Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Simulación de un convertidor DC/DC para aplicación fotovoltaica mediante plataforma embebida

Autor: María Aguilar Vega

Tutor: Sergio Vázquez Pérez

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

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Simulación de un convertidor DC/DC para aplicación fotovoltaica mediante plataforma embebida

Autor:
María Aguilar Vega

Tutor:
Sergio Vázquez Pérez
Profesor Contratado Doctor

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

Sevilla, 2017

Trabajo Fin de Grado:	Simulación de un convertidor DC/DC para aplicación fotovoltaica mediante plataforma embebida

Autor: María Aguilar Vega
Tutor: Sergio Vázquez Pérez

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

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

El valor de una idea
radica en el
uso de la misma.

— Thomas A. Edison.

Agradecimientos

Este proyecto supone el final de un deseado objetivo perseguido durante años. Una etapa que, aunque parezca contradictorio, ha sido dura, divertida, amarga, dulce, nueva y sobre todo, enriquecedora. Con todo esto, me gustaría dedicar algunas palabras a todas aquellas personas que, de alguna manera, han hecho posible el alcance de esta ansiada meta:

A mi familia, por la paciencia y el apoyo recibido. Especialmente en épocas duras y fechas de exámenes, cuando el estrés se apoderaba de mí y me hacía ver todo mucho más oscuro; por hacerme entender desde hace muchos años que, con esfuerzo, todo se consigue.

A Pablo, por su inestimable apoyo a kilómetros de distancia. Por haberme inundado de inmejorables experiencias durante todos estos años.

A mis amigos y amigas, a los que mis logros los saborean como propios; a quienes han sabido mantener el contacto en la distancia y a quienes han hecho que no me quiera despedir de Sevilla.

Y por último a Sergio, por su infinita paciencia y valiosas aportaciones. Resulta una motivación constante poder aprender de profesionales con ganas de transmitir, de enseñar y sobre todo, con ganas de ayudarte a ser cada vez mejor.

Sinceramente, gracias.

María Aguilar Vega

Sevilla, junio de 2017

Resumen

El presente Trabajo Fin de Grado trata del uso de un sistema embebido para aplicaciones en el campo de la Electrónica de Potencia. Concretamente, consiste en la implementación de un esquema de control de un convertidor DC/DC para un sistema fotovoltaico en condiciones de simulación mediante lo que se conoce como PIL (Processor In the Loop).

Para ello, el circuito de la planta de potencia se emula mediante la plataforma de software de simulación PLECS, mientras que el esquema de control del convertidor se desarrolla en pseudo-tiempo real en el hardware *DSP-TMS320F28335*, perteneciente a la familia *C2000* del fabricante Texas Instruments.

El objetivo primordial de este trabajo es implementar el control del convertidor elevador para el seguimiento del punto de máxima potencia de una planta de paneles fovoltaicos con las herramientas descritas anteriormente.

Índice general

Agradecimientos III

Resumen V

Índice general VII

Índice de figuras IX

1. Introducción 1
- 1.1. Principio de funcionamiento de una célula fotovoltaica 2
- 1.2. Tipos de células fotovoltaicas 4
- 1.3. Curvas características de una planta de paneles 5
- 1.4. Uso de convertidores para la integración en la red 7
- 1.5. Punto de partida y objetivos 12

2. Modelo del sistema 13
- 2.1. Modelo del convertidor elevador 14
- 2.2. Modelo de la planta de paneles 15
- 2.3. Resultados caracteristicos del modelado 16

3. Algoritmos MPPT y diseño del control 19
- 3.1. Método Perturbance and Observe (P&O) 19
- 3.2. Método de tensión constante 20
- 3.3. Método de conductancia incremental 21
- 3.4. Formulación del lazo externo de control 25
- 3.5. Formulación del lazo interno de control 27

4. Processor In the Loop (PIL) 29
- 4.1. Visión general 29
- 4.2. Bloque PIL 31
- 4.3. Entorno de trabajo 33
  - 4.3.1. PIL Prep Tool 33
  - 4.3.2. Variables 33
    - 4.3.2.1. Read Probes 33
    - 4.3.2.2. Override Probes 34

4.3.3. Calibraciones 36
4.3.4. Identidad del código 36
4.3.5. Agente remoto 36
4.3.5.1. LLamadas de comunicación 36
4.3.5.2. Comunicación serie 37
4.3.5.3. Comunicación paralelo 37
4.4. Ejecución 37
4.4.1. Funcionalidad del Control Callback 38
4.4.2. Configuración del entorno 41

5. Implementación **43**
5.1. Solución mediante simulación de bloques 43
5.1.1. Bloque Boost Controller 46
5.1.2. Bloque PWM 47
5.1.3. Resultados de simulación 48
5.2. Implementación con controladores en C 51
5.2.1. Resultados de simulación 52
5.3. Implementación en PIL 54
5.3.1. Configuración en Code Composer Studio 55
5.3.2. Montaje en PLECS 55
5.3.2.1. Adaptación de medidas 56
5.3.2.2. C2000 ADC 58
5.3.2.3. C2000 ePWM 59
5.3.3. Configuración del DSP-TMS320F28335 60
5.3.4. Diagramas de flujo del código en C 61
5.3.5. Resultados de simulación 64

6. Conclusiones y trabajos futuros **71**

Anexos **73**

A. Scripts **75**

B. Ficheros de cabecera (Header Files) **81**

C. Ficheros de C **95**

Bibliografía **131**

Índice de figuras

1.1. Funcionamiento célula fotovoltaica. 3
1.2. Modelo de una célula fotovoltaica. 4
1.3. Curva I-V de una célula fotovoltica. 4
1.4. Célula fotovoltaica. 5
1.5. Curvas de funcionamiento en condiciones nominales. 6
1.6. Efecto de la irradiancia y la temperatura en las curvas de funcionamiento. 7
1.7. Configuración de inversor centralizado. 8
1.8. Configuración de inversores en hileras. 9
1.9. Configuración multihileras. 9
1.10. Configuración modular. 10
1.11. Estructura de control MPPT para etapa simple. 11
1.12. Estructura de control MPPT para doble etapa. 11

2.1. Sistema de partida. 13
2.2. Esquema simplificado de un convertidor elevador. 14
2.3. Planta de paneles. 15
2.4. Máscara del bloque panel. 16
2.5. Curva P-V de la planta de paneles. 17
2.6. Curva I-V de la planta de paneles. 17

3.1. Método Perturbance and Observe (P&O). 20
3.2. Método de la tensión a circuito abierto fraccional. 21
3.3. Concepto del algoritmo de conductancia incremental. 22
3.4. Método conductancia incremental convencional. 23
3.5. Método conductancia incremental propuesto. 24
3.6. Esquema del convertidor y la planta de paneles. 25
3.7. Evolución deseada del sistema. 26
3.8. Etapa de evacuación de potencia del convertidor Boost. 27

4.1. Esquema de funcionamiento del Process In the Loop. 30
4.2. Bloque PIL. 31
4.3. Configuración general del PIL. 32
4.4. Entradas y salidas de PIL. 32
4.5. Ejecución en tiempo real. 38
4.6. Ejecución en pseudo-tiempo real. 39

5.1. Implementación en bloques. 44
5.2. Implementación en bloques. 45
5.3. Bloque Boost Controller. 46
5.4. External Control Loop. 46
5.5. Inner Control Loop. 47
5.6. Generación de PWM con tiempos muertos. 47
5.7. Disparos de los transistores con tiempos muertos. 48
5.8. Resultado tras la simulación de bloques. 49
5.9. Seguimiento de tensión con bloques. 50
5.10. Seguimiento en corriente con bloques. 50
5.11. Aproximación Euler I. 51
5.12. Seguimiento de tensión con controladores en C. 52
5.13. Seguimiento en corriente con controladores en C. 52
5.14. Resultados de simulación con C-Scripts. 53
5.15. Implementación en bloques. 54
5.16. Adaptación de medidas. 56
5.17. Implementación con PIL. 57
5.18. Bloque ADC. 58
5.19. Bloque C2000 ePWM. 59
5.20. ePWM tipo 1. 59
5.21. Comportamiento del driver. 59
5.22. Configuración ePWM. 61
5.23. Diagrama de flujo de la función principal. 62
5.24. Diagrama de flujo de la interrupción del ADC. 63
5.25. Seguimiento de tensión en PIL. 64
5.26. Seguimiento de corriente en PIL. 65
5.27. Resultado tras la simulación con PIL. 66
5.28. Resultado con PIL ante un cambio en irradiancia. 67
5.29. Máquina de estados. 68
5.30. Diagrama de flujos de la interrupción con máquina de estados. 69

Capítulo 1

Introducción

La energía solar fotovoltaica es una fuente limpia y renovable, con una larga vida útil y una alta fiabilidad. Se trata de una energía libre y abundante en la mayor parte del mundo, que ha demostrado ser una fuente económica en muchas aplicaciones de la actualidad: alimentación de dispositivos autónomos, producción de electricidad, etc.

Como señalan Djamila Rekioua y Ernest Matagne en [1], la tecnología fotovoltaica (PV) es una tecnología en la cual, la energía radiante del sol se convierte directamente en corriente eléctrica continua mediante un dispositivo semiconductor denomiado célula fotovoltaica[1]. El material más comúnmente empleado en las células de los paneles fotovoltaicos es el silicio. Posteriormente se expondrá el principio de funcionamiento de una célula y se realizará una clasificación de las mismas.

Existen múltiples ventajas sobre el uso de sistemas fotovoltaicos. Entre ellas, caben destacar:

- No existen partículas móviles ni combustibles consumidos o emitidos.
- Los procesos fotovoltaicos son completamente sólidos y autónomos.
- No necesitan conexión a una fuente de alimentación o suministro de combustible existente.
- Pueden soportar condiciones climáticas estrictas, como el tiempo nublado.
- Se pueden combinar con otras fuente de alimentación para aumentar la fiabilidad del sistema.
- Su diseño es modular, por lo que permite una actualización de la planta a medida que aumenta la demanda de energía.
- Su coste de producción es prácticamente nulo.

En cualquier caso, como cualquier tipo de tecnología también presenta inconvenientes, tales como la dependencia de la irradiancia en cada instante de tiempo y por lo tanto, de las limitaciones en la producción de energía en función de la radiación solar existente en cada instante. Se trata de una energía no despachable, por lo que para

[1] El término «fotovoltaico» se comenzó a usar en Reino Unido en el año 1849. Proviene del griego $\varphi\omega\varsigma$: phos, que significa «luz», y de -voltaico, que proviene del ámbito de la electricidad, en honor al físico italiano Alejandro Volta.

hacer que sea despachable es necesario el uso de almacenamiento, lo cual puede llegar a ser bastante costoso.

Aunque el balance económico en esta tecnología parece ser positivo, existe cierto enfrentamiento entre la inversión en fuentes de energía renovables y el mantenimiento de las convencionales donde la inversión en centrales fósiles ha sido considerable. En el caso de España, los cambios de gobierno y pautas políticas han influido de tal manera que las inversiones en el sector fósil no pueden dejar de ser consideradas: la introducción de renovables en la red eléctrica supondría un coste de producción cero frente a las centrales convencionales, lo cual impide una recuperación de las inversiones en las segundas. Es decir, la adopción de las energías renovables es lenta: una vez construida una central nuclear (o de cualquier otro tipo) no es viable apagarla hasta el fin de su vida útil.

A pesar de este tipo de problemáticas, la energía fotovoltaica ya es pionera en muchos países tales como Alemania, Escocia o Costa Rica.

1.1. Principio de funcionamiento de una célula fotovoltaica

Como se ha comentado en el apartado 1, una célula fotovoltaica es un dispositivo electrónico que transforma la energía solar en corriente continua.

Para entender el funcionamiento de una célula fotovoltaica es necesario recordar los conceptos de electrón, hueco y par electrón-hueco (eh). Supóngase entonces un material semiconductor caracterizado por una energía E_g entre la banda de conducción y la de valencia. Entonces, dada una temperatura algunos de los electrones consiguen la energía suficiente como para desprenderse de los átomos: estos se conocen como electrones libres. Igualmente a los enlaces vacíos se les denomina como huecos. Ambos procesos están asociados y por tanto se habla de par electrón-hueco.

En el caso de una célula fotovoltaica los procesos de generación[2] y recombinación[3] se producen cuando el semiconductor dopado se expone a radiación electromagnética. Lo que ocurre es que un fotón consigue desprenderse, de manera que es capaz de arrancar un electrón creando un hueco en el propio átomo. Es decir, la energía que necesita el electrón proviene del fotón, aunque la interacción de la luz con la materia puede no llegar a ser efectiva pudiendo pasar a través del material (fotones de baja energía) o puede ser que la luz se vea reflejada.

Para la generación de corriente eléctrica es necesario la existencia de un campo eléctrico que logre separar los pares eh, de forma que es necesario que el semiconductor esté formado por dos capas: una capa tipo p y otra tipo n. Así los huecos, se dirigen hacia el contacto del lado p, provocando la extracción de un electrón desde el metal

[2]Proceso en el que un electrón consigue la suficiente energía para pasar de la banda de valencia a la de conducción.

[3]Proceso análogo a la generación: el electrón cede energía pasando de la banda de conducción a la de valencia.

que constituye el contacto. Por su parte, los electrones se dirigen hacia el contacto del lado n inyectándolos en el metal. Este fenómeno induce una corriente eléctrica por un circuito externo, haciendo que la célula funcione como generador. En la figura 1.1 se ilustra el funcionamiento de una célula fotovoltaica cuando se conecta a un circuito externo a través de los contactos metálicos.

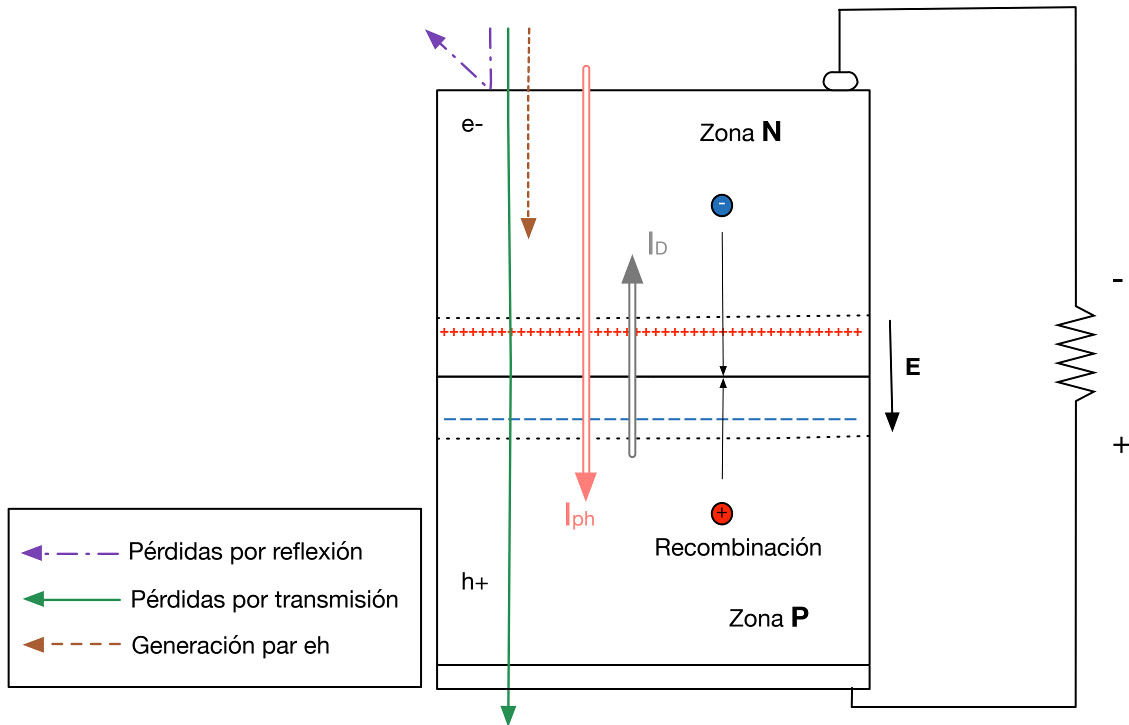


Figura 1.1: Funcionamiento célula fotovoltaica.

De esta forma, la corriente entregada por una célula se calcula como la suma neta de dos corrientes que se oponen:

• Corriente de iluminación: producida como consecuencia de la generación de portadores que produce la propia iluminación (fotones). Es función dependiente de la radiación luminosa y la temperatura.

$$I_{ph} = I_L \tag{1.1}$$

• Corriente de oscuridad: surge como consecuencia del proceso de recombinación de portadores debido al voltaje externo para poder entregar energía a la carga. Se puede expresar matemáticamente según la expresión (1.2).

$$I_D = I_O \left[e^{\frac{eV_D}{mkT}} - 1 \right] \tag{1.2}$$

Con las definiciones previas, se puede comprender fácilmente el modelo de una célula fotovoltaica representado en la figura 1.2, donde la polarización p-n viene representada

por el propio diodo y las resistencias R_s y R_p modelan, respectivamente, las pérdidas internas del movimiento de los electrones y de los contactos metálicos.

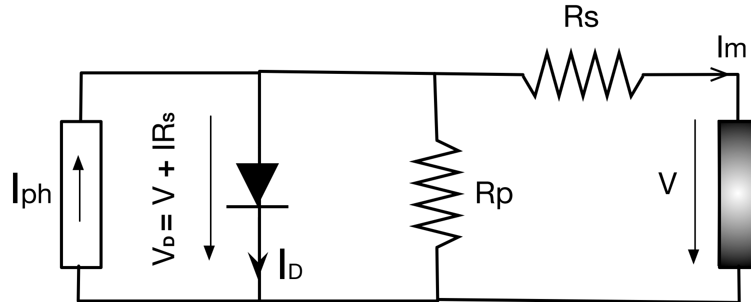


Figura 1.2: Modelo de una célula fotovoltaica.

El valor de la corriente I_m generado por la célula viene dada por la ecuación (1.3), donde m es un factor constante (normalmente comprendido entre 1 y 2) conocido como factor de idealidad del diodo, k la constante de Boltzman y T la temperatura absoluta en Kelvin. También se representa en la figura 1.3 la curva característica de una célula fotovoltaica deducida a partir de la ecuación (1.3).

$$I_m = I_{ph} - I_O \left[e^{\frac{e(V+IR_s)}{mkT}} - 1 \right] - \frac{(V + IR_s)}{R_p} \tag{1.3}$$

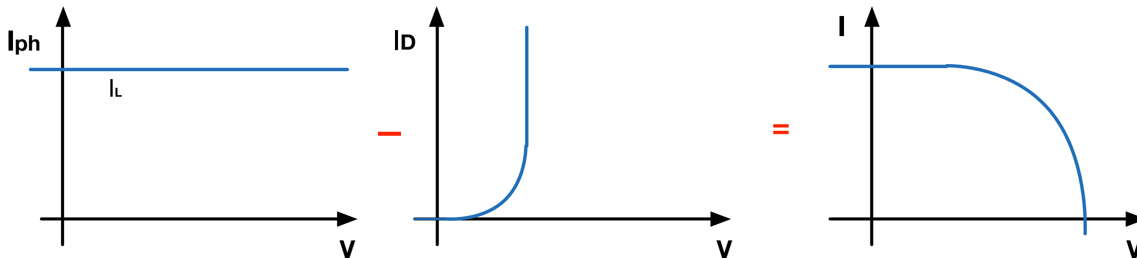


Figura 1.3: Curva I-V de una célula fotovoltica.

1.2. Tipos de células fotovoltaicas

Actualmente, la tecnología de fabricación de células fotovoltaicas más empleada es el silicio monocristalino. En una de las capas de silicio se emplea un dopado P (boro, por ejemplo), mientras que para la otra se usa un elemento donador (fósforo) para así crear la unión p-n.

La clasificación más usual que se puede encontrar en bibliografía como [1] y [2] se realiza según el tipo de silicio con el que se fabrican. Así se pueden distinguir células de:

• Silicio monocristalino: se obtiene del propio silicio fundido y dopado con boro. Presenta típicos colores azules y el conexionado es visible. En cuanto al rendimiento en campo, sus valores típicos compenden entre el 15 y el 18 %.

• Silicio policristalino: el proceso de solidificación se hace en múltiples cristales. La fabricación de este tipo de células es muy semejante al anterior con una simplificación en el número de pasos. Pueden aprecierase los cristales y se caracteriza por su tonalidad azul. El rendimiento oscila entre el 12 y el 14 %.

• Silicio amorfo: se fabrica depositando una lamina delgada sobre sustrato de vidrio o plástico. A diferencia de las otras dos, presenta un color marrón homogéneo (aunque se pueden fabricar de diferentes colores) y no se aprecian las conexiones. A pesar del atractivo que presenta por el poco espesor a utilizar, su rendimiento en campo no llega al 10 %.

La disposición final de una célula fotovoltaica queda como aparece representado en la figura 1.4. En dicha representación, se muestra de forma descendente en la leyenda las capas que componen una célula desde la capa más interna (contacto metálico) hasta la más externa (cristal de protección).

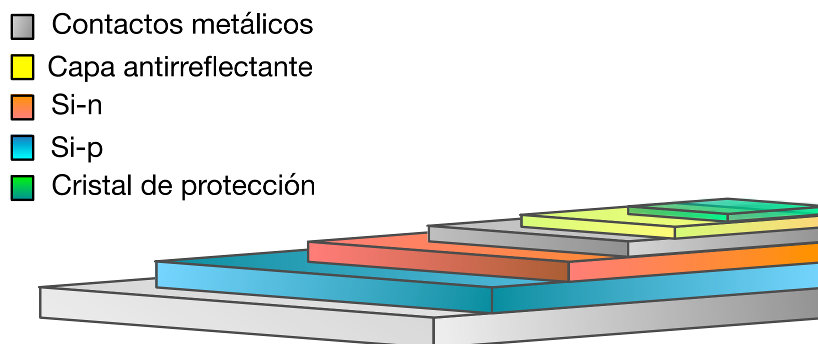


Figura 1.4: Célula fotovoltaica.

Para conseguir una célula fotovoltaica práctica es necesario añadir contactos eléctricos para poder extraer la energía que se genera. Además, se usa una capa antireflectante para garantizar un nivel alto de absorción de fotones y que aumenta el rendimiento de la propia célula. También es importante proteger la célula con un cristal de protección que sea capaz de dejar pasar la luz.

1.3. Curvas características de una planta de paneles

Cada panel está formado por un número determinado de células conectadas en serie. Igualmente, una planta o huerto fotovoltaico está compuesto por diferentes paneles

dispuestos en serie o en paralelo. De esta forma se puede hacer referencia a las curvas de funcionamiento de la planta o de un módulo individual. En la figura 1.5 se representan dichas gráficas en condiciones ideales: irradiancia de 1 kW/m^2 y 25ºC de temperatura del panel.

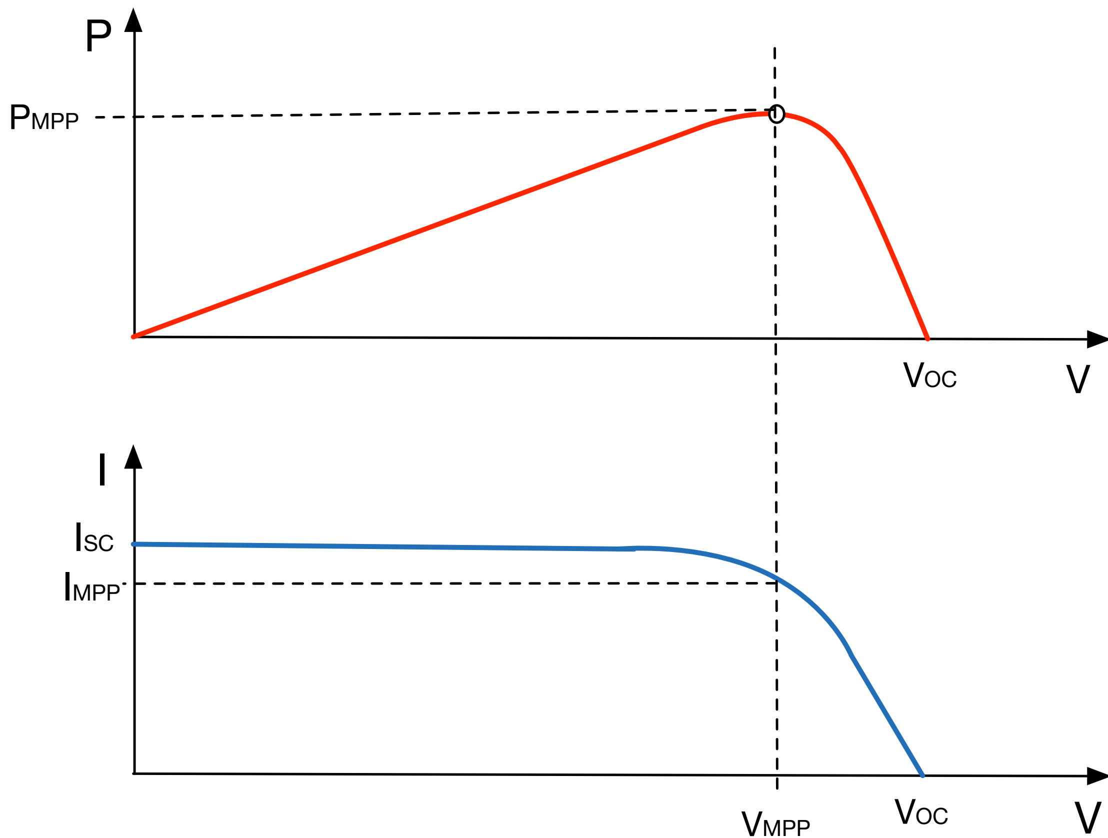


Figura 1.5: Curvas de funcionamiento en condiciones nominales.

Como puede observarse en la figura 1.5, existe un punto de funcionamiento para el cual la potencia generada, por el panel o el conjunto de paneles, es máxima. Dicho punto es conocido como punto de máxima potencia (MPP). También se definen otros puntos de interés que permiten definir completamente el comportamiento del panel:

- V_{MPP}: tensión para la que se alcanza el punto de máxima potencia.
- I_{MPP}: corriente para la que se alcanza el punto de máxima potencia.
- V_{OC}: del inglés, open-circuit. Es la tensión de circuito abierto, definida como la diferencia de potencial que aparece en los extremos del circuito (bornas de los paneles) cuando no existe ninguna carga externa conectada.
- I_{SC}: del inglés, short-circuit. Se trata de la corriente de cortocircuito, definida como la corriente que circula por las bornas cuando se cortocircuita la salida.

Como es de esperar, la posición y trayectoria de las curvas se ven modificadas por la influencia de la irradiación y temperatura, tal y como se muestra en la figura 1.6. La potencia generada será menor cuanto menor sea la irradiancia y mayor sea la temperatura del panel.

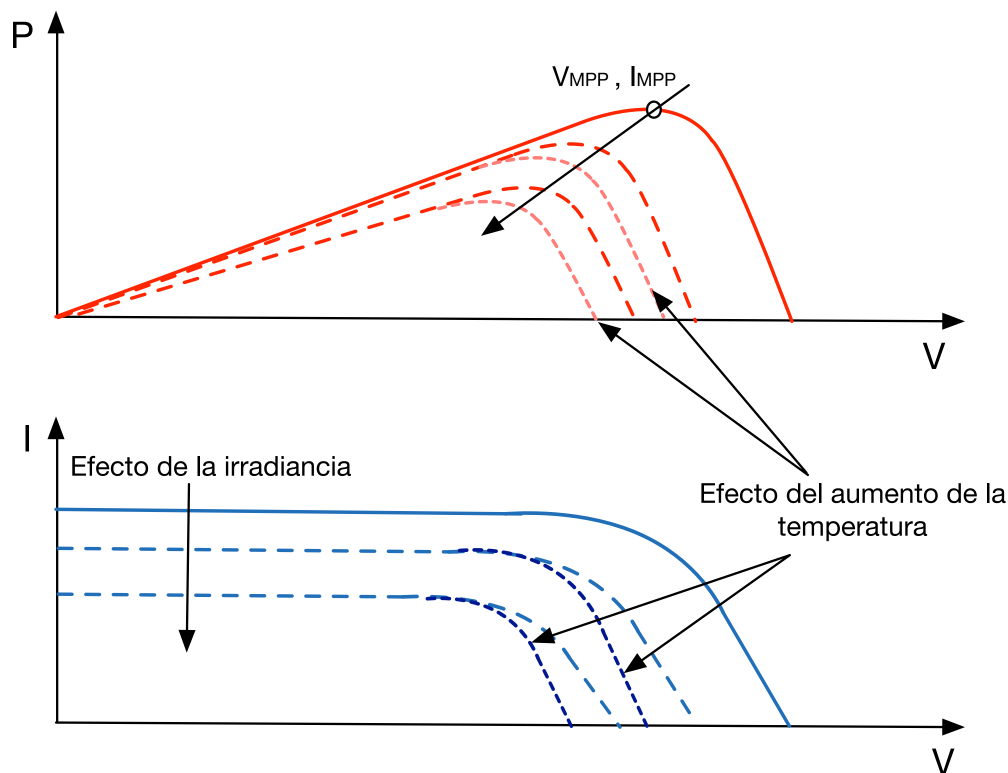


Figura 1.6: Efecto de la irradiancia y la temperatura en las curvas de funcionamiento.

1.4. Uso de convertidores para la integración en la red

Una planta solar fotovoltaica cuenta con distintos elementos que permiten el funcionamiento de la misma. Uno de los más importantes es el convertidor de potencia para la transformación de la corriente continua en corriente alterna. Dependiendo del tipo de integración del sistema en la red de distribución podrá ser necesario el uso de convertidores DC/DC, lo cual permite incrementar el nivel de tensión en la entrada del inversor. A continuación se desarrollarán los tipos de conexiones de paneles PV.

Inversor centralizado

En este tipo de configuración sólo se hace uso de un inversor central al que llega la corriente de todas las hileras, tal y como se puede observar en el esquema de la figura 1.7. Cada hilera necesita un diodo de paso[4] para evitar que la hilera que produce menos energía consuma energía de la que produce más. Entre sus ventajas destacan su alta eficiencia (con conexión de paneles en serie), su bajo coste y una alta fiabilidad. Por el contrario, presenta el inconveniente de que toda la potencia es evacuada a través de un único convertidor, por lo que en caso de fallo la producción se detiene. Además la búsqueda del punto de máxima potencia proporcionará un punto de funcionamiento global a todas las hileras (que en condiciones generales, trabajarán con distintas condiciones ambientes), por lo que no se podrá extraer el cien por cien de la potencia.

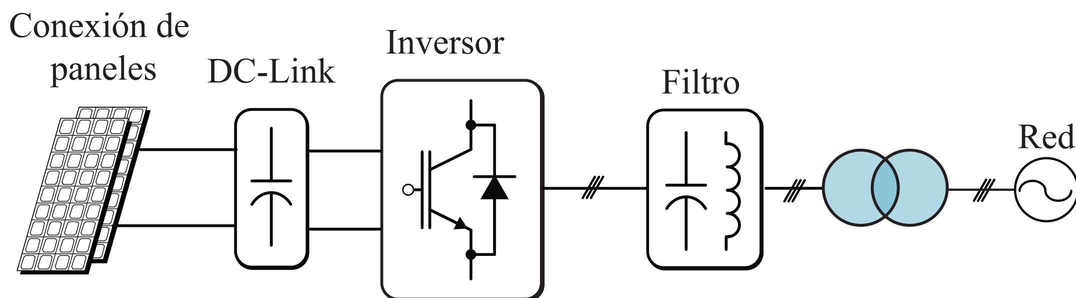


Figura 1.7: Configuración de inversor centralizado.

Inversores en hileras

Cada hilera tiene su propio inversor de forma que es posible encontrar el punto de máxima potencia para cada hilera, por lo que el hecho de que las hileras presenten diferentes condiciones de trabajo deja de ser un problema mayor. Resulta más costoso que la configuración anterior debido a que el número de inversores necesarios es mayor.

En la figura 1.8 aparece representada dicha configuración. Normalmente cada hilera representa una fase, instalando una hilera entre una fase y el neutro de la red, de manera que se pueden simultanear la fases, aunque a partir de $5kW$ es necesaria la inversión en trifásica.

[4]En la realidad, en el diseño de huertos solares también se utilizan fusibles. Esto permite una disminución de las pérdidas debido a la caída de tensión en los diodos.

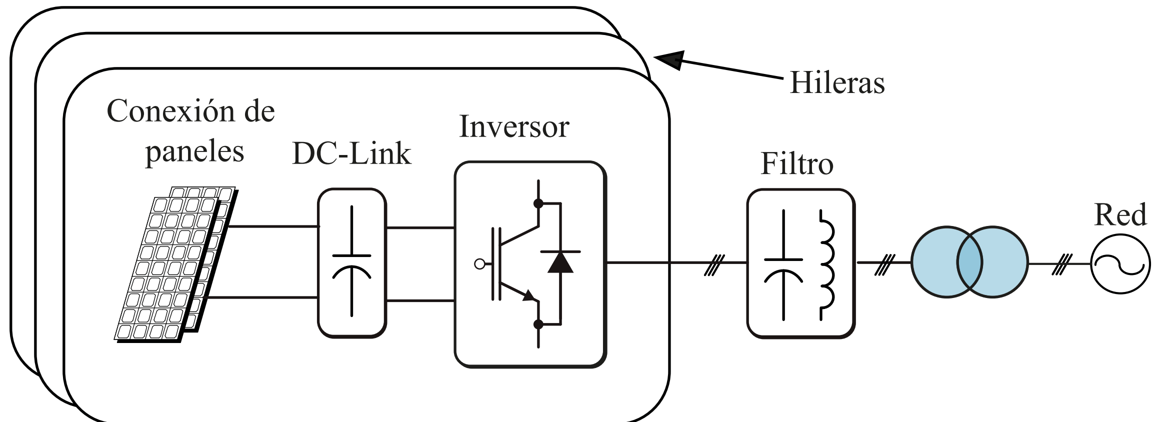


Figura 1.8: Configuración de inversores en hileras.

Configuración multihilera

Se trata de una evolución de la configuración en hileras, como se puede observar en el esquema de la figura 1.9. Se emplea un convertidor Boost para cada hilera y la salida de cada convertidor se conecta a un bus común de continua que a su vez se conecta al inversor.

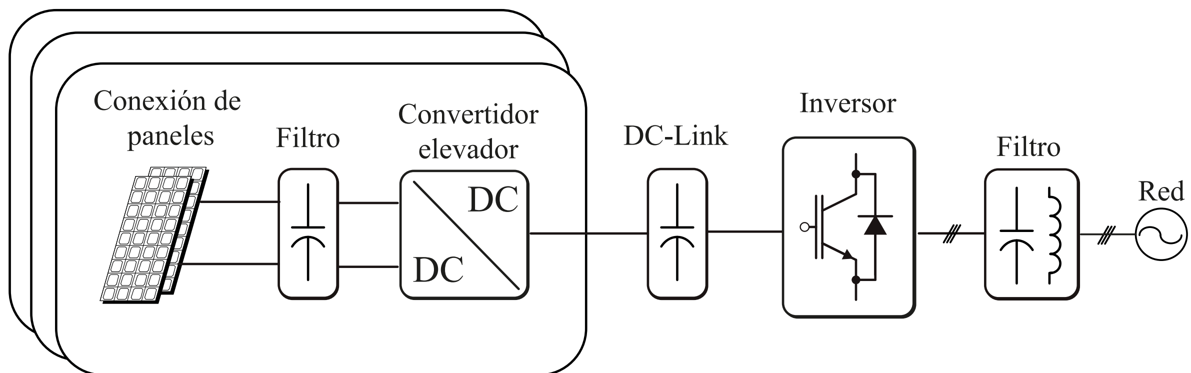


Figura 1.9: Configuración multihileras.

Con esta distribución es posible encontrar en punto de máxima potencia por cada hilera y usar un inversor central con mejor rendimiento que los de configuración en hilera. El mayor inconveniente se encuentra en el uso de los dos convertidores (DC/DC-DC/AC), ya que al tener una disposición en cascada las pérdidas se suman. El bus de alterna puede ser monofásico o trifásico.

Configuración modular

Este tipo de configuración también se conoce como "paneles CA". Se caracterizan por no tener cableado en corriente continua, ya que se integra un inversor con cada panel.

Igualmente, al ser modular, es posible garantizar una mayor estabilidad y flexibilidad al sistema: es posible trabajar en el punto de máxima potencia de cada panel y existe cierto respaldo en la inversión de energía y generación en caso de fallos en algún módulo. Su configuración se muestra esquemáticamente en la figura 1.10.

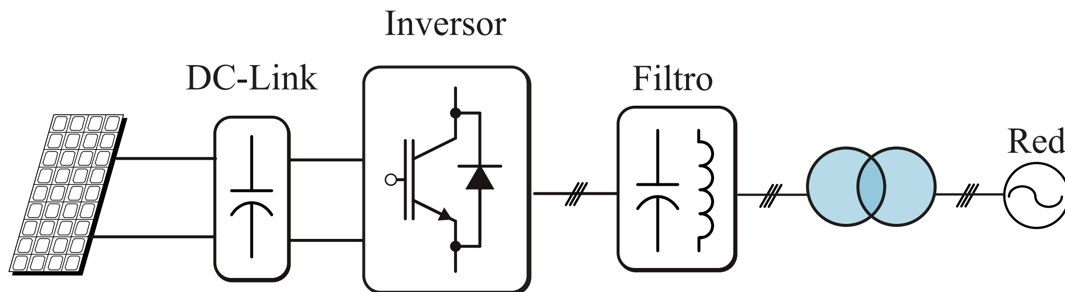


Figura 1.10: Configuración modular.

El principal inconveniente es el coste de la inverisón y el propio mantenimiento, sobre todo si se habla de plantas con una extensión considerable.

Atendiendo a la configuración que se implemente, la estructura de control del MPPT se podrá llevar a cabo de diferentes formas. En la figura 1.11 se representan dos posibles estructuras para el control del MPPT con un inversor de etapa simple. La opción A) el MPPT controla directamente la corriente del bus de alterna mientras que en B) el MPPT controla la tensión del bus de continua y la corriente del lado de alterna es controlada a través de un controlador de tensión DC. También es posible modificar el control de B) realizando un control feedforward de la potencia, lo que permite enriquecer la dinámica del control MPPT.

Como se ha comentado al principio del subcapítulo 1.4, también es posible realizar una configuración en doble etapa. En este caso, el MPPT es controlado por el convertidor elevador (Boost). El inversor se encarga de controlar la corriente de salida en alterna regulando la tensión del bus de continua a un valor de referencia (ver figura 1.12). Igualmente, también es posible realizar un control feedforward de la potencia que enriquecería la dinámica del control.

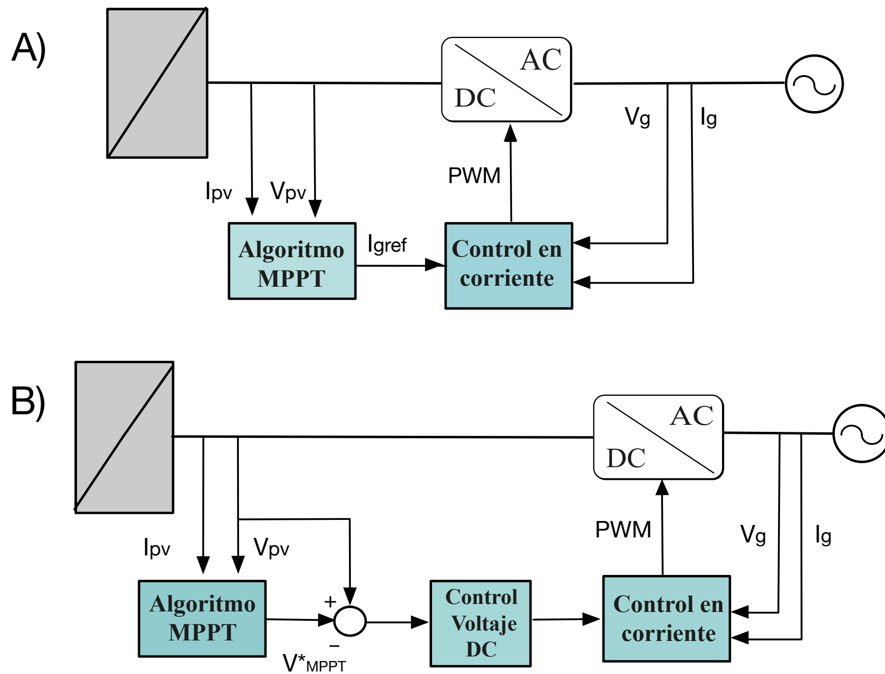


Figura 1.11: Estructura de control MPPT para etapa simple.

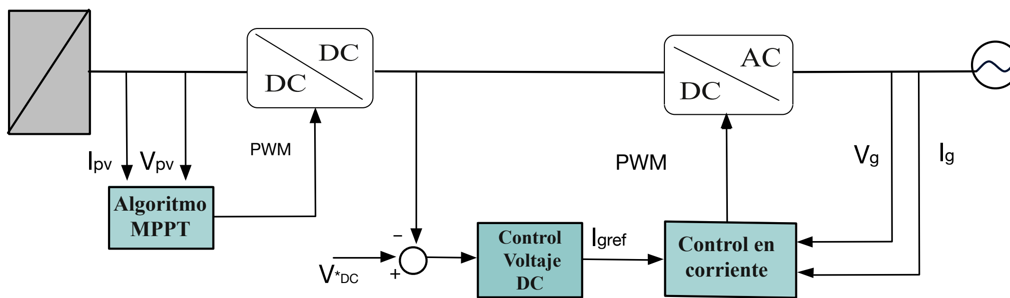


Figura 1.12: Estructura de control MPPT para doble etapa.

1.5. Punto de partida y objetivos

Este trabajo se centrará en desarrollar el algoritmo de control el convertidor elevador de una estructura en doble etapa para la obtención del punto de máxima potencia.

Para el desarrollo del proyecto se usará PLECS como herramienta de simulación de la planta y los componentes de electrónica de potencia. Se trata de un software de simulación desarrollado por Plexim que cuenta con numerosas herramientas que pueden ser utilizadas en muchas materias de la electrónica de potencia. PLECS cuenta con un módulo de Process-In-the-Loop que permite ejectuar el código de control en el DSP-TMS320F28335 en pseudo-tiempo real.

Con el fin de alcanzar el punto de máxima de potencia de una planta de paneles se van a desarrollar dos aspectos: por un lado se implementará un algoritmo del seguimiento del MPP, que se denotará como MPPT, y por otro los lazos internos y externos de control para el control de la tensión del condensador de salida de los paneles y la corriente de salida del convertidor. Todos estos aspectos (modelo de la planta, algoritmo MPPT, etc) se describirán con más detalles en secciones posteriores.

Para mejorar el sistema y asegurar protecciones en una posible ejecución e implementación física del proyecto, también se implementará una máquina de estados que habilite o deshabilite el sistema en caso de detectar algún error.

Capítulo 2

Modelo del sistema

En este capítulo se presenta el modelo de la planta que se va a simular. Se realizará una explicación generalizada del sistema y posteriormente se profundizará en el modelado de la planta de paneles fotovoltaicos y del convertidor elevador. También se mostrarán las simulaciones necesarias para comprobar el correcto funcionamiento de los diferentes modelos. El objetivo no es entrar en detalle en el modelo de cada componente ni mucho menos explicar cómo usar la herramienta de simulación, si no presentar el modelo que permitirá partir de la situación inicial. En cualquier caso, puede consultarse en [3] cualquier asunto relacionado con el funcionamiento de plecs y los componentes de la librería.

Para simplificar el modelo y centrarse sólo en el control del convertidor elevador, se sustituye el condensador de la etapa de salida del convertidor Boost por una fuente de tensión constante: el control del inversor aseguraría la regulación de tensión en este condensador. Dicha fuente de tensión constante se representa en la figura 2.1 como $V_{_dc}$.

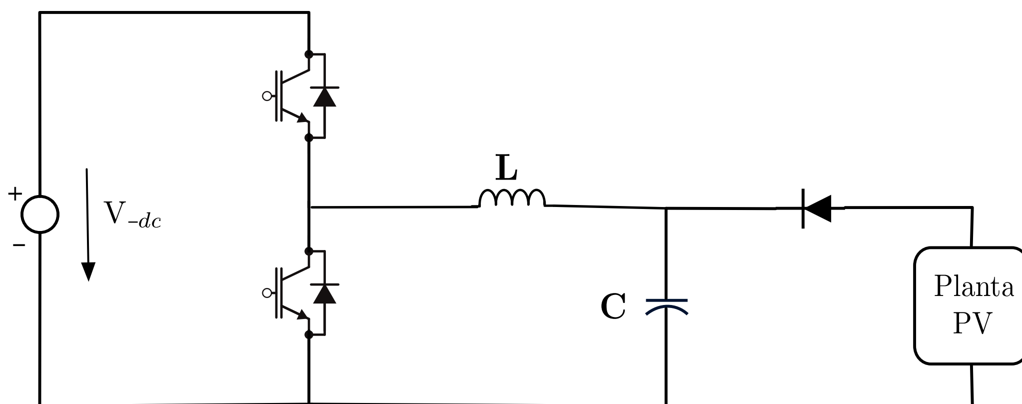


Figura 2.1: Sistema de partida.

La simulación de la planta se llevará a cabo de manera continua mediante el solver incorporado del software RADAU (stiff), mientras que el algoritmo de búsqueda del

MPPT y los lazos de control se simularán en discreto antes de incorporar el funcionamiento con el microcontrolador. El solver RADAU está basado en el método Runge-Kutta. En cuanto a la rigidez del solver, según Plexim, no existe una regla general estándar para lo que es un sistema rígido y no rígido, pero usar el tipo incorrecto para un modelo puede producir resultados lentos y/o inexactos. Una sugerencia, si no se está seguro de si se trata de un modelo rígido, es tratar de aplicar ambos (rígido y no rígido) con la misma configuración del sistema y hacer una determinación de la precisión general y la velocidad de los resultados. En este caso, un indicador obvio de rigidez es el uso de los componentes de conmutador dinámico como el modelo de diodo.

2.1. Modelo del convertidor elevador

Como su propio nombre indica, el convertidor elevador tiene como misión la de elevar la tensión transferida por la planta de paneles fotovoltaicos. El esquema general de un convertidor elevador es el que se muestra en la figura 2.2. Su funcionamiento, en general, es sencillo: cuando el interruptor está cerrado la bobina es capaz de acumular energía para descargarla en la etapa de salida cuando el interruptor se abre. El interruptor representa un transistor en estados de corte y saturación.

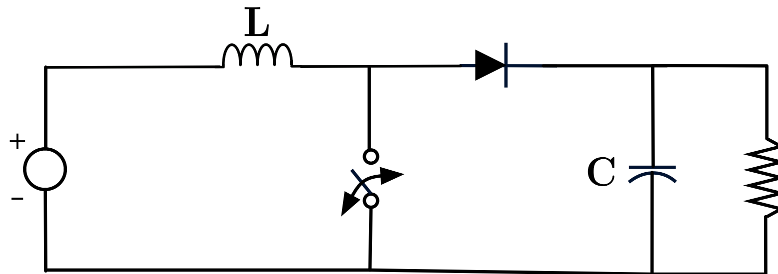


Figura 2.2: Esquema simplificado de un convertidor elevador.

Como puede observarse en la figura 2.1, en el modelo realizado en PLECS, además del condensador, también se ha sustituido el transistor y el diodo por una rama completa por razones económicas y técnicas, ya que el uso de dos transistores permite obtener cierta bidireccionalidad. El modelo de transistor a elegir dependerá de los niveles de tensión, de la corriente y la frecuencia. Para este caso en concreto, se trabajará con el modelo de IGBTs. También hay que decir que el modelo de los transistores ofrecido por PLECS, es lo suficientemente realista como para que la generación de tiempos muertos sea necesaria para evitar un posible cortocircuito.

El valor de los parámetros de diseño del convertidor del sistema de la figura 2.1 que se va a simular son:

- L: inductancia de la bobina, su valor es de 5 mH.
- C: capacidad del condensador salida de los paneles, su valor es de 5 mF.

- V_{-dc}: tensión requerida del bus de continua, su valor es de 600 V.

Los demás elementos son ideales a efecto de simulación, por lo que no es necesario atender a características como, por ejemplo, la tensión en inversa de los diodos.

2.2. Modelo de la planta de paneles

Como puede verse en la figura 2.3 la planta está compuesta por 8 paneles en serie (cada uno compuesto por 96 células en serie y 5 cadenas en paralelo), haciendo uso de diodos de paso para que, en aquellos paneles que estén generando una tensión negativa (condición de sombra), los diodos entren en conducción envitando reducir la potencia máxima que la hilera pueda dar. En caso de prescindir de ellos, algunas celdas podrían convertirse en cargas, disipando parte de la potencia producida por las demás. De manera análoga se coloca un diodo de bloqueo que aparece en la figura 2.1 para evitar las descargas nocturnas o que la hilera actúe como carga.

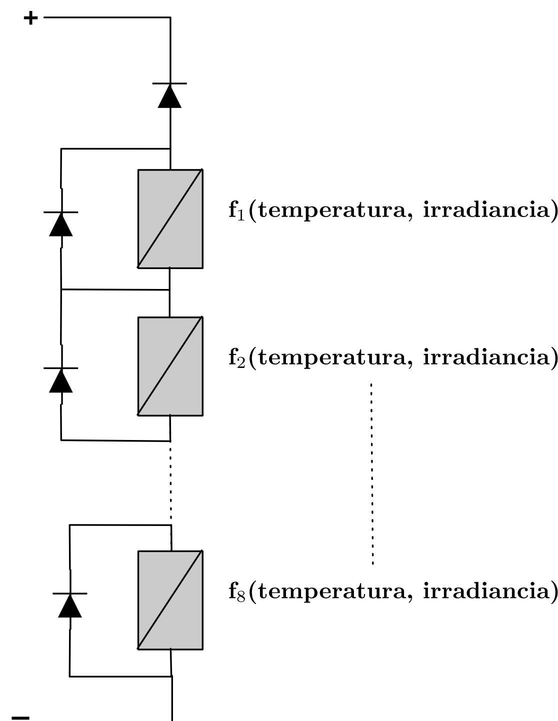


Figura 2.3: Planta de paneles.

Como modelo de paneles se ha usado uno de los bloques facilitados por el software para emular el comportamiento del panel $SunPowerSPR-305-WHT$, cuyo catálogo puede consultarse en [10]. A través de la máscara del bloque (figura 2.4) es posible simular cualquier modelo de panel si se conocen sus parámetros.

Internamente, el modelo está constinuido por una ganancia que representa el valor de irradiancia nominal, por lo que la entrada de irradiancia debe ser un porcentaje sobre la irradiancia nominal en tanto por uno. La ecuación que regula la corriente generada por la unidad viene dada por la ecuación (2.1). Este valor de corriente se satura inferiormente a cero para evitar dar una corriente en sentido negativo.

$$I_m = I_{ph} - I_O \left[e^{\frac{e(V+R_s I)}{Vm}} - 1 \right] \tag{2.1}$$

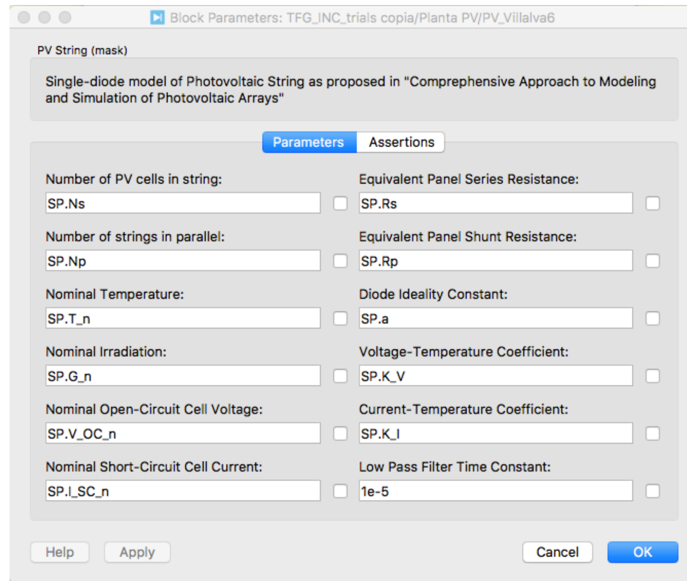


Figura 2.4: Máscara del bloque panel.

2.3. Resultados caracteristicos del modelado

Para comprobar el funcionamiento de la planta y su diseño se realiza una simulación que permite caracterizar correctamente los parámetros que definen las curvas de las figuras 2.5 y 2.6. El punto de máxima potencia está en 431,6 V y 28,1 A para dar una potencia de 12,1 kW.

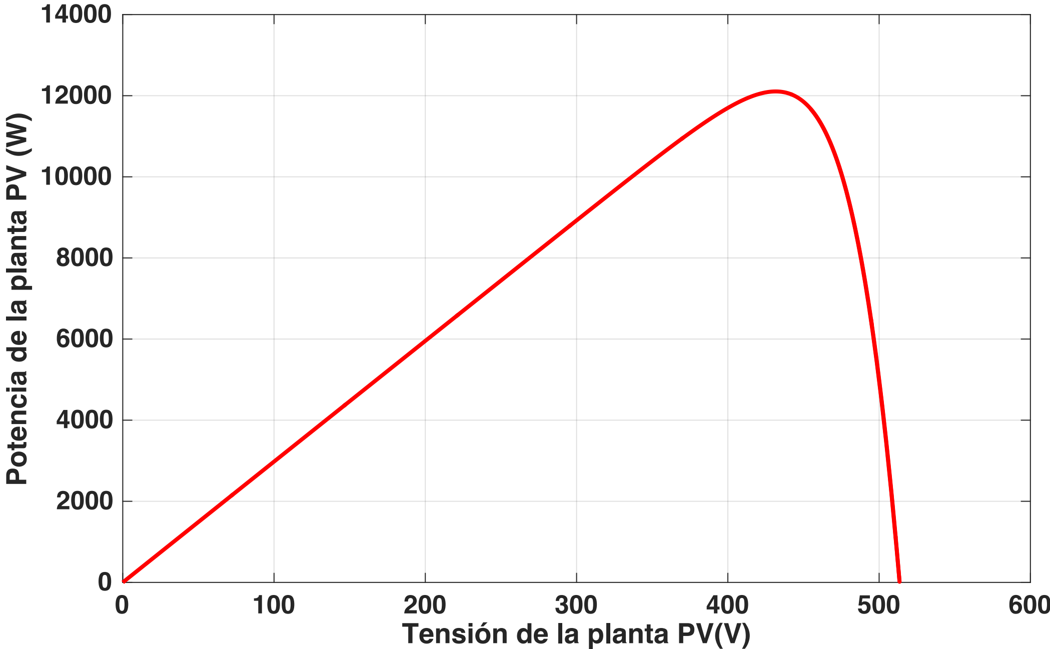


Figura 2.5: Curva P-V de la planta de paneles.

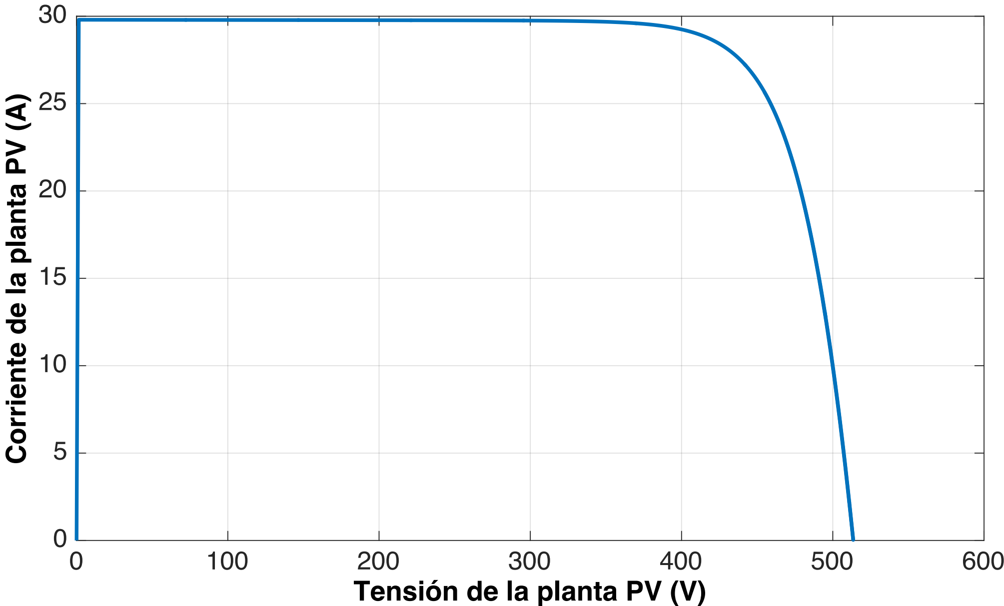


Figura 2.6: Curva I-V de la planta de paneles.

Capítulo 3

Algoritmos MPPT y diseño del control

Los algoritmos de seguimiento del punto de máxima potencia (MPPT) son utilizados en sistemas fotovoltaicos para maximizar la energía entregada por los paneles. La variación de esta energía se consigue a través de un cambio del ciclo del convertidor de potencia, en este caso, del convertidor Boost.

En este capítulo se explica los algoritmos más usados para el cálculo del MPP: P&O, método de conductancia incremental y el método de tensión constante, aunque se centrará más en el método de conductancia incremental, que es el se va a implementar en el modelo. También existen otros algoritmos menos utilizados como el fuzzy logic control o el de redes neuronales. Estos algoritmos difieren en su complejidad, velocidad de convergencia, efectividad y costo, como puede comprobarse en [1].

Por último, también se desarrollará el diseño de los lazos interno y externo de control del convertidor elevador con sus respectivas ecuaciones.

3.1. Método Perturbance and Observe (P&O)

Se trata de uno de los algoritmos más utilizados para el cálculo del MPP. Su comportamiento se especifica en el diagrama de flujos representado en la figura 3.1.

Como su propio nombre indica, se funcionamiento se basa en la propia observación del sistema: para un instante de tiempo k se establece una V_{ref} y en el instante $k+1$ medimos el valor de voltaje a la salida del panel. Atendiendo a la curva característica potencia-voltaje de una planta de paneles, si el voltaje en el instante $k+1$ es mayor que en el instante k y la potencia ha aumentado, quiere decir que el punto de trabajo se encuentra a la izquierda del punto de máxima potencia y por tanto, se debe incrementar V_{ref}. En caso contrario, estaría posicionado en la parte derecha con pendiente negativa, por lo que la referencia de voltaje debe disminuir. Dichos aumentos o decrementos en la tensión de referencia de los paneles se corresponderá, respectivamente, con un incremento o decremento del valor del ciclo de trabajo del convertidor elevador.

Las razones principales por el cuál se trata de un algoritmo muy utilizado reside básicamente en la facilidad de su implementación y su bajo coste computacional. Sin embargo, este algoritmo produce oscilaciones alrededor del MPP (lo cual conlleva a más pérdidas por conmutación), aunque el método tradicional puede ser modificado de tal forma que con factores de aceleraciones e incrementos de la referencia variable sea posible favorecer la convergencia del algoritmo. Otro inconveniente que hace que no sea del todo eficiente es que no presenta muy buena respuesta cuando la irradiancia es alta.

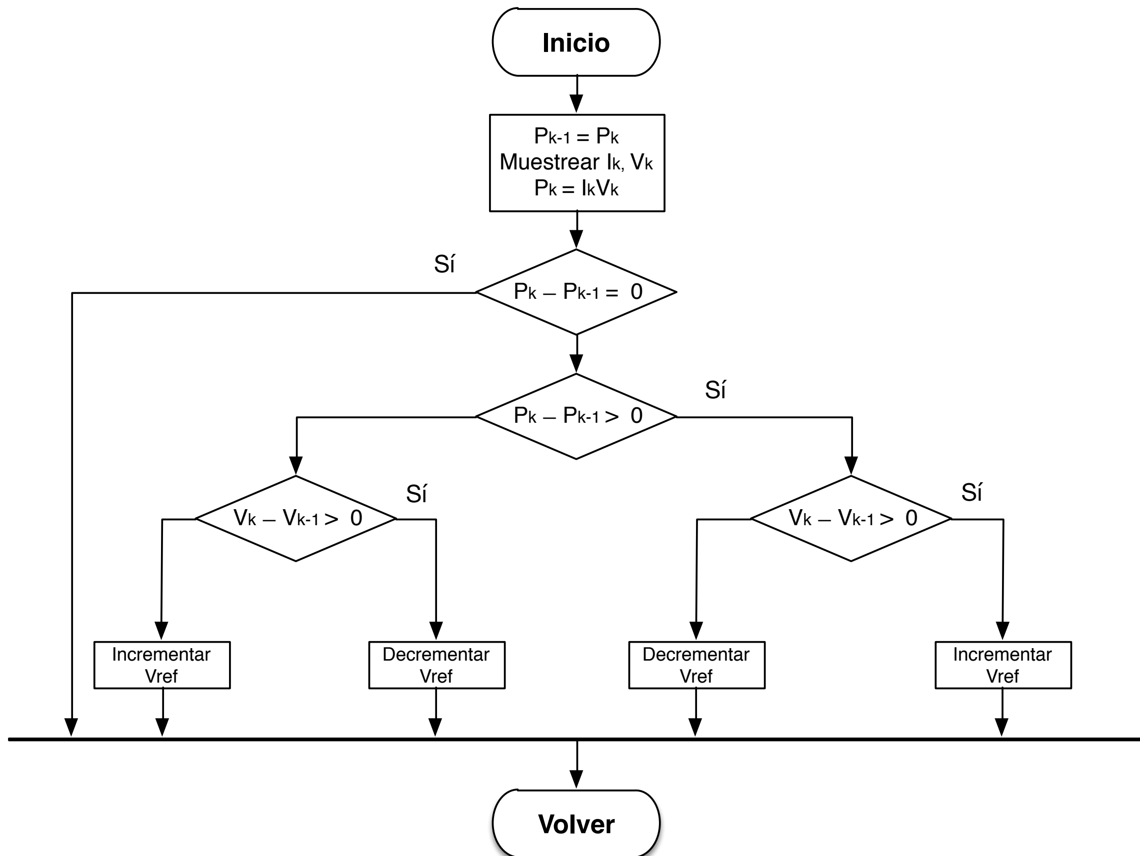


Figura 3.1: Método Perturbance and Observe (P&O).

3.2. Método de tensión constante

El método de tensión constante o método de la tensión de circuito abierto fraccional se basa en una aproximación proporcional entre la tensión del punto de máxima potencia y la tensión de circuito abierto. Se considera que

$$V_{MPP} \simeq K_v \cdot V_{oc} \tag{3.1}$$

donde K_v es un factor de proporcionalidad comprendido entre, aproximadamente, 0,71 y 0,78. Su algoritmo se representa esquemáticamente en la figura 3.2.

La aproximación (3.1) resulta ser válida si se entiende que la tensión del panel varía relativamente poco con la irradiación.

Comparte las mismas ventajas que el algoritmo P&O: fácil y bajo coste de implementación. El inconveniente que presenta es que de forma general, no es capaz de trabajar de una manera más o menos precisa en el punto de máxima potencia. Este efecto se hace cada vez más considerable cuanto menor es la irradiancia. También son considerables las pérdidas de potencia recurrente, ya hay que detener la generación de potencia para poder conocer la tensión de circuito abierto.

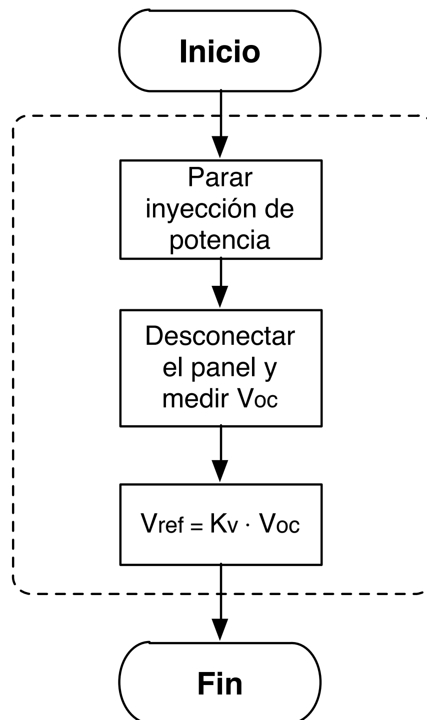


Figura 3.2: Método de la tensión a circuito abierto fraccional.

3.3. Método de conductancia incremental

El fundamento del algoritmo de conductancia incremental reside en el cálculo de la pendiente de la curva característica de potencia vista en el capítulo 1, concretamente, en la figura 1.5.

Como ocurre con el máximo de cualquier función matemática, la pendiente en el MPP es nula. Igualmente, como indica la figura 3.3, según se esté a la izquierda o derecha del máximo, la pendiente será, respectivamente, positiva o negativa. Por este motivo, este método utiliza el cálculo de la pendiente como se expresa en la ecuación (3.2).

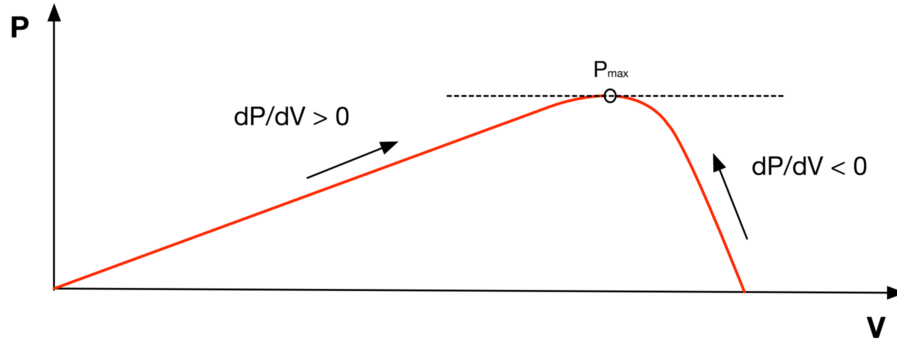


Figura 3.3: Concepto del algoritmo de conductancia incremental.

$$\frac{\mathrm{d}P}{\mathrm{d}V} = \frac{\mathrm{d}(VI)}{\mathrm{d}V} = I + V\frac{\mathrm{d}I}{\mathrm{d}V} \simeq I + V\frac{\Delta I}{\Delta V} \tag{3.2}$$

Por tanto, los tres posibles valores para la pendiente serán:

- $\frac{\Delta I}{\Delta V} = -\frac{I}{V}$: la pendiente es nula, estamos en el MPP.
- $\frac{\Delta I}{\Delta V} > -\frac{I}{V}$: estamos a la izquierda del MPP.
- $\frac{\Delta I}{\Delta V} < -\frac{I}{V}$: nos situamos a la derecha del MPP.

donde $\frac{I}{V}$ se le denota como conductancia instantánea y $\frac{\Delta I}{\Delta V}$ como conductancia incremental.

Una vez entendido el fundamento teórico del algoritmo resulta sencillo comprender el diagrama de flujos propuesto en la figura 3.4.

Al igual que ocurre con otros algoritmos, este método también se presta al uso de factores de aceleración para facilitar su convergencia y evitar posibles oscilaciones alrededor del MPP. Para el desarrollo de este proyecto, se propone el algoritmo que se muestra en la figura 3.5. En dicha implementación los factores de aceleración se denotan como K_1 y K_2: el incremento de la tensión referencia se calcula como un porcentaje de la pendiente de la potencia o del incremento de corriente entre los intervalos k y $k+1$. El decremento o incremento de V_{ref} se consigue por el signo de ΔV_{ref}, ya que se corresponderá con el signo de el incremento de la pendiente o de la corriente.

Las razones por las cuáles se ha optado por esta solución para la búsqueda del punto de básicamente son, principalmente, la buena respuesta que presenta el algoritmo ante variaciones bruscas de la irradiancia y prácticamente, la inexistencia de oscilaciones alrededor del punto deseado de trabajo. Por el contrario, se debe que asumir una mayor complejidad y coste computacional, aunque con las tecnologías y recursos existentes esto no suponga un gran problema.

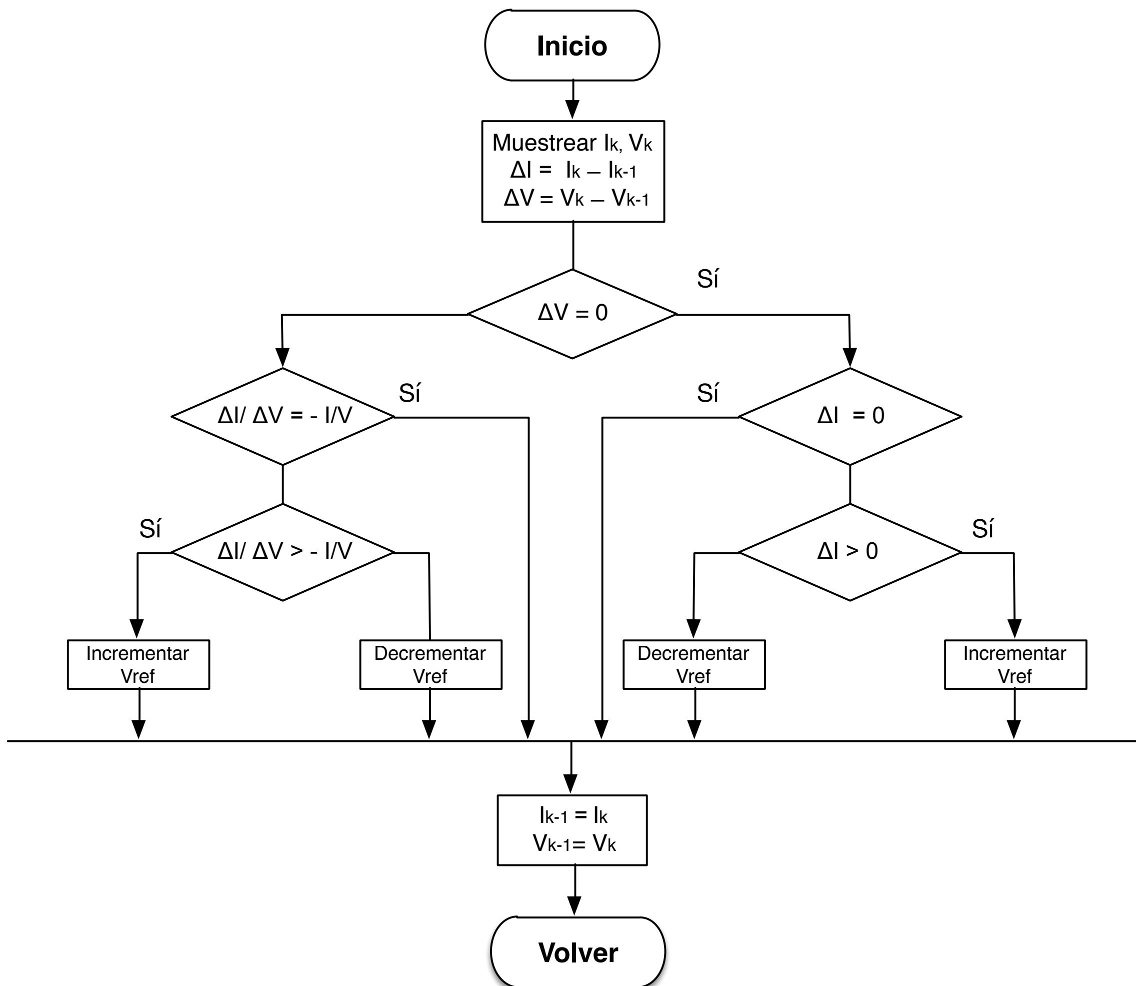


Figura 3.4: Método conductancia incremental convencional.

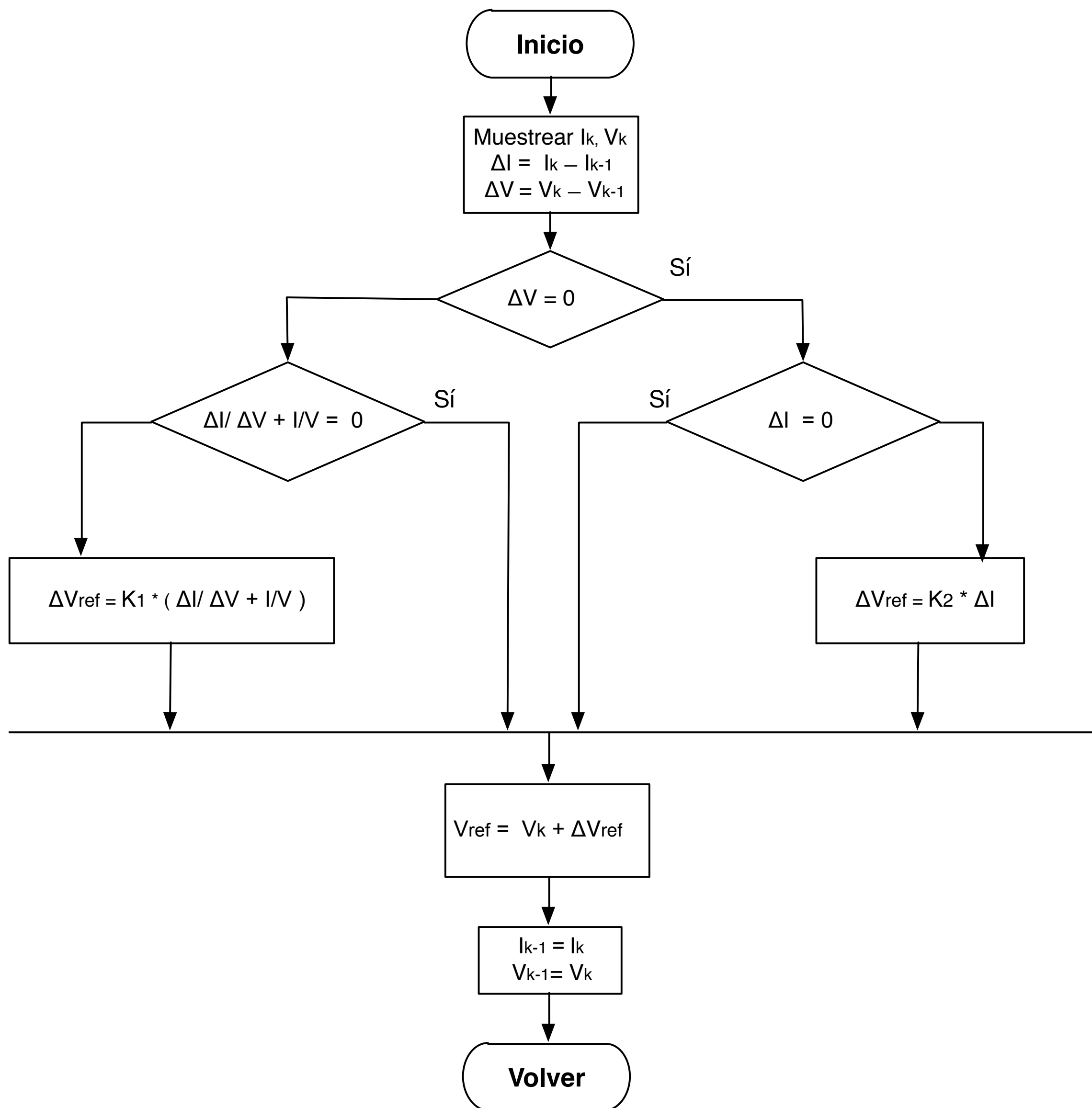


Figura 3.5: Método conductancia incremental propuesto.

3.4. Formulación del lazo externo de control

Para el diseño del lazo externo de control se realiza un balance de potencia en el condensador de salida de la planta de paneles fotovoltaicos. Se parte entonces del modelo que se presenta en la figura 3.6. La fuente constante de corriente modela la fuente de energía renovable y, por su parte, como se ha comentado en capítulo 2, el control del inversor asegurará una tensión constante en el condensador de la etapa de salida del convertidor Boost (fuente de tensión constante).

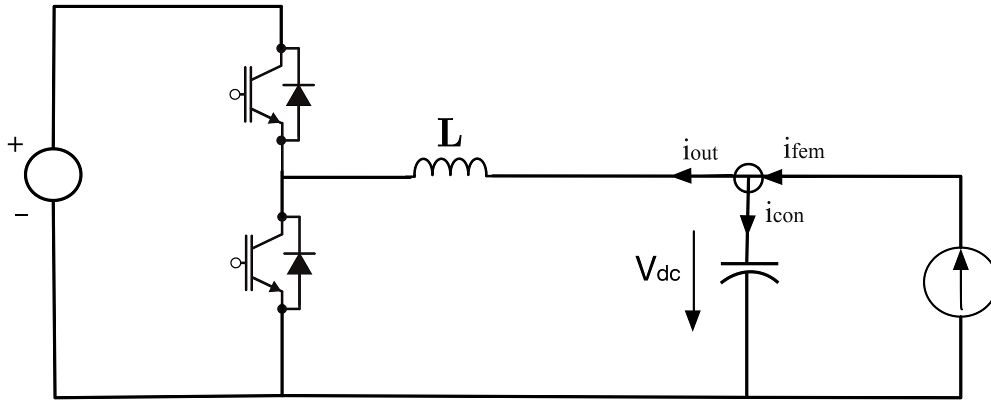


Figura 3.6: Esquema del convertidor y la planta de paneles.

Se denota como i_{out} a la corriente que circula por la bobina e i_{fem} a la de la fuente de energía renovable (FEM). Aplicando la ley de Kirchoff en el nodo señalado en la figura 3.6, se tiene que:

$$C\frac{\mathrm{d}V_{dc}}{\mathrm{d}t} + i_{out} = i_{fem} \tag{3.3}$$

Multiplicando en ambos miembros de la ecuación (3.3) por V_{dc} se obtiene el siguiente balance de potencias instantáneas:

$$CV_{dc}\frac{\mathrm{d}V_{dc}}{\mathrm{d}t} = p_{fem} - p_{out} \tag{3.4}$$

donde p_{out} es la potencia que debe salir del convertidor y p_{fem} la que proporciona la fuente. Esta última expresión puede reformularse como:

$$C\frac{1}{2}\frac{\mathrm{d}V_{dc}^2}{\mathrm{d}t} = p_{fem} - p_{out} \tag{3.5}$$

Para resolver la ecuación resulta conveniente plantear un cambio de variables que permita resolver una ecuación diferencial ordinaria de primer orden lineal.

$$p_{out} = -C\frac{\mathrm{d}z}{\mathrm{d}t} + p_{fem} \tag{3.6}$$

donde $z \equiv \frac{V_{dc}^2}{2}$, por lo que encontrar V_{dc}^* de referencia equivale a encontrar z^*.

El comportamiento deseado de la evolución del sistema ante un cambio en la referencia y la evolución del error, $\widetilde{z}$, se muestran en la figura 3.7. En color rojo y con línea intermitente se representa el cambio en la referencia y en azul continuo la evolución del sistema y el error.

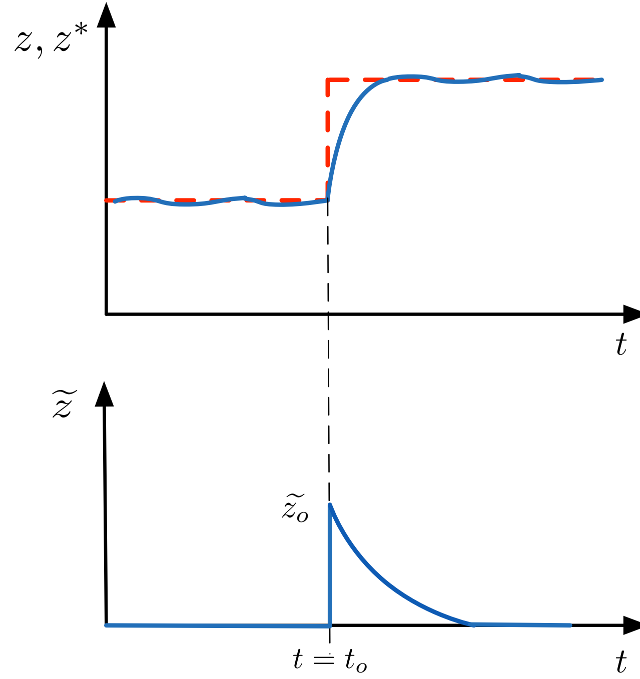


Figura 3.7: Evolución deseada del sistema.

El comportamiento del error, $\widetilde{z}$, se corresponde con la ecuación (3.7),

$$\widetilde{z}(t) = \widetilde{z}_o \; e^{\frac{-T}{\tau}} \tag{3.7}$$

que expresado en forma de ecuación diferencial queda como sigue:

$$\tau \frac{\mathrm{d}\widetilde{z}}{\mathrm{d}t} + \widetilde{z} = 0 \tag{3.8}$$

Con todo esto, se puede definir p_{out} de la ecuación (3.6) de tal forma que la dinámica de $\widetilde{z}$ coincida con (3.8).

Sabiendo que $\widetilde{z} = z - z^*$, donde z^* es el valor de referencia y z el valor alcanzado por el sistema en un instante, se suman y se restan los términos $C\frac{\mathrm{d}z^*}{\mathrm{d}t}$ y $K_p\widetilde{z}$ a la ecuación (3.6).

$$C\frac{\mathrm{d}z}{\mathrm{d}t} + C\frac{\mathrm{d}z^*}{\mathrm{d}t} - C\frac{\mathrm{d}z^*}{\mathrm{d}t} + K_p\widetilde{z} - K_p\widetilde{z} = p_{fem} - p_{out} \tag{3.9}$$

Reorganizando la expresión anterior, queda como:

$$C\frac{\mathrm{d}\widetilde{z}}{\mathrm{d}t} + C\frac{\mathrm{d}z^*}{\mathrm{d}t} + K_p\widetilde{z} - K_p\widetilde{z} = p_{fem} - p_{out} \tag{3.10}$$

por lo que p_{out} debe definirse como:

$$p^*_{out} = p_{fem} - C\frac{\mathrm{d}z^*}{\mathrm{d}t} + K_p\widetilde{z} \tag{3.11}$$

La potencia generada por la fuente puede considerarse una perturbación del sistema que varía lentamente (ya que los cambios de temperatura e irradiancia no presentan variaciones bruscas en la realidad), por lo que se puede estimar como un término integral. De esta forma, en régimen permanente, la derivada de la referencia es nula y se tiene concluye que:

$$p^*_{out} = K_p\widetilde{z} + K_i\int \widetilde{z}\mathrm{d}t \tag{3.12}$$

donde $\widetilde{z} = \frac{(V_{dc})^2}{2} - \frac{(V^*_{dc})^2}{2}$.

3.5. Formulación del lazo interno de control

Para el desarrollo del lazo interno de control, es necesario partir del modelo simplificado que se muestra en la figura 3.8. En dicha figura, V_m representa una tensión modulada, de tal forma que el comportamiento de la tensión de salida del convertidor puede representarse según la ecuación (3.13), donde V_o es la tensión de salida del convertidor, D el ciclo de trabajo y V_m representa la tensión modulada.

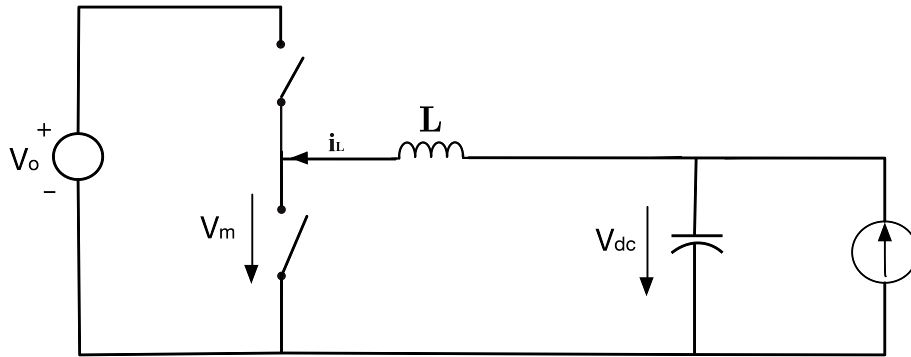


Figura 3.8: Etapa de evacuación de potencia del convertidor Boost.

$$\mathrm{V}_m = D\mathrm{V}_o \tag{3.13}$$

Para desarrollar el control, es necesario plantear las ecuaciones en la malla de la bobina, para llegar a la igualdad mostrada en la ecuación (3.14).

$$V_{dc} = L\frac{\mathrm{d}i_L}{\mathrm{d}t} + \mathrm{V}_m \tag{3.14}$$

Como se desea que el comportamiento del error sea como el que se muestra en la figura 3.7, se suman y se restan los términos $L\frac{\mathrm{d}i_L^*}{\mathrm{d}t}$ y $K_p\widetilde{i}$, siendo $\widetilde{i} = i_L - i_L^*$. Así la ecuación (3.14) queda como:

$$V_{dc} = L\frac{\mathrm{d}i_L}{\mathrm{d}t} + L\frac{\mathrm{d}i_L^*}{\mathrm{d}t} - L\frac{\mathrm{d}i_L^*}{\mathrm{d}t} + K_p\widetilde{i} - K_p\widetilde{i} + \mathrm{V}_m \tag{3.15}$$

Así es posible definir una V_m^* de referencia que haga que la dinámica sea la deseada. En la ecuación 3.16 se define V_m^* tal que $\tau\frac{\mathrm{d}\widetilde{i}}{\mathrm{d}t} + \widetilde{i} = 0$.

$$\mathrm{V}_m^* = \mathrm{V}_{dc} - L\frac{\mathrm{d}i_L^*}{\mathrm{d}t} + K_p\widetilde{i} \tag{3.16}$$

En este caso, el término V_{dc} de la ecuación 3.16 es un término de realimentación que no hace falta considerarlo, pero que si se hace, mejora la dinámica del sistema proporcionando un control más rapido.

Para compensar los errores y simplificaciones de modelado, se añade un término integral compensatorio, de tal manera que V_m^* se define, finalmente, según la ecuación (3.17), teniendo en cuenta que la derivada de la referencia en régimen permanente es nula.

$$\mathrm{V}_m^* = K_p\widetilde{i} + K_i\int\widetilde{i}\mathrm{d}t \tag{3.17}$$

Capítulo 4

Processor In the Loop (PIL)

Cuando se desarrollan algoritmos de control embebidos o empotrados, es bastante habitual probarlos ejecutándolos dentro de un simulador. Utilizando PLECS (o cualquier otra herramienta de simulación como Matlab-Simulink), esta tarea resulta sencilla gracias a los bloques proporcionados como C-Script. Se trata de herramientas que permiten compilar el código y ejecutar los algoritmos dentro de un entorno de simulación.

Sin embargo, el enfoque de PIL permite ejecutar los algoritmos de control en el propio hardware embebido. En lugar de leer la lectura de los sensores del sistema, estos valores son calculados por la herramienta de simulación y se utilizan como entradas del algoritmo que se está ejecutando en la plataforma real. Del mismo modo, las salidas que se obtienen del algoritmo de control ejecutado en el procesador del hardware se devuelven a la simulación.

La intención de este capítulo no es otra que la de mostrar cómo funciona esta herramienta para que, con la ayuda de capítulos posteriores, pueda entenderse correctamente el código presentado en los anexos, así como secciones futuras.

4.1. Visión general

Considerando cierto nivel de abstracción, el funcionamiento de PIL puede entenderse fácilmente con la figura 4.1. En la parte superior de la figura se representa el entorno de simulación, mientras que la parte inferior representa esquemáticamente el hardware real.

Las variables de entrada de la tarjeta, tales como las medidas de corriente y voltaje, se proporcionan a la misma con valores obtenidos en la simulación de PLECS. De manera análoga, las variables de salida del hardware, como por ejemplo el registro de comparación del PWM, son leídos y realimentados a la simulación.

De esta manera se pueden definir dos tipos de variables: aquellas cuyo valor es proporcionado por PLECS y que son leídas por el DSP, denominadas como *Override Probes*, y las *Read Probes*, aquellas cuyo valor es proporcionado por el algoritmo de control y que son devueltas a la simulación.

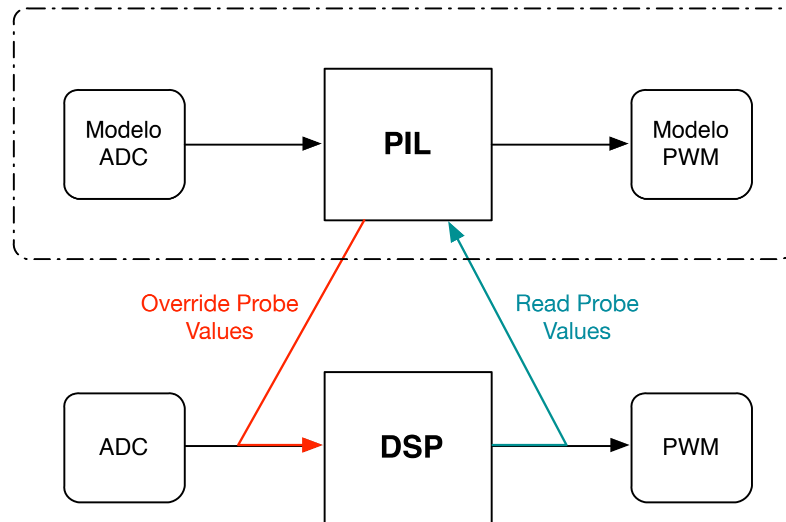


Figura 4.1: Esquema de funcionamiento del Process In the Loop.

Mientras que las *Override Probes* se establecen y las *Read Probes* son leídas por la simulación, la ejecución del algoritmo de control debe detenerse. En otras palabras, las tareas de control deben permanecer paradas mientras PLECS está actualizando el modelo que se está simulando. Esto quiere decir que el algoritmo de control opera con un paso de simulación durante una simulación PIL. Sin embargo, cuando el código se está ejecutando, su comportamiento es idéntico al que se espera en una ejecución en tiempo real; es por ello que este modo de funcionamiento se conoce como comportamiento en psuedo-tiempo real.

Después de recibir la interrupción del hardware, el sistema detiene la ejecución del control y entra en un lazo de comunicación donde los valores de los dos tipos de variables se intercambian con el modelo de PLECS. Una vez que se recibe un nuevo paso de simulación, el despacho de tareas se reinicia y la tarea de control (o tareas) se ejecutan durante la duración de un período de interrupción. Lo único que se detiene es la tarea de control pero lo que es el objetivo en sí nunca se detiene, ya que la comunicación con PLECS debe mantenerse en todo instante.

El concepto de usar los tipos de variables *Override Probes* y *Read Probes* permite enlazar el código de control real que se ejecuta en una MCU con una simulación con PLECS sin la necesidad de recompilar específicamente para PIL.

Se puede pensar que las *Override Probes* y *Read Probes* son el equivalente de puntos de prueba que se pueden dejar en el software embebido siempre que se desee. Los módulos del software con tales puntos de prueba pueden ser ligados a una simulación PIL en cualquier momento. A menudo, estas variables están configuradas para acceder a los registros de los periféricos de la MCU, como los convertidores analógico digitales (ADC) o el PWM.

Para permitir transiciones seguras y controladas entre la ejecución en tiempo real

del código de control, la conducción de una planta real y la pseudo ejecución en tiempo real, en conjunto con una planta simulada, existen dos modos del PIL:

- Normal Operation: como su propio nombre indica, se trata del funcionamiento normal, en el que se inhibe las simulaciones.
- Ready for PIL: corresponde con el estado de simulación.

La transición entre los dos modos de operación se puede controlar con la propia aplicación embebida, usando por ejemplo un conjunto de señales digitales de entrada o bien, manualmente desde PLECS.

4.2. Bloque PIL

El bloque PIL (figura 4.2) vincula un procesador a una simulación en PLECS haciendo que las variables configuradas en el propio hardware como *Override Probes* y *Read Probes* puedan actuar como entradas o salidas del sistema.

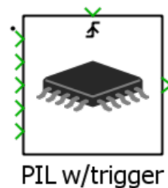


Figura 4.2: Bloque PIL.

La ejecución del bloque puede ser en un modo discreto-periódico fijo (configurando el tiempo de muestro en un valor positivo) o se puede seleccionar un tiempo de muestreo heredado.

Es posible activar un puerto de habilitación mediante el cuadro de disparo externo que presenta la máscara del bloque, tal y como se desarrolla en este proyecto. Esta característica es de gran utilidad cuando la interrupción forma parte del circuito de PLECS, como ocurre con los modelos del ADC o PWM. Normalmente, el tiempo de muestreo heredado se usa al mismo tiempo que el puerto de disparo. De esta forma, si se especifica una velocidad discreta, el puerto de disparo se muestreará a la velodidad especificada. También es posible ajustar el retraso en la salida para aproximarlo al cálculo del procesador, aunque también se puede configurar con un retraso nulo tal y como puede verse en la figura 4.3.

El bloque extrae los nombres de las variables que han sido definidas como *Override Probes* y *Read Probes* de un archivo de extensión *.out* seleccionado. Se trata en código máquina (proporcionado por una herramienta que se explicará posteriormente) y que contiene toda la información de la configuración de variables así como del propio código

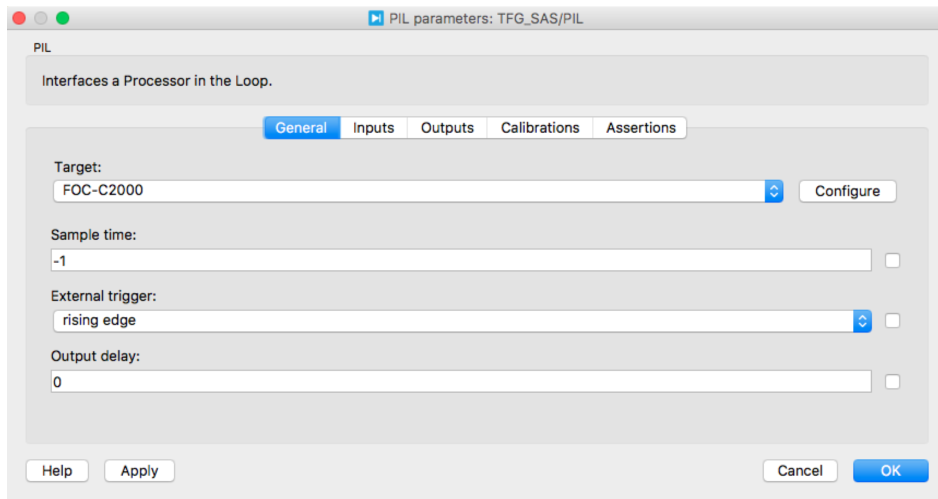
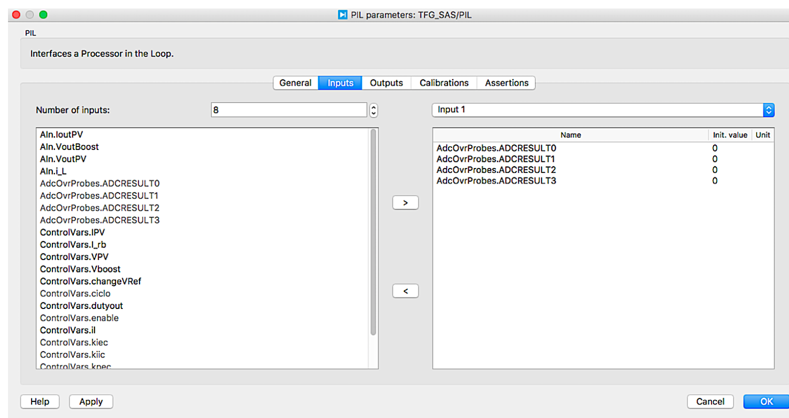
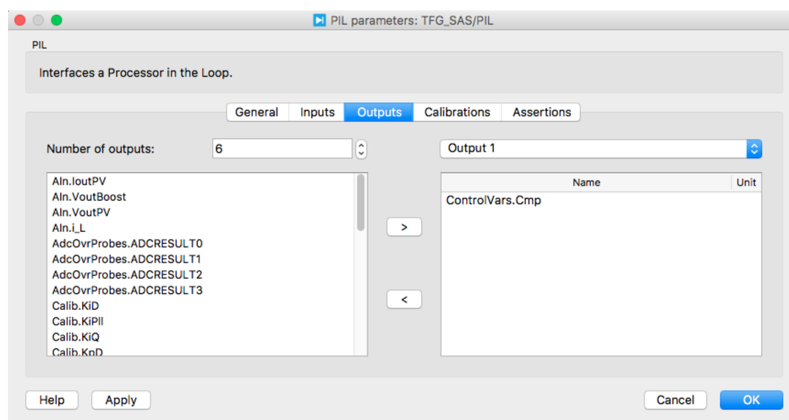


Figura 4.3: Configuración general del PIL.



(a) Entradas



(b) Salidas

Figura 4.4: Entradas y salidas de PIL.

de control (dicho archivo se selecciona accediento en *Configure* en la máscara proporcionada en la figura 4.3). De esta manera presenta una lista de variables que deben ser seleccionadas manualmente como entradas o salidas (ver figura 4.4).

Aunque no se utiliza para el desarrollo de este proyecto, también es posible configurar condiciones iniciales para las *Override Probes* en la sección de *Calibrations.* Este apartado permite modificar los ajustes del código embebido como coeficientes de filtros.

4.3. Entorno de trabajo

PIL está desarrollado para familias de procesadores específicas, que tienen la funcionalidad incorporada necesaria para la operación PIL. Actualmente, dichos marcos y aplicaciones demo asociadas están disponibles para las familias de *ST Microelectronics 32bit F4*, *Microchip* $dsPIC33F$ *MCU* y *Texas Instruments (TI)* $C2000^{TM}$. En cualquier caso, el soporte para otras plataformas puede desarrollarse si se cumple los requisitos que aparecen en [4]:

- Las herramientas de generación de código son capaces de generar archivos binarios del formato ELF que contienen información de depuración de $DWARF$.
- La unidad menos direccionable (LAU) del procesador no supera los 16 bits.
- El ancho de la dirección del procesador no supera los 32 bits.

4.3.1. PIL Prep Tool

Como puede deducirse de su traducción, se trata de una herramienta proporcionada por *Plexim* y que forma parte del entorno de ejecución. Esta herramienta debe ejecutarse en la compilación del código y se encarga de analizar las macros especificadas de PIL para generar automáticamente archivos auxiliares que se compilan y se enlazan con las macros del código. Estos archivos auxiliares contienen funciones de inicialización e incluyen un identificador único global (GUID) que permite identificar el código embebido, de tal manera que a una compilación sólo le corresponde un único fichero *.out*.

4.3.2. Variables

Para la definición de los dos tipos de variables que se pueden definir para la comunicación con PLECS, es necesario declararlas a través de macros e instrucciones facilitadas por *Plexim*.

4.3.2.1. Read Probes

Estas variables son las que reciben un valor en el entorno embebido y son enviadas a la simulación de PLECS.

Para declarar una variable como *Read Probe* se utiliza la macro `PIL_READ_PROBE` tal y como se muestra a continuación.

```
PIL_READ_PROBE(tipo de variable, nombrevariable, n, referencia,
"unidad");
```

Así, `tipo de variable` se refiere al tipo de variable que se pretende declarar (entero, flotante, etc), `nombrevariable` es el nombre que se le quiere dar a la variable, `n` y `referencia` son variables de prescalado del valor que toma la variable (el valor `Valor_read` leído por la simulación corresponde con la ecuación (4.1)) y `unidad` es la unidad que lleva la variable declarada (V para voltios, A para amperios, etc).

$$\text{Valor_read} = \frac{\text{Valor}}{2^n}\text{referencia} \tag{4.1}$$

A continuación se muestra un ejemplo de declaración de una variable tipo `float` cuyo nombre es `Vsalida`:

```
PIL_READ_PROBE(float, Vsalida, 10, 5, "V");
```

El valor leído por PLECS para un valor de la variable de 400 V será:

$$\text{Valor_read} = \frac{400}{2^{10}}\,5 = 1{,}95V$$

Esta macro permite la definición de una variable de cualquier tipo (int, float...) y también es reconocida por la herramienta *Pil Prep Tool* generando la siguiente línea en el archivo autogenerado.

```
PIL_SYMBOL_DEF(nombrevariable, n, referencia, "unidad");
```

Esta macro, a su vez, se trata de una variable tipo estructura estática de tipo constante:

```
typedef struct
{
int q;
float ref;
char *unit;
} pil_var;

const pil_var PIL_V_nombrevariable = {n, referencia, "unidad"}
```

4.3.2.2. Override Probes

Este tipo de variables son aquellas que se obtienen de la simulación para usarlas en la tarea de control y se definen mediante la macro `PIL_OVERRIDE_PROBE`. Por tanto, sólo las variables definidas como *Override Probes* podrán ser configuradas como entradas del bloque PIL.

```
PIL_OVERRIDE_PROBE(tipo de variable, nombrevariable, n, referencia,
"unidad");
```

Esta macro se expande en una definición de variable incrementada con dos símbolos auxiliares que permiten que la variable sea transmitida al módulo PIL.

```
struct nombrevariable
{
int variable
int variable_probeV;
int variable_probeF;
};
```

Durante el análisis del archivo binario para la recopilación de los símbolos, PLECS detecta variables con las definiciones `_probeF` y `_probeV` e identificas las variables *Override Probes*.

Al igual que ocurría con las *Read Probes*, la herramienta *PIL Prep Tool* reconoce la macro `PIL_OVERRIDE_PROBE` y genera la siguiente macro auxiliar:

```
PIL_SYMBOL_DEF(nombrevariable, n, referencia, "unidad");
```

Este tipo de variables tiene un comportamiento similar al de un interruptor con dos estados: uno de avance y otro de anulación. En el primero, el valor de la variable es proporcionado por la aplicación embebida, mientras que en el segundo el valor es suministrado por PLECS.

Su estado se almacena en la variable auxiliar `_probeF` y se puede conmutar dinámicamente en tiempo de ejecución. Gracias a esto, se puede usar la misma compilación de la aplicación embebida para controlar el hardware real o ejecutar la simulación en PLECS, simplemente cambiando el modo de estas variables sin necesidad de recompilar. Si se utiliza el valor suministrado por PLECS, el valor está almacenado en la variable auxiliar `_probeV`.

Para que la interacción con PLECS se lleve a cabo correctamente, el código en el hardware debe acceder a estas variables a través de las siguientes macros:

- `INIT_OPROBE(probe)`: debe ser ejecutada en la inicialización del programa para inicializar la variable.

- `SET_OPROBE(probe)`: asigna un valor a la variable.

La propia herramienta de PIL genera una función llamada `PilInitOverrideProbes()` que contiene la macro de inicialización de todas las variables *Override Probes*. Como es de esperar, esta función debe ejecutarse en la inicialización del código antes de que alguna variable de este tipo sea utilizada.

4.3.3. Calibraciones

PIL también ofrece un entorno para las calibraciones de variables como las ganancias o umbrales del código embebido. Esta operación se lleva a cabo a través de la macro `PIL_CALIBRATION` tal y como se muestra a continuación.

```
PIL_CALIBRATION(tipodevariable, variable,n, referencia, "unidad",
minrango, maxrango, valorpordefecto);
```

Se puede observar la semejanza que presenta con las macros introducidas anteriormente. Los tres nuevos valores del final del comando son, respectivamente, el valor mínimo permitido, el valor máximo y su valor por defecto.

Esta macro desemboca en la definición de una macro `PIL_SYMBOL_CAL_DEF`, la cual proporciona la información que se necesita para que PLECS interprete y maneje la calibración. Para ello es necesario llamar a la función `PilInitCalibrations()` en el inicio de la ejecución del código para establecer el valor de la variable al valor de defecto. Para deshacer los cambios realizados durante la simulación, esta función también debe ser llamada en `PIL_CLBK_TERMINATE_SIMULATION`, que se explica en el apartado 4.4.1.

4.3.4. Identidad del código

Ya se ha comentado en la sección 4.3.1 que es importante asegurarse de que el archivo binario seleccionado en el bloque PIL coincida con el código que se está ejecutando en el DSP. La explicación reside en que el bloque PIL extrae la dirección de una variable dada de la información del debug contenida en el archivo binario. Por tanto, si se asegura que el archivo es el correspondiente a la compilación es posible que el bloque acceda a ubicaciones de memorias erróneas.

Para conseguir esta identificación, se compara un identificador (GUID, que se genera al ejecutar la herramienta *Pil Prep Tool*) almacenado en el archivo *.out* con el suministrado por el hardware. Esta comprobación se realiza al inicio de la simulación pero es fácil comprobarlo manualmente en la propia configuración del bloque PIL: el propio bloque te informa del error y por tanto no estará disponible el modo *Ready for PIL*.

4.3.5. Agente remoto

El agente remoto es el que efectúa el enlace de comunicación (bien en serie o bien en paralelo) con PLECS y procesa los comandos recibidos de PLECS para acceder a las variables y se encarga de pasar el código durante los pasos de simulación.

4.3.5.1. LLamadas de comunicación

PIL interactúa con el driver de comunicación específico mediante funciones de comunicación. Estas funciones son:

- `CommCallback()` : se ejecuta en cada interrupción del sistema desde la llamada `PIL_beginInterruptCall()`, cuyo funcionamiento se justificará la sección 4.4.
- `BackgroundCommCallback()` : es llamada periódicamente por `PIL_backgroundCall()`, la cual también se explicará posteriormente en el apartado 4.4.

Dependiendo del tipo de comunicación que se quiera llevar a cabo es necesario hacer uso de las funciones explicadas en las siguientes subsecciones 4.3.5.2 y 4.3.5.3. Para el desarrollo del proyecto se utiliza la comunicación a través de la interfaz de comunicación serie del DSP.

4.3.5.2. Comunicación serie

Para llevar a cabo este tipo de comunicación, el asistente remoto utiliza una capa de mensajes y chequeo de errores, lo que hace el protocolo sea válido y apropiado para otros muchos enlaces.

Es necesario asegurar que no se pierdan los caracteres durante la comunicación, por lo que se llama a la función `CommCallback()` desde la interrupción para dar servicio. Existen dos llamadas para su implementación:

- `PIL_RA_serialIn()`: para la recepción.
- `PIL_RA_serialOut()`: para la transmisión.

4.3.5.3. Comunicación paralelo

La comunicación en paralelo suele ser atendida por medio de las dos siguientes llamadas:

- `PIL_RA_parallelIn()`: hace posible la recepción de un mensaje.
- `PIL_RA_parallelOut()`: llamada cuando se realiza el envío de un mensaje.

Los mensaje se intercambian directamente a través de vectores de 16 bits de enteros. No realiza ningún tipo de verificación y resulta bastante conveniente para intercambiar mensajes a través de memoria compartida, donde la posibilidad de que exista algún error en la transmisión es muy improbable.

4.4. Ejecución

Las acciones realizadas por `PIL_beginInterruptCall()` y `PIL_backgroundCall()` dependerá de si la simulación PIL se está ejecutando o no. Estas dos funciones se llaman de manera períodica por la aplicación embebida para habilitar la funcionalidad de PIL.

Para proporcionar un mejor entendimiento del funcionamiento en ejecución, se van a comparar la actuación en tiempo real y en pseudo-tiempo real.

En la figura 4.5 se observa la ejecución en tiempo real para un sistema con dos tareas de control, una rápida y prioritaria y otra más lenta que será interrumpida por la primera. Al principio de la rutina de servicio de interrupción del hardware, se ejecuta la función `PIL_beginInterruptCall()`, que solo llama a `CommCallback()`, que se encarga de establecer servicio para una comunicación serie y asegurar que no se pierde ningún caracter.

Por su parte, `PIL_backgroundCall()` se ejecuta como parte del bucle principal; se encarga de analizar los mensajes entrantes y llama al mismo tiempo a la función `BackgroundCommCallback()` si está configurado.

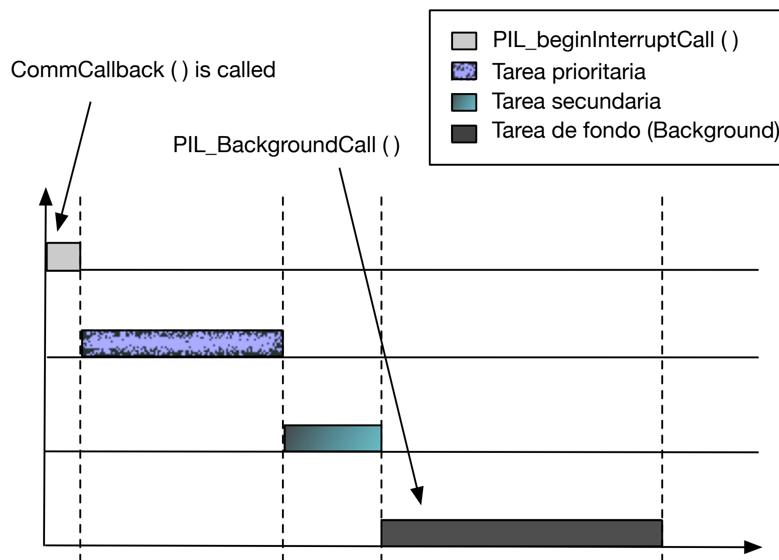


Figura 4.5: Ejecución en tiempo real.

Como se comentó en el apartado 4.1, el comportamiento de la ejecución con la herramienta PIL no coincide con las condiciones de ejecución en tiempo real. En la ejecución en pseudo-tiempo real, la ejecución de la tarea de control sigue el ritmo de la simulación del modelo en PLECS. Al comienzo de la interrupción de la rutina de servicio, se para la ejecución de la tarea mediante la macro `PILCLBK_STOP_TIMERS` para entrar en el bucle de comunicación.

En el bucle de comunicación se ejecutarán las llamadas de comunicación y el análisis de funciones. Comparando las figuras 4.5 y 4.6, se observan claramente las diferencias. En la primera de ellas, una vez que se recibe la petición para un nuevo “step”, se reaunda la tarea de control que se quiere ejecutar y continua hasta que se produce la siguiente interrupción del hardware.

4.4.1. Funcionalidad del Control Callback

Las acciones de control de comunicación entre el hardware y la simulación se llevan a cabo mediante las funciones `PilCallback()`. Se trata de una máquina de estados

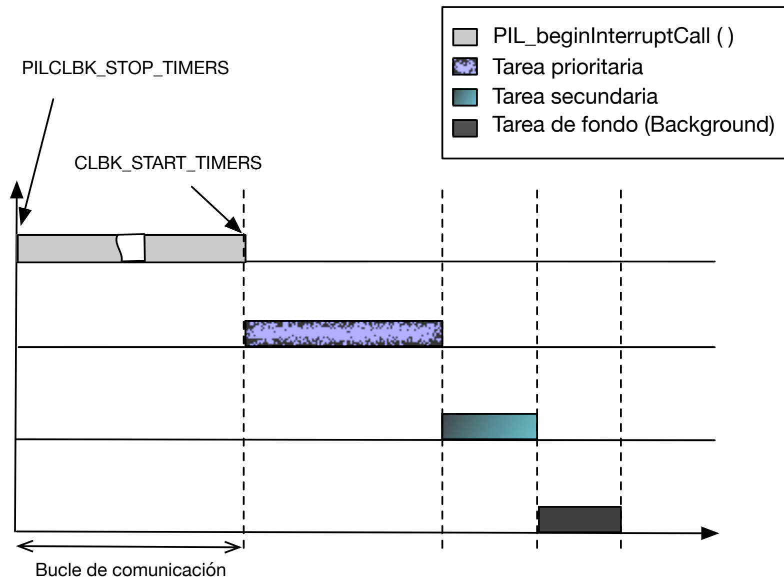


Figura 4.6: Ejecución en pseudo-tiempo real.

de la ejecución del PIL que permite tomar las acciones necesarias en función de una variable que mide el estado que se quiere tomar.

Para llevar a cabo esta implementación se define una máquina de estado que se está ejecutando en todo instante. A continuación se muestra el código con todas las acciones definidas:

```
#include "includes.h"

void PilCallback(PIL_CtrlCallbackReq_t aCallbackReq)
{
switch(aCallbackReq)
{

case  PIL_CLBK_ENTER_NORMAL_OPERATION_REQ:
    PIL_inhibitPilSimulation();
    EnableActuation();
return;


case PIL_CLBK_LEAVE_NORMAL_OPERATION_REQ:
    DisableActuation();
    PIL_allowPilSimulation();
return;
```

```
case PIL_CLBK_INITIALIZE_SIMULATION:
   ResetControl();
   ResetControlDispatcher();
return;


case PIL_CLBK_TERMINATE_SIMULATION:
    PilInitCalibrations();
break;


case PIL_CLBK_STOP_TIMERS:
EALLOW;
       SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 0;
EDIS;
return;


case PIL_CLBK_START_TIMERS:
EALLOW;
   SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;
EDIS;
return;

}

}
```

Aunque por los nombres resulta bastante intuitivo, se tratará de explicar brevemente cada uno de los estados posibles:

- `PIL_CLBK_ENTER_NORMAL_OPERATION_REQ`: este estado se llama cuando el DSP quiere trabajar en modo de operación normal (Normal Operation). Una vez que entra en dicho modo, se ejecuta `PIL_inhibitPilSimulation()` para indicar que se ha entrado en dicho estado de ejecución.

- `PIL_CLBK_LEAVE_NORMAL_OPERATION_REQ`: la tarjeta pide entrar en el entorno de simulación (Ready for PIL). Ejecutando la función `PIL_allowPilSimulacion()` la aplicación confirma que es posible entrar en dicho modo.

- `PIL_CLBK_INITIALIZE_SIMULATION`: se usa al principio de una simulación PIL para resetear los controladores y la atención de tareas a las condiciones iniciales.

- `PIL_CLBK_TERMINATE_SIMULATION`: llamada al final de cualquier simulación PIL.

- `PIL_CLBK_STOP_TIMERS`: hace su función al principio de la interrupción de control cuando se usa PIL para parar todos los temporizadores y contadores asociados a las tareas de control.

- `PIL_CLBK_START_SIMULATION`: se llama antes de reanudar las tareas de control cuando se está en modo PIL. Se consigue reiniciar los contadores y los temporizadores asociados a las tareas de control.

4.4.2. Configuración del entorno

Para configurar PIL es necesario llamar a una serie de funciones que se encargan de la configuración y la inicialización.

- `PIL_init()`: debe llamarse al principio de cualquier otra llamada de ejecución vista en la sección anterior.

- `PIL_setLinkParams()`: establece los parámetros obligatorios para la comunicación. Necesita un puntero al primer elemento del campo GUID y otro a *CommCallback*. Utiliza la definición de un tipo (`typedef void(*PIL_CommCallbackPtr_t)()`) para la comunicación, que permite que PIL llame a la aplicación utilizando un puntero para recivir los datos entrantes y solicitar la transmisión de mensajes.

- `PIL_setCtrlCallback()`: registra las llamadas del control para las simulaciones de PIL. Para su correcta ejecución es necesario utilizar la definicion del tipo `typedef void(*PIL_CtrlCallbackPtr_t)(PIL_CtrlCallbackReq_t)`. Este tipo de puntero es utilizado para que PIL llame a la aplicación y pueda comunicarse correctamente.

Existen otro tipo de configuraciones adicionales que, por ejemplo, permiten la transmisión serie múltiple, pero que no se utilizan en el proyecto y por lo tanto, no van a ser detalladas. Puede consultarse [4] para más información sobre estas configuraciones.

Para hacer que la configuración y las evaluaciones estén disponibles para PLECS es necesario utilizar la macro PIL_CONST_DEF, a través de la cual se pueden definir el nombre de usuario, información sobre el tiempo de compilación, nombre del proyecto y de la tarjeta. Como requisito mínimo, `Guid[]` debe estar definido y si se usa la comunicación serie entre el hardware y PLECS, también es necesario especificar la velocidad de comunicación a través de la definición del `BaudRate`.

Capítulo 5

Implementación

Desde que se plantea una situación de partida inicial hasta que se consigue alcanzar una solución particular del problema planteado, existen una serie de logros intermedios que permiten verificar los pasos del proceso.

Para asegurar de que se avanza de manera consistente, se construye la solución verificando que cada modelado y cada código se comporta de la manera esperada. En este capítulo se muestra de manera detallada cada una de estas verificacones, y se profundiza en la explicación de la solución que se pretende conseguir: implementar el código en el hardware real y comunicarlo con PLECS para una ejecución en pseudo-tiempo real.

5.1. Solución mediante simulación de bloques

En primer lugar, se prescinde del hardware, ya que hay que comprobar que el algoritmo del MPP funciona. Una vez obtenido el diseño de los lazos de control y comprobar que el diseño de la planta de partida funciona correctamente, se implementa el algoritmo de conductancia incremental para la búsqueda del punto de máxima potencia.

De forma general, lo que se busca implementar en el modelo de la simulación es lo que aparece representado en la figura 5.1. En dicha figura puede observarse que para el control del sistema, son necesarios tres bloques: un bloque PWM encargado de generar los disparos, un bloque MPPT para la implementación de la búsqueda del punto de máxima potencia y otro bloque de control, que en realidad se divide en un bloque para la regulación de la tensión en el condensador de salida de los paneles y otro para el control en corriente (external control loop e inner control loop). Para la generación de los disparos y la activación de los bloques de control y MPPT es conveniente contar con una señal de habilitación en el sistema, de forma que sólo comiencen a ejecutarse cuando dicha señal esté habilitada. Igualmente, la dinámica del MPPT es más lenta que la de los lazos de control, por lo que deben establecerse correctamente los períodos de ejecución de los bloques. Para el desarrollo del proyecto, se ha establecido que la ejecución del MPPT se lleve a cabo cada 50 ms mientras que el control se ejecutará cada 100 μs. El modelo de paneles y el convertidor se ejecuta en continuo.

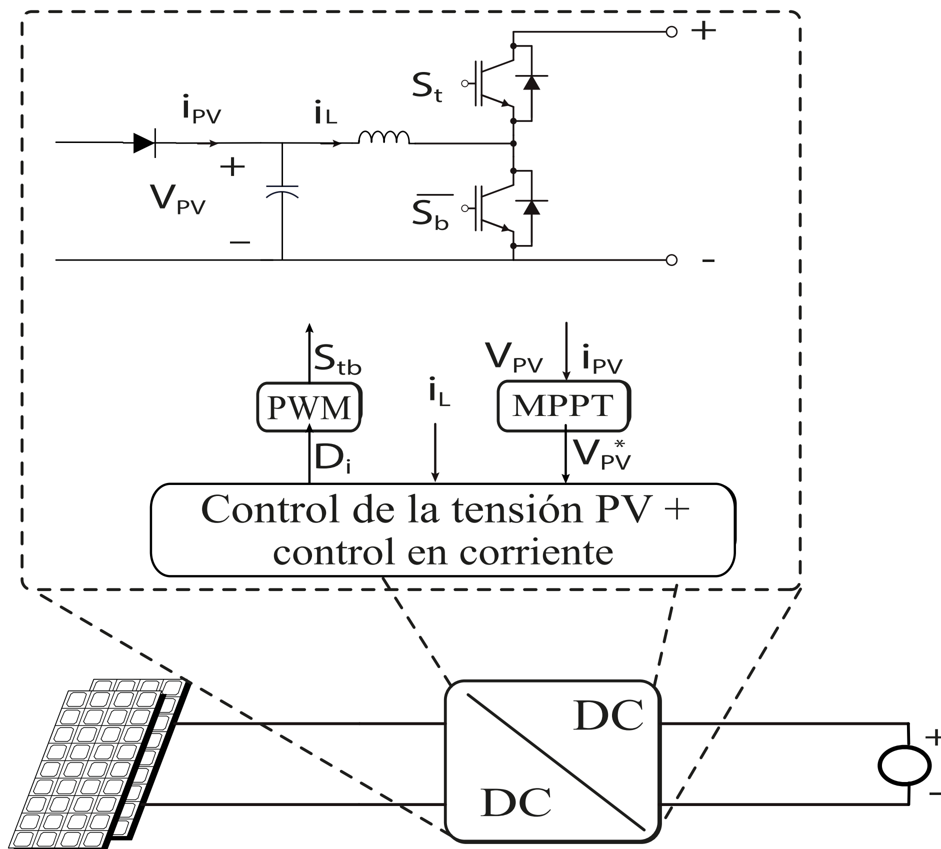


Figura 5.1: Implementación en bloques.

Para poder implementar el bloque PWM es necesario una señal triangular así como algún bloque que permita retrasar una señal para la generación de los disparos con tiempos muertos. Por su parte, el bloque MPPT debe contar con alguna herramienta que permita la implantación del algoritmo de conductancia incremental en C. Además, para la implementación del control es necesario disponer de bloques PID que permitan implantar los controladores formulados en las ecuaciones (3.12) y (3.17). En caso de no disponer directamente de bloques PID, pueden usarse ganancias e integradores para su implementación.

En el resto de este apartado, se propone PLECS como una solución particular a la herramienta de simulación.En la figura 5.2 puede observarse la disposición inicial. En primer lugar, el sistema cuenta con una señal de habilitación (*enable*) que se encarga de habilitar el control y los disparos de los transistores. Los bloques inferiores son los del control del convertidor elevador a la izquierda y, a la derecha, la generación de los disparos. A continuación, se explica de manera más detallada cada uno de los bloques implementados en PLECS; puede comprobarse la similitud que presenta con otros simuladores como Simulink.

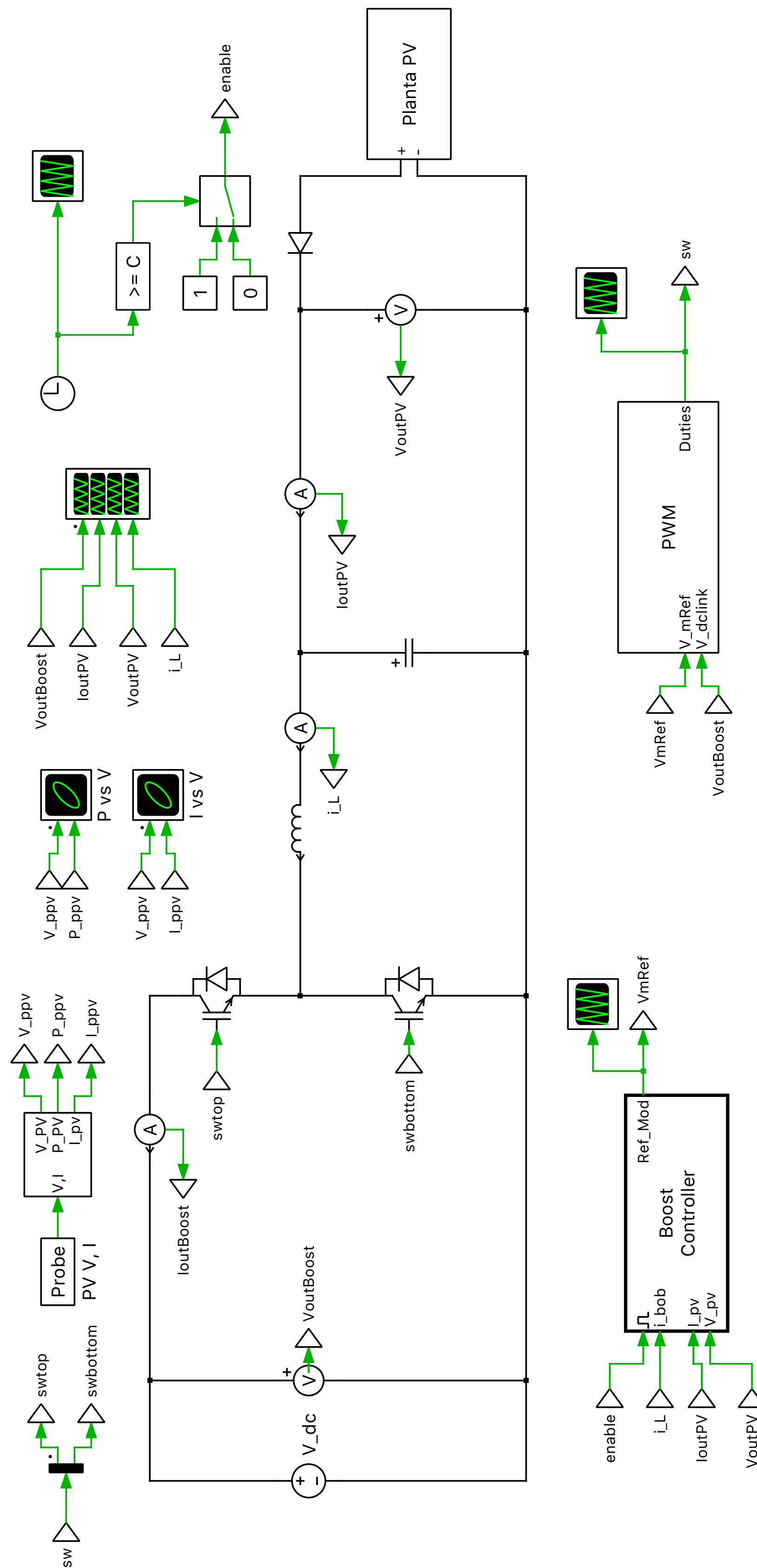


Figura 5.2: Implementación en bloques.

5.1.1. Bloque Boost Controller

En este bloque se pueden distinguir a su vez tres instancias diferentes. El bloque MPPT, donde se encuentra el código para la persecución del punto de máxima potencia, obtiene a su salida la tensión de referencia de los paneles. Esta tensión de referencia entra en el bucle externo de control para obtener la potencia de referencia que necesita el bucle interno y así poder obtener una referencia modulada para la generación del pwm.

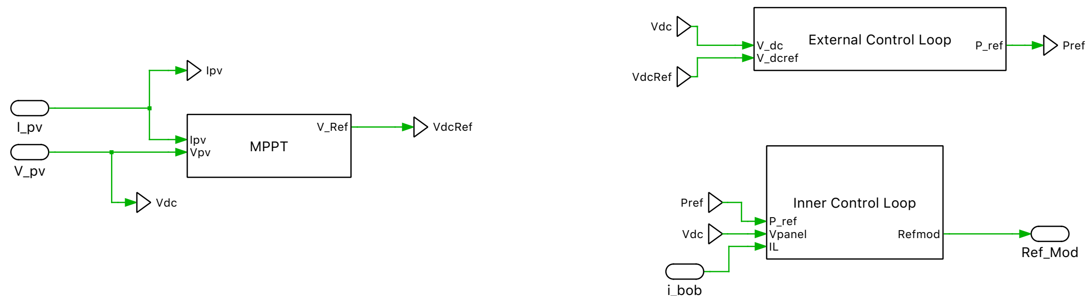


Figura 5.3: Bloque Boost Controller.

Bloque MPPT

Este bloque sólo está compuesto por un C-Script (herramienta proporcionada por la librería de PLECS que permite la implementación de algoritmos en C) donde se implementa el método de conductancia incremental. Este bloque de la librería de PLECS contiene, entre otros, apartados para la actualización de variables en cada iteración, funciones de inicio, declaraciones de variables y función de salida del bloque. De manera general, se implementa el algoritmo de conductancia incremental de la figura 3.5. Puede consultarse [3] para más información sobre la configuración de dichos bloques y el anexo A para ver la implementación del algoritmo.

Bloque External Control Loop

Este bloque se implementa simplemente teniendo presente las ecuaciones presentadas en la sección 3.4 del capítulo 3. Así, implementación de la figura 5.4, corresponde con la ecuación (3.12).

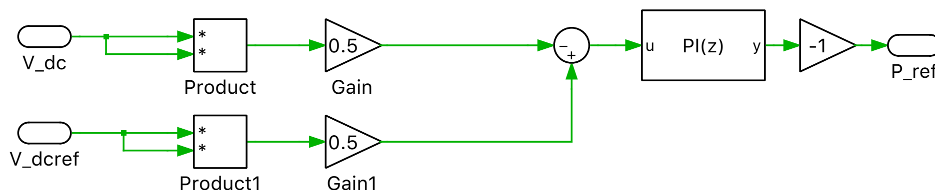


Figura 5.4: External Control Loop.

Inner Control Loop

Teniendo en cuenta las ecuaciones desarrolladas en el apartado 3.5, puede desarrollarse la implementación mostrada en 5.5. Las entradas de potencia de referencia y tensión de paneles se utilizan para poder obtener una corriente de referencia que, posteriormente, se compara con la que circula por la bobina para poder obtener una referencia de tensión modulada.

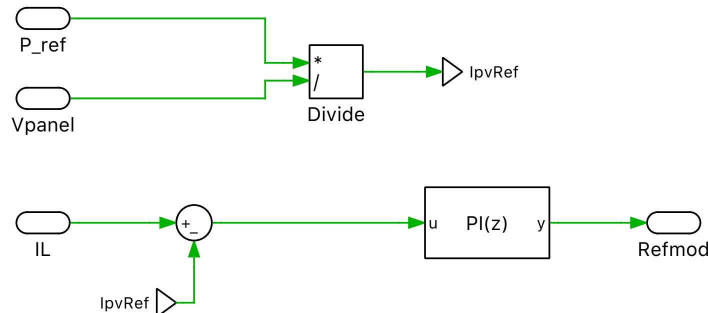


Figura 5.5: Inner Control Loop.

5.1.2. Bloque PWM

En este bloque se implementa la generación de los disparo de la rama de los transistores con tiempos muertos, ya que el modelo de los dispositivos no es ideal y por tanto existe un momento durante la conmutación en el que ambos transistores están en estado de conducción, provocando un cortocircuito en el condensador de salida (o la fuente de alimentación de tensión en este caso).

En la figura 5.6 aparece el modelo para la generación del PWM con tiempos muertos. Primero, la tensión modulada se divide entre la tensión del condensador de salida para generar un cierto ciclo de trabajo (cuyo valor comprenderá entre 0 y 1) que se compara con una señal triangular configurada a 10 kHz. También hace uso de la señal *enable* para la habilitación de los disparos.

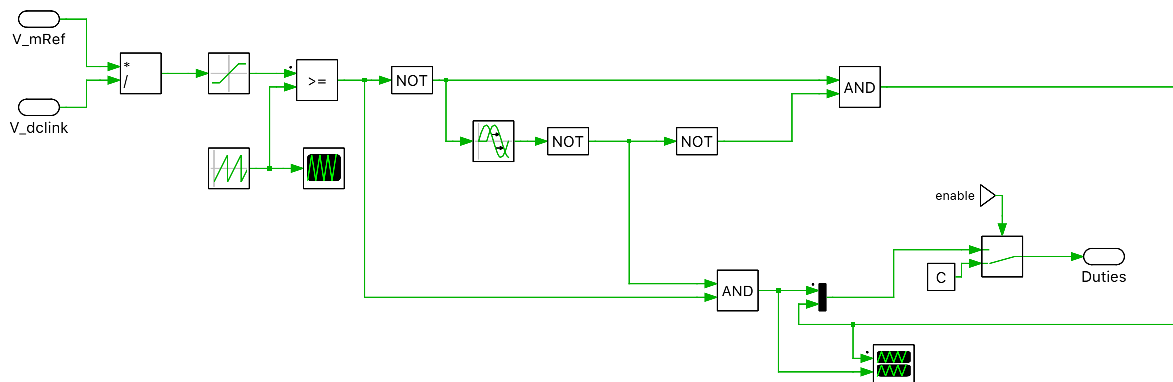


Figura 5.6: Generación de PWM con tiempos muertos.

Para generar los tiempos muertos hay que retrasar el disparo en uno de los interruptores utilizando puertas lógicas (el modelo no tiene en cuenta los retrasos de las puertas lógicas) y un bloque de retraso que simularía un circuito RC. En la figura 5.7 puede verse representado los tiempos muertos generados por el circuito de la figura 5.6 con un retraso de 2 μs.

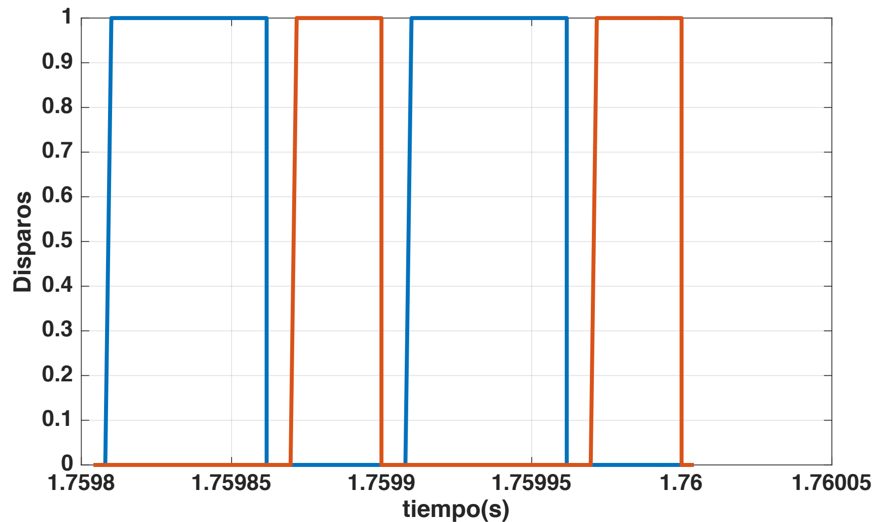


Figura 5.7: Disparos de los transistores con tiempos muertos.

5.1.3. Resultados de simulación

A continuación se muestran los resultados de la simulación con este primer acercamiento a la implementación de la solución. En la figura 5.8 se muestran la tensión de salida, la corriente y tensión de salida de la planta de paneles, así como la corriente de la bobina para una simulación de 2 s con una señal de habilitación en 0,5 s. El valor de la señal de habilitación no tiene un valor arbitrario, si no que está elegido de tal manera que exista una precarga en el condensador de salida de la planta de paneles. Los parámetros de la simulación puede consultarse en la tabla 5.1.

Se puede comprobar en la figura 5.8 que la planta alcanza los 431,6 V y 12,1 kW, que corresponde con el punto de máxima potencia para una irradiación nominal de la planta.

El rizado de corriente en la bobina en régimen permanente puede visualizarse en la última gráfica de la figura 5.8. En esta simulación dicho rizado es de 2,44 A, comprobándose el funcionamiento en continuo del convertidor. También puede observarse en las figuras 5.9 (referencia en rojo) y 5.10 (referencia en verde) que existe, respectivamente, seguimiento de tensión y de corriente.

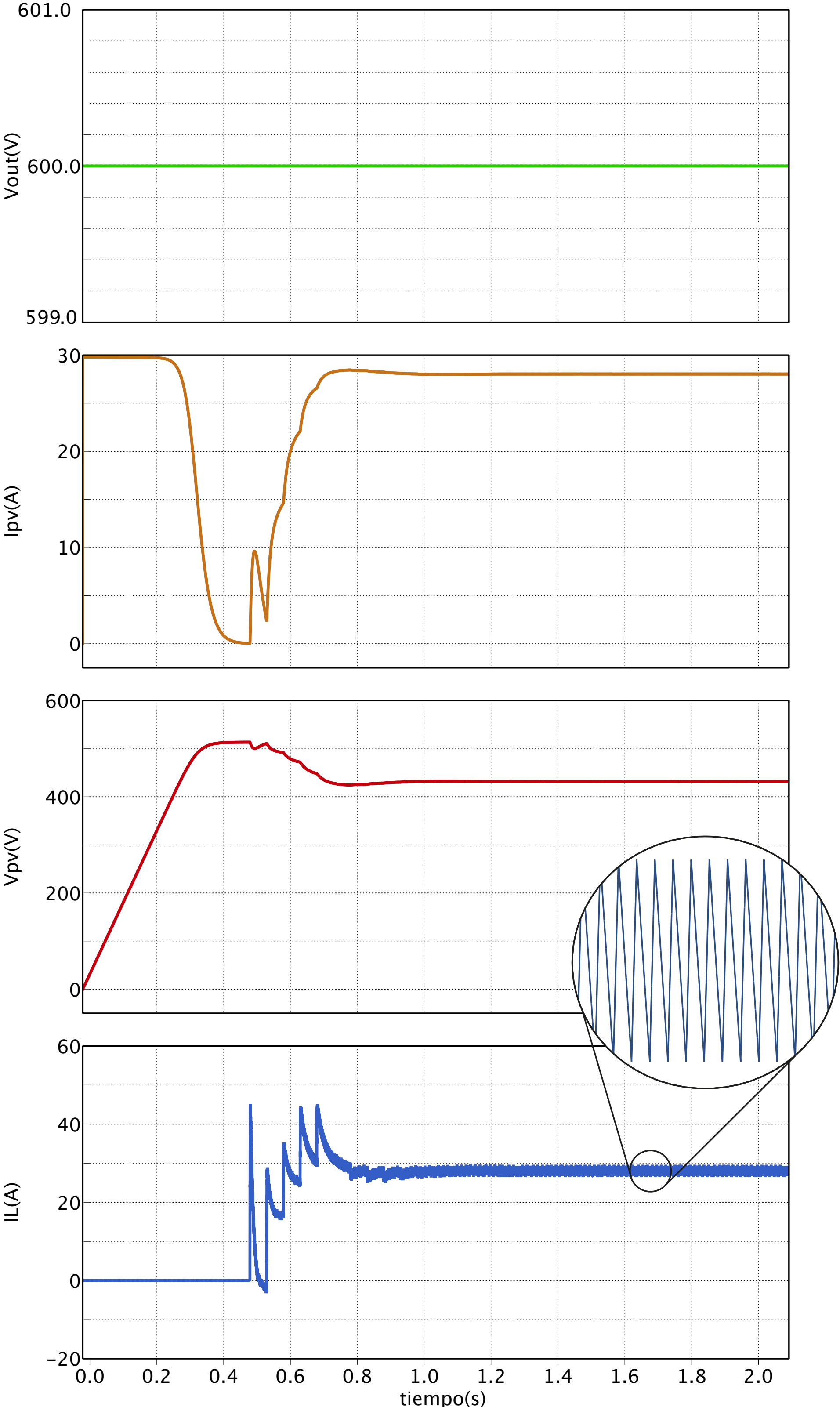


Figura 5.8: Resultado tras la simulación de bloques.

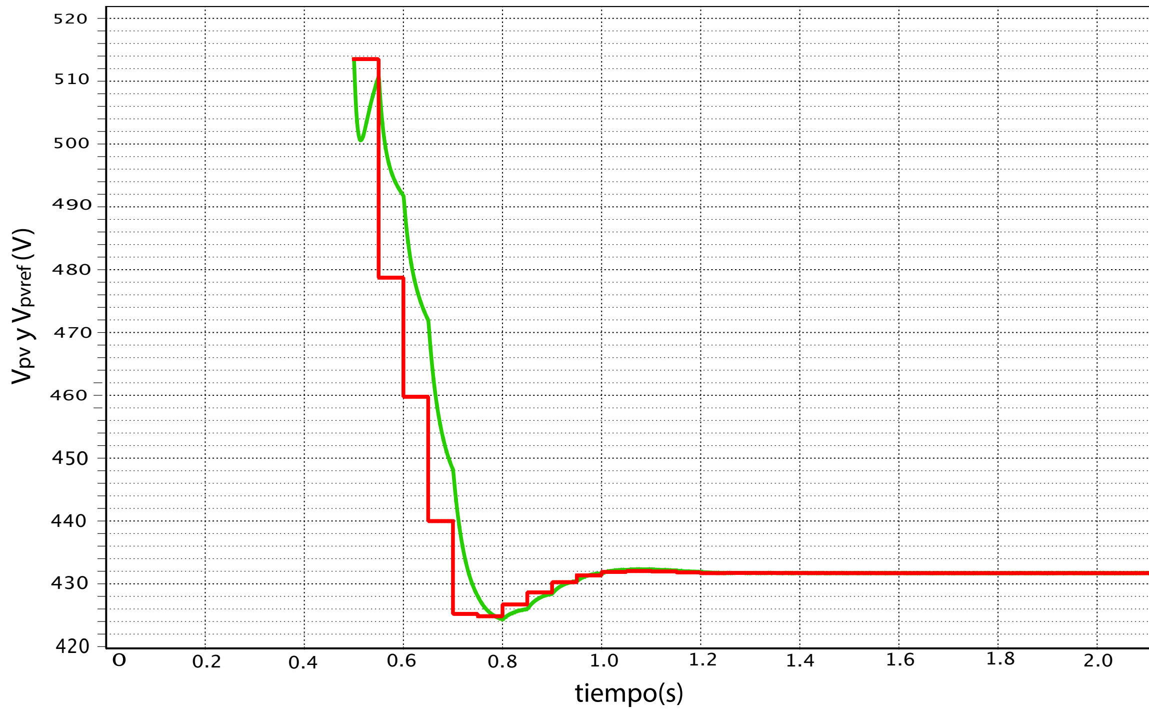


Figura 5.9: Seguimiento de tensión con bloques.

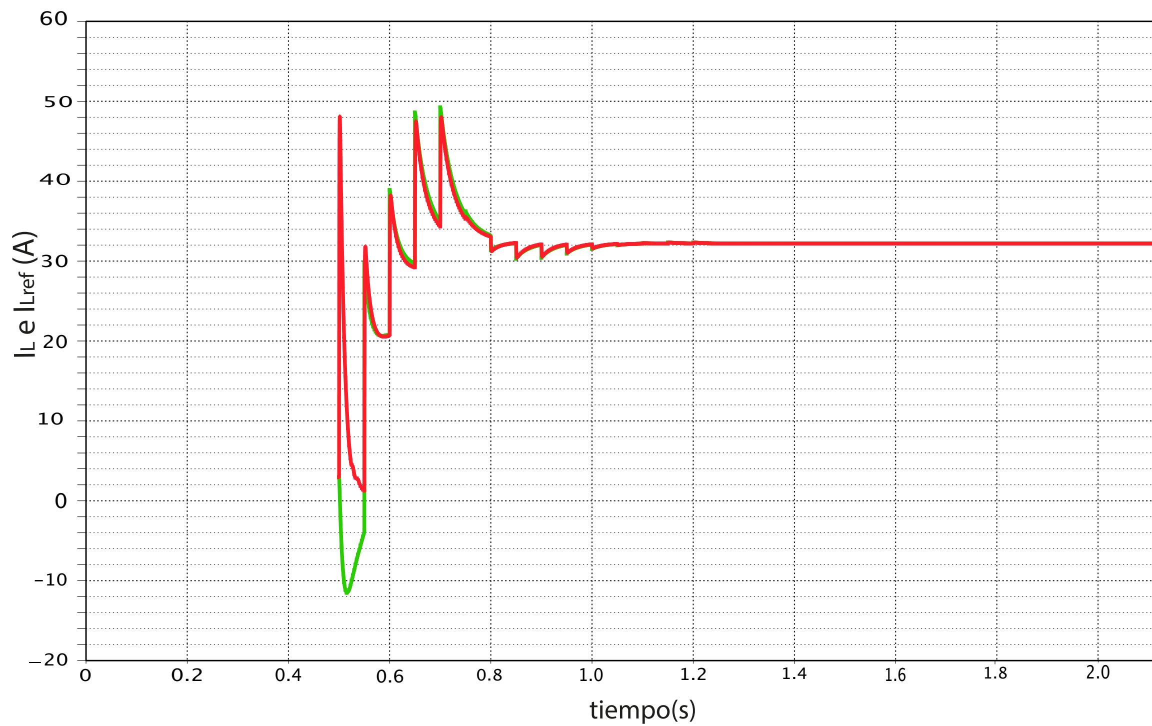


Figura 5.10: Seguimiento en corriente con bloques.

5.2. Implementación con controladores en C

La única diferencia que existe con respecto a la implementación de la sección 5.1 es la implementación de los controladores. Tras comprobar que el algoritmo MPPT funciona correctamente, el siguiente paso es implementar los controladores en un bloque que también permita la implementación de los PI en C.

Para la aproximación de los controladores se utiliza la aproximación rectangular hacia delante, conocida también como Euler I, representada en la figura 5.1. Entonces, la aproximación de la derivada del error y la integral quedan, respectivamente como la expresiones (5.1) y (5.2).

$$\frac{\mathrm{d}e(t)}{\mathrm{d}t} = \frac{e_k - e_{k-1}}{T} \tag{5.1}$$

$$\int_0^t e(\tau)\mathrm{d}\tau = \sum_{i=0}^{k-1} e(i)T = T\sum_{i=0}^{k-1} e_i \tag{5.2}$$

Por tanto, discretizando la ecuación de un PI continuo se llega a la ecuación del PI discreto con la aproximación adoptada mostrada en la ecuación (5.3)

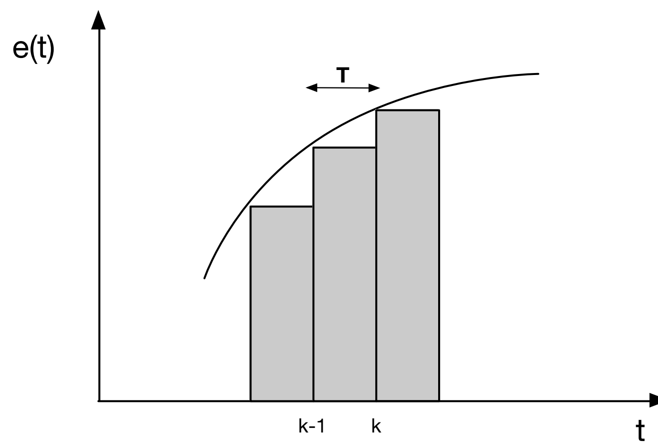


Figura 5.11: Aproximación Euler I.

$$u_k = u_{k-1} + k_p e_k - k_p e_{k-1} + K_i T_s e_{k-1} \tag{5.3}$$

Para el caso particular de la implementación en PLECS, se han utilizado los bloques C-Script para la implementación de los lazos de control (external control loop e inner control loop). Ambos controladores siguen la aproximación Euler I de la ecuación (5.3). Puede consultarse el anexo A para consultar la configuración de dichos bloques en PLECS.

5.2.1. Resultados de simulación

Como es de esperar, el comportamiento de las simulaciones es idéntico. Puede comprobarse en las figuras 5.12 y 5.13 que existe seguimiento en corriente y tensión.

En la figura 5.14 se muestran los resultados de la simulación para la señal de habilitación activada a los 0,5 s.

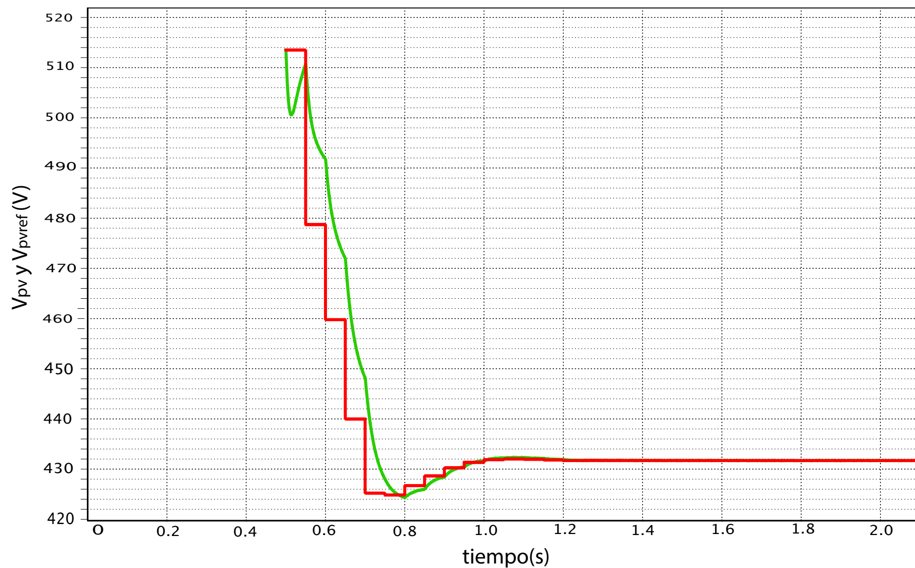


Figura 5.12: Seguimiento de tensión con controladores en C.

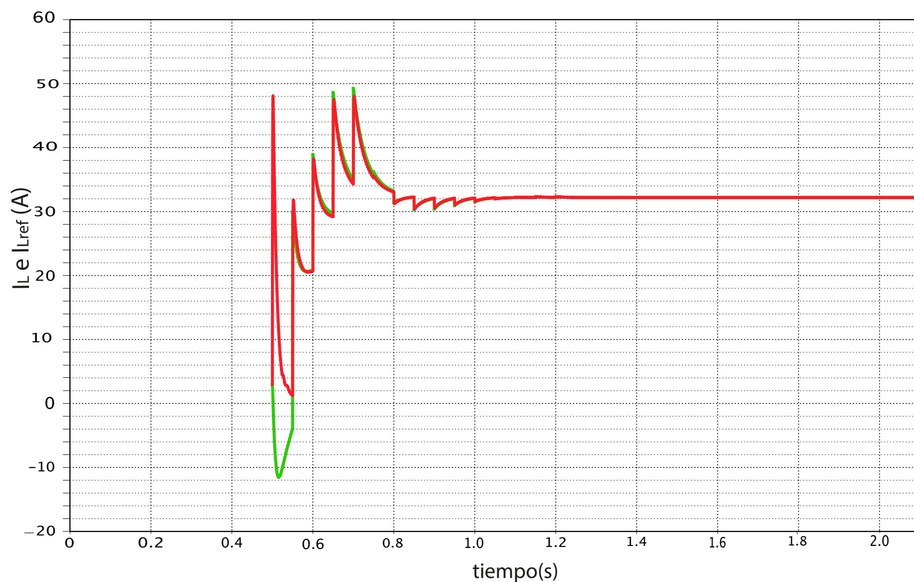


Figura 5.13: Seguimiento en corriente con controladores en C.

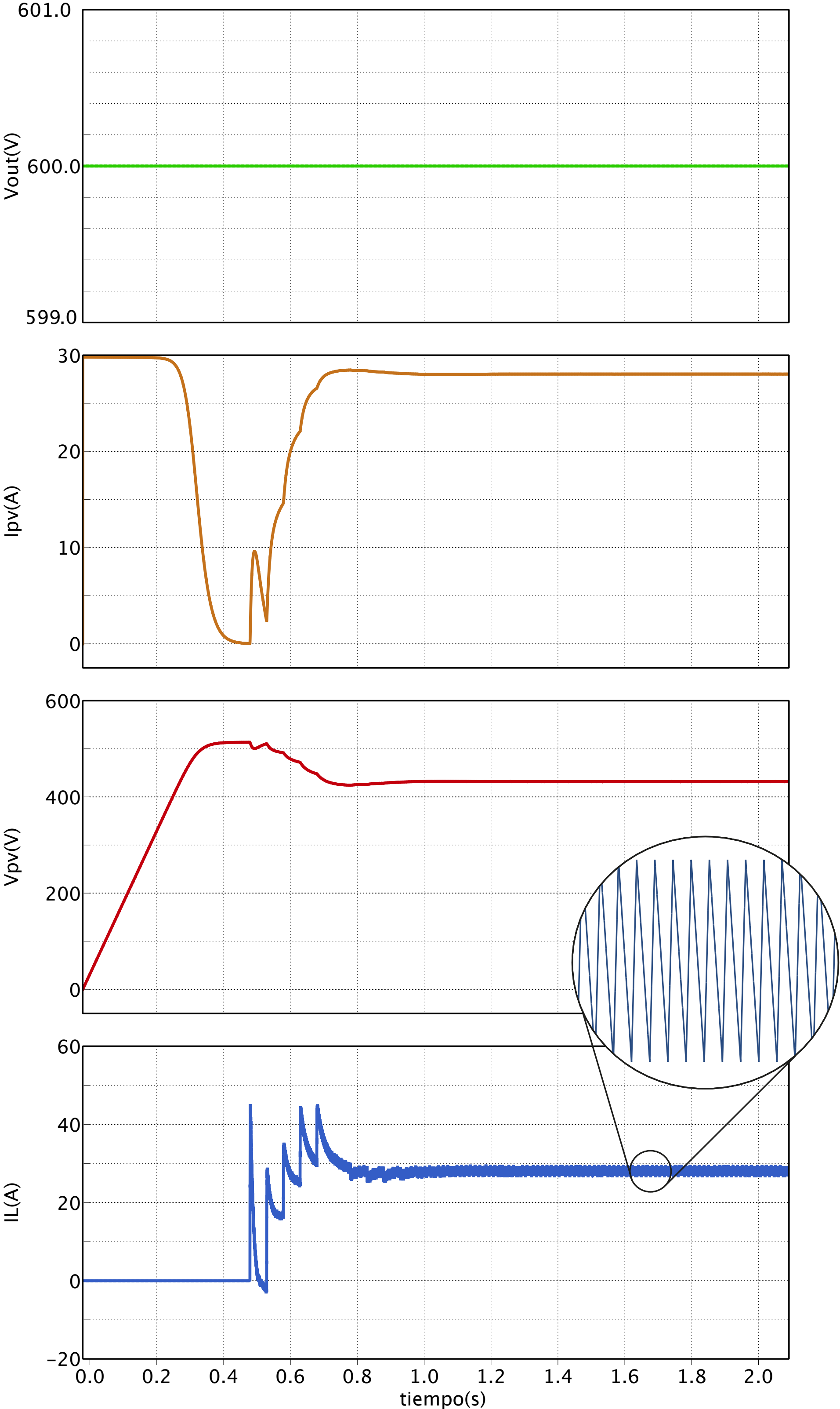


Figura 5.14: Resultados de simulación con C-Scripts.

Parámetro	Valor
K_p lazo externo	1
K_p lazo interno	10
K_i lazo externo	10
K_i lazo interno	500
f_{sw} (disparos)	10 kHz
$T_{control}$ (disparos)	100 μs
T_{MPPT}	50 ms
Max step simulación	1 μs

Tabla 5.1: Parámetros de la simulación.

5.3. Implementación en PIL

Una vez se comprueba que el algoritmo de búsqueda del punto de máxima potencia y que los controladores funcionan correctamente, se procede a utilizar la herramienta PIL. Por un lado debe modificarse el modelo de la simulación (ya que hay que incluir el nuevo bloque de PIL y el modelo de los periféricos del DSP) y por otro lado se debe configurar *Code Composer Studio* para poder trabajar con el DSP, tal y como se muestra en el esquemático de la figura 5.15.

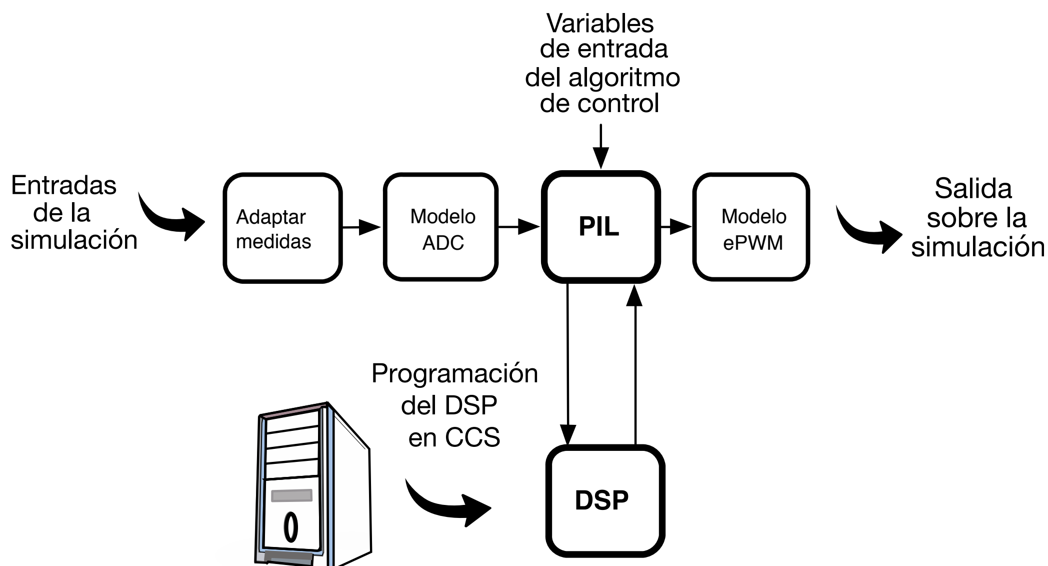


Figura 5.15: Implementación en bloques.

Por un lado, se tiene el entorno de simulación que permite obtener las variables a controlar por el DSP. Estas variables deben pasar por un modelo del convertidor

analógico digital por lo que, para hacer la simulación lo más realista posible, estas señales deben introducirse en un bloque de etapa de adaptación que simule el comportamiento de los sensores de medición. Además de contar con modelos suficientemente precisos de los periféricos del controlador, también es necesario contar con una herramienta PIL que permita comunicarse con el microcontrolador y así poder ejecutar el código de control en el entorno de simulación (pseudo-tiempo real).

En lo que resta de sección, se va a explicar cómo se lleva a cabo la ejecución con PLECS y el DSP elegido. Para ello es necesario hacer una breve explicación de cómo configurar *Code Composer Studio* y cómo se utilizan los modelos de los periféricos proporcionados por la librería de PLECS.

5.3.1. Configuración en Code Composer Studio

Para empezar a realizar nuestro proyecto en *Code Composer Studio*, es necesario primero configurarlo para que pueda reconocer la herramienta *Pil Prep Tool* proporcionada por *Plexim*. De esta manera, es necesario crear una variable de entorno que contenga la ruta absoluta de dicha herramienta.

Por otra parte, también se deben definir las propiedades del proyecto. Hay que asegurarse que lo que en lo que se conoce como *Linker Command File* se ha seleccionado *F28335.cmd* y que la librería que se va a utilizar es la *rts2800_fpu32.lib*, que permite trabajar de manera optima con flotantes. Aunque se puede usar cualquier versión de *CCS*, se comprobó que no es válida cualquier versión del compilador. Las nuevas versiones incorporan un compilador que es capaz de de generar el fichero de salida con la configuración del hardware pero PIL no es capaz de reconocerlo. La versión del compilador que se ha usado para la realización del proyecto es la *C2000_6.2.9*.

Cualquier proyecto que se quiera realizar es necesario incluir el header file de pil *pil.h*, que contiene las cabeceras de todas las funciones necesarias para la comunicación. Este fichero está incluido en la carpeta de frameworks. El contenido puede verse en los anexos A, al final de la documentación.

5.3.2. Montaje en PLECS

Ahora el modelo del sistema es un modelo más realista de una implementación física desde el punto de vista del control. En la figura 5.17 pueden verse los diferentes bloques para la comunicación con el hardware.

La utilización de modelos precisos de los periféricos es un requisito fundamental para las simulaciones con PIL. Estos modelos deben ser configurables exactamente igual que el propio hardware. Los modelos simplificados de los periféricos son útiles a gran nivel de abstracción, pero se pierden detalles importantes de la temporización y la cuantificación por ejemplo, en el convertidor analógico-digital y la generación de pulsos.

5.3.2.1. Adaptación de medidas

El ADC del microcontrolador trabaja en un rango de tensiones comprendido entre 0 y 3,3 V, por lo que en la realidad es necesario adaptar todas las medidas de corriente y tensión del convertidor a este rango de tensiones a través de, por ejemplo, amplificadores operacionales o sensores de efecto Hall. Para simular esta etapa en nuestro circuito de potencia se introduce el bloque de adaptación de medidas de la figura 5.16 que convierte las señales muestreadas al rango de trabajo del ADC utilizando ganancias. El valor de las ganancias utilizadas puede consultarse en la tabla 5.2.

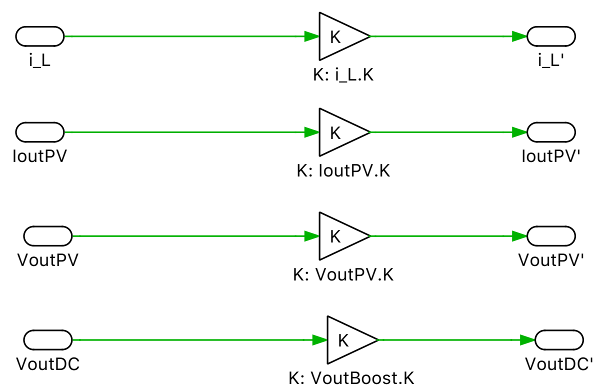


Figura 5.16: Adaptación de medidas.

De esta forma el valor leído por el ADC será:

$$Valor_ADC = \frac{3{,}3}{Valor_max}\text{Valor_entrada} \tag{5.4}$$

donde $Valor_ADC$ será el valor de entrada al ADC, $Valor_max$ es el valor máximo de tensión o corriente que se desea medir y Valor_entrada el valor de entrada previo a la adaptación.

Ganancia	Valor
$K : i_L$	3,3/125
$K : i_{outPV}$	3,3/40
$K : V_{outPV}$	3,3/520
$K : V_{outBoost}$	3,3/600

Tabla 5.2: Ganancias de la etapa de adaptación.

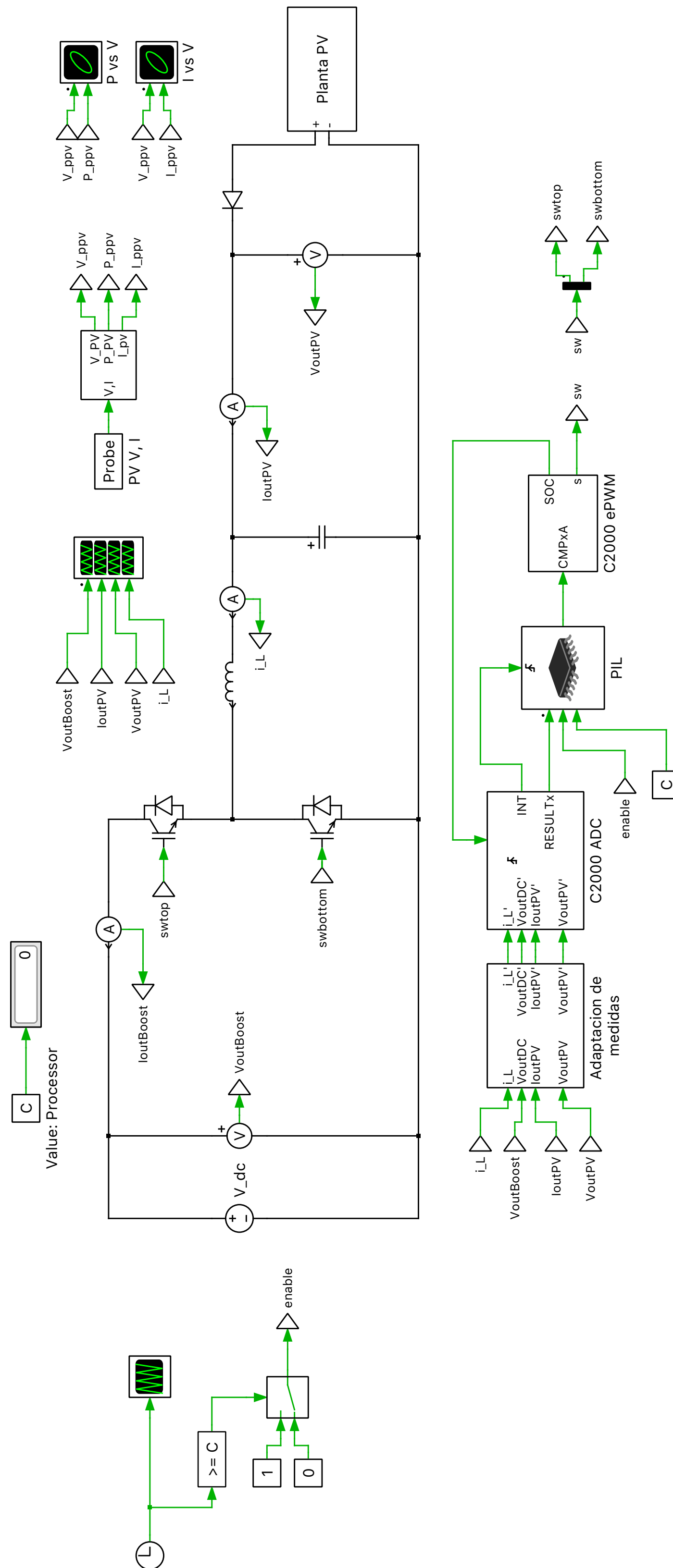


Figura 5.17: Implementación con PIL.

5.3.2.2. C2000 ADC

Para emular el comportamiendo del ADC se utiliza el bloque tipo 2 proporcionado en la librería de PLECS y que se muestra en la figura 5.18. Como puede observarse, en el modelo aparecen los propios registros del ADC del *DSP-TMS320F28335*. La programación se basa en el valor de los registros del propio hardware. La configuración que se lleva a cabo en el proyecto se explicará más adelante. En cualquier caso puede comprobarse que el modelo del convertidor analógico-digital proporciona los registros habituales para configurarlos:

- *ePWM_SOCx*: entrada para disparar las conversiones del ADC.
- *MAX_CONVx*: número de conversiones por secuencias.
- *RST_SEQx*: puerto para resetear las secuencias.
- *ADCINA/B*: entradas para las medidas.
- *ADCRESULTx*: salidas con el valor de las conversiones.
- *ADC INT_SEQx*: salidas para disparar la interrupción al final de cada secuencia.

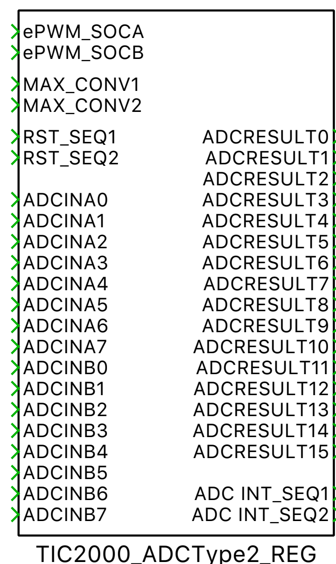


Figura 5.18: Bloque ADC.

Existen otros tipos de modelos proporcionados por PLECS que, en principio, son igualmente válidos. Puede consultarse la bibliografia [4] para conocer más detalles de los modelos que se proporcionan.

5.3.2.3. C2000 ePWM

El bloque ePWM cuenta, a su vez, con otros subsistemas, como puede observarse en la figura 5.19. En primer lugar cuenta con un bloque de configuración donde se configuran los comparadores y los registros de *AQCTLx* y *AQCSFRC* (registros de control para establecer la acción del PWM).

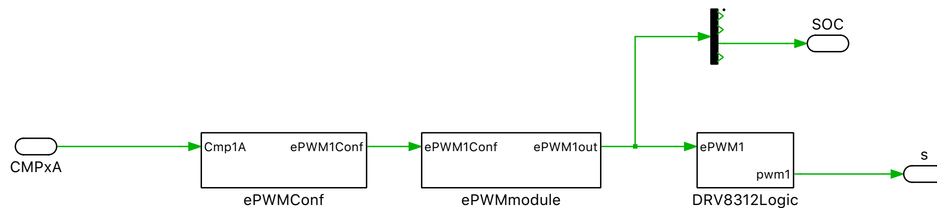


Figura 5.19: Bloque C2000 ePWM.

El segundo subsistema está formado por la propia configuración del bloque ePWM proporicionado en la librería. Se ha escogido el tipo 1, ya que permite la configuración basada en los registros de la tarjeta, lo que hace posible reflejar exactamente la configuración que se ha aplicado en el hardware. Este modelo cuenta con los módulos de *Time-Base*, *Counter-Compare*, *Action-Qualifier*, *Dead-Band* y *Event-Trigger*. La salida de este bloque proporciona los disparos de los transistores a través de las salidas *PWMxA* y *PWMxB* así como los eventos para el inicio de la conversión del ADC (*EPWMSOCA* y *EPWMSOCB*).

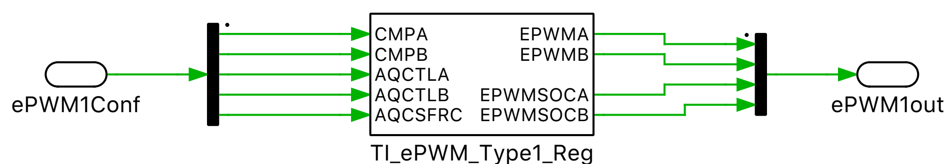


Figura 5.20: ePWM tipo 1.

El subsistema DRIVER cuyo modelado se muestra en la figura 5.21 trata de emular el comportamiento de un driver que, dada la configuración de un PWM, ofrece el complementario para el disparo de un transistor de la misma rama con cierto retraso. El funcionamiento de este driver corresponde con el comportamiento de la figura 5.6, con la diferencia de que el PWM lo proporciona el DSP.

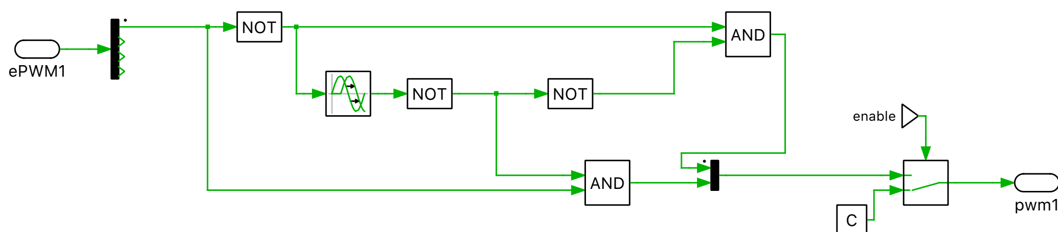


Figura 5.21: Comportamiento del driver.

5.3.3. Configuración del DSP-TMS320F28335

En este apartado, vamos a explicar cómo se configura el DSP para la realización del proyecto. Para entrar en el detalle se puede consultar el datasheet que ofrece *Texas Instruments* y que aprecen en la bibliografía.

Flash

Para mejorar el rendimiento se ha activado el modo Pipe line. Igualmente se definen los estados mínimos de espera en los registros necesarios tal y como indica el datasheet, ya que esto depende de la velocidad establecida de la CPU.

Es importante que las secciones *DSP/BIOS.trcdata* se ejecuten desde la RAM, por lo que hay que copiar manualmente esta sección desde la dirección donde se carga hasta la de ejecución. Igualmente ocurre con la sección *DSP/BIOS.hwi_vec*, que contiene la tabla de interrupciones. Todo este proceso se realiza a través de punteros que apuntan al origen y al destino y debe de ejecutarse antes de inicializar los registros de control de la memoria Flash.

SCI

Para la interfaz de comunicación serie se eligen los pines 28 y 29 de la tarjeta como pin de recepción y transmisión respectivamente. Cuenta con un bit de parada y un tamaño de caracteres de 8 bits.

Como se comentó en el capítulo 4.4.2, para la comunicación serie es necesario definir la velocidad en baudios de la transmisión. Se ha configurado para una velocidad de $115200L$.

PWM

Para la generación de pulsos, se configura el PWM1 para una frecuencia de 10 kHz. La triangular se establece en modo simétrico de tal forma que cuando el comparador sea mayor que el contador a la salida está activa a nivel alto y, por el contrario, cuando es menor que el contador la salida está activa a nivel bajo.

La generación de los pulsos para el inicio de la conversión del ADC se realiza a cada inicio del contador. La forma de los pulsos generados puede verse en la figura 5.22.

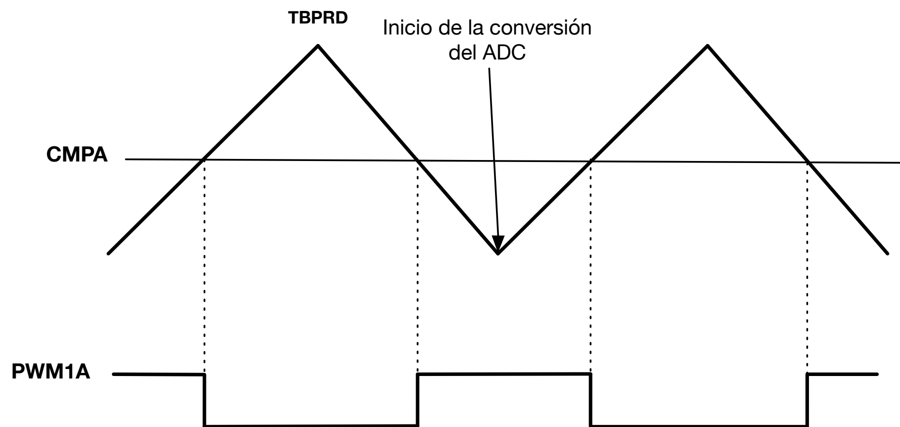


Figura 5.22: Configuración ePWM.

ADC

La conversión analógica digital se realiza en una sola secuencia (secuencia 1) con cuatro conversiones en la misma. Al final de cada conversión, cada 100 μs se realiza una interrupción que ejecutará el algoritmo de control. La única parte del código que se ejecuta en pseudo-tiempo real es la interrupción del ADC.

5.3.4. Diagramas de flujo del código en C

En esta subsección se representan los diagramas de flujo del código. En el diagrama 5.23 se representa el algoritmo para la función principal (main), que conlleva todo la comunicación y funcionamiento del bloque PIL, además de la propia inicialización de la tarjeta, periféricos y rutinas de control.

Por otro lado el diagrama de flujos 5.24 explica la secuencia que se lleva a cabo en la interrupción del ADC. La explicación es fácil de comprender. A cada interrupción del ADC se calculan los valores reales del ADC (ya que provienen de la etapa de adaptación). La ejecución del algoritmo de búsqueda de máxima potencia y del control sólo se lleva a cabo si el sistema está habilitado: el MPPT se ejecuta cada 50 ms a través de una variable tipo contador que lleva la cuenta del ciclo del MPPT y el control cada 100 μs. En caso de que se habilite el sistema, se fuerza a que se ejecute el algoritmo MPPT por primera vez. En cualquier caso, se facilita el código completo en los Anexos al final del documento.

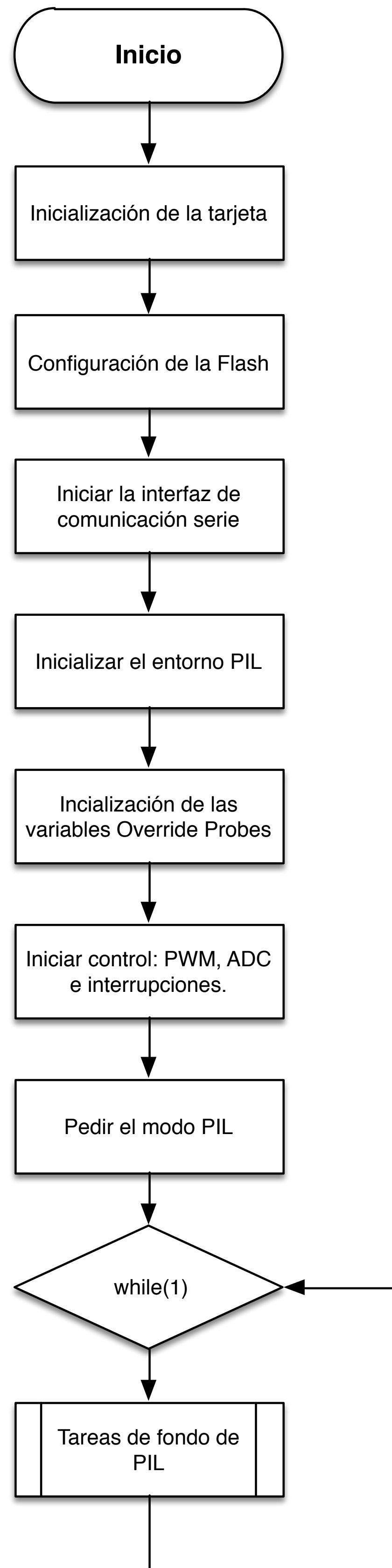


Figura 5.23: Diagrama de flujo de la función principal.

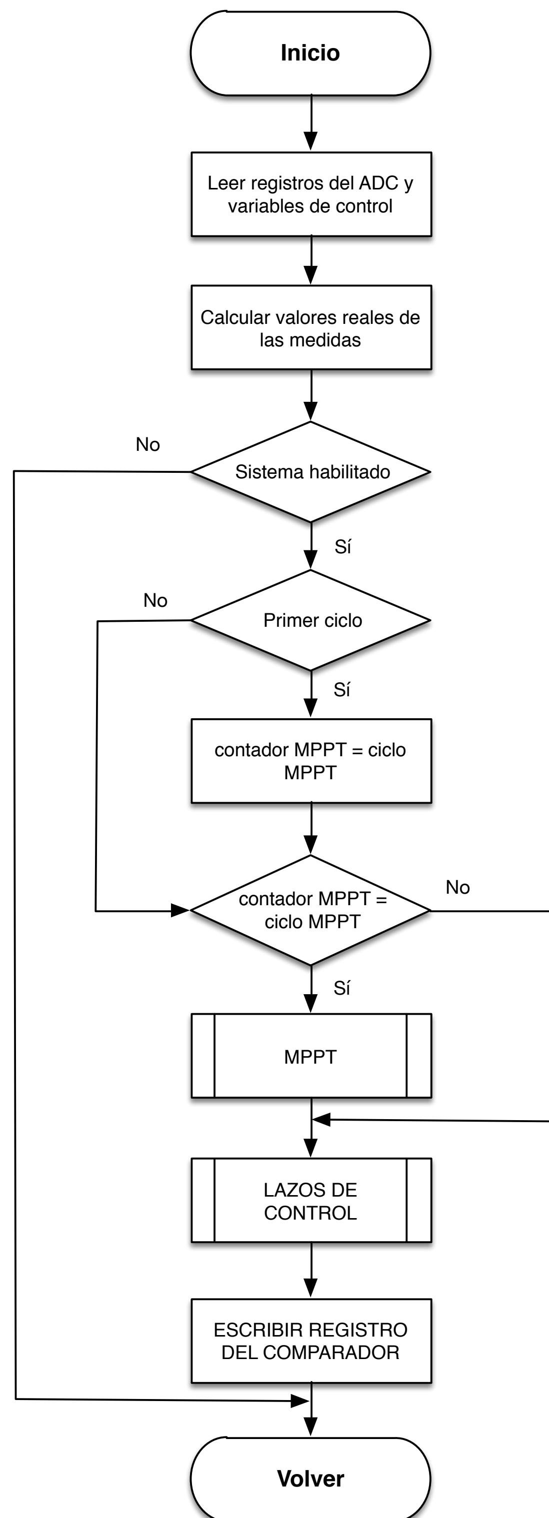


Figura 5.24: Diagrama de flujo de la interrupción del ADC.

5.3.5. Resultados de simulación

En la figura 5.27, se representan las corrientes y tensiones representativas del sistema para una irradiancia nominal y para una señal de habilitación en 0,5 s.

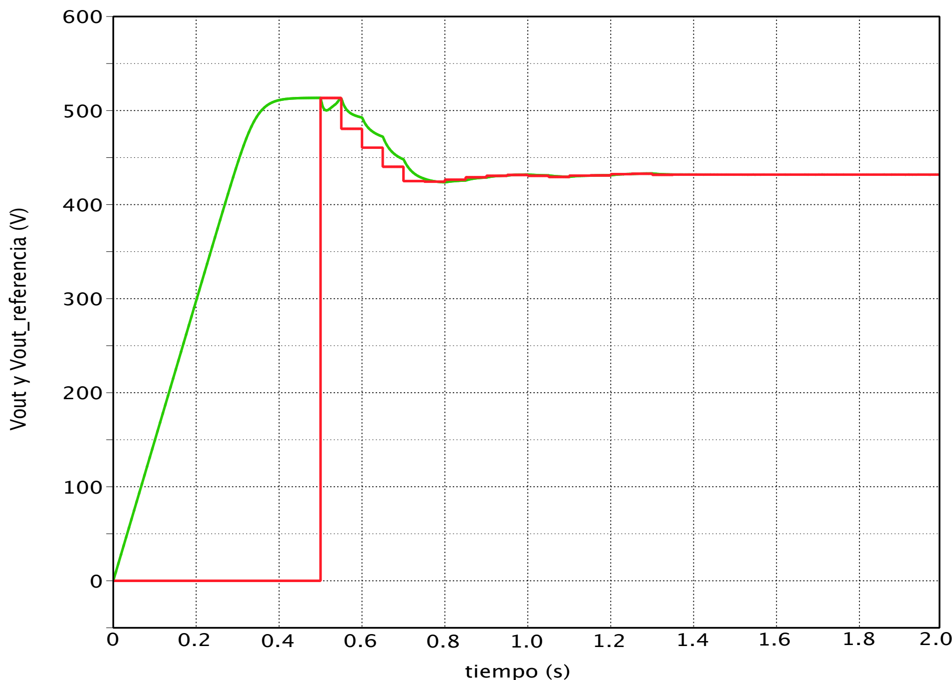


Figura 5.25: Seguimiento de tensión en PIL.

En la figura 5.25 puede observarse el seguimiento en tensión (en rojo se muestra la tensión de referencia y en verde el comportamiento de la tensión de la planta). En esta gráfica se hace notar el efecto de la existencia un cero de fase no mínima en el sistema debido a la dinámica de carga y descarga del condensador (obsevar justo en el instante de habilitación del sistema, $t = 0{,}5$ s). El valor de la capacidad del condensador de salida de la planta de paneles debe ser lo suficientemente grande para conseguir un rizado mínimo pero también lo suficientemente pequeño para que durante, el intervalo de control y muestreo de la tensíon, pueda ofrecer cambios de tensión.

También puede comprobarse en la figura 5.26 y el seguimiento en corriente. El rizado de la corriente de la bobina que se muestra en el zoom de la figura 5.27 es de 2,44 en régimen permanente.

Una vez comprobado que el algoritmo MPPT y el control funciona en PIL de manera similar que a las simulaciones previas, puede realizarse cualquier tipo de prueba en la simulación, como por ejemplo, comprobar la consistencia del algoritmo de conductancia incremental. En la figura 5.28 se representan los resultados para un cambio de irradiancia en $t = 1{,}2$ s.

Como puede comprobarse, PIL es una herramienta consistente que permite comprobar si el código de control funciona correctamente. Se trata de una función, junto

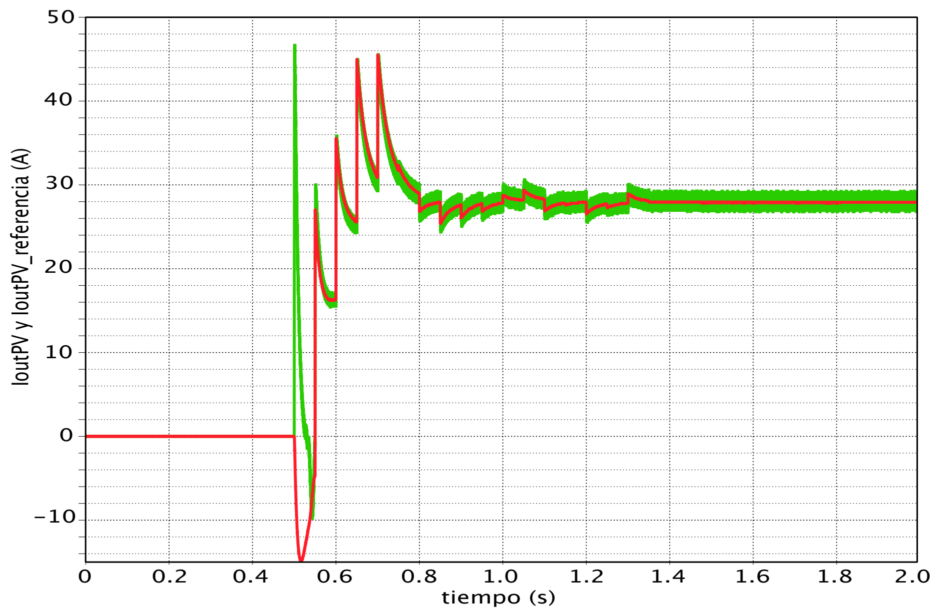


Figura 5.26: Seguimiento de corriente en PIL.

con PLECS, de gran utilidad que en principio permite el control de cualquier sistema de potencia. Con el fin de completar el sistema para una posible implementación real, también se implementa la máquina de estados que se muestra en la figura 5.29. Esta máquina de estados cuenta con tres señales de entrada: una señal para la habilitación del sistema ($\mathtt{r} = 1$), otra de parada ($\mathtt{r} = 0$) y otra de reset del sistema. Los estados posibles de la máquina de estados son:

- Inicio: este es el estado de reposo del sistema. El sistema se mantendrá en este estado siempre y cuando la señal de habilitación tenga un valor 0.
- Precarga: para que el sistema funcione correctamente, es necesario realizar una precarga en el condensador de salida de los paneles. Se permanece en dicho estado siempre y cuando la precarga no se haya completado (el nivel de la tensión del condensador sea inferior a la tensión de circuito abierto de la planta).
- Run: se trata del estado normal de ejecución donde la planta opera correctamente. Se activan los disparos de los transistores y es el único estado donde se habilita la ejecución del MPPT y del control.
- Error: se trata de un estado accesible por todos los estados anteriores. En caso de error, dicho estado se encarga de deshabilitar los disparos y sólo se podra salir de este estado a través de la habilitación del reset, lo que permite llevar al sistema a la situación inicial.

Para la implementación de la máquina de estados, el algoritmo del control en la interrupción del ADC debe modificarse como se muestra en el diagrama de flujo de la figura 5.30.

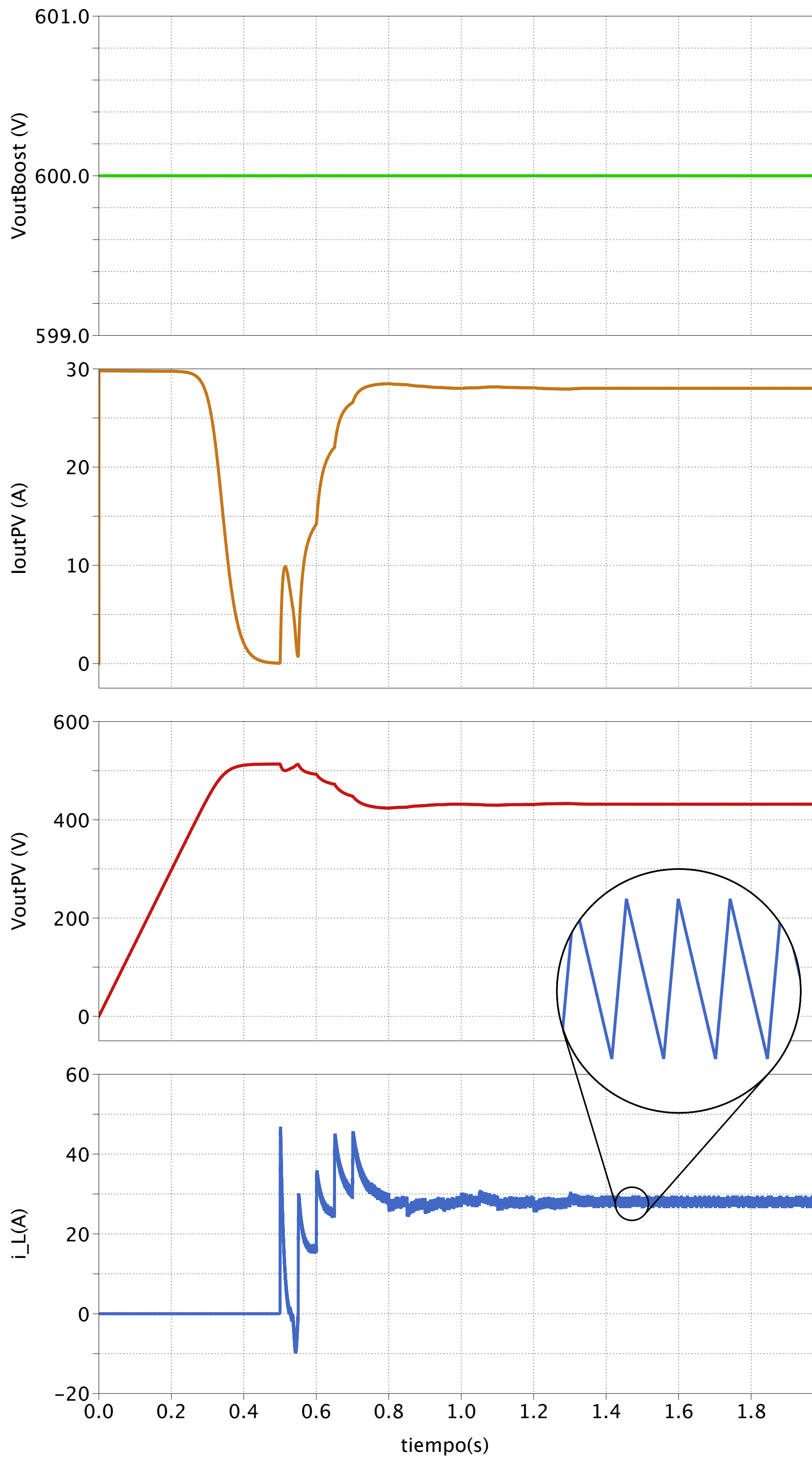


Figura 5.27: Resultado tras la simulación con PIL.

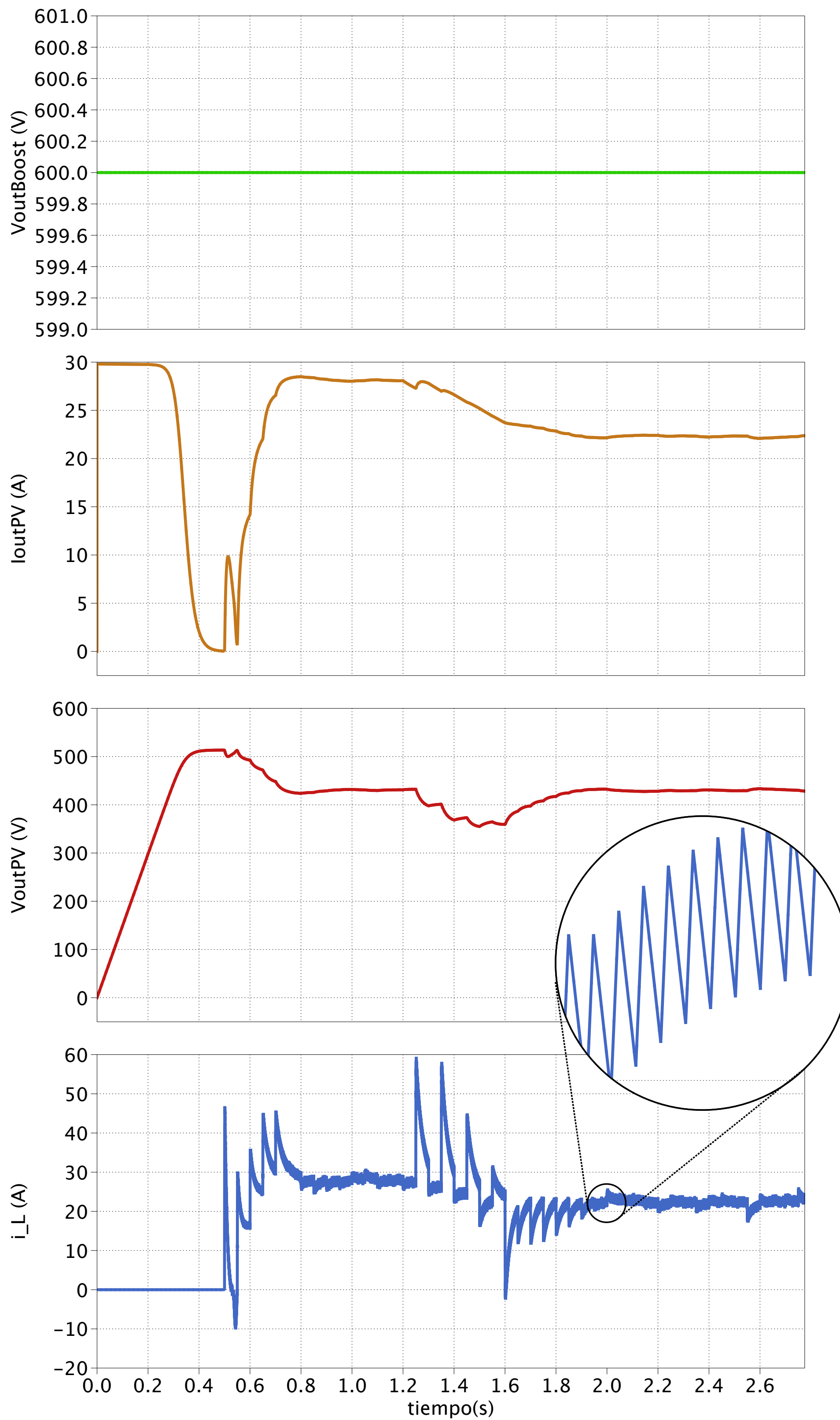


Figura 5.28: Resultado con PIL ante un cambio en irradiancia.

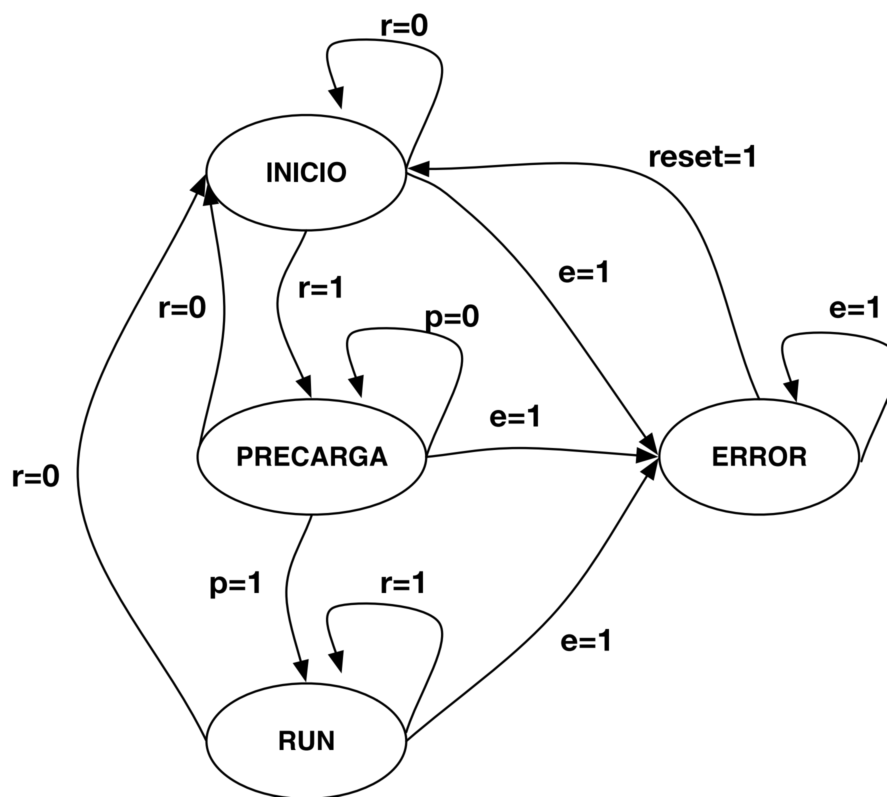


Figura 5.29: Máquina de estados.

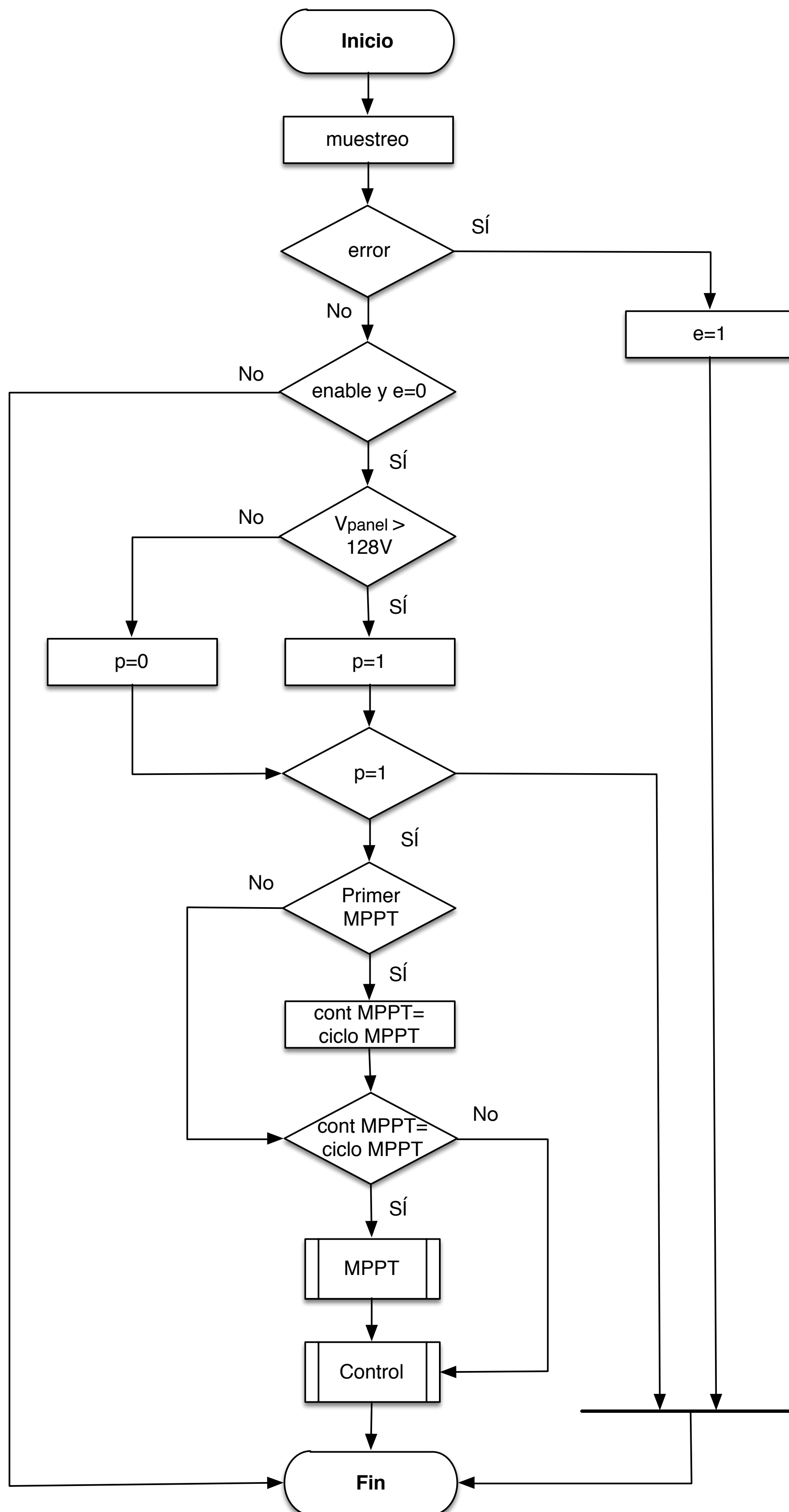


Figura 5.30: Diagrama de flujos de la interrupción con máquina de estados.

Capítulo 6

Conclusiones y trabajos futuros

Este proyecto ha supuesto un avance en la simulación de sistemas de potencia mediante plataformas embebidas. Con modelos suficientemente buenos de los dispositivos es posible reproducir el comportamiento de un sistema con cierta exactitud y un mínimo error. Gracias a simulaciones en tiempo o pseudo-tiempo real es posible minimizar errores del algoritmo, minimizando pérdidas económicas y materiales y, por lo tanto, acotando los posibles errores de la puesta en marcha de multitud de proyectos, ya que tienen en cuenta efectos de modelado y comportamiento que en simulaciones convencionales se desprecian.

Aunque el algoritmo de conductancia incremental funciona ante cambios de irradiancia y situaciones de sombreado parcial, puede mejorarse el algoritmo MPPT, añadiendo una mejora para la búsqueda de un óptimo global que mejore la estabilidad del sistema y su rapidez de convergencia, tanto para casos de sombreado parcial y como para cambios bruscos de irradiancia.

Para completar el sistema, puede ampliarse el modelo de simulación sustituyendo la fuente de tensión de salida del convertidor elevador por un condensador, que estaría regulado por el control del inversor, cuyo modelo también habría que incorporar. Resulta interesante simular el sistema completo para la integración de la planta en la red eléctrica.

El atractivo más interesante, a pesar de todas las ampliaciones posibles en el ámbito de la simulación, es comprobar el modo normal de funcionamiento de PIL. Es decir, probar el código de control en un entorno real donde el valor de las variables de entradas sean las del propio circuito de potencia. Para ello, se propone utilizar el emulador de paneles fotovoltaicos 14360A de *Agilent Technologies*, capaz de simular unas curvas características dadas. El control de los paneles para la programación de las curvas puede realizarse de manera remota a través de un servidor.

Anexos

Anexo A

Scripts

MPPT

```
1 #include <math.h>
2
3 //////VARIABLES DE ENTRADA Y LAS DE ACTUALIZACION/////////////////
4 #define V_IN InputSignal(0,0)
5 #define I_IN InputSignal(0,1)
6
7 #define Vref  DiscState(0)
8 #define I_OLD DiscState(1)
9 #define V_OLD DiscState(2)
10
11 ////////////////DECLARACION DE VARIABLES/////////////////////
12
13 double changeInVRef;
14 double deltaV, deltaI;
15 typedef char bool_t;
16 static bool_t InitializationFlag = TRUE;
17
18
19
20 typedef struct
21 {
22     double deltaVoltage;            //umbral de voltaje
23     double deltaCurrent;            //umbral de corriente
24     double deltaPower;              //umbral de potencia
25     double alpha;                   //Factor de aceleracion
26     double beta;                    //Factor de aceleracion
27     double MaxAbsoluteValue_dPdV;   //Maximo valor de la derivada
28 } INCModeFactors_t;
29
30 static INCModeFactors_t INCModeFactors;
31
32 static double INCModeFactors.deltaVoltage =  1e-4;
33 static double INCModeFactors.deltaCurrent =  1e-3;
34 static double INCModeFactors.deltaPower =  1e-3;
```

```
35
36 static double INCModeFactors.alpha = 0.4;
37 static double INCModeFactors.beta = 0.1;
38
39 static double INCModeFactors.MaxAbsoluteValue_dPdV = 80;
40
41 ///////CODIGO DE LA FUNCION DE SALIDA DEL BLOQUE CSCRIPT////////
42
43 if(InitializationFlag == TRUE)
44 {
45     V_OLD = V_IN;   // Actualizar voltaje
46     I_OLD = I_IN;   // Actualizar corriente
47
48     InitializationFlag = FALSE;
49
50 }
51
52 /*Calculamos los incrementos*/
53
54 deltaV = V_IN - V_OLD;
55 deltaI = I_IN - I_OLD;
56
57
58 changeInVRef = IncrementalConductance(V_IN,I_IN,deltaV,deltaI,INCModeFactors.
       MaxAbsoluteValue_dPdV);
59
60
61 Vref = V_IN + changeInVRef;
62
63 static double DetermineINCDPDV (double aV_In, double aI_In, double aDeltaV, double aDeltaI,
       double aMaxdPdVMagnitude)
64 {
65     double dPdV;
66
67     dPdV= aV_In*(aI_In/aV_In + aDeltaI/aDeltaV);   //Calcular dP/dV
68
69     if (dPdV > aMaxdPdVMagnitude)    // Saturacion superior
70     {
71         dPdV=aMaxdPdVMagnitude;
72     }
73     if (dPdV<-aMaxdPdVMagnitude)     //Saturacion inferior
74     {
75         dPdV=-aMaxdPdVMagnitude;
76     }
77
78     return dPdV;
79 }
80
81
82 ////////////FUNCIONES PARA EL CALCULO DEL MPPT////////////////////
83
84 static double DetermineINCChangeInReferenceVoltage(double aV_In,double aI_In,double aDeltaV,
       double aDeltaI,double aDPDV)
```

```
85 {
86     double changeInVRef;
87 
88     if( (aDeltaV < INCModeFactors.deltaVoltage) && (aDeltaV > −INCModeFactors.deltaVoltage)
      )
89     {
90         if( (aDeltaI < INCModeFactors.deltaCurrent) && (aDeltaI > −INCModeFactors.deltaCurrent
      ) )
91         {
92             changeInVRef = 0;          //Estamos en el MPPT y no hacemos cambios en la referencia
93         }
94         else
95         {
96             changeInVRef = INCModeFactors.beta ∗ aDeltaI;
97         }
98     }
99     else
100    {
101        double tempPowerVariable = aI_In/aV_In + aDeltaI/aDeltaV;    //Calculamos DI/DV+I/V
102 
103        if( (tempPowerVariable < INCModeFactors.deltaPower) && (tempPowerVariable > −
      INCModeFactors.deltaPower) )
104        {
105            changeInVRef = 0;
106        }
107        else
108        {
109            changeInVRef = INCModeFactors.alpha ∗ aDPDV;
110        }
111    }
112 
113    return changeInVRef;       //Devolvemos el cambio en la referencia
114 }
115 
116 
117 
118 static double IncrementalConductance(double aV_In,double aI_In,double aDeltaV,double aDeltaI,
      double aMaxdPdVMagnitude)
119 {
120    double changeInVRef;
121    double dPdV;
122 
123    dPdV = DetermineINCDPDV(aV_In,aI_In,aDeltaV,aDeltaI, aMaxdPdVMagnitude);
124 
125    changeInVRef = DetermineINCChangeInReferenceVoltage(aV_In,aI_In,aDeltaV,aDeltaI,dPdV);
126 
127    return changeInVRef;
128 }
```

External Control Loop

```
1 #define V_dc InputSignal(0,0)
```

```
2 #define V_Ref InputSignal(0,1)
3
4 //DEFINICION DE VARIABLES ACTUALIZABLES///
5 #define P_ref DiscState(0)
6 #define error DiscState(1)
7 #define errora DiscState(2)
8
9 //DEFINICION CONSTANTES//
10 static int kp=1;
11 static int ki=10;
12 static double Ts=100e-6;
13
14 //ALGORITMO DE SALIDA (cada 100us)//
15 error=0.5*((V_Ref*V_Ref) - (V_dc*V_dc));
16
17 P_ref= (P_ref + kp*error - kp*errora + ki*Ts*errora);
18
19 errora=error;
20
21 OutputSignal(0,0) = -P_ref;
22 ///////////////////////
```

Inner Control Loop

```
1 #include <math.h>
2
3 //DEFINICION ENTRADAS Y SALIDAS//
4 #define P_ref InputSignal(0,0)
5 #define V_pv InputSignal(0,1)
6 #define i_L InputSignal(0,2)
7
8 //DEFINICION DE VARIABLES ACTUALIZABLES//
9 #define Ref_mod DiscState(0)
10 #define error DiscState(1)
11 #define errora DiscState(2)
12
13 //DEFINICION DE CONSTANTES Y VARIABLES//
14 static int kp=10;
15 static int ki=500;
16 static double Ts=100e-6;
17
18 float IpvRef;
19
20 //CODIGO DE LA FUNCION DE SALIDA (cada 100us)//
21
22 IpvRef=P_ref/V_pv;
23
24 error=(i_L - IpvRef);
25
26 Ref_mod= (Ref_mod + kp*error - kp*errora + ki*Ts*errora);
27
28 errora=error;
```

```
29
30 OutputSignal(0,0) = Ref_mod;
31
32 //////////////////////////////////////////////
```

Anexo B

Ficheros de cabecera (Header Files)

pil.h

```
#ifndef PIL_H_
#define PIL_H_

#ifdef __cplusplus
extern "C" {
#endif

#include <stdint.h>
#include <stdbool.h>

#ifndef PIL_CLA_CODE
#include <stdlib.h>
#endif

//@{
/** Additional types and constant definitions */
#ifndef NULL
#define NULL 0
#endif
//@}

/**
 * Macro for declaring a read probe.
 */
#ifndef PIL_PREP_TOOL
#define PIL_READ_PROBE(type, name, qloc, ref, unit) type name
#endif

/**
 * Macro for declaring an override probe.
 */
#ifndef PIL_PREP_TOOL
// full list of arguments used by PIL prep tool: type, name, qloc, ref, unit
#ifdef PREFIX_PROBES
```

```
35 #define PIL_OVERRIDE_PROBE(type, name, ...)\
36       type name
37 #else
38 #define PIL_OVERRIDE_PROBE(type, name, ...)\
39       type name;\
40       type name ## _probeV;\
41       int16_t name ## _probeF
42 #endif
43 #endif
44
45 /**
46  * Macro for declaring a calibration.
47  */
48 #ifndef PIL_PREP_TOOL
49 #define PIL_CALIBRATION(type, name, qloc, ref, unit, min, max, dval) type name
50 #endif
51
52 /**
53  * Macro for defining a PIL configuration constant.
54  */
55 #define PIL_CONFIG_DEF(type, name, value)\
56  const type PIL_C_ ## name __attribute__ ((used))  = value
57
58 /**
59  * Macro for setting the value of an override probe.
60  * Important: Do not set override probes using direct assignments ("=").
61  * Always use this macro instead.
62  */
63 #ifdef PREFIX_PROBES
64 #define SET_OPROBE(probe, value) \
65  do{\
66     probe = value; \
67     if(PIL_Probe_F_ ## probe == 1){\
68      probe = PIL_Probe_V_ ## probe; \
69     }\
70  } while (0)
71 #else
72 #define SET_OPROBE(probe, value) \
73   do{\
74      probe = value; \
75      if(probe ## _probeF == 1){\
76         probe = probe ## _probeV; \
77      }\
78   } while (0)
79 #endif
80 /**
81  * Control-callback identifiers.
82  * These constants identify individual actions which the PIL framework
83  * requests from the application.
84  */
85 typedef enum
86 {
87   PIL_CLBK_LEAVE_NORMAL_OPERATION_REQ, //!< Request system to enter PIL mode (i.e.
```

```
        disable actuators)
88    PIL_CLBK_ENTER_NORMAL_OPERATION_REQ, //!< Release system from PIL mode (i.e.
        allow re−enabling of actuators)
89    PIL_CLBK_PREINIT_SIMULATION,        //!< Called (with interrupts disabled) to allow PIL pre
        −initialization
90    PIL_CLBK_INITIALIZE_SIMULATION,      //!< Called at start of PIL simulation
91    PIL_CLBK_TERMINATE_SIMULATION,       //!< Called at end of PIL simulation
92    PIL_CLBK_STOP_TIMERS,                //!< Called to request freezing of timers
93    PIL_CLBK_START_TIMERS                //!< Called to request release of frozen timers
94 } PIL_CtrlCallbackReq_t;
95
96 /**
97  * Initializes override probes.
98  * This function is autogenerated by the PIL prep tool and has to be called once
99  * during the initialization of the code.
100 */
101 extern void PilInitOverrideProbes(void);
102
103 /**
104  * Initializes calibrations to their default values.
105  * This function is auto−generated by the PIL prep tool and has to be called during
106  * the initialization of the code. It is also recommended that the function be called
107  * in the PIL_CLBK_TERMINATE_SIMULATION callback to ensure calibrations are restored
108  * to their original values after a PIL simulations.
109  */
110 extern void PilInitCalibrations(void);
111
112 /**
113  * Control callback type definition.
114  * The PIL framework calls the application using this callback pointer.
115  */
116 typedef void(*PIL_CtrlCallbackPtr_t)(PIL_CtrlCallbackReq_t);
117
118 /**
119  * Communication callback type definition.
120  * The PIL framework calls the application using this callback pointer
121  * to receive incoming data and to request the transmission of outgoing messages.
122  */
123 typedef void(*PIL_CommCallbackPtr_t)();
124
125 /**
126  * Initializes the PIL framework.
127  * This function must be called before any other framework functions.
128  */
129 extern void PIL_init(void);
130
131 /**
132  * Sets the mandatory link parameters
133  *
134  * @param  aGuid      Pointer to first element of GUID field
135  * @param aCommPtr    Pointer to CommCallback
136  * @return None
137  */
```

```
138 extern void PIL_setLinkParams(const unsigned char* aGuid, PIL_CommCallbackPtr_t aCommPtr);
139
140 /**
141  * Sets desired node address
142  *
143  * @param aNode Desired Station/Node Address
144  * @return None
145  */
146 extern void PIL_setNodeAddress(unsigned char aNode);
147
148 /**
149  * Sets pointer to the PIL control callback.
150  *
151  * @param  aCtrlPtr   Pointer to CtrlCallback
152  * @return None
153  */
154 extern void PIL_setCtrlCallback(PIL_CtrlCallbackPtr_t aCtrlPtr);
155
156 /**
157  * Sets pointer to the PIL background communication callback.
158  *
159  * @param  aBackgroundCommPtr   Pointer to BackgroundCommCallback
160  * @return None
161  */
162 extern void PIL_setBackgroundCommCallback(PIL_CommCallbackPtr_t aBackgroundCommPtr);
163
164 /**
165  * Indicates to the framework that the system is ready for a PIL simulation
166  * Invokes a Framework State change to the Ready Mode
167  * This method is used as a response to an LEAVE_NORMAL_OPERATION_REQ and has no effect
       in PIL Mode
168  *
169  * @return None
170  */
171 extern void PIL_allowPilSimulation(void);
172
173 /**
174  * Indicates to the framework that the system is not ready for a PIL simulation
175  * Invokes a Framework State change to Normal Operation
176  * This method is used as a response to an ENTER_NORMAL_OPERATION_REQ and has no
      effect in PIL Mode
177  *
178  * @return None
179  */
180 extern void PIL_inhibitPilSimulation(void);
181
182 /**
183  * Used to Request the PIL_CLBK_ENTER_NORMAL_OPERATION_REQ Callback from the
      Application
184  * Only has an effect outside PIL Mode
185  *
186  * @return None
187  */
```

```
188 void PIL_requestNormalMode(void);
189
190 /**
191  * Used to Request the PIL_CLBK_LEAVE_NORMAL_OPERATION_REQ Callback from the
       Application
192  * Only has an effect outside PIL Mode
193  *
194  * @return None
195  */
196 void PIL_requestReadyMode(void);
197
198 /**
199  * Used to check if a PIL simulation is active
200  *
201  * @return None
202  */
203 extern bool PIL_simulationActive(void);
204
205 /**
206  * Receive function.
207  * This function is used by the communication callback to
208  * pass received characters (e.g. from the SCI link) to the PIL
209  * message parser.
210  *
211  * @param aChar character received over communication link
212  */
213 extern void PIL_RA_serialIn(int16_t aChar);
214
215 /**
216  * Transmit function.
217  * This function is used by the communication callback to
218  * obtain characters from the PIL framework that are ready for
219  * transmission (e.g. over the SCI link).
220  *
221  * @param aChar pointer to hold potential character to be transmitted
222  * @return true if a character is ready for transmission and written to aChar
223  */
224 extern bool PIL_RA_serialOut(int16_t* aChar);
225
226 /**
227  * Returns true if sync frame has been transmitted.
228  * Can be used to synchronize communication speed change.
229  */
230 extern bool PIL_RA_serialOutIsSynchronized();
231
232 /**
233  * PIL service function to be called at the beginning of the control
234  * interrupt routine.
235  */
236 extern void PIL_beginInterruptCall(void);
237
238 /**
239  * PIL service function to be called in the background task.
```

```
240  */
241 extern void PIL_backgroundCall(void);
242
243 /**
244  * Receive function for Direct Link.
245  * This function is used by the communication callback to store incoming messages to the PIL
       framework.
246  *
247  * IMPORTANT: This function must be called by the background task.
248  *
249  * This function copys the content of the buffer aBuf from the first element to the (aLen−1)th
       element
250  * into the framework buffer where it is parsed and processed.
251  *
252  * @param *aBuf Pointer to the buffer where the incoming message is stored
253  * @param aLen  Length of incoming message in aBuf
254  * @return None
255  */
256 extern void PIL_RA_parallelIn(uint16_t *aBuf, uint16_t aLen);
257
258 /**
259  * Transmit function for Direct Link.
260  * This function is used by the communication callback to
261  * obtain messages(stored in an internal buffer) from the PIL framework that
262  * are ready for transmission.
263  *
264  * IMPORTANT: This function must be called by the background task.
265  *
266  * If an outgoing message exists, this functions writes the whole message into the buffer specified
267  * in the parameters. Furthermore, the message length is provided for further processing in the
268  * communication Callback.
269  *
270  * @param *aBuf Pointer to the buffer where the outgoing message needs to be stored
271  * @param *aLen Pointer to the variable where the length of the actually outgoing message is stored
272  * @return None
273  */
274 extern void PIL_RA_parallelOut(uint16_t *aBuf, uint16_t *aLen);
275
276 /**
277  * Framework revision.
278  */
279 #define PIL_FRAMEWORK_VERSION 0x0301 // v3.1
280
281 /**
282  * Maximal length for PIL message
283  */
284 #define PIL_MSG_DATA_LEN 30 // 30 words = maximal data length of RA Link data frame
285
286 /**
287  * Globally unique identifier (GUID) autogenerated by PIL prep tool.
288  */
289 extern const unsigned char PIL_D_Guid[];
290
```

```
291 /**
292  * Pointer to PIL GUID.
293  */
294 #define PIL_GUID_PTR (unsigned char*)&PIL_D_Guid[0]
295
296 /**
297  * MACRO for PIL_D_x definitions.
298  */
299 #define PIL_CONST_DEF(type, name, value)\
300   const type PIL_D_ ## name __attribute__ ((used)) = value
301
302 /**
303  * Helper-structure for configuring all PIL symbols.
304  */
305 typedef struct
306 {
307    int q;        //!< fixed-point location
308    float ref;    //!< reference value
309    char *unit;   //!< unit string
310 } pil_var;
311
312 /**
313  * Helper-structure for configuring calibrations (write-access).
314  */
315 typedef struct
316 {
317    float min; //!< minimal value (inclusive)
318    float max; //!< maximal value (inclusive)
319    float dval;  //!< default value
320    int level; //!< access level (-1 = no direct writes allowed)
321 } pil_var_wa;
322
323 /**
324  * Macro for encapsulating read and override probe parameters.
325  */
326 #define PIL_SYMBOL_DEF(name, qloc, ref, unit)\
327   const pil_var PIL_V_ ## name  __attribute__ ((used)) = {qloc, ref, unit}
328
329 /**
330  * Macro for encapsulating calibration parameters.
331  */
332 #define PIL_SYMBOL_CAL_DEF(name, qloc, ref, unit, min, max, dval)\
333     const pil_var PIL_V_ ## name __attribute__ ((used))  = {qloc, ref, unit};\
334     const pil_var_wa PIL_WA_ ## name __attribute__ ((used))  = {min, max, dval, 1}
335
336 /**
337  * Macro for initializing an override probe.
338  */
339 #ifndef PIL_PREP_TOOL
340 #ifdef PREFIX_PROBES
341 #define INIT_OPROBE(probe) \
342    do{\
343      PIL_Probe_F_ ## probe = 0;\
```

```
344   } while (0)
345 #else
346 #define INIT_OPROBE(probe) \
347   do{\
348     probe ## _probeF = 0;\
349   } while (0)
350 #endif
351 #endif
352
353 /**
354  * Macro for initializing a calibration.
355  */
356 #ifndef PIL_PREP_TOOL
357 #define PIL_INIT_CALIBRATION(N, T, P, Q, R) \
358   N = (T)(P * (double)(1L << Q) / (double)(R))
359 #endif
360
361 #ifdef __cplusplus
362 }
363 #endif
364 #endif /* PIL_H_ */
```

control.h

```
1 #ifndef MPC_28335_CONTROL_H_
2 #define MPC_28335_CONTROL_H_
3
4 #include "pil.h"
5
6
7 struct CONTROL_VARS {
8
9   PIL_READ_PROBE(uint16_t, backgroundTaskNumber, 0, 1.0, "");
10  PIL_READ_PROBE(uint16_t, task1StepCtr, 0, 1.0, "");
11  PIL_READ_PROBE(uint16_t, task2StepCtr, 0, 1.0, ""); /*Por si se quiere introducir una tarea
        lenta*/
12  PIL_READ_PROBE(uint16_t, Fs, 0, 1, "");
13
14
15  PIL_OVERRIDE_PROBE(float, il, 0, 1.0, "A");      /*Corriente que circula por la bobina*/
16  PIL_OVERRIDE_PROBE(float, IPV, 0, 1.0, "A");     /*Corriente panel*/
17  PIL_OVERRIDE_PROBE(float, VPV, 0, 1.0, "V");     /*Tension panel*/
18  PIL_OVERRIDE_PROBE(float, Vboost, 0, 1.0, "V");  /*Tensión del "condensador" de salida del
        boost*/
19  PIL_OVERRIDE_PROBE(float, I_rb, 0, 1.0, "V");    /*Corriente de referencia de la bobina*/
20
21  PIL_OVERRIDE_PROBE(int, enable, 0, 1, " ");       /*Señal de habilitacion*/
22  PIL_OVERRIDE_PROBE(int, ciclo, 0, 1, " ");       /*Cada cic/2 de ejecuta el algoritmo, se lee de
        la simulación*/
23  PIL_OVERRIDE_PROBE(float, dutyout, 0, 1, " ");  /*Valor del duty antes de saturación*/
24
```

```
25  PIL_OVERRIDE_PROBE(float, changeVRef, 0, 1.0, "V");    /*Variaciones en la tensión de
       referencia*/
26  PIL_READ_PROBE(float, VRef, 0, 1.0, "V");              /*Tensión de referencia del MPPT*/
27
28  PIL_READ_PROBE(uint16_t, Cmp, 0, 1, "");
29
30  /*Variables para almacenar el valor de tensión y corrientes anteriores, para los delta*/
31
32  float VPVa;
33  float IPVa;
34
35 };
36
37 extern struct CONTROL_VARS ControlVars;
38
39 extern void InitControl(void);
40 extern void DisableActuation(void);
41 extern void EnableActuation(void);
42 extern void ResetControl(void);
43
44 extern void ControlTask1(void);
45 extern void ControlBackground(void);
46
47
48 #endif /*MPC_28335_CONTROL_H_*/
```

includes.h

```
1 #ifndef _INCLUDES_H_
2 #define _INCLUDES_H_
3
4 #define LED_ON_34
5 #define CRYSTAL 30
6 #define BAUD_RATE 115200L
7
8 #if (CRYSTAL == 40)
9 #define SYSCLK_HZ (140000000l)
10 #define LSPCLK_HZ (SYSCLK_HZ / 6l)
11 #define PLX_CPU_RATE 7.143L   // for a 140MHz CPU clock speed (SYSCLKOUT)
12 #elif (CRYSTAL == 30)
13 #define SYSCLK_HZ (30000000l *5l)
14 #define LSPCLK_HZ (SYSCLK_HZ / 6l)
15 #define PLX_CPU_RATE 6.667L   // for a 150MHz CPU clock speed (SYSCLKOUT)
16 #else
17 #error Unsupported value for CRYSTAL
18 #endif
19
20 #define PLX_DELAY_US(A)  DSP28x_usDelay( \
21        ((((long double) A * \
22          1000.0L) / \
23         (long double)PLX_CPU_RATE) - 9.0L) / 5.0L)
```

```
24
25 #define PWM_HZ (10000)
26 #define CONTROL_HZ (10000)
27 #define TICKS_PER_MS (CONTROL_HZ / 1000) //  ticks per millisecond
28
29 #define TASK2_PERIOD 10 // measured in Task 1 ticks
30
31 #include "PeripheralHeaderIncludes.h" // Include all Peripheral Headers
32 #include "DSP2833x_EPwm_defines.h"
33
34 #include <stdint.h>
35 #include "main.h"
36 #include "sci.h"
37 #include "io.h"
38 #include "macros.h"
39 #include "control.h"
40
41 // Pil includes
42 #include "pil.h"
43 #include "pil_ctrl.h"
44
45 #endif
```

io.h

```
1 #ifndef MPC_28335_IO_H_
2 #define MPC_28335_IO_H_
3
4 #include "pil.h"
5
6 struct ANALOG_INPUTS {   //Estas son las entradas que pasarían por el ADC del DSP
7
8   PIL_OVERRIDE_PROBE(float, i_L, 0, 1.0, "A");
9   PIL_OVERRIDE_PROBE(float, IoutPV, 0, 1.0, "A");
10  PIL_OVERRIDE_PROBE(float, VoutPV, 0, 1.0, "V");
11  PIL_OVERRIDE_PROBE(float, VoutBoost, 0, 1.0, "V");
12 };
13
14 struct ADC_OPROBES
15 {
16  PIL_OVERRIDE_PROBE(Uint16, ADCRESULT0, 0, 1.0, "");
17  PIL_OVERRIDE_PROBE(Uint16, ADCRESULT1, 0, 1.0, "");
18  PIL_OVERRIDE_PROBE(Uint16, ADCRESULT2, 0, 1.0, "");
19  PIL_OVERRIDE_PROBE(Uint16, ADCRESULT3, 0, 1.0, "");
20 };
21
22 extern struct ANALOG_INPUTS AIn;
23 extern struct ADC_OPROBES AdcOvrProbes;
24
25 extern void InitGPIO();
26 extern void InitADC();
```

```
extern void InitPWM(uint16_t aPeriod);

#endif
```

macros.h

```
#ifndef MPC_28335_IO_H_
#define MPC_28335_IO_H_

#include "pil.h"

struct ANALOG_INPUTS {   //Estas son las entradas que pasarían por el ADC del DSP

  PIL_OVERRIDE_PROBE(float, i_L, 0, 1.0, "A");
  PIL_OVERRIDE_PROBE(float, IoutPV, 0, 1.0, "A");
  PIL_OVERRIDE_PROBE(float, VoutPV, 0, 1.0, "V");
  PIL_OVERRIDE_PROBE(float, VoutBoost, 0, 1.0, "V");
};

struct ADC_OPROBES
{
  PIL_OVERRIDE_PROBE(Uint16, ADCRESULT0, 0, 1.0, "");
  PIL_OVERRIDE_PROBE(Uint16, ADCRESULT1, 0, 1.0, "");
  PIL_OVERRIDE_PROBE(Uint16, ADCRESULT2, 0, 1.0, "");
  PIL_OVERRIDE_PROBE(Uint16, ADCRESULT3, 0, 1.0, "");
};

extern struct ANALOG_INPUTS AIn;
extern struct ADC_OPROBES AdcOvrProbes;

extern void InitGPIO();
extern void InitADC();
extern void InitPWM(uint16_t aPeriod);

#endif
```

main.h

```
#ifndef MPC_28335_IO_H_
#define MPC_28335_IO_H_

#include "pil.h"

struct ANALOG_INPUTS {   //Estas son las entradas que pasarían por el ADC del DSP

  PIL_OVERRIDE_PROBE(float, i_L, 0, 1.0, "A");
  PIL_OVERRIDE_PROBE(float, IoutPV, 0, 1.0, "A");
  PIL_OVERRIDE_PROBE(float, VoutPV, 0, 1.0, "V");
  PIL_OVERRIDE_PROBE(float, VoutBoost, 0, 1.0, "V");
};
```

```
13
14 struct ADC_OPROBES
15 {
16   PIL_OVERRIDE_PROBE(Uint16, ADCRESULT0, 0, 1.0, "");
17   PIL_OVERRIDE_PROBE(Uint16, ADCRESULT1, 0, 1.0, "");
18   PIL_OVERRIDE_PROBE(Uint16, ADCRESULT2, 0, 1.0, "");
19   PIL_OVERRIDE_PROBE(Uint16, ADCRESULT3, 0, 1.0, "");
20 };
21
22 extern struct ANALOG_INPUTS AIn;
23 extern struct ADC_OPROBES AdcOvrProbes;
24
25 extern void InitGPIO();
26 extern void InitADC();
27 extern void InitPWM(uint16_t aPeriod);
28
29 #endif
```

pil_ctrl.h

```
1 #ifndef PIL_CTRL_H
2 #define PIL_CTRL_H
3
4 extern void PilCallback(PIL_CtrlCallbackReq_t action);
5
6 #endif
```

sci.h

```
1 #ifndef MPC_28335_SCI_H_
2 #define MPC_28335_SCI_H_
3
4 // bit-masks for "mode"
5 #define ONE_STOPBIT 0x00
6 #define TWO_STOPBITS 0x80
7 #define NO_PARITY 0x00
8 #define ODD_PARITY 0x20
9 #define EVEN_PARITY 0x60
10 #define SEVEN_BITS 0x06
11 #define EIGHT_BITS 0x07
12 #define SCI_LOOPBACK 0x10
13
14 // compile options
15 #define USE_SCI_FIFO
16
17 // receive character
18 extern int16_t SCIGetChar(void);
19 // send character
20 extern void SCIPutChar(char c);
21 // initialize serial interface
```

```
22 extern void SCIInit (uint16_t mode, uint32_t baudrate, uint32_t clk);
23 // write string
24 extern void SCIWriteString(const char *s);
25
26 extern void SCIPoll();
27
28 #endif /* MPC_28335_SCI_H_ */
```

Anexo C

Ficheros de C

main.c

```c
#include "includes.h"

/*Prototipo de funciones*/
void DeviceInit(void);
void MemCopy(Uint16 *SourceAddr, Uint16 *SourceEndAddr, Uint16 *DestAddr);
void InitFlash();

/*linker addresses, necesarios para copiar desde la flash a la RAM*/
extern Uint16 RamfuncsLoadStart, RamfuncsLoadEnd, RamfuncsRunStart;

#define LED_BLINK_PERIOD_2 (500 * TICKS_PER_MS)
Uint16 LEDTimer = 0;
struct CTRL_DISPATCH_VARS CtrlDispatchVars;

interrupt void ControlDispatcher(void)
{
  CtrlDispatchVars.interruptNesting++;

  PIL_beginInterruptCall();

  /*Reinicializar para la siguiente secuencia del ADC*/
  AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;          // Reset SEQ1
  AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;        // Clear INT SEQ1 bit
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;    //Reconocer la interrupción
  IER |= M_INT1;

  LEDTimer++;

  EINT;

  ControlTask1();

  DINT;
  CtrlDispatchVars.interruptNesting--;
```

```
35 }
36
37
38 void InitControlDispatcher()
39 {
40   /*Configuración de los disparos del ADC*/
41   EPwm1Regs.ETSEL.bit.SOCAEN = 1;          // Habilitar SOC en el grupo A
42   EPwm1Regs.ETSEL.bit.SOCASEL = 6;         // Seleccionar SOC cuando CMPB en upcount
43   EPwm1Regs.ETPS.bit.SOCAPRD = 1;          // Genera un pulso en el primer evento
44   AdcRegs.ADCTRL2.bit.EPWM_SOCA_SEQ1 = 1;// Habilitar SOCA desde el ePWM para
       empezar la secuencia
45
46   /*Habilitar la interrupción de la secuencia cada EOS*/
47   AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;
48   EALLOW;
49   PieVectTable.ADCINT = &ControlDispatcher;
50   EDIS;
51   PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
52   IER |= M_INT1;
53
54   CtrlDispatchVars.interruptNesting = 0;
55 }
56
57 void BlinkLed(void){
58   if(LEDTimer > LED_BLINK_PERIOD_2){
59 #ifdef LED_ON_33
60     GpioDataRegs.GPBTOGGLE.bit.GPIO33 = 1;
61 #endif
62 #ifdef LED_ON_32
63     GpioDataRegs.GPBTOGGLE.bit.GPIO32 = 1;
64 #endif
65 #ifdef LED_ON_34
66     GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1;
67 #endif
68     LEDTimer = 0;
69   }
70 }
71
72 void main(void)
73 {
74   DeviceInit();
75
76   MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
77
78   InitFlash();
79
80   EALLOW;
81 #ifdef LED_ON_33
82   GpioCtrlRegs.GPBMUX1.bit.GPIO33 = 0;
83   GpioCtrlRegs.GPBDIR.bit.GPIO33 = 1;
84 #endif
85 #ifdef LED_ON_32
86   GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 0;
```

```
87   GpioCtrlRegs.GPBDIR.bit.GPIO32 = 1;
88 #endif
89 #ifdef LED_ON_34
90   GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0;
91   GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;
92 #endif
93   EDIS;
94 
95   DINT;
96   IER = 0x0000;
97   IFR = 0x0000;
98 
99   EALLOW;
100  SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
101  EDIS;
102 
103  /*Configuración de la interfaz serie*/
104  SCIInit(ONE_STOPBIT+ NO_PARITY + EIGHT_BITS, BAUD_RATE, LSPCLK_HZ);
105  SCIWriteString("\n\rRS-232 Initialized.\n\r");
106 
107  /*Inicialización del PIL*/
108  PIL_init();
109  PIL_setLinkParams(PIL_GUID_PTR, (PIL_CommCallbackPtr_t)SCIPoll);
110  PIL_setCtrlCallback((PIL_CtrlCallbackPtr_t)PilCallback);
111 
112  /*Inicialización de variables*/
113  PilInitOverrideProbes();
114 
115  /*Iniciamos las tareas de control*/
116  InitControl();
117  InitControlDispatcher();
118 
119  /*Habilitar interrumpimientos globales y eventos de depuración en tiempo real de mayor prioridad:
       */
120  EINT;   // Habilitar la interrupción global INTM
121  ERTM;   // Habilitar la interrupción global de tiempo real DBGM
122 
123  EALLOW;
124  SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
125  EDIS;
126 
127  /*Ready for PIL?*/
128  PIL_requestReadyMode();
129 
130  /*Bucle principal*/
131  for(;;)
132  {
133    BlinkLed();
134    PIL_backgroundCall();
135    ControlBackground();
136  }
137 }
```

control.c

```
#include "includes.h"
#include "math.h"

extern void DSP28x_usDelay(unsigned long aCount);

#define INV_SQRT3_Q15 (int16_t)(1.0 / 1.73 * 32768.0)
#define ONE_HALF_Q15 (uint16_t)(0.5 * (double)(32768))

struct CONTROL_VARS ControlVars;

float TPWM; //Tpwm en seg

float Vref;
float changeinref;
float i_L, il;
float I_pv, I_pva;
float V_pv, V_pva;
float V_Boost;
float deltaV, deltaI;
int flag;
float dPdV;
float tempPowerVariable;
int cont, start, runfirstflag;
int cic;


float kpe=1;
float kpi=10;
float kie=10;
float kii=500;
float Ts=0.0001;
float errore, errorea;
float errori, erroria;
float D;
float Iref;
float P_ref;
float ref_mod;

/**
 *Inicializa la tarea de control
 */
void InitControl(void){
  uint16_t pwmPeriod = (uint16_t)(SYSCLK_HZ / 2 / PWM_HZ / 2);

  InitPWM(pwmPeriod);
  InitADC();

  ControlVars.Fs = CONTROL_HZ;


```

```
51  ControlVars.Cmp = 0;
52
53  EPwm1Regs.CMPA.half.CMPA = ControlVars.Cmp;
54
55  TPWM = 1./ControlVars.Fs;
56
57  flag=1;
58  cont=0;
59  runfirstflag=1;
60  changeinref=0.0;
61  Vref=0.0;
62  D=0.0;
63
64  P_ref=0;
65  ref_mod=0;
66
67  SET_OPROBE(ControlVars.dutyout, 0);
68
69  errorea=0.0;
70  erroria=0.0;
71
72  DisableActuation();
73  ResetControl();
74 }
75
76 /**
77  * Se llama cuando se para, aqui hay que deshabiltiar los pwm.
78  */
79 void DisableActuation(void){
80
81  EALLOW;
82  EPwm1Regs.TZFRC.bit.OST=1;
83  EDIS;
84 }
85
86
87 void EnableActuation(void){
88
89  ResetControl();
90
91 #ifndef KEEP_POWERSTAGE_OFF
92  // enable actuators
93  EALLOW;
94  EPwm1Regs.TZCLR.bit.OST=1;
95  EDIS;
96 #endif
97 }
98
99 /**
100  *Se llama cuando se comienza el control
101  */
102 void ResetControl(void){
103
```

```
104   ControlVars.task1StepCtr = 0;
105   ControlVars.task2StepCtr = 0;
106   ControlVars.backgroundTaskNumber = 0;
107
108   changeinref=0.0;
109   Vref=0.0;
110   V_pva=0.0;
111   I_pva=0.0;
112   cont=0;
113   P_ref=0;
114   ref_mod=0;
115
116   flag=1;
117   runfirstflag=1;
118   cont=0;
119   D=0.5;
120   errorea=0.0;
121   erroria=0.0;
122
123   ControlVars.Cmp = 0;
124
125   EPwm1Regs.CMPA.half.CMPA = ControlVars.Cmp;
126
127   ControlVars.VRef= Vref;
128   ControlVars.VPVa= V_pva;
129   ControlVars.IPVa= I_pva;
130   SET_OPROBE(ControlVars.changeVRef, changeinref);
131
132 }
133
134
135 void ControlTask1(){
136
137   ControlVars.task1StepCtr++;
138
139
140   SET_OPROBE(ControlVars.enable, 0);
141   start = ControlVars.enable;
142
143   SET_OPROBE(ControlVars.ciclo, 0);
144   cic = ControlVars.ciclo;
145
146   /*Lectura de entradas Analógicas*/
147
148   SET_OPROBE(AdcOvrProbes.ADCRESULT0, 0);
149   SET_OPROBE(AdcOvrProbes.ADCRESULT1, 0);
150   SET_OPROBE(AdcOvrProbes.ADCRESULT2, 0);
151   SET_OPROBE(AdcOvrProbes.ADCRESULT3, 0);
152
153   /*Calculo del valor de las medidas reales (debido a la fase de adaptación)*/
154
155   i_L =  (AdcOvrProbes.ADCRESULT0)*0.030517578;
156   I_pv = (AdcOvrProbes.ADCRESULT1)*0.00732421875;
```

```
157 V_pv = (AdcOvrProbes.ADCRESULT2)*0.126953125;
158 V_Boost = (AdcOvrProbes.ADCRESULT3)*0.146484375;
159
160 /*Las guardamos en variables para poder sacarlas y ver que se corresponden*/
161
162 SET_OPROBE(ControlVars.il, i_L);
163 SET_OPROBE(ControlVars.IPV, I_pv);
164 SET_OPROBE(ControlVars.VPV, V_pv);
165 SET_OPROBE(ControlVars.Vboost, V_Boost);
166
167 if (start==1) {
168
169
170  if (runfirstflag==1)
171  {
172   cont=(cic-1);
173   runfirstflag=0;
174  }
175
176 cont++;
177
178 if (cont==cic) {   /*El MPPT se recalcula cada 0.0cic s*/
179
180  cont=0;
181
182 /*Leemos el valor de las variables de la iteracion anterior*/
183
184 V_pva = ControlVars.VPVa;
185 I_pva = ControlVars.IPVa;
186
187 /*Solo en la primera iteracion*/
188
189 if(flag){
190   V_pva = V_pv;
191   I_pva = I_pv;
192   flag=0; }
193
194 /*Calculo de los incrementos*/
195
196 deltaV = (V_pv - V_pva);
197 deltaI = (I_pv - I_pva);
198
199 //
    ////////////////////////////////////////////////////////////////////////////////////////////////////////
200
201 if( (deltaV < 0.0001) && (deltaV > -0.0001) )           /*Si deltaV = 0....*/
202    {
203       if( (deltaI < 0.001) && (deltaI > -0.001) )   /*Vemos si deltaI = 0 (MPPT) ...*/
204       {
205          changeinref = 0;
206
207       }
```

```
208        else
209        {
210          changeinref = (0.1 * deltaI);
211        }
212     }
213     else
214     {
215       dPdV= V_pv*((I_pv/V_pv) + (deltaI/deltaV));            /*Calculo de la pendiente*/
216         tempPowerVariable = ((I_pv/V_pv) + (deltaI/deltaV));  /*Calculamos DI/DV + I/V para
        ver el error que cometemos*/
217
218        if( (tempPowerVariable < 0.001) && (tempPowerVariable > -0.001) )
219        {
220           changeinref = 0;
221
222        }
223        else
224        {
225          changeinref = (0.4 * dPdV);
226        }
227     }
228
229   Vref= (V_pv + changeinref);
230
231   ControlVars.VPVa = V_pv;
232   ControlVars.IPVa = I_pv;
233
234   SET_OPROBE(ControlVars.changeVRef, changeinref);
235
236  } /*Fin if cic*/
237
238
239  /*External control loop*/
240    errore=0.5*((Vref*Vref) - (V_pv*V_pv));
241  P_ref= (P_ref + kpe*errore - kpe*errorea + kie*Ts*errorea);
242  errorea=errore;
243
244  /*Inner control loop*/
245  Iref= -P_ref/V_pv;
246  errori=(i_L - Iref);
247  ref_mod= (ref_mod + kpi*errori - kpi*erroria + kii*Ts*erroria);
248  erroria=errori;
249
250
251  D=(ref_mod/V_Boost);
252
253  SET_OPROBE(ControlVars.dutyout, D);
254
255    if(D>1){
256      D=1.0;}
257    if(D<0){
258      D=0.0;}
259
```

```
260   SET_OPROBE(ControlVars.I_rb, Iref);
261   ControlVars.Cmp = (int)((float)(EPwm1Regs.TBPRD)*D);
262   EPwm1Regs.CMPA.half.CMPA = ControlVars.Cmp;
263
264
265 }  /*Fin if*/
266
267 //
      ////////////////////////////////////////////////////////////////////////////////////////////////
268
269    ControlVars.VRef = Vref;
270
271 //
      ////////////////////////////////////////////////////////////////////////////////////////////////
272
273
274  }
275
276
277 /**
278  * Funcion del bucle, maquina de estados
279  */
280 void ControlBackground(void)
281 {
282   switch(ControlVars.backgroundTaskNumber)
283   {
284     case 0:
285       PLX_DELAY_US(300);
286       break;
287
288     case 1:
289       PLX_DELAY_US(1000);
290       break;
291
292     case 2:
293       PLX_DELAY_US(600);
294       break;
295
296   }
297   ControlVars.backgroundTaskNumber++;
298   if(ControlVars.backgroundTaskNumber > 2)
299   {
300     ControlVars.backgroundTaskNumber = 0;
301   }
302 }
```

DevInit_F2833x.c

```
1 #include "includes.h"
2
```

```
3 // Funciones  que deben ejecutarse desde la RAM necesitan ser asignadas a una nueva sección.
4 //Esta sección será mapeada a una dirección de carga y ejecución usando un archivo cmd
5
6 #pragma CODE_SECTION(InitFlash, "ramfuncs");
7
8 //Prototipo de funciones
9 void DeviceInit(void);
10 void PieCntlInit(void);
11 void PieVectTableInit(void);
12 void WDogDisable(void);
13 void PLLset(Uint16);
14 void ISR_ILLEGAL(void);
15
16 //
   ------------------------------------------------------------------------
17 //  Configure Device for target Application Here
18 //
   ------------------------------------------------------------------------
19 void DeviceInit(void)
20 {
21   WDogDisable();
22   DINT;     // Deshabilita todas las interrupciones
23   IER = 0x0000; // Desahibilita interrupcoines de la CPU
24   IFR = 0x0000; // Limpia todos los flags de interrupción de  CPU
25
26
27 #if (CRYSTAL == 40)
28   PLLset(0x7);  // x 3.4 = 140
29 #elif (CRYSTAL == 30)
30   PLLset(0xA);  // x 5 = 150
31 #else
32 #error Unsupported value for CRYSTAL
33 #endif
34
35 // Inicializa la interrupción
36
37   PieCntlInit();
38   PieVectTableInit();
39
40    EALLOW; //Permitir acceso a los registros protegidos.
41
42 // HIGH / LOW SPEED CLOCKS prescalado
43
44    SysCtrlRegs.HISPCP.all = 0x0003;   // Sysclk / 6 (25 MHz)
45    SysCtrlRegs.LOSPCP.all = 0x0003;   // Sysclk / 6 (25 MHz)
46
47
48 // PERIPHERAL CLOCK ENABLES
49
50
51    SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;    // ADC
```

```
52    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
53    SysCtrlRegs.PCLKCR0.bit.I2CAENCLK = 0;   // I2C
54    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
55    SysCtrlRegs.PCLKCR0.bit.SPIAENCLK=1;     // SPI−A
56    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
57    SysCtrlRegs.PCLKCR0.bit.SCIAENCLK=1;     // SCI−A
58    SysCtrlRegs.PCLKCR0.bit.SCIBENCLK=0;     // SCI−B
59    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
60    SysCtrlRegs.PCLKCR0.bit.ECANAENCLK=0;    // eCAN−A
61    SysCtrlRegs.PCLKCR0.bit.ECANBENCLK=0;    // eCAN−B
62    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
63    SysCtrlRegs.PCLKCR1.bit.ECAP1ENCLK = 0;  // eCAP1
64    SysCtrlRegs.PCLKCR1.bit.ECAP2ENCLK = 0;  // eCAP2
65    SysCtrlRegs.PCLKCR1.bit.ECAP3ENCLK = 0;  // eCAP3
66    SysCtrlRegs.PCLKCR1.bit.ECAP4ENCLK = 0;  // eCAP4
67    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
68    SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;   // ePWM1
69    SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 0;   // ePWM2
70    SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 0;   // ePWM3
71    SysCtrlRegs.PCLKCR1.bit.EPWM4ENCLK = 0;   // ePWM4
72    SysCtrlRegs.PCLKCR1.bit.EPWM5ENCLK = 0;   // ePWM5
73    SysCtrlRegs.PCLKCR1.bit.EPWM6ENCLK = 0;   // ePWM6
74    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
75    SysCtrlRegs.PCLKCR1.bit.EQEP1ENCLK = 0;   // eQEP1
76    SysCtrlRegs.PCLKCR1.bit.EQEP2ENCLK = 0;   // eQEP2
77    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
78    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;   // Enable TBCLK
79    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
80
81
82  // GPIO (GENERAL PURPOSE I/O) CONFIG
83
84  //
      −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
85  //
      −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
86  //  GPIO−00 − PIN FUNCTION = −−Spare−−
87   GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 0;   // 0=GPIO,  1=EPWM1A,  2=Resv,  3=Resv
88   GpioCtrlRegs.GPADIR.bit.GPIO0 = 0;    // 1=OUTput,  0=INput
89  //  GpioDataRegs.GPACLEAR.bit.GPIO0 = 1;  // uncomment if −−> Set Low initially
90  //  GpioDataRegs.GPASET.bit.GPIO0 = 1;    // uncomment if −−> Set High initially
91  //
      −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
92  //  GPIO−01 − PIN FUNCTION = −−Spare−−
93   GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 0;   // 0=GPIO,  1=EPWM1B,  2=ECAP6,  3=MFSR−B
94   GpioCtrlRegs.GPADIR.bit.GPIO1 = 0;    // 1=OUTput,  0=INput
95  //  GpioDataRegs.GPACLEAR.bit.GPIO1 = 1;  // uncomment if −−> Set Low initially
96  //  GpioDataRegs.GPASET.bit.GPIO1 = 1;    // uncomment if −−> Set High initially
97  //
      −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
```

```
98 //  GPIO-02 - PIN FUNCTION = --Spare--
99   GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 0;   // 0=GPIO, 1=EPWM2A, 2=Resv, 3=Resv
100  GpioCtrlRegs.GPADIR.bit.GPIO2 = 0;    // 1=OUTput, 0=INput
101 //  GpioDataRegs.GPACLEAR.bit.GPIO2 = 1;  // uncomment if --> Set Low initially
102 //  GpioDataRegs.GPASET.bit.GPIO2 = 1;    // uncomment if --> Set High initially
103 //
    -----------------------------------------------------------------------
104 //  GPIO-03 - PIN FUNCTION = --Spare--
105  GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 0;   // 0=GPIO, 1=EPWM2B, 2=ECAP5, 3=MCLKR-
     B
106  GpioCtrlRegs.GPADIR.bit.GPIO3 = 0;    // 1=OUTput, 0=INput
107 //  GpioDataRegs.GPACLEAR.bit.GPIO3 = 1;  // uncomment if --> Set Low initially
108 //  GpioDataRegs.GPASET.bit.GPIO3 = 1;    // uncomment if --> Set High initially
109 //
    -----------------------------------------------------------------------
110 //  GPIO-04 - PIN FUNCTION = --Spare--
111  GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 0;   // 0=GPIO, 1=EPWM3A, 2=Resv, 3=Resv
112  GpioCtrlRegs.GPADIR.bit.GPIO4 = 0;    // 1=OUTput, 0=INput
113 //  GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;  // uncomment if --> Set Low initially
114 //  GpioDataRegs.GPASET.bit.GPIO4 = 1;    // uncomment if --> Set High initially
115 //
    -----------------------------------------------------------------------
116 //  GPIO-05 - PIN FUNCTION = --Spare--
117  GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 0;   // 0=GPIO, 1=EPWM3B, 2=MFSR-A, 3=ECAP1
118  GpioCtrlRegs.GPADIR.bit.GPIO5 = 0;    // 1=OUTput, 0=INput
119 //  GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;  // uncomment if --> Set Low initially
120 //  GpioDataRegs.GPASET.bit.GPIO5 = 1;    // uncomment if --> Set High initially
121 //
    -----------------------------------------------------------------------
122 //  GPIO-06 - PIN FUNCTION = --Spare--
123  GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 0;   // 0=GPIO, 1=EPWM4A, 2=SYNCI, 3=SYNCO
124  GpioCtrlRegs.GPADIR.bit.GPIO6 = 0;    // 1=OUTput, 0=INput
125 //  GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;  // uncomment if --> Set Low initially
126 //  GpioDataRegs.GPASET.bit.GPIO6 = 1;    // uncomment if --> Set High initially
127 //
    -----------------------------------------------------------------------
128 //  GPIO-07 - PIN FUNCTION = --Spare--
129  GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 0;   // 0=GPIO, 1=EPWM4B, 2=MCLKR-A, 3=
     ECAP2
130  GpioCtrlRegs.GPADIR.bit.GPIO7 = 0;    // 1=OUTput, 0=INput
131 //  GpioDataRegs.GPACLEAR.bit.GPIO7 = 1;  // uncomment if --> Set Low initially
132 //  GpioDataRegs.GPASET.bit.GPIO7 = 1;    // uncomment if --> Set High initially
133 //
    -----------------------------------------------------------------------
134 //  GPIO-08 - PIN FUNCTION = --Spare--
135  GpioCtrlRegs.GPAMUX1.bit.GPIO8 = 0;   // 0=GPIO, 1=EPWM5A, 2=CANTX-B, 3=
```

```
      ADCSOC−A
136   GpioCtrlRegs.GPADIR.bit.GPIO8 = 0;    // 1=OUTput,  0=INput
137 //  GpioDataRegs.GPACLEAR.bit.GPIO8 = 1;  // uncomment if −−> Set Low initially
138 //  GpioDataRegs.GPASET.bit.GPIO8 = 1;    // uncomment if −−> Set High initially
139 //
      −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
140 //  GPIO−09 − PIN FUNCTION = −−Spare−−
141   GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 0;   // 0=GPIO,  1=EPWM5B,  2=SCITX−B,  3=
      ECAP3
142   GpioCtrlRegs.GPADIR.bit.GPIO9 = 0;    // 1=OUTput,  0=INput
143 //  GpioDataRegs.GPACLEAR.bit.GPIO9 = 1;  // uncomment if −−> Set Low initially
144 //  GpioDataRegs.GPASET.bit.GPIO9 = 1;    // uncomment if −−> Set High initially
145 //
      −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
146 //  GPIO−10 − PIN FUNCTION = −−Spare−−
147   GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 0;  // 0=GPIO,  1=EPWM6A,  2=CANRX−B,  3=
      ADCSOC−B
148   GpioCtrlRegs.GPADIR.bit.GPIO10 = 0;   // 1=OUTput,  0=INput
149 //  GpioDataRegs.GPACLEAR.bit.GPIO10 = 1; // uncomment if −−> Set Low initially
150 //  GpioDataRegs.GPASET.bit.GPIO10 = 1;   // uncomment if −−> Set High initially
151 //
      −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
152 //  GPIO−11 − PIN FUNCTION = −−Spare−−
153   GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 0;  // 0=GPIO,  1=EPWM6B,  2=SCIRX−B,  3=
      ECAP4
154   GpioCtrlRegs.GPADIR.bit.GPIO11 = 0;   // 1=OUTput,  0=INput
155 //  GpioDataRegs.GPACLEAR.bit.GPIO11 = 1; // uncomment if −−> Set Low initially
156 //  GpioDataRegs.GPASET.bit.GPIO11 = 1;   // uncomment if −−> Set High initially
157 //
      −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
158 //  GPIO−12 − PIN FUNCTION = −−Spare−−
159   GpioCtrlRegs.GPAMUX1.bit.GPIO12 = 0;  // 0=GPIO,  1=TZ1,  2=CANTX−B,  3=MDX−B
160   GpioCtrlRegs.GPADIR.bit.GPIO12 = 0;   // 1=OUTput,  0=INput
161 //  GpioDataRegs.GPACLEAR.bit.GPIO12 = 1; // uncomment if −−> Set Low initially
162 //  GpioDataRegs.GPASET.bit.GPIO12 = 1;   // uncomment if −−> Set High initially
163 //
      −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
164 //  GPIO−13 − PIN FUNCTION = −−Spare−−
165   GpioCtrlRegs.GPAMUX1.bit.GPIO13 = 0;  // 0=GPIO,  1=TZ2,  2=CANRX−B,  3=MDR−B
166   GpioCtrlRegs.GPADIR.bit.GPIO13 = 0;   // 1=OUTput,  0=INput
167 //  GpioDataRegs.GPACLEAR.bit.GPIO13 = 1; // uncomment if −−> Set Low initially
168 //  GpioDataRegs.GPASET.bit.GPIO13 = 1;   // uncomment if −−> Set High initially
169 //
      −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
170 //  GPIO−14 − PIN FUNCTION = −−Spare−−
171   GpioCtrlRegs.GPAMUX1.bit.GPIO14 = 0;  // 0=GPIO,  1=TZ3,  2=SCITX−B,  3=MCLKX−B
172   GpioCtrlRegs.GPADIR.bit.GPIO14 = 0;   // 1=OUTput,  0=INput
```

```
173 //  GpioDataRegs.GPACLEAR.bit.GPIO14 = 1; // uncomment if --> Set Low initially
174 //  GpioDataRegs.GPASET.bit.GPIO14 = 1;   // uncomment if --> Set High initially
175 //
    --------------------------------------------------------------------------
176 //  GPIO-15 - PIN FUNCTION = --Spare--
177   GpioCtrlRegs.GPAMUX1.bit.GPIO15 = 0;  // 0=GPIO,  1=TZ4,  2=SCIRX-B,  3=MFSX-B
178   GpioCtrlRegs.GPADIR.bit.GPIO15 = 0;   // 1=OUTput,  0=INput
179 //  GpioDataRegs.GPACLEAR.bit.GPIO15 = 1; // uncomment if --> Set Low initially
180 //  GpioDataRegs.GPASET.bit.GPIO15 = 1;   // uncomment if --> Set High initially
181 //
    --------------------------------------------------------------------------
182 //
    --------------------------------------------------------------------------
183 //  GPIO-16 - PIN FUNCTION = --Spare--
184   GpioCtrlRegs.GPAMUX2.bit.GPIO16 = 0;  // 0=GPIO,  1=SPISIMO-A,  2=CANTX-B,  3=
    TZ5
185   GpioCtrlRegs.GPADIR.bit.GPIO16 = 0;   // 1=OUTput,  0=INput
186 //  GpioDataRegs.GPACLEAR.bit.GPIO16 = 1; // uncomment if --> Set Low initially
187 //  GpioDataRegs.GPASET.bit.GPIO16 = 1;   // uncomment if --> Set High initially
188 //
    --------------------------------------------------------------------------
189 //  GPIO-17 - PIN FUNCTION = --Spare--
190   GpioCtrlRegs.GPAMUX2.bit.GPIO17 = 0;  // 0=GPIO,  1=SPISOMI-A,  2=CANRX-B,  3=
    TZ6
191   GpioCtrlRegs.GPADIR.bit.GPIO17 = 0;   // 1=OUTput,  0=INput
192 //  GpioDataRegs.GPACLEAR.bit.GPIO17 = 1; // uncomment if --> Set Low initially
193 //  GpioDataRegs.GPASET.bit.GPIO17 = 1;   // uncomment if --> Set High initially
194 //
    --------------------------------------------------------------------------
195 //  GPIO-18 - PIN FUNCTION = --Spare--
196   GpioCtrlRegs.GPAMUX2.bit.GPIO18 = 0;  // 0=GPIO,  1=SPICLK-A,  2=SCITX-B,  3=
    CANRX-A
197   GpioCtrlRegs.GPADIR.bit.GPIO18 = 0;   // 1=OUTput,  0=INput
198 //  GpioDataRegs.GPACLEAR.bit.GPIO18 = 1; // uncomment if --> Set Low initially
199 //  GpioDataRegs.GPASET.bit.GPIO18 = 1;   // uncomment if --> Set High initially
200 //
    --------------------------------------------------------------------------
201 //  GPIO-19 - PIN FUNCTION = --Spare--
202   GpioCtrlRegs.GPAMUX2.bit.GPIO19 = 0;  // 0=GPIO,  1=SPISTE-A,  2=SCIRX-B,  3=
    CANTX-A
203   GpioCtrlRegs.GPADIR.bit.GPIO19 = 0;   // 1=OUTput,  0=INput
204 //  GpioDataRegs.GPACLEAR.bit.GPIO19 = 1; // uncomment if --> Set Low initially
205 //  GpioDataRegs.GPASET.bit.GPIO19 = 1;   // uncomment if --> Set High initially
206 //
    --------------------------------------------------------------------------
207 //  GPIO-20 - PIN FUNCTION = --Spare--
```

```
208   GpioCtrlRegs.GPAMUX2.bit.GPIO20 = 0;  // 0=GPIO,  1=EQEPA-1,  2=MDX-A,  3=
        CANTX-B
209   GpioCtrlRegs.GPADIR.bit.GPIO20 = 0;   // 1=OUTput,  0=INput
210 //  GpioDataRegs.GPACLEAR.bit.GPIO20 = 1; // uncomment if --> Set Low initially
211 //  GpioDataRegs.GPASET.bit.GPIO20 = 1;   // uncomment if --> Set High initially
212 //
        -----------------------------------------------------------------
213 //  GPIO-21 - PIN FUNCTION = --Spare--
214   GpioCtrlRegs.GPAMUX2.bit.GPIO21 = 0;  // 0=GPIO,  1=EQEPB-1,  2=MDR-A,  3=
        CANRX-B
215   GpioCtrlRegs.GPADIR.bit.GPIO21 = 0;   // 1=OUTput,  0=INput
216 //  GpioDataRegs.GPACLEAR.bit.GPIO21 = 1; // uncomment if --> Set Low initially
217 //  GpioDataRegs.GPASET.bit.GPIO21 = 1;   // uncomment if --> Set High initially
218 //
        -----------------------------------------------------------------
219 //  GPIO-22 - PIN FUNCTION = --Spare--
220   GpioCtrlRegs.GPAMUX2.bit.GPIO22 = 0;  // 0=GPIO,  1=EQEPS-1,  2=MCLKX-A,  3=
        SCITX-B
221   GpioCtrlRegs.GPADIR.bit.GPIO22 = 0;   // 1=OUTput,  0=INput
222 //  GpioDataRegs.GPACLEAR.bit.GPIO22 = 1; // uncomment if --> Set Low initially
223 //  GpioDataRegs.GPASET.bit.GPIO22 = 1;   // uncomment if --> Set High initially
224 //
        -----------------------------------------------------------------
225 //  GPIO-23 - PIN FUNCTION = --Spare--
226   GpioCtrlRegs.GPAMUX2.bit.GPIO23 = 0;  // 0=GPIO,  1=EQEPI-1,  2=MFSX-A,  3=SCIRX
        -B
227   GpioCtrlRegs.GPADIR.bit.GPIO23 = 0;   // 1=OUTput,  0=INput
228 //  GpioDataRegs.GPACLEAR.bit.GPIO23 = 1; // uncomment if --> Set Low initially
229 //  GpioDataRegs.GPASET.bit.GPIO23 = 1;   // uncomment if --> Set High initially
230 //
        -----------------------------------------------------------------
231 //  GPIO-24 - PIN FUNCTION = --Spare--
232   GpioCtrlRegs.GPAMUX2.bit.GPIO24 = 0;  // 0=GPIO,  1=ECAP1,  2=EQEPA-2,  3=MDX-B
233   GpioCtrlRegs.GPADIR.bit.GPIO24 = 0;   // 1=OUTput,  0=INput
234 //  GpioDataRegs.GPACLEAR.bit.GPIO24 = 1; // uncomment if --> Set Low initially
235 //  GpioDataRegs.GPASET.bit.GPIO24 = 1;   // uncomment if --> Set High initially
236 //
        -----------------------------------------------------------------
237 //  GPIO-25 - PIN FUNCTION = --Spare--
238   GpioCtrlRegs.GPAMUX2.bit.GPIO25 = 0;  // 0=GPIO,  1=ECAP2,  2=EQEPB-2,  3=MDR-B
239   GpioCtrlRegs.GPADIR.bit.GPIO25 = 0;   // 1=OUTput,  0=INput
240 //  GpioDataRegs.GPACLEAR.bit.GPIO25 = 1; // uncomment if --> Set Low initially
241 //  GpioDataRegs.GPASET.bit.GPIO25 = 1;   // uncomment if --> Set High initially
242 //
        -----------------------------------------------------------------
243 //  GPIO-26 - PIN FUNCTION = --Spare--
244   GpioCtrlRegs.GPAMUX2.bit.GPIO26 = 0;  // 0=GPIO,  1=ECAP3,  2=EQEPI-2,  3=MCLKX
```

```
        −B
245   GpioCtrlRegs.GPADIR.bit.GPIO26 = 0;   // 1=OUTput,  0=INput
246 //  GpioDataRegs.GPACLEAR.bit.GPIO26 = 1; // uncomment if −−> Set Low initially
247 //  GpioDataRegs.GPASET.bit.GPIO26 = 1;   // uncomment if −−> Set High initially
248 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
249 //  GPIO−27 − PIN FUNCTION = −−Spare−−
250   GpioCtrlRegs.GPAMUX2.bit.GPIO27 = 0;  // 0=GPIO,  1=ECAP4,  2=EQEPS−2,  3=MFSX−
        B
251   GpioCtrlRegs.GPADIR.bit.GPIO27 = 0;   // 1=OUTput,  0=INput
252 //  GpioDataRegs.GPACLEAR.bit.GPIO27 = 1; // uncomment if −−> Set Low initially
253 //  GpioDataRegs.GPASET.bit.GPIO27 = 1;   // uncomment if −−> Set High initially
254 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
255 //  GPIO−28 − PIN FUNCTION = SCI−RX
256   GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 0;  // 0=GPIO,  1=SCIRX−A,  2=Resv,  3=Resv
257   GpioCtrlRegs.GPADIR.bit.GPIO28 = 0;   // 1=OUTput,  0=INput
258 //  GpioDataRegs.GPACLEAR.bit.GPIO28 = 1; // uncomment if −−> Set Low initially
259 //  GpioDataRegs.GPASET.bit.GPIO28 = 1;   // uncomment if −−> Set High initially
260 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
261 //  GPIO−29 − PIN FUNCTION = SCI−TX
262   GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 0;  // 0=GPIO,  1=SCITXD−A,  2=XA19,  3=Resv
263   GpioCtrlRegs.GPADIR.bit.GPIO29 = 0;   // 1=OUTput,  0=INput
264 //  GpioDataRegs.GPACLEAR.bit.GPIO29 = 1; // uncomment if −−> Set Low initially
265 //  GpioDataRegs.GPASET.bit.GPIO29 = 1;   // uncomment if −−> Set High initially
266 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
267 //  GPIO−30 − PIN FUNCTION = −−Spare−−
268   GpioCtrlRegs.GPAMUX2.bit.GPIO30 = 0;  // 0=GPIO,  1=CANRX−A,  2=XA18,  3=Resv
269   GpioCtrlRegs.GPADIR.bit.GPIO30 = 0;   // 1=OUTput,  0=INput
270 //  GpioDataRegs.GPACLEAR.bit.GPIO30 = 1; // uncomment if −−> Set Low initially
271 //  GpioDataRegs.GPASET.bit.GPIO30 = 1;   // uncomment if −−> Set High initially
272 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
273 //  GPIO−31 − PIN FUNCTION = LED2 (for Release 1.1 and up F2833x controlCARDs)
274   GpioCtrlRegs.GPAMUX2.bit.GPIO31 = 0;  // 0=GPIO,  1=CANTX−A,  2=XA17,  3=Resv
275   GpioCtrlRegs.GPADIR.bit.GPIO31 = 0;   // 1=OUTput,  0=INput
276 //  GpioDataRegs.GPACLEAR.bit.GPIO31 = 1; // uncomment if −−> Set Low initially
277 //  GpioDataRegs.GPASET.bit.GPIO31 = 1;   // uncomment if −−> Set High initially
278 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

279 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
280 //  GPIO−32 − PIN FUNCTION = −−Spare−−
281   GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 0;  // 0=GPIO,  1=I2C−SDA,  2=SYNCI,  3=
```

```
        ADCSOCA
282   GpioCtrlRegs.GPBDIR.bit.GPIO32 = 0;   // 1=OUTput,  0=INput
283 //  GpioDataRegs.GPBCLEAR.bit.GPIO32 = 1; // uncomment if --> Set Low initially
284 //  GpioDataRegs.GPBSET.bit.GPIO32 = 1;   // uncomment if --> Set High initially
285 //
        --------------------------------------------------------------------
286 //  GPIO-33 - PIN FUNCTION = --Spare--
287   GpioCtrlRegs.GPBMUX1.bit.GPIO33 = 0;  // 0=GPIO,  1=I2C-SCL,  2=SYNCO,  3=
        ADCSOCB
288   GpioCtrlRegs.GPBDIR.bit.GPIO33 = 0;   // 1=OUTput,  0=INput
289 //  GpioDataRegs.GPBCLEAR.bit.GPIO33 = 1; // uncomment if --> Set Low initially
290 //  GpioDataRegs.GPBSET.bit.GPIO33 = 1;   // uncomment if --> Set High initially
291 //
        --------------------------------------------------------------------
292 //  GPIO-34 - PIN FUNCTION = LED3 (for Release 1.1 and up F2833x controlCARDs)
293   GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0;  // 0=GPIO,  1=ECAP1,  2=Resv,  3=Resv
294   GpioCtrlRegs.GPBDIR.bit.GPIO34 = 0;   // 1=OUTput,  0=INput
295 //  GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1; // uncomment if --> Set Low initially
296 //  GpioDataRegs.GPBSET.bit.GPIO34 = 1;   // uncomment if --> Set High initially
297 //
        --------------------------------------------------------------------

298 //
        --------------------------------------------------------------------
299 //  GPIO-35 - PIN FUNCTION = --Spare-- (SCI-TX on R1 F2833x controlCARD)
300   GpioCtrlRegs.GPBMUX1.bit.GPIO35 = 0;  // 0=GPIO,  1=SCIA-TX,  2=Resv,  3=Resv
301   GpioCtrlRegs.GPBDIR.bit.GPIO35 = 0;   // 1=OUTput,  0=INput
302 //  GpioDataRegs.GPBCLEAR.bit.GPIO35 = 1; // uncomment if --> Set Low initially
303 //  GpioDataRegs.GPBSET.bit.GPIO35 = 1;   // uncomment if --> Set High initially
304 //
        --------------------------------------------------------------------
305 //  GPIO-36 - PIN FUNCTION = --Spare-- (SCI-RX on R1 F2833x controlCARD)
306   GpioCtrlRegs.GPBMUX1.bit.GPIO36 = 0;  // 0=GPIO,  1=SCIA-RX,  2=Resv,  3=Resv
307   GpioCtrlRegs.GPBDIR.bit.GPIO36 = 0;   // 1=OUTput,  0=INput
308 //  GpioDataRegs.GPBCLEAR.bit.GPIO36 = 1; // uncomment if --> Set Low initially
309 //  GpioDataRegs.GPBSET.bit.GPIO36 = 1;   // uncomment if --> Set High initially
310 //
        --------------------------------------------------------------------
311 //  GPIO-38 - PIN FUNCTION = LED2 (for Release 1 F2833x controlCARDs)
312   GpioCtrlRegs.GPBMUX1.bit.GPIO38 = 0;  // 0=GPIO,  1=Resv,  2=Resv,  3=Resv
313   GpioCtrlRegs.GPBDIR.bit.GPIO38 = 0;   // 1=OUTput,  0=INput
314 //  GpioDataRegs.GPBCLEAR.bit.GPIO38 = 1; // uncomment if --> Set Low initially
315 //  GpioDataRegs.GPBSET.bit.GPIO38 = 1;   // uncomment if --> Set High initially
316 //
        --------------------------------------------------------------------
317 //  GPIO-39 - PIN FUNCTION = LED3 (for Release 1 F2833x controlCARDs)
318   GpioCtrlRegs.GPBMUX1.bit.GPIO39 = 0;  // 0=GPIO,  1=Resv,  2=XA16,  3=Resv
```

```
319  GpioCtrlRegs.GPBDIR.bit.GPIO39 = 0;   // 1=OUTput,  0=INput
320 //  GpioDataRegs.GPBCLEAR.bit.GPIO39 = 1; // uncomment if --> Set Low initially
321 //  GpioDataRegs.GPBSET.bit.GPIO39 = 1;   // uncomment if --> Set High initially
322 //
      ---------------------------------------------------------------------
323
324 //  GPIO-48 - PIN FUNCTION = --Spare--
325  GpioCtrlRegs.GPBMUX2.bit.GPIO48 = 0;  // 0=GPIO,  1=ECAP5,  2=XD31,  3=Resv
326  GpioCtrlRegs.GPBDIR.bit.GPIO48 = 0;   // 1=OUTput,  0=INput
327 //  GpioDataRegs.GPBCLEAR.bit.GPIO48 = 1; // uncomment if --> Set Low initially
328 //  GpioDataRegs.GPBSET.bit.GPIO48 = 1;   // uncomment if --> Set High initially
329 //
      ---------------------------------------------------------------------
330 //  GPIO-49 - PIN FUNCTION = --Spare--
331  GpioCtrlRegs.GPBMUX2.bit.GPIO49 = 0;  // 0=GPIO,  1=ECAP6,  2=XD30,  3=Resv
332  GpioCtrlRegs.GPBDIR.bit.GPIO49 = 0;   // 1=OUTput,  0=INput
333 //  GpioDataRegs.GPBCLEAR.bit.GPIO49 = 1; // uncomment if --> Set Low initially
334 //  GpioDataRegs.GPBSET.bit.GPIO49 = 1;   // uncomment if --> Set High initially
335 //
      ---------------------------------------------------------------------
336
337 //  GPIO-58 - PIN FUNCTION = --Spare--
338  GpioCtrlRegs.GPBMUX2.bit.GPIO58 = 0;  // 0=GPIO,  1=MCLKR-A,  2=XD21,  3=Resv
339  GpioCtrlRegs.GPBDIR.bit.GPIO58 = 0;   // 1=OUTput,  0=INput
340 //  GpioDataRegs.GPBCLEAR.bit.GPIO58 = 1; // uncomment if --> Set Low initially
341 //  GpioDataRegs.GPBSET.bit.GPIO58 = 1;   // uncomment if --> Set High initially
342 //
      ---------------------------------------------------------------------
343 //  GPIO-59 - PIN FUNCTION = --Spare--
344  GpioCtrlRegs.GPBMUX2.bit.GPIO59 = 0;  // 0=GPIO,  1=MFSR-A,  2=XD20,  3=Resv
345  GpioCtrlRegs.GPBDIR.bit.GPIO59 = 0;   // 1=OUTput,  0=INput
346 //  GpioDataRegs.GPBCLEAR.bit.GPIO59 = 1; // uncomment if --> Set Low initially
347 //  GpioDataRegs.GPBSET.bit.GPIO59 = 1;   // uncomment if --> Set High initially
348 //
      ---------------------------------------------------------------------
349 //  GPIO-60 - PIN FUNCTION = --Spare--
350  GpioCtrlRegs.GPBMUX2.bit.GPIO60 = 0;  // 0=GPIO,  1=MCLKR-B,  2=XD19,  3=Resv
351  GpioCtrlRegs.GPBDIR.bit.GPIO60 = 0;   // 1=OUTput,  0=INput
352 //  GpioDataRegs.GPBCLEAR.bit.GPIO60 = 1; // uncomment if --> Set Low initially
353 //  GpioDataRegs.GPBSET.bit.GPIO60 = 1;   // uncomment if --> Set High initially
354 //
      ---------------------------------------------------------------------
355 //  GPIO-61 - PIN FUNCTION = --Spare--
356  GpioCtrlRegs.GPBMUX2.bit.GPIO61 = 0;  // 0=GPIO,  1=MFSR-B,  2=XD18,  3=Resv
357  GpioCtrlRegs.GPBDIR.bit.GPIO61 = 0;   // 1=OUTput,  0=INput
358 //  GpioDataRegs.GPBCLEAR.bit.GPIO61 = 1; // uncomment if --> Set Low initially
359 //  GpioDataRegs.GPBSET.bit.GPIO61 = 1;   // uncomment if --> Set High initially
```

```
360 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
361 //  GPIO−62 − PIN FUNCTION = −−Spare−−
362   GpioCtrlRegs.GPBMUX2.bit.GPIO62 = 0;  // 0=GPIO,  1=SCIRX−C,  2=XD17,  3=Resv
363   GpioCtrlRegs.GPBDIR.bit.GPIO62 = 0;   // 1=OUTput,  0=INput
364 //  GpioDataRegs.GPBCLEAR.bit.GPIO62 = 1; // uncomment if −−> Set Low initially
365 //  GpioDataRegs.GPBSET.bit.GPIO62 = 1;   // uncomment if −−> Set High initially
366 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
367 //  GPIO−63 − PIN FUNCTION = −−Spare−−
368   GpioCtrlRegs.GPBMUX2.bit.GPIO63 = 0;  // 0=GPIO,  1=SCITX−C,  2=XD16,  3=Resv
369   GpioCtrlRegs.GPBDIR.bit.GPIO63 = 0;   // 1=OUTput,  0=INput
370 //  GpioDataRegs.GPBCLEAR.bit.GPIO63 = 1; // uncomment if −−> Set Low initially
371 //  GpioDataRegs.GPBSET.bit.GPIO63 = 1;   // uncomment if −−> Set High initially
372 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
373
374 //  GPIO−84 − PIN FUNCTION = −−Spare−−
375   GpioCtrlRegs.GPCMUX2.bit.GPIO84 = 0;  // 0=GPIO,  1=GPIO,  2=XA12,  3=Resv
376   GpioCtrlRegs.GPCDIR.bit.GPIO84 = 0;   // 1=OUTput,  0=INput
377 //  GpioDataRegs.GPCCLEAR.bit.GPIO84 = 1; // uncomment if −−> Set Low initially
378 //  GpioDataRegs.GPCSET.bit.GPIO84 = 1;   // uncomment if −−> Set High initially
379 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
380 //  GPIO−85 − PIN FUNCTION = −−Spare−−
381   GpioCtrlRegs.GPCMUX2.bit.GPIO85 = 0;  // 0=GPIO,  1=GPIO,  2=XA13,  3=Resv
382   GpioCtrlRegs.GPCDIR.bit.GPIO85 = 0;   // 1=OUTput,  0=INput
383 //  GpioDataRegs.GPCCLEAR.bit.GPIO85 = 1; // uncomment if −−> Set Low initially
384 //  GpioDataRegs.GPCSET.bit.GPIO85 = 1;   // uncomment if −−> Set High initially
385 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
386 //  GPIO−86 − PIN FUNCTION = −−Spare−−
387   GpioCtrlRegs.GPCMUX2.bit.GPIO86 = 0;  // 0=GPIO,  1=GPIO,  2=XA14,  3=Resv
388   GpioCtrlRegs.GPCDIR.bit.GPIO86 = 0;   // 1=OUTput,  0=INput
389 //  GpioDataRegs.GPCCLEAR.bit.GPIO86 = 1; // uncomment if −−> Set Low initially
390 //  GpioDataRegs.GPCSET.bit.GPIO86 = 1;   // uncomment if −−> Set High initially
391 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
392 //  GPIO−87 − PIN FUNCTION = −−Spare−−
393   GpioCtrlRegs.GPCMUX2.bit.GPIO87 = 0;  // 0=GPIO,  1=GPIO,  2=XA15,  3=Resv
394   GpioCtrlRegs.GPCDIR.bit.GPIO87 = 0;   // 1=OUTput,  0=INput
395 //  GpioDataRegs.GPCCLEAR.bit.GPIO87 = 1; // uncomment if −−> Set Low initially
396 //  GpioDataRegs.GPCSET.bit.GPIO87 = 1;   // uncomment if −−> Set High initially
397 //
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
398
```

```
399  EDIS; // Disable register access
400 }
401
402 //
        =====================================================================
403 // NOTA (PLEXIM):
404 // IN MOST APPLICATIONS THE FUNCTIONS AFTER THIS POINT CAN BE LEFT
        UNCHANGED
405 // THE USER NEED NOT REALLY UNDERSTAND THE BELOW CODE TO SUCCESSFULLY
        RUN THIS
406 // APPLICATION.
407 //
        =====================================================================
408
409 void WDogDisable(void)
410 {
411     EALLOW;
412     SysCtrlRegs.WDCR= 0x0068;
413     EDIS;
414 }
415
416 // This function initializes the PLLCR register.
417 //void InitPll(Uint16 val, Uint16 clkindiv)
418 void PLLset(Uint16 val)
419 {
420    volatile Uint16 iVol;
421
422    // Make sure the PLL is not running in limp mode
423    if (SysCtrlRegs.PLLSTS.bit.MCLKSTS != 0)
424    {
425       // Missing external clock has been detected
426       // Replace this line with a call to an appropriate
427       // SystemShutdown(); function.
428     while(1){
429       continue;
430     }
431       //asm("        ESTOP0");
432    }
433
434    // CLKINDIV MUST be 0 before PLLCR can be changed from
435    // 0x0000. It is set to 0 by an external reset XRSn
436
437    // Change the PLLCR
438    if (SysCtrlRegs.PLLCR.bit.DIV != val)
439    {
440
441       EALLOW;
442       // Before setting PLLCR turn off missing clock detect logic
443       SysCtrlRegs.PLLSTS.bit.MCLKOFF = 1;
444       SysCtrlRegs.PLLCR.bit.DIV = val;
445       EDIS;
```

```
446
447      // Optional: Wait for PLL to lock.
448      // During this time the CPU will switch to OSCCLK/2 until
449      // the PLL is stable.  Once the PLL is stable the CPU will
450      // switch to the new PLL value.
451      //
452      // This time-to-lock is monitored by a PLL lock counter.
453      //
454      // Code is not required to sit and wait for the PLL to lock.
455      // However, if the code does anything that is timing critical,
456      // and requires the correct clock be locked, then it is best to
457      // wait until this switching has completed.
458
459      // Wait for the PLL lock bit to be set.
460      // The watchdog should be disabled before this loop, or fed within
461      // the loop via ServiceDog().
462
463     // Uncomment to disable the watchdog
464      WDogDisable();
465
466      while(SysCtrlRegs.PLLSTS.bit.PLLLOCKS != 1){
467        // Uncomment to service the watchdog
468          // ServiceDog();
469      }
470
471      EALLOW;
472      SysCtrlRegs.PLLSTS.bit.MCLKOFF = 0;
473
474      EDIS;
475    }
476 }
477
478
479 // Esta función inicializa los registros de control PIE a un estado conocido
480 //
481 void PieCntlInit(void)
482 {
483    // Disable Interrupts at the CPU level:
484    DINT;
485
486    // Disable the PIE
487    PieCtrlRegs.PIECTRL.bit.ENPIE = 0;
488
489  // Clear all PIEIER registers:
490  PieCtrlRegs.PIEIER1.all = 0;
491  PieCtrlRegs.PIEIER2.all = 0;
492  PieCtrlRegs.PIEIER3.all = 0;
493  PieCtrlRegs.PIEIER4.all = 0;
494  PieCtrlRegs.PIEIER5.all = 0;
495  PieCtrlRegs.PIEIER6.all = 0;
496  PieCtrlRegs.PIEIER7.all = 0;
497  PieCtrlRegs.PIEIER8.all = 0;
498  PieCtrlRegs.PIEIER9.all = 0;
```

```
499   PieCtrlRegs.PIEIER10.all = 0;
500   PieCtrlRegs.PIEIER11.all = 0;
501   PieCtrlRegs.PIEIER12.all = 0;
502
503   // Clear all PIEIFR registers:
504   PieCtrlRegs.PIEIFR1.all = 0;
505   PieCtrlRegs.PIEIFR2.all = 0;
506   PieCtrlRegs.PIEIFR3.all = 0;
507   PieCtrlRegs.PIEIFR4.all = 0;
508   PieCtrlRegs.PIEIFR5.all = 0;
509   PieCtrlRegs.PIEIFR6.all = 0;
510   PieCtrlRegs.PIEIFR7.all = 0;
511   PieCtrlRegs.PIEIFR8.all = 0;
512   PieCtrlRegs.PIEIFR9.all = 0;
513   PieCtrlRegs.PIEIFR10.all = 0;
514   PieCtrlRegs.PIEIFR11.all = 0;
515   PieCtrlRegs.PIEIFR12.all = 0;
516 }
517
518
519 void PieVectTableInit(void)
520 {
521   int16 i;
522   Uint32 *Source = (void *) &ISR_ILLEGAL;
523   Uint32 *Dest = (void *) &PieVectTable;
524
525   EALLOW;
526   for(i=0; i < 128; i++)
527     *Dest++ = *Source;
528   EDIS;
529
530   // Enable the PIE Vector Table
531   PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
532 }
533
534 interrupt void ISR_ILLEGAL(void)   // Illegal operation TRAP
535 {
536   // Insert ISR Code here
537
538   // Next two lines for debug only to halt the processor here
539   // Remove after inserting ISR Code
540   asm("          ESTOP0");
541   for(;;);
542
543 }
544
545 //Inicializa los registro de control de la Flash, debe ser ejecutada fuera de la RAM
546 void InitFlash(void)
547 {
548    EALLOW;
549    //Habilitar Flash Pipeline modo, mejora el rendimiento del codigo ejecutado desde la FLASH
550    FlashRegs.FOPT.bit.ENPIPE = 1;
551
```

```
552   // Paged Waitstate
553   FlashRegs.FBANKWAIT.bit.PAGEWAIT = 3;
554
555   //Random Waitstate
556   FlashRegs.FBANKWAIT.bit.RANDWAIT = 3;
557
558   //Waitstate for the OTP
559   FlashRegs.FOTPWAIT.bit.OTPWAIT = 5;
560
561   FlashRegs.FSTDBYWAIT.bit.STDBYWAIT = 0x01FF;
562   FlashRegs.FACTIVEWAIT.bit.ACTIVEWAIT = 0x01FF;
563   EDIS;
564
565   //Frozar mediante flush para asegurar que la escritura en el último registro configurado antes de
        que vuelva la funcion
566
567   asm(" RPT #7 || NOP");
568 }
569
570
571 // Esta función copia los contenidos de la memoria especificada a otro destino
572 //
573 //  Uint16 *SourceAddr        Puntero a la primera palabra que se quiere mover
574 //                  SourceAddr < SourceEndAddr
575 //  Uint16* SourceEndAddr     Puntero a la última palabra que se quiere mover
576 //  Uint16* DestAddr          Puntero a la primera dirección de memoria del destino
577 void MemCopy(Uint16 *SourceAddr, Uint16* SourceEndAddr, Uint16* DestAddr)
578 {
579    while(SourceAddr < SourceEndAddr)
580    {
581       *DestAddr++ = *SourceAddr++;
582    }
583    return;
584 }
```

DSP2833x_GlobalVariableDefs.c

```
1 //#include "DSP2833x_Device.h"        // DSP280x Headerfile Include File
2 #include "PeripheralHeaderIncludes.h"     // DSP2833x Headerfile Include File
3
4 //
    -----------------------------------------------------------------------
5 // Define Global Peripheral Variables:
6 //
7 //-----------------------------------------
8 #ifdef __cplusplus
9 #pragma DATA_SECTION("AdcRegsFile")
10 #else
11 #pragma DATA_SECTION(AdcRegs,"AdcRegsFile");
12 #endif
13 volatile struct ADC_REGS AdcRegs;
```

```
14
15 //--------------------------------------
16 #ifdef __cplusplus
17 #pragma DATA_SECTION("AdcMirrorFile")
18 #else
19 #pragma DATA_SECTION(AdcMirror,"AdcMirrorFile");
20 #endif
21 volatile struct ADC_RESULT_MIRROR_REGS AdcMirror;
22
23 //--------------------------------------
24 #ifdef __cplusplus
25 #pragma DATA_SECTION("CpuTimer0RegsFile")
26 #else
27 #pragma DATA_SECTION(CpuTimer0Regs,"CpuTimer0RegsFile");
28 #endif
29 volatile struct CPUTIMER_REGS CpuTimer0Regs;
30
31 //--------------------------------------
32 #ifdef __cplusplus
33 #pragma DATA_SECTION("CpuTimer1RegsFile")
34 #else
35 #pragma DATA_SECTION(CpuTimer1Regs,"CpuTimer1RegsFile");
36 #endif
37 volatile struct CPUTIMER_REGS CpuTimer1Regs;
38
39
40 //--------------------------------------
41 #ifdef __cplusplus
42 #pragma DATA_SECTION("CpuTimer2RegsFile")
43 #else
44 #pragma DATA_SECTION(CpuTimer2Regs,"CpuTimer2RegsFile");
45 #endif
46 volatile struct CPUTIMER_REGS CpuTimer2Regs;
47
48
49 //--------------------------------------
50 #ifdef __cplusplus
51 #pragma DATA_SECTION("CsmPwlFile")
52 #else
53 #pragma DATA_SECTION(CsmPwl,"CsmPwlFile");
54 #endif
55 volatile struct CSM_PWL CsmPwl;
56
57 //--------------------------------------
58 #ifdef __cplusplus
59 #pragma DATA_SECTION("CsmRegsFile")
60 #else
61 #pragma DATA_SECTION(CsmRegs,"CsmRegsFile");
62 #endif
63 volatile struct CSM_REGS CsmRegs;
64
65
66
```

```
67 //---------------------------------------------------------------------------
68 #ifdef __cplusplus
69 #pragma DATA_SECTION("DevEmuRegsFile")
70 #else
71 #pragma DATA_SECTION(DevEmuRegs,"DevEmuRegsFile");
72 #endif
73 volatile struct DEV_EMU_REGS DevEmuRegs;
74
75 //---------------------------------------------------------------------------
76 #ifdef __cplusplus
77 #pragma DATA_SECTION("DmaRegsFile")
78 #else
79 #pragma DATA_SECTION(DmaRegs,"DmaRegsFile");
80 #endif
81 volatile struct DMA_REGS DmaRegs;
82
83
84 //---------------------------------------------------------------------------
85 #ifdef __cplusplus
86 #pragma DATA_SECTION("ECanaRegsFile")
87 #else
88 #pragma DATA_SECTION(ECanaRegs,"ECanaRegsFile");
89 #endif
90 volatile struct ECAN_REGS ECanaRegs;
91
92 //---------------------------------------------------------------------------
93 #ifdef __cplusplus
94 #pragma DATA_SECTION("ECanaMboxesFile")
95 #else
96 #pragma DATA_SECTION(ECanaMboxes,"ECanaMboxesFile");
97 #endif
98 volatile struct ECAN_MBOXES ECanaMboxes;
99
100 //---------------------------------------------------------------------------
101 #ifdef __cplusplus
102 #pragma DATA_SECTION("ECanaLAMRegsFile")
103 #else
104 #pragma DATA_SECTION(ECanaLAMRegs,"ECanaLAMRegsFile");
105 #endif
106 volatile struct LAM_REGS ECanaLAMRegs;
107
108 //---------------------------------------------------------------------------
109 #ifdef __cplusplus
110 #pragma DATA_SECTION("ECanaMOTSRegsFile")
111 #else
112 #pragma DATA_SECTION(ECanaMOTSRegs,"ECanaMOTSRegsFile");
113 #endif
114 volatile struct MOTS_REGS ECanaMOTSRegs;
115
116
117 //---------------------------------------------------------------------------
118 #ifdef __cplusplus
119 #pragma DATA_SECTION("ECanaMOTORegsFile")
```

```
120 #else
121 #pragma DATA_SECTION(ECanaMOTORegs,"ECanaMOTORegsFile");
122 #endif
123 volatile struct MOTO_REGS ECanaMOTORegs;
124
125
126 //---------------------------------------------------------------------------
127 #ifdef __cplusplus
128 #pragma DATA_SECTION("ECanbRegsFile")
129 #else
130 #pragma DATA_SECTION(ECanbRegs,"ECanbRegsFile");
131 #endif
132 volatile struct ECAN_REGS ECanbRegs;
133
134 //---------------------------------------------------------------------------
135 #ifdef __cplusplus
136 #pragma DATA_SECTION("ECanbMboxesFile")
137 #else
138 #pragma DATA_SECTION(ECanbMboxes,"ECanbMboxesFile");
139 #endif
140 volatile struct ECAN_MBOXES ECanbMboxes;
141
142 //---------------------------------------------------------------------------
143 #ifdef __cplusplus
144 #pragma DATA_SECTION("ECanbLAMRegsFile")
145 #else
146 #pragma DATA_SECTION(ECanbLAMRegs,"ECanbLAMRegsFile");
147 #endif
148 volatile struct LAM_REGS ECanbLAMRegs;
149
150 //---------------------------------------------------------------------------
151 #ifdef __cplusplus
152 #pragma DATA_SECTION("ECanbMOTSRegsFile")
153 #else
154 #pragma DATA_SECTION(ECanbMOTSRegs,"ECanbMOTSRegsFile");
155 #endif
156 volatile struct MOTS_REGS ECanbMOTSRegs;
157
158
159 //---------------------------------------------------------------------------
160 #ifdef __cplusplus
161 #pragma DATA_SECTION("ECanbMOTORegsFile")
162 #else
163 #pragma DATA_SECTION(ECanbMOTORegs,"ECanbMOTORegsFile");
164 #endif
165 volatile struct MOTO_REGS ECanbMOTORegs;
166
167 //---------------------------------------------------------------------------
168 #ifdef __cplusplus
169 #pragma DATA_SECTION("EPwm1RegsFile")
170 #else
171 #pragma DATA_SECTION(EPwm1Regs,"EPwm1RegsFile");
172 #endif
```

```
173 volatile struct EPWM_REGS EPwm1Regs;
174
175
176 //---------------------------------------------------------------------------
177 #ifdef __cplusplus
178 #pragma DATA_SECTION("EPwm2RegsFile")
179 #else
180 #pragma DATA_SECTION(EPwm2Regs,"EPwm2RegsFile");
181 #endif
182 volatile struct EPWM_REGS EPwm2Regs;
183
184 //---------------------------------------------------------------------------
185 #ifdef __cplusplus
186 #pragma DATA_SECTION("EPwm3RegsFile")
187 #else
188 #pragma DATA_SECTION(EPwm3Regs,"EPwm3RegsFile");
189 #endif
190 volatile struct EPWM_REGS EPwm3Regs;
191
192 //---------------------------------------------------------------------------
193 #ifdef __cplusplus
194 #pragma DATA_SECTION("EPwm4RegsFile")
195 #else
196 #pragma DATA_SECTION(EPwm4Regs,"EPwm4RegsFile");
197 #endif
198 volatile struct EPWM_REGS EPwm4Regs;
199
200 //---------------------------------------------------------------------------
201 #ifdef __cplusplus
202 #pragma DATA_SECTION("EPwm5RegsFile")
203 #else
204 #pragma DATA_SECTION(EPwm5Regs,"EPwm5RegsFile");
205 #endif
206 volatile struct EPWM_REGS EPwm5Regs;
207
208 //---------------------------------------------------------------------------
209 #ifdef __cplusplus
210 #pragma DATA_SECTION("EPwm6RegsFile")
211 #else
212 #pragma DATA_SECTION(EPwm6Regs,"EPwm6RegsFile");
213 #endif
214 volatile struct EPWM_REGS EPwm6Regs;
215
216
217 //---------------------------------------------------------------------------
218 #ifdef __cplusplus
219 #pragma DATA_SECTION("ECap1RegsFile")
220 #else
221 #pragma DATA_SECTION(ECap1Regs,"ECap1RegsFile");
222 #endif
223 volatile struct ECAP_REGS ECap1Regs;
224
225
```

```
226 //---------------------------------------
227 #ifdef __cplusplus
228 #pragma DATA_SECTION("ECap2RegsFile")
229 #else
230 #pragma DATA_SECTION(ECap2Regs,"ECap2RegsFile");
231 #endif
232 volatile struct ECAP_REGS ECap2Regs;
233
234 //---------------------------------------
235 #ifdef __cplusplus
236 #pragma DATA_SECTION("ECap3RegsFile")
237 #else
238 #pragma DATA_SECTION(ECap3Regs,"ECap3RegsFile");
239 #endif
240 volatile struct ECAP_REGS ECap3Regs;
241
242 //---------------------------------------
243 #ifdef __cplusplus
244 #pragma DATA_SECTION("ECap4RegsFile")
245 #else
246 #pragma DATA_SECTION(ECap4Regs,"ECap4RegsFile");
247 #endif
248 volatile struct ECAP_REGS ECap4Regs;
249
250 //---------------------------------------
251 #ifdef __cplusplus
252 #pragma DATA_SECTION("ECap5RegsFile")
253 #else
254 #pragma DATA_SECTION(ECap5Regs,"ECap5RegsFile");
255 #endif
256 volatile struct ECAP_REGS ECap5Regs;
257
258 //---------------------------------------
259 #ifdef __cplusplus
260 #pragma DATA_SECTION("ECap6RegsFile")
261 #else
262 #pragma DATA_SECTION(ECap6Regs,"ECap6RegsFile");
263 #endif
264 volatile struct ECAP_REGS ECap6Regs;
265
266 //---------------------------------------
267 #ifdef __cplusplus
268 #pragma DATA_SECTION("EQep1RegsFile")
269 #else
270 #pragma DATA_SECTION(EQep1Regs,"EQep1RegsFile");
271 #endif
272 volatile struct EQEP_REGS EQep1Regs;
273
274 //---------------------------------------
275 #ifdef __cplusplus
276 #pragma DATA_SECTION("EQep2RegsFile")
277 #else
278 #pragma DATA_SECTION(EQep2Regs,"EQep2RegsFile");
```

```
279 #endif
280 volatile struct EQEP_REGS EQep2Regs;
281
282 //----------------------------------------
283 #ifdef __cplusplus
284 #pragma DATA_SECTION("GpioCtrlRegsFile")
285 #else
286 #pragma DATA_SECTION(GpioCtrlRegs,"GpioCtrlRegsFile");
287 #endif
288 volatile struct GPIO_CTRL_REGS GpioCtrlRegs;
289
290 //----------------------------------------
291 #ifdef __cplusplus
292 #pragma DATA_SECTION("GpioDataRegsFile")
293 #else
294 #pragma DATA_SECTION(GpioDataRegs,"GpioDataRegsFile");
295 #endif
296 volatile struct GPIO_DATA_REGS GpioDataRegs;
297
298 //----------------------------------------
299 #ifdef __cplusplus
300 #pragma DATA_SECTION("GpioIntRegsFile")
301 #else
302 #pragma DATA_SECTION(GpioIntRegs,"GpioIntRegsFile");
303 #endif
304 volatile struct GPIO_INT_REGS GpioIntRegs;
305
306 //----------------------------------------
307 #ifdef __cplusplus
308 #pragma DATA_SECTION("I2caRegsFile")
309 #else
310 #pragma DATA_SECTION(I2caRegs,"I2caRegsFile");
311 #endif
312 volatile struct I2C_REGS I2caRegs;
313
314 //----------------------------------------
315 #ifdef __cplusplus
316 #pragma DATA_SECTION("McbspaRegsFile")
317 #else
318 #pragma DATA_SECTION(McbspaRegs,"McbspaRegsFile");
319 #endif
320 volatile struct MCBSP_REGS McbspaRegs;
321
322 //----------------------------------------
323 #ifdef __cplusplus
324 #pragma DATA_SECTION("McbspbRegsFile")
325 #else
326 #pragma DATA_SECTION(McbspbRegs,"McbspbRegsFile");
327 #endif
328 volatile struct MCBSP_REGS McbspbRegs;
329
330 //----------------------------------------
331 #ifdef __cplusplus
```

```
332 #pragma DATA_SECTION("PieCtrlRegsFile")
333 #else
334 #pragma DATA_SECTION(PieCtrlRegs,"PieCtrlRegsFile");
335 #endif
336 volatile struct PIE_CTRL_REGS PieCtrlRegs;
337
338 //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
339 #ifdef __cplusplus
340 #pragma DATA_SECTION("PieVectTableFile")
341 #else
342 #pragma DATA_SECTION(PieVectTable,"PieVectTableFile");
343 #endif
344 struct PIE_VECT_TABLE PieVectTable;
345
346 //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
347 #ifdef __cplusplus
348 #pragma DATA_SECTION("SciaRegsFile")
349 #else
350 #pragma DATA_SECTION(SciaRegs,"SciaRegsFile");
351 #endif
352 volatile struct SCI_REGS SciaRegs;
353
354 //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
355 #ifdef __cplusplus
356 #pragma DATA_SECTION("ScibRegsFile")
357 #else
358 #pragma DATA_SECTION(ScibRegs,"ScibRegsFile");
359 #endif
360 volatile struct SCI_REGS ScibRegs;
361
362 //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
363 #ifdef __cplusplus
364 #pragma DATA_SECTION("ScicRegsFile")
365 #else
366 #pragma DATA_SECTION(ScicRegs,"ScicRegsFile");
367 #endif
368 volatile struct SCI_REGS ScicRegs;
369
370
371 //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
372 #ifdef __cplusplus
373 #pragma DATA_SECTION("SpiaRegsFile")
374 #else
375 #pragma DATA_SECTION(SpiaRegs,"SpiaRegsFile");
376 #endif
377 volatile struct SPI_REGS SpiaRegs;
378
379 //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
380 #ifdef __cplusplus
381 #pragma DATA_SECTION("SysCtrlRegsFile")
382 #else
383 #pragma DATA_SECTION(SysCtrlRegs,"SysCtrlRegsFile");
384 #endif
```

```
385 volatile struct SYS_CTRL_REGS SysCtrlRegs;
386
387 //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
388 #ifdef __cplusplus
389 #pragma DATA_SECTION("FlashRegsFile")
390 #else
391 #pragma DATA_SECTION(FlashRegs,"FlashRegsFile");
392 #endif
393 volatile struct FLASH_REGS FlashRegs;
394
395 //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
396 #ifdef __cplusplus
397 #pragma DATA_SECTION("XIntruptRegsFile")
398 #else
399 #pragma DATA_SECTION(XIntruptRegs,"XIntruptRegsFile");
400 #endif
401 volatile struct XINTRUPT_REGS XIntruptRegs;
402
403 //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
404 #ifdef __cplusplus
405 #pragma DATA_SECTION("XintfRegsFile")
406 #else
407 #pragma DATA_SECTION(XintfRegs,"XintfRegsFile");
408 #endif
409 volatile struct XINTF_REGS XintfRegs;
410
411
412
413 //
    ===========================================================================
414 // End of file.
415 //
    ===========================================================================
```

io.c

```
1
2 #include "includes.h"
3
4 struct ANALOG_INPUTS AIn;
5 struct ADC_OPROBES AdcOvrProbes;
6
7 static void PowerupADC()
8 {
9 }
10
11 void InitADC()
12 {
13   PowerupADC();
14   AdcRegs.ADCMAXCONV.all = 0x0003;        // Setup 4 conv's on SEQ1
```

```
15  AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0; // Setup ADCINA0 as 1st SEQ1 conv.
16  AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x1; // Setup ADCINA1 as 2nd SEQ1 conv.
17  AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x2; // Setup ADCINA2 as 3rd SEQ1 conv.
18  AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x3; // Setup ADCINA3 as 4th SEQ1 conv.
19 }
20
21 void InitPWM(uint16_t aPeriod){
22  EALLOW;
23  SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1; // ePWM1
24  EDIS;
25
26
27  EPwm1Regs.TBPRD = aPeriod;
28  EPwm1Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
29  EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetrical mode
30  EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Master module
31  EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;
32  EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO; // Sync down-stream module
33  EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
34  EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
35  EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
36  EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
37  EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR; // set actions for EPWM1A
38  EPwm1Regs.AQCTLA.bit.CAD = AQ_SET;
39  EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable Dead-band module
40  EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active Hi complementary
41  EPwm1Regs.DBFED = 50; // FED = 50 TBCLKs
42  EPwm1Regs.DBRED = 50; // RED = 50 TBCLKs
43
44 }
45
46 void InitGPIO(){
47  EALLOW;
48
49  //  GPIO-34 - PIN FUNCTION = LED3 on controlCARD
50  GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0;  // 0=GPIO,  1=COMP2OUT,  2=Resv,  3=
      COMP3OUT
51  GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;   // 1=OUTput,  0=INput
52  GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1;
53
54  EDIS;
55 }
```

sci.c

```
1
2 #include "includes.h"
3
4 /*
5  * Initializes serial interface (SCI)
6  */
7 void SCIInit (uint16_t mode, // parity, bits, and stop-bit
```

```
8         uint32_t baudrate,  // baudrate
9         uint32_t clk // low-speed clk frequency
10        ){
11   int16_t brr;
12
13   asm(" eallow");
14
15   SysCtrlRegs.PCLKCR0.bit.SCIAENCLK=1;
16
17     SciaRegs.SCICTL1.all = 0;         // reset SCI
18     SciaRegs.SCICCR.all = mode;
19     SciaRegs.SCICTL1.all = 0x0013;    // enable TX, RX, Internal SCICLK,disable RX ERR,
       SLEEP, TXWAKE
20
21   brr = (Uint16)(clk /8l /baudrate) - 1;
22     SciaRegs.SCIHBAUD = 0xFF & (brr>>8);
23     SciaRegs.SCILBAUD = 0xFF & brr;
24
25 #ifdef USE_SCI_FIFO
26     // setup FIFO
27     SciaRegs.SCIFFTX.all=0xC000;
28     SciaRegs.SCIFFRX.all=0x0000;
29     SciaRegs.SCIFFCT.all=0x00;
30 #endif
31
32     SciaRegs.SCICTL1.all = 0x0033;          // relinquish SCI from Reset */
33 #ifdef USE_SCI_FIFO
34     SciaRegs.SCIFFTX.bit.TXFIFOXRESET=1;
35     SciaRegs.SCIFFRX.bit.RXFIFORESET=1;
36 #endif
37
38   /* enable pins */
39   GpioCtrlRegs.GPAPUD.bit.GPIO28 = 0;    // enable pull-up for GPIO28 (SCIRXDA)
40   GpioCtrlRegs.GPAPUD.bit.GPIO29 = 0;    // enable pull-up for GPIO29 (SCITXDA)
41   GpioCtrlRegs.GPAQSEL2.bit.GPIO28 = 3;  // asynch input GPIO28 (SCIRXDA)
42   GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 1;   // configure GPIO28 for SCIRXDA operation
43   GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 1;   // configure GPIO29 for SCITXDA operation
44
45   asm(" edis");
46 }
47
48 /*
49  * Receive character
50  */
51 int16_t SCIGetChar(void){
52 #ifdef USE_SCI_FIFO
53   if(SciaRegs.SCIFFRX.bit.RXFFST == 0){
54 #else
55   if(SciaRegs.SCIRXST.bit.RXRDY == 0){
56 #endif
57       // nothing there...
58         return(-1);
59     }else{
```

```
60     return((int16)SciaRegs.SCIRXBUF.all);
61     }
62 }
63
64 /*
65  * Transmit character
66  */
67 void SCIPutChar(char c){
68   // make sure transmit buffer is ready
69     while(SciaRegs.SCICTL2.bit.TXRDY == 0){
70     continue;
71     }
72
73   SciaRegs.SCITXBUF = (uint16_t)c;
74 }
75
76 /*
77  * Write string
78  */
79 void SCIWriteString(const char *s){
80   uint16_t sc;
81
82   sc = (char)(*s);
83   while(sc != 0){
84     SCIPutChar(sc);
85     s++;
86     sc = (char)(*s);
87   }
88 }
89
90 #ifndef USE_SCI_FIFO
91 #error SCIPoll() implementation assumes use of FIFO!
92 #endif
93 void SCIPoll()
94 {
95   while(SciaRegs.SCIFFRX.bit.RXFFST != 0)
96   {
97     // assuming that there will be a "break" when FIFO is empty
98     PIL_RA_serialIn((int16)SciaRegs.SCIRXBUF.all);
99   }
100
101   int16_t ch;
102   if(SciaRegs.SCICTL2.bit.TXRDY == 1){
103     if(PIL_RA_serialOut(&ch))
104     {
105       SciaRegs.SCITXBUF = ch;
106     }
107   }
108 }
```

pil_ctrl.c

```c
#include "includes.h"

void PilCallback(PIL_CtrlCallbackReq_t aCallbackReq)
{
 switch(aCallbackReq)
 {
   case  PIL_CLBK_ENTER_NORMAL_OPERATION_REQ:
     // enabling the hardware actuation (if desired by user)
     PIL_inhibitPilSimulation();
     EnableActuation();
     return;

   case PIL_CLBK_LEAVE_NORMAL_OPERATION_REQ:
     // stopping the hardware actuation or invoking a transfer to a safe system state
     DisableActuation();
     PIL_allowPilSimulation();
     return;

   case PIL_CLBK_INITIALIZE_SIMULATION:
     // resetting the controller variables
     ResetControl();
     return;

   case PIL_CLBK_TERMINATE_SIMULATION:
       //Termination Request
      PilInitCalibrations();
       break;

   case PIL_CLBK_STOP_TIMERS:
     // stopping relevant timers
     EALLOW;
     SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 0;
     SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 0;
     SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 0;
     EDIS;
     return;

   case PIL_CLBK_START_TIMERS:
     // starting relevant timers
     EALLOW;
     SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;
     SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 1;
     SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 1;
     EDIS;
     return;
  }
}
```

pil_symbols_c.c

```c
#include "pil.h"
#include "includes.h"
```

```
3 #include "pidq.h"
4 #include "pi.h"
5
6 // this file can be linked such that no Flash memory is consumed
7
8 #ifndef PIL_PREP_TOOL
9 #include "pil_symbols_c.inc" // will be automatically generated
10 #endif
11
12 #pragma RETAIN(PIL_D_Guid)
13 PIL_CONST_DEF(unsigned char, Guid[], CODE_GUID);
14
15 #pragma RETAIN(PIL_D_CompiledDate)
16 PIL_CONST_DEF(unsigned char, CompiledDate[], COMPILE_TIME_DATE_STR);
17
18 #pragma RETAIN(PIL_D_CompiledBy)
19 PIL_CONST_DEF(unsigned char, CompiledBy[], USER_NAME);
20
21 #pragma RETAIN(PIL_D_FrameworkVersion)
22 PIL_CONST_DEF(uint16_t, FrameworkVersion, PIL_FRAMEWORK_VERSION);
23
24 #pragma RETAIN(PIL_D_BaudRate)
25 PIL_CONST_DEF(uint32_t, BaudRate, BAUD_RATE);
26
27 #pragma RETAIN(PIL_D_StationAddress)
28 PIL_CONST_DEF(uint16_t, StationAddress, 0);
29
30 #pragma RETAIN(PIL_D_FirmwareDescription)
31 PIL_CONST_DEF(char, FirmwareDescription[], "FOC demo project");
```

pil_symbols_p.c

```
1 #include "pil.h"
2 #include "pi.h"
3 #include "includes.h"
4
5 // this file can be linked such that no Flash memory is consumed
6 #ifndef PIL_PREP_TOOL
7 #include "pil_symbols_p.inc" // will be automatically generated
8 #endif
9 // manual configurations
10
11 PIL_CONFIG_DEF(uint32_t, SysClk, SYSCLK_HZ);
12 PIL_CONFIG_DEF(uint32_t, PwmFrequency, PWM_HZ);
13 PIL_CONFIG_DEF(uint32_t, ControlFrequency, CONTROL_HZ);
14 PIL_CONFIG_DEF(uint16_t, ProcessorPartNumber, 28335);
```

Bibliografía

[1] Djamila Rekioua y Ernest Matagne *Optimization of Photovoltaic Power Systems*, pp. 1-17 y 113-145. Springer, 2012.

[2] Sergio Vázquez Pérez y Juan Manuel Carrasco Solís, *Apuntes de Integración de Energías Renovables*. Departamento de Ingeniería Electrónica de la Universidad de Sevilla, 2016.

[3] Plexim GmbH, *PLECS User Manual*. Plexim, Julio 2016.

[4] Plexim GmbH, *PIL User Manual*. Plexim, Julio 2016.

[5] David M.Alter, *Running an Application from Internal Flash Memory on the TMS320F28xxx DSP*. Application Report, Texas Instruments, Enero 2013.

[6] Texas Instruments, *TMS320x2833x, 2823x Enhanced Pulse Width Modulator (ePWM) Module* Reference Guide, Julio 2009.

[7] Texas Instruments, *TMS320x2833x Analog-to-Digital Converter (ADC) Module* Reference Guide, Julio 2009.

[8] Texas Instruments, *TMS320x2833x, 2823x System Control and Interrupts* Reference Guide, Julio 2009.

[9] Texas Instruments, *TMS320x2833x, 2823x Serial Communications Interface (SCI)* Reference Guide, Julio 2009.

[10] Sun Power Corporation, *305 SOLAR PANEL Exceptional Efficiency and Performance* Catálogo, Octubre 2007.

[11] Mariusz Malinowski, Jose I. León, y HaitHaM Abu-Rub, *Solar Photovoltaic and Thermal Energy Systems: Current Technology and Future Trends*, Artículo procedente del IEEE.

[12] Células fotovoltaicas, https://en.wikipedia.org/wiki/celulas_fotovoltaicas. Wikipedia, 2017.