Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Diseño y desarrollo del BMS de un monoplaza de Formula Student Electric

Autor: Germán Emilio Macías Gago
Tutor: Alfredo Pérez Vega-Leal

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

Sevilla, 2017

Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Diseño y desarrollo del BMS de un monoplaza de Formula Student Electric

Autor:
Germán Emilio Macías Gago

Tutor:
Alfredo Pérez Vega-Leal
Profesor contratado doctor

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Proyecto de Fin de Grado: Diseño y desarrollo del BMS de un monoplaza de Formula Student Electric

Autor: Germán Emilio Macías Gago

Tutor: Alfredo Pérez Vega-Leal

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mi familia, y en especial a mi madre María Dolores, quienes me han dado los valores de trabajo, dedicación y esfuerzo que hoy me hacen ser la persona que soy y me lo han facilitado todo para llegar hasta aquí, con inconmensurable paciencia y cariño.

A mi tutor Alfredo, por ofrecerme la oportunidad de trabajar junto a él en este proyecto, por dedicarme su tiempo y su constante atención.

Resumen

La Formula SAE o Formula Student es una competición a nivel universitario en la que 600 universidades de todo el mundo diseñan, desarrollan y construyen monoplazas de carreras para superar una serie de pruebas. Este documento recoge el desarrollo y prototipado del Battery Management System de la batería del futuro ART18e, el primer monoplaza eléctrico del equipo ARUS.

Pese a la existencia de soluciones comerciales que cumplen los requisitos establecidos por la competición, su elevado costo y el afán de aprendizaje de nuevas tecnologías existente en el equipo son motivos suficientes para optar por el desarrollo propio en los laboratorios de la Universidad de Sevilla.

El proyecto trata de la forma más estructurada posible los subsistemas y funciones que comprenden este BMS, tales como monitorización de voltaje, temperatura, corriente y balanceo, ajustándose a los requisitos exigidos por el monoplaza y la normativa de FSAEE.

Partiendo desde una breve introducción de las tecnologías que envuelven a las celdas de ion litio y su caracterización se llega a la fabricación de cada PCB que conforma el proyecto, pasando por la definición de la topología, la elección de cada componente, el diseño de cada parte y el desarrollo hardware que lo acompaña, sin olvidar la interfaz de comunicaciones seguida con cada uno de los numerosos elementos.

Se recoge aquí la materialización de muchos meses de trabajo y estudio de tecnologías completamente nuevas para mí y para el equipo, y sienta el precedente de la que a partir de ahora será una tradición en ARUS.

Abstract

Formula SAE, also known as Formula Student is a student engineering competition in which 600 universities around the world design, develop and build a formula style racing car to overcome a series of tests. This document gather the development and prototyping phase of the Battery Management System of the future ART18e, the first electric car of the ARUS team.

Although commercial solutions which satisfies the requirements of the competition can be found, its elevated cost and the knowledge persecution of the team are reasons enough to choose the self-development option in the laboratories of the University of Seville.

This project describes the subsystems and functions of this BMS, like voltage monitoring, temperature monitoring, current monitoring and cell balancing adjusting to the car topology and FSAEE rules requirements.

Starting of a brief introducction to the lithium ion cells and his characteristics, we reach to each PCB manufacturing covered in this project, passing by the topology definition, each component choice, each subsystem desing and the hardware developmente attached to it, without forgetting the communication interface of each element.

Many months of work are here reflected while completelly unknown technologies for me and the team have been studied, so this project should set a precedent and a new tradition of the team, the self developed BMS.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

AD	Amplificador Diferencial
ADC	Convertidor analógico-digital
AIR	Relé electromagnético de aislamiento
BMS	Sistema de gestión de baterías
CAN	Controller Area Network
CCS	Code Composer Studio
CS	Chip Select
EMI	Interferencia electromagnética
FSAE	Formula Student
FSAEE	Formula Student Electric
HV	Alto Voltaje >60VDC
IC	Circuito Integrado
LED	Diodo Emisor de Luz
LV	Bajo Voltaje <60VDC
MISO	Master Imput Slave Output
MOSI	Master Output Slave Imput
PCB	Placa de circuito impreso
PTC	Posistor
SCLK	Señal de reloj
SOC	Estado de carga
SPE	Sistema de Propulsión Eléctrica
SPI	Serial Peripheral Interface
UART	Transmisor-Receptor Asíncrono Universal
µC	Microcontrolador

1 INTRODUCCIÓN

EL reciente interés en los vehículos urbanos eléctricos e híbridos y el auge por la movilidad eléctrica han provocado un grandísimo desarrollo de las tecnologías electrónicas en el ámbito del automovilismo, consolidando el desarrollo del Sistema de Propulsión Eléctrica como el eje central del área. Tanto es así, que han nacido dos categorías oficiales de competición con monoplazas tipo Formula centradas principalmente en el desarrollo del SPE, la Formula E y la Formula Student Electric.

El BMS, o Battery Management System, es una parte fundamental en el Sistema de Propulsión Eléctrica de un coche, debe asegurar la correcta gestión de la energía contenida en la batería hacia el resto del vehículo, no dejando en ningún momento que la batería opere en un rango de funcionamiento inseguro.

Aunque el desarrollo del BMS sea algo anterior a la llegada del vehículo eléctrico, el problema a resolver en el interior de un monoplaza de competición cambia por completo las especificaciones de lo conocido hasta ahora y exige partir casi desde cero en la resolución del problema.

En este proyecto se va a proceder al diseño desde cero de un BMS para la batería del futuro monoplaza del equipo ARUS de la Universidad de Sevilla.

1.1 Motivaciones

La construcción de un monoplaza eléctrico es gran reto tecnológico a manos de la Ingeniería Eléctrica y Electrónica, y dado el interés por parte del equipo ARUS en participar en la Formula Student Electric y el gran desarrollo en tecnologías eléctricas existente en la propia Universidad de Sevilla, se tomó la decisión de desarrollar propiamente el máximo de sistemas posibles del vehículo, sin otro objetivo mas que la adquisición de conocimientos de los miembros y profesores implicados y el aprendizaje de nuevas tecnologías relacionadas con el automovilismo. Así pues, dado el elevado coste de las soluciones comerciales válidas para la batería del monoplaza, se optó desde un comienzo por el diseño propio del BMS, otorgándonos un infinito conocimiento sobre nuestro propio sistema, una mayor versatilidad y un menor coste.

El desarrollo de un proyecto de esta magnitud me ha otorgado un inmenso conocimiento práctico que no ha sido posible adquirir mediante el grado, que complementa a la perfección mi formación como futuro ingeniero.

1.2 Alcance del Proyecto

El BMS desarrollado en este proyecto trata de cumplir las especificaciones exigidas por la FSAEE adaptado a la topología del coche. No contiene en sí mismo las mismas especificaciones del BMS de una batería cualquiera.

El aquí propuesto cuenta con:

- Medición de voltaje de cada celda
- Medición de temperatura de las celdas
- Medición de corriente de salida o entrada a la batería

- Balanceo pasivo
- Estimación del SOC
- Capacidad de comunicación CAN con el resto del vehículo
- Aislamiento eléctrico respecto al sistema de LV del vehículo

Así mismo, y como contempla la normativa de FSAEE, el BMS debe ser el encargado de transmitir la orden de permitir el flujo de corriente de la batería a la gestión de potencia del vehículo en caso de situación peligrosa para la misma, el poder de corte recae en unos relés electromecánicos que no forman parte del BMS.

El objetivo de este proyecto ha sido el desarrollo de una base hardware y unas funciones mínimas del BMS exigidas por la normativa de FSAEE, con opciones a seguir un desarrollo software más completo y con más funcionalidades durante el restante año de evolución.

1.3 Topología del sistema

Existen diferentes formas de disponer los circuitos de medida, descarga y procesado necesarios para implementar un sistema de gestión de baterías, pero básicamente pueden clasificarse en tres las topologías de un BMS: centralizado, modular y distribuido.

En un BMS centralizado todos los circuitos pertenecen a la misma PCB, de la que salen los cables a cada celda, requiere de una gran PCB si el tamaño de la batería es considerable y es la peor opción en términos de organización del cableado.

Un BMS modular es similar en concepto a un BMS centralizado, pero subdivido en distintos módulos más pequeños, que permiten una mejor organización. Puede tener una jerarquía *Master-Slave* o los módulos pueden ser independientes entre sí.

En un BMS distribuido cada celda tiene su propia PCB para la medida de voltaje y temperatura, con su circuito de balanceo y comunicaciones. Es la topología más eficiente en organización y cableado, pero eleva el coste y la complejidad sea cual sea el número de celdas.

La topología elegida es la de BMS modular, es decir, un *Master* central que gobierna a 8 *Slaves* o módulos. Cada uno de estos módulos está asociado a cada uno de los segmentos o *stacks* de la batería.

Cada *stack* está compuesto por 16 celdas de ion-litio en serie, por lo que obviamente cada BMS *Slave* debe tener dicha capacidad. El conjunto de la batería está considerado por los 8 *stacks* en serie, con un total de 128 celdas en serie.

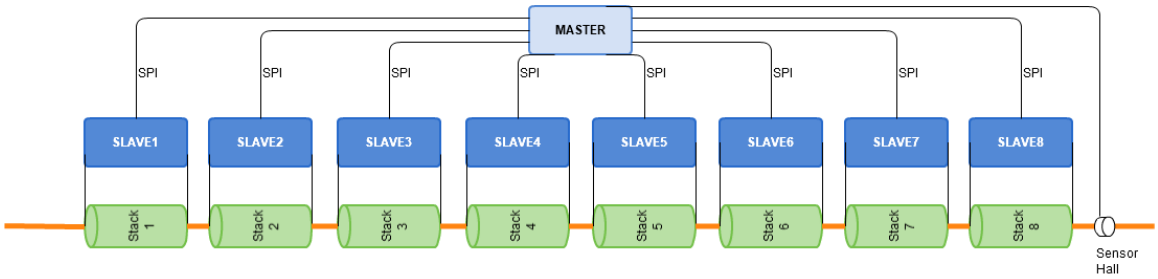

Figura 1-1. Topología del BMS

El BMS *Master* recopila la información de cada *Slave* y tiene un conocimiento general de toda la batería, así como la temperatura y tensión de cada celda. La comunicación entre *Master* y *Slaves* se realiza por SPI, que se explicará de una forma más detalla en el apartado 5 de comunicaciones. Como se puede observar, la detección de corriente es también tarea del *Master*, con un sensor de efecto Hall en el terminal negativo de la batería. El siguiente diagrama de bloques de la imagen recoge las funciones y su distribución en cada PCB.

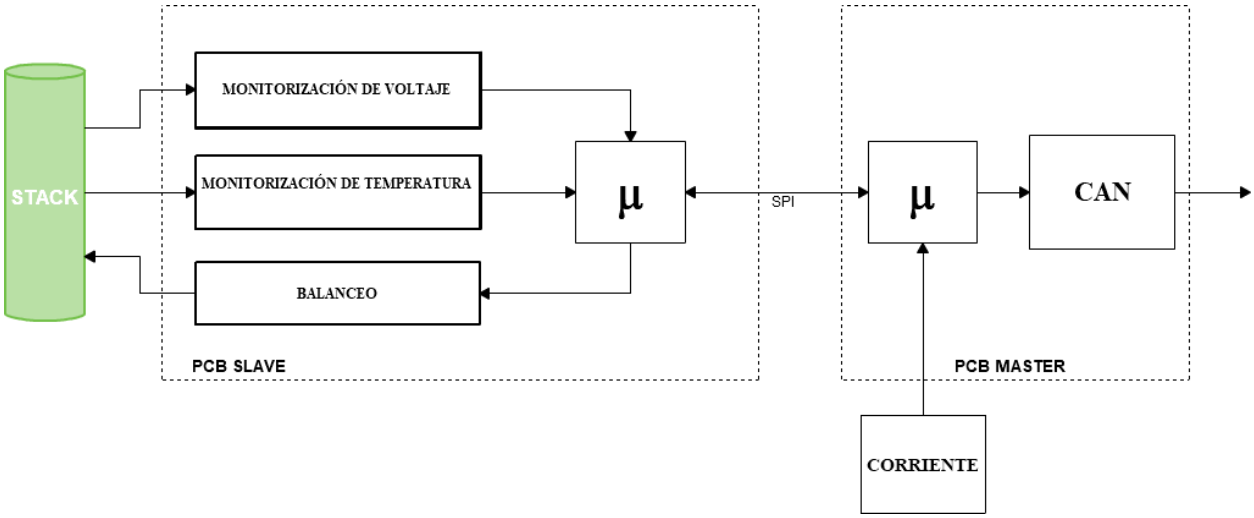

Figura 1-2. Diagrama de bloques de las PCBs

2 CELDAS DE ION-LITIO

PAra entender la necesidad de un BMS en una batería es necesario conocer el carácter sensible de las celdas de ion-litio y las consecuencias que tiene llevar las celdas a sus respectivos límites. En este apartado se explicará brevemente lo más importante para entender el comportamiento de una celda de ion-litio y los parámetros con los que puede quedar caracterizada.

2.1 Contexto tecnológico

Una batería de ion-litio es un tipo de batería recargable en la que los iones de litio se desplazan desde el electrodo negativo al electrodo positivo durante la descarga, y vuelven durante la carga.

Las celdas de ion-litio poseen una serie de propiedas tales como alta densidad energética, ligereza en sus componentes, alta tasa de descarga, mínimo efecto memoria y alto rendimiento. Sin embargo, su rápida degradación y sensibilidad a las elevadas temperaturas, pueden resultar en su destrucción por inflamación o incluso explosión.

Estas celdas pueden ser sencillamente modeladas como una fuente de tensión real, es decir, una fuente de tensión ideal en serie con una resistencia interna, que suele ser del orden de varios mΩ.

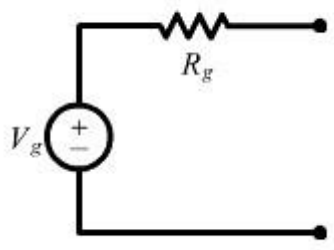

FIgura 2-1. Modelo de una celda

2.1.1 Temperatura

Una celda de ion-litio requiere operar en un rango de temperatura preciso, comúnmente recomendado por los fabricantes entre 0ºC y 60ºC. El mayor problema para operar en tal rango es el sobrecalentamiento producido por las mismas celdas durante su propio uso, dado que cuanta más carga se le exige a las mismas, mayor corriente pasa por la resistencia interna de la celda, que se traduce en mayor calor según la expresión del efecto Joule.

$$Q = I^2 \cdot R \cdot t$$

Donde Q es el calor, I la intensidad que circula por la celda, R su resistencia interna, y t el tiempo.

Los efectos de la temperatura en una celda pueden ser variados, desde una pequeña pérdida de rendimiento y capacidad, hasta una cadena de reacciones químicas incontrolables en el interior que producen una acumulación de gases provocando la ignición y a la explosion de la propia celda, poniendo en serio peligro el resto del conjunto si conviven más de una.

Por lo tanto, el BMS debe estar monitorizando continuamente la temperatura de las celdas críticas y mandar una señal en caso de ser necesario al circuito de refrigeración de la batería. Si la temperatura de cualquier celda excede los 57 grados centígrados, se mandará una señal al circuito de gestión de potencia que permitirá abrir los relés de seguridad, previo aviso al piloto.

2.1.2 Voltaje

La tension que proporciona una celda de ion-litio está caracterizada por tres valores; tension máxima (Vmax), tensión mínima (Vmin) y tensión nominal (Vnom). Al contrario que las fuentes de tensión ideales, la tensión de una celda es variable según su SOC, asímismo, también lo es según la intensidad que se extraiga de la misma, lo que significa que su tensión en circuito abierto no coincide con la tension sometida a una carga, debido a la caida de tensión que se experimenta en la resistencia interna. En la siguiente imagen se puede observar de una forma detallada (aunque algo exagerado) dicho comportamiento ante una intensidad constante de descarga.

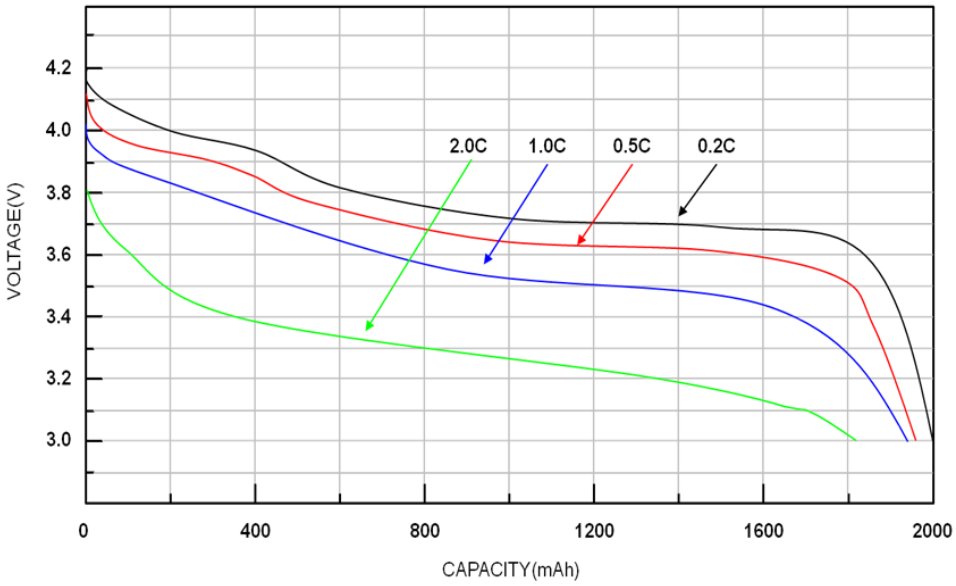

Figura 2-2. Curvas de descarga de una celda de ion-litio

Matizar que C es la unidad que se utiliza para medir la carga o descarga de una celda, significando 1C la intensidad necesaria para descargar una celda en una hora, es decir, si una celda tiene 2300mAh de capacidad[1], que descargue o cargue a 1C significa que la intensidad que cede o entra en la celda son 2300mA, si lo hace a 2C, la intensidad será de 4600mA.

Como se observa en la gráfica, las curvas de tensión tienen distinto comportamiento según la intensidad de descarga, cuanto menor sea la intensidad, más horizontal es la curva (más tiempo se mantiene la tensión cerca de Vnom), más capacidad se aprovecha de la batería, mejora el mantenimiento de la propia celda a la larga y más eficiente es la tranmisión de energía, disipando menos calor. Cuando la intensidad de descarga es mayor, aumenta la caida de tensión en la resistencia interna, quedando desaprovechada una parte de la tensión de la propia celda y alcanzando antes su tensión minima, lo que provoca una pérdida de la capacidad.

En la tensión máxima y minima de la celda reside la importancia del BMS. Una celda de ion-litio suele estar caracterizada entre 2.5V y 4.2V, siendo por seguridad no revasables ambos límites. Si una celda entra en lo que se conoce como *undervoltage* o *overvoltage* aumenta la temperatura de la misma exponencialmente perdiendo capacidades con daño permanente, y como se ha descrito anteriormente,

[1] Se define capacidad de una celda como el valor máximo de carga que puede almacenar. Se mide en amperios hora (Ah).

un aumento excesivo de la temperatura puede producir inflamación e ignición en la celda. Es por eso por lo que el BMS debe evitar que la tensión de cualquier celda supere los 4.2V y disminuya de 2.5V, aunque en ocasiones por seguridad se deja un pequeño margen inferior que ayuda a alargar la vida de las celdas, como se observa en la Figura 2-2, se detiene la descarga a 3.0V.

2.1.3 Encapsulado

El encapsulado de una celda es el empaquetado o el recubrimiento mecánico que la contiene. Las de ion-litio se fabrican por lo general en tres encapsulados diferentes: Semiflexible o Pouch, Cilíndrico y Prismático.

Figura 2-3. Encapsulado Pouch, Cilíndrico y Prismático

En las celdas tipo Pouch es común encontrarse la densidad energética más alta en comparación al resto, no disponen de recubrimiento metálico y son fácilmente perforables. Por otro lado, las celdas Cilíndricas suelen ofrecer una tasa de descarga muy alta con peor densidad salvo contados casos, además disponen de varios empaquetados estandarizados como 18650 o 26650 y las recubre una carcasa metálica. Las celdas prismáticas suelen ser algo pobres en densidad energética y en tasa de descarga, pero suelen tener capacidades muy altas. Son de mayor tamaño que las demás y tienen un recubrimiento más compacto y seguro.

Aunque las características de las celdas no tienen por qué depender de su empaquetado, esto es simplemente una convención seguida por muchos fabricantes.

2.1.4 Energía

Despreciando el empaquetado de una celda, para un mismo tipo de reactivos la capacidad depende del volumen de los mismos, así pues, para dos celdas con la misma densidad energética, lógicamente una con mayor volumen tendrá mayor capacidad.

La energía almacenada en una celda se mide en Wh y se calcula mediante el producto de la tension en circuito abierto de la misma por su capacidad en Ah. La densidad energética se obtiene del cociente entre su energía almacenada en Wh y el volumen que ocupa en m^3.

Los dispositivos de almacenamiento de energía eléctrica se enfrentan a un compromiso entre la máxima potencia extraíble (alta tasa de descarga) y la capacidad de los mismos. En el caso de elegir una celda de alta capacidad, la máxima corriente extraible de esa celda sería baja y viceversa. Dicho compromiso se muestra en el diagrama de Ragone.

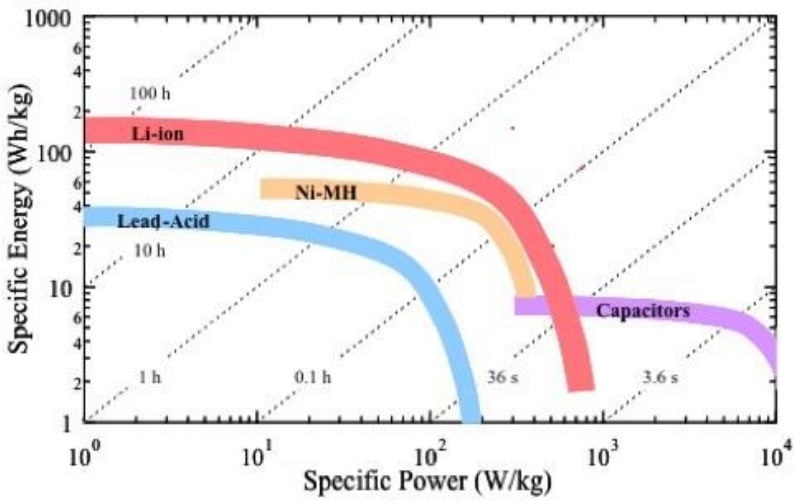

Figura 2-4. Diagrama de Ragone

En las tecnologías de vehículos eléctricos y gracias a las propiedades del ion-litio, la elección de celdas se produce para que se cumplan al máximo los dos requisitos, eligiéndose comúnmente las celdas comprendidas en la esquina de la curva roja.

2.1.5 Proceso de carga

La carga de una celda de ion-litio se realiza en dos etapas. En la primera etapa, el cargador de la batería proporciona una intensidad constante, mientras la tensión en los terminales va aumentando conforme aumenta la carga. Esta etapa concluye cuando se alcanza el valor máximo de tensión. Si la intensidad de la carga es mayor, el proceso es más rápido, pero más ineficiente (mayor disipación en la Rin y mayor temperatura). La tensión máxima de las celdas se ha alcanzado al finalizar esta etapa debido a la caída de tensión en la resistencia interna, sin embargo, las tensiones en circuito abierto no han llegado a su valor máximo.

En la segunda etapa, el cargador mantiene la tensión del valor máximo permitido constante y disminiye la intensidad pausadamente, partiendo del valor dado en la primera etapa. Bajando la intensidad la caída de tensión de la resistencia disminuye, por lo que poco a poco nos aproximamos a la carga total de cada celda en circuito abierto. El proceso concluye cuando la intensidad baja un cierto umbral, debido a que la curva que describe es una exponencial negativa y el proceso podría tardar indefinidamente.

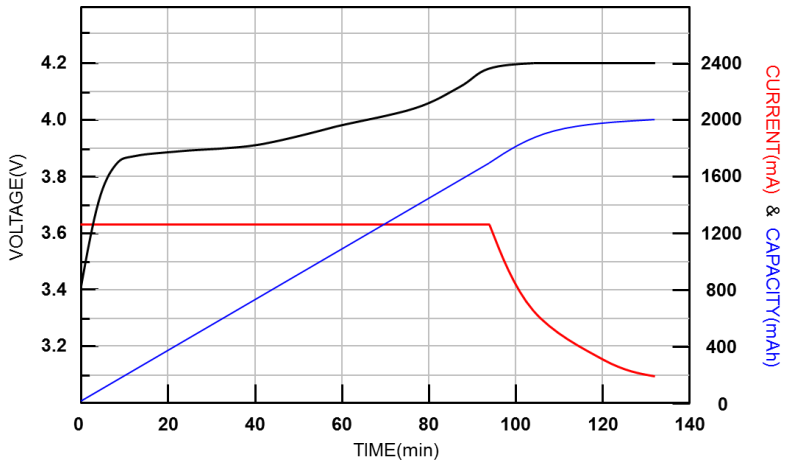

Figura 2-5. Ejemplo del proceso de carga de una celda de ion-litio

2.2 Especificaciones

En nuestro caso, y en base al prediseño de la batería, las celdas usadas serán cilíndricas, concretamente del estándar 18650. Mediante la agrupación de grupos de 6 celdas en paralelo se consiguie la corriente de descarga y capacidad necesaria, y se disponen 128 de estas agrupaciones en serie para conseguir el voltaje necesario, comprendiendo un total de 768 celdas.

Las celdas elegidas en el prediseño son las Samsung 25R, con 2.5Ah de capacidad y casi 50A de intensidad máxima de descarga.

Para monitorizar los 128 módulos de 6p[2] celdas, se dividen en 8 stacks de 16 celdas en serie cada uno, como se ha especificado en el apartado 1.3. El BMS considera cada uno de los módulos de 6p como una única celda (al estar en paralelo todas comparten la misma tensión).

Dado que la competición obliga a monitorizar la temperatura de una de cada tres celdas como mínimo, se dispondrán dos sensores de temperatura por cada modulo 6p, monitorizando el conjunto de la segunda y tercera celda y el conjunto de la cuarta y quinta celda.

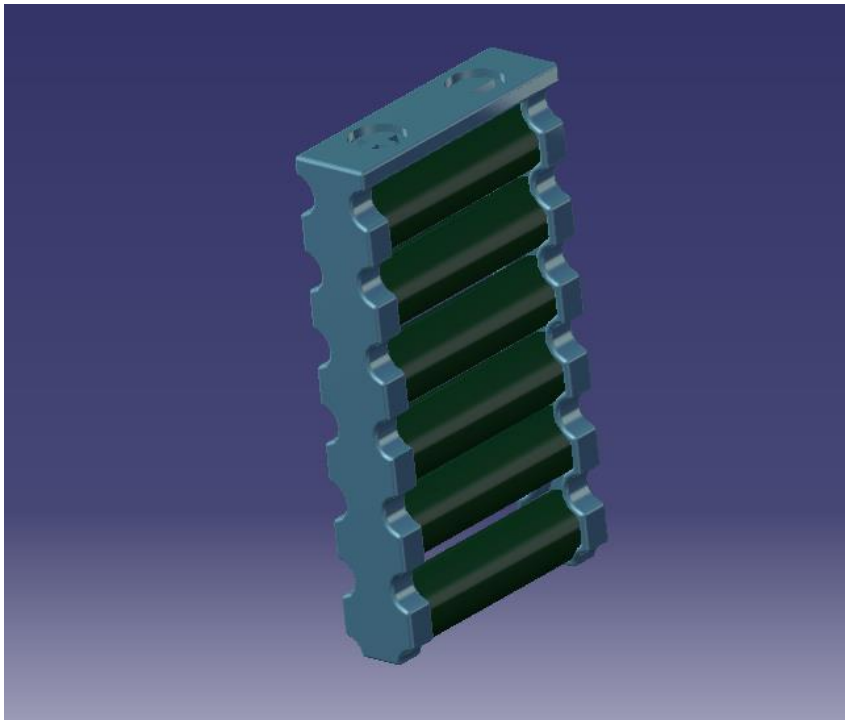

Figura 2-6. Modelo 3D de un módulo 6p

[2] La nomenclatura usada en las configuraciones de baterías consiste en el número de celdas seguidas según su disposición. Por ejemplo, una batería con 10 celdas en serie y 5 en paralelo se nombra como 10s5p.

3 GESTIÓN DE BATERÍAS

LA gestión de baterías conocida como *Battery Management* en inglés, se compone de dos partes, la monitorización y el balanceo. En este apartado se explicará detalladamente cómo se ha resuelto cada uno de estos problemas.

3.1 Monitorización de baterías

3.1.1 Criterios de diseño

El objetivo del sistema de monitorización es satisfacer los requisitos de la FSAEE en términos de medida de voltaje y temperatura, como se desarrollará más adelante en los puntos 3.2.1 y 3.2.2. El voltaje debe ser monitorizado precisa y continuamente, lo que deja una interpretación bastante abierta. Por otro lado, en la temperatura especifica que la medida debe ser realizada en el terminal negativo de cada celda, o a no más de un centímetro del mismo, y que mínimo una de cada tres celdas debe ser monitorizadas. Se ha tomado como objetivo adicional la mayor semejanza posible con las soluciones comerciales comúnmente usadas por los equipos, teniendo una mayor flexibilidad al ser nuestro propio sistema.

Tabla 3–1. Resumen de especificaciones

Especifiación	Objetivo
Rango de entrada de tensión por celda	De 0V a 5V
Precisión de medida de tensión	50mV
Tiempo de lectura de voltaje por celda	200µs
Sensores de voltaje	1 por celda
Rango de medición de Temperatura	0ºC a 70ºC
Sensores de Temperatura	Hasta 2 de cada 3 celdas

La medida de la corriente entrante o saliente del conjunto de la batería es también otra parte del sistema de monitorización y aunque no estén impuestos sus criterios de diseño por la FSAEE, el objetivo a cumplir ha sido el que cubra el rango necesario de amperaje con cierto margen de seguridad, además de obviamente una alta precisión en la medida que nos permita una estimación de la potencia y del SOC de mayor calidad.

3.1.2 Monitorización de voltaje

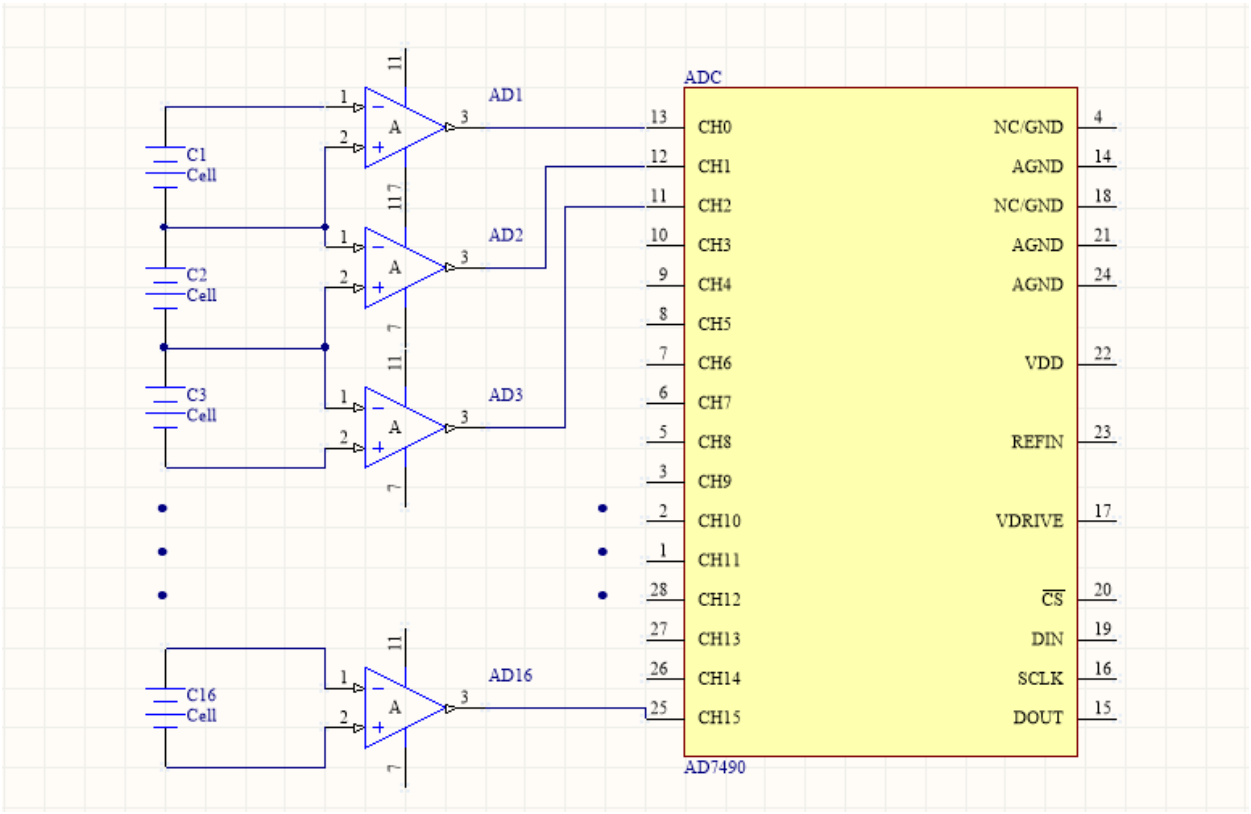

Figura 3-1. Esquema Monitorización de Voltaje

Para la medida en cada *Slave* de la tensión de cada una de las 16 celdas que hay en serie, se ha seguido el esquema de la imagen anterior. Pese a existir ICs que se encargan de la monitorización del voltaje de todas celdas, el número de celdas de entrada es fijo y menor a 16 en la mayoría de los casos, así que por versatilidad y por la aplicación de conocimientos adquiridos en Instrumentación Electrónica se ha optado por el uso de Amplificadores Diferenciales.

El modelo usado ha sido el LT1990A, que pese a no ser la opcion de menor coste, proporciona gran precisión y fiabilidad, estando preparado para la monitorización de celdas y para aplicaciones de automoción.

El LT1990 o LT1990A (variante con mayor rechazo a modo común) tiene una impedancia de entrada de 2MΩ en modo diferencial, es decir, que la corriente del efecto de carga es totalmente despreciable. Permite un ancho rango de alimentación, de entre 2.7V y 36V incluyendo la posibilidad de alimentación simétrica, aunque no vaya a ser usada en nuestro caso. Ambos modelos proporcionan alto rechazo al modo común, 70db mínimo de CMRR.

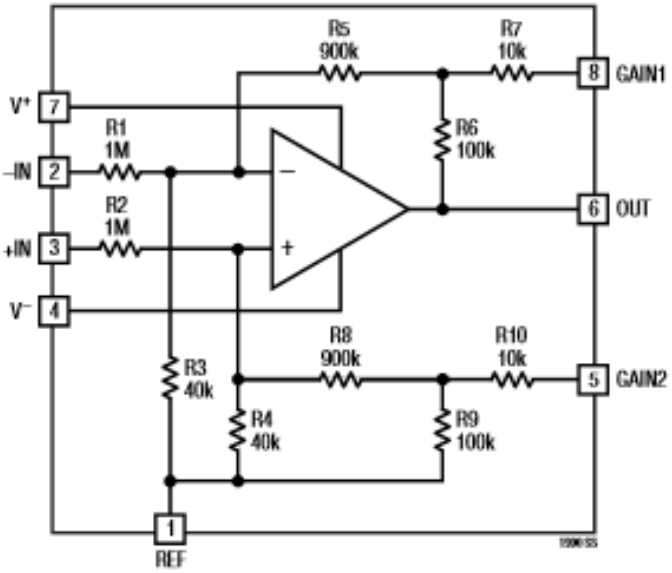

Figura 3-2. Diagrama interno LT1990

La ganancia es configurable mediante los pines GAIN1 y GAIN2, si los pines permanecen desconectados obtendremos una ganancia G=1, si ambos se conectan a V_{REF} entonces G=10. En nuestro caso, y debido a que la alimentación que se le proporciona es de 5V y el rango de tensiones de medida estará entre 2.5V y 4.2V, evidentemente no tiene sentido escoger la ganancia G=10, por la saturación a 5V que se produciría en la salida.

La ecuación que gobierna la tensión de salida es la siguiente:

$$V_{OUT} = G \cdot (VIN^{+} - VIN^{-}) + V_{REF}$$

El AD nos obliga a usar una referencia que no puede coincidir con V^{-}-, es decir, no es posible el uso del rango de medición completo, porque Vref no puede ser 0V. Como se ve en la ecuación anterior, siendo Vout como máximo V^{+}, y tomando como ganancia G=1, si despejamos la entrada diferencial de tensión queda:

$$\Delta VIN = (VIN^{+} - VIN^{-}) = Vout - Vref$$

Con lo cual todo lo que se aumente Vref se disminuye del rango de la tensión de entrada. Como se comentó en el apartado 2.1, la tensión de una celda de ion-litio oscila entre 4.2V y 2.5V, por lo que usando una referencia de 0.5V seríamos capaces de medir tensiones entre 0 y 4.5V. La razón del uso de Vref = 0.5V es experimental, puesto que el propio fabricante para el rango de alimentación usado recomienda Vref=1.25V [1], sin embargo, mediante la prueba en el laboratorio se ha comprobado que el resultado con la referencia elegida es igual de aceptable, permitiéndonos tener un rango válido de tensiones, aunque no se haya cumplido el objetivo inicial impuesto.

El encapsulado que nos proporciona Linear Technologies es SO-8, de reducido tamaño y sencillo de enrutar, con el pinout ilustrado en la siguiente figura.

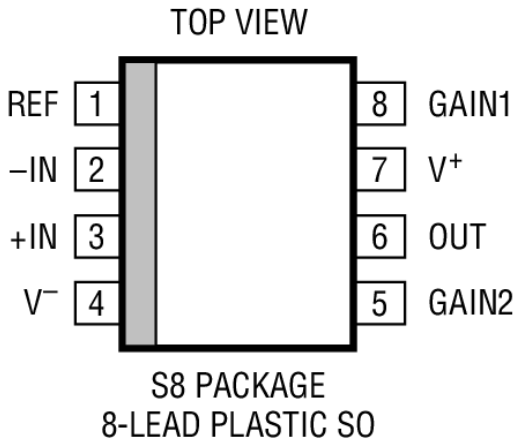

Figura 3-3. Pinout LT1990

Por último, las salidas de los AD se conenctan a un ADC de 16 canales (tal y como queda ejemplificado en la Figura 3-1) que transmitirá toda la información de las tensiones de cada celda en serie al microcontrolador para la monitorización.

Aunque se explicará en mayor detalle el funcionamiento del ADC AD7490 en los apartados de desarrollo hardware y código, aclarar que al tener una precisión de 12 bits y estar alimentado a 5V obtenemos una precisión de 1.22mV por LSB, y considerando todos los posibles errores contemplados en el datasheet, podemos obtener una precisión de alrededor de 10mV, mejor de lo contemplado en los objetivos iniciales.

3.1.3 Monitorización de temperatura

Partiendo de los requisitos de monitorización de temperatura antes mencionados, y dado que sólo existe un prediseño de la batería actual del coche, se ha tratado de realizar un sistema de medida lo más general posible, con capacidad para todas las alternativas barajadas hasta ahora.

Se consideraron incialmente diferentes opciones, entre ellas, el uso de PTCs y sus respectivos acondicionamientos, incluso se probó en el tercer prototipo (ver apartado 7 de evaluación de prototipos) el uso de una PT1000 con satisfactorios resultados. Pero a medida que se fue cerrando el prediseño de la batería, y al considerar la gran cantidad de celdas que llegarían a usarse y por ende la gran cantidad de sensores de temperatura, se optó por sensores comerciales preparados para aplicaciones de automoción, que proporcionaban una precisión similar a la PT1000 con mayor facilidad de implementación.

El modelo elegido es el LMT87LPG, es un modelo de sensor de temperatura CMOS con salida analógica lineal inversamente proporcional a la temperatura y precisión típica de 0.4ºC. Su funcionamiento es tremendamente sencillo al solo disponer de 3 pines. Dos para su alimentación (que requiere de un condensador de 10nF entre ellos) y uno para la salida, como se observa en la figura proporcionada en la hoja del fabricante [2].

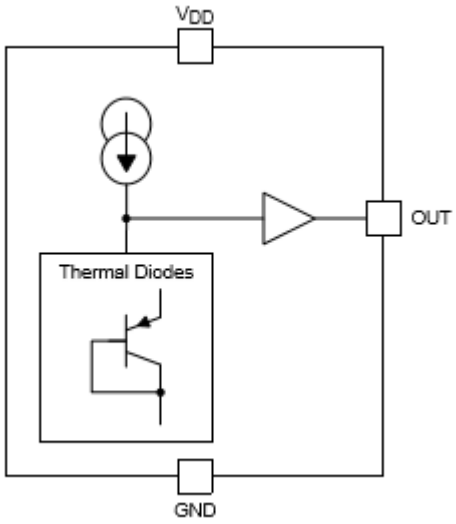

Figura 3-4. Diagrama de bloques de LMT87

El bajo consumo de cada sensor facilita la aplicación de un gran número de los mismos, y su rango de alimentación de ajusta al ya usado en los sistemas.

Como se ha mencionado anteriormente, el LMT87 ofrece una salida muy lineal en comparación con otras de las opciones consideradas, aunque Texas Instruments nos facilita una extensa tabla de valores, así como unas fórmulas de aproximación de la tabla, debido a que exite una pequeña forma parábolica apenas considerable en la gráfica.

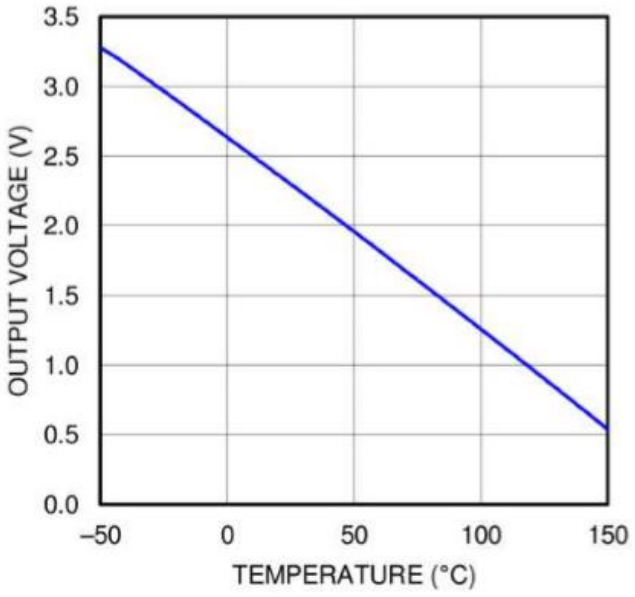

Figura 3-5. Relación de voltaje de salida y temperatura

$$T = \frac{13.582 - \sqrt{(-13.582)^2 + 4 \cdot 0.00433 \cdot (2230.8 - V_{TEMP}(mV))}}{2 \cdot (-0.00433)} + 30$$

Para obtener una aproximación más sencilla pero menos precisa válida para entre 0ºC y 60ºC también nos facilitan la siguiente expresión.

$$T = \frac{V(mV) - 2633}{-13.5667} \ ºC$$

En encapsulado elegido entre los posibles ha sido el TO-92S, necesariamente el through-hole para que fuese posible su montaje en la batería, y el 92S porque plantea el mejor comportamiento térmico de entre todos, con menor resistencia termal al ambiente y al empaquetado que el TO-92.

Por último, para que sea posible la lectura digital de las temperaturas por el microcontrolador del *Slave* se han usado dos ADC AD7490, el mismo modelo que el usado en la monitorización de voltaje. Aunque como se observa en el apartado del enrutado de las PCBs, pese a tener 16 canales cada uno, sólo se han usado 8. Esto es por lo comentado anteriormente de generalización, dado que en el prediseño de la batería se contemplan dos opciones, una de ellas requiere el doble de sensorización de temperatura que la otra, 32 sensores y 16 respectivamente. Por ello, si se necesitase finalmente de esas 32 entradas sólo habría que cambiar el enrutado y usar los 32 canales disponibles.

3.1.4 Monitorización de corriente

La última función del sistema de monitorización es la de la medida de corriente, que se efectua mediante un sensor de efecto Hall. El principio de dicho sensor se basa en la medición la perturbación magnética creada por el paso de corriente, lo que no afecta en el consumo ni perturba la corriente que circula por la batería.

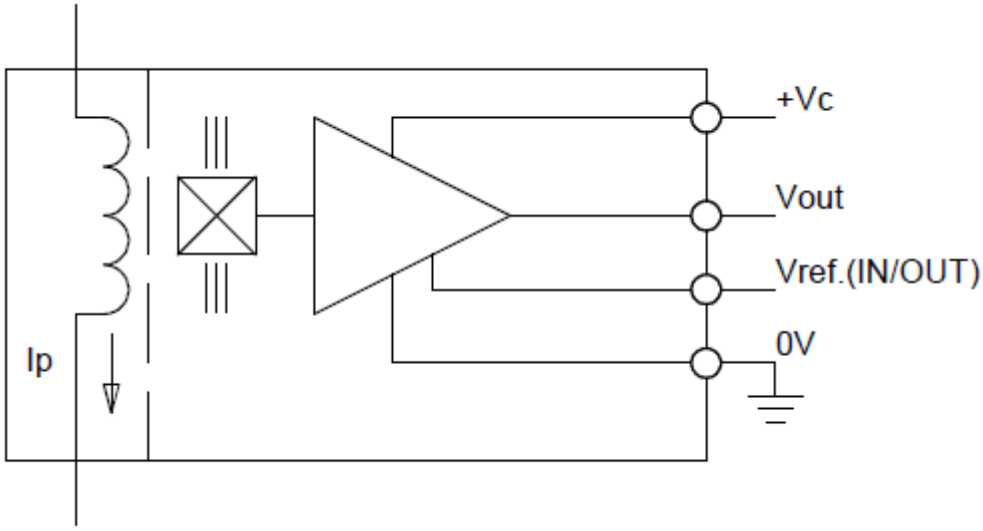

Figura 3-6. Esquema de funcionamiento del sensor LEM

En este caso, y dadas las altas corrientes que pueden salir o entrar en la batería, del orden de 250A, se ha elegido el sensor LEM HTFS 200-P, que además de tener gran facilidad de conexionado soporta hasta 300A.

Como se observa en la figura, tiene únicamente 4 pines, dos de los cuales son de alimentación, 5V y 0V. Se empleará como referencia 2.5V, lo que deja una gama partida de valores, la tensión de salida superará esa referencia cuando circule en el sentido de Ip especificado en la figura, y bajará de 2.5V cuando lo haga en el contario. La distribución de la salida es la siguiente:

$$V_{OUT} = V_{REF} \pm \left(1.25 \cdot \frac{I_P}{200}\right) V$$

En nuestro caso y para nuestro rango de intensidades la tensión de salida está acotada entre 4.0625V y 0.9375V, lo que deja un margen muy pequeño de tensión para el rango de corriente tan amplio que contemplamos. La forma de solventarlo es con un ADC capaz de medir a una altísima precisión, emplearemos el AD7988-5 de 16 bits alimentado a 5V, que nos otorga una precisión de LSB de 76.294 µV, lo que se traduce en una precisión de medida de corriente de 12.2mA sin tener en cuenta los posibles errores del ADC. Más que suficiente para la aplicación que necesitamos. Los detalles del ADC se explicarán en el apartado de desarrollo hardware.

3.2 Balanceo

3.2.1 Introducción

El balanceo es una técnica utilizada durante la carga que permite maximizar la capacidad y aumentar la longevidad de un paquete de baterías con varias celdas.

Para aprovechar al máximo la capacidad hay que asegurarse que durante la carga todas las celdas tengan como mínimo un valor de carga igual al de la celda con menor capacidad, por lo tanto, si dicha celda llegase a cargarse por completo sin que las otras lo estén, habría que descargar parte de esa energía para poder cargarlas todas de nuevo, actuando la celda de menor capacidad como la limitante de todo el conjunto.

Dado que las baterías suelen estar formadas por el mismo tipo de celda, podría parecer a priori que el balanceo no es necesario, puesto que al tener las mismas propiedades se cargarían a la misma velocidad y hasta la misma capacidad. Nada más lejos de la realidad, cada celda, aunque igual en fabricación a las otras, tiene un comportamiento ligeramente distinto debido a su naturaleza química, por lo que el balanceo es una herramienta imprescindible en cualquier BMS.

Para ello existen dos métodos:

- Balanceo pasivo: La energía sobrante de la celda a balancear se elimina mediante un circuito resistivo, disipándose en forma de calor. Aunque es la forma menos eficiente de hacerlo, su facilidad de implementación y fiabilidad, además de bajo coste, lo hacen la opción más favorable para este proyecto.
- Balanceo activo: La energía sobrante de la celda a balancear se transfiere a otras celdas con menor carga, así mientras la celda con excesiva carga la disminiye, las celdas con menor carga la aumentan. Normalmente se realiza mediante convertidores DC/DC, lo que aumenta considerablemente el coste, además de la complejidad, siendo necesario un sistema de control del flujo energético.

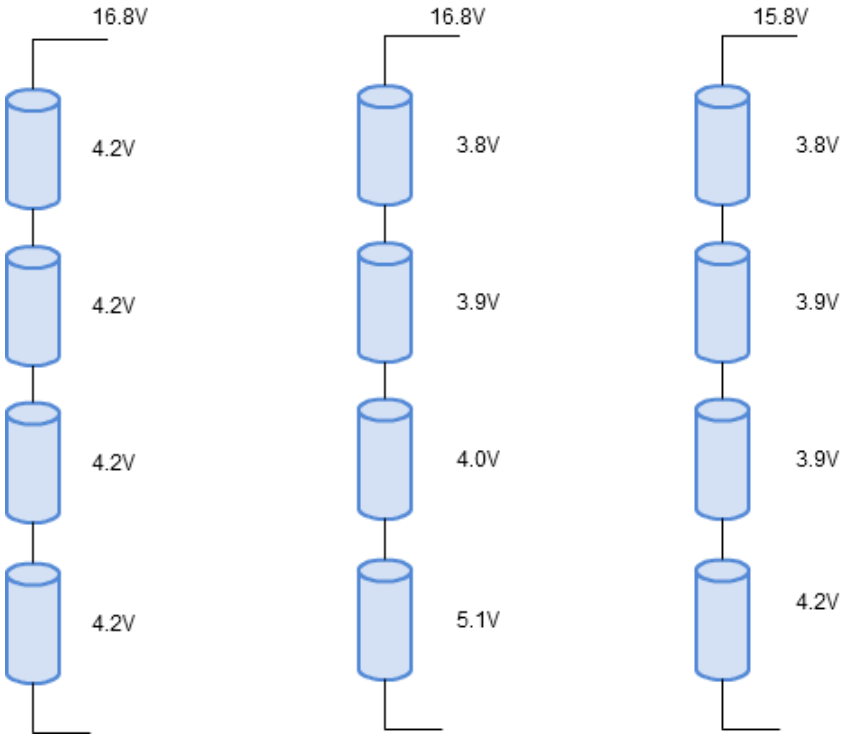

Figura 3-7. Pack de baterías con balanceo y sin balanceo

Como se observa en la imagen, la batería de la izquierda está completamente balanceada y se aprovecha el 100% de su capacidad. La batería del centro por otro lado, no tiene ningún tipo de balanceo y está aparentemente cargada a 16.8V, sin embargo, no posee un control del voltaje de cada celda y se han cargado independientemente poniendo a la batería en un estado peligroso. Por último, en la última batería se ha detenido la carga cuando una de las celdas ha llegado a su máxima capacidad, a pesar de ello, el resto no lo ha hecho y se ha desaprovechado una parte de la capacidad de la batería.

El proceso de balanceo se puede realizar de diversas formas; cuando alguna de las celdas supera en tensión a la celda con la tensión mínima, cuando alguna celda llegue al máximo, o también es común encontrarse que cada un cierto periodo se pare la carga y se igualen todas las tensiones, sin necesidad de haberse llegado al límite de tensión. El modo en que se realice dependerá de la aplicación o de la efectividad de cada método según la aplicación.

En la siguiente imagen se observa paso a paso cómo es el proceso de balanceo de una batería. Partiendo de la batería totalmente descargada (a), se conecta el cargador y tras un tiempo de carga se llega a unas tensiones dispares entre las celdas (b), desde ese momento se puede comenzar el balanceo mientras continua la carga del pack completo, provocando que las celdas de mayor carga disipen parte de la corriente entrante. Cuando la tensión de alguna celda llegue al límtite superior, se detiene la carga, mientras se sigue balanceando y descargando las celdas de mayor tensión. En el momento en el que esas celdas bajen ligeramente su tensión, se podría conectar el cargador de nuevo, pero para evitar la contínua conexión y desconexión del cargador, es recomendable volver a conectarlo cuando se hayan bajado las tensiones un cierto umbral o hayan sido igualadas a la mínima (d).

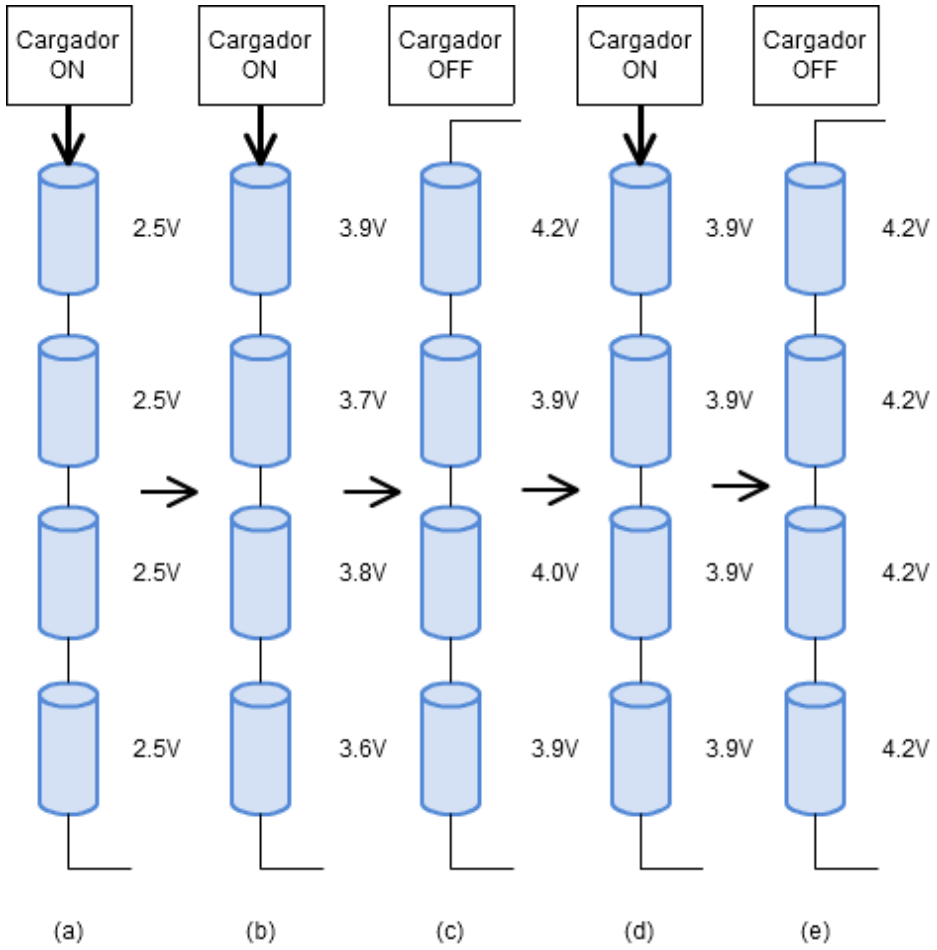

Figura 3-8. Ejemplo de proceso de balanceo

Con las tensiones igualadas en (d) se continua la carga, y tras un breve periodo de tiempo probablemente se vuelva a producir una situación similar a la (b) con las tensiones desigualadas, es por eso que el balanceo se considera un método iterativo, que no cesa hasta que se complete la carga completa de todas las celdas.

Debido a que las corrientes de fuga en circuito abierto de cada celda son distintas, una batería nunca permanece balanceada con el paso del tiempo, por tanto, el balanceo debe ser realizado siempre que se produzca una carga, tratando de cargar periódicamente la batería para que no haya mucha dispersión en el SOC de las celdas.

3.2.2 Criterio de diseño

Como antes se ha mencionado, por la facilidad de implementación, el balanceo pasivo ha sido la opción elegida en nuestro caso, cuya circuitería se explicará al detalle en el siguiente punto. Dado que el circuito de balanceo maneja una corriente considerable comparada con el resto de subsistemas, se ha aislado galvánicamente del microcontrolador, evitando cualquier corriente que pudiese llegar a destruirlo. Se ha dimensionado la resistencia conforme a que diese la máxima corriente posible sin necesidad de recurrir a empaquetados no SMD extremadamente grandes, con una disipación de casi 2W por cada celda en serie y una corriente de hasta 0.42A.

3.2.3 Circuito de balanceo

El circuito usado para el balanceo está basado en unos apuntes de la Universidad de Málaga de la asignatura Electrónica del Vehículo Eléctrico en los que se desarrolla un pequeño BMS basado en Arduino. El circuito previo a su aislamiento galvánico es el siguiente:

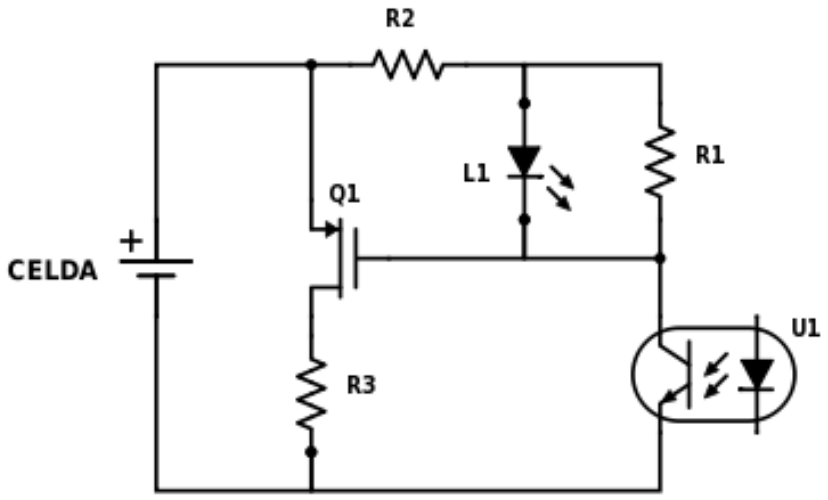

Figura 3-9. Circuito de balanceo

Siendo
R3=10Ω
R2=330Ω
R1=560kΩ
U1: Optoacoplador Toshiba TLP293-4
Q1: PMOS FDN304P
L1: Diodo LED naranja

Como se observa, la celda se dispone en pararelo con la resistencia de disipación R3 y un PMOS, cuyo objetivo es abrirse o cerrarse según conduzca o no el transistor del optoacoplador, actuando como un interruptor.

El diodo L1 sirve como indicativo externo de que el circuito está balanceando, puesto que cuando el MOSFET entra en corte el led también lo hace, y cuando Q1 entra en Saturación el LED conduce.

La resistencia R1 es la única diferencia respecto al circuito propuesto por los apuntes, dado que sin dicha resistencia las corrientes parásitas de Q1 no conseguían retornar cuando el transistor del optoacoplador estaba en corte.

Cuando el transistor de U1 está en corte, L1 y Q1 entran en corte también, por lo que $V_G = V_S = 0V$ y se cumple la condición de corte de Q1. Siendo $V_T = -0.8V$

$$V_{SG} < -V_T$$

Al cerrarse el transistor de U1, el diodo conduce y Q1 entra en saturación, y gracias a la baja $R_{DS(ON)}$ (del orden de 60mΩ) el comportamiento es similar a un cortocircuito en el MOSFET, dejando que casi toda la corriente pase por la resistencia de disipación R3 y se produzca el balanceo. El dimensionamiento de R3 se hizo para no sobrepasar los 2W, potencia máxima soportada por el empaquetado 2512. Despreciando $R_{DS(ON)}$ y a la máxima tensión que una celda puede dar, la corriente máxima que puede balancear sería:

$$I = \frac{V}{R} = \frac{4.2}{10} = 0.42A$$

Y la potencia disipada en ese caso:

$$P = \frac{V^2}{R} = 1.764W$$

Dado que el optoacoplador requiere una corriente mínima en el lado del diodo para que el transistor conduzca, 10mA mínimo según recomienda Toshiba en su datasheet, se ha situado una resistencia en Pull-Up en el ánodo de dicho diodo, alimentada a 3.3V, y que dada la tensión en directa del diodo V_F = 1.1V queda su dimensionamiento:

$$R = \frac{V_{DD} - V_F}{I} = \frac{3.3V - 1.1V}{0.01A} = 220\Omega$$

Para controlar el régimen de conducción o corte de cada optoacoplador, se ha dispuesto en el cátodo del diodo un expansor de entradas/salidas, el cual puede ser configurado como entrada o como salida en drenador-abierto como se observa en la figura.

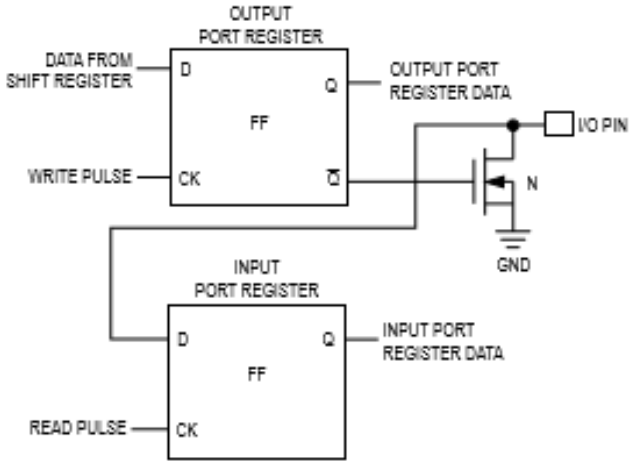

Figura 3-10. Puerto del expansor de E/S

De manera que alternando el valor del puerto de salida del expansor conseguiremos un control absoluto sobre la conducción o el corte del optoacoplador y, por ende, el control absoluto del circuito de balanceo.

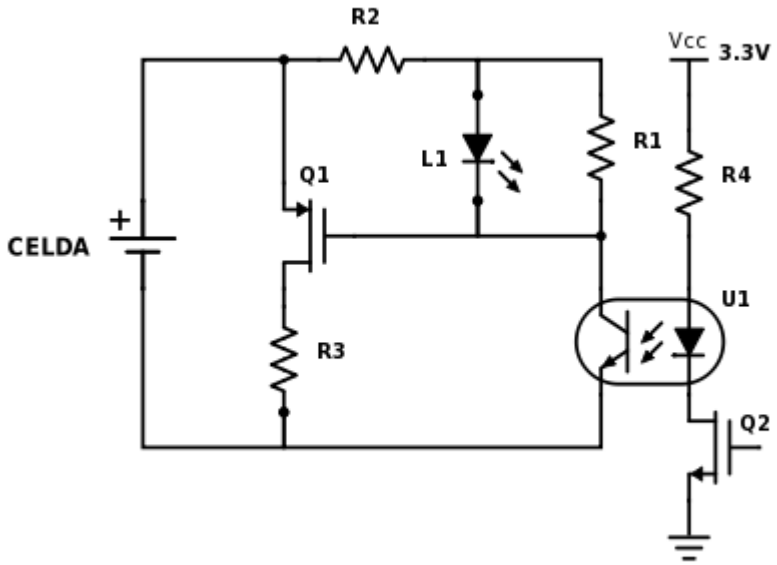

Figura 3-11. Esquema completo del circuito de balanceo

Aclarar que obviamente Q2 pertenece a uno de los expansores de E/S, no es un MOSFET independiente, y que se han usado expansores de E/S para no utilizar demasiadas salidas digitales del microcontrolador, ya que en la PCB hay 16 circuitos iguales al representado aquí arriba. Mediante el expansor podemos comunicarnos por el bus SPI ya usado con los ADCs y sólo requiere de un CS más, economizando al máximo los pines del µC.

Los expansores utilizados son los MAX7317, de 10 puertos cada uno, por lo que se han usado dos. Se eligió este modelo por la sensillez de la codificación, ya que con el cambio de un solo bit se conmuta el nivel de la salida entre alto y bajo. También se eligió porque opera en el mismo rango de tensiones que el µC, lo que hace posible la transmisión SPI sin necesidad de ningún adaptador de tensiones.

4 DESARROLLO HARDWARE

En este capítulo se incluirán el resto de subsitemas y partes del harware que no pertenecen a la gestión la batería o no han sido anteriormente explicadas, tratando de dar una visión más completa y estructurada de cada PCB. El desarrollo de las PCBs se ha realizado en Altium Designer 14.3 y se ha evoluciado mediante el prototipado y la corrección de errores, siendo el aquí expuesto el cuarto y último prototipo realizado. Los enrutados de las PCBs se han realizado a dos caras y tratando de minimizar el tamaño de la placa.

4.1 Módulo esclavo

Para facilitar la tarea de estructuración de la PCB y todas sus partes, a continuación se presenta un diagrama con el enrutado dividido en las secciones que trataremos a continuación.

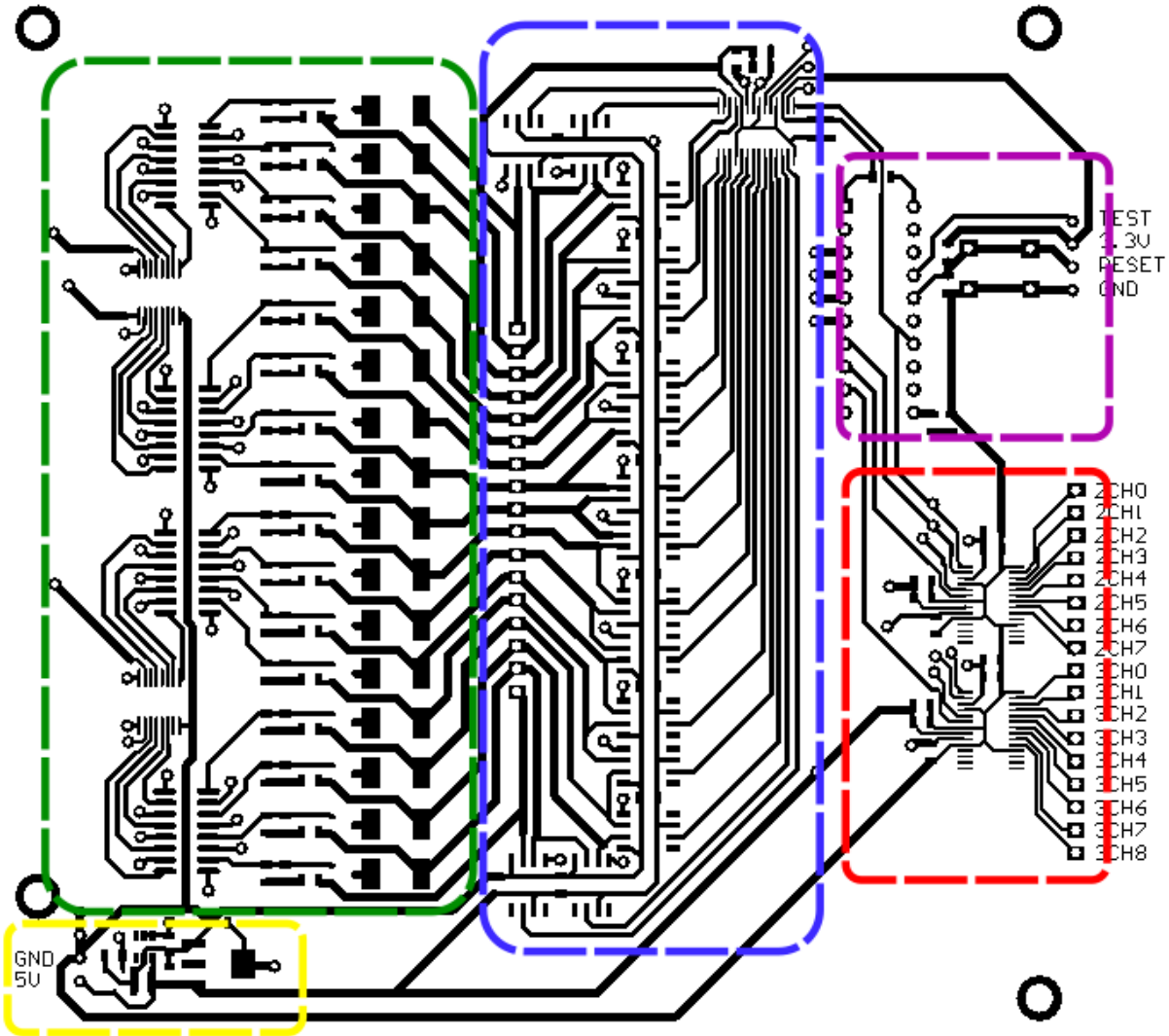

Figura 4-1. Subsistemas PCB Slave

4.1.1 Circuito de alimentación

El circuito de alimentación se corresponde con la parte bordeada de amarillo en la figura 4-1. La entrada de alimentación de las PCBs se produce a 5V distribuidos por el Maestro a cada uno de los BMS Esclavos. Dado que en cada Esclavo se requieren más tensiones de alimentación o referencia que los 5V, exiten tres dispositivos reguladores para satisfacerlo.

El AD1582 es una referencia de voltaje que convierte 5V de entrada en 2.5V y proporciona la tensión de referencia de todos los ADC de la PCB, funciona mediante una retroalimentación cerrada por sus condensadores de alimentación de 1µF y 4.7 µF.

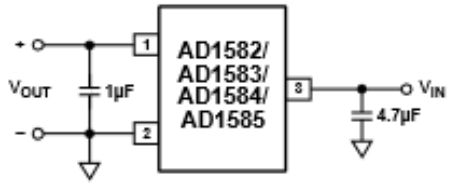

Figura 4-2. Diagrama de conexión típica del AD1582

El ADR130 es otra referencia de voltaje del fabricante Analog Devices, que convierte una tensión de entrada de entre 2V y 18V a una tensión de salida configurable, ya sean 1V o 0.5V. En nuestro caso, y como se explicó en el apartado 3.1.2, los ADs necesitarán una referencia de 0.5Vque consiguiremos uniendo el pin 5 de SET al pin 4 de VOUT. Requiere también de un condensador de 0.1µF a la entrada y salida.

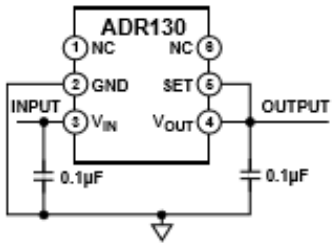

Figura 4-3. Diagrama de configuración del ADR130

Por último, y para conseguir la tensión de 3.3V necesaria para las resistencias de Pull-up, para la alimentación del microcontrolador y para los expansores de E/S, usaremos el regulador de tensión LM1117 en su versión de salida fija de 3.3V de Texas Instruments. Al igual que los dos anteriores, require los siguientes condensadores a la entrada y salida.

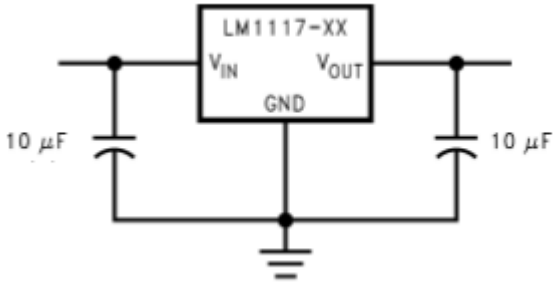

Figura 4-4. Diagrama de funcionamiento básico del LM1117

La PCB al completo está diseñada para economizar al máximo el espacio disponible debido a las altas limitaciones espaciales en el interior del monoplaza, el circuito de alimentación es una prueba de ello, ocupando todo lo anterior menos de 25 x 13mm. La localización del circuito ha sido elegida para estar lo más cerca posible del acceso a las pistas de 3.3V de los expansores y 5V de los AD.

El GND utilizado en toda la PCB y al que están referenciados los planos de tierra son el mismo para todos los Esclavos, que es el que otorga la salida del convertidor DC/DC del Maestro.

4.1.2 Subsistema de Balanceo

Corresponde con la parte cercada por el cuadrado verde en la figura 4-1. Pese a que la mayoría de los aspectos ya han sido mencionados en el apartado 3.2.3, hay ciertos detalles que antes no han sido explicados.

El IC usado para los optoacopladores es el TLP293-4, cada uno contiene cuatro optoacopladores independientes entre sí en su interior. Dado que se necesitan 16 circuitos resistivos para el balanceo, se necesetarán 4 circuitos integrados como el mostrado en la siguiente figura.

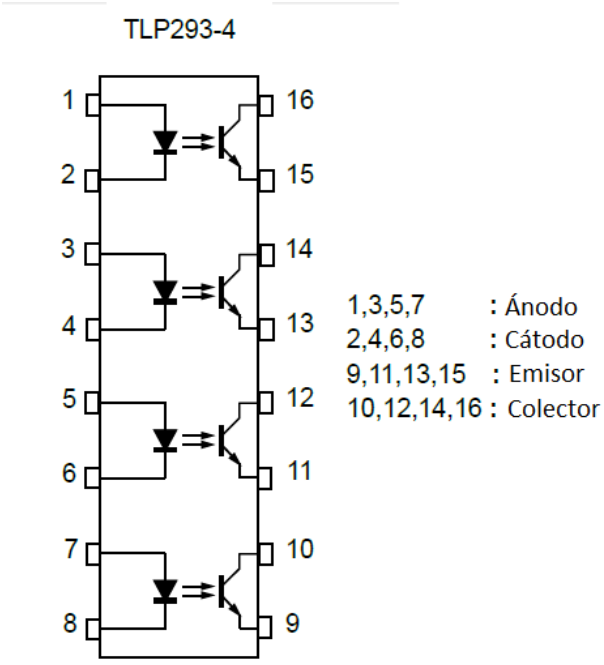

Figura 4-5. Esquema interno TLP293-4

La disposición en la que se conecta cada una de las patas sigue el esquema de U1 representado en la figura 3-11, tratando cada optoacoplador de forma independiente.

Entre cada dos TLP293-4 se dispone un expansor de E/S MAX7317 comunicado por SPI con el microcontrolador. El expansor está alimentado por los 3.3V dispuestos por el LM1117, con un condensador opcional de 0.047µF entre V^+ y GND.

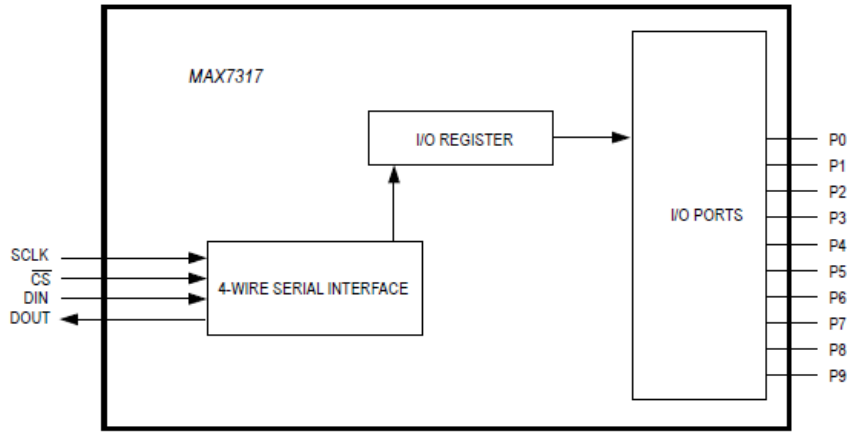

Figura 4-6. Esquema interno MAX7317

Los pines SCLK, DIN, CS y DOUT son los asignados para la comunicación SPI, dejando el DOUT o MISO sin unir el bus por motivos que se explicarán en el apartado 5 de Comunicaciones.

El circuito resistivo de cada celda ha sido compactado lo más horizontalmente posible para conseguir una disposición espacial ordenada, facilitando el conexionado. En la cara trasera, no apreciable en la figura 4-1, se han unido mediante vías las conexiones imposibles de realizar en la cara delantera por los numerosos cruces entre pistas.

Las resistencias usadas en el circuito han sido todas SMD de métrica 0805 por su reducido tamaño, excepto las resistencias de disipación de corriente (R3 en la figura 3-11), que son de métrica 2512 para disipar mejor el calor de la corriente pasante. Las pistas que reccore la corriente balanceada se han fabricado con un ancho de 35mils.

El P-MOS usado ha sido diseñado para aplicaciones especiales de gestión de baterías, con lo cual soporta corrientes de hasta 10A en forma de pulsos y 2.4A de forma continuada, más que suficiente para la carga a la que será sometido.

4.1.3 Microcontrolador

El microcontrolador del *Slave* y todo lo necesario para su funcionamiento está recogido por el recuadro rojo en la figura 4-1. El MSP430G2553 ha sido el elegido en su versión de 20 pines en formato DIP, por si fuese necesaria su extracción y uso en la launchpad.

El MSP430G2553 cuenta con unas características suficientes para nuestro uso:

- 16 MHz de frecuencia de reloj
- 24 pines GPIO
- 0.5 KB de RAM
- 16KB de memoria no volátil
- 2 puertos SPI
- Rango de alimentación de 1.8V a 3.6V

Para el uso del µC sin el launchpad el fabricante recomienda [3] una resistencia en Pull-Up de 10kΩ en el pin de RESET, un botón de RESET en paralelo a un condensador de 1nF y un condensador de 22pF entre 3.3V y GND, como se observa en la figura 4-7.

Los dos puertos SPI son fundamentales puesto que se usará el puerto de la USCI_B (P1_5, P1_6 y P1_7) para la comunicación como *Master* con los periféricos de la PCB y el puerto de la USCI_A (P1_1, P1_2 y P1_4) para la comunicación como *Slave* con el BMS *Master*. En total, y sumando los 3 pines fundamentales del bus SPI (SCLK, MISO, MOSI) el SPI_B necesita 8 pines para la comunicación con los cinco periféricos de la PCB (EXP1, EXP2, ADC_Volt, ADC_Temp1, ADC_Temp2), mientras que el puerto SPI_A sólo necesita los 3 fundamentales y uno de entrada del Chip Select, que se introducen a la PCB por el puerto H4.

Para la programación del µC "on board" son necesarios 4 pines de salida de la PCB que puedan ser conectados con unos jumpers de la launchpad, como muestra la figura 4-7 son el pin de TEST, RESET, 3.3V y GND del puerto H2.

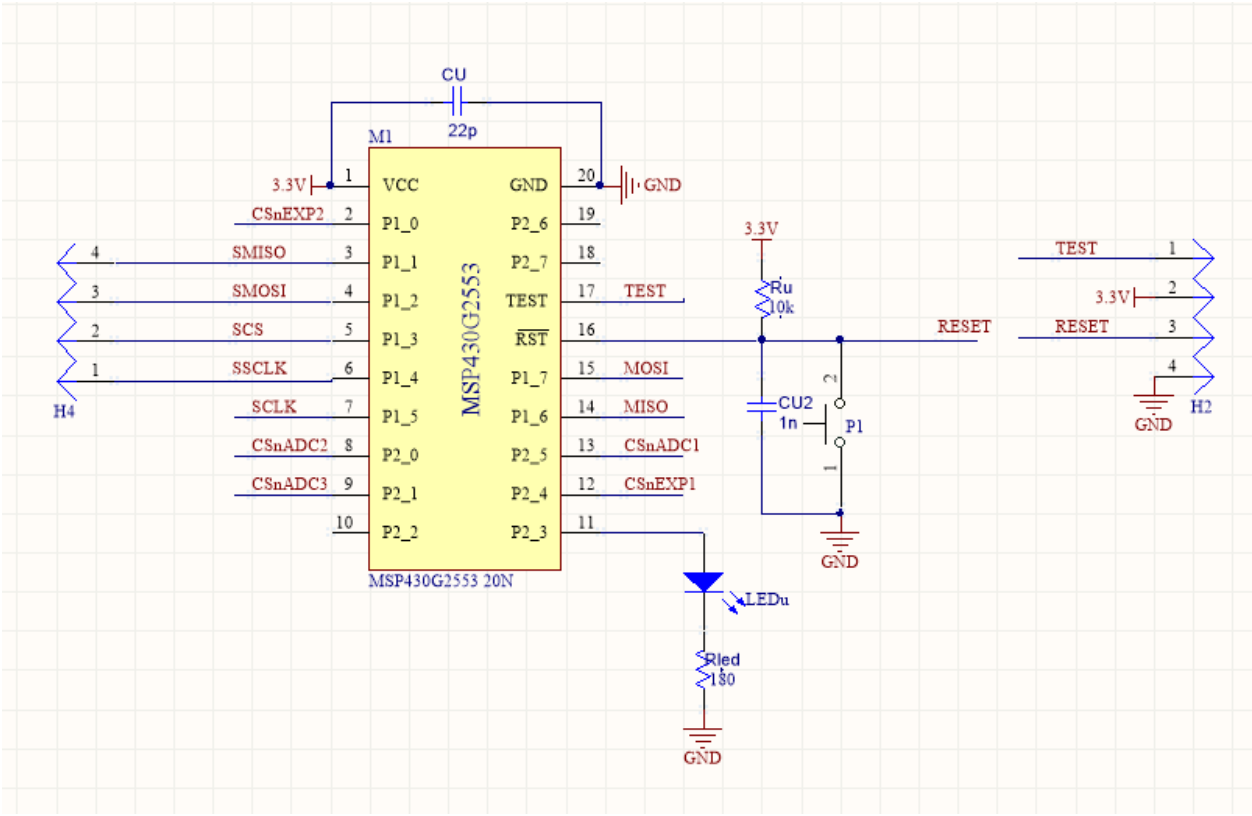

Figura 4-7. Esquemático del subsistema del MSP430G2553

Por último, como se puede observar en el pin 2_3, se ha instalado un led multipropósito, con el objetivo de usarlo como indicador siempre que fuese necesario o como señal en caso de error.

4.1.4 Subsistema de temperatura

Correspondiente a la zona con el borde rojo de la figura 4-1. En la PCB el único hardware de dicho subsistema es la entrada de los sensores analógicos a los ADCs.

Los ADCs usados son los AD7490, son convertidores de aproximaciones sucesivas de 12 bits y 16 canales, con empaquetado TTSOP de 28 pines y un tamaño muy reducido. Necesitan además de VDD y GND, una referencia de valor igual a la mitad de la alimentación (usaremos los 2.5V proporcionados por el AD1582) y una tensión V_{DRIVE} a la que transmitirá el bus SPI, sin necesidad de un aislador digital o un adaptator.

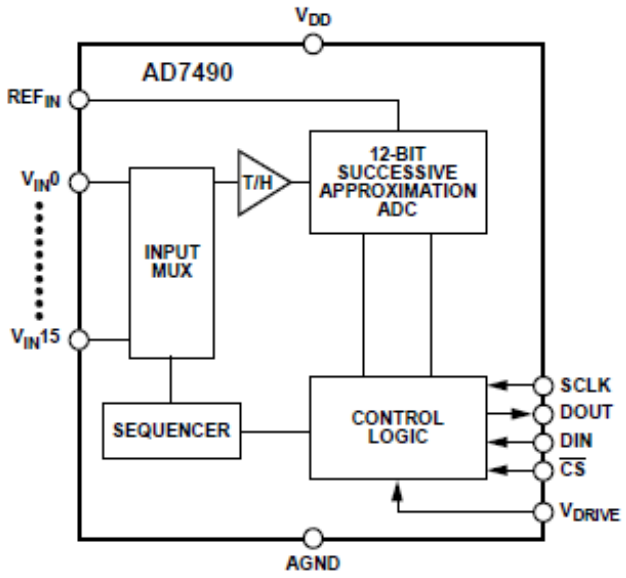

Figura 4-8. Esquema de funcionamiento AD7490

Para el correcto funcionamiento del AD7490 son necesarios unos condensadores para el suavizado de las tensiones de referencia, alimentación y drive:

- Un condensador de 10µF en paralelo con otro de 0.1µF entre VDD y GND
- Un condensador de 10µF en paralelo con otro de 0.1µF entre V_{DRIVE} y GND
- Un condensador de 0.1µF entre REF_{IN} y GND, lo más cerca del ADC posible

Así como también es necesario la unión entre sí de las 5 entradas de AGND existentes, según especifica el fabricante en la hoja de datos [4].

Por el puerto H5 entran las 16 entradas de tensión de los sensores, ampliables a 32 cambiando el conector y enrutado. Dado que los ADCs pueden leer una entrada de entre 0 y 5V, el modelo del sensor de temperatura puede ser intercambiado por otro con el mismo rango de tensión analógica de salida.

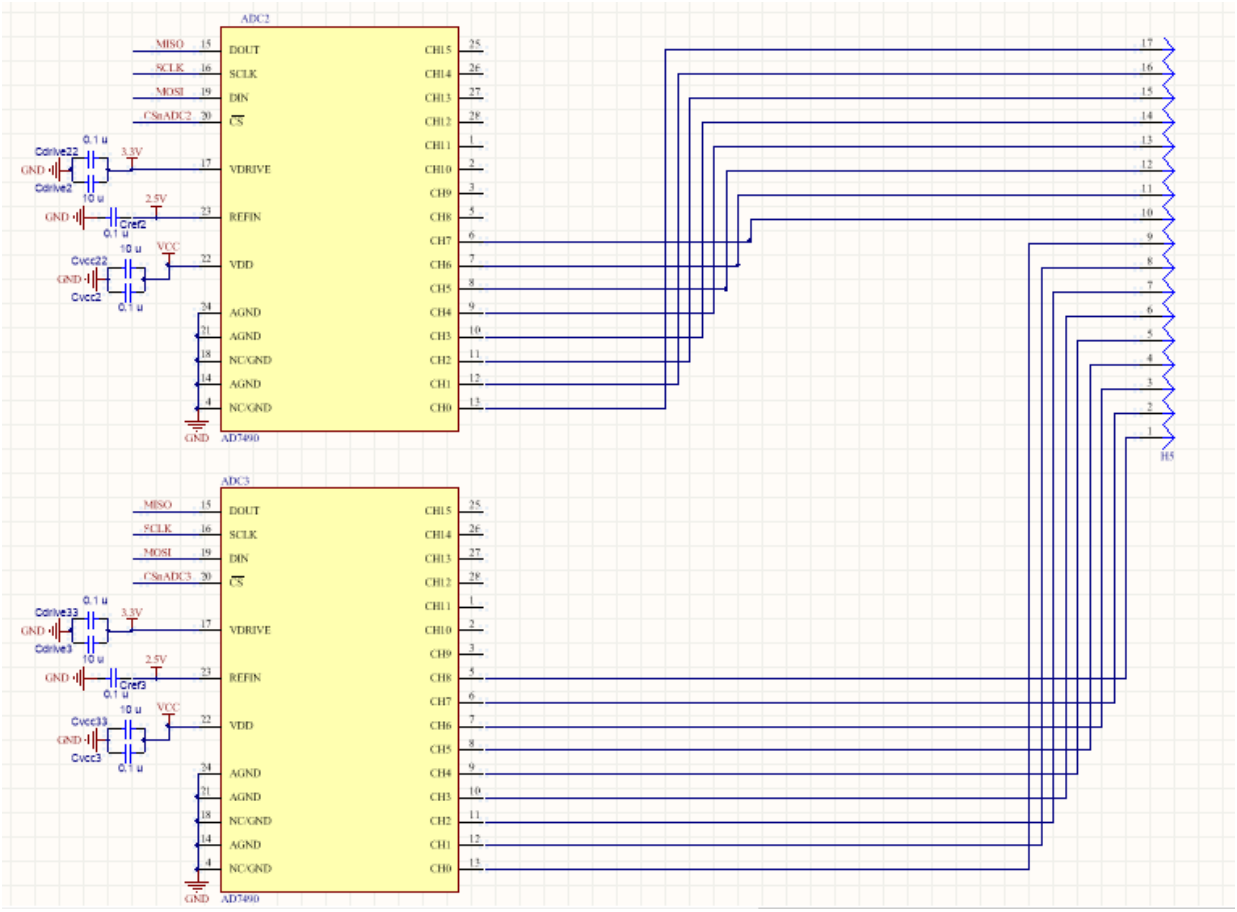

Figura 4-9. Esquemático del subsistema de temperatura

4.1.5 Subsistema de voltaje

Por último, y contenido por el recuadro azul en la figura 4-1, el subsistema de monitorización de voltaje, que como se explicó en el apartado de gestión de baterías, basa su grueso en los amplificadores diferenciales LT1990A.

La tensión de salida de cada uno de los 16 ADs entra directamente al AD7490, que tiene las mismas consideraciones para su funcionamiento que las explicadas en el apartado anterior.

Únicamente son necesarios 8 condensadores entre VCC y GND de 0.1µF para los ADs, con un condensador por cada dos LT1990A.

En la figura 4-10 se expone una parte representativa del esquemático del subsitema.

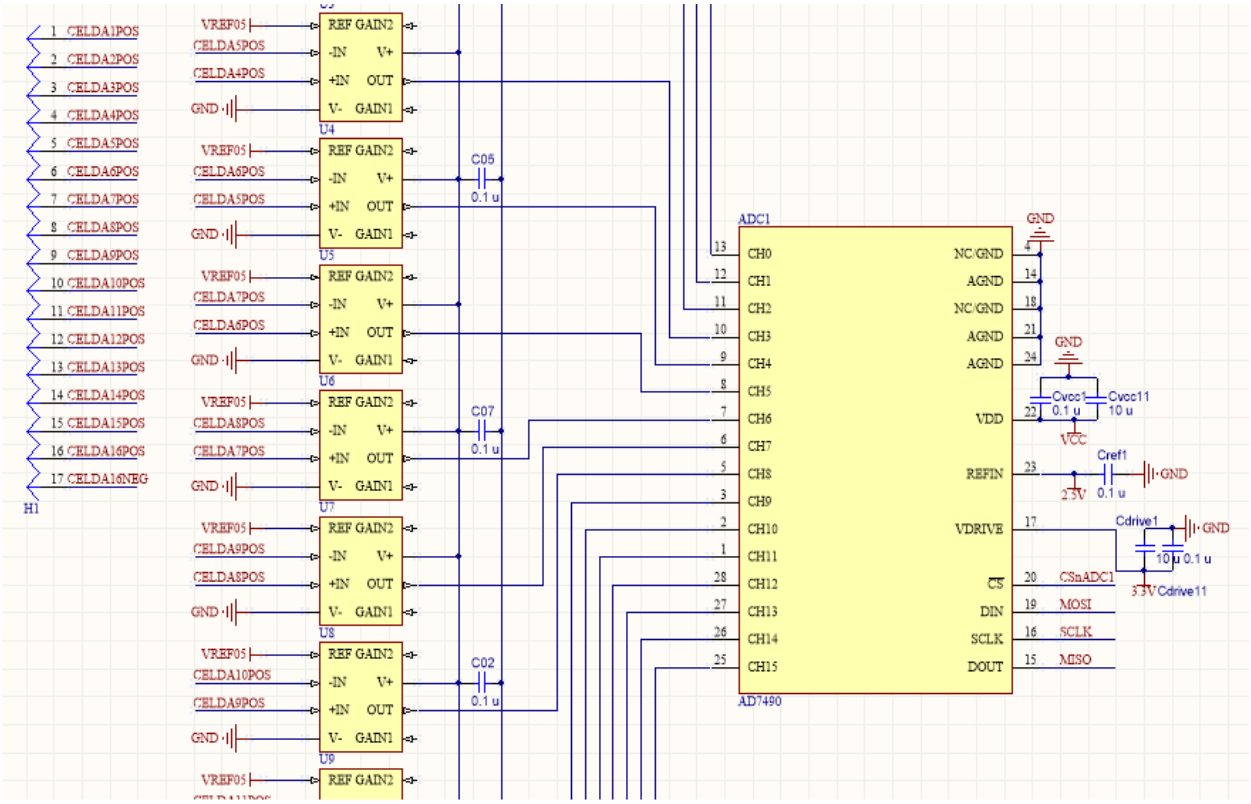

Figura 4-10. Esquemático del subsistema de voltaje

4.2 Módulo del Maestro

El siguiente esquema presenta la cara superior del enrutado de la PCB del Maestro, se han dividido los subsistemas por colores y se han mantenido los planos de tierra, a diferencia que en el anterior esquema.

A la izquierda en la cara superior, el plano es de VCC, en la cara inferior reside el plano de GND. A la derecha los planos son de LVGND en ambas caras.

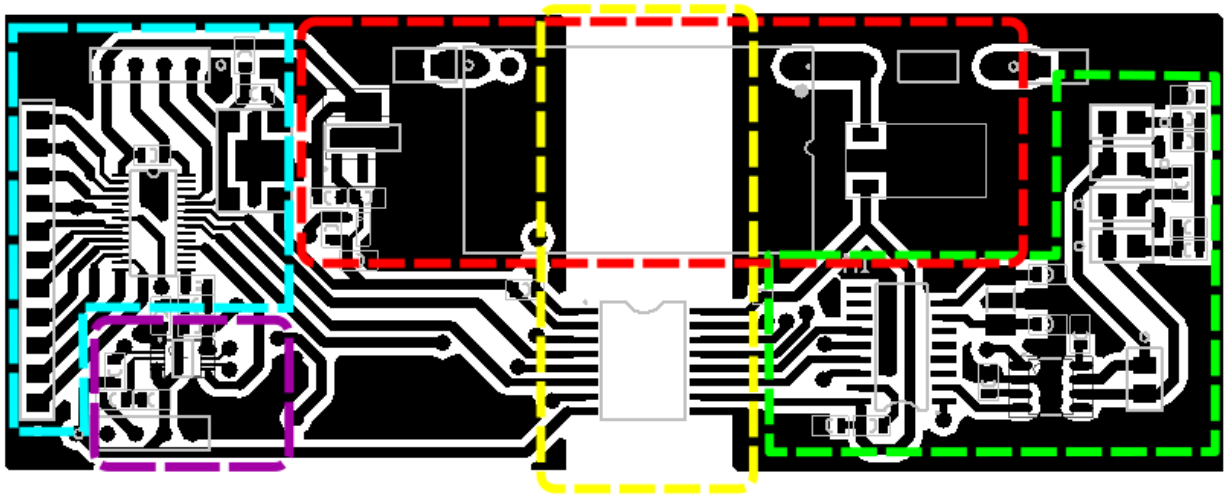

Figura 4-11. Subsistemas PCB Master

4.2.1 Subsistema de intensidad

Se corresponde en la PCB por la zona remarcada en violeta en la figura 4-11. Está compuesta por el puerto H1, que cede la alimentación y referencia al sensor FEM, y contiene su salida. Mediante un filtro RC la señal analógica del sensor entra en el AD7988-5, un ADC monocanal de 16 bits.

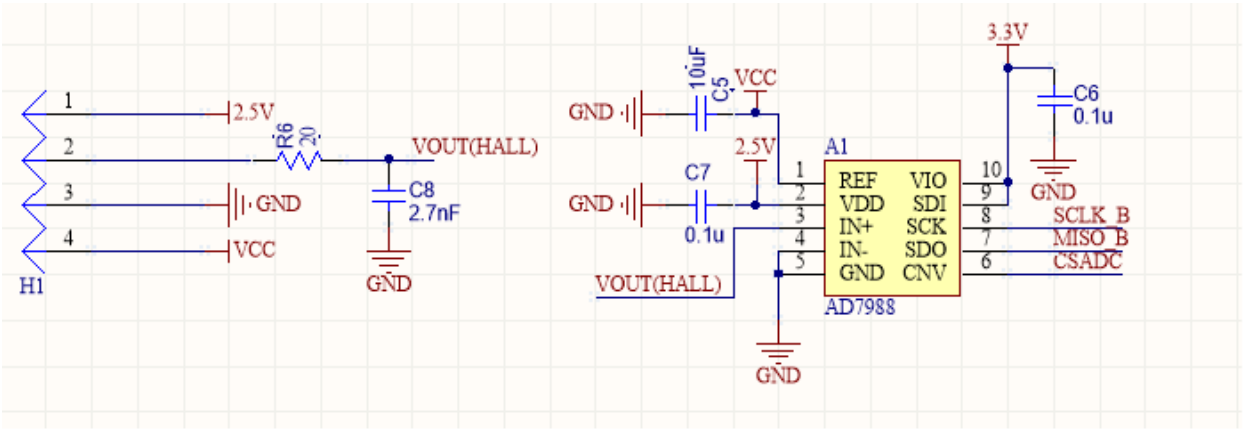

Figura 4-12. Esquemático subsitema de intensidad

A diferencia de la mayoría de ADCs, la alimentación y la referencia del AD7988-5 son independiendientes entre sí, siendo su alimentación 2.5V y la referencia la que marca el rango de tensiones de entrada (5V en nuestro caso). Ambas tensiones necesitan un condensador respecto a GND como se observa en la figura 4-12, que serán de 0.1µF y 10µF respectivamente.

El AD7988-5 nos brinda la posibilidad de una entrada diferencial de tensión entre IN^+ y IN^-, aunque en nuestro caso no es necesaria, por lo que la entrada negativa se ha conectado a GND.

Para la selección del modo CS, que nos permite la comunicación por SPI, el pin 9 de SDI debe estar unido con VIO, que será la tensión a la que se realizará la comunicación.

4.2.2 Aislamiento galvánico

Una de las restricciones más importantes en la normativa de FSAEE en la elaboración del BMS y otras PCBs es la obligatoriedad de un aislamiento galvánico entre el sistema tractor y el sistema de baja tensión.

El sistema de LV del coche proviene de una batería de litio de 12V que alimenta la mayoría de los subsistemas electrónicos. El sistema tractor queda definido como aquel que proporciona potencia al vehículo o esté en contacto con cualquier elemento de HV, puesto que en cada Slave los ADs están en contacto directo con las celdas midiendo su tensión y están alimentados a VCC y GND, todo lo existente en esas PCBs se considera parte del sistema tractor según la normativa.

Además del aislamiento galvánico, si los sistemas comparten PCB deben cumplir el siguiente requisito:

[5] *"EV4.1.7 Cuando el sistema GLV y sistema tractor coincidan en la misma PCB, debe de indicarse claramente en qué parte está cada sistema y habrá que dejar un espacio entre los componentes de cada uno:"*

Tensión	***En PCB***	***Aire***	***Con Aislamiento***
0 – 50 V DC	*1.6 mm*	*1.6 mm*	*1 mm*
50 – 150 V DC	*6.4 mm*	*3.2 mm*	*2 mm*
150 – 300 V DC	*9.5 mm*	*6.4 mm*	*3 mm*
300 – 600 V DC	*12.7 mm*	*9.5 mm*	*4 mm*

Dado que la batería de HV del monoplaza ronda los 540V, será necesario el mayor espaciado de la tabla. El espaciado de 12.7mm (o 0.5 pulgadas) debe distinguir dos planos de alimentación aislados entre sí, a la izquierda, un plano de VCC que sirve de alimentación de los Slaves y por tanto está en contacto con el sistema tractor, a la derecha el plano de tierra del sistema de LV.

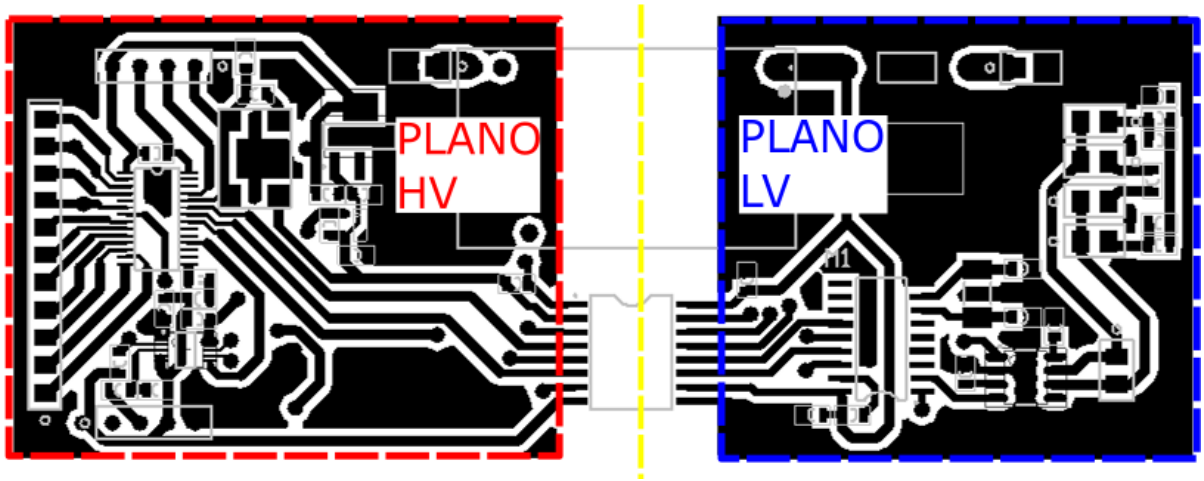

Figura 4-13. Separación de los planos de alimentación

Para conseguir 5V y 0V (nombrados VCC y GND durante todo el documento) en el plano de HV, se ha utilizado un convertidor DC/DC con aislamiento galvánico. Los valores entrantes en el convertidor han sido 12V y 0V (distintos a los 0V de salida), directamente desde el sistema de LV, que han sido nombrados LV12 y LVGND respectivamente, para facilitar la comprensión.

Dado que el BMS necesita comunicarse por CAN con el resto del vehículo, se han aislado las señales digitales procedentes del µC hacia el módulo de CAN mediante un aislador digital ISO7241C, que permite la comunicación por SPI sin contacto entre señales.

El ISO7241C es un dispositivo que proporciona entradas y salidas separadas por una barrera de aislamiento de dióxido de silicio, además de alimentaciones separadas a cada lado del aislamiento, que permiten la comunicación digital a diferentes tensiones: 5V-5V, 5V-3.3V, 3.3V-3.3V y 3.3V-5V. Su variante C proporciona además filtrado en las señales de entrada, para la eliminación del ruido.

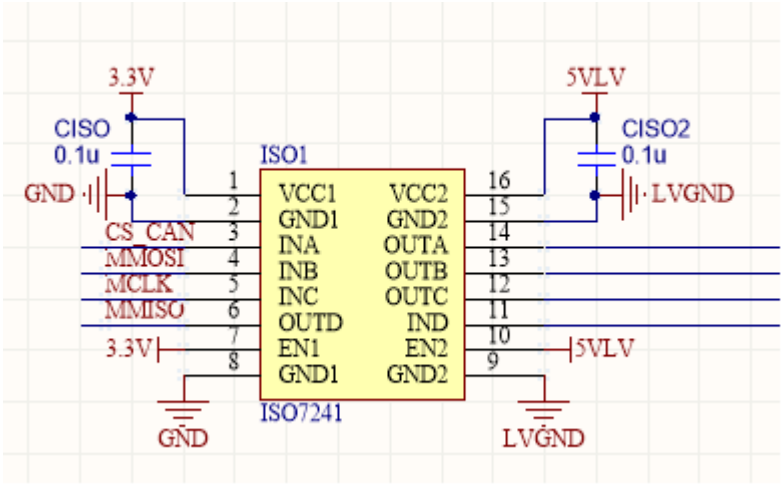

Figura 4-14. Esquemático ISO7241C

Para su correcto funcionamiento sólo son necesarios unos condensadores de 0.1µF entre la alimentación de cada lado y su respectiva referencia. Así como el pin de EN de cada lado a nivel alto para habilitar las salidas.

4.2.3 Circuito de alimentación

El circuito de alimentación de la PCB está delimitado por el recuadro rojo de la figura 4-11 y comprende cuatro dispositivos. El regulador LM1117, la referencia de voltaje AD1582, el convertidor DC/DC THM 10-1211 y el regulador LM7805.

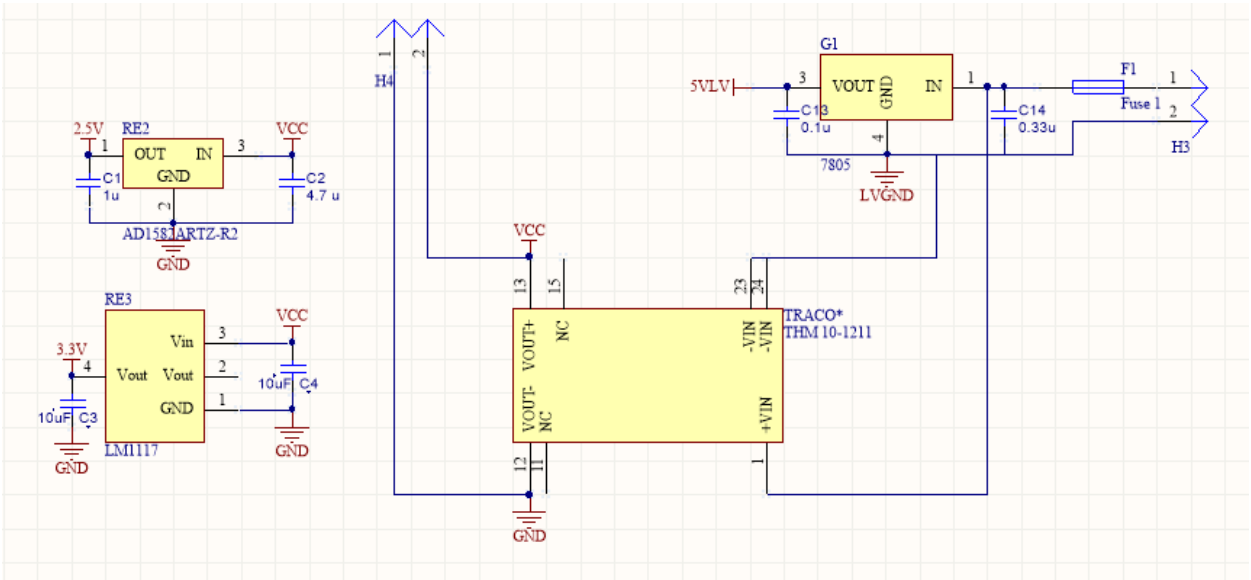

Figura 4-15. Esquemático del circuito de alimentación

El LM1117 y el AD1582 ya han sido explicados anteriormente en el apartado 4.1.1, el único matiz que es digno de mención es que su entrada en este caso es la salida del convertidor DC/DC y que por tanto pertenecen al lado de HV de la PCB.

El convertidor DC/DC elegido es el THM 10-1211, de la marca TRACOPOWER, que se encarga de la regulación de 12V de entrada a 5V, además del aislamiento galvánico entre cada una. Este modelo en concreto es un modelo de la serie THM dotado de altas capacidades de aislamiento (hasta 5000 VACrms) y alta eficiencia (85.5%). No requiere de acondicionamiento externo para su funcionamiento, y tiene empaquetado DIP24. Se ha escogido el modelo de 10W, que proporciona 2A a la salida, en base al cálculo de consumo máximo y un sobredimensionado margen de seguridad.

Ante la entrada del Traco y para su protección se ha predispuesto un fusible, que cortará la corriente en caso de cualquier fallo en el sistema de LV o sobreintensidad.

Por último, y para alimentar todos los ICs del subsistema de comunicación CAN, se ha utilizado un regulador de 12V a 5V en el lado del plano de LV, el LM7805 de Texas Instruments. Que únicamente necesita un condensador a la salida y a la entrada de 0.1µF y 0.33µF respectivamente.

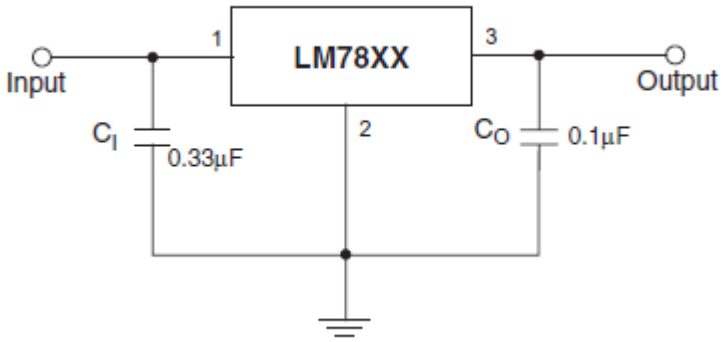

Figura 4-15. Diagrama de conexión típica del LM7805

4.2.4 Subsistema de comunicación CAN

Todo lo necesario para comunicar por CAN está contenido por la línea verde en la figura 4-11. Como se observa está en la zona de alimentación de LV, y tal y como se ha dicho en el apartado anterior, las señales provienen del aislador digital.

El primer elemento del sistema es el controlador CAN 2.0B MCP2515, que permite la transmisión y recepción de datos estándar y extendidos y se comunica con el µC del *Master* mediante SPI por los pines CS, SO, SI y SCK. Alimentado a 5V con un condensador de 0.1µF conectado a LVGND y una resistencia en Pull-Up al pin de RESET. Entre OSC2 y OSC1 necesitaremos colocar en configuración en estrella dos condensadores de 22pF y un cristal de 16MHz que actue como reloj del sistema. La salida del controlador son las señales TXCAN y RXCAN que van directamente al transceiver.

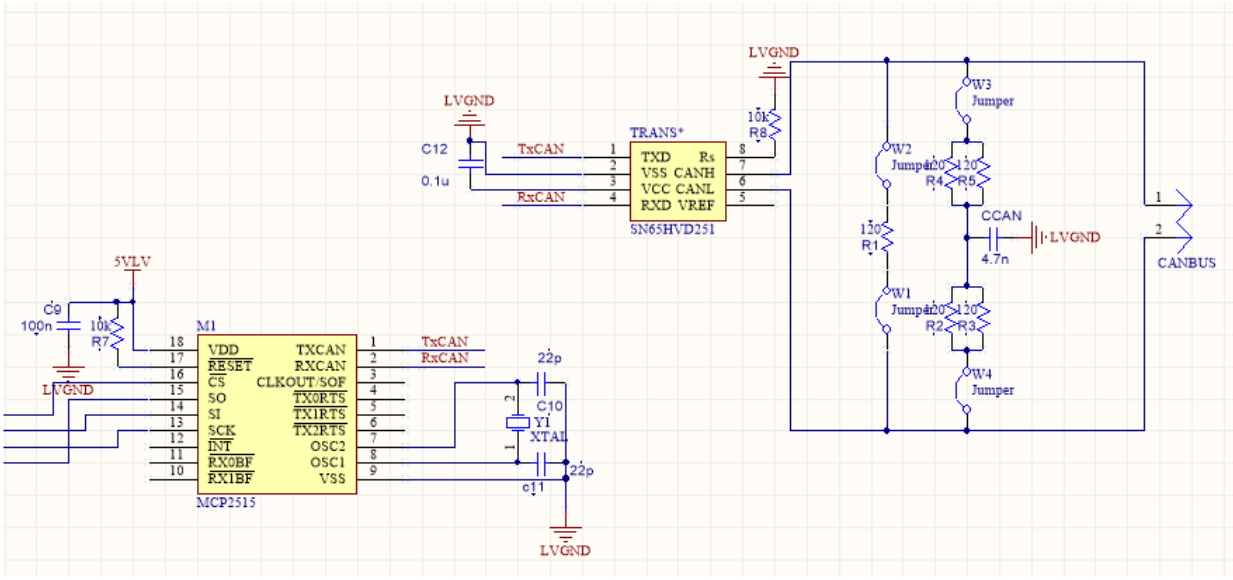

Figura 4-17. Esquemático del subsistema CAN

El transceiver SN65HVD251 supone el intermediario entre el bus y el controlador CAN, permitiendo establecer el conjunto de estándares ISO11898. Mediante el pin 8 se puede configurar el modo de operación, alta velocidad conectando el pin a tierra, control de pendiente mediante una resistencia o bajo consumo si está a nivel alto. Su salida se conecta directamente al bus.

La circuitería con jumpers que se observa después del transceiver nos permiten la configuración del bus según se quiten o pongan los jumpers, permitiendo configurar el bus en modo normal (impedancia de 120Ω) o modo estrella con terminaciones divididas para rechazar el modo común, tal y como indica el fabricante en la hoja de datos.

4.2.5 Microcontrolador y comunicación SPI

Por último, los componentes del microcontrolador del Maestro quedan recogidos en el recuadro azul de la figura 4-11. Aunque presenta muchas similitudes con el subsistema microntrolador de los *Slaves* hay ciertas diferencias que conviene destacar. En este caso el µC elegido también ha sido el MSP430G2553, pero lo ha sido en su versión de 28 pines que proporciona un puerto más (P3_x) albergar los numerosos CS que se van a manejar.

Su alimentación, condensadores, led multipropósito y puerto para programación permanecen igual que en el caso anterior. En esta ocasión se ha elegido el encapsulado TSSOP-28, dado que el espacio en la PCB del Master es considerablemente más reducido.

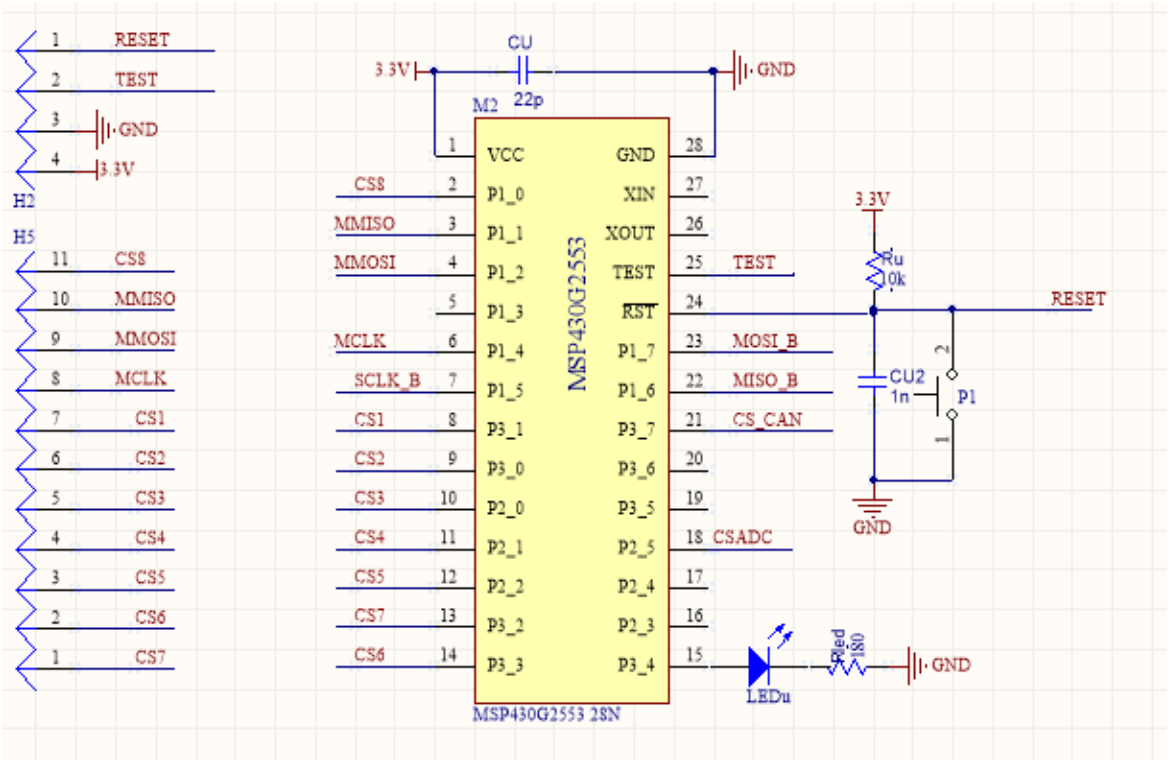

Figura 4-18. Esquemático del Microcontrolador

Como se observa en la imagen, el puerto de la USCI_A será utilizado para la comunicación SPI con los 8 Slaves y el módulo de CAN, mientras que el puerto de la USCI_B está dedicado exclusivamente al ADC del sensor de intensidad.

5 COMUNICACIONES

EN el apartado de comunicaciones se explicará detalladamente lo necesario para conocer el protocolo SPI, así como las consideraciones que han sido necesarias para las comunicaciones con cada sistema y la codificación en los mismos.

5.1 Introducción al protocolo SPI

El protocolo SPI (del inglés Serial Peripheral Interface) es un estándar de comunicaciones síncrono en serie usado principalmente para transferir información entre circuitos integrados a corta distancia. El protocolo fue desarrollado por Motorola a finales de los años 80 y se ha convertido en un estándar *de facto* por extendido uso y antiguedad.

La comunicación en SPI es completamente bidireccional con una arquitectura Maesto-Esclavo. El Maestro determina cuándo se va a comunicar, con quién lo va a hacer y a la frecuencia a la que lo va a hacer. Los dispositivos con más de un Esclavo están regulados a través de una linea de selección de Esclavos (Chip Select ó Slave Select).

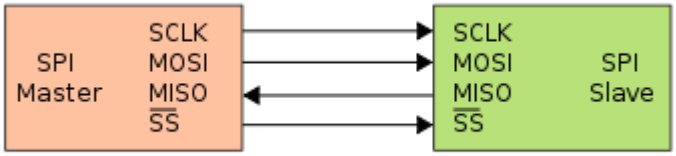

Figura 5-1. Ejemplo de SPI con un Maestro y un Esclavo

El bus SPI está compuesto por un mínimo de cuatro señales:

- SCLK: Reloj o pulso que marca la sincronización. Cada pulso se lee o se envía un bit.
- MOSI (Master Output Slave Imput): Salida de datos del Maestro y entrada al Esclavo.
- MISO (Master Imput Slave Output): Entrada de datos al Maestro y salida del Esclavo.
- CS/SS (Chip/Slave Select): Selección de un Esclavo y/o activación del mismo.

La información es enviada de manera síncrona con los pulsos de reloj, es decir, con cada pulso se envía un bit. Para que comience la transmisión el *Master* debe poner a nivel bajo el CS del *Slave* con el que se quiere comunicar, con esto, el se activa y comienza a comunicar. La transmisión de información se realiza con el primer pulso del reloj, transmitiendo información por MISO a la vez que lee el primer bit por MOSI, en estas ocasiones el *Slave* transmite basura o bits saturados, por no tener información aún de lo que el *Master* le pide. Cuando el CS vuelve a nivel alto o después de haber acabado la transmisión, el *Slave* tiene la información transmitida por el *Master* y está preparado para enviar lo que le ha pedido en la siguiente transmisión. Por otro lado el *Master* ha obtenido información que no tiene valor, y debe iniciar otra comunicación con el *Slave*. Este es el significado de full duplex, el *Master* comunica a la vez que el *Slave* por distintas lineas.

Los pulsos del reloj pueden estar programados para que la transmisión del bit se realice en cuatro modos diferentes, mediante los parámetros de polaridad y fase.

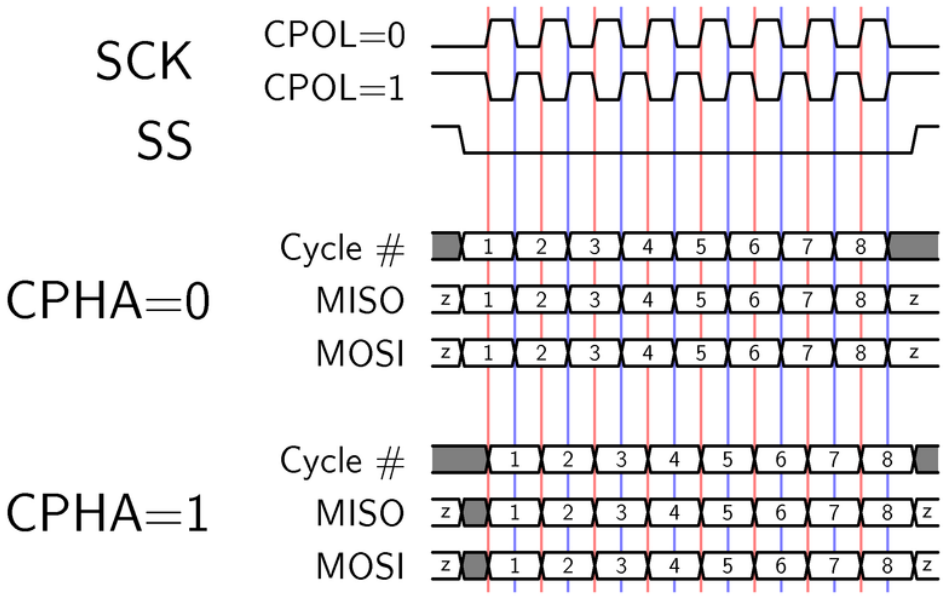

Figura 5-2. Configuración de polaridad y fase del reloj

Mediante la polaridad se configura la lectura de información en el flanco de bajada (CPOL=1) o en el flanco de subida (CPOL=0). Mediante la fase se determina el retraso de la transmisión de MISO y MOSI, si CPHA=0 los datos cambian en la mitad del ciclo de reloj de su predecesor, si CPHA=1 se retrasa para que lo hagan justo cuando empieza su ciclo.

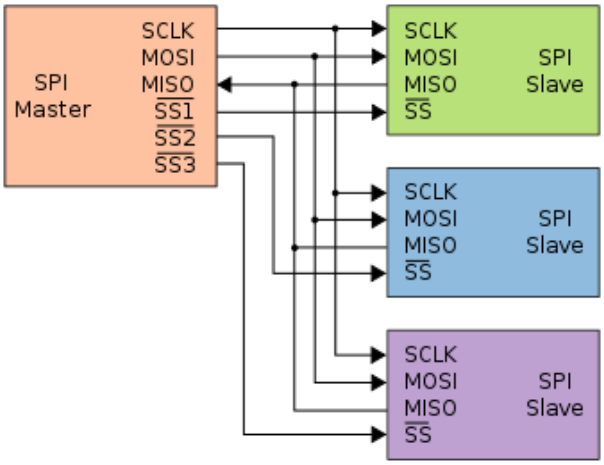

Figura 5-3. Ejemplo de SPI con un Maestro y varios Esclavos

Una de las ventajas que presenta el protocolo SPI es la reflejada en el ejemplo de la figura 5-3, en el que un solo *Master* se comunica con varios *Slaves*. Mientras que en muchos protocolos en serie son necesarias varias lineas de comunicación con cada *Slave*, en SPI se comparten SCLK, MISO y MOSI entre esclavos, y el CS define con cual de ellos el *Master* se comunicará en ese momento. Cuando el CS1 está a nivel alto toda la información del bus es inexistente para el *Slave1*, omitiendo su lectura y respuesta.

La transferencia de datos no está limitada a bloques de 8 bits, aunque suele ser lo habitual. Es posible la comunicación en bloques de 16bits, 24bits, etc. Y su implementación hardware es muy sencilla, dado que no requiere de circuitos para hacerla funcionar. Los esclavos no necesitan de su propio reloj puesto que lo proporciona el *Master*.

5.1.1 Lógica tri-estado

En electrónica digital, la lógica tri-estado permite a un puerto de salida asumir un estado de alta impedancia, lo que le permite permancer lógicamente desconectado del circuito cuando no transmite información.

Para que varios esclavos puedan comunicarse full dúplex con el mismo Maestro es necesario que tengan salidas tri-estado, para que su MISO se convierta en alta impedancia cuando comunica otro de los esclavos. Basta con que uno de los esclavos no tenga salida de alta impedancia, para corromper la información transmitida por cualquier esclavo comunicando, y la comunicación sólo podría realizarse de Maestro a Esclavo.

5.2 Comunicación con dispositivos

En este apartado se explicará exactamente como se ha realizado la comunicación con cada dispositivo, aunque que existan continuas alusiones al código, el código completo usado se mostrará y explicará en la sección 8. La mayoría de la información aquí recopilada está sacada o interpretada de las hojas de datos de los fabricantes sin ser de mi pertenencia ni descubrimiento.

La comunicación con los dispositivos la realiza el MSP430G2553 como *Master*, y se ha usado un osciloscopio del laboratorio de electrónica para facilitar la tarea de visualización en la emisión y recepción de datos. Sin embargo, y con objeto de facilitar las capturas de las transmisiones, las siguientes imágenes pertenecen a lo captado por un analizador lógico.

5.2.1 Comunicación con AD7490

Comunicación con el ADC utilizado para medir la tensión de las celdas y la temperatura.

5.2.1.1 Registro de control

El registro de control del AD7490 es un registro de 12 bits de sólo escritura. Los datos se cargan desde el pin DIN en el flanco de bajada de SCLK (CPOL=1) y se muestrean con adelanto de medio ciclo (CPHA=0). Los datos transferidos desde DIN corresponden a la configuración del dispositivo para la siguiente conversión. Se requiere de 16 pulsos de reloj para cada transferencia de datos, siendo sólo la información proporcionada en los primeros 12 flancos de bajada (después de la bajada del CS) la cargada en el registro de control. MSB (del inglés More Significant Bit) denota el primer bit del flujo de datos.

Tabla 5-1. Estructura del registro de control del AD7490

MSB											LSB
11	10	9	8	7	6	5	4	3	2	1	0
WRITE	SEQ	ADD3	ADD2	ADD1	ADD0	PM1	PM0	SHADOW	WEAK/$\overline{\text{TRI}}$	RANGE	CODING

Tabla 5–2. Funcion de los bits del registro de control del AD7490

Bit	Nombre	Descripción
11	WRITE	El valor de este bit determina si los siguientes 11 bits se cargan en el registro de control o no. Si es 1 los siguientes 11 bits se escriben en el registro de control, si es 0 no lo hacen.
10	SEQ	El valor del bit SEQ se utiliza junto con el bit SHADOW para controlar la función del secuenciador y el registro SHADOW.
9 a 6	ADD3 hasta ADD0	Estos cuatro bits de dirección seleccionan cual de los 16 canales disponibles será convertido en la siguiente tranmisión en serie. Su codificación es binaria, por lo que 0111 seleccionaría el canal 7 por ejemplo.
5, 4	PM1, PM0	Estos dos bits gestionan la potencia y el modo de operación del AD7490.
3	SHADOW	El valor del bit SHADOW se utiliza junto con el bit SEQ para controlar la función del secuenciador y el registro SHADOW
2	WEAK/TRI	Este bit selecciona el estado de DOUT al final de la transmisión. Si se establece en 1, DOUT es conducido débilmente al bit de dirección del canal ADD3 de la conversión venidera. Si este bit se pone a 0, DOUT regresa a tri-estado al final de la transferencia.
1	RANGE	Este bit selecciona el rango de entrada analógica del ADC. Si su valor es igual a 0, el rango analógico de entrada se extiende de 0 a $2\ x\ REF_{IN}$. Si se pone a 1, el rango queda fijado entre 0 y REF_{IN} en la siguiente conversión.
0	CODING	Este bit especifica el tipo de codificación a la salida del ADC. Si el bit =1 la conversión se realizará en binario directamente. Si el bit =0 la conversión se realizará en dos complementos.

En nuestras transmisiones el bit WRITE permanecerá siempre a 1, puesto que no exite situación en la que no nos interese cargar el restro de bits en el registro de control.

Los bits de SEQ y SHADOW son programados basándonos de nuevo en el datasheet, en una de las tablas especifica que si SEQ = 0 y SHADOW = 0 el secuenciador no es usado, y el funcionamiento del ADC es el de un ADC multicanal normal, que es el requerido en nuestro caso. El uso del secuenciador y el registro SHADOW es algo complejo y se va a mantener fuera del objeto de este TFG.

Tabla 5–3. Configuración del modo de operación del AD7490

PM0	PM1	Modo
1	1	Operación normal. En este modo el AD7490 permanece a máxima potencia permitiendo la máxima velocidad de transmisión.
1	0	Apagado completo. En este modo el AD7490 permanece apagado con toda su circuitería, aunque mantiene la información en el registro de control. Permanece apagado hasta que cambia el modo de operación.
0	1	Apagado automático. En este modo el AD7490 entra en modo apagado después de cada conversión. El tiempo para reactivarse es 1µs, el usuario debe manejar la gestión del tiempo para considerar la reactivación.
0	0	Auto Standby. En este modo algunas partes del AD7490 permanecen apagadas. Es similar a apagado automático, pero requiere un ciclo dummy para encenderse.

La inclusión de distintos modos de operación otorga una alta versatilidad al ADC en operaciones de ultra bajo consumo, aunque en nuestro caso no sean especialmente necesarias. En el laboratorio se hicieron pruebas del consumo de los cuatro modos de operación, el consumo en el modo de operación normal era minúsculo comparado con el resto de ICs de la PCB, del orden de 12mW, un consumo asumible dadas las características de alimentación, por lo que los bits PM1 y PM0 permanecerán a 1 en todas las operaciones.

El bit WEAK/TRI permanecerá a 0 también en todas las operaciones producidas con el ADC por los motivos comentados en el apartado 5.1.1.

El rango de entrada como ya sabemos debe cubrir las necesidades de los AD y de los sensores de temperatura. En el primer caso, la tensión oscila entre 2.5V y 4.2V, con lo cual no hay lugar a discusión y en el ADC de medida de voltajes el bit RANGE se mantendrá siempre a 0. En el caso de los sensores de temperatura, y según la figura 3-5 el rango de salida de los sensores puede variar entre 0.5 y 3.3V, con lo cual el bit RANGE también tomará el valor 0 en las transmisiones.

Por último, la codificación de salida puede ser *straight binary* (a lo que he otorgado anteriormente la traducción de directamente binario), que significa que la salida se da en un numero binario que aumentará linealmente entre 0 y $2xREF_{IN}/4096$ (aunque esto último depende de lo elegido en el bit de CODING) como muestra la gráfica en el propio datasheet [6].

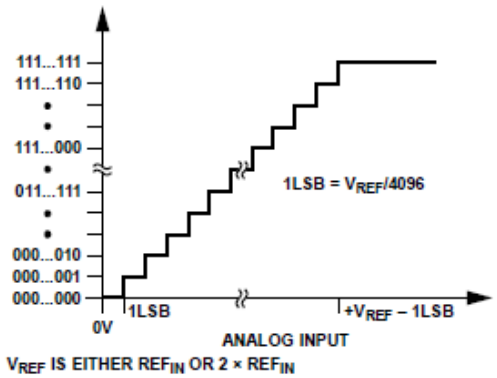

Figura 5-4. Codificación *Straight Binary*

Aunque también puede ser la codificación en dos complementos, siendo el 0 el valor de mitad de escala. El final de escala superior 011111111111 y el final de escala inferior 100000000000. Esta codificación se suele utilizar cuando la señal de entrada es bipolar, tomando la mitad superior para valores positivos y la mitad inferior de la escala para valores negativos. Al igual que antes, en el datasheet proporicionan una gráfica para ilustrarlo.

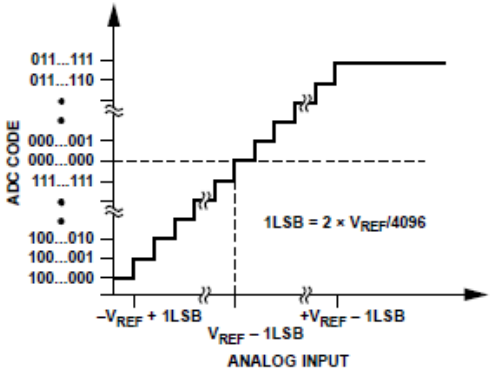

Figura 5-5. Codificación en dos complementos

Para simplificar, y dado que las tensiones de entrada son siempre positivas, el bit de coding permanecerá siempre en 1.

Recopilando todo lo dicho anteriormente; exceptuando los bits de dirección, todos los bits del registro de control mantendrán la siguiente configuración en todas las transmisiones que se vayan a realizar:

Tabla 5-4. Configuración del registro de control del AD7490

MSB											LSB
1	0					1	1	0	0	0	1
WRITE	SEQ	ADD3	ADD2	ADD1	ADD0	PM1	PM0	SHADOW	WEAK/$\overline{\text{TRI}}$	RANGE	CODING

5.2.1.2 Transmisión de datos

La respuesta a los datos enviados por el µC consiste en cuatro bits de dirección del canal que se está leyendo (que nos permiten asegurar de donde viene la información) seguidos por los 12 bits de la conversión.

Una vez cargados los datos en el registro de control después de la primera transmisión, se efectúan los cambios de la configuración en la siguiente, lo que provoca un desfase entre la petición de información y la recepción de la misma. En nuestro caso, y dado que la información se va a pedir circularmente (desde la primera celda a la decimosexta y después a la primera de nuevo), desfasando el vector de recepción de datos en un valor conseguiremos cuadrar la información solicitada y recibida, aunque esto se verá con detalle en el apartado de código.

A continuación, se muestra un ejemplo de comunicación. Dado que el µC sólo permite la recepción de 8 bits en 8 bits por el tamaño de RXBUF, las comunicaciones de 16 bits se parten en dos transmisiones de un byte.

En esta transmisión el µC envía 0xAF y 0x1F, o lo que es lo mismo, 10[1011]1100011111. Siendo los últimos 4 bits despeciables y los que están entre corchetes la dirección del canal a leer. Y se recibe la dirección del canal a leer (CH11) seguido del resultado de la conversión: 110000000111, en decimal 3079. Considerando que cada LSB son 1.22mV, el valor de tensión leído en el canal 11 en ese momento fue 3.7585V.

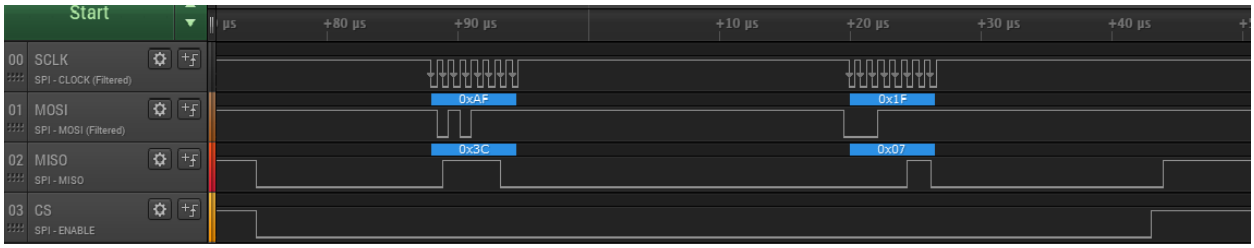

Figura 5-6. Ejemplo de transmisión completa con AD7490

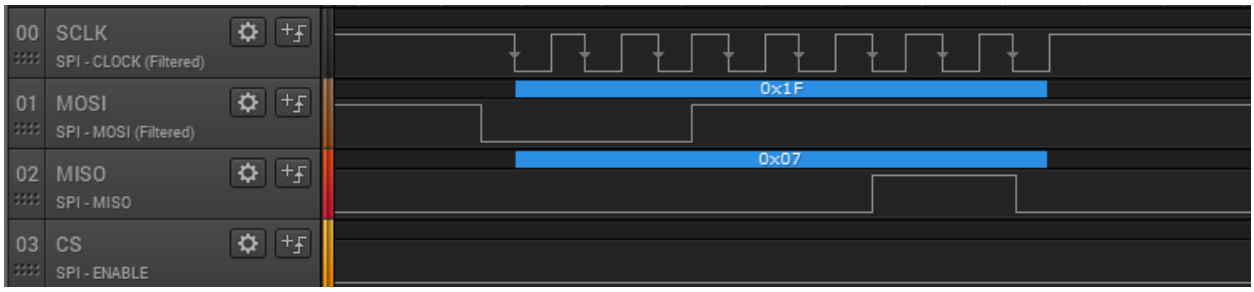

Figura 5-7. Detalle de un fragmento de transmisión con AD7490

5.2.2 Comunicación con MAX7317

La comunicación con el MAX7317 está diseñada para hacerse mediante el SPI de 4 vías descrito en el apartado 5.1. Sin embargo, y dado que el expansor no permanece en alta impedancia cuando no está transmitiendo información, el uso del MISO es incompatible con el de cualquier otro dispositivo de la PCB. Por eso y porque los puertos del expansor sólo se utilizarán como salidas sin ser necesaria la recogida de información en ningún momento, se ha optado por la eliminación del MISO del expansor, siendo así posible sólo la comunicación unidireccionalmente de µC a expansor.

La configuración del reloj que utiliza el MAX7317 se comprueba en los esquemas dados en la hoja de datos del fabricante [7], y se corresponde con CPOL=0 y CPHA=0.

5.2.2.1 Registros de control y operación

El MAX7313 contiene 10 registros internos que controlan los 10 puertos de E/S, con dirección entre 0x00 y 0x09. Dos direcciones 0x0E y 0x0F que no almacenan datos, pero devuelven el estado de ciertos puertos de entrada cuando son leidos. Cuatro direcciones virtuales que permiten a más de un registro ser escritos con las mismas órdenes para simplificar el software 0x0A hasta 0x0D. El registro RAM que proporciona 1 byte de memoria que puede ser usado para cualquier propósito. La dirección de no-op no causa efecto ninguno al escribirse o leerse, se suele utilizar como dummy.

Controlar el MAX7317 requiere mandar 16 pulsos de reloj. El primer byte, desde D15 hasta D8, es el byte de comando, que se encarga de elegir a quién va dirigida la información. El segundo byte, desde D7 hasta D0, es el byte de datos, que determina las acciones a realizar. La siguiente tabla del datasheet recopila todas las direcciones que pueden ser usadas en el byte de comando.

Tabla 5-5. Configuración del byte de comando del MAX7317

REGISTER	COMMAND ADDRESS								CODE (hex)
	D15	D14	D13	D12	D11	D10	D9	D8	
Port P0 output level	R/$\overline{W}$	0	0	0	0	0	0	0	0x00
Port P1 output level	R/$\overline{W}$	0	0	0	0	0	0	1	0x01
Port P2 output level	R/$\overline{W}$	0	0	0	0	0	1	0	0x02
Port P3 output level	R/$\overline{W}$	0	0	0	0	0	1	1	0x03
Port P4 output level	R/$\overline{W}$	0	0	0	0	1	0	0	0x04
Port P5 output level	R/$\overline{W}$	0	0	0	0	1	0	1	0x05
Port P6 output level	R/$\overline{W}$	0	0	0	0	1	1	0	0x06
Port P7 output level	R/$\overline{W}$	0	0	0	0	1	1	1	0x07
Port P8 output level	R/$\overline{W}$	0	0	0	1	0	0	0	0x08
Port P9 output level	R/$\overline{W}$	0	0	0	1	0	0	1	0x09
Write ports P0 through P9 with same output level	0	0	0	0	1	0	1	0	0x0A
Read port P0 output level	1								
Write ports P0 through P3 with same output level	0	0	0	0	1	0	1	1	0x0B
Read port P0 output level	1								
Write ports P4 through P7 with same output level	0	0	0	0	1	1	0	0	0x0C
Read port P4 output level	1								
Write ports P8 or P9 with same output level	0	0	0	0	1	1	0	1	0x0D
Read port P8 output level	1								
Read ports P7 through P0 inputs	1	0	0	0	1	1	1	0	0x0E
Read ports P9 and P8 inputs	1	0	0	0	1	1	1	1	0x0F
RAM	R/$\overline{W}$	0	0	1	0	0	1	1	0x13
No-op	R/$\overline{W}$	0	1	0	0	0	0	0	0x20
Factory reserved; do not write to this register	R/$\overline{W}$	1	1	1	1	1	0	1	0x7D

En nuestro caso, dado que el MISO no está conectado y no necesitamos leer el valor de ningún puerto, el bit D15 estará siempre a 0 para sólo escritura.

Respecto al byte de datos, y para la aplicación que nosotros necesitamos, simplemente bastará con permutar entre "open drain logic-low" y "open drain logic-high", escribiendo 0x00 o 0x01 respectivamente. Cuando el un puerto está en "open drain logic-high", el MOSFET conduce y cierra el circuito de balanceo de la figura 3-11. En la siguiente figura se muestran todos los posibles estados de salida:

Tabla 5-6. Formato de los registros de salida del MAX7317

Port P1 level	—	0x01	MSB	Port P1 level	LSB	0x00 or 0x01
Port P2 level	—	0x02	MSB	Port P2 level	LSB	
Port P3 level	—	0x03	MSB	Port P3 level	LSB	
Port P4 level	—	0x04	MSB	Port P4 level	LSB	
Port P5 level	—	0x05	MSB	Port P5 level	LSB	
Port P6 level	—	0x06	MSB	Port P6 level	LSB	
Port P7 level	—	0x07	MSB	Port P7 level	LSB	
Port P8 level	—	0x08	MSB	Port P8 level	LSB	
Port P9 level	—	0x09	MSB	Port P9 level	LSB	
Writes ports P0 through P9 with same level	0	0x0A	MSB	Ports P0 through P9 level	LSB	
Reads port P0 level	1		MSB	Port P0 level	LSB	
Writes ports P0 through P3 with same level	0	0x0B	MSB	Ports P0 through P3 level	LSB	
Reads port P0 level	1		MSB	Port P0 level	LSB	
Writes ports P4 through P7 with same level	0	0x0C	MSB	Ports P4 through P7 level	LSB	
Reads port P4 level	1		MSB	Port P4 level	LSB	
Write ports P8 and P9 with same level	0	0x0D	MSB	Ports P8, P9 level	LSB	
Read port P8 level	1		MSB	Port P8 level	LSB	

Sólo usaremos las órdenes de dirección de cada puerto individual (0x00 – 0x09) cada vez que queramos activar el circuito de balanceo de una celda y la escritura de todos los puestos iguales 0x0A para hacer un reset de todos los puertos.

Dado que no es necesaria la recepción de datos y no usaremos el buffer de repección de datos RXBUF, podemos hacer la transmisión en 16 bits. En el ejemplo expuesto, permutamos el valor del puerto 9 del expansor entre nivel alto y bajo (balanceando y sin balancear) 0x0900 y 0x0901.

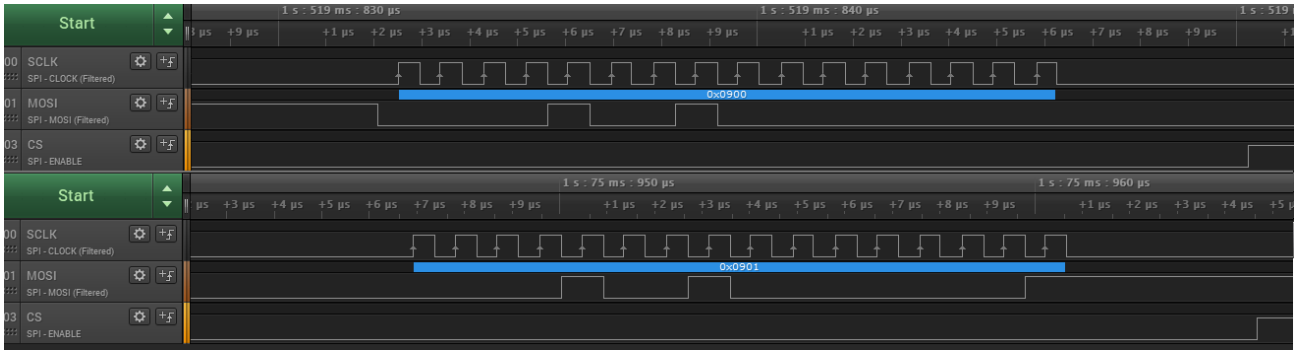

Figura 5-8. Ejemplo de transmisión con el MAX7317

5.2.3 Comunicación con AD7988-5

El AD7988 tiene diferentes modos de funcionamiento y, por tanto, de comunicación. El primero de ellos es el modo CS de tres vías, que está orientado a la comunicación de un sólo AD7988 con un host o *Master* compatible con SPI. El segundo modo es CS de cuatro vías, orientado a la conexión con un host compatible con SPI con múltiples AD7988. Por último, el modo Daisy Chain, que permite conectar varios AD7988 en cascada en un bus de sólo tres vías uniendo los pines SDO entre sí.

Lógicamente, usaremos el AD7988 en el modo CS de tres cables, dado que únicamente usaremos uno en la PCB del Master. Para ello el fabricante nos exige conectar el pin SDI a VIO [8], anulando el MOSI del bus SPI, que al ser monocanal, es completamente prescindible. La transmisión de información por tanto sólo se realizará de esclavo a maestro.

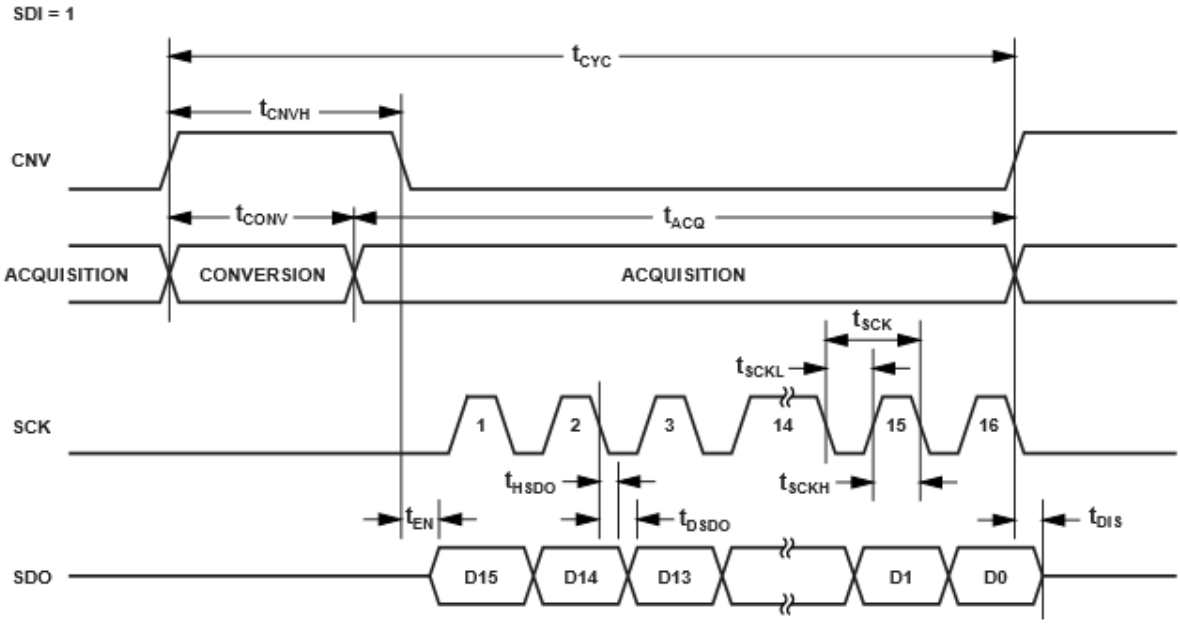

Figura 5-9. Configuración de la comunicación del AD7988

La comunicación con el ADC necesita de un flanco de subida en el CS para el comienzo del periodo de conversión, entonces se selecciona el modo CS y fuerza a SDO a un estado de alta impedancia. Cuando la conversión finaliza (marcada por el tiempo de conversión, que es fijo) entra en la fase de adquisición y se apaga.

Con el flanco de bajada del CS, el AD7988 sale del modo de alta impedancia y el MSB producto de la conversión se carga en SDO, los restantes bits de datos se cargan en los siguientes flancos de bajada del SCK (CPHA=1 y CPOL=0). Cuando se termina la transmisión de los 16 bits o cuando el CS vuelve a nivel alto (lo que ocurra antes) el SDO vuelve al estado de alta impedancia. Los 16 bits que se transmiten al Master son los 16 bits producto de la conversión anterior. Durante la transmisión de datos al SDO se produce la fase de adquisición de la siguiente transmisión.

El AD7988 posee dos variantes según la velocidad de conversión, el AD7988-1 ofrece 100kSPS mientras que el AD7988-5 ofrece 500kSPS, por lo que se ha elegido este último para obtener un tiempo de conversión de 1.6µs.

5.3 Comunicación entre µCs

Para explicar este apartado me basaré en código implementado durante las pruebas de comunicación entre dos MSP430G2553, explicando primero de una forma general los detalles de la comunicación y posteriormente la codificación de la misma.

5.3.1 Código de la comunicación

```
//Configuracion del puerto SPI del MASTER
    P1OUT |= BIT5; //CSn
    P1DIR |= BIT5;
    P1OUT |= BIT6; //LED
    P1DIR |= BIT6;
    P1SEL = BIT1 | BIT2 | BIT4; //P1.1=MISO, P1.2=MOSI, P1.4=SCLK
    P1SEL2 = BIT1 | BIT2 | BIT4; //P1SEL=11
    UCA0CTL1 = UCSWRST;
    UCA0CTL0 |= UCMSB + UCMST + UCSYNC + UCMODE_0; //UCMSB (More Significant Bit) Y
UCMST (modo MASTER), UCSYNC (Transmisión síncrona del SPI), UCMODEx (00-> 3pin/01 ->
4pin UCxSTE high/10-> 4pin UCxSTElow), UCCKPH (fase=0), UCCKPL (polaridad=1)

    UCA0CTL1 |= UCSSEL_2; // SMCLK master mode, ver en el regristro UCA0CTL0
    UCA0BR0 |= 0x01; // /1 mhz
    UCA0BR1 = 0; //
    UCA0MCTL = 0; // No modulation
    UCA0CTL1 &= ~UCSWRST;
```

Con este trozo de código declarado en el main, configuramos el puerto SPI de la USCI_A del *Master* para transmitir información a 1 MHz con CPOL=0 y CPHA=1, que ha sido la codificación que he elegido, el MSP no puede actuar como Slave con CPHA=0 por un bug del microcontrolador.

La configuración del µC como *Slave* es similar y varía sólo en lo siguiente:

```
    UCA0CTL0 |= UCMSB + UCSYNC + UCMODE_0;
    UCA0CTL1 |= UCSSEL_0;
```

Manteniéndose el resto de línas igual, el único cambio es la deselección del modo *Master* en el registro UCA0CTL0 y la modificación del modo de reloj (si entrante o saliente) en el registro UCA0CTL1.

Una vez configurado cada microcontrolador, pasamos a la comunicación del *Master*. Para él, la comunicación con el *Slave* es exactamente igual que con cualquier otro dispositivo, decide cuándo bajar el CS, introduce los primeros 8 ciclos del reloj a la vez que la codificación del primer byte del MOSI y recibe el primer byte de MISO, cuando lo completa, hace lo mismo con los siguientes 8 bits del reloj, introduce el segundo byte de MISO y recibe el segundo byte de MOSI. Como se ejemplifica en el siguiente fragmento de código:

```
P1OUT &= (~BIT5); // Se baja el Chip Select

 while (!(IFG2 & UCA0TXIFG)); // Se espera a que el buffer de transmisión esté libre
 UCA0TXBUF = 0x09; // Se transmite el primer byte de dato
 while (!(IFG2 & UCA0RXIFG)); // Se espera a que el buffer de recepción esté libre
 received_ch1 = UCA0RXBUF; // Se recibe el primer byte de dato

 while (!(IFG2 & UCA0TXIFG));
 UCA0TXBUF = 0x43; // Se transmite el segundo byte de dato
 while (!(IFG2 & UCA0RXIFG));
 received_ch2 = UCA0RXBUF; // Se recibe el segundo byte de dato

 aux1=received_ch1;
 aux2=received_ch2;

P1OUT |= (BIT5);
```

La información recogida por received_ch1 y received_ch2 se guarda en variables tipo char, que se traspasan a aux1 y aux2 para convertirlas a unsigned int (que normalmente nos permite dos bytes de almacenamiento). Teniendo los datos recibidos y almacenados, sólo queda unirlos e interpretarlos. Para unir los datos se utilizan los operadores de desplazamiento de bits:

```
dato = aux1 << 8;
dato = dato + aux2;
```

Debido a que en la configuración se ha especificado primero MSB, y que el *Slave* manda primero el MSB de la cadena de 16 bits, desplazamos el primer byte recibido 8 bits a la izquierda, dejando libre los 8 espacios que ocupará el segundo byte cuando se sumen. En 'dato' quedaría en teoría recogido lo que el Slave quería transmitir, sin embargo, la tranmisión del Slave al *Master* se retrasa un byte, es decir, si el *Master* envía por ejemplo 0x09 y 0x43 y el Slave estuviese configurado para devolver lo mismo que recibe, en la siguiente transmisión el *Slave* devolvería 0xFF (basura o desconocido) y 0x09, necesitando una tranmisión más para dar el byte restante. Este problema es fácilmente solucionable adelantando en un byte el vector de recepción de datos del *Master*, como se hace en el código final:

```
if (i==0)
 {
      recibido[i]=aux2;//La primera coordenada, al tener estar desfasada se desprecia
 }
else
 {
      recibido[i-1]= recibido[i-1]<<8 + aux1;
      recibido[i]=aux2;
 }
```

La interpretación del dato se realiza según la codificación elegida, y se explicará más adelante en el subapartado 5.3.2.

El *Slave* tiene ligeras pero importantes modificaciones en el código de recepción respecto al *Master*, que se muestran a continuación:

```
if((P1IN & BIT5) ==0) //si P1.5=1
{
	__delay_cycles(12.5); //Retraso entre CS y primer flanco del reloj

	while (!(IFG2 & UCA0TXIFG));
	UCA0TXBUF = transMSB;
	while (!(IFG2 & UCA0RXIFG));
	received_ch1 = UCA0RXBUF;

	recibido=received_ch1;

	__delay_cycles(25); //Delay intermedio entre primer byte y segundo

	while (!(IFG2 & UCA0TXIFG));
	 UCA0TXBUF = transLSB;
	while (!(IFG2 & UCA0RXIFG));
	received_ch2 = UCA0RXBUF;
	recibido2=received_ch2;

	while(!(P1IN & BIT5)); //Espera a acabar la transmision
```

Como se observa, es necesario esperar a la entrada del CS que proviene del *Master,* esto se puede hacer mediante interrupciones, tal y como se ha hecho en el código final, pero es menos ilustrativo.

También es necesario comprobar y programar el retraso entre el flanco de bajada del CS y el comienzo de la transmisión, sino habrá desfases y fallos en la transmisión, lo mismo entre el primer byte del reloj y el segundo. Como el desfase sólo afecta a la comunicación del *Slave* al *Master*, para recibir los datos del mismo sólo es necesario:

```
dato = recibido << 8;
dato = dato + recibido2;
```

Se puede comprobar que la transmisión al *Master* se hace mediante dos variables, esto se debe a que, según los comandos enviados por el Master, un Slave cualquiera podría responder un dato u otro, que se modifican posteriormente a la transmisión.

Por último, se espera a finalizar la transmisión (subida del CS) con un bucle vacío para evitar que se vuelva a entrar en el bucle en el tiempo que el CS se mantiene a nivel bajo al acabar la transmisión.

5.3.2 Codificación

En nuestro caso, y debido a que el *Slave* debe mandar siempre la misma información, es más cómodo que no sea necesario para el *Master* solicitarle dicha información, es decir, que el *Slave* mande siempre la misma información actualizándose y el *Master* haga lo mismo con la información que necesita el *Slave*.

El *Master* necesita de cada *Slave* las temperaturas y las tensiones de cada una de sus 16 celdas, mientras que el *Slave* necesita conocer la tensión mínima del pack completo de baterías y el estado de carga, es decir, si se está cargando o no la batería para poder balancear.

Cada una de las temperaturas y tensiones se transmite directamente como es leida a su ADC (eliminando los 4 bits de address), siendo el *Maste*r el encargado de interpretarlas, por lo que cada uno de los 32 valores necesitará 12 bits para transmitirse. Por comodidad se ha optado por hacer transmisiones de 16 bits, dejando los primeros 4 bits de cada transmisión en blanco, por si fuese necesario más adelante transmitir también su dirección. Eso hace como mínimo 32 transmisiones necesarias que, contando el desfase de 1 byte de la comunicación, ascienden al total de 33 transmisiones. Dado que el *Master* sólo necesita enviar un dato de 12 bits y un estado, durante todas las transmisiones mandará lo mismo al *Slave*, codificado con los primeros 12 LSB ocupados por Vmin, y el decimotercer bit ocupado por el estado de carga o descarga de la batería, para ello el *Master* suma 4096 al dato de Vmin que fuese a transmitir.

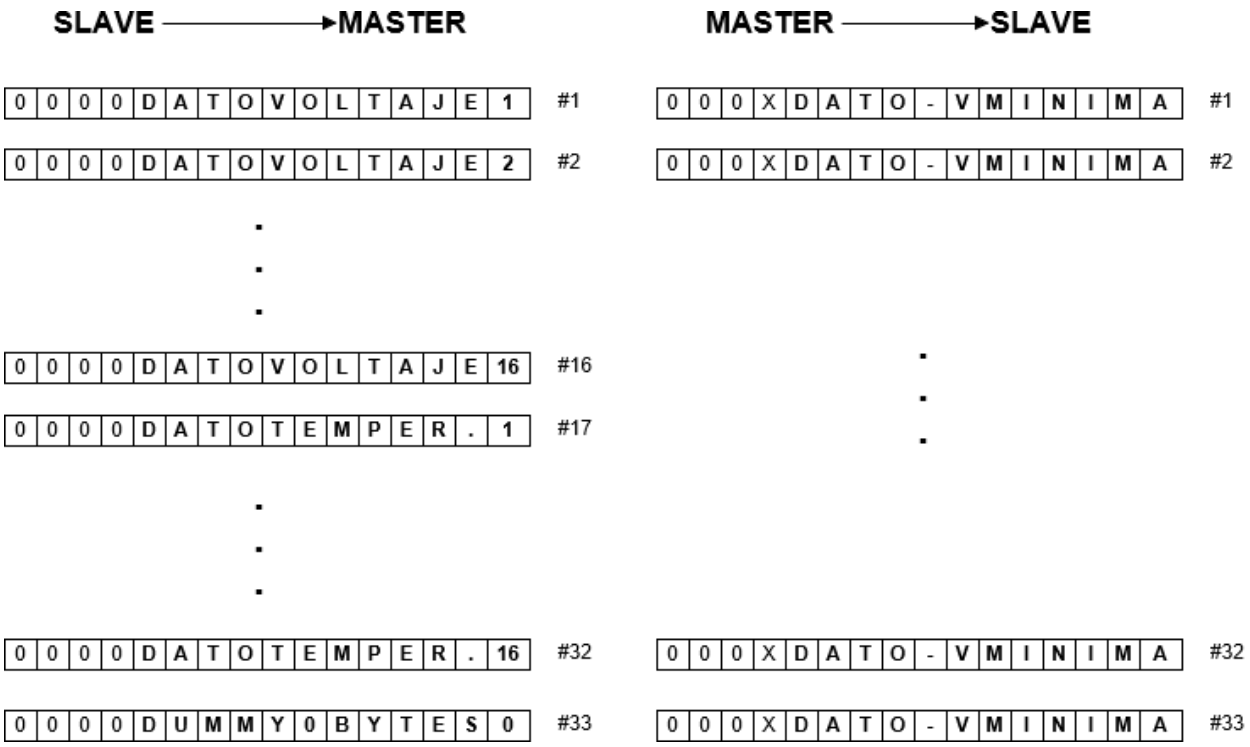

Figura 5-10. Codificación de la comunicación entre μCs

La programación necesaria para la comunicación es igual a la antes expuesta, con ligeras variaciones. Por ejemplo, en el *Master* para transmitir 33 veces con cada Slave se han utilizando bucles con el código de la comunicación, almacenando los datos recibidos en un vector de unsigned int adelantado como se explicó anteriormente. El bucle está contenido por el CS, para evitar bajarlo y subirlo (y sus correspondientes retrasos) cada transmisión de 16 bits.

El *Slave* para recibir los datos utiliza interrupciones, activadas con el cambio de nivel alto a nivel bajo de su CS. Utiliza bucles para las 33 transmisiones y al acabarlas todas interpreta el estado de carga de la batería.

6 FABRICACIÓN

A Pesar de las numerosas empresas que se dedican a la fabricación de PCBs y los costes cada vez más reducidos, la fabricación de todo el material necesario para este proyecto se ha realizado en el laboratorio de la Universidad de Sevilla.

6.1 Fabricación de PCBs

El método de fabricación que se sigue en el laboratorio usa la insoladora de rayos UV. Lo primero que necesitamos es imprimir la PCB que vayamos a fabricar en papel vegetal o papel cebolla mediante una impresora láser. Si la PCB es de dos caras, se deben imprimir separadas.

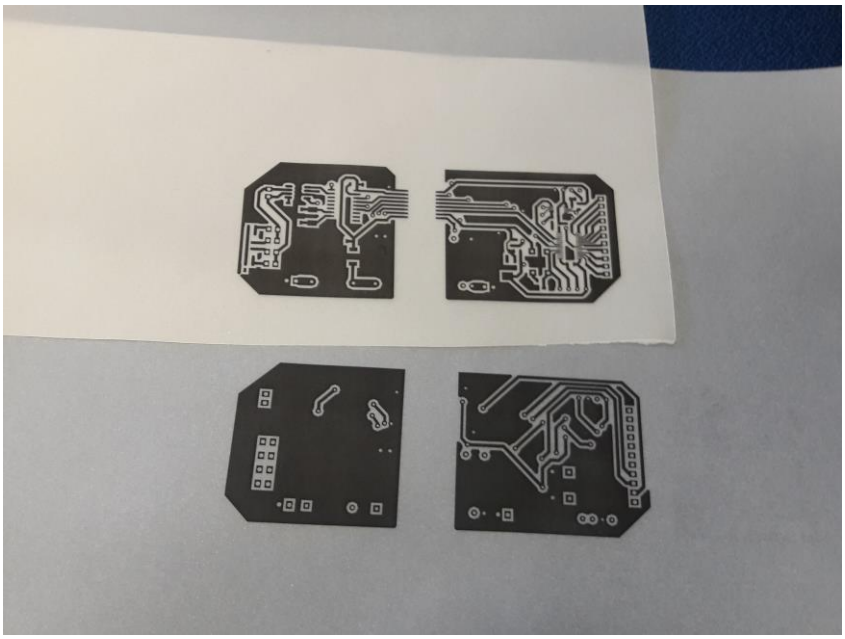

Figura 6-1. Caras de la PCB impresas en papel vegetal

Acto seguido, y ahora que disponemos de las dimensiones reales del espacio que ocupa la PCB, superponemos el impreso recortado sobre una placa de cobre virgen con emulsionado positivo de una o dos caras, sin retirar el precinto que protege el cobre. Recortamos con la guillotina del laboratorio de fabricación la placa de cobre para ajustarla al tamaño de la PCB, siempre con un reborde como margen de seguridad.

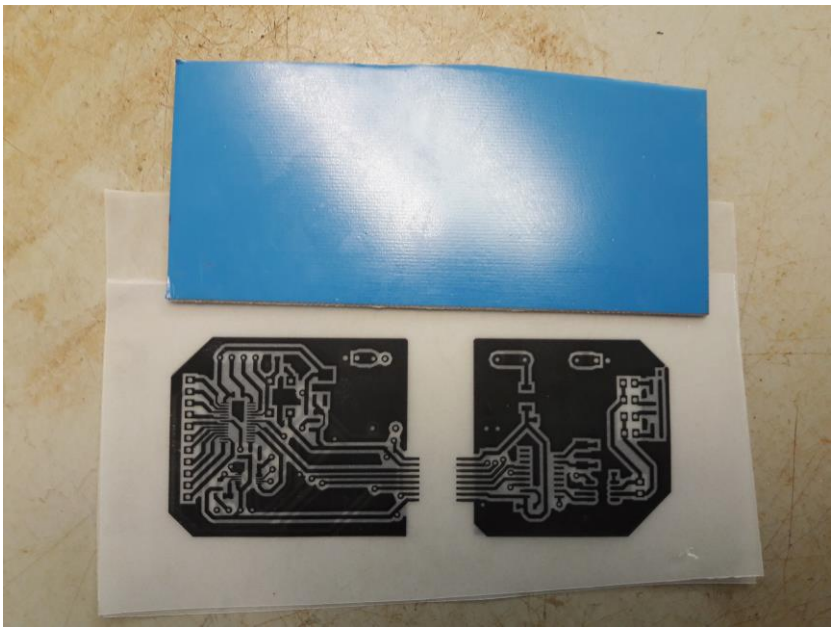

Figura 6-2. Placa virgen recortada

Se retiran los precintos protectores con cuidado de no dejar huellas sobre la superficie, dado que puede afectar a la fabricación la grasa o el sudor de las manos. Si la placa es de dos caras, se superpone en cada cara el dibujo de la PCB con sumo cuidado de que ambas queden exactamente en la misma posición, se puede hacer más fácilmente uniéndolas previamente con celo e introduciendo la placa entre cada cara. Si por otro lado la PCB sólo tiene una cara, con superponer el dibujo sobre el área de la placa es más que suficiente. Posteriormente se introduce la placa con el papel superpuesto en la insoladora.

Una insoladora es una caja que contiene en su interior una fuente de luz de tubos fluorescentes actinicos o leds de luz UV, que irradian luz ultravioleta sobre una placa de fibra de vidrio o baquelita apoyada sobre un cristal. Nos permite debilitar el barniz sensible a la luz UV que recubre la superficie de cobre de las placas. Al introducir las placas con el papel superpuesto en la insoladora, evitamos que en las zonas de dibujo de la PCB el barniz se debilite, permitiéndonos atacar las zonas de cobre sobrante con mayor rapidez.

Figura 6-3. Insoladora en funcionamiento

Al acabar el proceso de irradiación, se introduce la placa en un líquido revelador durante aproximadamente 60 segundos, que eliminará todo el barniz sobrante tras la insolación, quedando expuesto todo el cobre no necesario que será destruido en la fase de quemado. Tal y como se observa en la primera imagen de la figura 6-4.

Cuando hayamos acabado con el revelador, es muy importante lavar con agua la placa, quitando cualquier posible resto de revelador. Hecho esto, prepararemos una solución en una probeta a base de agua (H_2O), ácido clorhídrico (HCl) diluido al 20% y agua oxigenada (H_2O_2) , en proporción volumétrica 2, 1 y 1 respectivamente. Por cada volumen introducido de ácido clorhídrico, se introduce el mismo de agua oxigenada y el doble de agua normal. La cantidad dependerá del tamaño de la placa que se fuese a atacar, una cifra orientativa puede ser 20mL de ácido.

Una vez preparada la solución, la introducimos en un recipiente con la placa y dejamos que el ácido ataque el cobre sobrante. El tiempo necesario depende normalmente de la cantidad de ácido de la solución, siendo estrictamente necesaria una vigilancia constante del proceso, pues la solución puede neutralizar el cobre que formará nuestra PCB si se descuida.

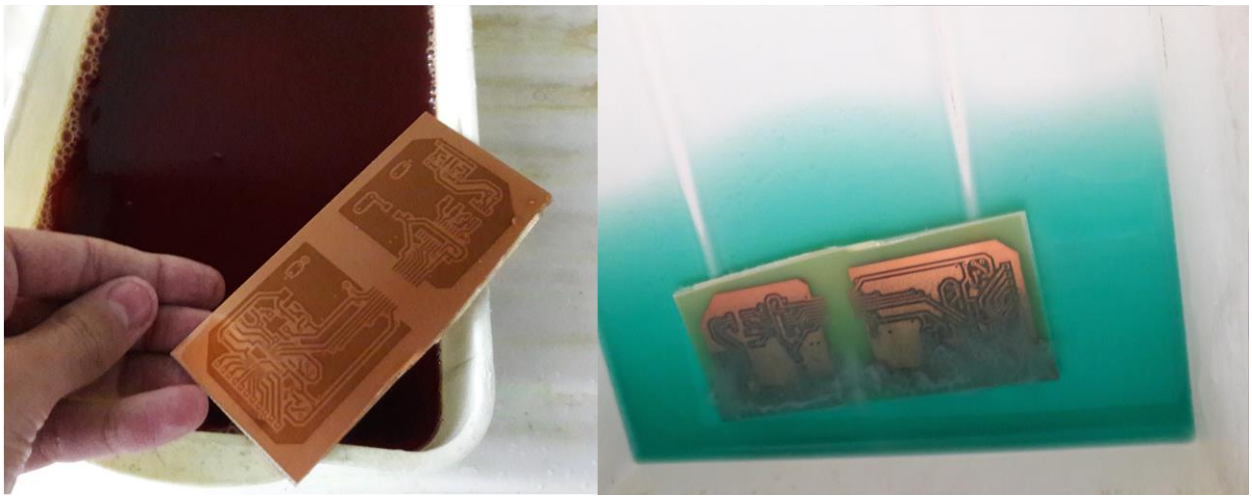

Figura 6-4. PCB durante el proceso de revelado y quemado

Acabado el quemado, se vuelve a lavar la placa con agua, se seca y se le aplica quitaesmalte, para eliminar los restos de barniz que pudiesen quedar sobre la PCB. Una vez seca y limpia, es recomendable comprobar la continuidad en las pistas, así como posibles cortocircuitos entre las mismas. Habrá que estañar si fuese necesario las pistas discontínuas y separar con una herramienta de corte (como un cuter o un taladro) las pistas unidas por algún resto.

Por último, se realizan todas las perforaciones de vías y conectores con el taladro del laboratorio y la broca adecuada para cada agujero. Se puede aplicar quitaesmalte después de hacer todos los taladros para limpiar los restos de polvo. La PCB ya estaría preparada para que le fuesen soldados todos los componentes.

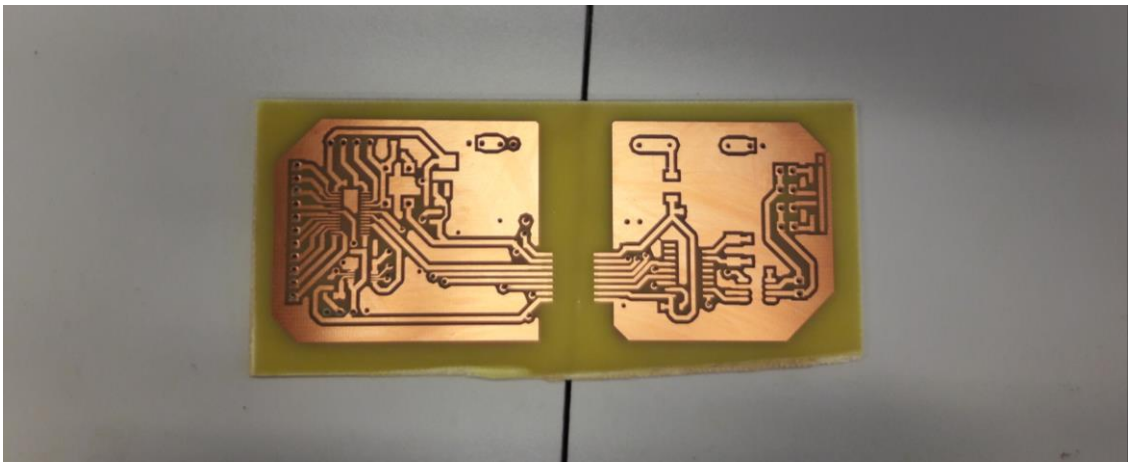

Figura 6-5. Acabado final de la PCB

6.2 Batería de pruebas

Para el testeo del BMS y probar que el código y el hardware funcionaran correctamente se fabricó posteriormente al segundo prototipo (ver apartado 7 de Evolución de prototipos) una batería de 8 celdas en serie de ion-litio. Cada una de las celdas tiene incorporada una protección contra sobretensión o contra descarga, que permite un uso seguro mientras se calibra el código. Las baterías me las proporcionó mi tutor Alfredo, y aunque no se disponga de demasiada información de ellas, son suficientes para simular procesos de carga y descarga y aprender sobre el comportamiento de las propias celdas en dichos procesos, además de proporcionar tensiones reales para los canales de medida de voltaje, sin necesidad de usar varias fuentes de alimentación para simularlas.

Nueve cables de bajo amperaje fueron soldados a las celdas para acceder a las tensiones de cada una y dos cables gruesos a los extremos para cargar o descargar la batería. El negativo de la batería fue dividido en dos con un conector por motivos de seguridad, para evitar posibles cortocircuitos accidentales.

Aunque el prototipo final tenga 16 celdas de capacidad, el objeto de la batería de pruebas fue aprender el comportamiento de las celdas durante la carga y descarga para tener el primer programa de balanceo, que se realizó durante el segundo prototipo y que ha sido adaptado al último.

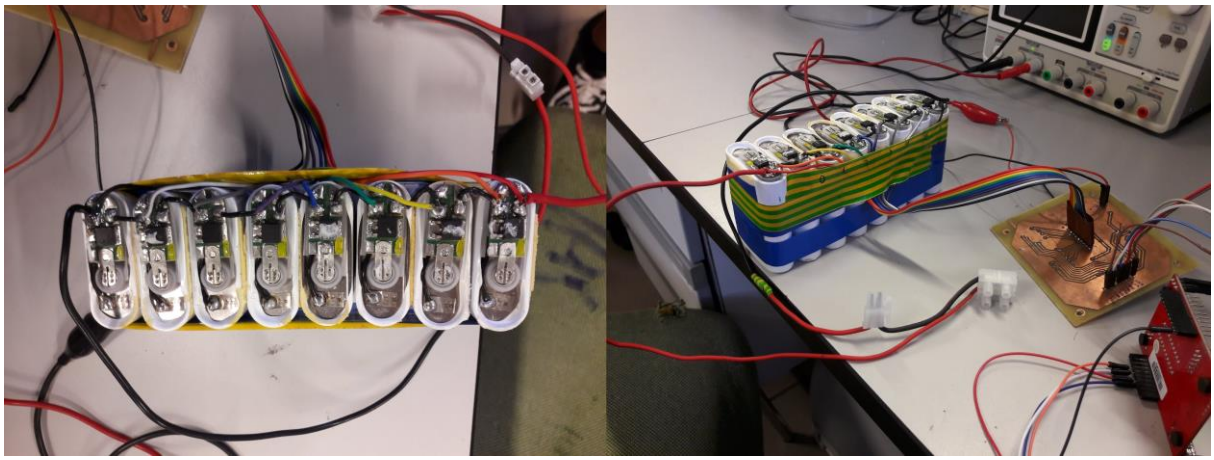

Figura 6-6. Batería de pruebas

7 EVOLUCIÓN DE PROTOTIPOS

PAra conseguir el resultado que se ha visto en este proyecto, ha sido necesaria una evolución y la adquisición de unos conocimientos y destrezas. Como anteriormente se ha mencionado, el aprendizaje se ha producido en base a prueba, ensayo y error en los prototipos que se han desarrollado. En este apartado se explicarán brevemente las correcciones producidas y las características de cada uno.

7.1 Primer prototipo de 8 celdas

Fue la primera toma de contacto con la instrumentación electrónica de forma práctica y el diseño electrónico propio, y fue el prototipo que más fallos ha albergado. No contaba con un Master, como ninguno de los prototipos que se expondrán excepto el definitivo. Sus únicas funciones fueron la monitorización de voltaje y el balanceo, sin contar si quiera con el microcontrolador integrado, la programación se realizó en el Launchpad y mediante dos puertos de salida con las señales del bus SPI se producía la comunicación con cada dispositivo, el MAX7317 y el AD7928.

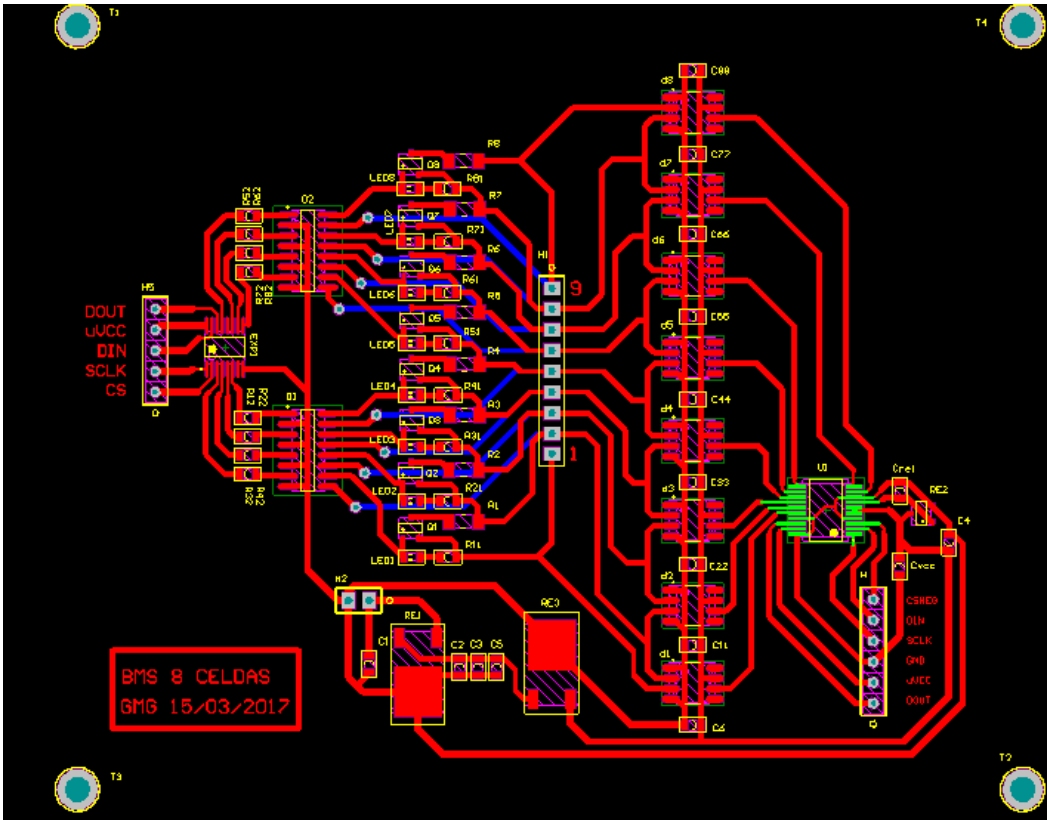

Figura 7-1. Primer prototipo de BMS

Como se observa en el enrutado, no había plano de tierra ni optimización del espacio y la PCB sólo tenia una cara. Los puertos SPI estaban separados, sólo había uno para cada dispositivo y las pistas de las señales eran demasiado finas. Los ADs no tenían referencia de voltaje que permitiese su funcionamiento y los condensadores utilizados en los reguladores (la placa se alimentaba hasta con 20V) no eran los de los valores que aconsejaba el fabricante. En ese momento se desconocía que el expansor de entradas/salidas presentaba su salida en drenador abierto, por lo que las salidas estaban conectadas directamente a los optoacopladores por una resistencia de 330Ω para limitar la posible corriente. El circuito usado en el balanceo no disponía de una resistencia en paralelo al led que permitiese que se descargarsen las corrientes parásitas del PMOS y por lo cual este nunca entraba en corte. El GND de la alimentación no estaba conectado al del microcontrolador, con lo cual la comunicación se hacía con tierra flotante. En general existía un gran desconocimiento del funcionamiento de los componentes de la placa, todos eran nuevos y incluso su elección fue complicada. El tamaño de la placa era de 11 x 12.7cm.

Sobre esta misma PCB se solventaron muchos de los problemas encontrados, soldando cables y desoldando componentes para probar nuevas conexiones y comprender al completo los errores encontrados, que produjo que el siguiente prototipo fuera un éxito.

7.2 Segundo prototipo de 8 celdas

El segundo prototipo ha sido uno de los más exitosos, y sobre el que más pruebas se han realizado. Las funciones que incluia eran exactamente las mismas que el anterior, pero solventando todos los problemas encontrados. Con las debidas correcciones y un enrutado mucho más pulido y cuidado se consiguió la reducción del tamaño del primero en un 50%, con inclusion del enrutado a dos caras y sus planos de tierra correspondientes.

Sobre esta PCB se desarrolló casi todo el código existente, al principio el código se programó sobre Energia, por su facilidad de uso y la cómoda interfaz. Se probó la comunicación con el AD7928 y se comprendió su funcionamiento, después se hizo lo mismo con el MAX7317 y por último, se creó el código que se comunicaba con los dos. En ese momento se descubrió la incompatibilidad del MISO del MAX7317 y se cortó la vía que lo unía con el resto del bus. Poco después se creó el primer programa de balanceo, que necesitó de muchas correcciones y calibraciones. Posteriormente y durante la fabricación del ultimo prototipo se traspasó y probó todo el código en Code Composer Studio sobre esta PCB.

Aún así, esta PCB albergaba pequeños fallos de diseño solventados por completo en el ultimo prototipo, como una resistencia de disipación de balanceo demasiado grande, que producía que la carga de la batería y el balanceo fuesen demasiado lentas, la ausencia del microcontrolador on board, alimentación mínima de 12V y la falta del resto de funciones de un BMS, como la medición de corriente y temperatura, que fueron las funciones que implementaba el siguiente prototipo.

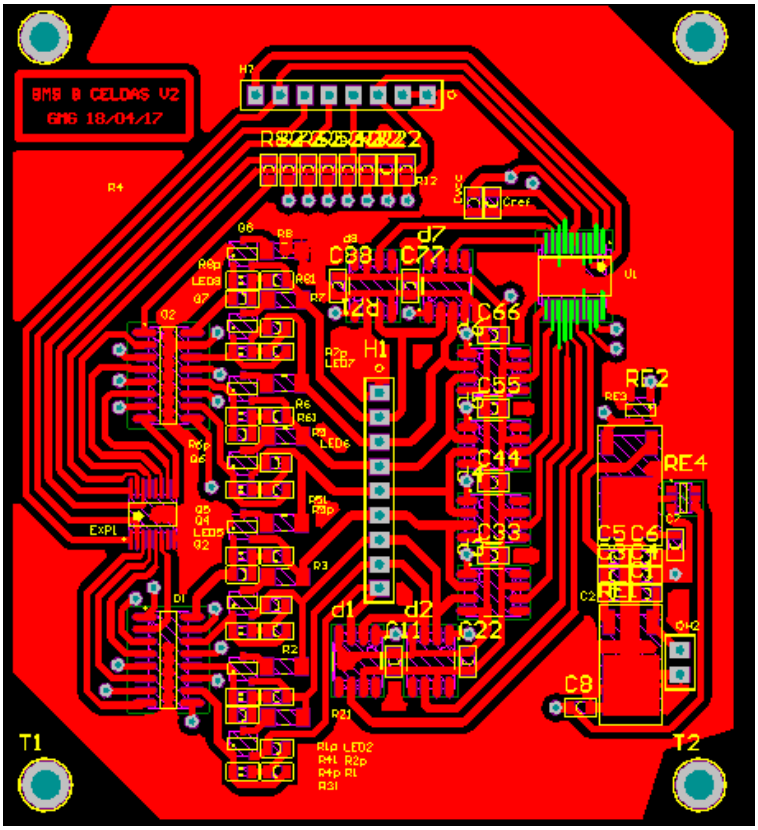

Figura 7-2. Segundo prototipo de BMS

7.3 Módulo de corriente y temperatura

El tercer prototipo o tercera PCB en fabricarse obtuvo el nombre de Módulo de Corriente y Temperatura, para complementar las funciones restantes del segundo prototipo. Como su propio nombre indica, incluía las funciones de medición de temperatura, medición de intensidad y incluía propiamente un relé de 12V para cortar la carga o descarga de la batería.

Coexistían dos entradas de PTC, seguidas por un filtro RC y con distinta resistencia en serie, para probar dos configuraciones de sensibilidad. Tras numerosas pruebas con cada configuración se descartaron como opción para el prototipo final, debido al coste tan alto que presentaba cada PT1000 y la precisión que mostraba no era significativamente mayor que los sensores de temperatura con salida de tensión analógica.

Para la medición de corriente se usó una resistencia de detección de corriente, y un amplificador de instrumentación en cascada con un amplificador diferencial. Las mediciones eran correctas, sin embargo, en el prototipo final, por pequeña que fuere la resistencia de detección, implicaría una potencia disipada, y bajo 300A esa potencia sería demasiado alta, por lo que no fue una alternativa real para el BMS final. La salida del AD oscilaba en torno a 2.5V de referencia según el sentido de la corriente, al igual que el sistema del último prototipo, lo que ha servido como familiarización conceptual y testeo.

Por último, un relé para cortar ante carga y descarga, que pese a que no es objeto del BMS, sino de la gestión de potencia del vehículo, fue de utilidad para gestionar esos procesos de forma autónoma.

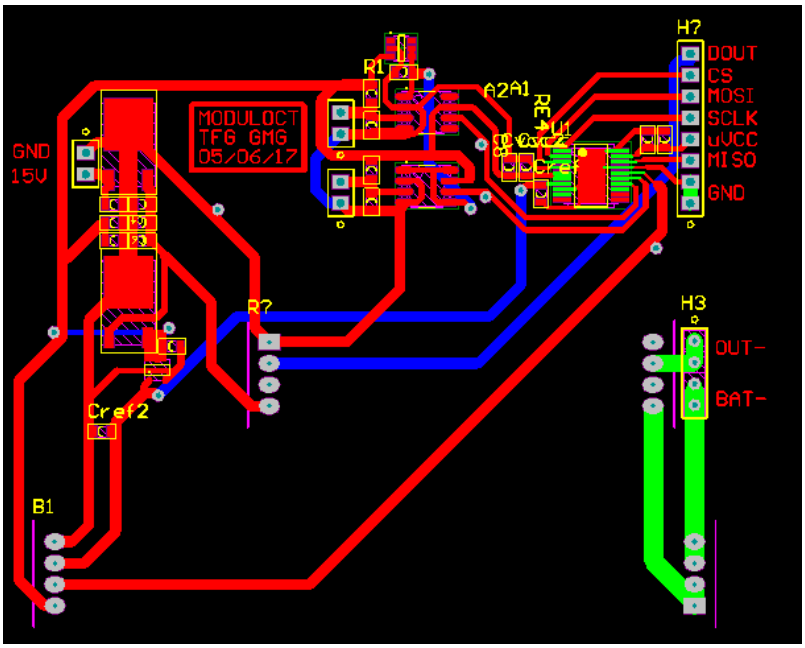

Figura 7-3. Módulo de corriente y temperatura

Pese a que no ha sido el prototipo que más haya aportado a la construcción del sistema final, si ha sido de utilidad para la familiarización con numerosos conceptos. Como se ha mencionado anteriormente, el aprendizaje durante este proyecto ha sido a base de prueba y error, y este prototipo ha servido para entender que opciones eran más viables.

7.4 Prototipo final de 16 celdas

Pese a que ya se ha descrito cada parte al detalle de este subsistema, conviene desctacar los cambios respecto a los anteriores prototipos.

- Introducción del microprocesador on board en ambas PCBs
- Introducción de la sensorización y medida de temperatura
- Ampliación de la capadidad a 16 celdas
- Cambio del modelo de los ADCs con 16 canales
- Alimentación definida por el Master
- Desarrollo del Master
- Sensor de corriente de efecto Hall
- Menor resistencia de disipación
- Aislamiento galvánico respecto a la alimentación
- Hardware de comunicación CAN
- Código completo

8 CÓDIGO

A Continuación se presenta todo el código diseñado para la implementación de las funciones del BMS, pese a que el código está cerrado y acabado, el objetivo de cara al próximo año es la continuación del desarrollo, calibración y testeo software, para pulir su funcionamiento.

8.1 Code Composer Studio

Code Composer Studio es el entorno que proporciona Texas Instruments para el desarrollo del código de sus microcontroladores de forma más profesional, aunque también nos proporciona Energia, un entorno básico similar a Arduino IDE, cuyas funcionalidades son muy restringidas.

El entorno Energia se utilizó en la programación del código de pruebas del primer y segundo prototipo, para comprobar que las funciones del hardware eran las esperadas y como toma de contacto con el protocolo SPI.

CSS nos permite una programación a un nivel más bajo, con mayor versatilidad, aunque con órdenes más complejas, la comunicación por SPI por ejemplo se realiza con dos líneas en Energia con mientras que en CCS se necesitan 8. Sin embargo, proporciona un control absoluto sobre las acciones del microcontrolador, como la longitud de las comunicaciones, la capacidad de transmisión sin recepción, habilita el uso de interrupciones, timers, etc. Requisitos fundamentales para el código final.

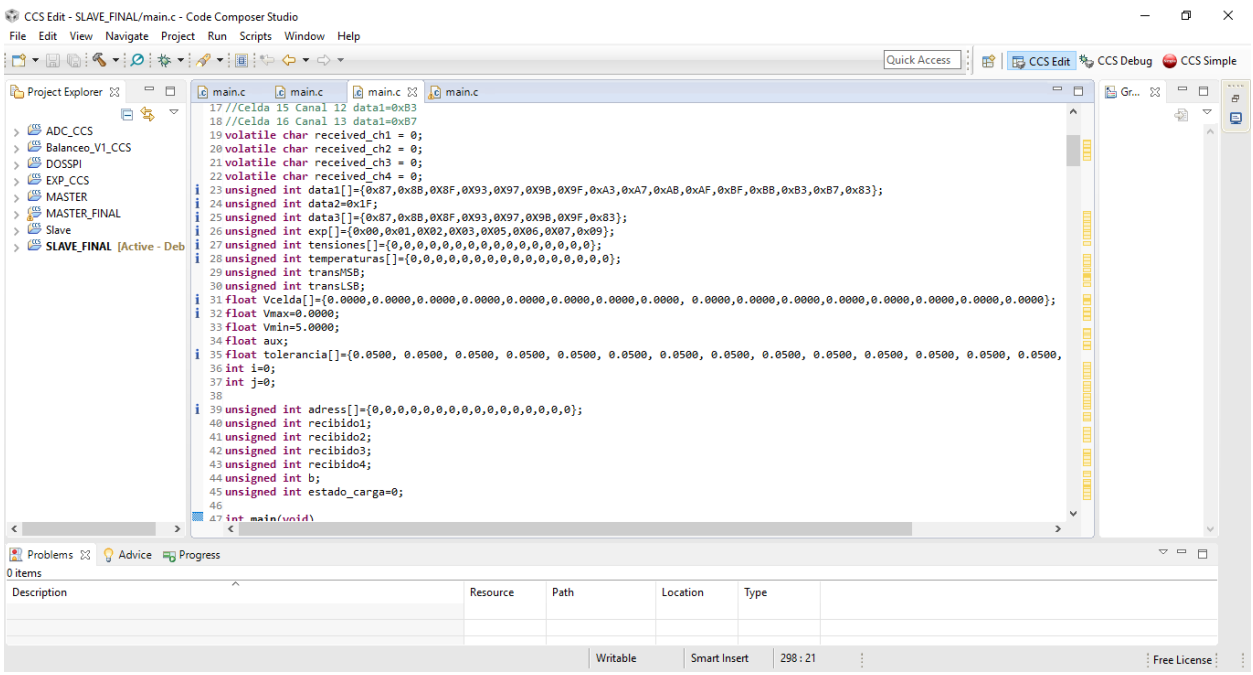

Figura 8-1. Entorno CCS Edit View

Para concluir, mencionar brevemente que el CCS dispone de dos modos o pantallas, CCS edit y CCS debug. La primera está orientada a la escritura y modificación del código, así como su compilación, acceso a otros proyectos, consola de errores, etc y la segunda a la ejecución del código, monitorización de variables, breakpoints, estado de ejecución, controles principales (play, pause, stop), etc.

8.2 Código del Slave

Para facilitar la comprensión del código, se expone el siguiente diagrama de flujo:

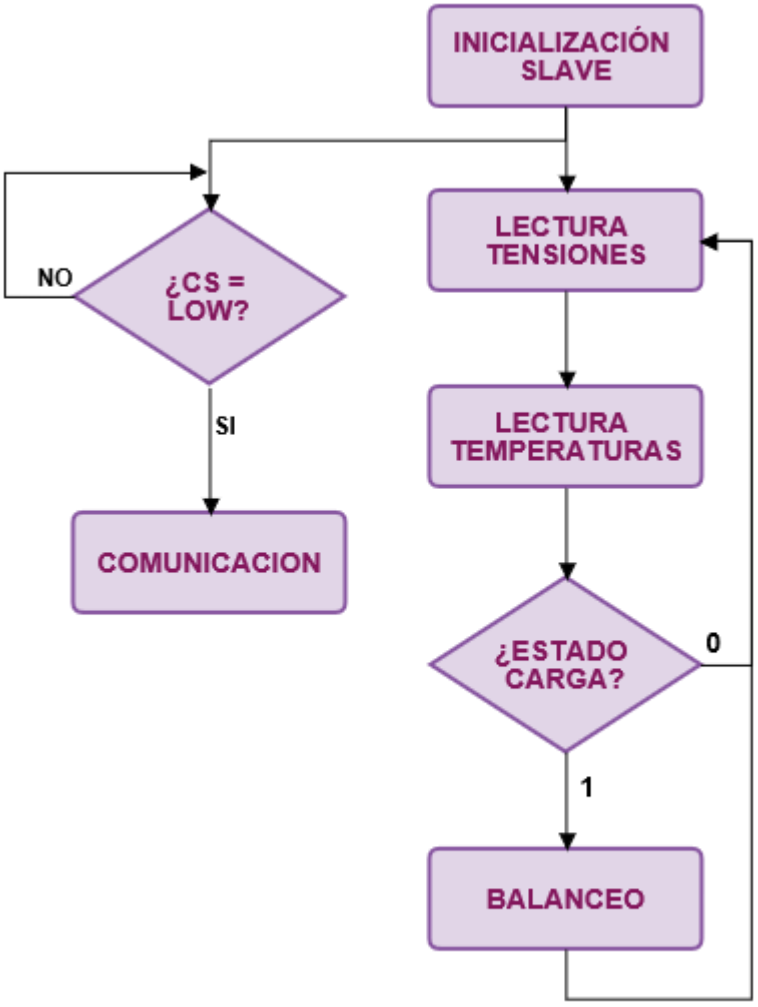

Figura 8-2. Diagrama de flujo del código del Slave

Aclarar que la comunicación en el *Slave* se maneja mediante interrupciones habilitadas por flanco de bajada en el CS, es decir, en el momento en el que el *Master* decida iniciarla, el *Slave* debe dejar la tarea que esté realizando para transferir los datos. Pese a que pueda resultar confuso, Estado_carga no hace referencia al SOC de la batería, sino al actual estado de la batería (cargándose o descargándose).

```
#include <msp430.h>
//Direcciones de cada canal del ADC de las tensiones
//Celda 1 Canal 0 data1=0x83
//Celda 2 Canal 1 data1=0x87
//Celda 3 Canal 2 data1=0X8B
//Celda 4 Canal 3 data1=0X8F
//Celda 5 Canal 4 data1=0X93
//Celda 6 Canal 5 data1=0X97
//Celda 7 Canal 6 data1=0X9B
//Celda 8 Canal 7 data1=0X9F
//Celda 9 Canal 8 data1=0xA3
//Celda 10 Canal 9 data1=0xA7
//Celda 11 Canal 10 data1=0XAB
//Celda 12 Canal 11 data1=0XAF
//Celda 13 Canal 15 data1=0XBF
//Celda 14 Canal 14 data1=0XBB
//Celda 15 Canal 12 data1=0xB3
//Celda 16 Canal 13 data1=0xB7
volatile char received_ch1 = 0; //Variables de recepción de datos del bucle principal
volatile char received_ch2 = 0;
volatile char received_ch3 = 0; //Variables de recepción de datos de la interrupción
volatile char received_ch4 = 0;
//Se eligen variables distintas para que la interrupción no altere el valor de
ninguna lectura que se haya podido realizar en ese momento
unsigned int data1[]={0x87,0x8B,0X8F,0X93,0X97,0X9B,0X9F,0xA3,0xA7,0xAB,0xAF,0xBF,
0xBB,0xB3,0xB7,0x83}; //Vector de direcciones codificadas del ADC de tensiones, como
se observba, se ha desfasado en uno para que la última comunicación prepare la
siguiente.
unsigned int data2=0x1F; //El segundo byte permanece siempre igual
unsigned int data3[]={0x87,0x8B,0X8F,0X93,0X97,0X9B,0X9F,0x83}; //Vector de
direcciones de los ADC de temperatura. Igualmente, desfasado.
unsigned int exp[]={0x00,0x01,0X02,0X03,0X05,0X06,0X07,0x09}; //Vector de direcciones
de los expansores de E/S
unsigned int tensiones[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //Vector de
almacenamiento de las tensiones sin descodificar.
unsigned int temperaturas[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //Vector de
almacenamiento de las temperaturas sin descodificar.
unsigned int transMSB; //Codificación de datos a mandar al Master, se toma como
variable para ir introduciendo cada valor de tensión y temperatura.
unsigned int transLSB;
float Vcelda[]={0.0000,0.0000,0.0000,0.0000,0.0000,0.0000,0.0000,0.0000,
0.0000,0.0000,0.0000,0.0000,0.0000,0.0000,0.0000,0.0000}; //Vector de almacenamiento
de las tensiones reales de cada celda.

float Vmin=5.0000; //Inicialización de una tensión mínima hasta que el Master pase el
primer valor
float aux; //Variable auxiliar que ayuda a convertir a las tensiones reales
float tolerancia[]={0.0500, 0.0500, 0.0500, 0.0500, 0.0500, 0.0500, 0.0500, 0.0500,
0.0500, 0.0500, 0.0500, 0.0500, 0.0500, 0.0500, 0.0500, 0.0500}; //Vector de
tolerancias de balanceo de cada celda, variará si la celda balancea o no.

int i=0;
int j=0;
```

```
unsigned int adress[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //Vector de direcciones de
las celdas, se utiliza durante el testeo para comprobar correcta lectura
unsigned int recibido1; //Variables de almacenamiento de datos recibidos, 1 y 2 para
el bucle principal, 3 y 4 para almacenar lo recibido por el Master
unsigned int recibido2;
unsigned int recibido3;
unsigned int recibido4;
unsigned int b; //Variable auxiliar para la decodificación de los datos leidos
unsigned int estado_carga=0; //Variable que refleja si la batería esta cargándose
(=1) o descargándose (=0), a cero inicialmente para evitar el balanceo.

int main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop WDT
//Configuracion de los CS de cada dispositivo

    P2OUT |= BIT5; //CSnADC1
    P2DIR |= BIT5;
    P2OUT |= BIT0; //CSnADC2
    P2DIR |= BIT0;
    P2OUT |= BIT1; //CSnADC3
    P2DIR |= BIT1;
    P2OUT |= BIT4; //CSnEXP1
    P2DIR |= BIT4;
    P1OUT |= BIT0; //CSnEXP2
    P1DIR |= BIT0;

    P1IE |= BIT3; // P1.3 Interrupciones activadas
    P1IES |= BIT3; // P1.3 Flanco de subida a bajada
    P1IFG &= ~BIT3; // P1.3 Flag de interrupciones limpio


 //Configuración del SPI de los periféricos

 P1SEL = BIT1 | BIT2 | BIT4| BIT5 | BIT6 |BIT7; //P1.1=MISO, P1.2= MOSI, P1.4=SCLK
//P1.5=SCLK, P1.6=MISO, P1.7=MOSI
 P1SEL2 = BIT1 | BIT2 | BIT4| BIT5 | BIT6 |BIT7; //P1SEL=11
 UCB0CTL1 = UCSWRST;
 UCB0CTL0 |= UCMSB + UCMST + UCSYNC + UCMODE_0 + UCCKPL +UCCKPH;
 UCB0CTL1 |= UCSSEL_2; // SMCLK master mode, ver en el regristro UCB0CTL0
 UCB0BR0 |= 0x01; // /1 mhz
 UCB0BR1 = 0; //
 UCB0CTL1 &= ~UCSWRST; // **Initialize USCI state machine**

//Configuracion del SPI como esclavo

 UCA0CTL1 = UCSWRST;
 UCA0CTL0 |= UCMSB + UCSYNC + UCMODE_0;
 UCA0CTL1 |= UCSSEL_0; // SMCLK slave mode, ver en el regristro UCA0CTL0
 UCA0BR0 |= 0x01; // /1 mhz
 UCA0BR1 = 0; //
 UCA0MCTL = 0; // No modulation
 UCA0CTL1 &= ~UCSWRST;

//Primera transmisión a cada ADC para que cargue el primer dato
      __bic_SR_register(GIE); // Desactivamos las interrupciones
            P2OUT &= (~BIT5); // Select ADC1
            while (!(IFG2 & UCB0TXIFG));
            UCB0TXBUF = 0x83;
            while (!(IFG2 & UCB0RXIFG));
```

```
            while (!(IFG2 & UCB0TXIFG));
            UCB0TXBUF = data2;
            while (!(IFG2 & UCB0RXIFG));
            P2OUT |= (BIT5); // Unselect Device


            P2OUT &= (~BIT0); // Select ADC2 (temperatura de las 8 primeras celdas)
            while (!(IFG2 & UCB0TXIFG));
             UCB0TXBUF = 0x83;
            while (!(IFG2 & UCB0RXIFG));
            while (!(IFG2 & UCB0TXIFG));
            UCB0TXBUF = data2;
            while (!(IFG2 & UCB0RXIFG));
            P2OUT |= (BIT0); // Unselect Device

            P2OUT &= (~BIT1); // Select ADC3
            while (!(IFG2 & UCB0TXIFG));
            UCB0TXBUF = 0x83;
            while (!(IFG2 & UCB0RXIFG));
            while (!(IFG2 & UCB0TXIFG));
            UCB0TXBUF = data2;
            while (!(IFG2 & UCB0RXIFG));
            P2OUT |= (BIT1); // Unselect Device
      __bis_SR_register(GIE);//Habilitamos interrupciones

    while(1){
//Lectura de todas las tensiones de celda
      for(i=0;i<16;i++)
      {
          P1SEL = BIT5 | BIT6 | BIT7; //Reconfiguramos el puerto, puesto que al
comunicarnos con el expansor la configuración cambia.
          P1SEL2 = BIT5 | BIT6 | BIT7;
          UCB0CTL1 = UCSWRST;
          UCB0CTL0 |= UCMSB + UCMST + UCSYNC + UCMODE_0 + UCCKPL +UCCKPH;
          UCB0CTL1 |= UCSSEL_2;
          UCB0BR0 |= 0x01; // /1 mhz
          UCB0BR1 = 0; //
          UCB0CTL1 &= ~UCSWRST;
      __bic_SR_register(GIE); //Antes de cada transmisión anulamos las
interrupciones
                  P2OUT &= (~BIT5); // Select Device
                      while (!(IFG2 & UCB0TXIFG));
                      UCB0TXBUF = data1[i]; //El vector data1 tiene ordenadas las
direcciones para que se lea de la primera a la decimo sexta celda
                      while (!(IFG2 & UCB0RXIFG));
                      received_ch1 = UCB0RXBUF;
                      while (!(IFG2 & UCB0TXIFG));
                      UCB0TXBUF = data2;
                      while (!(IFG2 & UCB0RXIFG));
                      received_ch2 = UCB0RXBUF;
                  P2OUT |= (BIT5); // Unselect Device
       __bis_SR_register(GIE);
// Transformamos el dato leído en un valor útil
            recibido1 = received_ch1;
            recibido2 = received_ch2;
            adress[i]=recibido1>>4; //Almacenamos su address
            b=recibido1 << 8;
            b=b+recibido2;
            b=b<<4; //Eliminamos los 4 bits de address
            b=b>>4;
```

```
            tensiones[i]=b; //Guardamos el valor de 12 bits en el vector tensiones
            aux=b*500.0000;
            aux=aux/4096.0000;
            aux=aux/100.0000;
            Vcelda[i]=aux-0.5000; //Eliminamos el offset que introduce la referencia
de los AD
      }

//Lectura de las temperaturas
//Segundo ADC, primeras 8 temperaturas
      for(i=0;i<8;i++)
            {
            P1SEL = BIT5 | BIT6 | BIT7;
            P1SEL2 = BIT5 | BIT6 | BIT7;
            UCB0CTL1 = UCSWRST;
            UCB0CTL0 |= UCMSB + UCMST + UCSYNC + UCMODE_0 + UCCKPL +UCCKPH;
            UCB0CTL1 |= UCSSEL_2;
            UCB0BR0 |= 0x01; // /1 mhz
            UCB0BR1 = 0; //
            UCB0CTL1 &= ~UCSWRST;
      __bic_SR_register(GIE);
                  P2OUT &= (~BIT0); // Select Device
                  while (!(IFG2 & UCB0TXIFG));
                  UCB0TXBUF = data3[i];
                  while (!(IFG2 & UCB0RXIFG));
                  received_ch1 = UCB0RXBUF;
                  while (!(IFG2 & UCB0TXIFG));
                  UCB0TXBUF = data2;
                  while (!(IFG2 & UCB0RXIFG));
                  received_ch2 = UCB0RXBUF;
                  P2OUT |= (BIT0); // Unselect Device
       __bis_SR_register(GIE);
                  recibido1 = received_ch1;
                  recibido2 = received_ch2;
                  adress[i]=recibido1>>4;
                  b=recibido1 << 8;
                  b=b+recibido2;
                  b=b<<4;
                  b=b>>4;
                  temperaturas[i]=b;
            }

//Tercer ADC, segundas 8 temperaturas
       for(i=0;i<8;i++)
            {
                  P1SEL = BIT5 | BIT6 | BIT7;
                  P1SEL2 = BIT5 | BIT6 | BIT7;
                  UCB0CTL1 = UCSWRST;
                  UCB0CTL0 |= UCMSB + UCMST + UCSYNC + UCMODE_0 + UCCKPL +UCCKPH;
                  UCB0CTL1 |= UCSSEL_2;
                  UCB0BR0 |= 0x01; // /1 mhz
                  UCB0BR1 = 0; //
                  UCB0CTL1 &= ~UCSWRST;
            __bic_SR_register(GIE);
                  P2OUT &= (~BIT1); // Select Device
                  while (!(IFG2 & UCB0TXIFG));
                  UCB0TXBUF = data3[i];
                  while (!(IFG2 & UCB0RXIFG));
                  received_ch1 = UCB0RXBUF;
                  while (!(IFG2 & UCB0TXIFG));
```

```
                    UCB0TXBUF = data2;
                    while (!(IFG2 & UCB0RXIFG));
                    received_ch2 = UCB0RXBUF;
                    P2OUT |= (BIT1); // Unselect Device
                __bis_SR_register(GIE);
                   recibido1 = received_ch1;
                recibido2 = received_ch2;
                adress[i]=recibido1>>4;
                b=recibido1 << 8;
                b=b+recibido2;
                b=b<<4;
                b=b>>4;
                temperaturas[8+i]=b; //Se almacenan las temperaturas de la 8 a 16
                            }

//Config SPI del EXPANSOR

                P1SEL = BIT5 | BIT6 | BIT7;
                P1SEL2 = BIT5 | BIT6 | BIT7;
                UCB0CTL1 = UCSWRST;
                UCB0CTL0 |= UCMSB + UCMST + UCSYNC + UCMODE_0 +UCCKPH; //CPHA=0 y CPOL=0
                UCB0CTL1 |= UCSSEL_2;
                UCB0BR0 |= 0x01; // /1 mhz
                UCB0BR1 = 0; //
                UCB0CTL1 &= ~UCSWRST;
if (estado_carga == 1) //Batería cargándose
{
//Primer EXP
for(i=0;i<8;i++) //En este bucle se recorre celda a celda comparando su tensión con
la mínima más una pequeña tolerancia, la tolerancia estima la caída de tensión de la
celda durante el balanceo, si la celda se va a balancear se elimina su tolerancia,
para que al volver a pasar la condicion no deje de balancear, si no se va a balancear
su tolerancia vuelve al valor inicial.
{

   if (Vcelda[i] > Vmin + tolerancia[i])
    {
      __bic_SR_register(GIE);
        tolerancia[i]=0.0000;
        P2OUT &= (~BIT4); // Select Device EXP1
        while (!(IFG2 & UCB0TXIFG));
        UCB0TXBUF = exp[i];
        UCB0TXBUF = 0x00;
        while (!(IFG2 & UCB0TXIFG));
        P2OUT |= (BIT4); // Unselect Device EXP1
       __bis_SR_register(GIE);
    }
   else
    {
     __bic_SR_register(GIE);
      tolerancia[i]=0.0500;
      P2OUT &= (~BIT4); // Select Device EXP1
       while (!(IFG2 & UCB0TXIFG));
       UCB0TXBUF = exp[i];
       UCB0TXBUF = 0x01;
       while (!(IFG2 & UCB0TXIFG));
       P2OUT |= (BIT4); // Unselect Device EXP1
     __bis_SR_register(GIE);
    }
}
```

```
 //Segundo EXP, lo mismo que el primero, pero para las 8 siguientes celdas
for(i=0;i<8;i++)
{

  if (Vcelda[i+8] > Vmin + tolerancia[i+8])
   {
     __bic_SR_register(GIE);
      tolerancia[8+i]=0.0000;
      P1OUT &= (~BIT0); // Select Device EXP2
      while (!(IFG2 & UCB0TXIFG));
      UCB0TXBUF = exp[i];
      UCB0TXBUF = 0x00;
      while (!(IFG2 & UCB0TXIFG));
      P1OUT |= (BIT0); // Unselect Device EXP2
     __bis_SR_register(GIE);
  }
 else
  {
     __bic_SR_register(GIE);
      tolerancia[8+i]=0.0500;
      P1OUT &= (~BIT0); // Select Device EXP2
      while (!(IFG2 & UCB0TXIFG)); // USCI_A0 TX buffer ready?
      UCB0TXBUF = exp[i];
      UCB0TXBUF = 0x01;
      while (!(IFG2 & UCB0TXIFG)); //Espera a acabar la transmision
      P1OUT |= (BIT0); // Unselect Device EXP2
    __bis_SR_register(GIE);
   }
}
}
else //Si la batería no está en carga, apaga todos los circuitos de balanceo
 {
      __bic_SR_register(GIE);
      P2OUT &= (~BIT4); // Select Device EXP1
      while (!(IFG2 & UCB0TXIFG));
      UCB0TXBUF = 0x0A; //Selecciona todos los puertos del expansor
      UCB0TXBUF = 0x01;
      while (!(IFG2 & UCB0TXIFG));
      P2OUT |= (BIT4); // Unselect Device EXP1

      P1OUT &= (~BIT0); // Select Device EXP2
      while (!(IFG2 & UCB0TXIFG));
      UCB0TXBUF = 0x0A;
      UCB0TXBUF = 0x01;
      while (!(IFG2 & UCB0TXIFG));
      P1OUT |= (BIT0); // Unselect Device EXP2
      __bis_SR_register(GIE);
      tolerancia[]={0.0500, 0.0500, 0.0500, 0.0500, 0.0500, 0.0500, 0.0500, 0.0500,
0.0500, 0.0500, 0.0500, 0.0500, 0.0500, 0.0500, 0.0500, 0.0500}; //Se reestablecen
las tolerancias
  }
 }
}

#pragma vector=PORT1_VECTOR //Vector de interrupción dedicado a la comunicación con
el Master
__interrupt void Port_1(void)
{
      for(j=0;j<33;j++)
      {
```

```
       if(j < 16) //Los 16 primeros datos que envía son tensiones, para ello
modifica transMSB y trans LSB que son las variables que envía
      {
      transLSB=tensiones[i]<<8;
      transLSB=transLSB>>8;
      transMSB=tensiones[i]>>8;
      }
      else if(j>=16 && j<32) //Los 16 siguientes son temperaturas
      {
      transLSB=temperaturas[i]<<8;
      transLSB=transLSB>>8;
      transMSB=temperaturas[i]>>8;
      }
      else if(j==32) //El último dato es basura, para sacar el byte desfasado
      {
      transMSB=0xFF;
      transLSB=0xFF;
      }
    __delay_cycles(12.5);
      while (!(IFG2 & UCA0TXIFG));
      UCA0TXBUF = transMSB;
      while (!(IFG2 & UCA0RXIFG));
      received_ch3 = UCA0RXBUF;
      recibido3=received_ch3;
    __delay_cycles(25); //Delay intermedio entre primer byte y segundo

      while (!(IFG2 & UCA0TXIFG));
      UCA0TXBUF = transLSB;
      while (!(IFG2 & UCA0RXIFG));
      received_ch4 = UCA0RXBUF;
      recibido4=received_ch4;

      Vmin=recibido3<<8; //Guarda el dato recibido
      Vmin=Vmin + recibido4;
      }
      while(!(P1IN & BIT3)); // Espera a acabar la transmision

         if (Vmin > 4095) //Dado que el dato de tensión mínima es de 12 bits (hasta
4095) la forma más sencilla de comprobar que la batería esta cargando es preguntando
si sobrepasa ese valor (recordar el bit de estado_carga es el bit13 contando por la
derecha, ver apartado de comunicación entre µCs)
         {
             Vmin = Vmin - 4096;
             estado_carga=1;
         }
         else if (Vmin <= 4095)
         {
              estado_carga=0;
         }

P1IFG &=~BIT3; // P1.3 Limpia el flag de interrupciones
}
```

8.3 Código del Master

Para facilitar la comprensión del código del *Master*, se ha elaborado el siguiente diagrama de flujo:

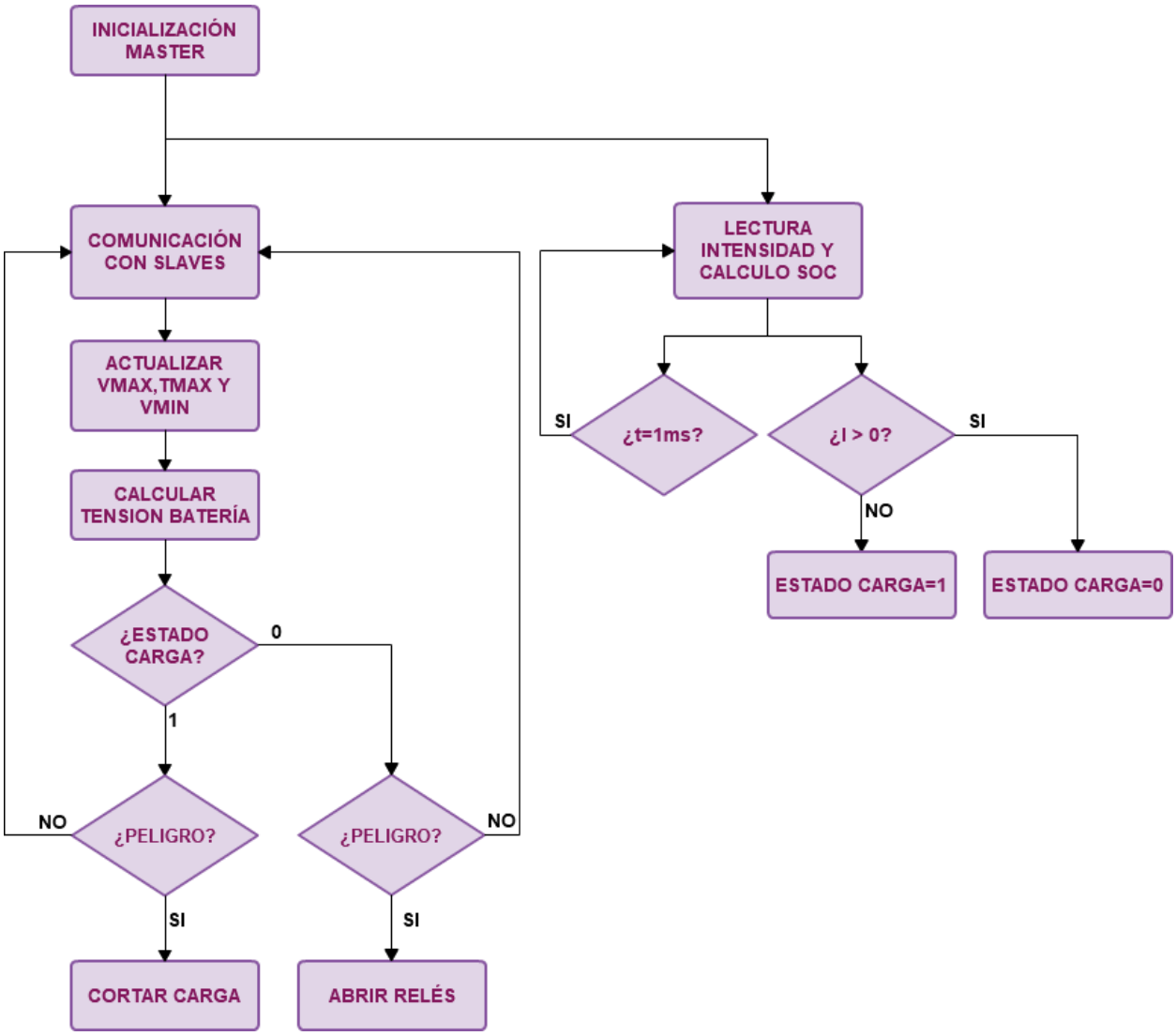

Figura 8-3. Diagrama de flujo del código del Master

Aclarar que la lectura de intensidad, actualización de la variable estado_carga y el cálculo del SOC se produce mediante una interrupción que se ejecuta cada 1ms. En Comunicación con Slaves, la comunicación la hace una a una y en orden de CS.

```
#include <msp430.h>
void guarda_flash(char dato);

volatile char received_ch1 = 0;
volatile char received_ch2 = 0;
volatile char received_ch3 = 0;
volatile char received_ch4 = 0;
```

```
unsigned int capac_bat=19440; //En culombios = XAh * 3600;
unsigned int reciint1; //Variables para recibir los datos de lectura de intensidad
unsigned int reciint2;
unsigned int suma_inten;

unsigned int envia_dato; //Dato que se enviará a los esclavos
unsigned int i=0;
unsigned int recibido[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0}; //Vector de valores leidos recibidos, por limitaciones de
memoria no se pueden almacenar todas las tensiones ni temperatura, por lo que se irá
sobreescribiendo en cada Slave
unsigned int aux1;
unsigned int aux2;

unsigned int abrerele=0; //Abre los AIR ante situación insegura
unsigned int estado_carga=0; //Situación de carga o descarga de la batería
unsigned int cortacarga=0; //Orden al cargador de cortar la carga

unsigned int Vmin=4095; //Inicialmente y hasta que no se hayan leido todas las
tensiones no se balancea nada, se da una Vmin alta para ello
unsigned int Vmincompare=4095; //Valor para ir comparando la tensión mínima
unsigned int Vmax=0; //valor inicial que cambiará despues de leerlas todas, no
influye, representará la mayor tensión de celda
unsigned int Vmaxcompare=0;
unsigned int Tempcompare=4095; //Su valor inicial es alto porque el LMT87 da
tensiones altas para temperaturas bajas, con lo cual se buscará el mínimo para la
temperatura más alta.
unsigned int TempMAX=0; //Inicialmente no se tiene en cuenta la temperatura
unsigned int transMSB=0x0F; //Transmite inicialmente una Vmin saturada, 5V, para
evitar el balanceo antes de leer todas las tensiones
unsigned int transLSB=0xFF;
char *Puntero = (char *) 0xC400; // Puntero que apunta al segmento donde se guardan
los datos en la memoria flash.

char SOC_save; // Variable para guardar el SOC (de 1 a 100) en la memoria flash.
double current; // Corriente entrante o saliente de la batería
double aux=0.0000000;

float SOC;
float aux3;
float tension_bat=0.00000;
float suma_tension=0.00000;
float carga;


int main(void)
{
WDTCTL = WDTPW + WDTHOLD; // Stop WDT

//Configuración Chip selects
    P3OUT |= BIT1; //CS1
    P3DIR |= BIT1;
    P3OUT |= BIT0; //CS2
    P3DIR |= BIT0;
    P2OUT |= BIT0; //CS3
    P2DIR |= BIT0;
    P2OUT |= BIT1; //CS4
    P2DIR |= BIT1;
    P2OUT |= BIT2; //CS5
```

```
    P2DIR |= BIT2;
    P3OUT |= BIT3; //CS6
    P3DIR |= BIT3;
    P3OUT |= BIT2; //CS7
    P3DIR |= BIT2;
    P1OUT |= BIT0; //CS8
    P1DIR |= BIT0;
    P2OUT |= BIT5; //CSADC
    P2DIR |= BIT5;
    P3OUT |= BIT7; //CSCAN
    P3DIR |= BIT7;

//Interrupción del timer de interrupción

    CCTL0 = CCIE;                             // CCR0 interrupt enabled
    TACTL = TASSEL_2 + MC_1 + ID_3;           // SMCLK/8, upmode
    CCR0 =  125;                             // SMCLK/8*125= 1ms

//Configuracion del SPI USCI_A
       P1SEL = BIT1 | BIT2 | BIT4| BIT5 | BIT6 |BIT7;
//P1.1=MISO,P1.2=MOSI,P1.4=SCLK. P1.5= SCLK, P1.6=MISO, P1.7=MOSI
       P1SEL2 = BIT1 | BIT2 | BIT4| BIT5 | BIT6 |BIT7; //P1SEL=11
       UCA0CTL1 = UCSWRST;
       UCA0CTL0 |= UCMSB + UCMST + UCSYNC + UCMODE_0; //CPOL=0 y CPHA=1
       UCA0CTL1 |= UCSSEL_2; // SMCLK master mode, ver enl regristro UCA0CTL0
       UCA0BR0 |= 0x01; // /1 mhz
       UCA0BR1 = 0; //
       UCA0MCTL = 0; // No modulation
       UCA0CTL1 &= ~UCSWRST;

//Configuracion del SPI USCI_B (monitorización de corriente)
       UCB0CTL1 = UCSWRST;
       UCB0CTL0 |= UCMSB + UCMST + UCSYNC + UCMODE_0; //CPOL=0 y CPHA=1
       UCB0CTL1 |= UCSSEL_2; // SMCLK master mode, ver en el regristro UCB0CTL0
       UCB0BR0 |= 0x01; // /1 mhz
       UCB0BR1 = 0; //
       UCB0CTL1 &= ~UCSWRST;

       __bis_SR_register(GIE);  //Habilitamos interrupciones

 SOC=*Puntero; //Recoge el valor guardado de SOC en la memoria
 carga=(SOC * capac_bat)/100; //Valor de carga inicial


while(1){

       P3OUT&= (~BIT1); // ------------------PRIMERA TRANSMISION-----------------
       __bic_SR_register(GIE);
       for(i=0;i<33;i++)
             {
                    while (!(IFG2 & UCA0TXIFG));
                    UCA0TXBUF =transMSB;
                    while (!(IFG2 & UCA0RXIFG));
                    received_ch1 = UCA0RXBUF;
                    while (!(IFG2 & UCA0TXIFG));
                    UCA0TXBUF =transLSB;
                    while (!(IFG2 & UCA0RXIFG));
                    received_ch2 = UCA0RXBUF;
                    aux1=received_ch1;
                    aux2=received_ch2;
```

```
                    if (i==0) //Desplaza el vector recibido hacia adelante un byte
(ver apartado de comunicación entre µCs)
                    {
                    recibido[i]=aux2;
                    }


                   else
                    {
                    recibido[i-1]= recibido[i-1]<<8 + aux1;
                    recibido[i]=aux2;
                    }
              }
    P3OUT |= (BIT1);
    __bis_SR_register(GIE);


              for (i=0;i<16;i++) //Recorre las tensiones leidas y las compara con los
valores máximos y mínimos.
              {
                    if(recibido[i] < Vmincompare)
                    {
                          Vmincompare=recibido[i];
                    }
                    if(recibido[i]> Vmaxcompare)
                    {
                          Vmaxcompare=recibido[i];
                    }
                    aux3=recibido[i]*500.0000;
                    aux3=aux3/4096.0000;
                    aux3=aux3/100.0000;
                    aux3=aux3-0.500000;
       suma_tension=suma_tension + aux3; //Transformamos el valor de 12 bits a float
y calculamos la tension de la batería completa mediante una suma.
              }

              for (i=16;i<32;i++) //Recorre las tensiones de temperaturas leidas y las
compara
              {
                    if(recibido[i] < Tempcompare)
                        {
                        Tempcompare=recibido[i];
                        }
              }
           __delay_cycles(500); // Pequeño retraso entre transmisiones de 0.5ms

       P3OUT&= (~BIT0); // ------------------SEGUNDA TRANSMISION-----------------
Exactamente igual a la primera cambiando el CS
       __bic_SR_register(GIE);
              for(i=0;i<33;i++)
                    {

                    while (!(IFG2 & UCA0TXIFG));
                    UCA0TXBUF = transMSB;
                    while (!(IFG2 & UCA0RXIFG));
                     received_ch1 = UCA0RXBUF;
                    while (!(IFG2 & UCA0TXIFG));
                    UCA0TXBUF = transLSB;
                    while (!(IFG2 & UCA0RXIFG));
                    received_ch2 = UCA0RXBUF;
```

```
                  aux1=received_ch1;
                  aux2=received_ch2;

                  if (i==0) //Desplaza el vector recibido hacia adelante un byte
                        {
                        recibido[i]=aux2;
                  }
            else
             {
             recibido[i-1]= recibido[i-1]<<8 + aux1;
            recibido[i]=aux2;
             }
            }
    P3OUT |= (BIT0);
    __bis_SR_register(GIE);


            for (i=0;i<16;i++) //Recorre las tensiones leidas y las compara
                  {
                        if(recibido[i] < Vmincompare)
                        {
                              Vmincompare=recibido[i];
                        }
                        if(recibido[i]> Vmaxcompare)
                        {
                              Vmaxcompare=recibido[i];
                        }
                     aux3=recibido[i]*500.0000;
                     aux3=aux3/4096.0000;
                     aux3=aux3/100.0000;
                     aux3=aux3-0.5000;

             suma_tension=suma_tension + aux3; //Calculamos la tension de la batería
completa
                  }

            for (i=16;i<32;i++) //Recorre las tensiones de temperaturas leidas y las
compara
                  {
                        if(recibido[i] < Tempcompare)
                            {
                            Tempcompare=recibido[i];
                            }
                  }
        __delay_cycles(500);

        P2OUT&= (~BIT0); // ------------------TERCERA TRANSMISION-----------------
        __bic_SR_register(GIE);
             for(i=0;i<33;i++)
                  {

                        while (!(IFG2 & UCA0TXIFG));
                        UCA0TXBUF = transMSB;
                        while (!(IFG2 & UCA0RXIFG));
                        received_ch1 = UCA0RXBUF;
                        while (!(IFG2 & UCA0TXIFG));
                        UCA0TXBUF = transLSB;
                        while (!(IFG2 & UCA0RXIFG));
                        received_ch2 = UCA0RXBUF;
                        aux1=received_ch1;
```

```
                        aux2=received_ch2;
                  if (i==0) //Desplaza el vector recibido hacia adelante un byte
                              {
                              recibido[i]=aux2;
                              }

                     else
                              {
                              recibido[i-1]= recibido[i-1]<<8 + aux1;
                              recibido[i]=aux2;
                              }
                  }
            P2OUT |= (BIT0);
            __bis_SR_register(GIE);


             for (i=0;i<16;i++) //Recorre las tensiones leidas y las compara
                  {
                        if(recibido[i] < Vmincompare)
                        {
                              Vmincompare=recibido[i];
                        }
                        if(recibido[i]> Vmaxcompare)
                        {
                              Vmaxcompare=recibido[i];
                        }
                        aux3=recibido[i]*500.0000;
                            aux3=aux3/4096.0000;
                            aux3=aux3-0.5000;

                  suma_tension=suma_tension + aux3; //Calculamos la tension de la
batería completa
                  }

             for (i=16;i<32;i++) //Recorre las tensiones de temperaturas leidas y las
compara
                  {
                        if(recibido[i] < Tempcompare)
                            {
                            Tempcompare=recibido[i];
                            }
                  }
                __delay_cycles(500);

           P2OUT&= (~BIT1); // ------------------CUARTA TRANSMISION-----------------
             __bic_SR_register(GIE);
               for(i=0;i<33;i++)
                  {

                        while (!(IFG2 & UCA0TXIFG));
                        UCA0TXBUF = transMSB;
                        while (!(IFG2 & UCA0RXIFG));
                        received_ch1 = UCA0RXBUF;
                         while (!(IFG2 & UCA0TXIFG));
                        UCA0TXBUF = transLSB;
                        while (!(IFG2 & UCA0RXIFG));
                        received_ch2 = UCA0RXBUF;
                        aux1=received_ch1;
                        aux2=received_ch2;
```

```
                    if (i==0) //Desplaza el vector recibido hacia adelante un byte
                                {
                                recibido[i]=aux2;
                                }
                    else
                                {
                                recibido[i-1]= recibido[i-1]<<8 + aux1;
                                recibido[i]=aux2;
                                }
                            }
      P2OUT |= (BIT1);
      __bis_SR_register(GIE);

                for (i=0;i<16;i++) //Recorre las tensiones leidas y las compara
                            {
                                  if(recibido[i] < Vmincompare)
                                  {
                                        Vmincompare=recibido[i];
                                  }
                                  if(recibido[i]> Vmaxcompare)
                                  {
                                        Vmaxcompare=recibido[i];
                                  }
                                aux3=recibido[i]*500.0000;
                                aux3=aux3/4096.0000;
                                aux3=aux3/100.0000;
                                aux3=aux3-0.5000;

                            suma_tension=suma_tension + aux3; //Calculamos la tension
de la batería completa
                            }

                for (i=16;i<32;i++) //Recorre las tensiones de temperaturas leidas y
las compara
                            {
                                  if(recibido[i] < Tempcompare)
                                      {
                                      Tempcompare=recibido[i];
                                      }
                            }
                     __delay_cycles(500);

            P2OUT&= (~BIT2); // ------------------QUINTA TRANSMISION-----------------
            __bic_SR_register(GIE);
                for(i=0;i<33;i++)
                    {

                          while (!(IFG2 & UCA0TXIFG));
                          UCA0TXBUF = transMSB;
                          while (!(IFG2 & UCA0RXIFG));
                          received_ch1 = UCA0RXBUF;
                          while (!(IFG2 & UCA0TXIFG));
                          UCA0TXBUF = transLSB;
                          while (!(IFG2 & UCA0RXIFG));
                          received_ch2 = UCA0RXBUF;
                          aux1=received_ch1;
                          aux2=received_ch2;

                    if (i==0) //Desplaza el vector recibido hacia adelante un byte
                                      {
```

```
                                recibido[i]=aux2;
                                }
                        else
                                {
                                recibido[i-1]= recibido[i-1]<<8 + aux1;
                                recibido[i]=aux2;
                                }
                         }
                P2OUT |= (BIT2);
                 __bis_SR_register(GIE);


                      for (i=0;i<16;i++) //Recorre las tensiones leidas y las compara
                            {
                                if(recibido[i] < Vmincompare)
                                {
                                      Vmincompare=recibido[i];
                                }
                                if(recibido[i]> Vmaxcompare)
                                {
                                      Vmaxcompare=recibido[i];
                                }
                               aux3=recibido[i]*500.0000;
                               aux3=aux3/4096.0000;
                               aux3=aux3/100.0000;
                               aux3=aux3-0.5000;

                                suma_tension=suma_tension + aux3; //Calculamos la tension
de la batería completa
                            }

                       for (i=16;i<32;i++) //Recorre las tensiones de temperaturas
leidas y las compara
                                  {
                                      if(recibido[i] < Tempcompare)
                                          {
                                          Tempcompare=recibido[i];
                                          }
                                  }

                    __delay_cycles(500);

                   P3OUT&= (~BIT3); // ------------------SEXTA TRANSMISION-----------------
                          __bic_SR_register(GIE);
                          for(i=0;i<33;i++)
                              {

                                while (!(IFG2 & UCA0TXIFG));
                                UCA0TXBUF = transMSB;
                                while (!(IFG2 & UCA0RXIFG));
                                received_ch1 = UCA0RXBUF;
                                while (!(IFG2 & UCA0TXIFG));
                                UCA0TXBUF = transLSB;
                                while (!(IFG2 & UCA0RXIFG));
                                received_ch2 = UCA0RXBUF;
                                aux1=received_ch1;
                                aux2=received_ch2;

                         if (i==0) //Desplaza el vector recibido hacia adelante un byte
                                                {
```

```
                                        recibido[i]=aux2;
                                        }
                    else
                                        {
                                        recibido[i-1]= recibido[i-1]<<8 + aux1;
                                        recibido[i]=aux2;
                                        }
                      }
            P3OUT |= (BIT3);
            __bis_SR_register(GIE);


                     for (i=0;i<16;i++) //Recorre las tensiones leidas y las compara
                              {
                                    if(recibido[i] < Vmincompare)
                                    {
                                          Vmincompare=recibido[i];
                                    }
                                    if(recibido[i]> Vmaxcompare)
                                    {
                                          Vmaxcompare=recibido[i];
                                    }
                                    aux3=recibido[i]*500.0000;
                                        aux3=aux3/4096.0000;
                                        aux3=aux3/100.0000;
                                        aux3=aux3-0.5000;

                                suma_tension=suma_tension + aux3; //Calculamos la
tension de la batería completa
                                }

                      for (i=16;i<32;i++) //Recorre las tensiones de temperaturas
leidas y las compara
                                {
                                    if(recibido[i] < Tempcompare)
                                        {
                                        Tempcompare=recibido[i];
                                        }
                                }
            __delay_cycles(500);

           P3OUT&= (~BIT2); // ------------------SÉPTIMA TRANSMISION-----------------
            __bic_SR_register(GIE);
                      for(i=0;i<33;i++)
                                {

                                while (!(IFG2 & UCA0TXIFG));
                                UCA0TXBUF = transMSB;
                                while (!(IFG2 & UCA0RXIFG));
                                received_ch1 = UCA0RXBUF;
                                while (!(IFG2 & UCA0TXIFG));
                                UCA0TXBUF = transLSB;
                                while (!(IFG2 & UCA0RXIFG));
                                received_ch2 = UCA0RXBUF;
                                aux1=received_ch1;
                                aux2=received_ch2;

                      if (i==0) //Desplaza el vector recibido hacia adelante un byte
                                        {
                                        recibido[i]=aux2;
```

```
                                        }
                        else
                                        {
                                        recibido[i-1]= recibido[i-1]<<8 + aux1;
                                        recibido[i]=aux2;
                                        }
                                    }
              P3OUT |= (BIT2);
              __bis_SR_register(GIE);


                        for (i=0;i<16;i++) //Recorre las tensiones leidas y las compara
                                    {
                                        if(recibido[i] < Vmincompare)
                                        {
                                              Vmincompare=recibido[i];
                                        }
                                        if(recibido[i]> Vmaxcompare)
                                        {
                                              Vmaxcompare=recibido[i];
                                        }
                                    aux3=recibido[i]*500.0000;
                                    aux3=aux3/4096.0000;
                                    aux3=aux3/100.0000;
                                    aux3=aux3-0.5000;

                          suma_tension=suma_tension + aux3; //Calculamos la tension de
la batería completa
                                    }

                        for (i=16;i<32;i++) //Recorre las tensiones de temperaturas
leidas y las compara
                                    {
                                        if(recibido[i] < Tempcompare)
                                            {
                                            Tempcompare=recibido[i];
                                            }
                                    }
                         __delay_cycles(500);

            P1OUT&= (~BIT0); // ------------------OCTAVA TRANSMISION-----------------
            __bic_SR_register(GIE);
                        for(i=0;i<33;i++)
                                  {

                                  while (!(IFG2 & UCA0TXIFG));
                                  UCA0TXBUF = transMSB;
                                  while (!(IFG2 & UCA0RXIFG));
                                  received_ch1 = UCA0RXBUF;
                                  while (!(IFG2 & UCA0TXIFG));
                                  UCA0TXBUF = transLSB;
                                  while (!(IFG2 & UCA0RXIFG));
                                  received_ch2 = UCA0RXBUF;
                                  aux1=received_ch1;
                                  aux2=received_ch2;

                         if (i==0) //Desplaza el vector recibido hacia adelante un byte
                                        {
                                        recibido[i]=aux2;
                                        }
```

```
                        else
                                                {
                                                recibido[i-1]= recibido[i-1]<<8 + aux1;
                                                 recibido[i]=aux2;
                                                }
                        }
            P1OUT |= (BIT0);
            __bis_SR_register(GIE);

                    for (i=0;i<16;i++) //Recorre las tensiones leidas y las compara
                                {
                                            if(recibido[i] < Vmincompare)
                                            {
                                                  Vmincompare=recibido[i];
                                            }
                                            if(recibido[i]> Vmaxcompare)
                                            {
                                                  Vmaxcompare=recibido[i];
                                            }
                                            aux3=recibido[i]*500.0000;
                                            aux3=aux3/4096.0000;
                                            aux3=aux3/100.0000;
                                            aux3=aux3-0.5000;

                                    suma_tension=suma_tension + aux3; //Calculamos la
tension de la batería completa
                                    }

                    for (i=16;i<32;i++) //Recorre las tensiones de temperaturas
leidas y las compara
                                    {
                                          if(recibido[i] < Tempcompare)
                                          {
                                          Tempcompare=recibido[i];
                                          }
                                    }
                     __delay_cycles(500);

          Vmin=Vmincompare; //Se prepara la Vmin para la siguiente conversión
          Vmincompare=4095;
          Vmax=Vmaxcompare; //Se recopila la Vmax
          Vmaxcompare=0;
          TempMAX=Tempcompare; //Se recopila la temperatura máxima
          Tempcompare=4095;
          tension_bat=suma_tension; //Tensión total de la batería
          suma_tension=0.00000000;

          envia_dato=Vmin; //Codificación para enviar


if (estado_carga==1) //Cargando la batería
      {
      if (Vmax > 3850 || TempMAX < 1513 || ((current < 0.20000) && (SOC > 20)))
            {
            cortacarga=1; //Tensión mayor de 4.69V en las celdas (menos la
referencia 4.19V), temperatura mayor a 57ºC o carga residual (corriente baja cuando
las celdas estan casi cargadas)
            }
      envia_dato=envia_dato+4096; // Si el estado_carga=1 se pone a uno el 13º bit
del dato que se va a enviar (sumar 4096)
```

```
      }
else if(estado_carga==0) //Descargando la batería
      {
      if (Vmin < 2490 || TempMAX < 1513 || current >240)
            {
            abrerele=1;//Tensión por debajo de 3.04V, sin la referencia del AD =>
2.54V, temperatura mayor a 57ºC y protección contra sobrecorriente
                  }
            }
   transLSB=envia_dato<<8; // Sumado o no el bit de estado_carga, se prepara para la
siguiente transmisión del dato a enviar.
   transLSB=transLSB>>8;
   transMSB=envia_dato>>8;
  }
}
// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR //Programamos una interrupción cada milisegundo para
leer la corriente y determinar carga o descarga y el SOC
__interrupt void Timer_A (void)
{
       // Lectura de corriente
      P2OUT &= (~BIT5);

            while (!(IFG2 & UCB0TXIFG));
            UCB0TXBUF = 0xFF; //El ADC no tiene MOSI
            while (!(IFG2 & UCB0RXIFG));
            received_ch3 = UCB0RXBUF;
            while (!(IFG2 & UCB0TXIFG));
            UCB0TXBUF = 0xFF; //El ADC no tiene MOSI
            while (!(IFG2 & UCB0RXIFG));
            received_ch4 = UCB0RXBUF;

      P2OUT |= (BIT5);
       reciint1=received_ch3;
       reciint2=received_ch4;
       suma_inten = reciint1 << 8;
       suma_inten= suma_inten + reciint2;
       aux=suma_inten*5.0000;
     aux=aux/655.360000;
     aux=aux/100.0000;
     current= (aux-2.500000)*160;

     //Condición de carga o descarga
      if (current >= 0)
      {
            estado_carga=0;
      }
      else if (current <0)
      {
            estado_carga=1;
      }
      //SOC

      carga = carga - current/1000; //Integramos la corriente, contando culombios.
La corriente es negativa si entra en la batería. Dado que un Amperio es 1C/s o
0.001C/ms, como medimos cada ms, dividimos entre mil (o multiplicamos por 0.001) la
corriente medida en ese momento, y se le resta a la carga (en culombios) dado que si
la corriente es positiva (intensidad sale de la batería) se disminuye la carga.
      SOC = carga*100/ capac_bat; //Teniendo la carga de la batería en culombios y
su capacidad calculamos el SOC
```

```
      SOC_save=SOC;
      guarda_flash(SOC_save); //Y guardamos el SOC en la memoria flash para la
siguiente conversión.

}

void guarda_flash(char dato) //La función para guardar en la memoria flash está
recogida de los apuntes de SED de GITI
{
char *Puntero = (char *) 0xC400; // Apunta al segmento donde guardan los datos
      __bic_SR_register(GIE);
FCTL1 = FWKEY + ERASE;    // activa  Erase
FCTL3 = FWKEY;            // Borra Lock (pone a 0)
*Puntero = 0;             // Escribe algo para borrar el segmento
FCTL1 = FWKEY + WRT;      // Activa WRT y borra ERASE
*Puntero= dato;           // Escribe el dato en 0x1000
FCTL1 = FWKEY;            // Borra bit WRT
FCTL3 = FWKEY + LOCK;     //activa LOCK
      __bis_SR_register(GIE);   //Enable interrupt
}
```

9 CONCLUSIÓN

FInalmente y tras toda la explicación del diseño, queda demostrada la viabilidad técnica del desarrollo propio del BMS para un monoplaza de FSAEE, equiparable a muchas soluciones comerciales y con infinita versatilidad. Sin embargo, el diseño anterior no está exento de posibles mejoras, para ello en el siguiente apartado se comentarán varias soluciones y modificaciones a realizar en el futuro. Por último, se analizará la viabilidad económica del proyecto en el presupuesto.

9.1 Líneas futuras del sistema

Como se ha mencionado repetidas veces en este documento, la función del proyecto ha sido sentar una base hardware y software para el desarrollo durante el próximo año, para corregir los errores existentes y mejorar algunos aspectos del diseño.

El funcionamiento de un sistema electrónico en un laboratorio difiere del mostrado en el interior de una batería de 80KW, es por ello que la protección ante EMIs debe ser uno de los objetivos fundamentales a tratar, mediante mejores técnicas de enrutado, blindaje de las señales de comunicación, de las PCBs, etc. Se necesita probar el sistema bajo una fuente de ruido electromagnético que permita comprobar los fallos concretas, y los subsistemas que se puedan ver más afectados.

El uso de los ICs nombrados en el apartado de monitorización de voltaje debería ser considerado si finalmente ese subsistema es víctima del ruido electromagnético, pues una correcta medición de las tensiónes es el requisito mínimo de un BMS seguro.

Otra de las partes vitales que necesita optimización y mejora en el sistema es el consumo. Un consumo de hasta 8W es inadmisible siendo dependiente de una pequeña batería de 12V, que además debe proporcionar alimentación a otros sistemas. La corriente necesaria para la activación de cada optoacoplador, 10mA, es la mayor causante del alto consumo, existiendo 128 circuitos de balanceo y requeriendo 10mA cada uno, se pierden 6.4W si todos se activasen a la vez. Una solución propuesta puede ser la eliminiación de los optoacopladores, y su sustitución por una red de polarización con BJT que permita las mismas funciones. Así mismo, los modos de bajo consumo deben ser implementados en la programación de los microcontroladores y los ADC si se quiere perseguir una optimización completa de la gestión energética.

En el BMS que se desarrolle posteriormente a este, es muy importante la disponibilidad de puertos de entrada o salidas digitales libres para la comunicación mediante pulsos con algún dispositivo, muy útiles para manejar respuestas de seguridad inmediatas, que no requieran codificación. Para ello son necesarios más aisladores digitales que permitan conectarse con los pines libres del microcontrolador del *Master*.

También es fundamental la elección de los conectores correctos, preparados para aplicaciones de automoción y capaces de resistir grandes esfuerzos mecánicos estando inmunizados al ruido.

Por último, el código necesita un amplio testeo y calibración con las celdas que finalmente vayan a ser usadas, puesto que cada batería presenta un comportamiento único, y para mejorar el rendimiento y la seguridad de las mismas es necesario el conocimiento completo del sistema a controlar. Por ejemplo, conociendo las curvas de descarga y la resistencia interna de la batería, se puede crear el modelo de la batería y maximizar su capacidad durante la descarga, o estimar la caida de tensión durante el balanceo que permitan hacerlo de forma más inteligente.

9.2 Presupuesto

El coste de las soluciones comerciales disponibles suele rondar entre los 1200€ y 2000€. Por ello, en esta sección analizaremos el coste total del sistema desarrollado.

Tabla 9-1. Desglose de precios de los componentes

Componente	Coste unitario	Cantidad	Coste total
MSP430G2553 20N	2.12	8	16.96
MSP430G2553 28N	2.80	1	2.80
LM1117-3.3	1.48	9	13.32
AD1582	1.81	9	16.29
AD7988-5	13.81	1	13.81
HTFS-200P/SP2	18.95	1	18.95
SN65HVD251	2.85	1	2.85
ISO7241C	6.21	1	6.21
MCP2515	1.83	1	1.83
LM7805	0.51	1	0.51
TRACO THM101211	45.35	1	45.35
LT1990A	1.41	128	180.48
AD7490	6.09	24	146.16
ADR130	2.09	8	16.72
FDN304P	0.208	128	26.624
TLP293-4	0.669	32	21.408
MAX7313	2.85	16	45.12
Cap 10µF	0.011	67	0.737

Cap 4.7µF	0.017	9	0.153
Cap 1µF	0.05	9	0.45
Cap 0.33µF	0.064	1	0.064
Cap 0.1µF	0.012	151	1.812
Cap 4.7nF	0.0536	1	0.0536
Cap 2.7nF	0.147	1	0.147
Cap 1nF	0.021	9	1.89
Cap 22pF	0.018	11	0.198
Cristal 16Mhz	0.50	1	0.50
LED SMD 0805	0.0765	137	10.4805
Res SMD 0805 560kΩ	0.0064	128	0.8192
Res SMD 0805 10k Ω	0.022	11	0.242
Res SMD 0805 330Ω	0.0045	128	0.576
Res SMD 0805 220Ω	0.0045	128	0.576
Res SMD 0805 180Ω	0.0263	9	0.237
Res SMD 0805 120Ω	0.0263	5	0.1315
Res SMD 0805 20Ω	0.0263	1	0.0263
Res SMD 2512 10Ω	0.363	128	46.464
Interruptor táctil	0.18	9	1.62
Header 18 contactos	0.28	20	5.6
Total			**648.03€**

Pese a que a priori el precio de los componentes pueda parecer elevado, normalmente en el equipo se consigue patrocinio de suministradores de componentes, que pagan la totalidad de los mismos. El precio de la fabricación de las PCBs tampoco se incluye por el mismo motivo.

Igualmente, el precio sin patrocinio de componentes es menos de la mitad del objetivo propuesto por los BMS comerciales, lo que demuestra también la viabilidad económica del proyecto.

10 DOCUMENTOS

EN este apartado se han recopilado las capas de cada PCB con el enrutado completo, así como una recopilación con los puntos de la normativa de la FSAEE 2018 que han afectado al diseño.

10.1 Layout PCB

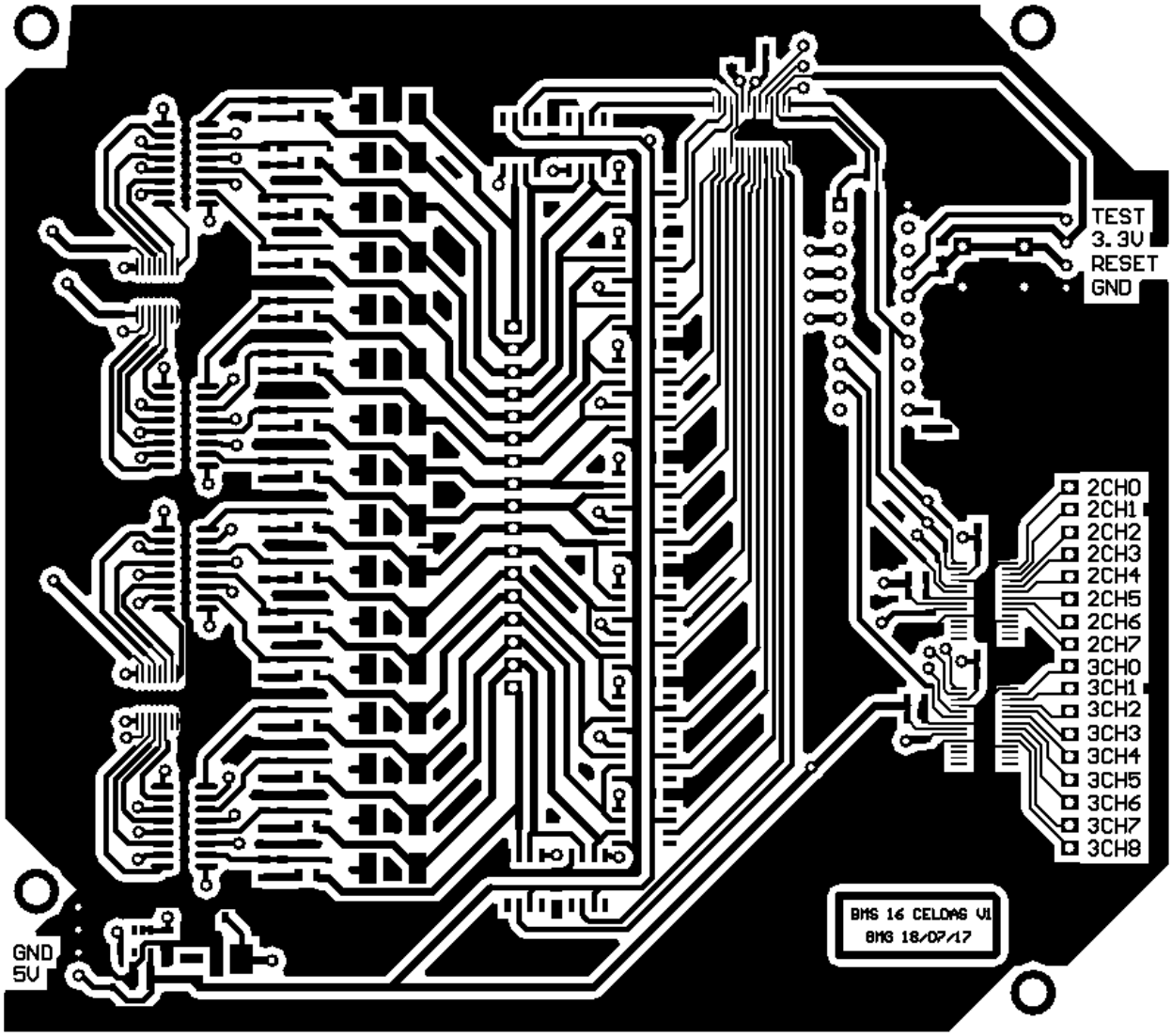

Figura 10-1. Cara superior del Slave

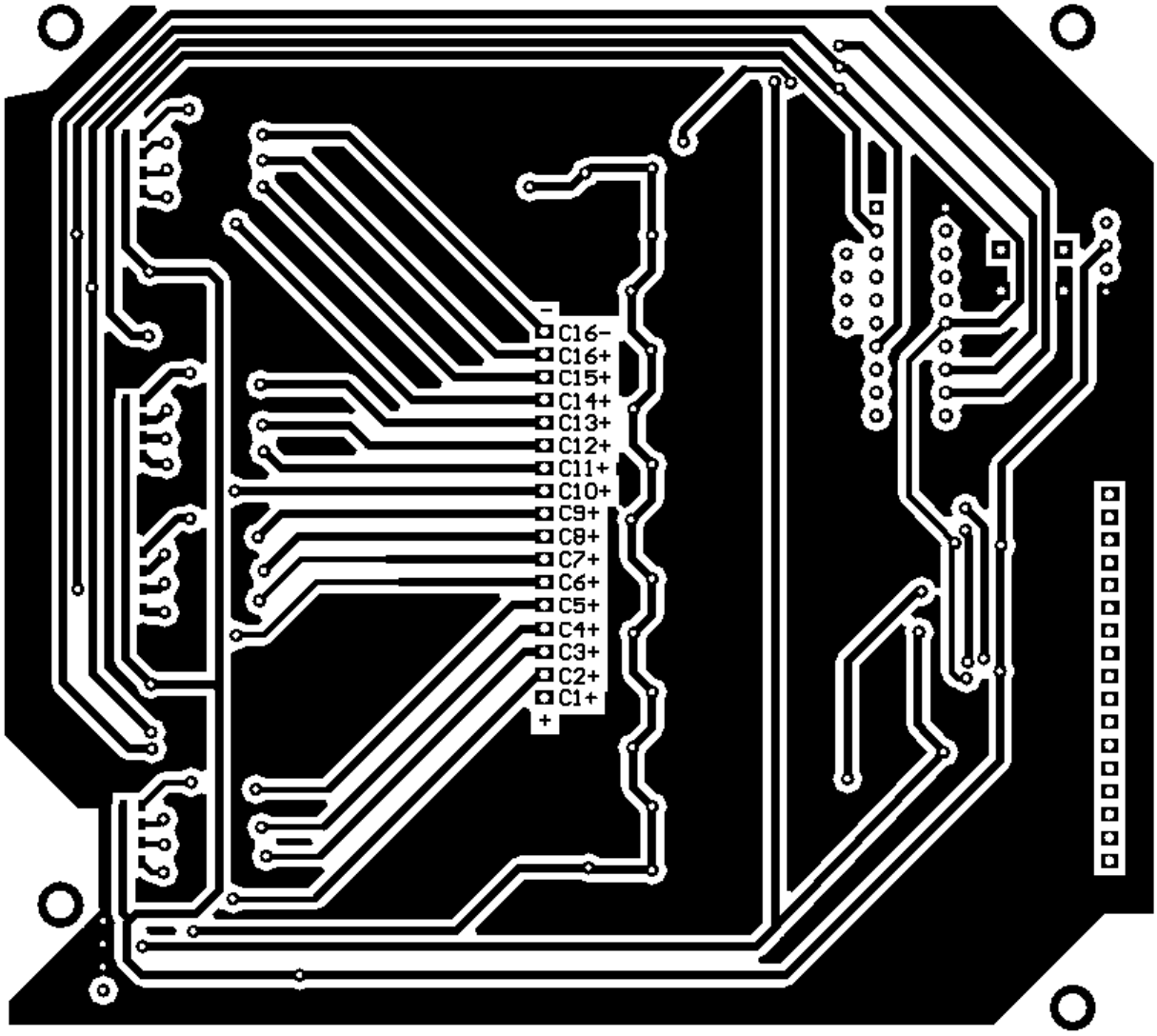

Figura 10-2. Cara inferior del Slave

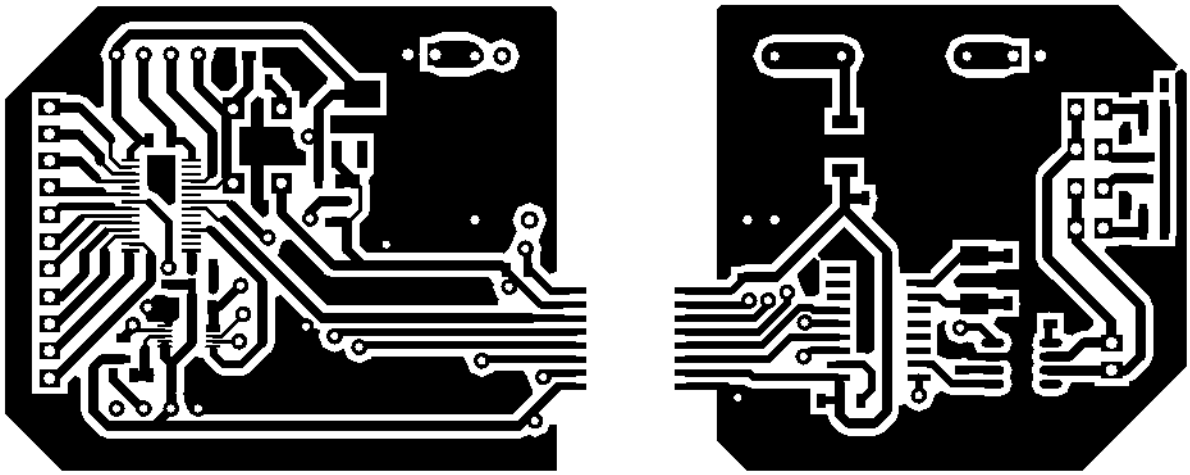

Figura 10-3. Cara superior del Master

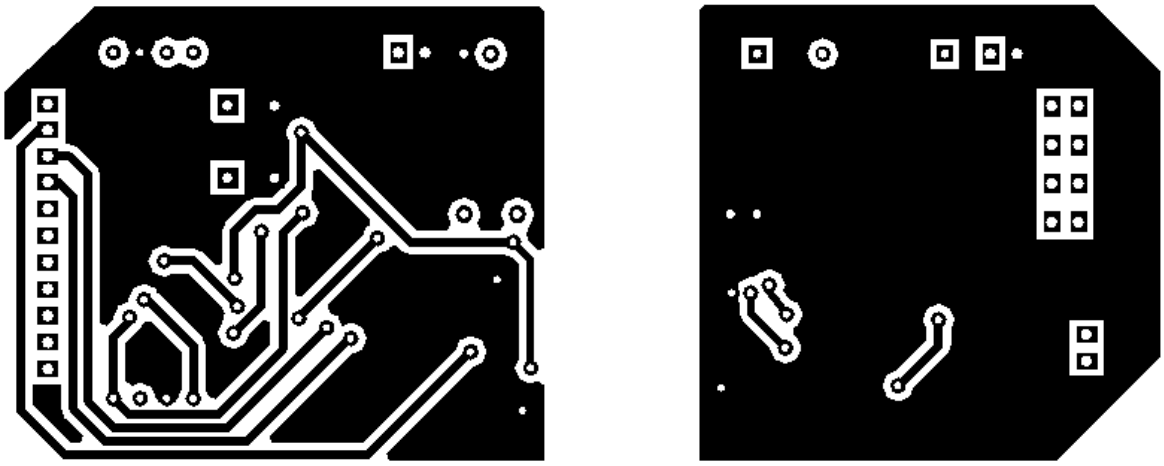

Figura 10-4. Cara inferior del Master

10.2 Normativa FSAEE

Los artículos dispuestos a continuación comprenden los requisitos seguidos en el diseño del BMS

- ***EV3.6*** *Sistema de gestión del acumulador (AMS ó BMS):*

- *EV3.6.1 Cada acumulador debe ser controlado por un sistema BMS siempre que el sistema tractor esté activo o la batería esté siendo cargada.*

- *EV3.6.2 El BMS debe medir continuamente la tensión de todas las celdas, a fin de mantenerlas dentro del rango de tensiones permitido en el datasheet. Para un set de celdas conectadas en paralelo, sólo una medición es necesaria.*

- *EV3.6.3 El BMS debe de medir continuamente la temperatura de puntos críticos del acumulador, a fin de mantener las celdas a menos del límite del datasheet o 60 ºC (el menor de los dos valores). La medición debe de ser realizada en el terminal negativo de cada celda o en el "bus bar", permitiendo un alejamiento máximo de 10 mm del negativo de la celda. Nota: en algunas competiciones se imponen dispositivos especiales para medir la temperatura.*

- *EV3.6.4 Para sistemas centralizados de BMS (una o más celdas por placa), todos los cables de medición de tensión deben de estar protegidos por fusibles link wires o fusibles a fin de no superar la intensidad máxima admisible en caso de cortocircuito. Para BMS distribuidos (una celda por placa) los cables conectados a los terminales positivos no deberán de estar protegidos si miden menos de 25 mm y la PCB tiene protección propia frente a CC. Si un cable se desprende, el BMS deberá de detectarlo y notificar un problema crítico de tensión.*

- *EV3.6.5 Cualquier conexión del GLV al BMS debe de estar galvánicamente aislada del sistema tractor.*

- *EV3.6.6 Para baterías de litio, al menos el 30% de las celdas deberá estar sujeto al control de temperatura del BMS, de manera homogéneamente distribuida.*

- *EV3.6.7 El BMS tiene que abrir los AIRs en caso de recibir valores críticos de tensión o temperatura, según el datasheet de las celdas y la resolución de las medidas. En caso de que el BMS abra los AIR, un LED rojo en el cockpit deberá de iluminarse.*

- ***EV4.1** Separación entre el sistema tractor y el GLV.*

- *EV4.1.3 Los sistemas GLV y tractor deben de estar separados, sin atravesar los mismos agujeros o circular bajo el mismo conducto, a excepción de circuitos de enclavamiento.*

- *EV4.1.6 El espaciado entre GLV y sistema tractor debe de estar claramente definido, impidiendo el movimiento a aquellos elementos que puedan moverse a fin de asegurar dicho espaciado.*

- *EV4.1.7 Cuando GLV y sistema tractor coincidan en la misma PCB, debe de indicarse claramente en qué parte está cada sistema y habrá que dejar un espacio entre los componentes de cada uno:*

Tensión	*En PCB*	*Aire*	*Con Aislamiento*
0 – 50 V DC	*1.6 mm*	*1.6 mm*	*1 mm*
50 – 150 V DC	*6.4 mm*	*3.2 mm*	*2 mm*
150 – 300 V DC	*9.5 mm*	*6.4 mm*	*3 mm*
300 – 600 V DC	*12.7 mm*	*9.5 mm*	*4 mm*

REFERENCIAS

[1] Linear Technologies, [En línea]. Disponible: http://cds.linear.com/docs/en/datasheet/1990fb.pdf.

[2] Texas Instruments, «ti.com», Enero 2014. [En línea]. Disponible: http://www.ti.com/lit/ds/snis170d/snis170d.pdf.

[3] Texas Instruments, «ti.com», Abril 2011. [En línea]. Disponible: http://www.ti.com/lit/ds/symlink/msp430g2553.pdf.

[4] Analog Devices, [En línea]. Disponible: http://www.analog.com/media/en/technical-documentation/data-sheets/AD7490.pdf.

[5] SAE, «fsaeonline.com», 2 Septiembre 2016. [En línea]. Disponible: http://www.fsaeonline.com/content/2017-18%20fsae%20rules%209.2.16a.pdf.

[6] Analog Devices, [En línea]. Disponible: http://www.analog.com/media/en/technical-documentation/data-sheets/AD1582_1583_1584_1585.pdf.

[7] Maxim Integrated, [En línea]. Disponible: https://datasheets.maximintegrated.com/en/ds/MAX7317.pdf.

[8] Analog Devices, [En línea]. Disponible: http://www.analog.com/media/en/technical-documentation/data-sheets/AD7988-1_7988-5.pdf.