene sentido inicializarlo.

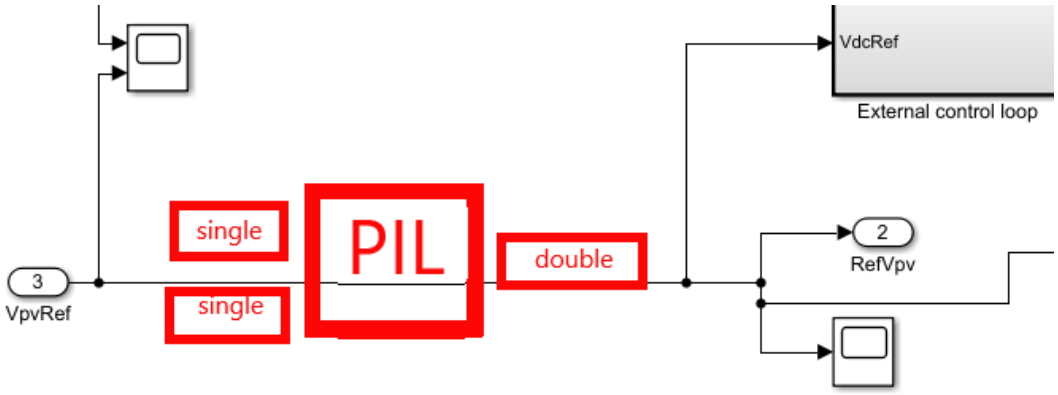


Figura 5-24. Se "corta" el bloque MPPT situado den el bloque "enabled" dentro del bloque elevador.

Los bloques que están situados en la silueta roja de la imagen anterior los cortamos (ctrl+x), subimos un nivel para desplazarnos a nivel del elevador y ahí los pegamos (ctrl+v). En esta próxima figura veremos cómo quedan los bloques anteriores colocados. Una vez que hemos recolocado los bloques, necesitamos conectar adecuadamente las señales para que exista coherencia. Para esto último, necesitaremos que la tercera entrada del bloque "enabled", situado en la parte inferior derecha de la **Figura 5-25**, sea la salida del bloque MPPT, es decir, la tensión de referencia de los paneles. Anteriormente, la tercera entrada del bloque "enabled" era la intensidad a la salida de los paneles.

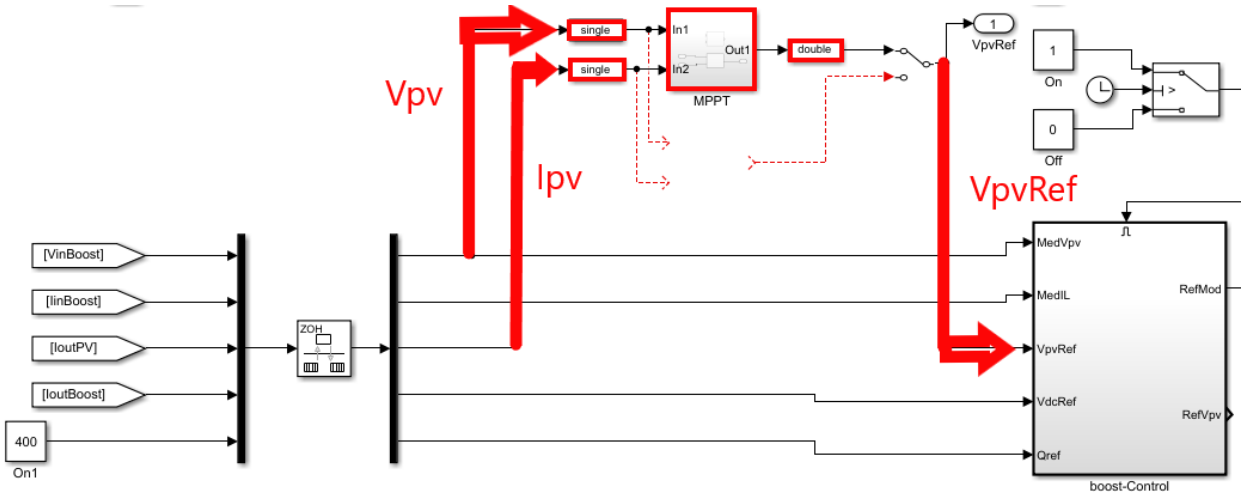


Figura 5-25. Se "pega" el bloque MPPT dentro del bloque elevador.

Sin embargo, cuando intentamos desplegar el subsistema en la placa de desarrollo (haciendo clic derecho sobre el subsistema que queremos implementar y posteriormente hacer clic en "C/C++ Code > Deploy this Subsystem to Hardware"), al hacer clic sobre construir, nos aparece el siguiente error.

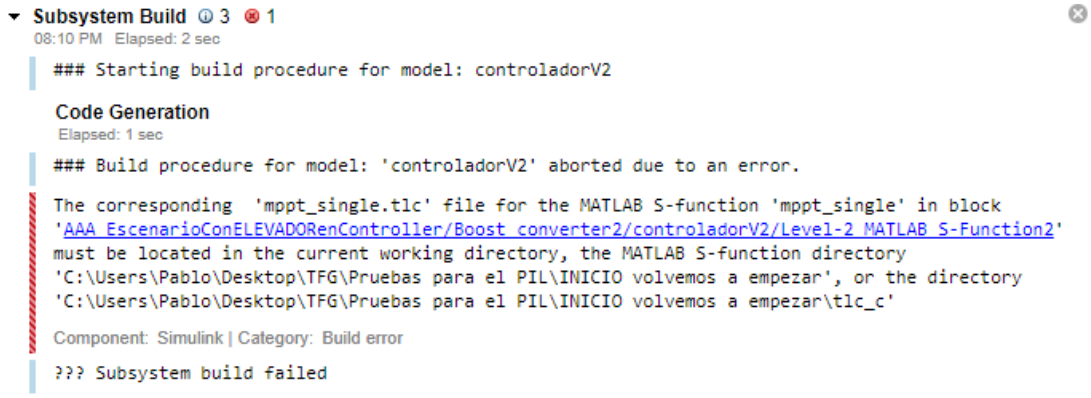


Figura 5-26. Error al inentar implementar la técnica PIL con "Level-2 S-function" sin su correspondiente "*archivo*.tlc".

El error se produce en el bloque del algoritmo MPPT. Debido a este error, se decide investigar en la página de MathWorks [12] y dentro de ella encontramos un apartado dedicado a comparar los distintos bloques configurables empleados para la generación de código. Nuestro bloque configurable es del tipo **Level-2 MATLAB S-function** y su función es ejecutar el algoritmo MPPT. A modo simplificado, cuando intentamos desplegar un subsistema de nuestro escenario a la placa de desarrollo, Simulink se encarga de generar el código automáticamente y posteriormente lo descarga en la placa.

El mensaje de error de la imagen anterior nos dice que ha habido un fallo porque debe haber un fichero "**.TLC**" (Target Language Compiler) en el directorio de trabajo y este no se encuentra. Efectivamente, ese fichero que se nos exige no existe debido a que este escenario, cuando se creó no estaba pensado para ser implementado en una placa de desarrollo y por tanto no se creó dicho archivo ".TLC". Según está publicado en la página de MathWorks, podemos extraer que con el tipo de bloque **Level-2 MATLAB S-function** (nuestro caso original) solo se generará código si implementamos el algoritmo (MPPT en nuestro caso) usando este tipo de lenguaje.

Modelling Considerations

Custom Block Type	Model State Dynamics	Simulation Performance	Code Generation
Interpreted MATLAB Function	No	Less fast	Not supported
Level-2 MATLAB S-function	Yes	Less fast	Requires a TLC file
MATLAB Function	No	Fast	Supported with exceptions
MATLAB System	Yes	Fast	Supported with exceptions
S-Function	Yes	Fast	Requires a TLC file or non-inline S-Function support
C Caller	No	Fast	Supported
S-Function Builder	Yes	Fast	Supported
Simulink Function	Yes	Fast	Supported
Subsystem	Yes	Fast	Supported

Figura 5-27. Comparativa de los distintos tipos de bloques configurables.

Dado que no disponemos de ese fichero TLC se nos presenta un abanico con diferentes maneras de implementar el código MPPT dentro de nuestro controlador, el cual queremos implementar en nuestra placa de desarrollo. Indagando por la página de MathWorks [13], encontramos el algoritmo implementado en el tipo de bloque configurable **MATLAB Function**.

Como podemos observar en la imagen anterior y situado justo debajo del tipo de bloque señalado en rojo, con el tipo de bloque **MATLAB Function** se permite la generación de código y, por consecuente, la plicación de la técnica de simulación PIL con algunas excepciones. Dado el ejemplo encontrado en la página web de MathWorks y la posibilidad de permitir la generación de código, decidimos seguir investigando por esta rama.

Un poco más adelante, siguiendo en la misma página web que en la imagen anterior podemos concluir que este nuevo tipo de bloque que estamos investigando, **MATLAB Function**, nos permite a nuestro modelo trabajar en el domino del tiempo discreto siempre que las variables las definamos como **persistent**. En primer lugar, nos interesa trabajar en el dominio del tiempo discreto, pues es uno de los requisitos para poder trabajar con la técnica PIL. En segundo lugar, comprobamos satisfactoriamente que en el ejemplo encontrado en la web de MathWorks las variables ya viene definidas como "persistent".

Model State Behavior

You need to model the state behavior for a block that requires some or all of its previous outputs to compute its current outputs. See State variable for more information.

Custom Block Type	Notes
Interpreted MATLAB Function, C Caller	Does not allow you to model state behavior.
MATLAB Function	Allows you to model a discrete state using `persistent` variables.
Level-2 MATLAB® S-Function	Allows you to model both continuous and discrete state behavior using the `ContStates` or `Dwork` run-time object methods in combination with block callback methods. For a list of supported methods, see Level-2 MATLAB S-Function Callback Methods in Write Level-2 MATLAB S-Functions.
MATLAB System	Allows you to model discrete state behavior using `DiscreteState` properties of the System object, in combination with block callback methods. This block uses System object™ methods for callback methods: `mdlOutputs` (`stepImpl`, `outputImpl`), `mdlUpdate` (`updateImpl`), `mdlInitializeConditions` (`resetImpl`), `mdlStart` (`setupImpl`), `mdlTerminate` (`releaseImpl`). For more information see What Are System Objects? (MATLAB).
C MEX S-Function, S-Function Builder	Allows you to model both continuous and discrete state behavior in combination with block callback methods. For more information, see Callback Methods for C MEX S-Functions
Simulink Function	Communicates directly with the engine. You can model the state behavior using appropriate blocks from the continuous and discrete Simulink block libraries. When multiple calls to this function originate from different callers, the state values are also persistent between these calls. For more information, see Call a Simulink Function block from multiple sites.
Subsystem	Communicates directly with the engine. You can model the state behavior using appropriate blocks from the continuous and discrete Simulink block libraries.

Figura 5-28. Comportamiento de los estados para los distintos tipos de bloques configurables.

Antes de poder sustituir el actual bloque que contiene el algoritmo MPPT (código **mppt_single.m** para el tipo de bloque **Level-2 MATLAB S-function**) por el nuevo bloque (código **mpptPandO** para el tipo de bloque **MATLAB Function**) debemos tener en cuenta un par de detalles. Por un lado, tenemos que modificar las condiciones iniciales de las variables para que se ajuste a los valores iniciales con los que trabajarán los paneles. Por otro lado, debemos forzar que el tipo de dato de dichas variables sean **single**, ya que, como sabemos, este tipo de dato es de carácter obligatorio dentro del controlador con el cual realizaremos la técnica PIL.

El bloque **MATLAB Function** contendrá la función **mpptPandO** cuyo código será adjuntado en el **Anexo D**. Allí, se podrá observar cómo hemos inicializados las distintas variables y cómo hemos forzado el tipo de dato single para adaptar la función a nuestro proyecto. Por lo tanto, una vez obtenido el bloque y habiendo configurado correctamente la función, nos disponemos a sustituir el antiguo bloque encargado de ejecutar el algoritmo MPPT.

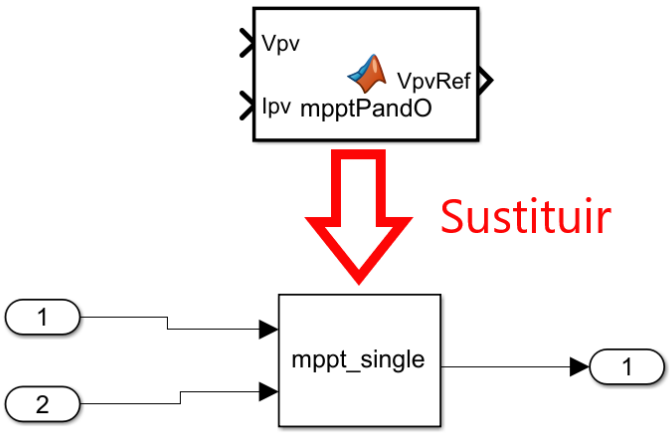


Figura 5-29. Algoritmo MPPT: sustitución del bloque "Level-2 MATLAB S-function" por "MATLAB Function".

A modo de resumen, nos situamos dentro del bloque elevador, el primer bloque existente a la salida de los panales fotovoltaicos. Dentro de este convertidor, encontramos toda la parte de control dentro del bloque habilitador, pero hemos visto que, con la generación de código, y por tanto para la técnica PIL, no se permiten llamadas a bloques de ejecución condicional. Nuestro bloque habilitador es de esta naturaleza. Es por eso que decidimos sacar la parte de control fuera del bloque habilitador, para poder implementar nuestro controlador en nuestra placa de desarrollo.

Por otro lado, también nos percatamos que para que nuestro bloque original (**Level-2 MATLAB S-function**) pueda ejecutar el algoritmo MPPT necesita un fichero ".TLC" en su mismo directorio. Decidimos, por tanto, probar con otro bloque distinto (**MATLAB Function**) que no nos exija un fichero que no poseemos. Después de esta recapitulación, procedemos a implementar el bloque controlador del elevador, que de momento solo incluye el bloque **MATLAB Function**.

Para ello, de nuevo, tenemos que hacer clic derecho sobre el subsistema que queremos implementar y posteriormente hacer clic en "C/C++ Code > Deploy this Subsystem to Hardware". Posteriormente hacemos clic en "Build". Una vez creado el bloque PIL en la ventana emergente, copiamos el bloque y pegamos paralelo a nuestro controlador, como hasta ahora se ha ido haciendo a lo largo del proyecto. Y, por último, pero no menos importante, ejecutamos el modelo. En la próxima figura se muestra cómo debería quedar el interior del bloque convertidor DC/DC elevador.

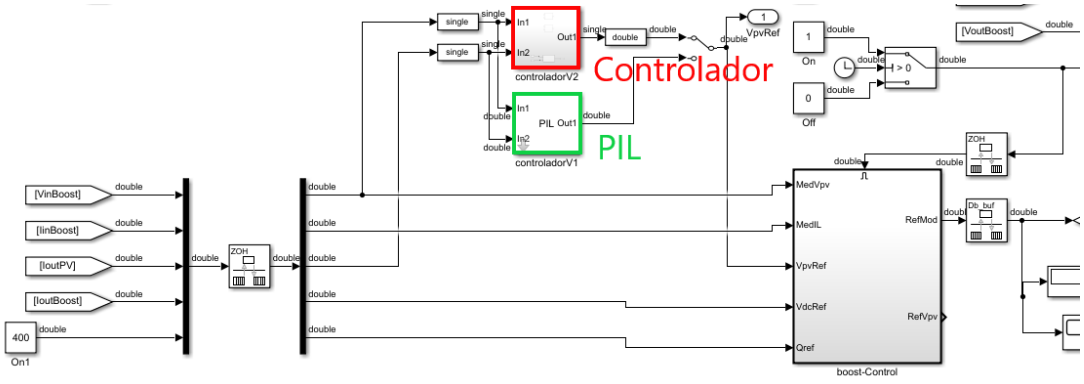


Figura 5-30. Convertidor elevador: algotirmo MPPT implementado con la técnica PIL.

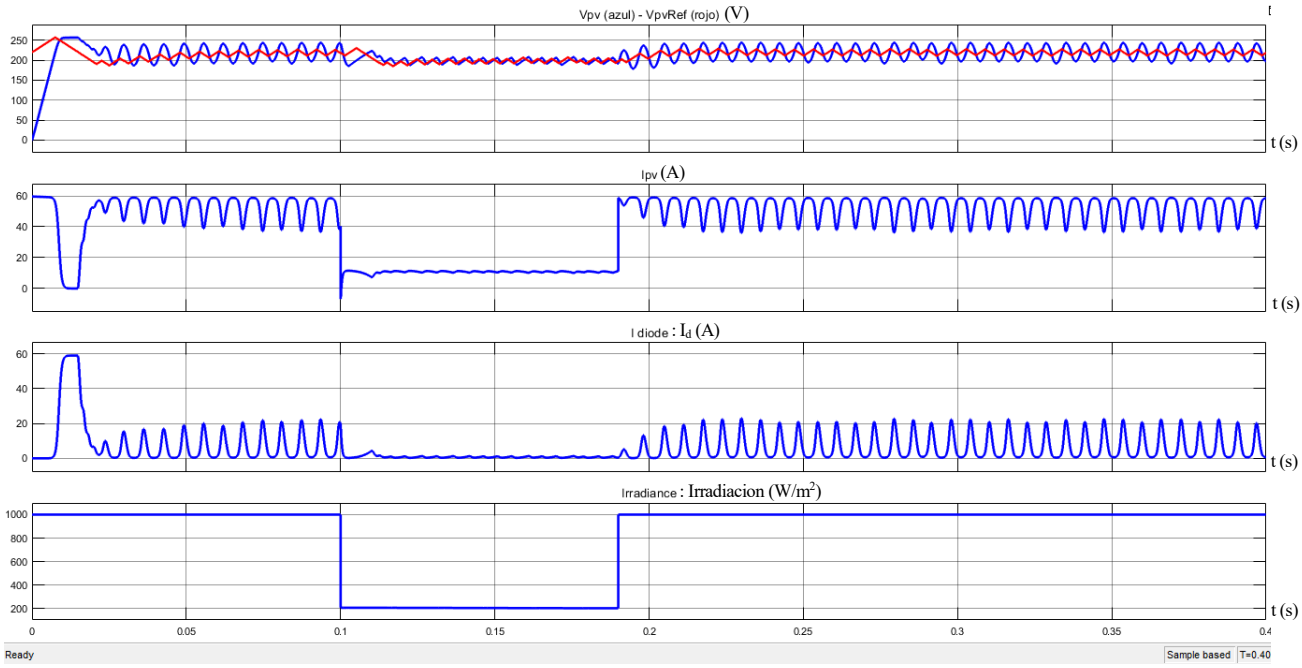


Figura 5-31. Escenario completo, variables visibles en el modelo de Simulink de la figura 5-10: Tesión (V_{pv}) e intensidad (I_{pv}) a la salida de los paneles, la intensidad del diodo (I_d) del modelo equivalente a los paneles solares y la irradiancia (Irradiacion). Algotirmo MPPT implementado con la técnica PIL. Variable de salida (V_{pvRef}) del bloque MPPT (curva roja)

Por fin hacemos uso de la técnica PIL sin ningún error. Para esta simulación hemos añadido a la primera gráfica una segunda señal, la tensión de referencia, es decir, la salida del algoritmo MPPT. Queda, por lo tanto, alcanzado un importante punto de control para poder continuar y lograr nuestro objetivo final, implementar toda la parte de control del convertidor elevador con la técnica PIL.

5.3.2 Implementación del lazo de control externo

Una vez que el bloque que realiza el algoritmo MPPT se ha desplazado fuera del bloque habilitador, necesario para realizar la técnica PIL ya que dentro del bloque habilitador hemos comprobado que no es posible, nos toca realizar la misma acción con el resto de bloques situados dentro del bloque habilitador. Dentro de dicho bloque siguen quedando el lazo de control externo y el interno.

Decidimos que la implementación de los bloques sea de izquierda a derecha, es decir, en orden de ejecución de los bloques. Por lo tanto, la siguiente sección que va a ser desplazada fuera del bloque habilitador (o "enabled") es el subsistema que contiene el lazo del control externo, por orden del criterio que hemos decidido anteriormente.

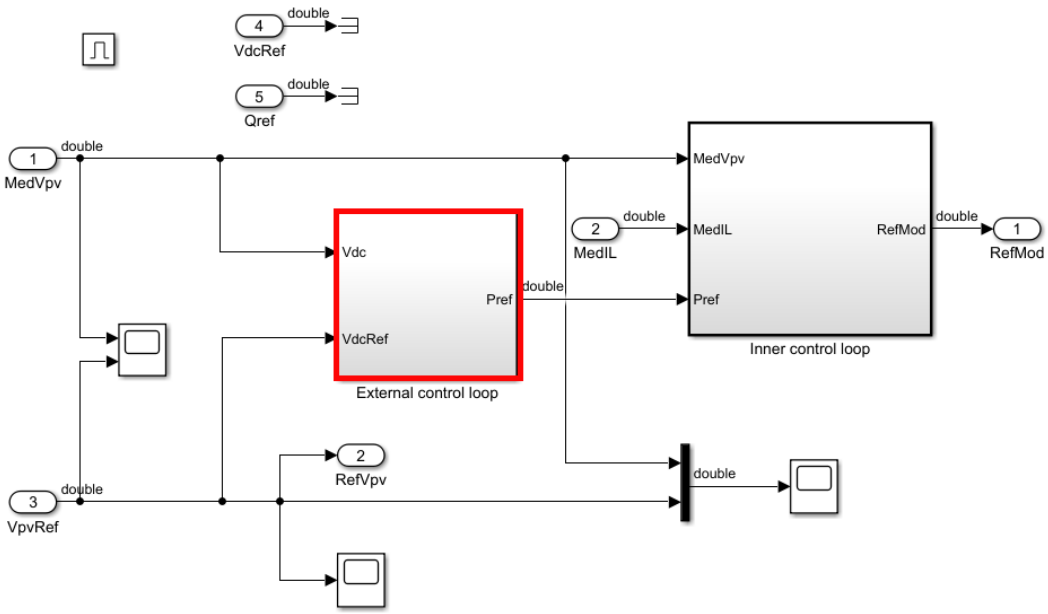


Figura 5-32. Sistema de control (*boost-control*) del convertidor DC/DC elevador: lazo de control externo + lazo de control interno.

La forma en la que vamos a proceder para realizar esta traslación de bloques es de manera conservadora, es decir, iremos trasladando bloque a bloque y cada vez que desplacemos cada bloque al subsistema donde ya se encuentra el bloque MPPT, simularemos el escenario completo para ir comprobando de manera más sencilla el correcto funcionamiento del circuito. Además, de esta manera, si ocurre algún fallo, podremos saber exactamente qué bloque es el que lo está provocando. Empezamos entonces entrando en el subsistema del lazo externo cogemos los primeros bloques, que serán el bloque multiplicador y la ganancia y lo añadimos al bloque del controlador donde ya se encuentra el bloque MPPT.

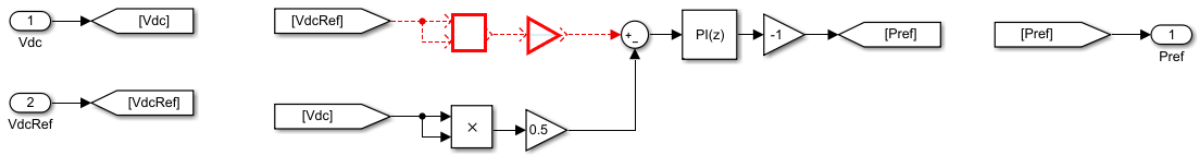


Figura 5-33. Sistema de control (*boost-control*) > Lazo de control externo: se "cortan" un multiplicador y una ganancia (V_{pv}Ref).

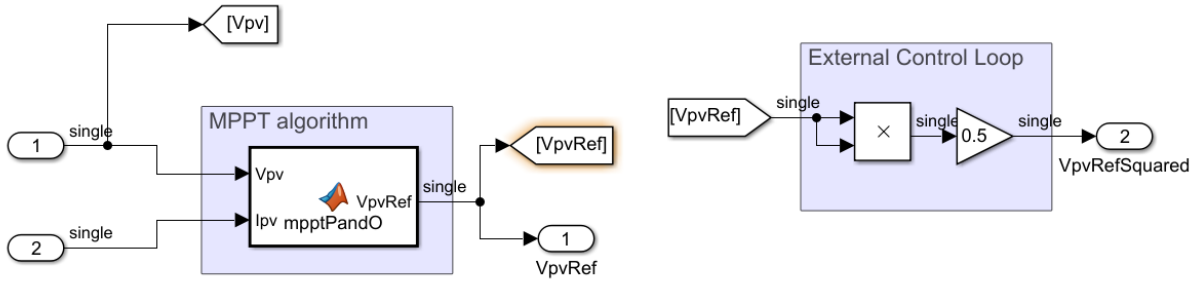


Figura 5-34. Convertidor elevador > sistema de control para implementación PIL: se "pegan" un multiplicador y una ganancia ($V_{pv}Ref$).

Al haber desplazado ciertos bloques tenemos que asegurarnos de que las señales tienen coherencia, y si no, modificarlas para que la tengan. Para esta modificación que hemos realizado, será necesario modificar ciertas señales en el nivel del convertidor elevador, del bloque habilitador y del subsistema del lazo de control externo. Entonces, la distribución de las señales para la modificación realizada debería quedar de la siguiente manera.

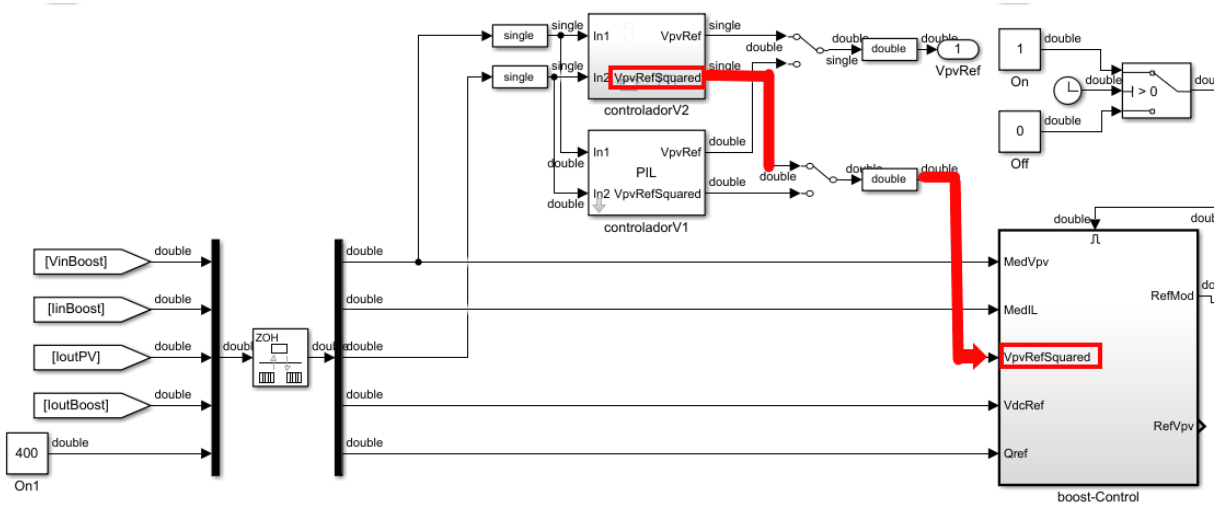


Figura 5-35. Convertidor elevador: bloque multiplicador + ganancia añadidos ($V_{pv}Ref$).

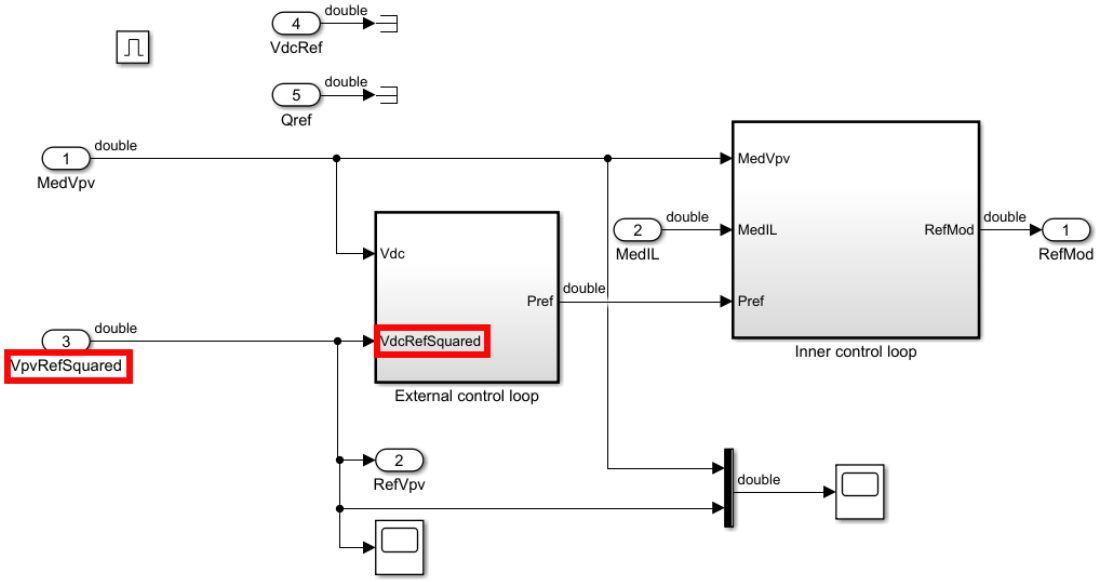


Figura 5-36. Sistema de control (*boost-control*) del convertidor DC/DC elevador: bloques multiplicador + ganancia (V_{pv}Ref) extraidos.

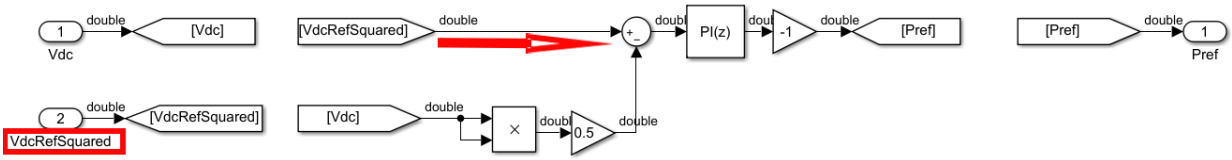


Figura 5-37. Sistema de control (*boost-control*) > Lazo de control externo: bloques multiplicador + ganancia (V_{pv}Ref) extraidos.

- Para simular con la técnica PIL, de nuevo, tenemos que hacer clic derecho sobre el subsistema que queremos implementar (se encuentra entrando al nivel del convertidor elevador) y posteriormente hacer clic en "C/C++ Code > Deploy this Subsystem to Hardware". Posteriormente hacemos clic en "Build". Una vez creado el bloque PIL en la ventana emergente, copiamos el bloque y pegamos paralelo a nuestro controlador, como hasta ahora se ha ido haciendo a lo largo del proyecto. Y, por último, pero no menos importante, ejecutamos el modelo.

Podemos tomar como referencia las últimas gráficas obtenidas o las primeras, para comprobar si las tendencias de las curvas tienen sentido. En nuestro caso, lo tienen.

Continuamos con el traslado de los bloques desde el bloque habilitador hacia el bloque en el que se encuentra el bloque MPPT y, además, parte del lazo de control. La siguiente parte será el bloque multiplicador y la ganancia que afectan a la tensión a la salida de los paneles.

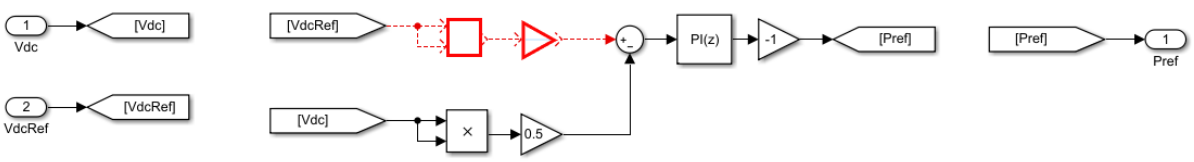


Figura 5-38. Sistema de control (*boost-control*) > Lazo de control externo: se "cortan" un multiplicador y una

ganancia (V_{pv}).

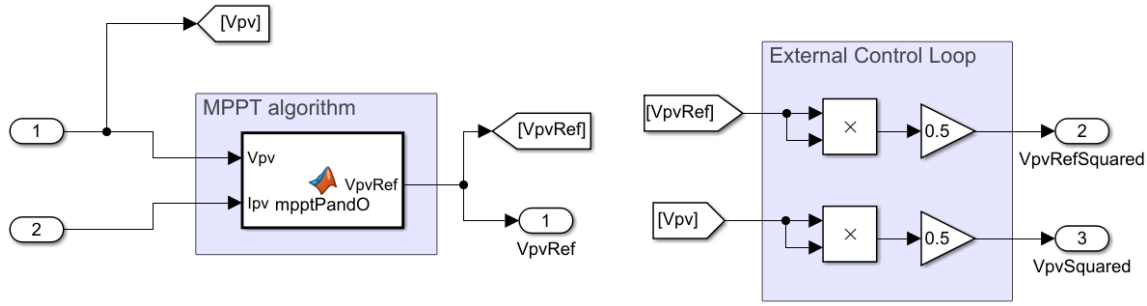


Figura 5-39. Convertidor elevador > sistema de control para implementación PIL: se "pegan" un multiplicador y una ganancia (V_{pv}).

Con motivo del desplazamiento de bloques, vuelve a ser necesario modificar la ruta de las señales afectadas por dicho desplazamiento. A continuación, mostraremos varias imágenes que indicarán de manera más clara la nueva ruta de las señales. En las imágenes podremos observar que se modifican las señales a nivel de convertidor elevador, bloque habilitador y lazo de control externo, respectivamente. Por tanto, el enrutamiento de las señales queda como sigue.

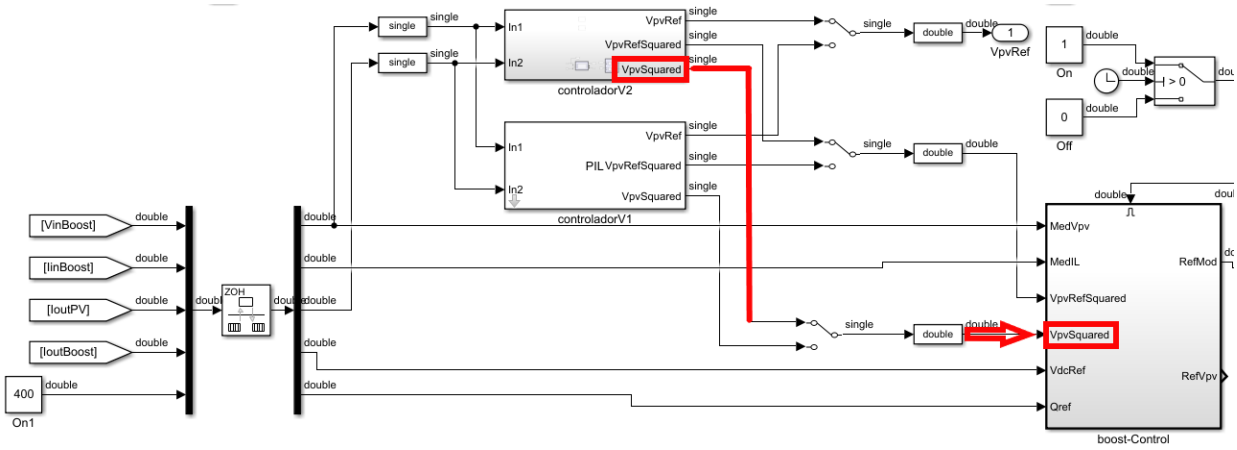


Figura 5-40. Convertidor elevador: bloque multiplicador + ganancia añadidos (V_{pv}).

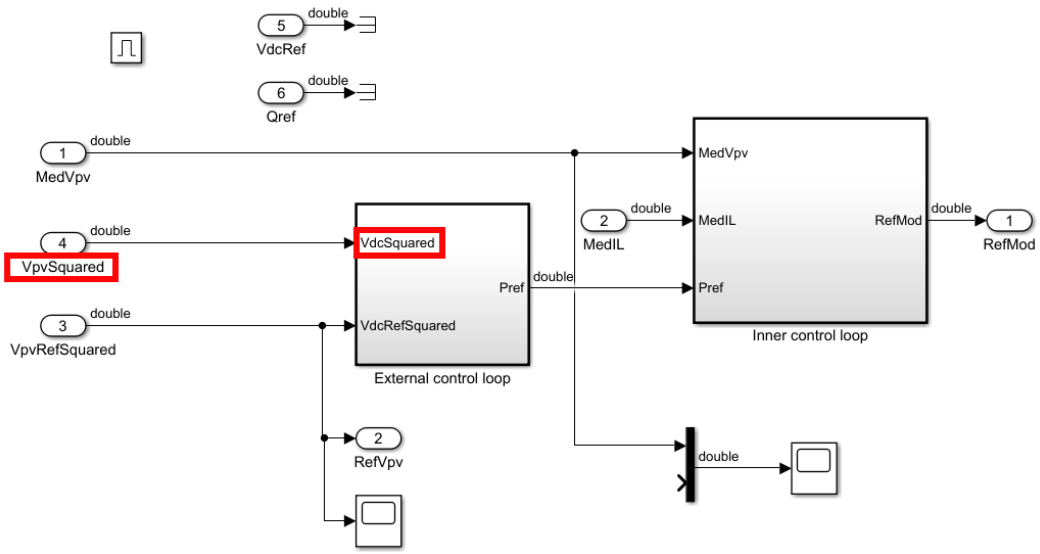


Figura 5-41. Sistema de control (*boost-control*) del convertidor DC/DC elevador: bloques multiplicador + ganancia (V_{pv}) extraidos.

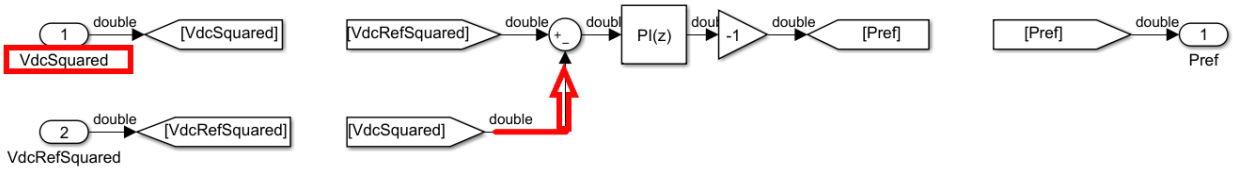


Figura 5-42. Sistema de control (*boost-control*) > Lazo de control externo: bloques multiplicador + ganancia (V_{pv}) extraidos.

Para crear el bloque PIL, procedemos de la misma manera que en el párrafo correspondiente situado en la página **73**. La arquiectura, situados en el nivel del convertidor DC/DC elevador, que debe quedar se observa en la **Figura 5-40**. Después de ejecutar el modelo, nos aseguramos que las gráficas obtenidas en esta última simulación tengan el comportamiento esperado.

Como comprobamos que el comportamiento de las gráficas tiene sentido, continuamos con el desplazamiento de los bloques desde el bloque habilitador hacia fuera del mismo. En esta traslación vamos a intentar desplazar los bloques restantes que quedan en el interior del lazo de control externo, es decir, el bloque sumador, el controlador PI (Proporcional-Integrador) y la ganancia.

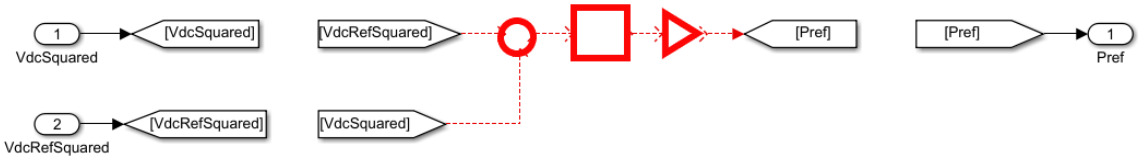


Figura 5-43. Sistema de control (*boost-control*) > Lazo de control externo: se "cortan" un sumador, un controlador PI y una ganancia.

La **Figura 5-43** se corresponde con el interior del bloque encargado del lazo de control externo. Una vez realizado el traslado de todos los bloques de su interior, solo quedarían etiquetas, que simplemente nos sirven para hacer un enrutado de señales más limpio. Es por ello, que al no tener sentido mantener el bloque del lazo de control externo, decidimos eliminarlo.

Un poco más adelante se ve cómo queda la implementación en el interior del bloque habilitador donde se podrá apreciar que el bloque sin utilidad del que hablamos, ha sido eliminado. Por otro lado, el bloque de control al que estamos trasladando los bloques, quedaría como se observa en la **Figura 5-44**. Cabe recalcar, que todos los bloques pertenecientes al lazo de control externo, ya han sido trasladados.

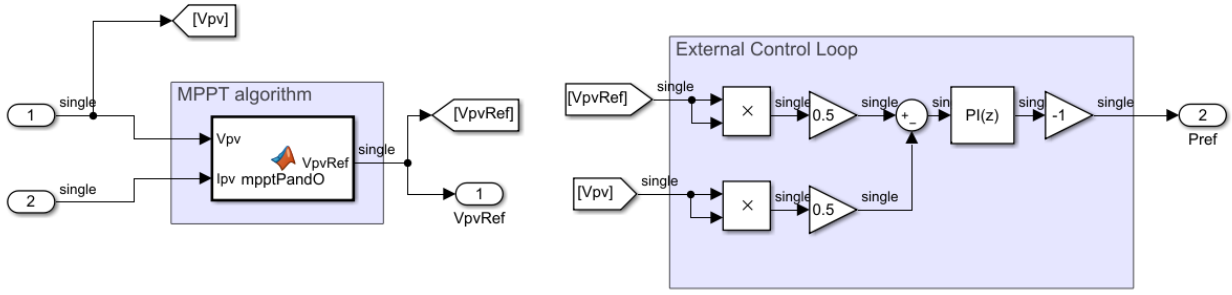


Figura 5-44. Convertidor elevador > sistema de control para implementación PIL: se "pegan" un sumador, un controlador PI y una ganancia.

Debido a este desplazamiento de bloques, ahora debemos modificar algunas señales para que las señales del circuito no varíen. Por esta razón, se adecuarán las señales a nivel del convertidor elevador y a nivel del bloque habilitador. La señal de salida de nuestro bloque controlador es ahora la potencia de referencia (P_{ref}), se trata además de la salida del bloque encargado del lazo de control externo.

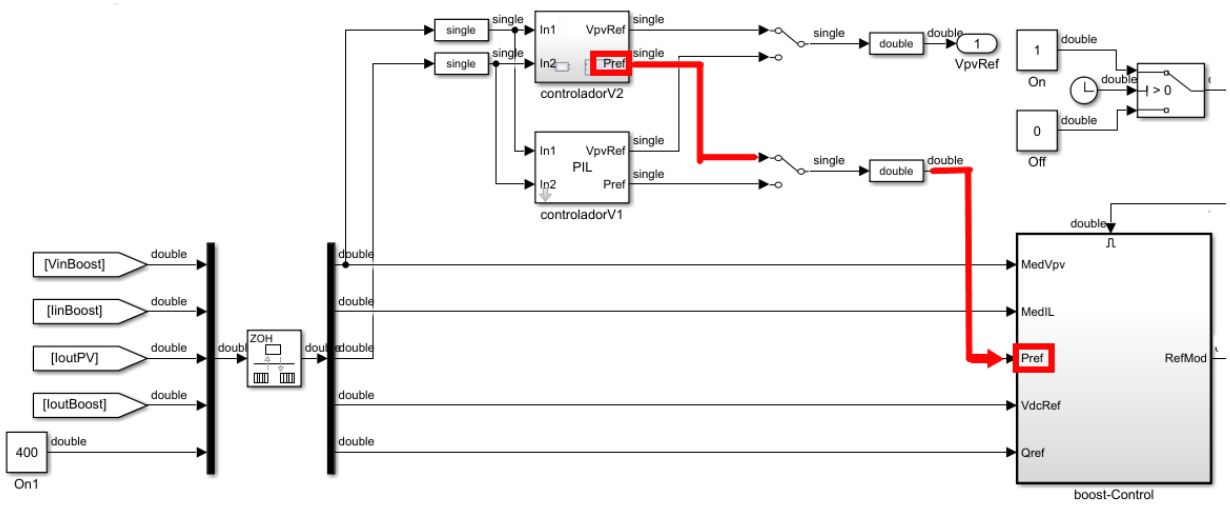


Figura 5-45. Convertidor elevador: bloque "lazo de control externo" implementado.

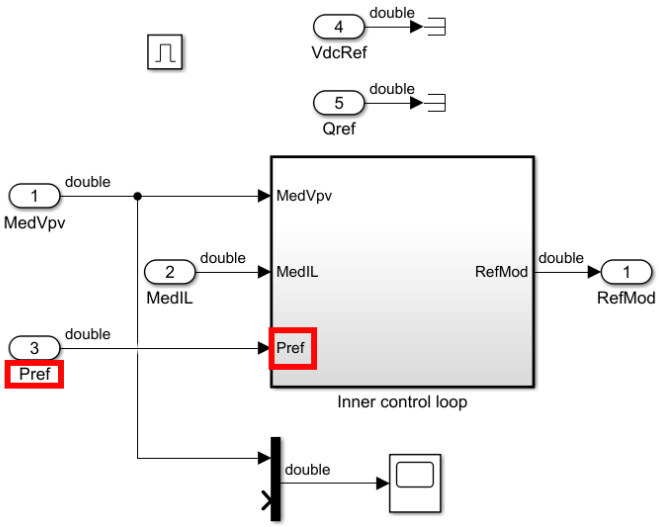


Figura 5-46. Sistema de control (*boost-control*) del convertidor DC/DC elevador: bloque un sumador + un controlador PI + una ganancia extraidos.

Como habíamos comentado anteriormente, el subsistema que se encargaba del lazo de control externo queda eliminado, ya que, todos los bloques del mismo has sido desplazados fuera. Para crear el bloque PIL, procedemos de la misma manera que en el párrafo correspondiente situado en la página **73**. Entonces, confirmamos que las gráficas responden de manera correcta ante esta última simulación con la técnica PIL.

Aunque, hasta el momento, no se muestren más gráficas referente a la tensión a salida de los paneles, la intensidad a la salida de los paneles ni la tensión de referencia (salida del algoritmo MPPT, no significa que no las tengamos en cuenta. Es más, es muy recomendable simular el escenario completo después de cada cambio que realicemos, aunque pueda parecer a prior insignificante.

Y una vez simulado, es de gran ayuda prestar atención en las gráficas de las señales comentadas en ente mismo párrafo para comprobar que tienen el comportamiento esperado. Las gráficas se mostrarán algo más adelante, cuando completemos el traslado de todos los bloques pertenecientes al bloque habilitador hacia su exterior. De esta manera podemos comparar de qué graficas hemos partido y a dónde hemos llegado después de todas las modificaciones realizadas.

5.3.3 Implementación del lazo de control interno

Como se puede comprobar en la **Figura 5-46**, en el interior del bloque habilitador solo permanece el subsistema encargado del lazo interno de control y sus correspondientes bloques. Por tanto, nos disponemos a realizar la traslación de dichos bloques uno a uno. Vamos a comenzar con el bloque sumador que se encarga de restar a la señal de intensidad proveniente de la bobina la señal potencia de referencia, esta última es la señal salida del bloque inmediatamente anterior, el lazo de control externo que ya se encuentra trasladado fuera del bloque habilitador.

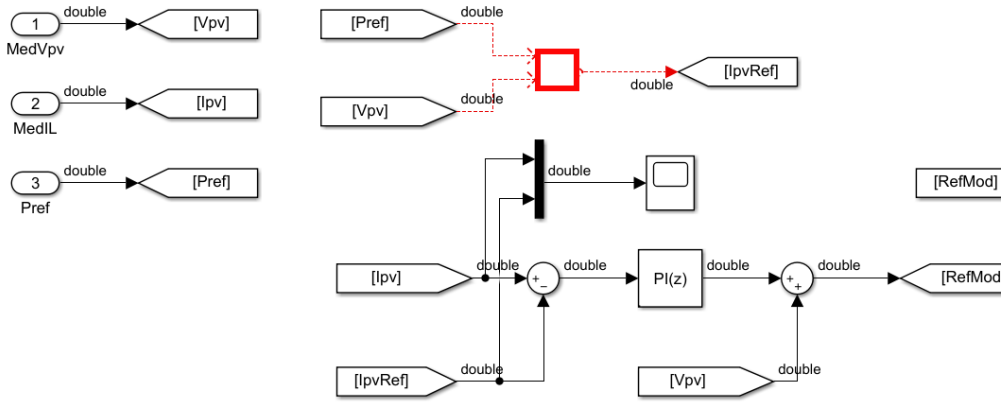


Figura 5-47. Sistema de control (*boost-control*) > Lazo de control interno: se "cortan" un bloque producto.

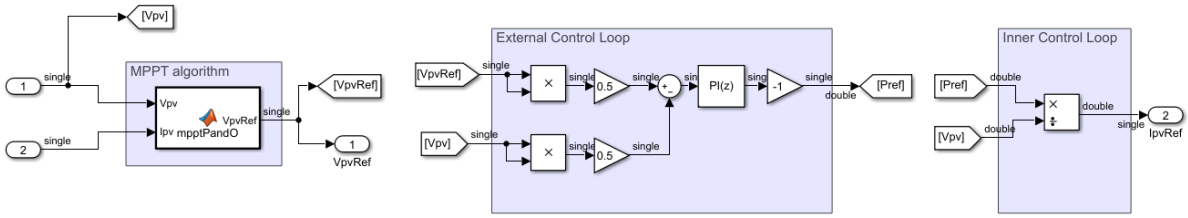


Figura 5-48. Convertidor elevador > sistema de control para implementación PIL: se "pegan" un bloque producto.

Como consecuencia de este último desplazamiento del bloque producto es necesario modificar la disposición de las señales para que el circuito en su conjunto tenga coherencia. Las siguientes tres imágenes mostrarán la disposición con la que quedan los bloques y las señales para poder simular con la técnica PIL. Los bloques que se encuentran dentro del controlador que se usará para dicha técnica son los de la **Figura 5-48**. En las próximas tres imágenes, la señal que va a aparecer es la intensidad de referencia, la salida del bloque que acabamos de desplazar.

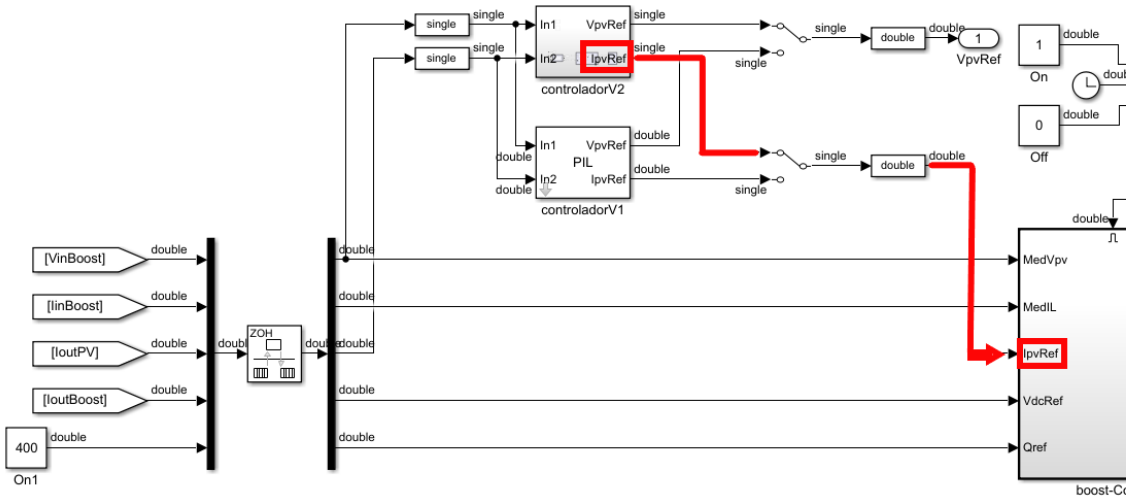


Figura 5-49. Convertidor elevador: bloque producto añadido.

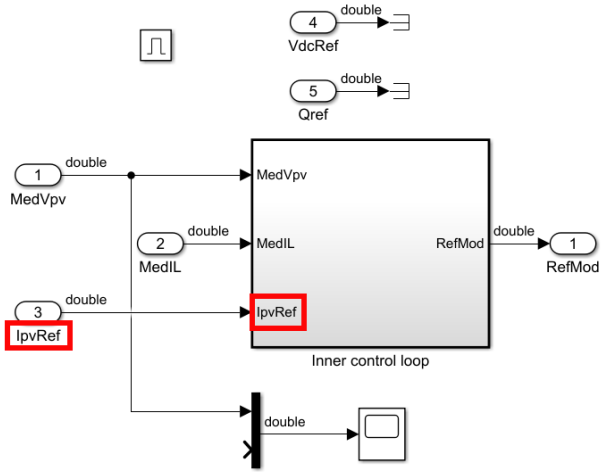


Figura 5-50. Sistema de control (*boost-control*) del convertidor DC/DC elevador: bloque producto extraido.

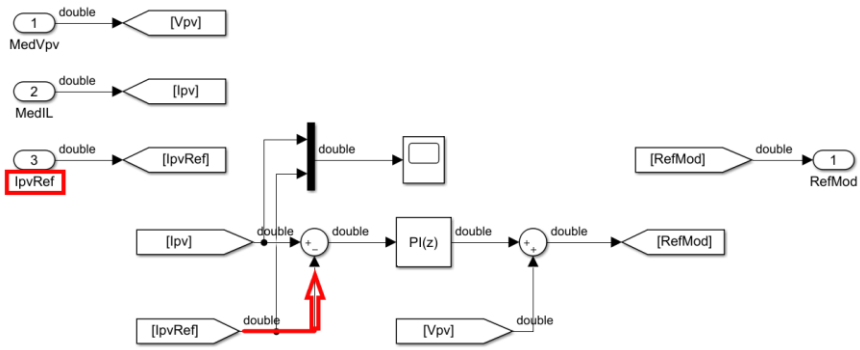


Figura 5-51. Sistema de control (*boost-control*) > Lazo de control externo: bloque producto extraido.

Por supuesto, para la creación del bloque PIL, procedemos de la misma manera que en el párrafo correspondiente situado en la página **73**. Tras simular con la técnica PIL, se debe confirmar que las gráficas responden de la manera esperada.

Observando en el interior del subsistema encargado del lazo de control interno nos percatamos de que el próximo bloque según nuestro criterio (de izquierda a derecha) es el sumador. En las próximas imágenes veremos el desplazamiento del bloque y la modificación de las señales que se puedan ver afectadas.

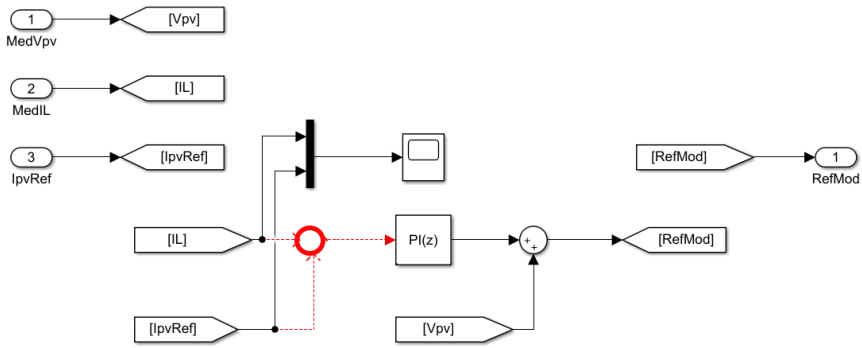


Figura 5-52. Sistema de control (*boost-control*) > Lazo de control interno: se "cortan" un bloque sumador.

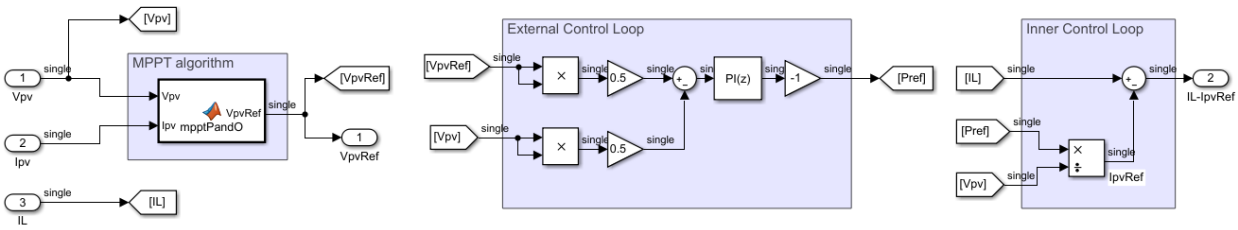


Figura 5-53. Convertidor elevador > sistema de control para implementación PIL: se "pegan" un bloque sumador.

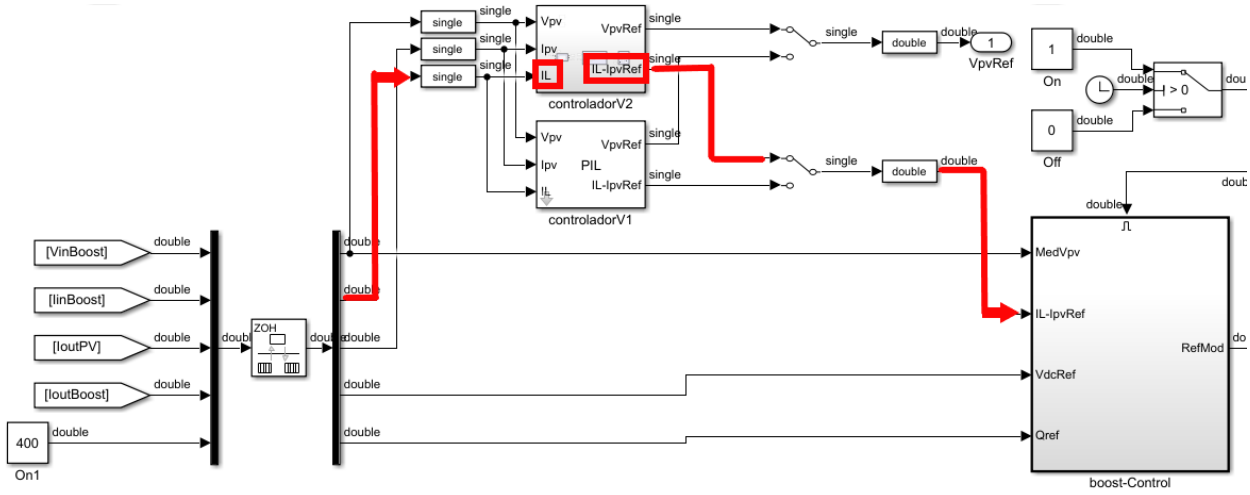


Figura 5-54. Convertidor elevador: bloque sumador añadido.

En esta ocasión, además de modificar la señal de salida del bloque controlador necesitamos añadir una nueva entrada, la intensidad de la bobina situada al inicio del circuito del convertidor elevador.

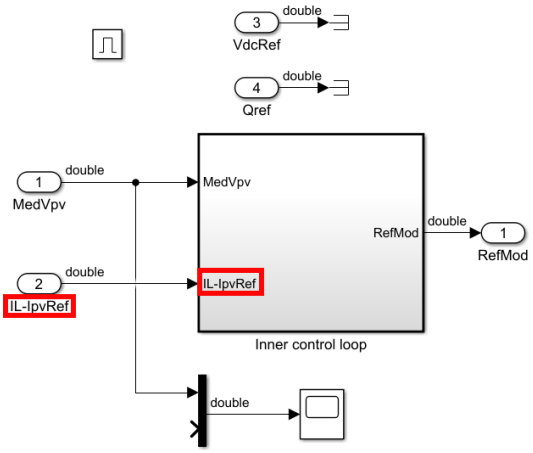


Figura 5-55. Sistema de control (*boost-control*) del convertidor DC/DC elevador: bloque un sumador extraido.

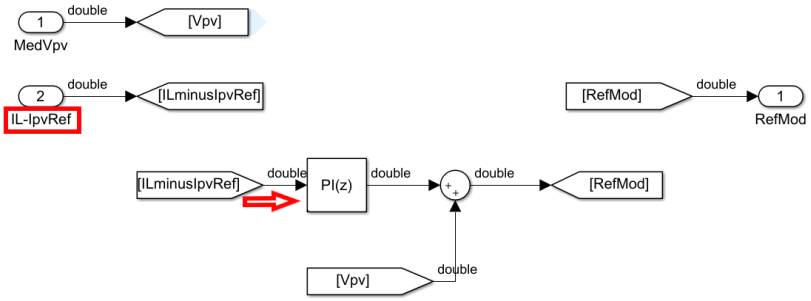


Figura 5-56. Sistema de control (*boost-control*) > Lazo de control externo: bloque sumador extraido.

Como de costumbre, creamos el bloque PIL y simulamos con el mismo. Para la creación de dicho bloque, debemos hacer clic derecho sobre el controlador en cuestión que queremos implementar con la técnica PIL. Posteriormente hacer clic en "C/C++ Code > Deploy this Subsystem to Hardware". A continuación, clic en "Build". Por último, copiamos el bloque PIL que aparece de la ventana emergente, lo pegamos en nuestro escenario ejecutamos el escenario. Confirmamos que las gráficas responden de manera correcta ante esta última simulación con la técnica PIL.

Observando la **Figura 5-56**, se ve que aún permanecen dos bloques en el interior del subsistema habilitador. El siguiente bloque en ser desplazado será el controlador PI.

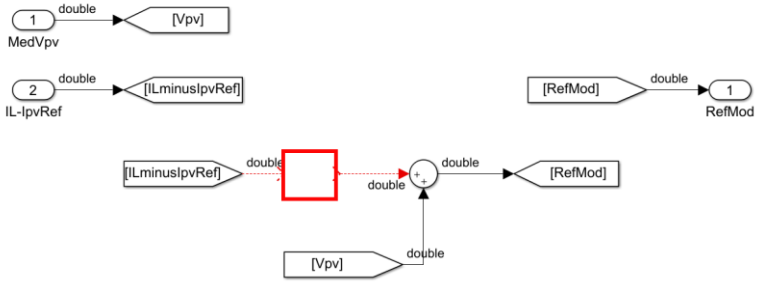


Figura 5-57. Sistema de control (*boost-control*) > Lazo de control interno: se "cortan" un bloque controldor PI.

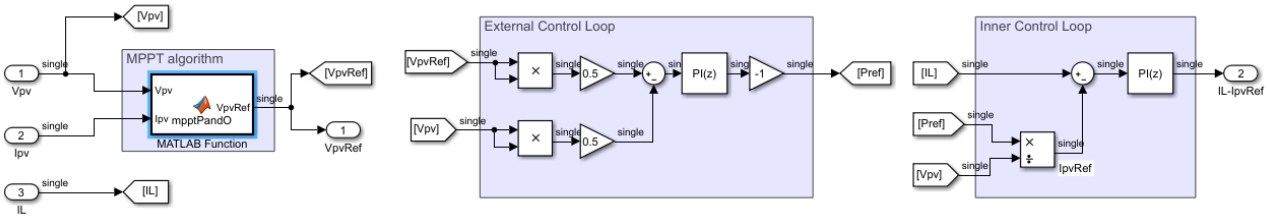


Figura 5-58. Convertidor elevador > sistema de control para implementación PIL: se "pegan" un bloque controldor PI.

Dentro del nivel del convertidor no se ha realizado ningún cambio, por lo que sirve de referencia la última imagen de dicho nivel. Por otra parte, dentro del bloque habilitador, el único cambio realizado ha sido el desplazamiento al exterior del bloque, el bloque controlador PI. Por lo tanto, ahora no aparece dicho bloque controlador PI.

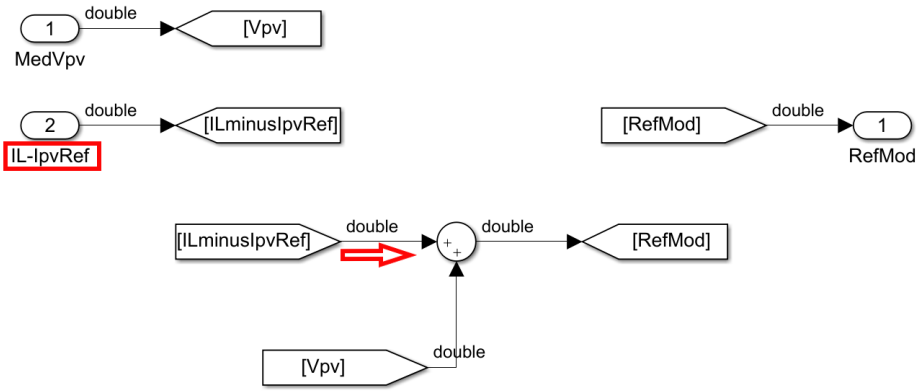


Figura 5-59. Sistema de control (*boost-control*) > Lazo de control externo: bloque sumador extraido.

Sin embargo, las gráficas de las señales de referencia para nuestro caso (V_{pv}, I_{pv} e V_{ref}) **no** siguen el comportamiento esperado, parece que el sistema de control se vuelve inestable. Por lo que no podemos proseguir con el traslado de bloques dentro del subsistema habilitador. Primeramente, debemos resolver este incidente.

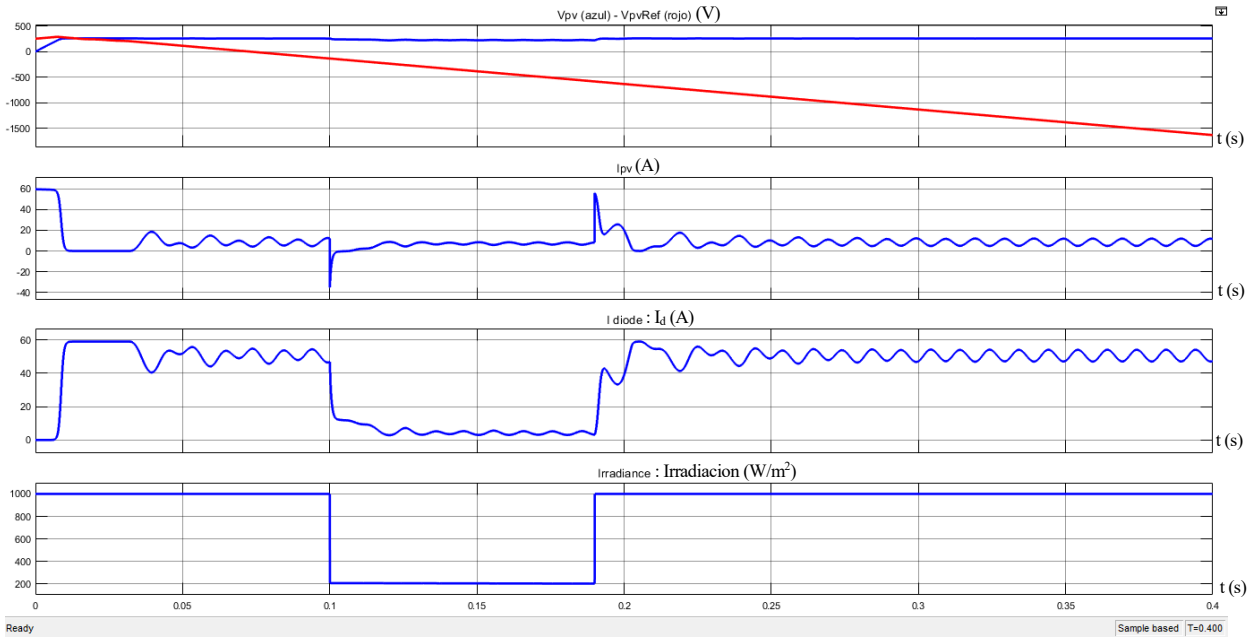


Figura 5-60. Escenario completo, variables visibles en el modelo de Simulink de la figura 5-10: Tesión (V_{pv}) e intensidad (I_{pv}) a la salida de los paneles, la intensidad del diodo (I_d) del modelo equivalente a los paneles solares y la irradiancia (Irradiación). Error en el sistema de control. Variable de salida (V_{pvRef}) del bloque MPPT (curva roja)

En la **Figura 5-60** se muestran los resultados de la arquitectura completa tras intentar implementar el bloque del controlador PI del lazo de control interno en la técnica PIL. Las variables visibles en dicha figura se corresponden con aquellas que se encuentran en el modelo de Simulink de la **Figura 5-10**. Por un lado, se ve la tesión (V_{pv}) y la intensidad (I_{pv}) a la salida de los paneles fotovoltaicos. Por otro lado, la intensidad del diodo (I_d), el cual forma parte del modelo equivalente a los paneles solares. Además, se encuentra la irradiancia (Irradiacion) la cual es señal de entrada de los paneles solares. Por último, se encuentra la señal V_{pvRef} (curva roja en la **Figura 5-60**), la cual es la señal de salida del algotirmo MPPT.

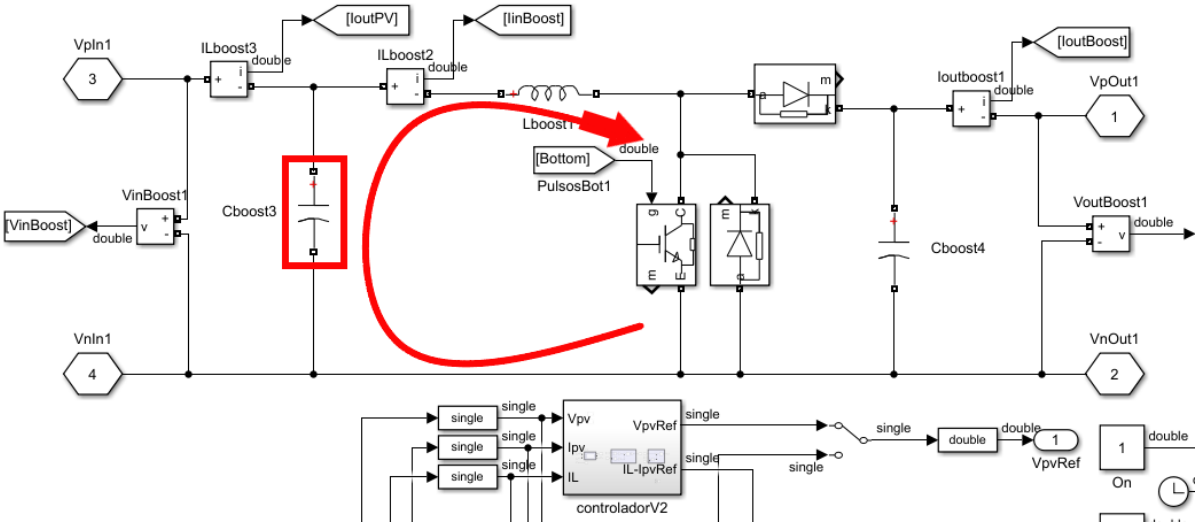


Figura 5-61. Convertidor elevado: efecto del condensador "Cboost3".

En la **Figura 5-61**, se observa el circuito que hace la propia función de convertidor elevador podemos apreciar que, en la malla situada a la izquierda, la tensión inicial depende solamente del condensador **Cboost3** dado que el transistor se encuentra abierto. Dicha tensión inicial no está inicializada a ningún valor, por lo que se puede

suponer que es tomada como cero. Dicho valor nulo, al ser usado como divisor y luego ser integrado por el controlador PI es muy probable haga inestable el sistema.

Como solución se decide inicializar la tensión de la capacidad a un valor distinto de cero, por ejemplo, a 100 voltios. Para llevar a cabo esta acción basta con hacer doble clic sobre la capacidad. A continuación, marcar la opción "set the initial capacitor voltage" y colocar el valor "100" en la casilla de los voltios.

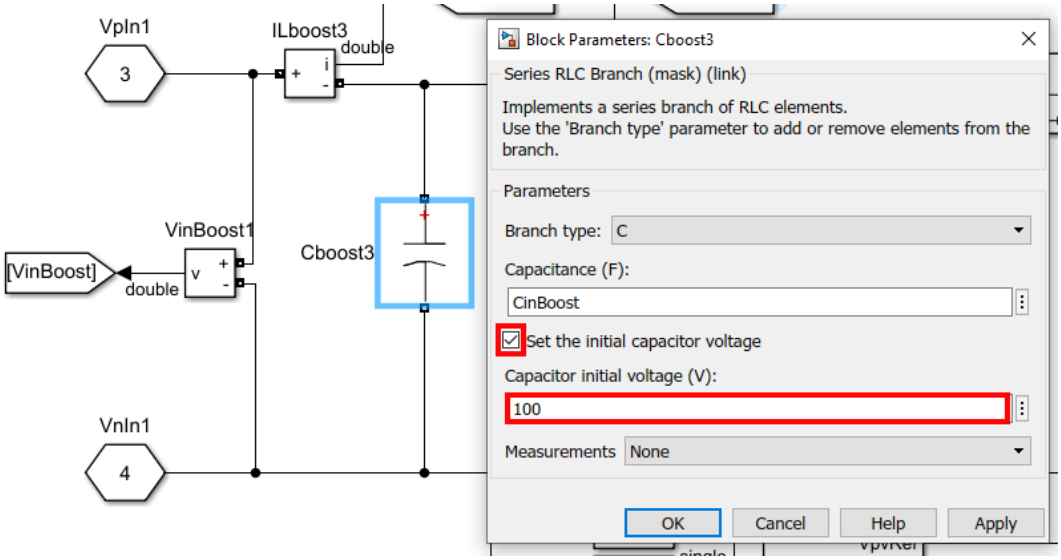


Figura 5-62. Convertidor elevado: parámetros del condensador "Cboost3".

Una vez se ha modificación la tensión inicial de la capacidad, las gráficas de las señales de referencia vuelven a mostrar la tendendia esperada, sin embargo, cabe destacar que la señal V_{pvRef} necesita algunas décimas de segundo más para alcanzar el régimen permanente. Por lo tanto, proseguirnos con nuestra tarea de desplazamiento de bloques.

Como podemos observar en la **Figura 5-63**, el último elemento que permanece dentro del subsistema encargado del control interno y por consecuente, dentro del bloque habilitador, es un bloque sumador. La señal de salida de este bloque es la tensión PWM que utilizaremos para crea la señal PWM, encargada de activar y desactivar los transistores IGBT.

A continuación, desplazaremos el bloque sumador y modificaremos las señales que se vean afectadas. Mostraremos, además, cual serán y cómo estarán distribuidos los componentes que finalmente formarán parte de nuestro escenario a nivel de convertidor elevador, a nivel de controlador (el cual implementaremos con la técnica PIL) y nivel de bloque habilitador.

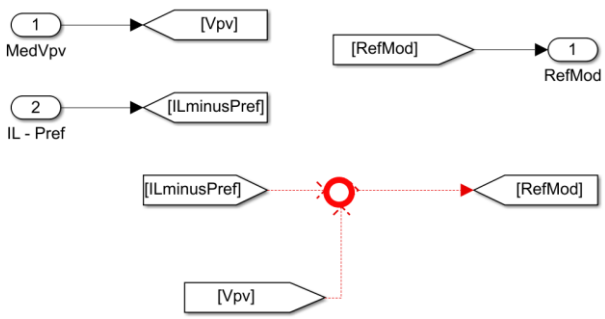


Figura 5-63. Sistema de control (*boost-control*) > Lazo de control interno: se "corta" un bloque controldor PI.

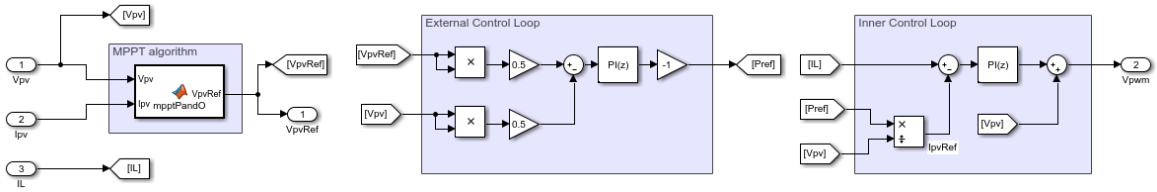


Figura 5-64. Convertidor elevador > sistema de control para implementación PIL: se "pega" un bloque sumador.

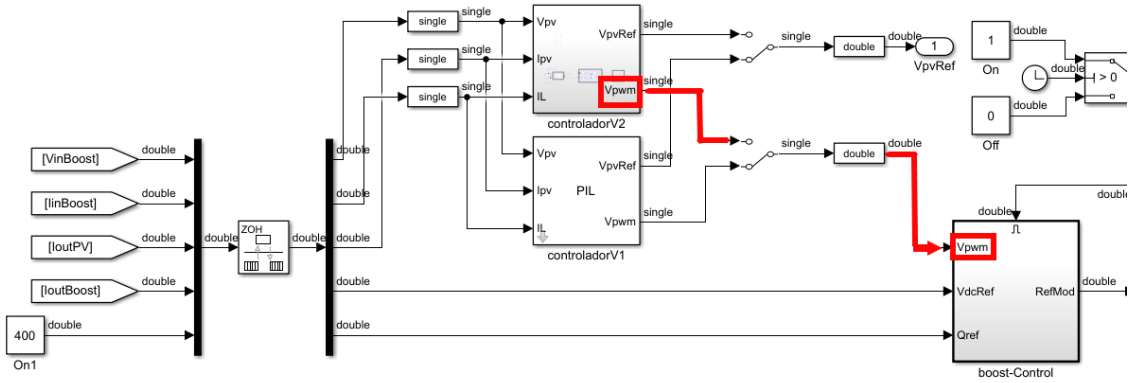


Figura 5-65. Convertidor elevador: bloque "lazo de control interno" implementado.

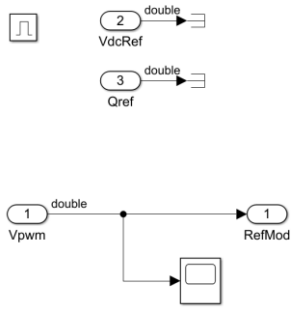


Figura 5-66. Sistema de control (*boost-control*): todos los bloques desplazados al sistema de control para la imlementación PIL.

Podemos ver en la **Figura 5-66** que en el interior del bloque habilitador simplemente permanecen dos señales que son terminadas, puesto que no se utilizarán y dos bloques más, uno de entrada y otro de salida de la señal. Esta disposición tiene sentido, pues decidimos desplazar toda la parte de control fuera del bloque habilitador para poder ejecutar la técnica PIL con el controlador del convertidor elevador.

Para poder simular con la técnica PIL, previamente tenemos que crear el bloque PIL. Entonces, volvemos a repetir el proceso que hemos estado llevando a cabo para poder implementar el controlador en la placa de desarrollo. En esta implementación se incluye la generación de código y la descarga de dicho código en la placa. Por lo tanto, el siguiente paso es crear el bloque PIL.

Para ello, debemos hacer clic derecho sobre el controlador en cuestión que queremos implementar con la técnica PIL. Posteriormente hacer clic en "C/C++ Code > Deploy this Subsystem to Hardware". A continuación, clic en "Build". Por último, copiamos el bloque PIL que aparece de la ventana emergente, lo pegamos en nuestro escenario ejecutamos el escenario. A continuación, se observa las gráficas de las señales

principales para confirmar que las gráficas responden de manera correcta ante esta última modificación del escenario.

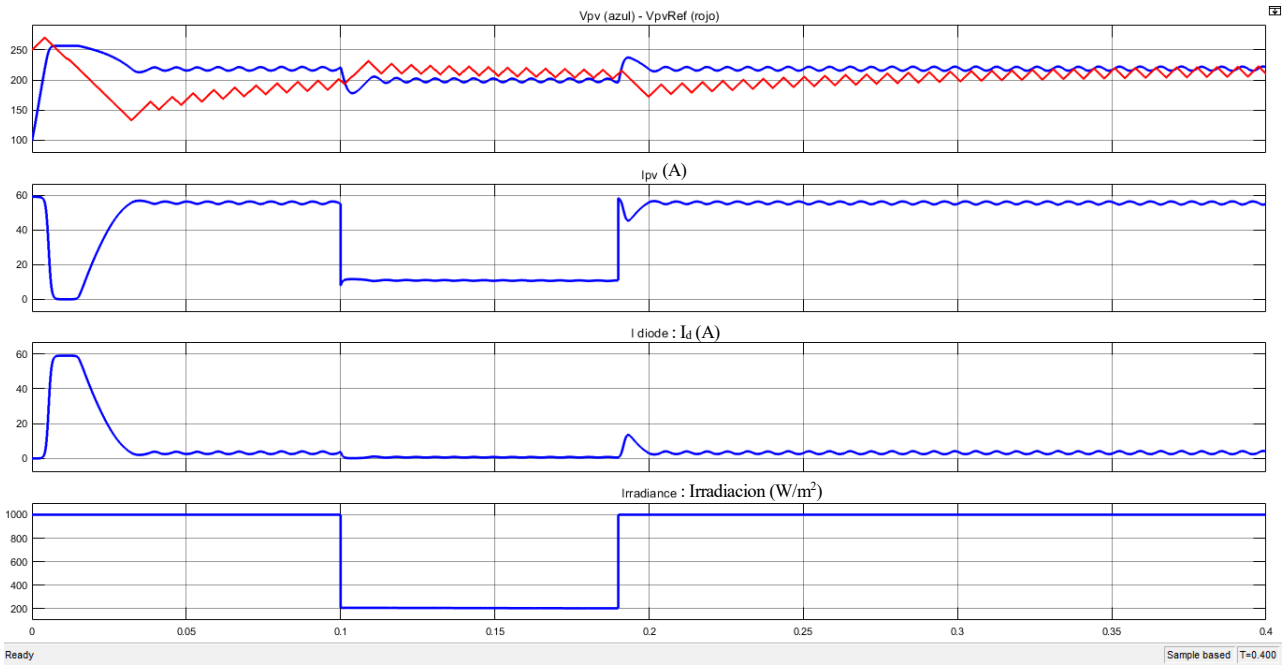


Figura 5-67. Escenario completo, variables visibles en el modelo de Simulink de la figura 5-10: Tesión (V_{pv}) e intensidad (I_{pv}) a la salida de los paneles, la intensidad del diodo (I_d) del modelo equivalente a los paneles solares y la irradiancia (Irradiacion). Sistema de control implementado con la técnica PIL. Variable de salida (V_{pvRef}) del bloque MPPT (curva roja)

A la vista de los resultados que se han representado, se concluye que la simulación del controlador del convertidor elevador mediante la técnica PIL ha sido un éxito. Es cierto que se ha tenido que modificar la arquitectura original, pero es comprensible ya que a priori el escenario no se creó con el objetivo que se ha pretendido en este proyecto. Se puede observar como la señal de irradiación pasa de los 1000 W/m² a 200 W/m² cuando el tiempo de simulación llega a 0.1s y vuelve a los 1000 W/m² para el tiempo de simulación igual a 0.19s.

Cabe destacar que, dando tiempo suficiente en el régimen transitorio, la variable de salida del bloque MPPT (V_{pvRef},curva roja en la **Figura 5-67**) es capaz de alcanzar la tensión a la salida de los paneles fotovoltaicos. Si bien es cierto, antes de que el tiempo de simulación llegue a los 0.05 segundos, se observa que la tensión de referencia no termina de realizar bien el seguimiento, pero finalmente consigue adaptarse el algoritmo MPPT. Además, se observa que la intensidad a la salida de los paneles (I_{pv}) llega a ser cero. Sin embargo, comparando con la **Figura 5-11**, se observa que tiene un comportamiento prácticamente idéntico.

Por lo tanto, el único cambio a destacar, es el tiempo de retardo con el que se adapta el algoritmo MPPT antes de los 0.05 segundos. Tiene sentido que haya diferencia en ese aspecto, pues se recuerda que originalmente existía un tiempo predefinido (TenablePV) para la puesta en funcionamiento del algoritmo MPPT a los 0.02 segundos. Sin embargo, para la implementación de la técnica PIL vimos necesario suprimir este tiempo para que comenzara al inicio de la simulación.

6 CONCLUSIONES DEL TRABAJO Y LÍNEAS DE INVESTIGACIÓN FUTURA

En este Trabajo Fin de Grado el alumno ha analizado y estudiado el funcionamiento del sistema asignado originalmente cuyo objetivo es la generación eléctrica aislada basado en energía fotovoltaica con batería. Para ello, se ha procedido a dividir el modelo en varias partes bien diferenciadas tales como, los paneles fotovoltaicos, el convertidor DC/DC elevador, el algoritmo de control MPPT, la batería, el convertidor DC/DC bidireccional y el convertidor DC/AC inversor. Esta parte de estudio es vital para poder entender el porqué de la tendencia de algunas variables y poder atajar con mayor eficacia los posibles problemas que se vayan presentando.

Una vez comprendida la teoría de la electrónica de potencia, al alumno le da lugar a adquirir el software necesario para implementar la técnica de simulación PIL. Para esta tarea, el alumno ha debido descargar varios softwares como CSS, ControlSUITE, MATLAB – Simulink y algún paquete específico de este último para la placa de desarrollo que se le asigna al alumno en este proyecto, la F28379D.

Además, el alumno debe cerciorarse de la correcta configuración del software para el hardware disponible. A través de la configuración, se ha observado el infinito catálogo disponible tanto para software como para hardware, por esta razón la utilidad de este proyecto no reside en un software de simulación o en una placa de desarrollo en concreto, sino que es escalable a otro tipo de modelos. Para esta configuración, se ha comprobado que es de mucha utilidad interactuar con los ejemplos que nos proporcionan los distintos softwares, especialmente aquellos creados específicamente para comprobar la correcta configuración de los mismos.

El alumno es este instante es capaz de abordar la correspondiente implementación de la técnica PIL. Se ha investigado el proceso necesario para adaptar un sistema a dicha técnica, entre las tareas principales para esta adaptación se destaca el cambio de domino del tiempo continuo al tiempo discreto y la conversión del tipo de dato a la entrada y salida tanto del bloque de control como en el bloque PIL. Se ha comprobado que la implementación de la técnica es delicada para dependiendo qué escenario. Sin duda, se ha observado como el código ejecutado en la placa de desarrollo ha sido capaz de replicar la tendencia de las curvas del sistema cuando actuaba el controlador original.

6.1 Resumen de las principales dificultades y modificaciones realizadas

En esta sección se va a profundizar un poco más en esas partes del proyecto que ha requerido una pausa para poder resolverlas y poder seguir avanzando.

6.1.1 Configuración inválida para ejecutar técnica PIL

Debido a la complejidad del escenario y al alto número de bloques que componen nuestro escenario es posible que se nos escape algún detalle que no hayamos tenido en cuenta o algún parámetro configurado inadecuadamente. Es posible, por tanto, que este error que pudiéramos pasar por alto nos impidiese poder implementar nuestro subsistema de control en nuestra placa. Uno de los errores que podrían surgirnos es que la configuración actual no permita la conectividad con el software CCS.

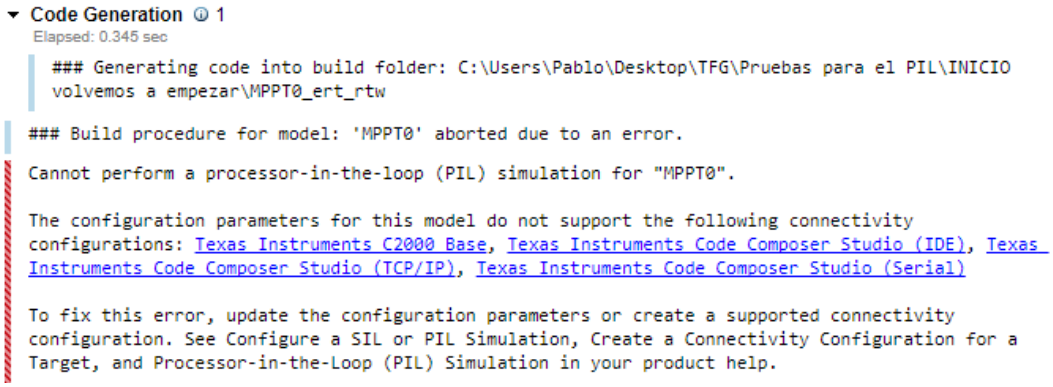


Figura 6-1. Error al simular con la técnica PIL debido a configuración de conectividad.

Si por algún motivo después de seguir los pasos para la configuración de los parámetros de un escenario complejo obtenemos errores similares a los de la imagen anterior, recomiendo encarecidamente actuar de la siguiente manera. Para empezar, nos trasladamos al apartado **4.3.1** donde se trata con el ejemplo **c2000_pil_block** que nos proporciona Matlab. Escribiendo el mismo nombre del ejemplo en la ventana de comandos de la aplicación Matlab, se consigue abrir el ejemplo.

Por otro lado, este ejemplo es elegido porque es la estrategia más sencilla de configurar de manera óptima cualquier escenario en el que se vaya a implementar la técnica PIL, debido a la poca cantidad de bloques que contiene dicho ejemplo. Entonces, una vez tenemos el escenario simple de ejemplo bien configurado, lo abrimos. Seguidamente, abrimos el escenario con el cual pudiéramos tener problemas de configuración y trasladamos todos sus bloques al escenario simple de ejemplo (**c2000_pil_block)** que sabemos con certeza que está bien configurado y que abrimos.

6.1.2 Configuración del *baud rate* de la placa y de la versión del compilador del CCS

Como hemos comentado anteriormente, el "baud rate" es la cantidad de bits por segundos que vamos a transmitir por el puerto serie. Gracias al contraste de información en MathWorks y a la guía del profesor Sergio Vázquez conseguimos establecer una ratio de 115200 bits/s con el que no tendremos conflicto para ninguno de los ejemplos mostrados a lo largo de este proyecto.

Además, tras hablar con el mismo profesor y dos de sus alumnos, llegamos a la conclusión de que una versión estable del compilador es clave para poder usar CCS correctamente y que no haya problemas de incompatibilidad a la hora de generar el código.

6.1.3 Adaptación del escenario definitivo para el uso de la técnica PIL

No solo ha sido necesario realizar las modificaciones esenciales para poder usar la técnica PIL, sino que ha habido una adaptación y una investigación detrás que ha permitido su ejecución. Entre algunos de los contratiempos se destacan:

- Incompatibilidad para ejecutar la técnica PIL dentro de un bloque con semántica "enable". Se toma como solución extraer todos los bloques que existen en su interior.
- Incompatibilidad de ciertos tipos de bloques personalizados para la ejecución del algoritmo MPPT dentro del bloque habilitador. Se opta por el bloque MATLAB Function.
- Incompatibilidad con las señales bus situadas en los bloques "display". Se resuelve colocando un conversor "Bus to vector".

Sin duda ha sido una de las tareas más desafiantes del propio proyecto dado que el escenario original no se creó con la intención de implementar la técnica PIL. Estas adaptaciones destacadas se llevan a cabo en la sección **5.3**.

6.1.4 Condición inicial del condensador situado en el convertidor DC/DC elevador

La tensión inicial debe ser distinta de cero debido a la aparición de las operaciones división e integración en presencia de la técnica PIL. De no cambiar la tensión inicial del condensador, la salida del bloque MPPT (V_{pvRef}) tendería a menos infinito, como podemos apreciar en la correspondiente **Figura 5-60** perteneciente a la sección **5.3**.

6.2 Líneas de investigación futura

En la presenta sección se va a analizar los límites de la técnica PIL. Por un lado, se va a tratar con el código generado automáticamente en la implementación de la técnica PIL y qué se puede crear con él. Por otro lado, se van a emplear los periféricos del DSP F28379D de la familia C200 del fabricante TI.

6.2.1 Código del controlador reprogramable

Como hemos podido comprobar a lo largo del proyecto, la realización de la técnica PIL conlleva la generación de un código y su posterior descarga en la placa de desarrollo. Una vez generado ese código pueden surgir varias preguntas, como, por ejemplo: ¿tenemos acceso a dicho código?, ¿podemos modificar líneas del código? O ¿podemos crear un archivo **.c** desde cero?

Supongamos que ya hemos implementado la técnica de simulación PIL, por lo tanto, el bloque PIL ya tiene que esta generado. Entonces, en nuestro directorio de trabajo en Matlab debe existir un directorio con nombre ***controller*_ert_rtw.c** donde *controller* es el nombre que le hemos asignado al bloque de control que queremos implementar con la técnica PIL. El archivo **.c** es accesible haciendo doble clic sobre el mismo.

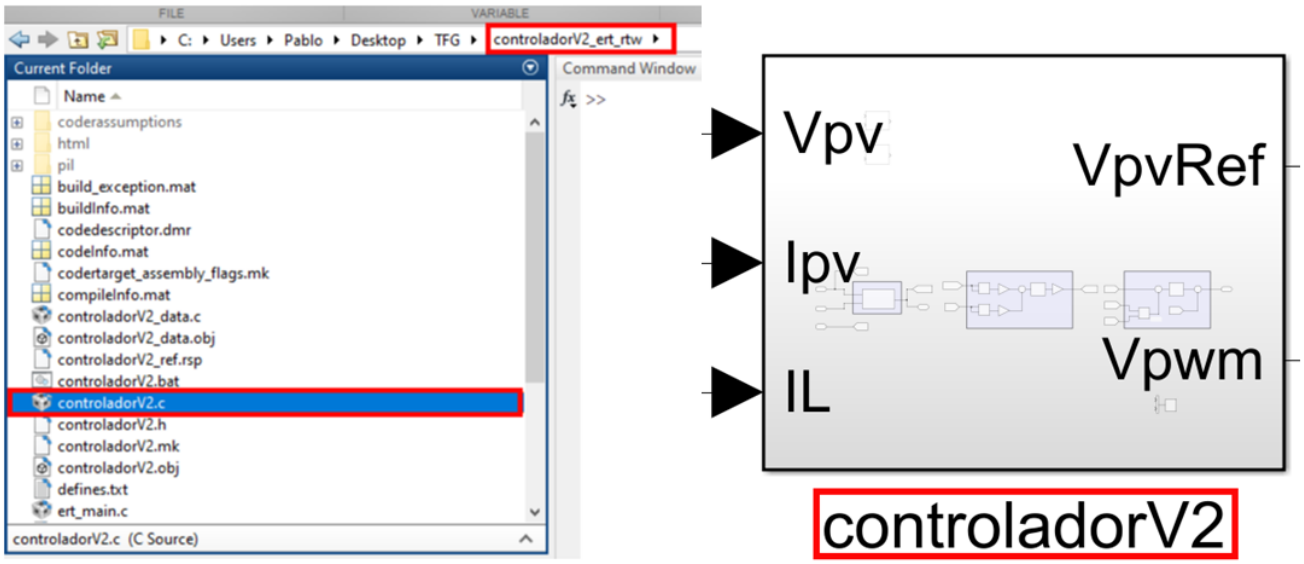


Figura 6-2. Acceso al código generado por la técnica PIL.

Vamos a abrir el archivo **.c** del controlador. Desde la línea 38 hasta la 86 del código está codificado el bloque Matlab Function, el encargado de realizar el algoritmo MPPT. A continuación, mostraremos a la izquierda parte del código original y a la derecha parte del código modificado. En el código modificado hemos comentado todo el código referente al algoritmo MPPT. Además, hemos sustituido la salida del algoritmo original para que se convierta la suma de las entradas del bloque (tensión e intensidad a la salida de los paneles fotovoltaicos).

Igualmente, adjuntaremos las imágenes correspondientes a la salida del bloque PIL con ambos códigos. La señal de color rojo es una entrada del bloque MPPT, se trata de la intensidad a la salida de los paneles (I_{pv})(A), la de color morada es la segunda entrada del bloque MPPT, se corresponde a la tensión de salida de los paneles (V_{pv})(V). Por último, la señal de color azul es la salida del bloque MPPT (V_{pvRef})(V).

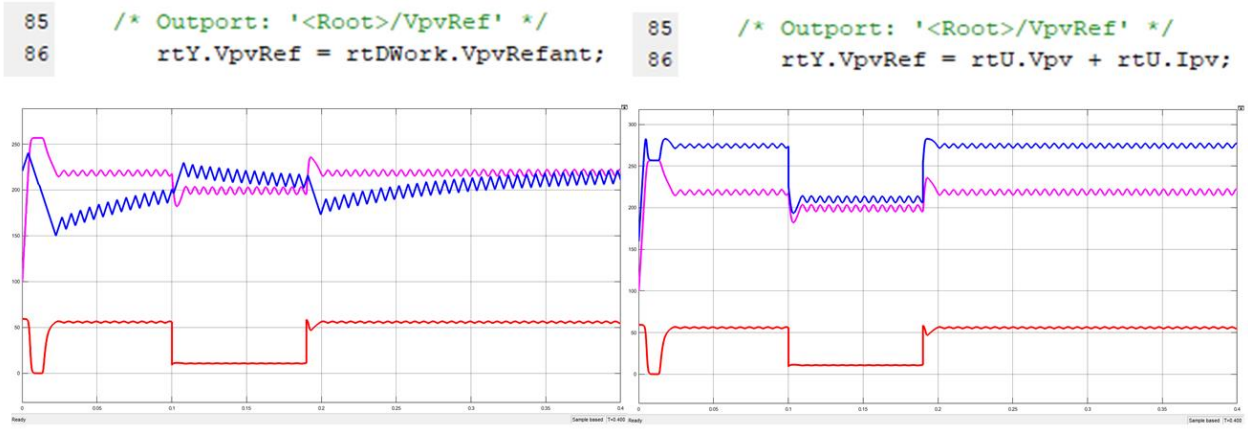


Figura 6-3. Modificación del código generado por la técnica PIL.

Aunque quizás sea un ejemplo algo simple, queda demostrado que se puede acceder al código generado y se puede modificar. Se puede apreciar perfectamente cómo es posible modificar el código y además responde coherente con lo codificado. Merece la pena mencionar que una vez modificado el código no es necesario volver a construir el bloque PIL, sino que se puede simular directamente el bloque PIL creado con anterioridad y que acabamos de modificar.

Por tanto, esta técnica PIL no solo nos permite generar código para su posterior verificación, sino que, además, nos permite modificarlo sin necesidad de emplear bloques que sean predefinidos por el software, al menos el código que pertenece a un bloque Matlab Function. Entonces, queda abierta una línea de investigación en la que cabe preguntarse, ¿qué operaciones se podrían implementar de esta manera?, ¿tiene límite?

Por ejemplo, se podría usar esta técnica de reprogramar el código para generar un controlador PID u otra versión del algoritmo MPPT que no esté basado en la técnica P&O. Lo que sí es claro es que esta técnica tiene la limitación de que necesitamos en primer lugar un controlador con bloques y variables originales de Simulink. Es decir, no podemos crear de la nada, sino reprogramar, por lo que dependemos de la previa creación de un bloque controlador, generar el bloque PIL y posteriormente reprogramarlo.

Debido a estas desventajas, la mejor manera de programar alguna funcionalidad desde cero sería empleando directamente el bloque personalizable **S-Function Builder** si queremos trabajar en el software **Simulink** o crear un nuevo archivo desde el software **CCS**.

6.2.2 Interactuar con los periféricos de entrada/salida de la placa

El objetivo final de esta idea sería combinar los algoritmos de Simulink con los bloques de la librería C2000. Esta librería debería ser accesible en el momento que realizamos la sección "Hardware Support Package (Matlab)" que se encuentra dentro del apartado de guía para la configuración y el uso del software.

Vamos a mostrar un ejemplo simple, y para ello vamos a crear un escenario desde cero. Necesitaremos por tanto los siguientes bloques: ganancia, operador relacional, constante, el bloque ADC y GPIO DO (Digital Output). Estos dos últimos bloques los tenemos accesible haciendo clic sobre el icono "Library Browser" situado en la barra de herramientas en la ventana de nuestro modelo de Simulink. Posteriormente, desplegamos la opción de "Embedded Coder Support Pakage for Texs Instruments C2000 Processors" y acto seguido, seleccionamos "F2837xD".

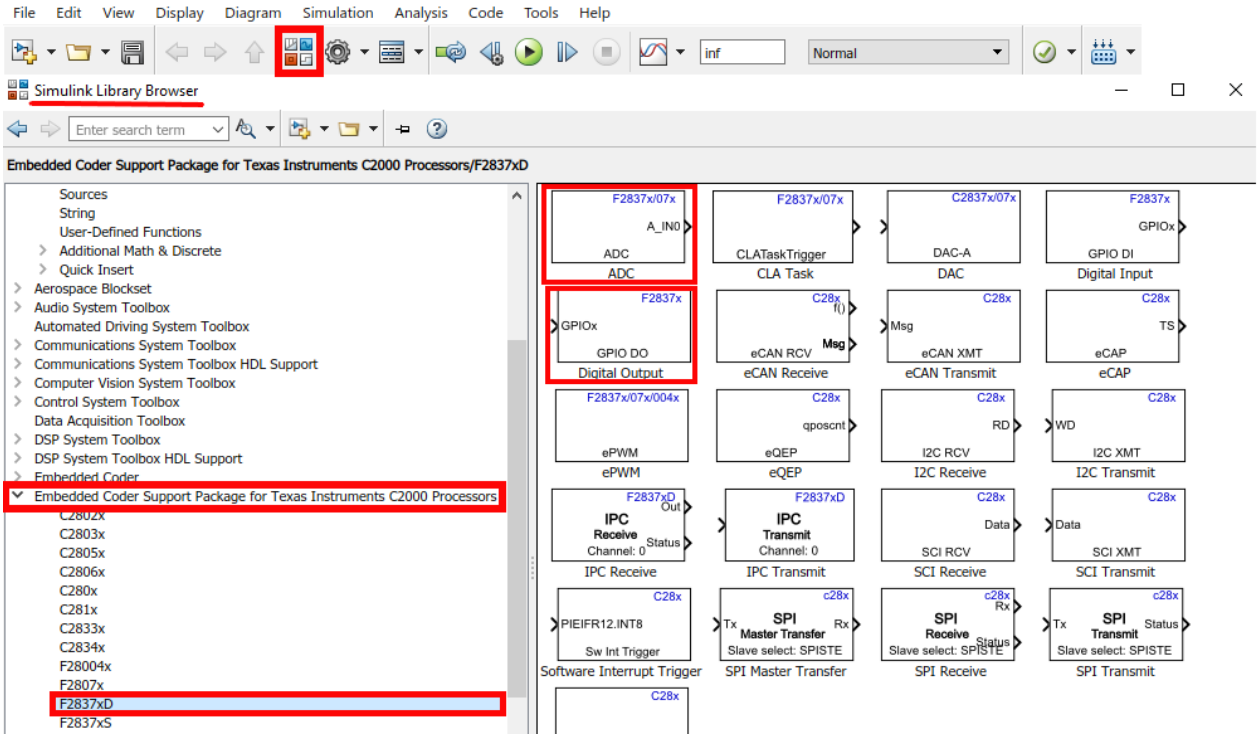


Figura 6-4. Librería de MATLAB-Simulink para DSP de la familia C200 de TI.

Para el ejemplo vamos a alimentar la placa de desarrollo con 5 voltios. Luego, si la señal que introducimos por el pin ADC (número 14) es mayor que 2.5 voltios, se debería encender el LED rojo, en otro caso se enciende el LED azul.

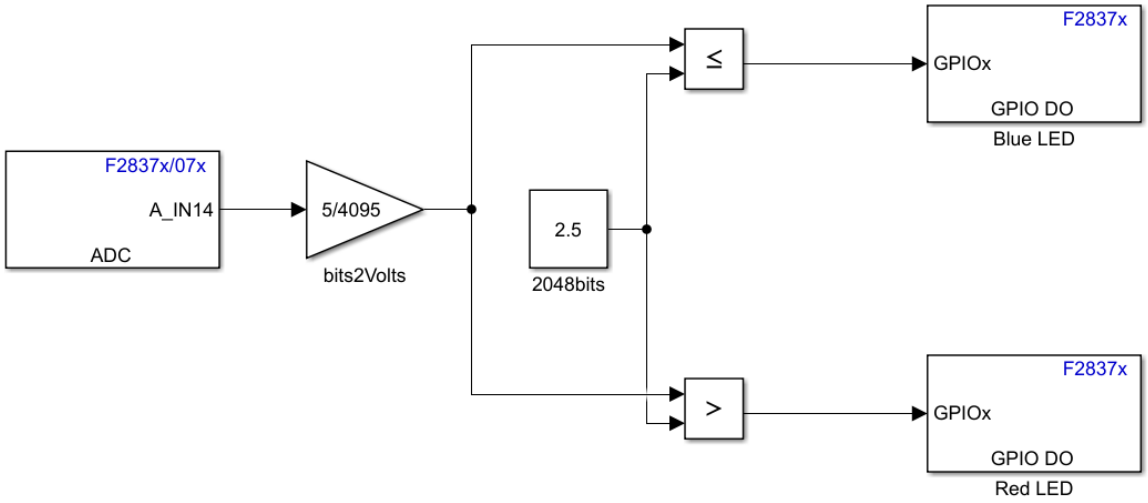


Figura 6-5. Librería de MATLAB-Simulink para DSP de la familia C200 de TI.

En la **Figura 6-6** se muestra la placa de desarrollo en el momento en el que es alimentada con más de 2.5 voltio (se enciende el LED rojo) y con menos de 2.5 voltios (se enciende el LED azul). Los pines empleados para esta prueba son, de izquierda a derecha: Alimentación, tierra y el periférico ADC (número 14).

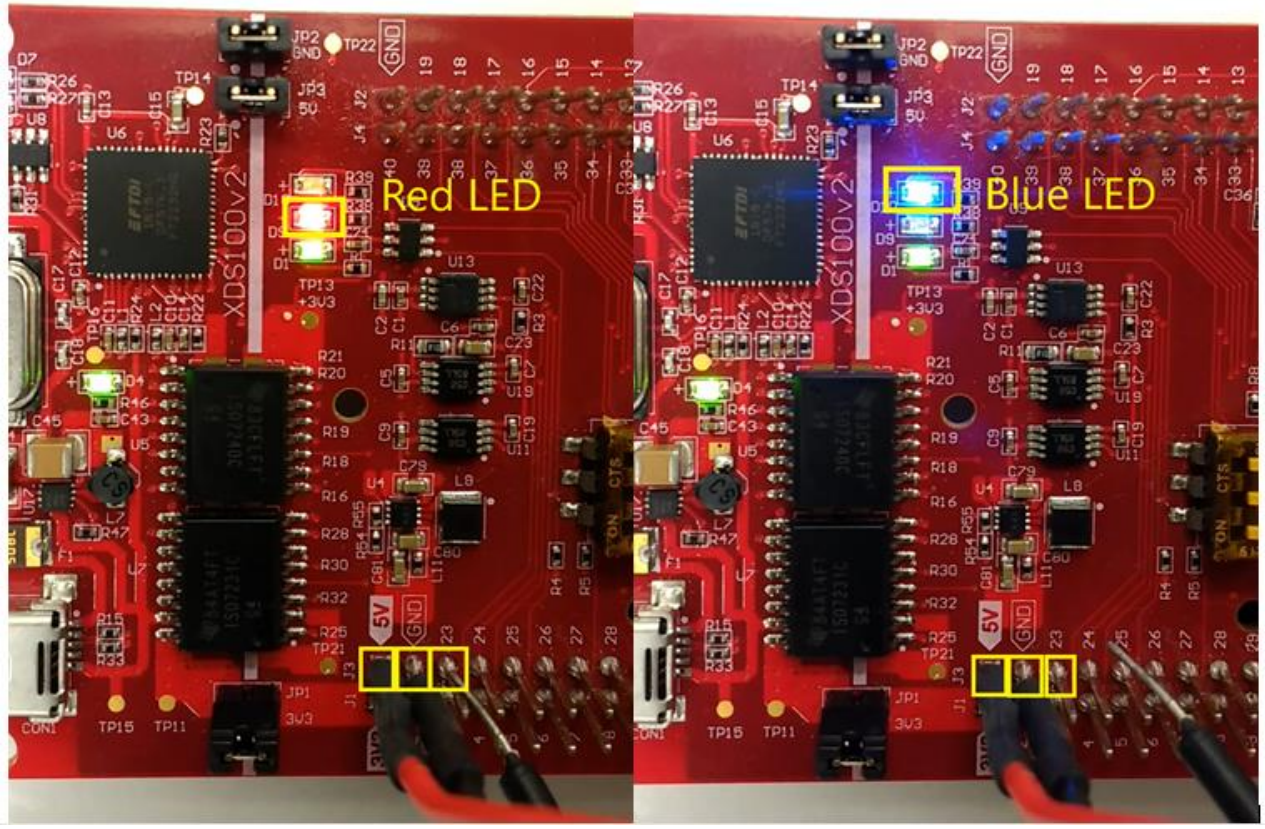


Figura 6-6. DSP F28379D: pines de alimentación, tierra y ADC nº 14 (de izquierda a derecha).

Sin embargo, parece que de esta forma no funciona como se espera. Esto puede ser debido a que la conversión de bits a voltios no se está realizando de la manera correcta. Sin embargo, dada la prohibición de acceso a los laboratorios debido a la pandemia del COVID-19, nos vemos obligados a cancelar el objetivo de interactuar con los periféricos de la placa de desarrollo. Puesto que no tenemos acceso al material necesario del laboratorio con el cual alimentar la placa. Aún así, es posible plantear las posibles soluciones a esta problemática de manera teórica.

Por un lado, indagando en el manual de referencias técnicas, nos percatamos de que es posible que haya que tener en cuenta una tensión de offset y una tensión mínima además de la conversión que ya hemos realizado en nuestro bloque de ganancia "bits2Volts" donde se tiene en cuenta el rango de valores representables con los 12 bits que disponemos. Por otro lado, podríamos emplear un bloque comparador que se encargue de comparar directamente el número de bits, sin necesidad de hacer conversión a voltios.

Se concluye que, a pesar de no tener la disponibilidad de los laboratorios para finalizar dicha prueba, hemos podido encontrar al menos un ejemplo (**video**) donde se usan placas de desarrollo especialmente preparadas para trabajar con convertidores de potencia y donde se accede a los periféricos ADC.

REFERENCIAS

[1] California Government, *List of Worldwide Scientific Organizations - Office of Planning and Research*. Rescatado de: https://www.opr.ca.gov/facts/list-of-scientific-organizations.html#skip-to-content. [Online – acceso junio 2020]

[2] Anpier (Asociación nacional de productores de energía fotovoltaica), *Anuario Fotovoltaico 2019*, Rescatado de: https://anpier.org/wp-content/uploads/2019/06/anuario245x173-DEF.junio-2019-WEB.pdf

[Online – acceso mayo 2020]

[3] G.T. Eléctronica, Electrónica de Potencia, *PRÁCTICA 7. Análisis mediante Simulación de un Sistema de Generación Fotovoltaica con Batería para Alimentación a una Carga Aislada*, Universidad de Sevilla, 2016-2017.

[4] Universidad de Jaén. *Teoría de semiconductores, el efecto fotovoltaico y la célula solar*. Rescatado de: http://www.ujaen.es/investiga/solar/07cursosolar/home_main_frame/03_celula/01_basico/3_celula_04.htm. [Online – acceso junio 2020]

[5] Departamento de física, Universidad de Burgos (2018). *EERR - T4: E. solar fotovoltaica - Efecto fotovoltaico*. Rescatado de: https://www.youtube.com/watch?v=9SVnpWChlQc. [Online – acceso junio 2020]

[6] Rashid, M., (2001). *Power Electronics Handbook*. 2nd ed., capítulo 23, p.540.

[7] G.T. Electrónica, Electrónica de Potencia, *Tema 5. Convertidores DC/AC. Inversores*, Universidad de Sevilla, 2015.

[8] Code Composer Studio (CCS). Rescatado de: https://www.ti.com/tool/CCSTUDIO [Online – acceso junio 2019]

[9] User's guide - F28379D, Rescatado de: https://www.ti.com/lit/ug/sprui77c/sprui77c.pdf. [Online – acceso mayo 2019]

[10] ControlSUITE – TI. Rescatado de: https://www.ti.com/tool/CONTROLSUITE. [Online – acceso mayo 2019]

[11] S. Vázquez, *PIL Tutorial*. Julio 2017.

[12] MathWorks. *Comparison of Custom Block Functionality when creating a custom block*. Rescatado de: https://www.mathworks.com/help/simulink/ug/comparison-of-custom-block-functionality.html. [Online – acceso octubre 2019]

[13] P. Gotika, *Implement Maximum Power Point Tracking Algorithms Using MATLAB and Simulink*, MathWorks (2015). Rescatado de: https://www.mathworks.com/videos/implement-maximum-power-point-tracking-algorithms-using-matlab-and-simulink-108209.html. [Online – acceso julio 2019]

ANEXOS

En esta sección se procede a mostrar los códigos que se han ido empleando a lo largo del desarrollo del proyecto.

A. Fichero parameter.m: Configuración de algunos parámetros del Sistema fotovoltaico

parameter.m

A continuación, se presenta el fichero ".m" que se nos ha sido asignado y cuya función es inicializar algunos de los parámetros usados en el escenario de Simulink donde se encuentra implementado el sistema de generación eléctrica aislado basado en energía fotovoltaica con batería.

```
%%This file clear the workspace and set the parameters to its initial
%%condition. It is automatically run when the simulation is started.

clear;
clc;
Tint = 1e-6;
fs = 10000;
Tsampling = 1./fs;
Tenable = 0.15;
TenablePV = 0.02;
TenableQ = 2.1;
Tload = 2.1;
Lred = 2e-3;
Lmodel = 1*Lred;
ps = Tint;
Lboost = 5e-3;
CinBoost = 2000e-6;
CoutBoost = 1000e-6;
IncVref = 0.5;
```

B. Fichero mppt.m: implementación del algoritmo MPPT

mppt.m

En este fichero está implementado el algoritmo de control MPPT con la técnica de P&O. Al algoritmo se le pasa como parámetro de entrada las variables de tensión e intensidad provenientes de las placas fotovoltaicas. A la salida, el algoritmo entregará la tensión de referencia, aquella que intentamos aproximar al MMP.

```
function mppt(block)
% This is a level-2 MATLAB file S-function which does the mppt P&O
% algorithm for a PV plant.
%   Copyright GTE Universidad de Sevilla.

  setup(block);

%endfunction

function setup(block)

  %% El incremento de VpvRef se introduce como un parámetro del bloque
  block.NumDialogPrms  = 1;

  %% Register number of input and output ports
  block.NumInputPorts  = 2;
  block.NumOutputPorts = 1;

  %% Setup functional port properties to dynamically
  %% inherited.
  block.SetPreCompInpPortInfoToDynamic;
  block.SetPreCompOutPortInfoToDynamic;

  block.InputPort(1).Dimensions        = 1;
  block.InputPort(1).DirectFeedthrough = false;

  block.InputPort(2).Dimensions        = 1;
  block.InputPort(2).DirectFeedthrough = false;

  block.OutputPort(1).Dimensions       = 1;

```

```
32      %% Set block sample time to inherited
33 -    block.SampleTimes = [100e-6 0];
34
35      %% Set the block simStateCompliance to default (i.e., same as a built-in
36      %  block)
37 -    block.SimStateCompliance = 'DefaultSimState';
38
39      %% Register methods
40 -    block.RegBlockMethod('PostPropagationSetup',    @DoPostPropSetup);
41 -    block.RegBlockMethod('InitializeConditions',    @InitConditions);
42 -    block.RegBlockMethod('Outputs',                 @Output);
43 -    block.RegBlockMethod('Update',                  @Update);
44
45
46
47    %endfunction
48
49    function DoPostPropSetup(block) ...
58
59    %endfunction
60
61    function InitConditions(block)
62
63 -    global vPVant;
64 -    global Pant;
65 -    global incVpvRef;
66 -    global VpvRefant;
67
68      %% Initialize Dwork
69 -    block.Dwork(1).Data = 0;
70 -    incVpvRef = block.DialogPrm(1).Data;
71 -    VpvRefant = block.InputPort(1).Data;
72 -    VpvRefant = 220;
73 -    vPVant = block.InputPort(1).Data;
74 -    iPVant = block.InputPort(2).Data;
75 -    Pant = vPVant*iPVant;
76
77    %endfunction
78
79    function Output(block)
80
81      %% Función MPPT
82
83 -    global vPVant;
84 -    global Pant;
85 -    global incVpvRef;
86 -    global VpvRefant;
87
88
89
90
91 -    vPV = block.InputPort(1).Data;
92 -    iPV = block.InputPort(2).Data;
93      %VpvRef = block.OutputPort(1).Data;
94
95 -    P  = vPV * iPV;
96 -    dV = vPV - vPVant;
97 -    dP = P - Pant;
98 -    vPVant = vPV;
99 -    Pant = P;
100
```

```
101
102
103 -     if dP ~= 0
104 -         if dP > 0
105 -             if dV > 0
106 -                 VpvRef = VpvRefant + incVpvRef;
107 -             else
108 -                 VpvRef = VpvRefant - incVpvRef;
109 -             end
110 -         else
111 -             if dV > 0
112 -                 VpvRef = VpvRefant - incVpvRef;
113 -             else
114 -                 VpvRef = VpvRefant + incVpvRef;
115 -             end
116 -         end
117 -     else
118 -         VpvRef = VpvRefant;
119 -     end
120
121 -     VpvRefant = VpvRef;
122 -     block.OutputPort(1).Data = VpvRef;
123
124     %endfunction
125
126     function Update(block)
127
128 -     block.Dwork(1).Data = block.InputPort(1).Data;
129
130     %endfunction
```

C. Fichero mppt_single.m: implementación del algoritmo MPPT con tipo de dato "single"

mppt_single.m

Este código es la adaptación a tipo de dato single del código mppt.m. esta adaptación es de carácter obligatorio pues sin este tipo de dato no se permitirá la implementación de la técnica PIL.

```
function mppt_single(block)
% This is a level-2 MATLAB file S-function which does the mppt P&O
% algorithm for a PV plant
%with single type of data.
%   Copyright GTE Universidad de Sevilla.

  setup(block);

%endfunction

function setup(block)

  %% El incremento de VpvRef se introduce como un parámetro del bloque
  block.NumDialogPrms  = 1;

  %% Register number of input and output ports
  block.NumInputPorts  = 2;
  block.NumOutputPorts = 1;

  %% Setup functional port properties to dynamically
  %% inherited.
  block.SetPreCompInpPortInfoToDynamic;
  block.SetPreCompOutPortInfoToDynamic;

 % Override input port 1 properties %%
  block.InputPort(1).Dimensions  = 1;
  block.InputPort(1).DatatypeID  = 1;   % ** Added Data Type Single **
  block.InputPort(1).DirectFeedthrough = false;

 % Override input port 2 properties %%
  block.InputPort(2).Dimensions  = 1;
```

```
32 -    block.InputPort(2).DatatypeID  = 1;   % ** Added Data Type Single **
33 -    block.InputPort(2).DirectFeedthrough = false;
34
35      % Override output port 1 properties %%
36 -    block.OutputPort(1).Dimensions  = 1;
37 -    block.OutputPort(1).DatatypeID  = 1;  % ** Added Data Type Single **
38
39      %% Set block sample time to inherited
40 -      block.SampleTimes = [100e-6 0];
41
42
43      %% Set the block simStateCompliance to default (i.e., same as a built-in
44      %  block)
45 -    block.SimStateCompliance = 'DefaultSimState';
46
47      %% Register methods
48 -    block.RegBlockMethod('PostPropagationSetup',    @DoPostPropSetup);
49 -    block.RegBlockMethod('InitializeConditions',    @InitConditions);
50 -    block.RegBlockMethod('Outputs',                 @Output);
51 -    block.RegBlockMethod('Update',                  @Update);
52
53
54
55    %endfunction
56
57    function DoPostPropSetup(block)
58
59      %% Setup Dwork
60 -    block.NumDworks = 1;
61 -    block.Dwork(1).Name = 'x0';
62 -    block.Dwork(1).Dimensions      = 1;
63 -    block.Dwork(1).DatatypeID      = 1; % ** Added Data Type Single **
64 -    block.Dwork(1).Complexity      = 'Real';
65 -    block.Dwork(1).UsedAsDiscState = true;
66
67    %endfunction
68
69    function InitConditions(block)
70
71 -    global vPVant;
72 -    global Pant;
73 -    global incVpvRef;
74 -    global VpvRefant;
75
76      %% Initialize Dwork
77 -    block.Dwork(1).Data = single(0);
78 -    incVpvRef = single(0.5);
79 -    VpvRefant = block.InputPort(1).Data;
80 -    VpvRefant = single(220);
81 -    vPVant = block.InputPort(1).Data;
82 -    iPVant = block.InputPort(2).Data;
83 -    Pant = vPVant*iPVant;
84
85    %endfunction
86
87    function Output(block)
88
89      %% Función MPPT
90
91 -    global vPVant;
92 -    global Pant;
93 -    global incVpvRef;
```

```
94 -    global VpvRefant;
95
96 -    vPV = block.InputPort(1).Data;
97 -    iPV = block.InputPort(2).Data;
98
99      % Aquí va el código para al algoritmo MPPT
100
101     %block.OutputPort(1).Data = VpvRef;
102
103 -    P  = vPV * iPV;
104 -    dV = vPV - vPVant;
105 -    dP = P - Pant;
106 -    vPVant = vPV;
107 -    Pant = P;
108
109
110 -    if dP ~= 0
111 -        if dP > 0
112 -            if dV > 0
113 -                VpvRef = VpvRefant + incVpvRef;
114 -            else
115 -                VpvRef = VpvRefant - incVpvRef;
116 -            end
117 -        else
118 -            if dV > 0
119 -                VpvRef = VpvRefant - incVpvRef;
120 -            else
121 -                VpvRef = VpvRefant + incVpvRef;
122 -            end
123 -        end
124 -    else
125 -        VpvRef = VpvRefant;
126 -    end
127
128 -    VpvRefant = VpvRef;
129 -    block.OutputPort(1).Data = single(VpvRef);
130
131 %endfunction
132
133 function Update(block)
134
135 -    block.Dwork(1).Data = block.InputPort(1).Data;
136
137 %endfunction
```

D. Implementación del algoritmo MPPT para el bloque MATLAB Function

mpptPandO

a continuación, se presenta el código obtenido de la web de MathWorks ya modificado para que se adapte a nuestro escenario. Para ello se ha modificado las condiciones iniciales de las variables y forzado a que el tipo de dato de dichas variables sean **single**. Dicho código va escrito en el bloque del tipo **MATLAB Function**. El código original está creado por Oscar Osorio.

```
function VpvRef = mpptPandO(Vpv,Ipv)
% Matlab implementation of Perturb and Observe algorithm for Maximun Power
% Point Tracking.
% ************************************************************************

% Define internal variables for the reference voltage, power and voltage
persistent VpvRefant Pprev Vprev %Revisar

% Initializate the values for the reference voltage, power and voltage
if isempty(VpvRefant)
    VpvRefant = single(260); % Tensión de circuito abierto (Voc)
    Vprev = single(0);
    Pprev = single(0);
end

% Initializate algorithm parameters andpower calculation
incVpv = single(0.5);
Ppv = Vpv*Ipv;

% Increments of power and voltage definition
dP = Ppv - Pprev;
dV = Vpv - Vprev;

% P&O approach
if dP ~= 0
      if dP > 0
          if dV > 0
              VpvRef = VpvRefant + incVpv;
          else
              VpvRef = VpvRefant - incVpv;
          end
      else
          if dV > 0
              VpvRef = VpvRefant - incVpv;
          else
              VpvRef = VpvRefant + incVpv;
          end
      end
  else
      VpvRef = VpvRefant;
end

%Update internal values
VpvRefant = VpvRef;
Vprev = Vpv;
Pprev = Ppv;
```