DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS ELECTRÓNICOS

TRABAJO FIN DE MÁSTER

TECNOLOGÍAS DE ALMACENAMIENTO E HIBRIDACIÓN APLICADOS AL AUTOCONSUMO

AUTOR: SAMUEL SÁNCHEZ BAEZA
TUTOR: FRANCISCO JOSÉ GIMENO SALES

SEPTIEMBRE, 2018

RESUMEN

El presente Trabajo Fin de Máster se basa en una pequeña parte del proyecto ALHACENA, desarrollado en el Instituto Tecnológico de la Energía (ITE). En este trabajo, se lleva a cabo la gestión de las baterías de Litio empleadas en el proyecto, además de la visualización de los datos de interés de dichas baterías a través de una aplicación de escritorio programada bajo el entorno de programación Visual Studio. El trabajo se divide en varios puntos:

Por un lado, se realiza el modelado de las baterías a emplear en el proyecto. Para ello se realizan una serie de ensayos que tendrán como objetivo lograr los parámetros de las baterías que fueran de interés[1], de esta forma se obtendrá el comportamiento de las baterías tanto en carga como descarga, otorgando un modelado completo de las baterías.

Por otro lado, se cuenta con la programación de un microcontrolador de la familia STM32F107x, empleando para ello el entorno de programación Keil uVision 4 bajo el lenguaje de programación C. Siendo más específicos, se va a emplear el microcontrolador STM32f107VC presente en la placa de evaluación MCBSTM32C. Tras aplicarle el algoritmo desarrollado, se va a disponer de comunicaciones por medio de CAN y UART para comunicar con la aplicación de escritorio y el Battery Management System (BMS) respectivamente. Este último (BMS), es caracterizado por la placa de Texas Instruments bq76PL455. Además, el micro es capaz de realizar los cálculos necesarios para obtener datos de interés como el SOC (State Of Charge) de las baterías, el SOH (State Of Health) de las mismas, tensiones del pack de baterías, etc.

Finalmente, se realiza la programación de la aplicación de escritorio para PC, esta se encuentra desarrollada en Visual Studio por medio del lenguaje C# y muestra los parámetros de las baterías, además de dar la posibilidad de modificar alguno de estos. También se dispondrá de un sistema de aviso de errores en el caso de que se produzca alguno de ellos.

Palabras clave: baterías, microcontrolador, Keil uVision, aplicación de escritorio, Visual Studio, modelado de baterías, algoritmo, SOC, SOH.

[1] Los parámetros a obtener serán los pertenecientes a las tensiones y corrientes de las baterías para cada ratio de corriente y temperatura establecido en los ensayos.

RESUM

El present Treball Fì de Màster es basa en una xicoteta part del projecte ALHACENA, desenvolupat en el “Instituto Tecnológico de la Energía” (ITE). En aquest treball es du a terme la gestió de les bateries de Liti empleades en el projecte, a més de la visualització de les dades d'interés de les dites bateries a través d'una aplicació d'escriptori programada davall l'entorn de programació Visual Studio. El treball es dividix en diversos punts:

D'una banda, es realitza el modelatge de les bateries a emprar en el projecte. Per a això es realitzen una sèrie d'assajos que tindran com a objectiu aconseguir els paràmetres de les bateries que foren d'interés, d’aquesta forma s’obtindrà el comportament de les bateries tant en càrrega com descàrrega, atorgant un modelatge complet de les bateries.

D'altra banda, es compta amb la programació d'un microcontrolador de la família STM32F107x, emprant per a això l'entorn de programació Keil uVision 4 davall el llenguatge de programació C. Sent més específics, es va a emprar el microcontrolador STM32f107VC present en la placa d'avaluació MCBSTM32C. Després d'aplicar-li l'algoritme desenvolupat, es va a disposar de comunicacions per mitjà de CAN i UART per a comunicar amb l'aplicació d'escriptori i el Battery Management System (BMS) respectivament. Este últim (BMS), és caracteritzat per la placa de Texas Instruments bq76PL455. A més, el micro és capaç de realitzar els càlculs necessaris per a obtindre dades d'interés com el SOC (State Of Charge) de les bateries, el SOH (State Of Health) de les mateixes, tensions del pack de bateries, etc.

Finalment, es realitza la programació de l'aplicació d'escriptori per a PC, esta es troba desenvolupada en Visual Studio per mitjà del llenguatge C# i mostra els paràmetres de les bateries, a més de donar la possibilitat de modificar algun d'estos. També es disposarà d’un sistema d’avís de errades en el cas que es produïsca algun d’ells.

Paraules clau: bateries, microcontrolador, Keil uVision, aplicació d'escriptori, Visual Studio, modelatge de bateries, algoritme, SOC, SOH.

ABSTRACT

The present Master Final Project is based on a small part of the ALHACENA project, developed in the “Instituto Tecnológico de la Energía” (ITE). In this work, the management of the lithium batteries used in the project is carried out, in addition to the visualization of the data of interest of the batteries through a desktop application programmed in the Visual Studio programming environment. The work is divided into several points:

On the one hand, the modeling of the batteries to be used in the project is carried out. For this, a series of tests are carried out that will have as objective to achieve the parameters of the batteries that were of interest, in this way the behavior of the batteries both in charge and discharge will be get, granting a complete modeling of the batteries.

On the other hand, there is programming of a microcontroller of the STM32F107x family, using the Keil uVision 4 programming environment under the C programming language. More specific, the STM32f107VC microcontroller present on the MCBSTM32C evaluation board will be used. After applying the algorithm developed, communications will be available through CAN and UART to communicate with the desktop application and the Battery Management System (BMS) respectively. The latter (BMS), is characterized by the Texas Instruments board bq76PL455. In addition, the microcontroller is able to perform the necessary calculations to obtain data of interest such as the SOC (State Of Charge) of the batteries, the SOH (State Of Health) of the same, voltages of the battery pack, etc.

Finally, the programming of the PC desktop application is done, it is developed in Visual Studio through the C# language and shows the parameters of the batteries, besides giving the possibility of modifying some of these. An error warning system is also available in case of any them occurs

Keywords: batteries, microcontroller, Keil uVision, desktop application, Visual Studio, battery modeling, algorithm, SOC, SOH.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE SIGLAS

ADC: Conversor Analógico Digital

AH: Amperios Hora

BMS: Battery Management System

CRC: Cyclic Redundancy Check

CAN: Controller Area Network

CL-DES: Controlador Local de unidades de almacenamiento energético

GPIO: General Purpose Input/Output

IDE: Entorno de Desarrollo Integrado

ITE: Instituto Tecnológico de la Energía

I2C: Inter-Integrated Circuit

JTAG: Join Test Action Group

LCD: Liquid Cristal Display

OCV: Open Circuit Voltage

SOC: State Of Charge

SOH: State Of Health

SPI: Serial Peripherial Interface

SRAM: Static Random Access Memory

TFT: Thin Film Transistor

TI: Texas Instruments

UART: Universal Asynchronous Receiver-Transmitter

USB: Universal Serial Bus

VOC: Tensión en circuito abierto.

VREF: Tensión de referencia

WIFI: Wireless Fidelity

1. INTRODUCCIÓN

En los últimos años el mercado energético se está orientando hacia la descarbonización global, generando un clima adecuado tanto para la inversión en las distintas energías renovables como en centrales con bajas emisiones, necesarias para garantizar la totalidad del suministro. Una de las principales preocupaciones hoy en día consiste en alcanzar un mundo “más limpio”, es por ello por lo que la mayoría de los países apuestan por las energías renovables, enfocando el actual mercado energético a estos sectores. Al sector de las energías renovables, en los últimos años también se están empezando a añadir las baterías para almacenamiento, en especial las baterías de Litio.

El presente Trabajo Fin de Máster surge a través del proyecto ALHACENA, el cual se encuentra embarcado dentro del ámbito del mercado energético. En él se desarrollan y validan diferentes escenarios de balance de energía inteligente, empleando para ello tecnologías de almacenamiento energético que faciliten la gestión de la demanda y contribuyan a la estabilidad en la red, evitando picos de demanda.

El objetivo de este proyecto es acabar desarrollando una herramienta de uso por los usuarios finales que sea capaz de realizar tomas de decisiones. Estas decisiones tendrán como prioridad ser transformadas en ahorros económicos en las facturas eléctricas, tanto para empresas como para particulares.

Los resultados que se esperan obtener tras la realización del proyecto son los siguientes:

- Desarrollo de un sistema de gestión universal de servicios de almacenamiento energético con hibridación de tecnologías.
- Sistema de gestión de servicios de almacenamiento energético en la nube.
- CL-DES. Controlador local de los sistemas de almacenamiento distribuidos.
- Banco de pruebas para celdas de litio poliméricas (tecnología de almacenamiento propia).
- Estudio sobre la respuesta de las distintas combinaciones de tecnologías de almacenamiento ante distintos escenarios.
- Estudio de la viabilidad económica de implantar la tecnología desarrollada en el presente proyecto para posibilitar el autoconsumo con almacenamiento hibridado.
- Estudio sobre la integración de la tecnología desarrollada en el presente proyecto en los distintos mercados eléctricos gestionados por el operador del mercado eléctrico (OMIE).

Como ya se ha mencionado anteriormente, este Trabajo Fin de Máster tan solo se centra en una parte de este proyecto, concretamente en el mantenimiento de las baterías empleadas.

1.1 Diagrama de bloques

Para conocer cada uno de los dispositivos que componen el sistema y con el objetivo de facilitar al lector el entendimiento de cada una de las funciones de cada dispositivo, se ha decidido incluir un diagrama de bloques.

El diagrama de bloques correspondiente al presente proyecto sería el que se puede visualizar en la siguiente figura:

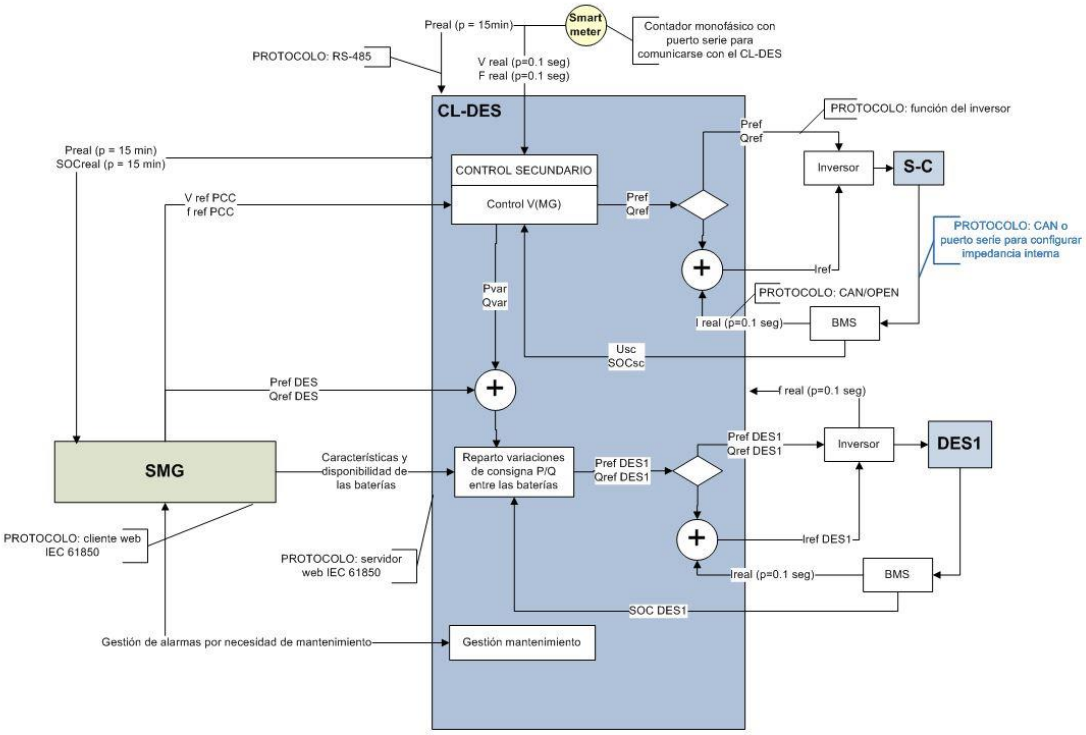


Figura 1. Diagrama de bloques del proyecto ALHACENA

Sin embargo, este diagrama de bloques sería demasiado amplio, ya que el presente Trabajo Fin de Máster tan solo se centraría en una pequeña parte de la totalidad del proyecto, siendo esta parte que se menciona la correspondiente al BMS.

Centrando el diagrama de bloques un poco más en el presente Trabajo Fin de Máster, se ha realizado otro diagrama de bloques más específico. En él se puede observar cómo se cuenta principalmente con 3 dispositivos en el sistema, siendo

las comunicaciones entre los distintos dispositivos las que se pueden observar en la Figura 2.

- Bq76PL455.
- MCBSTM32C.
- PC.

Figura 2. Diagrama de bloques del presente Trabajo Fin de Máster

A primera vista puede parecer un diagrama demasiado simple, pero al fin y al cabo muestra los aspectos estrictamente necesarios del presente trabajo. Visualizándose cada uno de los principales dispositivos que entran en funcionamiento en el proyecto y la comunicación existente entre ellos. Algo mucho más visual y fácil de entender para el lector que la Figura 1.

Una vez conocidos los dispositivos presentes en el actual trabajo se puede entrar más a fondo en los dispositivos que componen el proyecto. En primer lugar, el proyecto se planteó con la incorporación de un BMS de desarrollo propio para gestionar el mantenimiento de las baterías. Sin embargo, por problemas de logística finalmente se acabó planteando la opción de emplear una placa desarrollada por Texas Instruments (bq76PL455) que actuara como BMS. Esta se comporta como un monitor de las celdas de baterías que se conecten a ella, es capaz de medir los niveles de tensión de cada celda y darlos a conocer a través de una comunicación por medio de UART, además de otros datos de interés que serán mencionados más tarde. Cada placa es capaz de conectar 16 celdas, sin embargo, es posible aumentar el número de celdas conectando varias bq76PL455 en serie, hasta un máximo de 96 celdas.

Para realizar la comunicación por UART y recibir los datos que se envíen, se programa la placa de evaluación MCBSTM32C. Esta va a contar con un algoritmo que sea capaz de recibir por medio de comunicaciones UART las tensiones y temperaturas presentes en cada celda del pack de baterías y realizará los cálculos necesarios para obtener otros datos de interés como el SOC o el SOH de las baterías.

En el proyecto original, los datos de las baterías serán enviados por medio de comunicaciones CAN a la plataforma CL-DES y esta será la encargada de llevar a cabo la toma de decisiones, dependiendo de las órdenes que se le envíen desde una plataforma en la nube, véase Figura 1 para entender con mayor claridad los elementos que componen el sistema.

Puesto que la plataforma CL-DES no forma parte del presente Trabajo Fin de Máster, se ha tomado la decisión de realizar en este punto una modificación. Los datos que se reciban de las baterías, junto con los cálculos necesarios, se seguirán enviando a través de comunicaciones CAN, con la diferencia de que el receptor de estos datos sea una aplicación de escritorio, desarrollada con el objetivo de visualizar los parámetros que tengamos presentes en las baterías.

Desde esta aplicación para PC se podrá visualizar y modificar parámetros de las baterías. Además, servirá de ayuda para detectar los posibles errores que se puedan estar produciendo, ya que tiene un apartado de advertencias desde el que se podrá visualizar el error que se está generando en las baterías conectadas al bq76PL455 y actuar sobre dicho error a tiempo.

2. OBJETIVOS

Una vez claros los objetivos del proyecto ALHACENA, además de las partes que se llevarán a cabo dentro del Trabajo Fin de Máster, se puede realizar una valoración de cuáles serán los objetivos tanto generales como específicos en la realización del Trabajo Fin de Máster, siendo ligeramente distintos a los del proyecto original.

Objetivos generales:

- Realizar un modelado de las baterías de litio.
- Implementar, analizar y evaluar los algoritmos programados tanto para el entorno de desarrollo Keil uVision 4 como para Visual Studio.
- Ser capaces de emplear las baterías de litio para cualquier finalidad mientras se visualizan sus parámetros, determinando que estos siempre se encuentran dentro de los límites adecuados para asegurar un buen estado de salud de las baterías.

Objetivos específicos:

- Aplicación de escritorio:
  - Establecer una comunicación vía CAN.
  - Desarrollar una aplicación de PC donde visualizar y modificar los parámetros de las baterías.
  - Desarrollar una aplicación de PC que sea capaz de avisar de posibles errores que se puedan producir en las baterías, otorgando una forma de protección ante los posibles fallos que puedan ocurrir.
- Programación del microprocesador:
  - Manejo del entorno de programación Keil uVision 4.
  - Conocer las posibilidades de programación de la placa MCBSTM32C.
  - Establecer una comunicación vía CAN.
  - Establecer una comunicación vía UART.
- Modelado de las baterías:
  - Adquirir conocimientos acerca del modelado de baterías de litio.
  - Obtener las curvas necesarias para aplicar el modelado de baterías en el Firmware desarrollado.

3. MATERIAL Y SOFTWARE EMPLEADO

En este apartado se van a exponer cada uno de los materiales que se van a emplear en la realización del Trabajo Fin de Máster, seguido de una breve descripción del mismo para entender su función dentro del proyecto.

3.1 Bq76PL455A-Q1

El módulo de evaluación BQ76PL455 (Figura 3) es un módulo desarrollado por Texas Instruments que proporciona el balanceo y monitorización de un pack de baterías hasta un total de 16 celdas. Su rango de tensión de operación comprende desde los 16 V hasta los 79,2 V (alimentándose directamente de las baterías que se conectan a la placa) y cuenta con las siguientes características:

- 16 canales para monitorización de voltaje por celda y balanceo pasivo[2].
- 8 canales analógicos de medida de temperaturas o sensados auxiliares.
- 6 canales digitales.
- Comunicación serie aislada.
- Posibilidad de aumentar el número de celdas hasta 96.

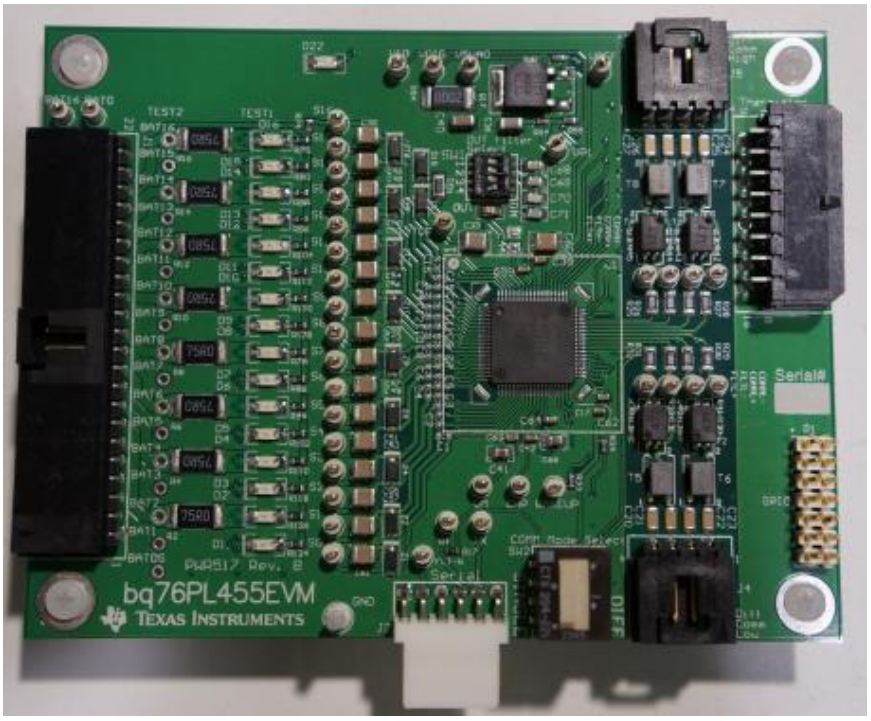


Figura 3. Módulo de evaluación BQ76PL455EVM

[2] El balanceo pasivo consiste en la descarga de la celda más cargada de un pack de baterías sobre una resistencia, equilibrando de esta forma la energía presente en cada celda. Por otro lado, el balanceo activo carga la celda que se encuentra menos cargada, inyectando sobre ella la energía sobrante de la más cargada.

Para proyectos en los que se desee contar con más de 16 celdas, se puede recurrir a la misma placa de evaluación, ya que es posible aumentar el número de celdas a sensar hasta 96 celdas colocando 6 módulos de evaluación en serie. Para ello se deberá hacer uso de los conectores de comunicación “High” y “Low” situados en los laterales de la placa de evaluación.

Para realizar la comunicación entre el PC y el bq76PL455, que en algunas ocasiones se podrá determinar como BMS de ahora en adelante, es necesario contar con el cable FTDI USB-to-TTL (5V), el cual viene incluido en la caja del BMS.

Analizando las especificaciones de la placa, se observa que el BMS cuenta con las siguientes características:

Componentes:

- Convertidor analógico-digital de 14 bits (ADC).
- Referencia de tensión de precisión.
- Extremo análogo de alta tensión (AFE).
- Interfaz de comunicación serial universal receptor / transceptor asíncrono (UART).
- Regulador de voltaje LDO.
- Lógica de control para funciones de monitorización, equilibrado y comunicación.

Características eléctricas:

- Rango de tensión de funcionamiento de 16 V a 79,2 V.
- Mide hasta 16 celdas de 1 V a 5 V.
- Máximo voltaje de circuito abierto por celda de 5.5 V
- Corriente de equilibrio de hasta 56 mA a 4,2 V.
- Temperatura de funcionamiento desde -40 ° C a 105 ° C.

Funciones:

- Monitorización del voltaje de celda.
- Comparadores de OV (Overvoltage) o UV (Undervoltage).
- Equilibrado de celdas de baterías.
- Control de temperaturas y señales auxiliares.
- Control integrado.
- Monitorización de fallos.
- Comunicaciones diferenciales aisladas.
- Comunicaciones serie a través de PC.
- Fuente de alimentación.

- Manejo de entradas y salidas (GPIO).

Estas funciones se pueden llevar a cabo gracias al hardware de la placa. En la siguiente figura, se puede observar un diagrama de boques funcional de la placa:

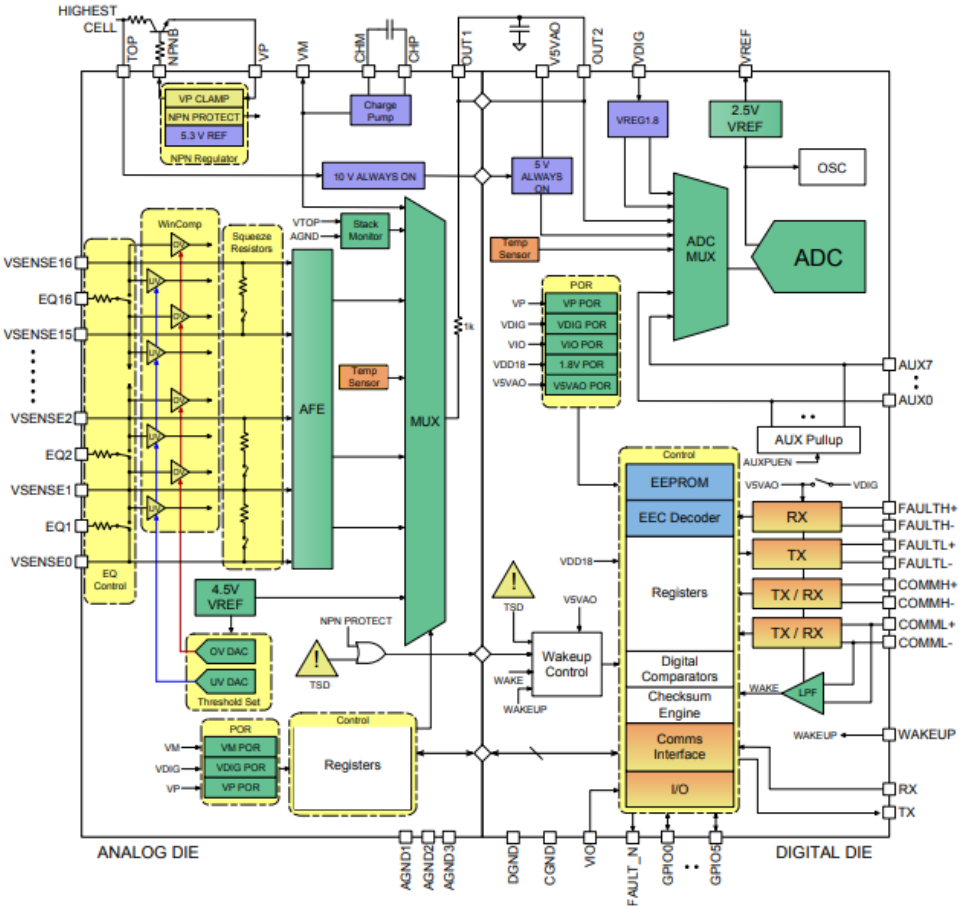


Figura 4. Diagrama de bloques funcional de la placa

Este módulo garantiza una operación de funcionamiento segura y fiable, sin embargo ha de tenerse en cuenta que los sistemas de almacenamiento pueden verse afectados con el paso del tiempo por la mala gestión de los ciclos de carga y descarga de las celdas de baterías empleadas. Esta mala gestión se alcanzará por las variaciones individuales (por celda) que se producen en el voltaje, capacidad de carga e impedancia interna de las baterías, que acaban provocando una reducción de la capacidad efectiva y eficiencia de la batería. Es por tanto imprescindible que si se detectan esos pequeños fallos se detengan los ensayos que se estén realizando con esas baterías y se cambien por unas nuevas.

Con el objetivo de averiguar cómo son las tramas que más adelante se deberán utilizar para comunicar con el BQ76PL455EVM, se ha procedido a la

lectura de datasheet y los demás documentos de información acerca de la placa de evaluación [15 y 16][3].

En estos documentos se puede ver cómo acceder a los registros, siendo este un aspecto importante para realizar la configuración necesaria para pedir mediante los comandos los datos que se desean leer.

Para proceder a la comunicación, se debe hacer uso del protocolo comando y respuesta que se propone desde Texas Instruments, este protocolo habilita a un host (en el caso del presente proyecto el microcontrolador) para comunicar con una o más placas de evaluación BQ76PL455EVM, siendo imposible que la placa de evaluación envíe una respuesta hasta que no se haya enviado previamente una trama de comando (petición de lectura).

Cuando se habla de las tramas que puede enviar/recibir el microcontrolador se distingue entre 5 tipos:

- Trama de inicialización.
- Dirección del dispositivo.
- Dirección de registros.
- Datos.
- Cyclic Redundancy Check (CRC).

Trama de inicialización

En la trama de inicialización será donde se distinga si se trata de un comando o una respuesta, esta trama debe ser siempre el primer byte de la trama.

La forma de diferenciar si se trata de una trama u otra será observando si el bit 7 del primer byte tiene su estado en 0 o 1. El bit 7 a 1 indicará que se trata de una trama comando mientras que el mismo bit a 0 indicará que la trama es una respuesta, véase Figura 5.

	7	6	5	4	3	2	1	0
Command Frame Init	FRM_TYPE = 1	REQ_TYPE			ADDR_SIZE	DATA_SIZE		
Response Frame Init	FRM_TYPE = 0	RESP_BYTES-1						

Figura 5. Tipos de tramas en el bq76PL455

Para el caso de una respuesta, en el resto de bits tan solo se indicará el número de bytes contenidos en la respuesta -1. Para el caso del presente Trabajo Fin de Máster, como se cuenta con 16 celdas, se leerá en RESP_BYTES-1 un 31.

[3] El número entre corchetes indica la numeración de las referencias bibliográficas empleadas para la obtención de la información. Véase apartado *10*. *Referencias bibliográficas*.

Una vez sabida la estructura de cada trama, se procede a montar las tramas para proceder a la petición mediante UART de los datos referentes a las tensiones de celda.

Para ello la trama empleada será la siguiente:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	0	0	0	0	0	1

Lo que equivale a:

- FRM_TYPE: command frame.
- REQ_TYPE: single device write with response.
- ADDR_SIZE: 8 bit register address.
- DATA_SIZE: 1 byte.

Trama comando: en el presente proyecto se cambia la dirección del dispositivo y se selecciona dispositivo 0, esto es simplemente una forma de tener localizada la dirección de cada dispositivo. Se ha escogido el 0 pero se podría haber escogido cualquier otro número (dentro de los límites) para direccionar el dispositivo. En el caso de querer modificarlo, desde los registros se puede cambiar el número de la dirección del dispositivo.

Un ejemplo de una trama comando se puede observar en la siguiente figura, donde está presente la configuración del dispositivo como dirección 1, se selecciona el registro 2 y se selecciona una visualización síncrona de cada canal.

Siguiendo este ejemplo, será sencillo realizar la configuración del dispositivo empleado en el Trabajo Fin de Máster, obteniendo la trama de configuración que se podrá visualizar en otros apartados más adelante.

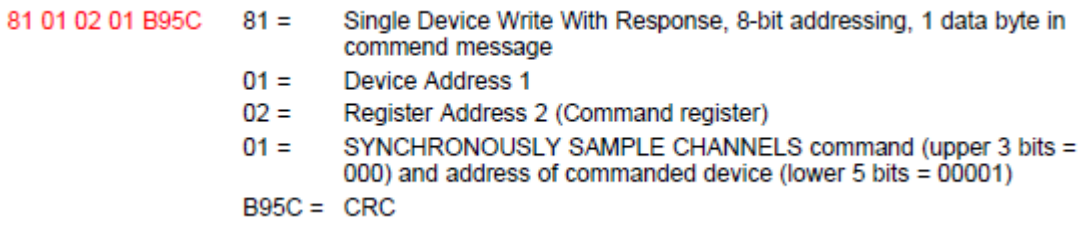
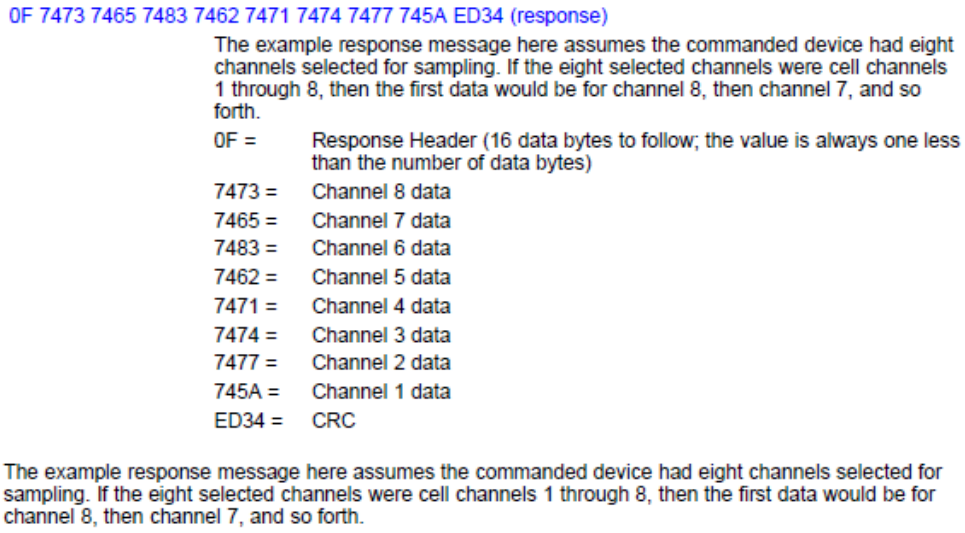


Figura 6. Ejemplo de una trama de comando para el bq76PL455

Trama respuesta:



0F 7473 7465 7483 7462 7471 7474 7477 745A ED34 (response)

The example response message here assumes the commanded device had eight channels selected for sampling. If the eight selected channels were cell channels 1 through 8, then the first data would be for channel 8, then channel 7, and so forth.

0F =	Response Header (16 data bytes to follow; the value is always one less than the number of data bytes)
7473 =	Channel 8 data
7465 =	Channel 7 data
7483 =	Channel 6 data
7462 =	Channel 5 data
7471 =	Channel 4 data
7474 =	Channel 3 data
7477 =	Channel 2 data
745A =	Channel 1 data
ED34 =	CRC

The example response message here assumes the commanded device had eight channels selected for sampling. If the eight selected channels were cell channels 1 through 8, then the first data would be for channel 8, then channel 7, and so forth.

Figura 7. Ejemplo de una trama de respuesta para el bq76PL455

En la respuesta se tiene la trama de inicialización en el primer byte, donde se indica que se trata de una respuesta. A continuación aparecen los datos de respuesta, estos se envían en orden descendente, desde el mayor canal que se haya seleccionado anteriormente en los registros hasta el menor. Es decir, si se habían seleccionado 13 canales se irá en orden descendente desde el canal 13 hasta el canal 0. En el caso del ejemplo de la Figura 7 se tiene desde el 8 al 0. Para el proyecto a realizar se deberá ir desde el canal 16 hasta el 0, se deberá acceder al registro del número de canales y modificarlo para indicar que se tienen 16 canales.

Como se puede observar, para cada canal se obtiene un valor de lectura en valor hexadecimal indicando el nivel de tensión de cada canal. El valor que se observa en la respuesta corresponde a la lectura directa del ADC, por tanto se debe hacer un cálculo matemático para poder calcular la tensión que se tiene en cada celda.

Para ello se aplica la siguiente fórmula[4]:

$$Vcell = \left(\frac{2 * VREF}{65535}\right) * READ\_ADC\_VALUE \quad (0)$$

Observando la fórmula anterior se determina que el valor que devuelve la placa de evaluación en la respuesta corresponde a la lectura del ADC mientras que la

[4] La fórmula a emplear para el cálculo de la temperatura por celda será la misma, ya que la adquisición de datos se realiza de la misma forma.

Vref es fija a 2,5 V, siendo esta una tensión generada directamente sobre el pin VREF, tal y como se puede ver en la siguiente figura:

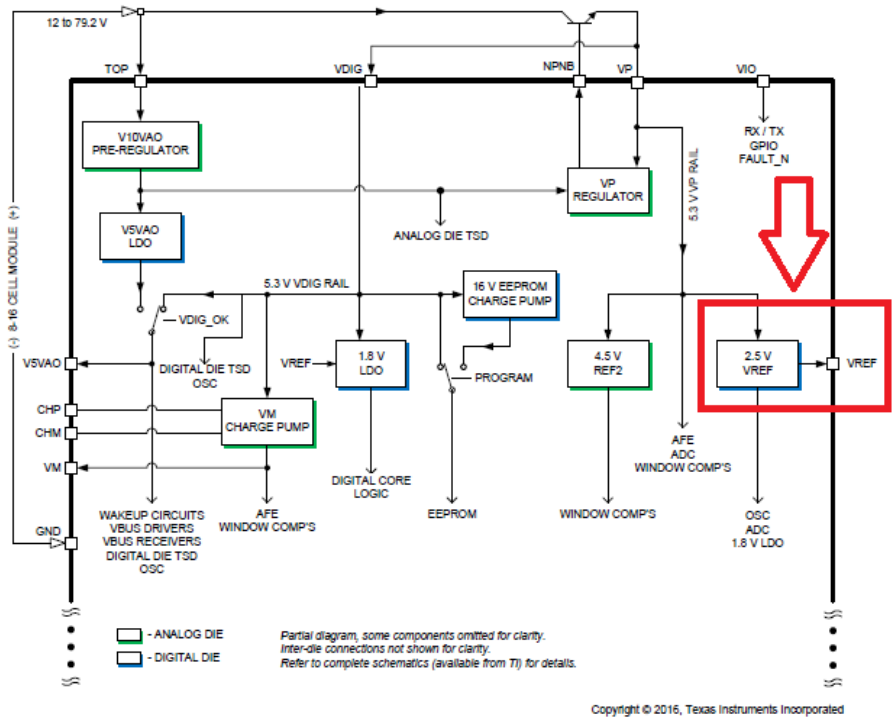


Figura 8. Diagrama de bloques del bq76PL455

Con el objetivo de aclarar cada uno de los cálculos en la adquisición de datos de las tensiones de celda, se va a realizar un breve ejemplo. Para ello se emplearán los datos que pueden visualizarse en la siguiente figura:

81 02 02 20 8944

81 = Single Device Write With Response (8-bit register addressing)
02 = Device Address 2
02 = Register Address 2 (Command register)
20 = READ SAMPLED VALUES command
8944 = CRC

0B 99B7 998C 99B2 99B3 99B0 99BF 2CB1 (response) (1)

The response here contains the data for selected channels from device 2. In the case of this response, the Command Channel Select register (address 3 through 6) was set to 0x05550000, which selects all odd cell channels from 1 to 11 and no AUX nor ancillary channels. This is a total of six cell channels. Two bytes of data are returned for each channel. The data are as follows:

0B = Response header byte. The most significant bit in a response message header byte is always 0, and the other bits represent the number of data bytes in the packet minus 1 (that is, in this case, 12 bytes of data bytes for 6 cell channels). 0x0B = 11, which is 12 – 1.
99B7 = Channel 11 data (3.0022 V)
998C = Channel 9 data (2.9990 V)
99B2 = Channel 7 data (3.0019 V)
99B3 = Channel 5 data (3.0019 V)
99B0 = Channel 7 data (3.0017 V)
99BF = Channel 5 data (3.0029 V)
2CB1 = CRC

Figura 9. Ejemplo de recepción de las tensiones de celda en el bq76PL455

Almacenando cada uno de los valores hexadecimales del ejemplo y convirtiéndolos a un valor decimal, se obtienen los siguientes valores:

Valor hexadecimal	Valor decimal
99B7	39351
998C	39308
99B2	39346
99B3	39347
99B0	39344
99BF	39359

Tabla 1. Valor (hexadecimal y decimal) de las tensiones de celda

Aplicando la fórmula (0), se podrán realizar los siguientes cálculos para cada celda, observando cómo de esta forma se obtiene cada una de las tensiones de celda que se observan en el ejemplo.

CHANNEL11: $Vcell = \left(\frac{2*2.5}{65535}\right) * 39351 = 3.0022\ V$

CHANNEL 9: $Vcell = \left(\frac{2*2.5}{65535}\right) * 39308 = 2.9990\ V$

CHANNEL 7: $Vcell = \left(\frac{2*2.5}{65535}\right) * 39346 = 3.0019\ V$

3.2 MCBSTM32C

La placa de evaluación MCBSTM32C viene equipada con el dispositivo STM32F107VC, perteneciente a la familia de microcontroladores STM32F107x. Esta placa contiene una gran variedad de componentes hardware para un sistema STM32x de un solo chip.

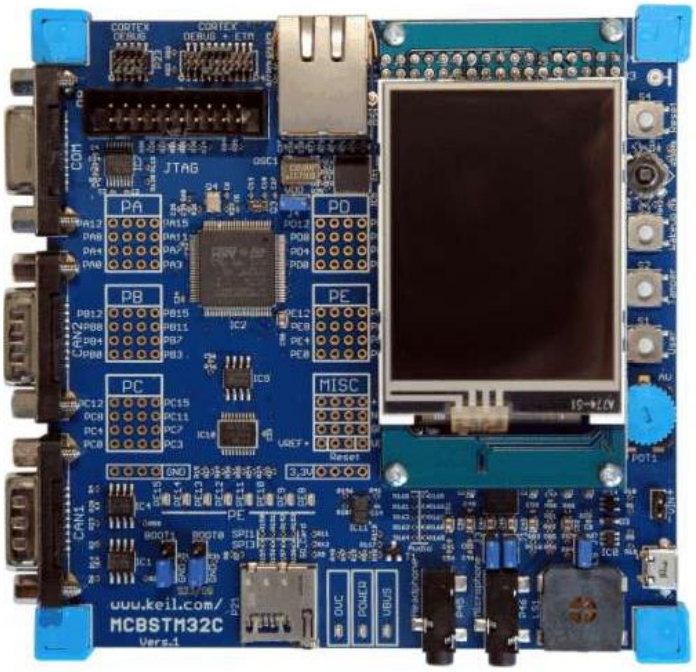


Figura 10. Placa MCBSTM32C

Esta placa (Figura 10), está basada en un procesador Cortex-M3 para generar aplicaciones en microcontroladores de STMicroelectronics. Las características de esta placa son las siguientes:

- Puerto serial: hay un conector estándar DB9 en la placa para las interfaces UART integradas del STM32F107. Este es un puerto serie dúplex completo.
- Puertos CAN: 2 conectores estándar DB9 para aplicaciones que requieren comunicaciones CAN.
- Puerto USB: conector USB tipo B estándar para aplicaciones de comunicaciones USB.
- Puerto Ethernet 100 / 10M: conector RJ45 estándar en la placa MCBSTM32C se conecta a un transceptor Ethernet incorporado para aplicaciones que requieren comunicaciones Ethernet.
- Pantalla LCD con pantalla táctil: pantalla LCD en color desmontable, 240x320 TFT, con pantalla táctil resistente de 4 hilos. Puede usar este

dispositivo de visualización gráfica para mostrar los mensajes de estado del programa y la depuración en tiempo real.

- CODEC estéreo y amplificador: CODEC estéreo de baja potencia con amplificador de auriculares y micrófono. Se puede usar este dispositivo a bordo para aplicaciones de audio.
- Sensor de movimiento: acelerómetro de 3 ejes de baja potencia a través de la interfaz I2C / SPI. Se puede usar este dispositivo a bordo para aplicaciones que requieren la detección de movimiento.
- Batería a bordo: el MCBSTM32C incluye una batería extraíble de litio para mantener la energía del reloj en tiempo real cuando la placa se desconecta de la fuente de alimentación principal.
- Control de voltaje analógico para entrada ADC: fuente de voltaje analógico ajustable para probar el convertidor analógico a digital integrado en el dispositivo STM32F107.
- Conector de tarjeta microSD: conector de tarjeta MicroSD para desarrollar aplicaciones que requieren acceso a tarjetas MicroSD.
- Descargar y depurar JTAG y Cortex / ETM: la placa MCBSTM32C incorpora una interfaz JTAG y una interfaz Cortex Debug + ETM. Cuando se combina con el adaptador ULINK2 USB-JTAG, la interfaz Serial Wire JTAG permite la programación y depuración de flash. Con el adaptador ULINKPro, la interfaz Cortex Debug / ETM permite la programación flash y la depuración del rastreo de instrucciones.

Todas estas funciones que presenta la placa se llevan a cabo mediante el microcontrolador STM32F107VC, el cual funciona a una frecuencia de 72 MHz. Cuenta con una memoria flash de hasta 256 Kbytes y una SRAM 64 Kbytes, además de entradas y salidas mejoradas y periféricos conectados a 2 buses APB. Las características completas se pueden observar en la siguiente figura.

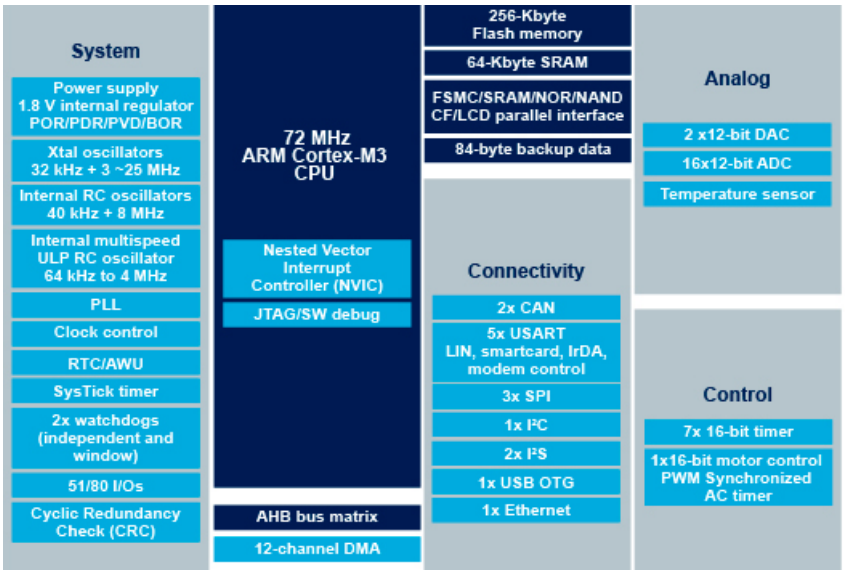


Figura 11. Características del STM32F107VC

En cuanto a la distribución de los pines, esta se observa en la Figura 12. Sin embargo, no se utilizarán todos los pines, tal y como se podrá observar en el apartado "*STM32CubeMX*".

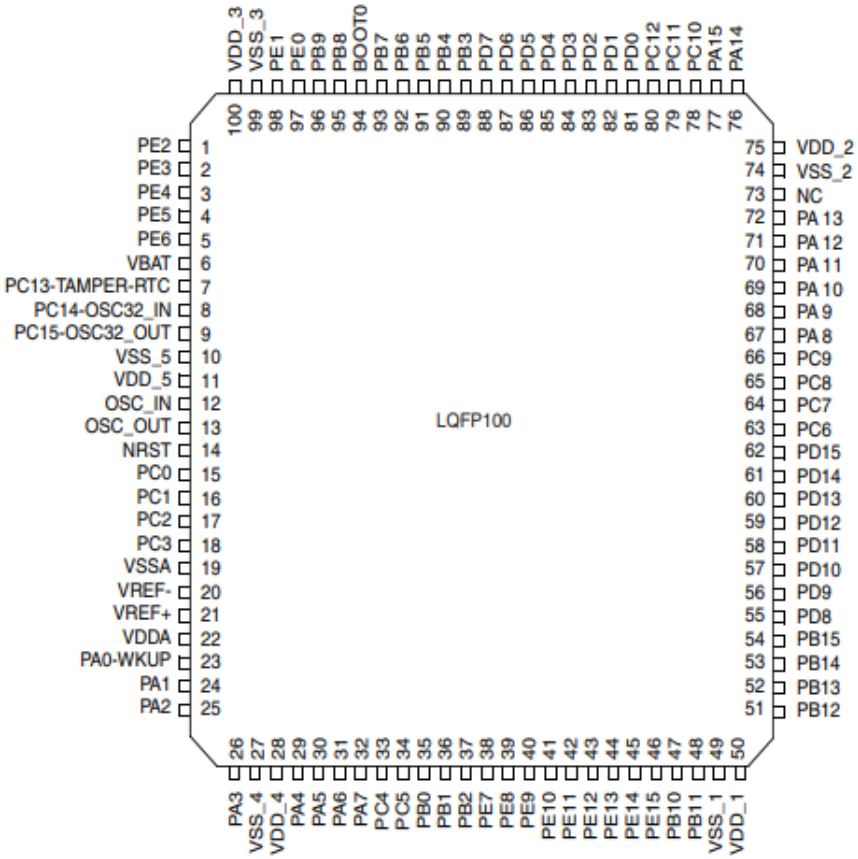


Figura 12. Distribución de pines del STM32F107

3.3 ST-Link V2

El ST-Link V2 (Figura 13) es un depurador y programador para trabajar con microcontroladores STM8 de 8 bits y STM32 de 32 bits. Las interfaces del módulo de interfaz de cable único (SWIM) y JTAG / serial wire debugging (SWD) se utilizan para comunicarse con cualquier STM8 o STM32, ubicado en una placa de aplicación.

Figura 13. Aspecto del dispositivo ST-Link V2

Las aplicaciones STM32 utilizan la interfaz USB de velocidad completa para comunicarse con los entornos de desarrollo integrados Atollic, IAR, Keil o TASKING, siendo el caso de Keil el empleado en el presente proyecto. Se puede ver a continuación las características del dispositivo:

- Alimentación de 5 V suministrada por un conector USB.
- Interfaz compatible con USB 2.0 de velocidad completa.
- Cable USB estándar A a Mini B.
- Características específicas de SWIM:
  - Voltaje de aplicación de 1.65 V a 5.5 V compatible con la interfaz SWIM.
  - SWIM capaz de soportar modos de baja velocidad y alta velocidad.
  - Velocidad de programación de SWIM: 9.7 Kbytes / s en baja velocidad y 12.8 Kbytes / s en alta velocidad.
  - Cable SWIM para la conexión a la aplicación a través de un estándar conector vertical ERNI (ref: 284697 o 214017) o un conector horizontal (ref: 214012).
  - Cable SWIM para la conexión a la aplicación a través de un pin header o un conector de 2.54 mm.
- Características JTAG / serial wire debugging (SWD):

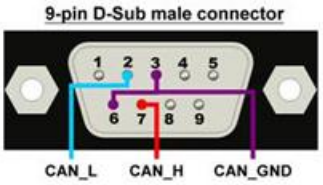
  - Voltaje de aplicación de 1.65 V a 3.6 V compatible con la interfaz JTAG / SWD.
  - Cable JTAG para la conexión a un estándar JTAG conector de paso de 20 pines de 2.54 mm.
  - Compatible con JTAG.
  - Comunicación SWD y visor de cables en serie (SWV) soportada.
- Función de actualización directa de firmware compatible(DFU).
- LED de estado que parpadea durante la comunicación con el PC.
- Temperatura de funcionamiento de 0 a 50 ° C.
- Alto voltaje de aislamiento de 1000 Vrms (ST-LINK / V2-ISOL solamente).

3.4 Dispositivos para comunicación CAN

Para realizar la comunicación vía CAN entre la aplicación de escritorio y la placa MCBSTM32C se hace uso de dos dispositivos CAN. Por un lado se cuenta con un dispositivo USB-CAN y por otro lado se cuenta con un dispositivo Wifi-CAN. La aplicación de escritorio está preparada para usar tanto un dispositivo como el otro, sin embargo, es importante remarcar que no se podrán usar ambos al mismo tiempo para proceder al envío o recepción de los mensajes CAN.

A continuación, se verá cada uno de estos dispositivos, junto a sus características principales.

ICPDAS i-7565: el i-7565 es un convertidor inteligente de USB a CAN de alto rendimiento capaz de proporcionar un bus CAN de gran rapidez para el envío y recepción de mensajes CAN. Se puede visualizar el dispositivo CAN junto a su distribución de pines en la Figura 14.



Terminal	2-wire CAN
1	Not Connect
2	CAN Low
3	CAN Ground
4	Not Connect
5	
6	CAN Ground
7	CAN High
8	Not Connect
9	

Figura 14. ICP-DAS i-7565. Distribución de pines

Este dispositivo cuenta con las siguientes características: diseño RoHS, compatibilidad con USB 1.1 / 2.0 (velocidad máxima), compatible con el estándar ISO 11898-2, admite tanto CAN 2.0A como CAN 2.0B, desarrollado por bus USB, velocidad de transmisión del bus CAN programable de 5Kbps a 1 Mbps o tasa de baudios seleccionada por el usuario, soporta la introducción de un filtro para la recepción de mensajes por el bus CAN, marca de tiempo del mensaje CAN con una precisión de ± 1 ms, actualización del Firmware a través del USB, biblioteca API para el desarrollo de programas de usuario, puente incorporado para introducir la resistencia terminadora de 120 ohmios, Indicador PWR / RUN / ERR para CAN y USB y Watchdog interno.

IFM CR3130: el CR3130 es un convertidor WiFi a CAN, este dispositivo permite el acceso inalámbrico a la interfaz CAN a la que se conecta para conseguir un intercambio de datos entre los dos dispositivos.

Para ello el CR3130 cuenta con una antena interna que genera una red WiFi a la cual se podrá acceder para establecer la conexión con el dispositivo y poder empezar a realizar comunicaciones CAN con otros dispositivos.

Para el presente proyecto tan solo es necesario conectar los pines de la interfaz CAN, los cuales pueden observarse en la siguiente figura.

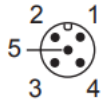
M12 connector (5 poles)		Pin	Potential
	Operating voltage	1	GND
		2	6...32 V DC
		3	Not connected
	CAN interface	4	CAN_H
		5	CAN_L

Figura 15. Distribución de pines IFM CR3130

En cuanto a los datos técnicos acerca del dispositivo, estos se pueden visualizar en la próxima imagen.

CANwireless		
Housing		Plastic
Connection		M12 CAN/power M12 service port
Protection rating Internal antenna External antenna		 IP 67 IP 65
Operating voltage	[V DC]	8...32
Current consumption sleep mode	[mA]	60 mA (24 V DC) < 1 mA (24V DC)
Temperature range Operation	[°C]	-40...75
CAN interface		CAN 2.0 A/B, ISO 11898-2 CANopen, Layer 2, J1939
wifi frequency		2.4 GHz / 5 GHz
wifi protocol		IEEE 802.11 a/b/g/n, IEEE 802.11 d/e/i/h
wifi / Bluetooth - range		75 m (internal antenna), 200 m (external antenna)
Bluetooth		Classic Bluetooth, Version: 2.1 + EDR, Serial Port Profile (SPP)
Indication		1 x status LED (2 colours)
Standards and tests		For further information, please refer to the data sheet: **www.ifm.com**

Figura 16. Datos técnicos del dispositivo CR3130

3.5 Keil uVision 4

El entorno de programación uVision 4 de Keil es una herramienta de carácter profesional mediante la cual se puede proceder a realizar programación de distintos microcontroladores. Esta herramienta se ha convertido en un estándar para el desarrollo de aplicaciones basadas en la familia de microcontroladores: MCS51.

El entorno uVision 4 contiene dos modos básicos de trabajo:

- Modo proyecto.
- Modo depuración.

En el modo proyecto, la interfaz del entorno permite una serie de funciones: gestión del proyecto de programación, definición de sus características, selección del tipo de procesador que se empleará, subsistemas en los que se divide el proyecto, gestión de los diferentes módulos de código fuente a la hora de realizar la aplicación final, tipo de archivos, ficheros generados al ensamblar (compilar), etc.

Una vez se finaliza la creación de un proyecto, bajo previa compilación sin errores resultantes, resulta necesario depurar el código desarrollado. Para ello, en el entorno de Keil uVision 4 se cuenta con el modo de depuración (para entrar en este modo se debe volcar en alguna placa de desarrollo el código que se ha realizado para la aplicación deseada), mediante el que se puede llevar a cabo una ejecución en tiempo real de la aplicación propuesta. Con este modo de funcionamiento se podrá comprobar que el algoritmo que se ha desarrollado cumple con las expectativas fijadas y se comporta de acuerdo con el deseo del desarrollador.

El entorno de uVision 4 está compuesto por una serie de componentes mediante los cuales se pueden llevar a cabo los procesos del desarrollo y sus posteriores depuraciones. Los componentes a los que se hace referencia son los siguientes:

IDE de uVision: Entorno de Desarrollo Integrado, incluye, entre otras, las siguientes características:

- Gestión de proyectos.
- Editor de código y dotado con corrección de errores de compilación.
- Diversidad de ajustes y opciones aplicables al entorno.
- Posibilidad de recurrir a “Ayudas”.

C51 (compilador) y A51 (Ensamblador): los archivos o ficheros fuente que se han desarrollado se envían al compilador C51 o al ensamblador A51. Estos

componentes los procesan y crean los ficheros objeto que serán procesados por el módulo montador. El compilador C51, cumple completamente con las especificaciones ANSI del lenguaje de programación C y contempla características adicionales para la arquitectura MCS51. Por otro lado, el ensamblador cuenta con el repertorio completo del 8051 y sus derivados.

Gestor de librerías LIB51: este componente será el encargado de crear librerías en base a los módulos objetos generados por el compilador y por el ensamblador. El gestor de librerías tendrá en su poder un conjunto de programas y subrutinas que serán invocados desde el proyecto de usuario y que pueden ser tomadas por el montador en el momento de generar la aplicación final.

Montador/ubicador BL51: el objetivo de este componente es crear un archivo o fichero tipo ELF/DWARF mediante los archivos creados por el compilador o ensamblador, además, si es el caso, de los extraídos de las librerías por ser referidos por el programador al crear el código de aplicación. La principal característica de estos ficheros objeto que se creen, será que no poseen código reubicable, sino que debe estar adecuadamente ubicado en posiciones concretas de memoria. Este archivo generado se utilizará para:

- Programar la ROM u otros tipos de memorias, usando el formato HEX.
- Depuración en placa.
- Verificación real del programa mediante un emulador.

Depurador: depurador simbólico al nivel del código fuente, es decir, el seguimiento de la ejecución y la interactuación se pueden realizar sobre el propio código fuente. Incluye un simulador veloz que es capaz de hacer simulaciones de un sistema completo basado en el 8051 o cualquier otro derivado de la misma familia, además de todos los periféricos existentes en la placa y el hardware externo. También permite probar la aplicación desarrollada, contando para ello con un depurador concreto, en el presente caso el STLINK V2.

El orden en el que se llevan a cabo los distintos procesos puede verse aplicado en la siguiente figura:

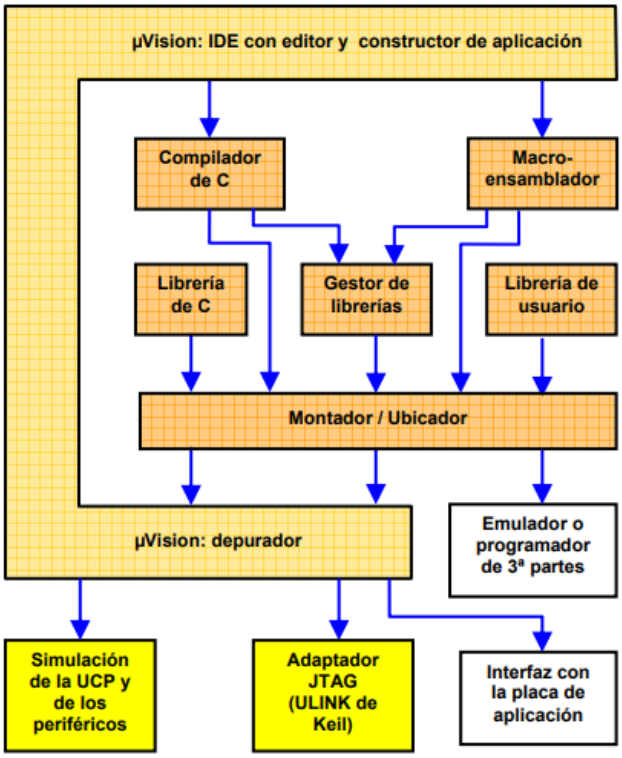


Figura 17. Componentes en Keil uVision 4

3.6 Visual Studio

Visual Studio es un entorno de desarrollo integrado, una herramienta que se emplea para sistemas operativos Windows y soporta una gran variedad de lenguajes de programación, como por ejemplo C#, C++, Java o Python entre otros.

En Visual Studio se permite a los desarrolladores crear sitios y aplicaciones web, así como cualquier otro entorno web que sea capaz de soportar la plataforma .NET. Además de servicios web también se podrán desarrollar otras funciones como aplicaciones de consola o aplicaciones de escritorio desde las que realizar una serie de acciones o eventos, siendo esta última función la que se empleará en el presente proyecto.

En el presente Trabajo Fin de Máster, se ha empleado las versiones de Visual Studio 2013 y 2017, la de 2013, fue la primera versión en añadir una versión "Community", la cual ofrece prácticamente las mismas características que la versión "Profesional", con la diferencia de que limita su uso a empresas de tamaño reducido, estudiantes y desarrolladores de software libre. Con esta versión de Visual Studio gratuita se podrá trabajar con los siguientes Frameworks:

- .NET Framework 2.0
- .NET Framework 3.0
- .NET Framework 3.5
- .NET Framework 4.0
- .NET Framework 4.5
- .NET Framework 4.5.1
- .NET Framework 4.5.2

Mientras que la versión de 2017 permite trabajar con los Frameworks:

- Mismos Frameworks que la versión 2013.
- .NET Framework 4.6
- .NET Framework 4.6.1
- .NET Framework 4.7

En resumen, mediante el IDE de Visual Studio, se cuenta con un entorno que dispone de un panel de inicio desde el cual se puede editar, depurar y compilar código y, después, publicar una aplicación; siendo en este proyecto necesaria la generación de una aplicación de escritorio capaz de permitir la visualización y modificación de parámetros, entre otras funciones mencionadas a lo largo de este proyecto.

4. ESTRUCTURA DEL TRABAJO FIN DE MÁSTER

En el presente apartado se pretende hacer entender al lector cómo acceder a cada uno de los archivos presentes en la carpeta del proyecto, archivos donde se encontrará el código de cada una de las programaciones realizadas.

Existe la posibilidad de añadir el código de estas aplicaciones como anexos, pero esto provocaría que el presente documento tuviera una longitud muy extensa. Para evitar este suceso se ha decidido incluir el presente apartado, explicando cómo acceder a cada uno de los ficheros de programación presentes en el proyecto.

Como ya se ha mencionado anteriormente, el proyecto se puede dividir principalmente en tres partes: modelado de las baterías, programación del microcontrolador y programación de la aplicación de escritorio.

Para cada una de estas partes, se ha desarrollado una programación diferente y se ha empleado un entorno de programación diferente. Cada uno de los ficheros necesarios para estas programaciones se encuentra dentro de la carpeta del proyecto TrabajoFinMasterSamuelSanchezBaeza.zip. Dentro de esta, se contará con 3 subcarpetas diferentes, cada una de ellas para cada una de las partes. De esta forma, se contará con las siguientes subcarpetas:

- Aplicación_PC.
- Modelado_Baterias.
- Programacion_Micro.

A lo largo del presente documento se hará referencia al código, el cual se encuentra en las subcarpetas anteriores. Por tanto, cada vez que se haga referencia a la carpeta TrabajoFinMasterSamuelSanchezBaeza.zip durante el presente documento, se estará haciendo referencia a los archivos que se encuentran dentro de la carpeta .zip, estando dentro de esta cada una de las programaciones que se han realizado en el presente Trabajo Fin de Máster.

En resumen, cuando se pretenda visualizar el código del modelado de baterías, desarrollado en Matlab, se accederá a la subcarpeta Modelado_Baterias. Cuando se desee observar el código de la programación del microcontrolador, en Keil uVision 4, se deberá de hacer uso de los archivos de la subcarpeta Programacion_Micro. Por último, se accederá a la subcarpeta Aplicación_PC en el caso de que se pretenda visualizar la programación desarrollada para la realización de la aplicación de escritorio, la cual está desarrollada bajo el lenguaje de programación C#.

5. MODELADO DE BATERÍAS

5.1 Estado del arte

En el presente apartado se van a exponer los diferentes modelos con los que se pretende simular el comportamiento de las baterías, haciendo hincapié en aquel que se ha empleado para la realización del modelado de las baterías, explicando además cómo se ha acabado realizando dicho modelado.

Cuando se pretende realizar un modelado de baterías existen diferentes formas de realizarlo, principalmente se pueden destacar los siguientes modelos: reacciones químicas, cajas negras y circuitos eléctricos equivalentes. En el presente Trabajo Fin de Máster se ha optado por la opción de implementar el modelo basado en circuitos eléctricos equivalentes.

Dentro del modelo de circuitos eléctricos equivalentes existe una gran variedad de circuitos que se pueden implementar, revisando los diferentes circuitos, principalmente se pueden destacar los modelos en 3 tipos:

- Experimentales.
- Electroquímicos.
- Eléctricos.

Los modelos experimentales y electroquímicos no generan de forma correcta la dinámica de las celdas cuando se pretende estimar el SOC del pack de baterías. Sin embargo, los modelos eléctricos presentan una gran utilidad a la hora de representar las características eléctricas de las baterías.

Es importante seleccionar una buena forma para realizar el modelado de las baterías, ya que pese a parecer que las baterías se comportan de forma simple, cuando se encuentran entregando o recibiendo energía, sufren una serie de procesos electroquímicos en dependencia de la temperatura que hacen que su modelado adquiera cierta dificultad. Contando con que el modelo de circuito eléctrico equivalente es el que mejor representa el SOC de un pack de baterías en un modelado dinámico, se escogerá dicho modelo para el presente proyecto. Este modelo puede definirse a grandes rasgos como una función no lineal de diversos parámetros de valor variable.

5.2 Variables de las baterías

Antes de indicar las ecuaciones empleadas para el cálculo de las variables de las baterías, se hará una breve explicación de cada una de ellas. De esta forma

será más fácil entender más adelante el uso de estas ecuaciones. Se van a destacar las siguientes variables:

- Tensión en circuito abierto (Voc).
- SOH.
- SOC.

Tensión de circuito abierto

El nivel de tensión Voc marca la diferencia de potencial existente entre los bornes de una batería al aire, es decir, sin ninguna carga conectada y la batería encontrándose en estado estable.

Cuando se conecta o desconecta una carga a una batería, es decir, cuando la batería es encuentra en carga o descarga, la tensión en los bornes de esta sufre una serie de transitorios hasta que se llega a una tensión denominada como tensión de relajación. Es importante medir la Voc después de estos transitorios ya que sino estos afectarían sobre la medida realizada.

Aparte de los transitorios mencionados, el nivel de tensión Voc también depende del nivel de SOC del que dispone la batería, como se puede observar en la Figura 18, el nivel de tensión Voc es prácticamente constante durante todo el estado de carga. Sin embargo, tiene unos picos abruptos en los momentos en los que las baterías se encuentran en su momento máximo y mínimo de carga. Es decir, observando la línea que siguen las baterías de Ion-Litio, se observa que las baterías mantienen una tensión casi constante (en torno a 4 V) en los rangos comprendidos entre el 20% y el 80% del estado de carga, al estar fuera de este rango el nivel de Voc es más inestable, variando su valor de tensión de forma más rápida.

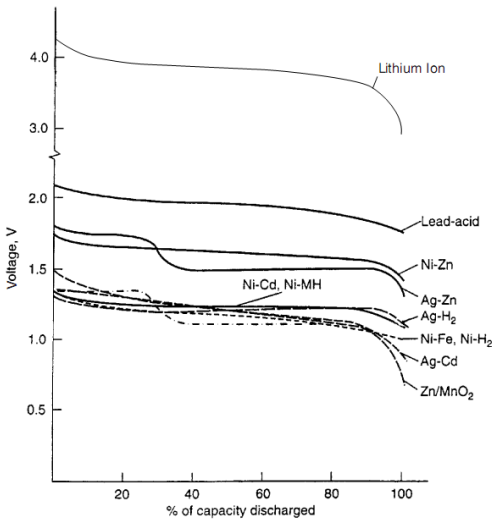


Figura 18. Relación entre Voc y SOC

Estado de salud

El estado de salud de una batería es definido como el periodo de tiempo transcurrido hasta alcanzar la tensión de corte a partir de la cual la batería deja de ser operativa (siguiendo el valor en el diseño de la batería), siendo este valor expresado con carácter porcentual. Por tanto, conociendo las condiciones de trabajo de las baterías junto con su ciclo de funcionamiento se tendrá la posibilidad de determinar el SOH. Definiéndolo de forma sencilla, el SOH será el porcentaje de vida útil de la batería que se está utilizando.

Como es de esperar, a medida que la batería se degrada, es decir, presenta un SOH más bajo, la tensión de la batería desciende. Este fenómeno puede observarse en la Figura 19.

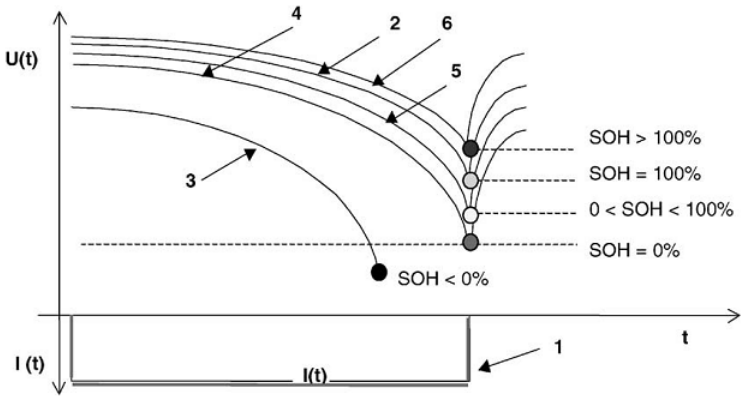


Figura 19. Imagen de ayuda en la definición de SOH

En la gráfica anterior se observa cómo con el paso del tiempo, el nivel de tensión ofrecido por las baterías es menor debido a la degradación de las mismas. El problema que esto supone, es que durante el transcurso del tiempo el nivel de tensión puede descender tanto que provoque que no alcancemos el nivel de voltaje necesario para el proyecto en el que se tenga instalada esta batería.

Estado de carga

El estado de carga (SOC) de una batería, mostrada en porcentaje, determina su nivel restante de carga respecto a la capacidad total que posee. Para determinar este porcentaje se deberá conocer la capacidad nominal de la batería, expresada en amperios hora (Ah), la carga total de la misma y la corriente que proporciona la batería, siendo positiva cuando se encuentra en descarga y negativa cuando se encuentra en carga.

5.3 Modelo de circuito equivalente y ecuaciones del modelo

Una vez conocidas las variables de las baterías, se puede pasar al método para obtenerlas, para ello se empleará un modelo de circuito equivalente como ya se ha mencionado anteriormente. Según se puede observar en [5, 7 y 14], se propone un modelo dinámico no lineal que se basa en el modelo de Randle para la obtención de la impedancia electroquímica, este ha sido el modelo empleado en el presente proyecto, obteniendo los diferentes parámetros mediante procesos experimentales. El modelo utilizado se muestra en la Figura 20.

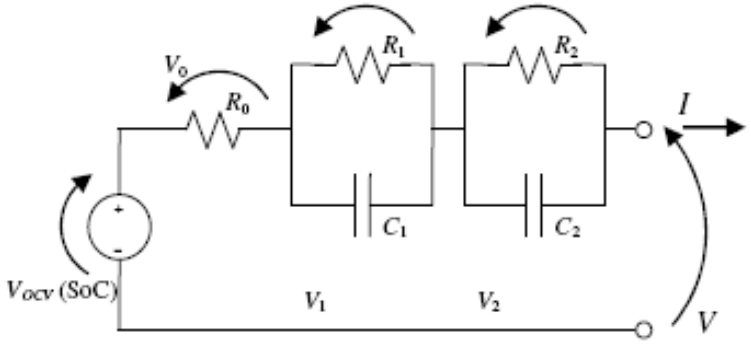


Figura 20. Circuito equivalente de una batería

Como se puede observar en la figura anterior, la tensión en circuito abierto Vocv de las baterías es modelada como una fuente de tensión ideal dependiente del SOC de la batería. Indagando un poco más en el diseño del circuito [18 y 14] se puede considerar en el diseño del circuito una resistencia R_0, que actuará como la resistencia interna de la batería, la capacidad de doble capa C_1, una resistencia de transferencia de carga R_1, una capacidad de difusión C_2 y finalmente una resistencia de difusión R_2. La primera RC (R_1 y C_1) modela la doble capa y la reacción cinemática, explicando de esta forma la subida exponencial entre los terminales de la batería durante los periodos de relajación, mientras que la segunda RC (R_2 y C_2) representa una deriva constante de tensión, siendo el retardo causado por estas del orden de horas.

Aplicando estos conceptos, se podrán aplicar las diferentes fórmulas para obtener las variables de las baterías. Se puede considerar que el voltaje existente entre los terminales de la batería se podrá determinar como la suma de la tensión en circuito abierto, sumado al voltaje óhmico, más el voltaje presente en la doble capa (V_1) y difusión (V_2), resultando la siguiente ecuación:

$$U = Vocv(SOC) + I \cdot R_0 + V_1 + V_2 \quad (1)$$

Donde:

$$\frac{dV_1}{dt} + \frac{1}{C_1 + R_1} = \frac{I}{C_1} \quad (2)$$

$$\frac{dV_2}{dt} + \frac{1}{C_2 + R_2} = \frac{I}{C_2} \tag{3}$$

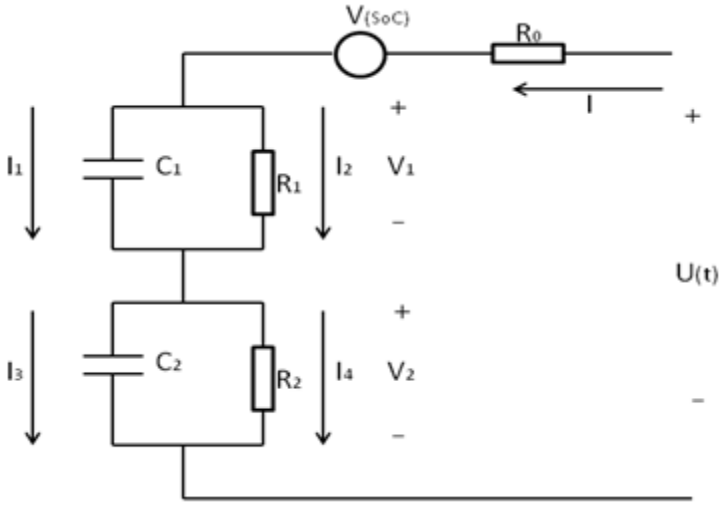


Figura 21. Caídas de tensión y corrientes en el modelo de circuito equivalente

Realizando un balance en las corrientes del circuito de la Figura 21 se puede obtener:

$$I = I_1 + I_2 = I_3 + I_4 \tag{4}$$

Mientras que las caídas de tensión V1 y V2:

$$V_1 = R_1 \cdot I_2\,(t) \tag{5}$$

$$V_2 = R_2 \cdot I_4\,(t) \tag{6}$$

Para el cálculo del SOC se utilizará el método de Coulomb Couting, este es el método más empleado para estimar el estado de carga de una batería, principalmente debido a su sencillez. Además, otra de las ventajas es que puede ser empleado para una primera estimación del estado de carga a pesar de no disponer de unas medidas de corriente totalmente fiables.

Para realizar el cálculo del SOC es necesario conocer el estado de carga inicial, la capacidad en Ah de la carga y las medidas de corriente. Una vez conocidos se podrá aplicar la siguiente ecuación:

$$SOC\,(t) = SOC(0) - \frac{1}{Ah_{nom}} \int_0^t I\,(t)dt \tag{7}$$

Según [6], la carga varía con el tiempo, de esta forma se cuenta con una corriente o potencia con dependencia del tiempo, en un periodo de duración determinado (t_0). Con estas características la tensión de la batería dispondrá de un voltaje mínimo (U_{min}). En casos en los que la carga se comporta de forma independiente a la batería, el voltaje mínimo se alcanza al final de t_0. Teniendo en cuenta la tensión mínima admitida bajo una carga determinada (U_1) y considerando como referencia la tensión mínima de una batería (U_{ref}), siendo esta considerada bajo unas condiciones de temperatura y nivel de carga, se puede determinar el estado de salud de una batería por medio de la siguiente ecuación.

$$SOH = \frac{U_{min} - U_1}{U_{ref} - U_1} \tag{8}$$

5.4 Obtención del modelado de baterías

En este apartado se explica cómo obtener el modelo matemático de las celdas de Litio con las que se ha trabajado. El objetivo será obtener un modelo que caracterice el comportamiento de las baterías en función de la temperatura y la corriente que se aplique sobre ellas. De este modo, se podrá obtener mediante la aplicación de las fórmulas del apartado anterior, el SOH y SOC de cada una de las celdas del pack de baterías.

Para la realización de este proyecto se ha contado con un pack de baterías formado por 16 celdas en serie. Las características de las celdas escogidas pueden apreciarse en la siguiente figura:

Large cell				
Model	Capacity	C-rate(Discharge)		Energy Density
		Continuous	Pulse	(Wh/kg)
SLPB100216216H	40	8	15	149
SLPB120216216HR2	46	12	15	138
SLPB120216216	53	5	8	169
SLPB130255255N	65	8	15	90
SLPB120255255	75	5	8	173
SLPB125255255H	75	8	15	156
SLPB132255255HR2	75	12	15	139
SLPB140460330	200	2	3	177
SLPB160460330	240	2	3	186

25Ah ~ 75Ah

Item	Unit	SLPB
Cell Capacity	Ah	2 ~ 240
Normal Voltage	VDC	3.7
Voltage Range	VDC	2.7 ~ 4.2

Figura 22. Características de la celda Kokam SLPB125255255H

Para la obtención de los distintos parámetros de las baterías se realizan una serie de ensayos con diferentes temperaturas y ratios de corriente, ya que como se puede observar en la Figura 23 y Figura 24, el comportamiento de las baterías es dependiente de la modificación tanto de las temperaturas como ratios de corriente.

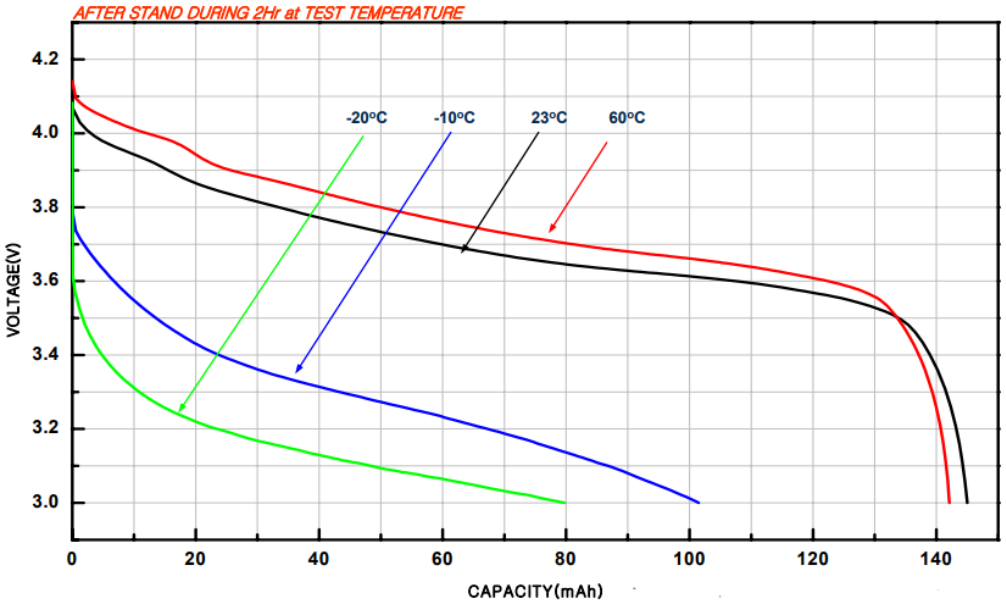


Figura 23. Efecto de la temperatura sobre las baterías

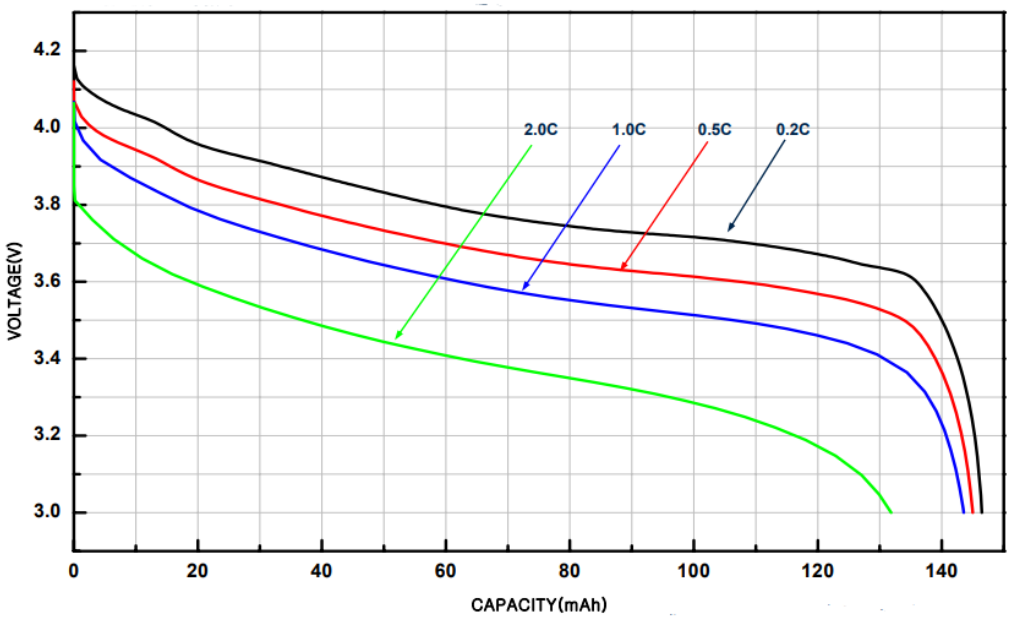


Figura 24. Efecto de la corriente sobre las baterías

Por tanto, se deberán hacer los ensayos a diferentes temperaturas y ratios de corriente, siempre dentro de los rangos de operación que marcan los fabricantes; obteniendo de esta forma una correcta modelación de la curva de tensión en circuito abierto en función del estado de carga, SOC y SOH.

Los ensayos a realizar cuentan con procesos tanto de carga como descarga, estos se realizan con diferentes ratios y temperaturas, siendo estos los que se pueden observar en la siguiente tabla:

Proceso	Ratios 1C = 75 Ah	Temperaturas (ºC)
Carga	0.2 - 0.4 – 0.6	0 – 20 - 40
descarga	0.2 - 0.4 – 0.6	0 – 20 - 40

Tabla 2. Ratios y temperaturas de los ensayos

Es importante mencionar que entre cada proceso de carga y descarga se realizarán pausas regulares con el objetivo de que las tensiones entre bornes de la batería se estabilicen en un valor. Los valores a monitorizar durante los ensayos serán tensión de celda (Voltios), corriente (Amperios) y temperatura de celda (ºC). Para llevar a cabo dichos ensayos se hace uso de una serie de materiales que servirán para automatizar el proceso y de esta forma realizarlo de forma segura y controlada, evitando problemas como sobretensiones, sobrecorrientes o temperaturas fuera de los rangos límite. Estos materiales que se han mencionado serán los que se pueden observar en la Figura 25.

Figura 25. Equipo necesario para la realización de los ensayos

El equipo está compuesto por:

- Carga programable.
- Fuente de alimentación programable.
- PC de monitorización y almacenamiento de datos.
- Rack de control.
- Cámaras climáticas.

Para la realización de los ensayos se colocan las baterías en el interior de cámaras climáticas, de esta forma seremos capaces de controlar la temperatura ambiente a la que se encuentran las baterías. Una vez conocidos el ratio de corriente y la temperatura aplicados sobre la celda se puede registrar los datos que se van obteniendo a lo largo de los ensayos, véase Figura 26. Estos ensayos se realizan con las diferentes características que hemos visto en la Tabla 2.

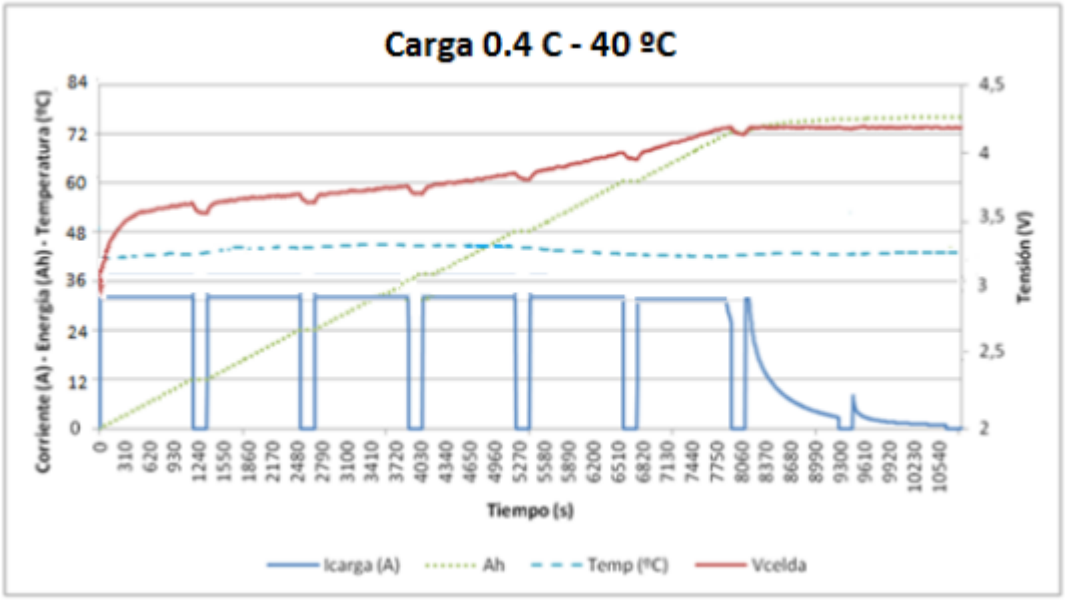


Figura 26. Resultados de los ensayos realizados a 0,4 C y 40 ºC

Como se puede observar en la Figura 26, durante los ensayos se producen microcortes en la inyección de la corriente, en estos microcortes la batería se encuentra en estado de relajación, provocando una pequeña caída de tensión en la tensión de circuito abierto de la celda. Este fenómeno es debido a que se deja de inyectar corriente sobre la batería, teniendo una caída de tensión a través de la resistencia interna de la batería. Por otro lado, se puede observar cómo la capacidad de la batería es directamente proporcional al aumento de la tensión de celda, teniendo un crecimiento lineal excepto en los extremos de la carga.

Una vez se ha alcanzado la tensión máxima de la celda, en este caso 4,2 V, se dejará de inyectar corriente lentamente para acabar con el proceso de carga.

Para la obtención de las tensiones de celda en circuito abierto para carga y descarga, se emplearán los valores de tensión que hay presentes en cada celda de baterías en los estados de relajación, se puede ver un ejemplo de la obtención de las curvas de carga y descarga en la Figura 27. En ella se puede visualizar cómo las líneas discontinuas (V_{oc} de carga y descarga) se obtienen con los microcortes de corriente que se producen durante los ensayos. Un ensayo con un mayor número de microcortes supondrá un ensayo con una mayor fiabilidad, ya que las curvas de carga y descarga tendrán más puntos de medida. También se puede observar cómo la tensión en circuito abierto será el punto intermedio entre las V_{oc} de carga y descarga, siendo este el punto que emplearemos en nuestro Firmware como referencia para el cálculo del SOC en circuito abierto.

Es importante mencionar que las caídas de tensión que se producen en las celdas cada vez son menores, debido a que a medida que se va progresando en

el proceso de carga, se inyecta una menor corriente sobre las baterías. En concordancia con esto, también se puede indicar que cuanto mayor es el ratio de corriente para carga o descarga, menor será la energía que se puede almacenar en las baterías, es decir, a medida que se aumenta el ratio de corriente obtenemos unas pérdidas mayores.

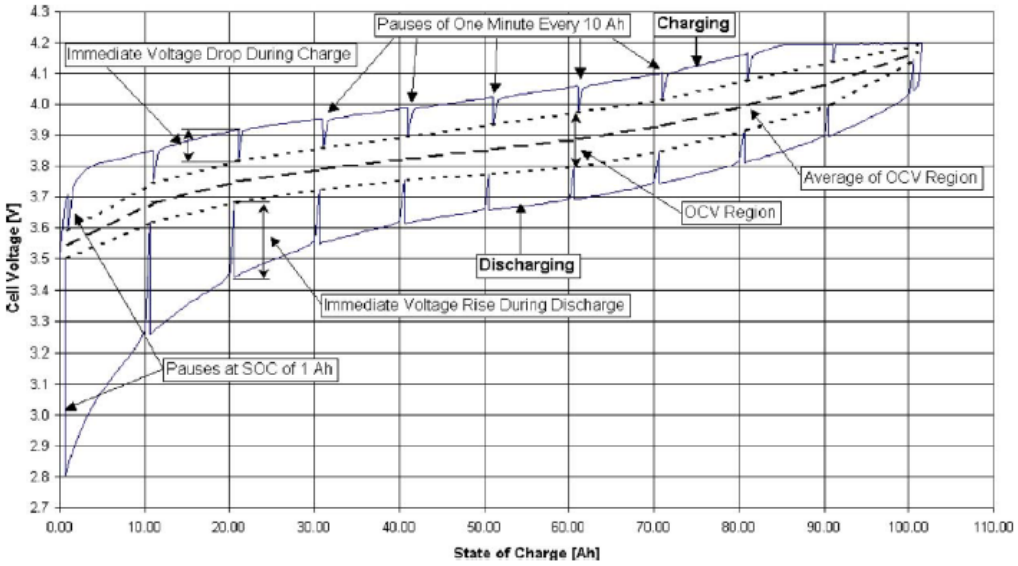


Figura 27. Tensión de celda durante carga y descarga

Si se quisiera hacer un cálculo aproximado de la tensión de celda en el estado de relajación de cada batería durante la carga, se podría aplicar la siguiente fórmula. Para el caso de que la celda se encontrará en descarga sería la misma fórmula con signo positivo.

$$V_{celda\ relax} = Vmedida - I \cdot R_0 \quad (9)$$

La tensión de celda en relajación será igual a la tensión medida en circuito abierto menos la corriente que circula por la resistencia interna de la batería. Con la resta del producto de la corriente y resistencia interna se compensará la diferencia existente entre la curva de carga y descarga que se empleará en el modelado y la curva que se obtiene en circuito abierto tanto para carga como descarga. A medida que avanzamos en el proceso de carga, la corriente inyectada sobre la batería será menor, obteniendo cada vez una menor diferencia en la caída de tensión entre la $V_{celda\ relax}$ y la V_{medida}. Este proceso queda reflejado en la Figura 27.

Empleando los valores de los parámetros del modelo obtenido, se han realizado simulaciones con el objetivo de validar los resultados. El resultado de todas las simulaciones realizadas es el siguiente, donde se representa la tensión

real que se ha medido durante los ensayos y la tensión que se obtendría según los cálculos con el modelo realizado.

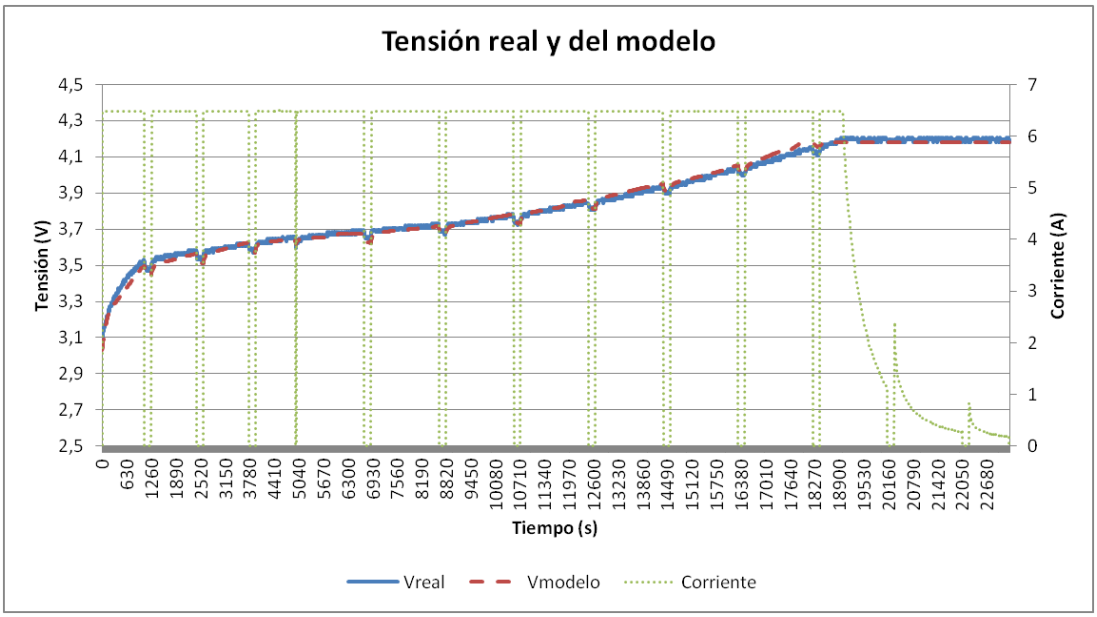


Figura 28. Respuesta del modelo a los datos reales de la batería

En la Figura 28, se comprueba que la respuesta del modelo realizado es buena, ya que se ajusta a los parámetros del comportamiento de una celda de batería. Sabiendo que el comportamiento de la celda es el adecuado, se puede garantizar que al aplicar los parámetros calculados obtendremos los valores de SOC y SOH adecuados, cada uno de ellos siendo dependientes de la tensión de celda en circuito abierto, corriente, capacidad y demás parámetros vistos a lo largo de este apartado.

Para la obtención de los parámetros del modelo, se emplea el circuito equivalente de la Figura 20 y se hace uso de la herramienta Matlab. El código utilizado para la obtención del modelado de las baterías en Matlab se puede encontrar dentro del archivo TrabajoFinMasterSamuelSanchezBaeza.zip, en el se han empleado los conceptos teóricos y prácticos mencionados en el presente documento. Veamos a continuación los resultados obtenidos para la caracterización de las celdas en la herramienta Matlab.

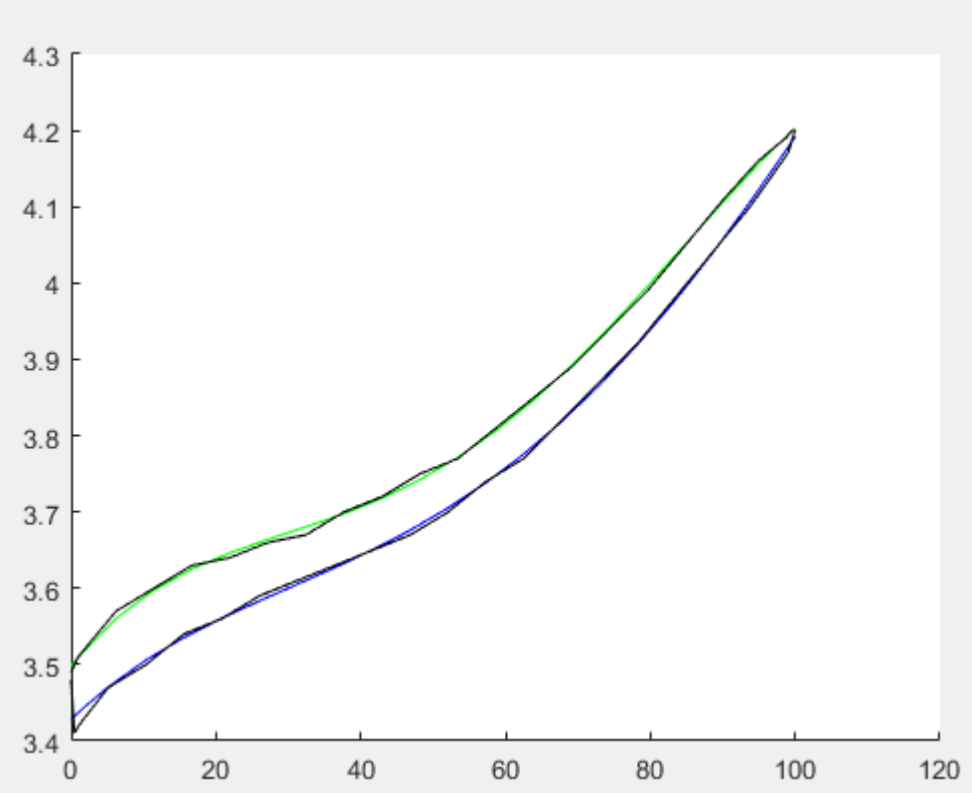


Figura 29. OCV-SOC para ensayo bajo 0.2C – 0ºC

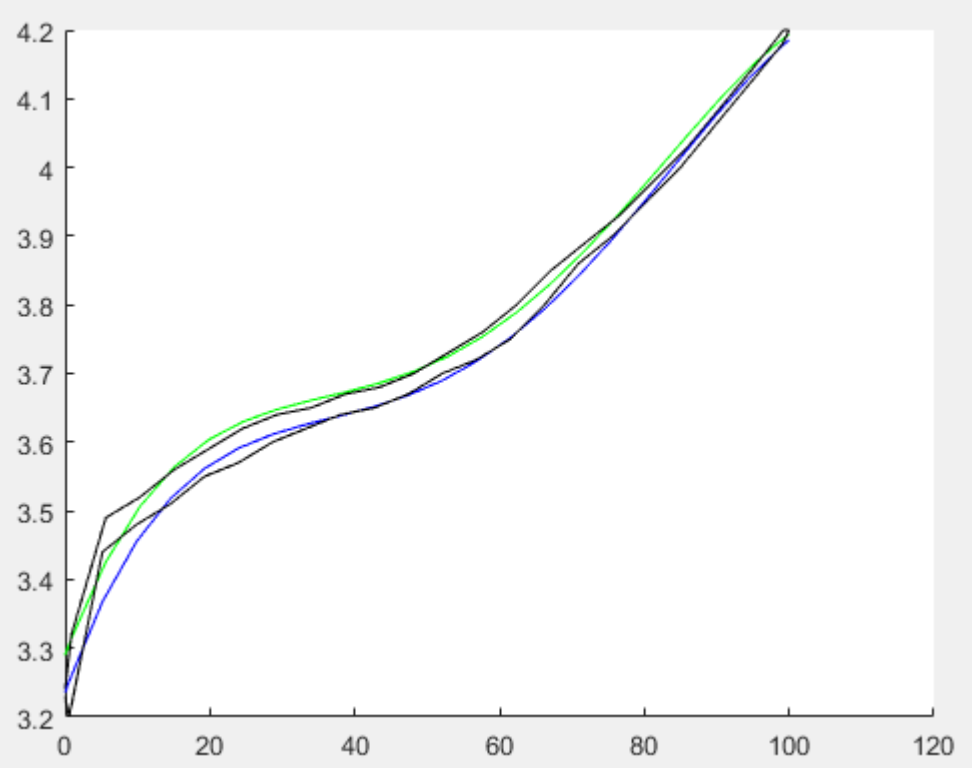


Figura 30. OCV-SOC para ensayo bajo 0.2C – 20ºC

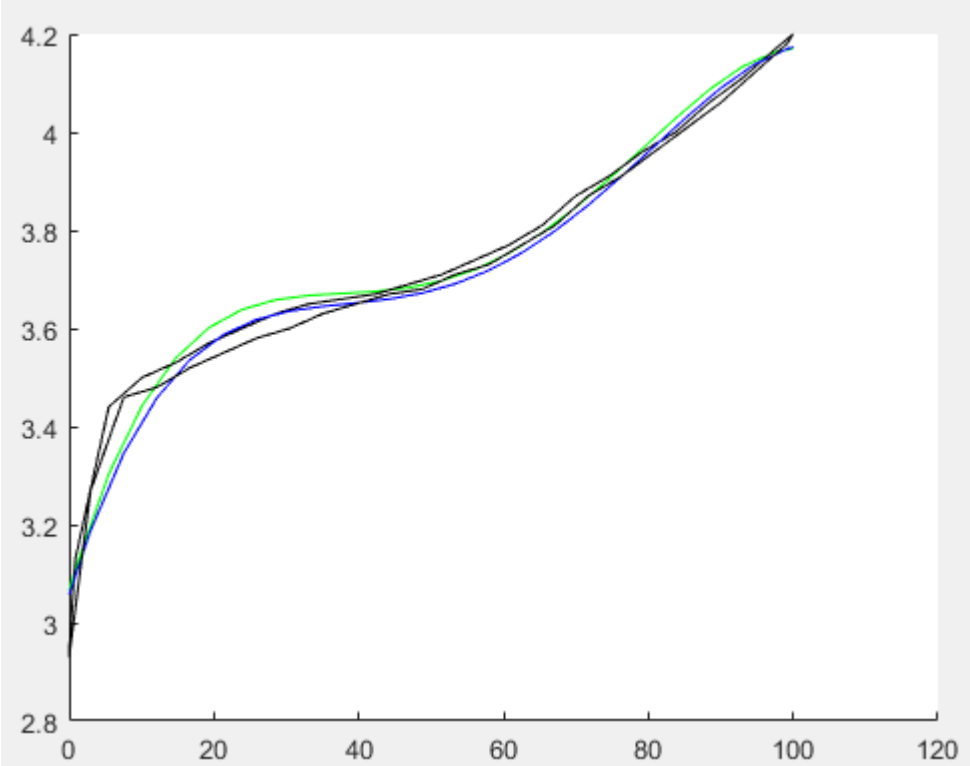


Figura 31. OCV-SOC para ensayo bajo 0.2C – 40ºC

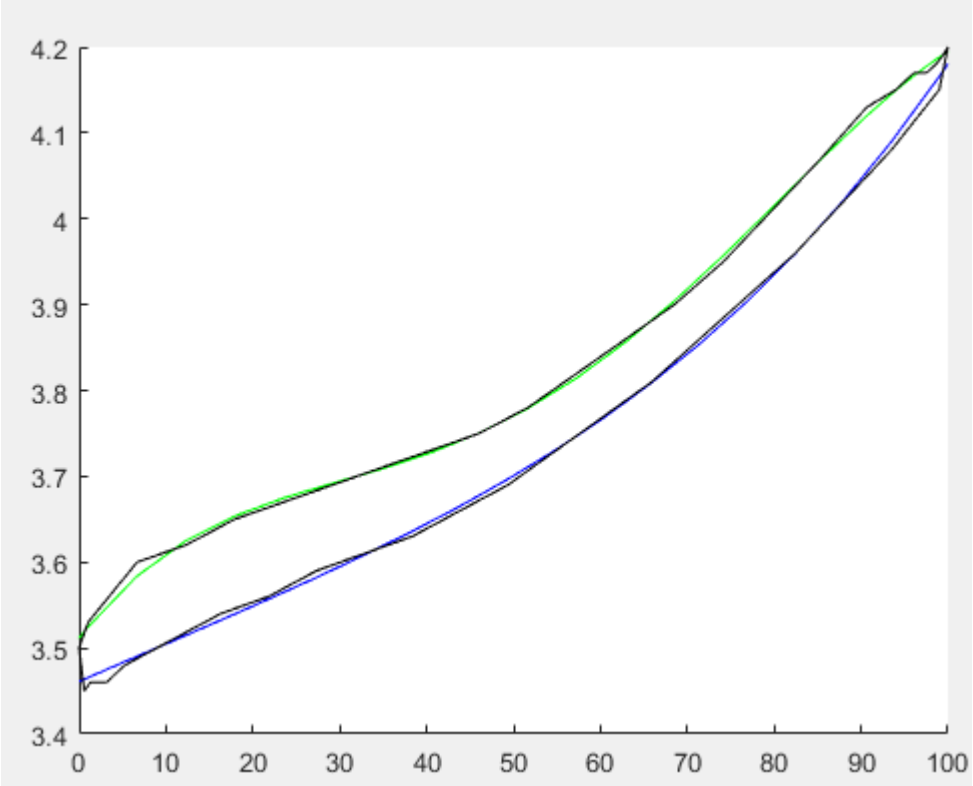


Figura 32. OCV-SOC para ensayo bajo 0.4C – 0ºC

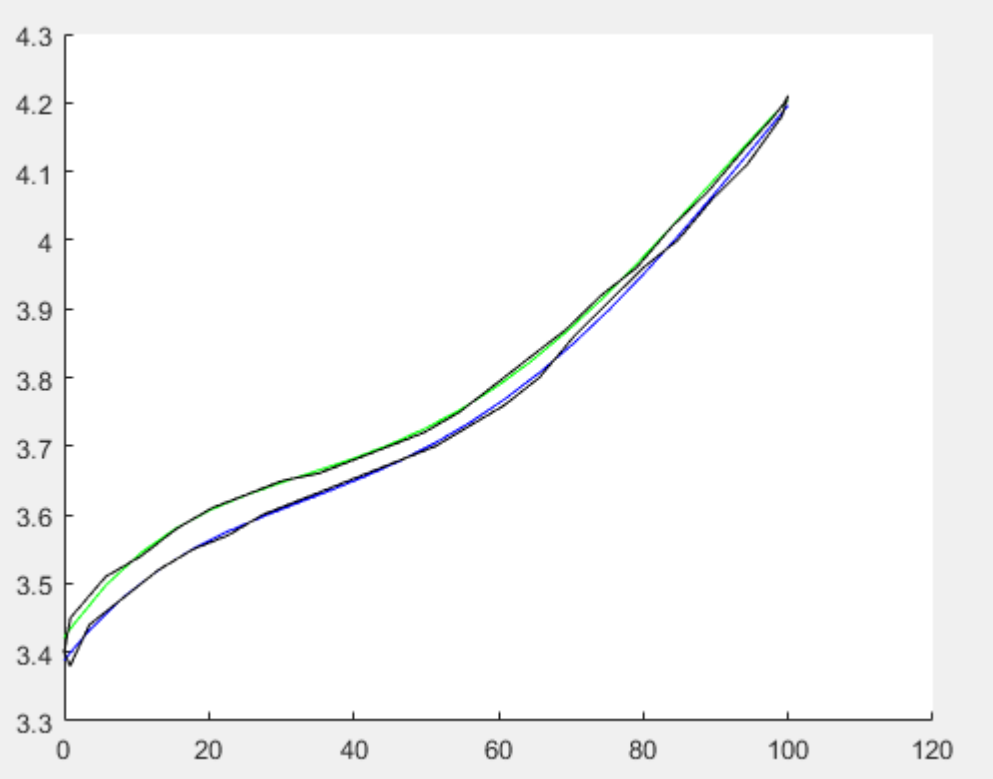


Figura 33. OCV-SOC para ensayo bajo 0.4C – 20ºC

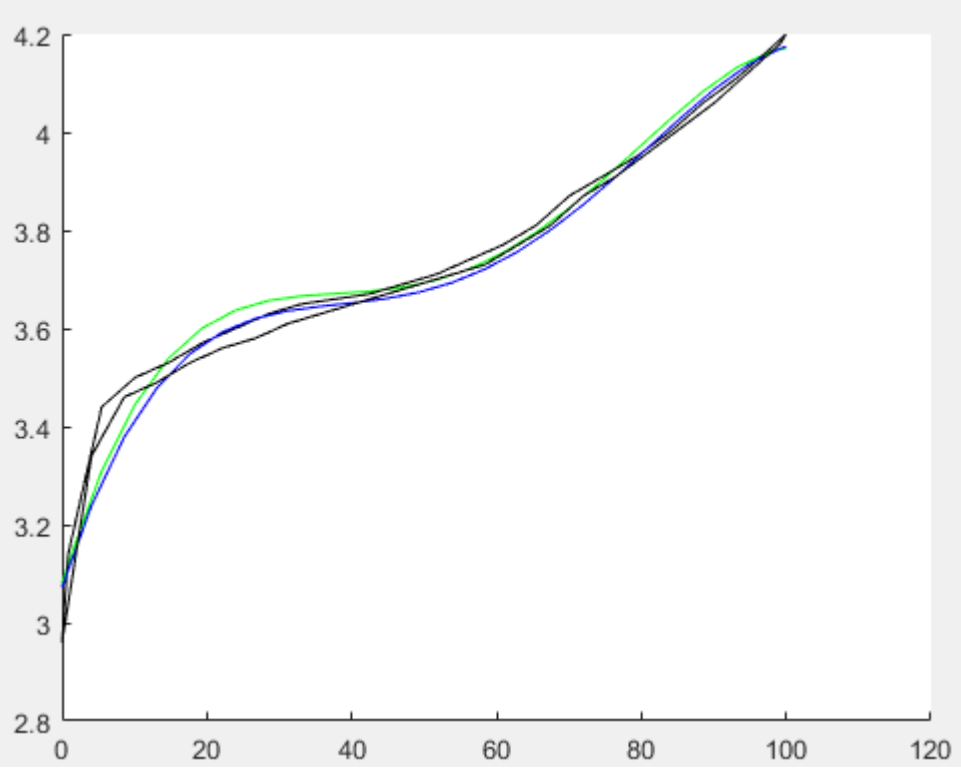


Figura 34. OCV-SOC para ensayo bajo 0.4C – 40ºC

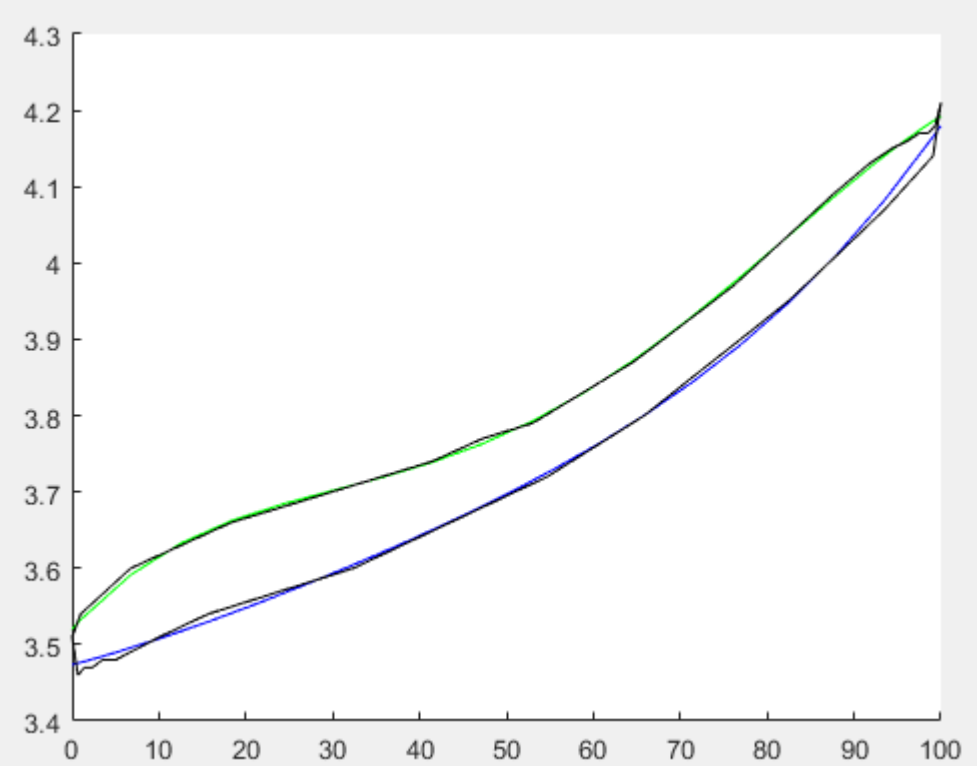


Figura 35. OCV-SOC para ensayo bajo 0.6C – 0ºC

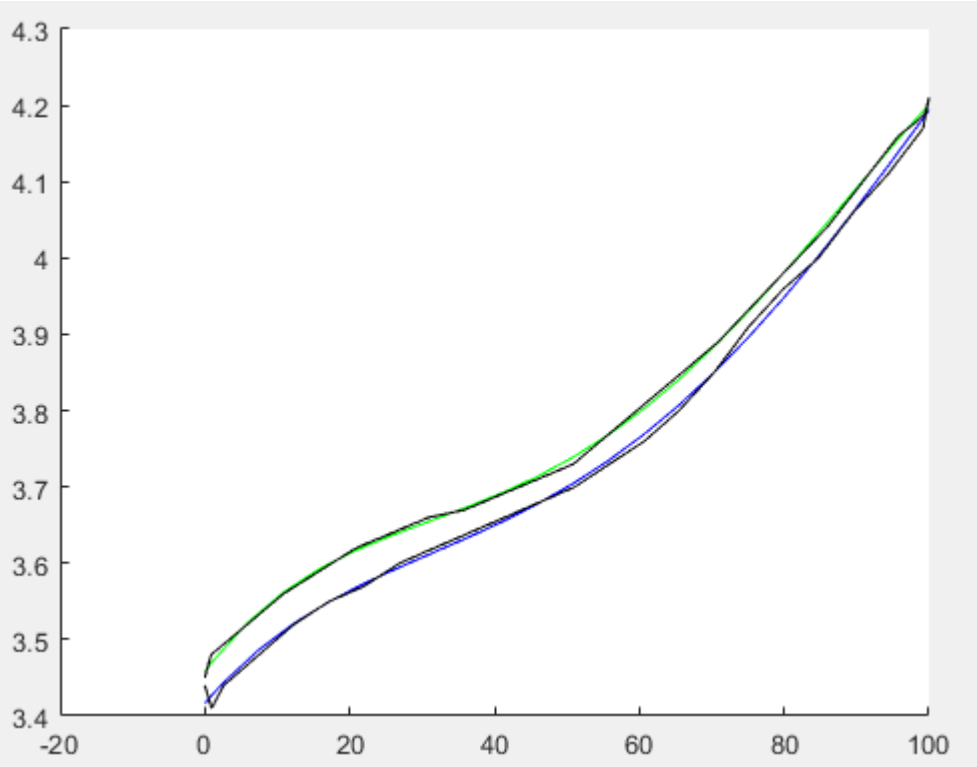


Figura 36. OCV-SOC para ensayo bajo 0.6C – 20ºC

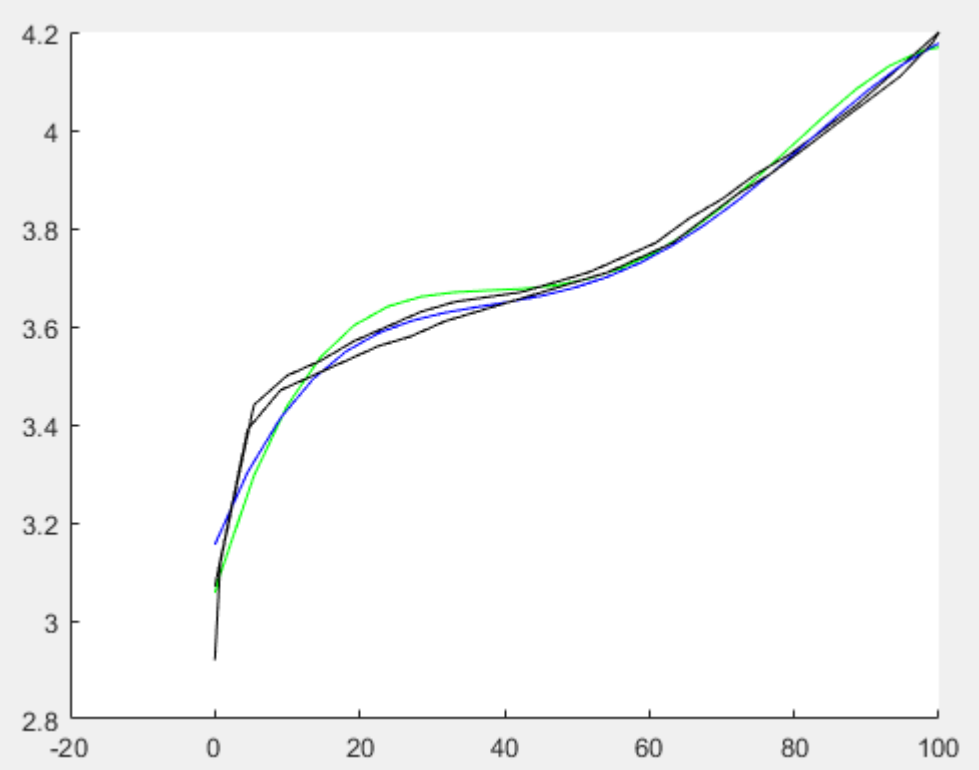


Figura 37. OCV-SOC para ensayo bajo 0.6C – 40ºC

Como se puede observar en las figuras resultantes, se ha logrado un modelado adecuado para las baterías de Litio ya que, observándose los resultados de las curvas tanto de carga como de descarga de las baterías empleadas, ambas tienen curvas parecidas a las del datasheet del fabricante (observar las curvas de tensión).

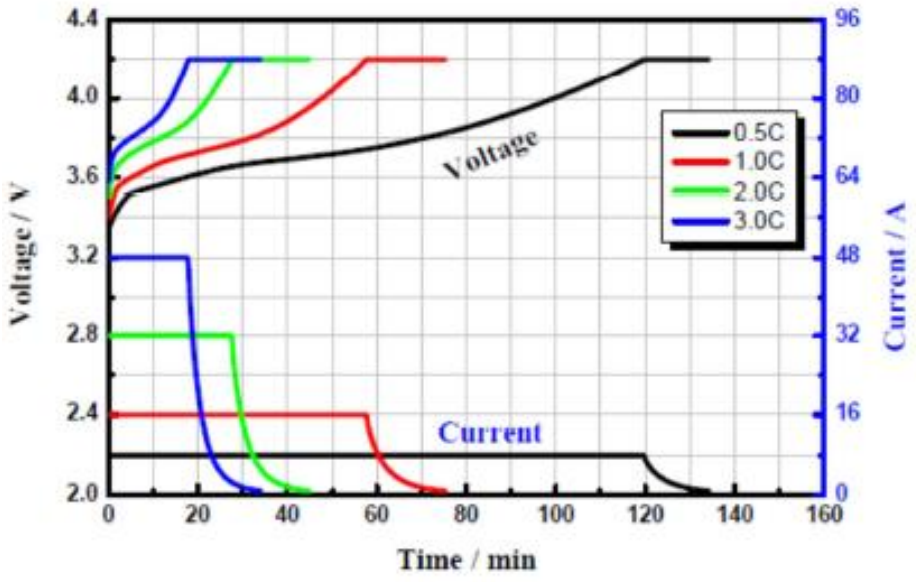


Figura 38. Curva característica de la celda

Como se ha podido ver, se ha obtenido 9 figuras, lo que supone 9 curvas distintas de OCV-SOC, esto se debe a que cada una de ellas se basa en una C y una temperatura diferente, obteniendo de este modo, una curva para cada uno de los ensayos realizados:

- 0’2 C – 0 ºC.
- 0’2 C – 20 ºC.
- 0’2 C – 40 ºC.
- 0’4 C – 0 ºC.
- 0’4 C – 20 ºC.
- 0’4 C – 40 ºC.
- 0’6 C – 0 ºC.
- 0’6 C – 20 ºC.
- 0’6 C – 40 ºC.

De las curvas resultantes de carga y descarga de las distintas figuras resultantes, se realizará la media entre ambas, obteniendo de esta forma, la relación SOC–OCV que será empleada en el Firmware para la obtención del cálculo del SOC. También existiría la posibilidad de incluir dos tablas en el Firmware y poner por separado las tablas de las curvas de carga y descarga. Sin embargo, se ha optado por realizar la media de ambas curvas y emplear una sola tabla en el Firmware tal y como se ha mencionado anteriormente.

Una vez se haya realizado la media de ambas curvas, se obtendrá una tabla de 100 puntos de esa curva, representando de esta forma la OCV en relación al SOC, representado del 0 al 100%. Esos 100 puntos de OCV serán almacenados en una tabla y copiados dentro del Firmware, véase Figura 39. Como se puede apreciar en la figura, no aparecen todos los puntos de la curva, ya que se debería disponer de 100 posiciones como se ha mencionado con anterioridad. Si se deseará visualizar la tabla al completo se podrá realizar visualizar el código al completo dentro de la carpeta TrabajoFinMasterSamuelSanchezBaeza.zip.

```
52
53  // --- Tabla para el cálculo del SOC en circuito abierto ---
54  const float Vocv[] =
55  {
56    3.31045578,
57    3.334789195,
58    3.357740199,
59    3.379364618,
60    3.399717326,
61    3.418852240,
62    3.436822322,
63    3.453679584,
64    3.469475079,
65    3.484258909,
66    3.498080220,
67    3.510987204,
68    3.523027100,
69    3.534246192,
70    3.544689809,
71    3.554402327,
72    3.563427166,
73    3.571806794,
74    3.579582722,
75    3.586795511,
76    3.593484763,
77    3.599689129,
78    3.605446304,
79    3.610793030,
80    3.615765093,
81    3.620397327,
82    3.624723611,
83    3.628776868,
84    3.632589070,
85    3.636191231,
86    3.639613414,
87    3.642884726,
88    3.646033321,
89    3.649086396,
90    3.652070197,
91    3.655010015,
92    3.657930185,
93    3.660854089,
```

Figura 39. Tabla para el cálculo del SOC en circuito abierto

Desde el Firmware, cada vez que se encuentren las baterías en vacío[5] se hará el cálculo de la tensión de cada celda en circuito abierto y una vez obtenido el valor será comparado con la tabla mencionada anteriormente. Se hará un recorrido de esa tabla y cuando se alcance el primer valor mayor al que se ha calculado, será el valor de SOC de dicha celda, ya que la tabla de OCV cuenta con 100 posiciones, cada una de ellas representando el porcentaje del SOC (del 0% al 100%). Cada posición incrementada dentro de la tabla supone un incremento del 1% en el SOC de la batería.

[5] Se entenderá como batería en vacio cuando esta no se encuentra ni en carga ni descarga, es decir, el flujo de corriente por la batería es nulo.

6. PROGRAMACIÓN DEL MICROCONTROLADOR EN C

En este apartado se van a explicar los pasos necesarios para completar la programación del microcontrolador, el cual actúa como puente entre las comunicaciones de la placa “BMS” de Texas Instruments y la aplicación de escritorio para PC desarrollada, realiza los cálculos necesarios para la obtención de parámetros como el SOC o SOH y detecta los errores que se pudieran estar produciendo en las baterías.

6.1 Diagramas de flujo

Como cada vez que se realiza un Firmware para una aplicación determinada, es aconsejable empezar con un correcto diagrama de flujo, donde se represente de forma gráfica el algoritmo que se va a emplear.

De esta forma se tendrá una perfecta puesta a punto, en la que se podrá determinar qué procesos deben realizarse en el algoritmo y el orden en el que estos se van a realizar.

Sin extenderse más en explicaciones se exponen los diagramas de flujo que se han realizado para la elaboración del presente Trabajo Fin de Máster:

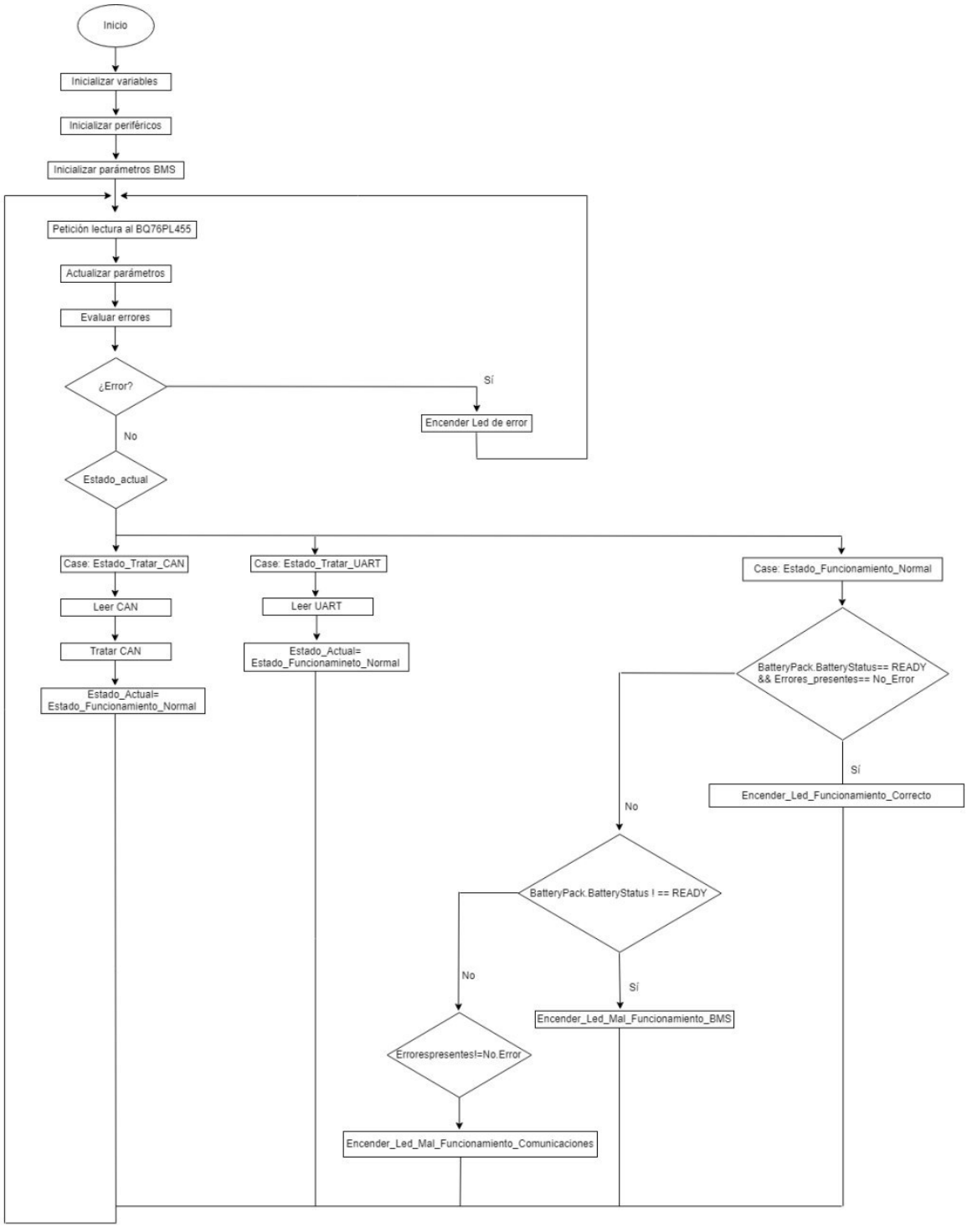


Figura 40. Diagrama de flujo del bucle principal (main.c)

En la figura anterior se puede ver el diagrama de flujo del bucle principal del algoritmo, el cual se repite continuamente en un bucle infinito.

Como se puede observar, una vez se inicia el algoritmo el primer paso consiste en inicializar las variables y periféricos presentes en el sistema, una vez inicializados, se inicia el bucle infinito, donde el principio de funcionamiento

seguido se basa en una máquina de estados, continuamente se hace una consulta por polling de la variable “Estado_Actual”, comprobando en cada momento en el estado en el que se encuentra esta.

Dependiendo de si se van a tratar los mensajes de las comunicaciones CAN (procedentes de la aplicación de PC) o UART (procedentes del bq76PL455), se llevarán a cabo unas funciones u otras, tal y como se ha observado en el diagrama de flujo. Además, la existencia de errores en el sistema también supondrá un desarrollo en el proceso del algoritmo diferente.

La actualización de la variable “Estado_Actual” se realizará cada vez que el microcontrolador recibe un mensaje de comunicaciones, saltando para ello la interrupción correspondiente. Este fenómeno podrá observare en la siguiente figura, donde se puede ver la gestión de cada una de las interrupciones habilitadas en el microcontrolador.

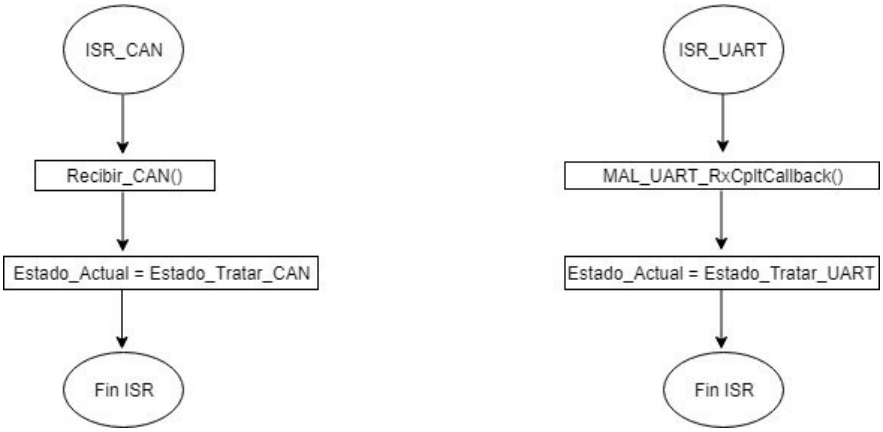


Figura 41. Diagrama de flujo de las interrupciones

En las comunicaciones, una vez salta la interrupción, el mensaje se almacena en un buffer de recepción para ser tratado más adelante. Para el caso de la comunicación CAN, el mensaje recibido se almacena en una cola FIFO, de la cual se irá leyendo los mensajes cuando el sistema se encuentre en el estado: “Estado_Tratar_CAN”. De igual forma ocurre en el caso de la UART, leyendo el mensaje del buffer de recepción cuando la variable “Estado_Actual” cambia su valor a “Estado_Tratar_UART”.

6.2 STM32CubeMX

Con el objetivo de evitar trabajo a la hora de realizar las configuraciones previas a la programación se hará uso de STM32CubeMX, mediante esta plataforma se consigue realizar las configuraciones de los pines del micro que vayan a ser utilizados posteriormente. Es decir, se evita tener que configurar manualmente todos los periféricos de nuestro proyecto.

De esta forma se puede conseguir ahorrar una cantidad importante de tiempo en la configuración de los pines. Sin embargo, es importante conocer el funcionamiento de esta aplicación. Como se puede observar en el código del Keil uVision 4, se marca en el fichero main.c los puntos donde el usuario podrá añadir código o no, con marcas como: "*USER CODE BEGIN*" o "*USER CODE END*"; es importante seguir estas consignas, ya que en el caso de que se desee añadir una modificación desde el STM32CubeMX y se genere de nuevo el código para aplicar las modificaciones realizadas, todo el código que no se encuentre dentro de los espacios reservados para la introducción de código por parte del usuario será borrado.

Para la realización del proyecto se necesita añadir y configurar los siguientes pines:

- ADC1: para la medida de la corriente que circula por las baterías.
- CAN1: comunicación entre el micro y el PC.
- UART2: comunicación entre el micro y placa Texas Instruments.
- GPIO: salidas digitales que controlan los leds de la placa. Posible aplicación: indicar al usuario con un juego de luces si el proceso se está realizando de forma correcta o no.
- SYS: indicar que para debug contamos con un JTAG de 5 pines.
- RCC: configuración del reloj de cristal.
- TIM1: timer para obtención de datos con un período determinado.

Una vez seleccionados cada uno de los pines que se van a utilizar, se deberá tener algo similar a lo que se puede observar en la Figura 42.

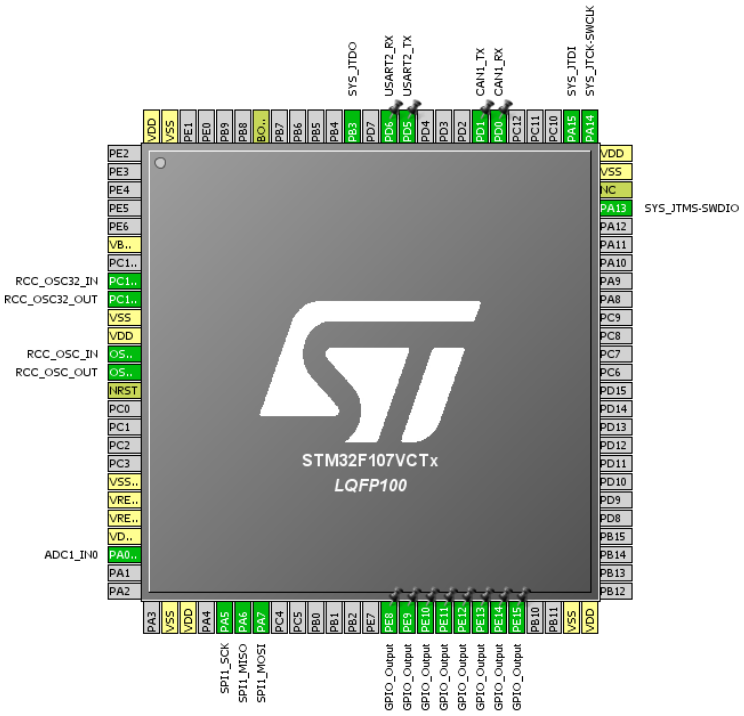


Figura 42. Pines utilizados del microprocesador

Cuando se tengan añadidos los pines que se van a utilizar se deberán de configurar de acuerdo a las características que se deseen para la realización del proyecto. En el presente Trabajo Fin de Máster, se configura cada uno de los pines como se puede observar en las siguientes figuras.

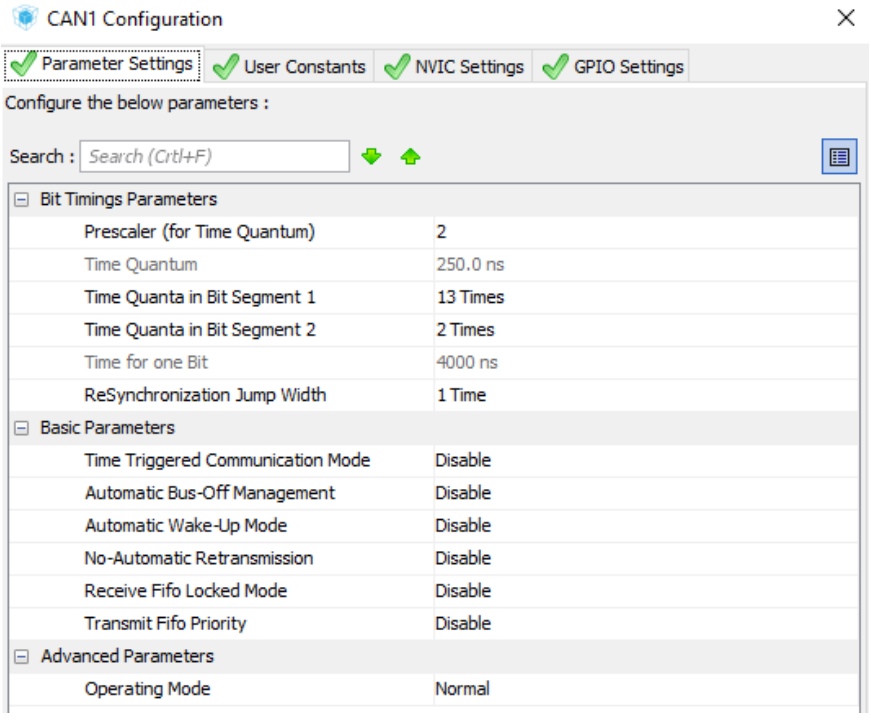
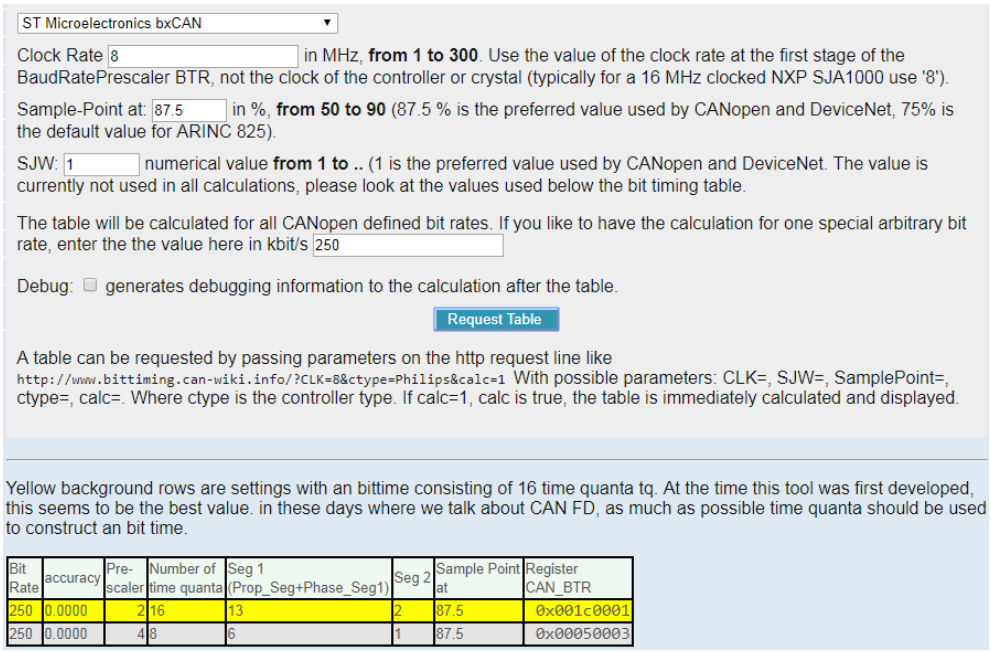


Figura 43. Configuración del CAN

La comunicación CAN deberá tener una velocidad de 250 kbit/s, ya que este es el baudrate que se ha establecido para la comunicación entre la aplicación de escritorio y el microcontrolador. Es importante que en ambos lados se cuente con el mismo baudrate ya que de lo contrario no se conseguirá establecer una comunicación correcta y por tanto, será imposible realizar la comunicación entre los dos dispositivos.

Para fijar el baudrate de 250 kbit/s, se debe añadir adecuadamente los parámetros que se solicitan. Para realizar los cálculos de cada uno de estos parámetros se ha empleado una herramienta online que se encarga de realizar el cálculo, introduciendo para ello una serie de parámetros que solicita. En la Figura 44, se puede observar cómo introduciendo los parámetros de la placa de evaluación y el baudrate deseado para la comunicación CAN, se obtienen los parámetros que se deberán añadir en la configuración del STM32CubeMX.


ST Microelectronics bxCAN

Clock Rate 8 in MHz, **from 1 to 300**. Use the value of the clock rate at the first stage of the BaudRatePrescaler BTR, not the clock of the controller or crystal (typically for a 16 MHz clocked NXP SJA1000 use '8').

Sample-Point at: 87.5 in %, **from 50 to 90** (87.5 % is the preferred value used by CANopen and DeviceNet, 75% is the default value for ARINC 825).

SJW: 1 numerical value **from 1 to ..** (1 is the preferred value used by CANopen and DeviceNet. The value is currently not used in all calculations, please look at the values used below the bit timing table.

The table will be calculated for all CANopen defined bit rates. If you like to have the calculation for one special arbitrary bit rate, enter the the value here in kbit/s 250

Debug: generates debugging information to the calculation after the table.

Request Table

A table can be requested by passing parameters on the http request line like `http://www.bittiming.can-wiki.info/?CLK=8&ctype=Philips&calc=1` With possible parameters: CLK=, SJW=, SamplePoint=, ctype=, calc=. Where ctype is the controller type. If calc=1, calc is true, the table is immediately calculated and displayed.

Yellow background rows are settings with an bittime consisting of 16 time quanta tq. At the time this tool was first developed, this seems to be the best value. in these days where we talk about CAN FD, as much as possible time quanta should be used to construct an bit time.

Bit Rate	accuracy	Pre-scaler	Number of time quanta	Seg 1 (Prop_Seg+Phase_Seg1)	Seg 2	Sample Point at	Register CAN_BTR
250	0.0000	2	16	13	2	87.5	0x001c0001
250	0.0000	4	8	6	1	87.5	0x00050003

Figura 44. Cálculos del bit timing de la comunicación CAN

La segunda comunicación que es necesario configurar, es por medio de UART, con ella se establecerá la comunicación entre la placa de Texas Instruments que actuará como BMS y la placa de evaluación.

Para establecer la configuración de la comunicación correctamente, se deberán introducir los siguientes parámetros básicos:

- Baudrate: 115200 bits/s.
- Word length: 8 bits (incluyendo paridad).
- Parity: No.
- Stop bits: 1.

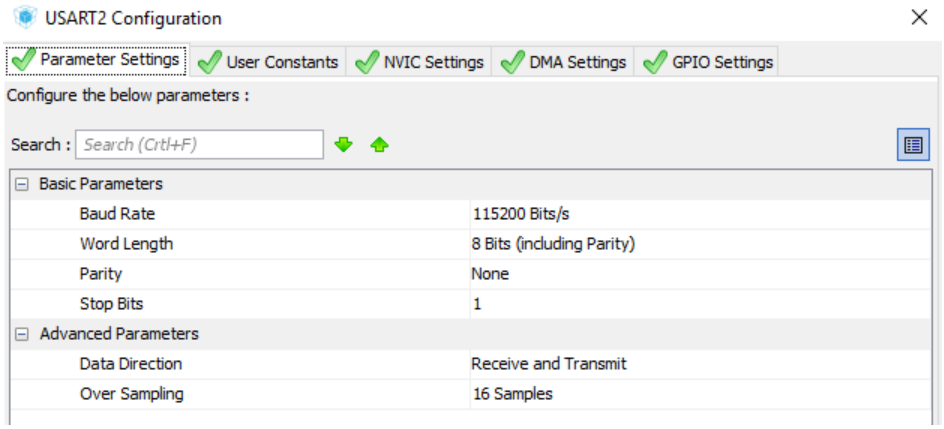


Figura 45. Configuración de la comunicación UART

Una vez configurada cada una de las comunicaciones, se procede a la configuración de las interrupciones. En este caso se han añadido las interrupciones correspondientes a la recepción de mensajes por CAN y por UART, de esta forma cada vez que se produzca la recepción de un mensaje por CAN o por UART saltará la interrupción y podremos gestionar la recepción de ese mensaje. Además también se añadieron las interrupciones del ADC, sin embargo, finalmente se ha optado por realizar la adquisición de los datos del ADC por medio de polling.

En la Figura 46, se puede visualizar cómo se han activado cada una de las interrupciones que han sido mencionadas en el párrafo anterior.

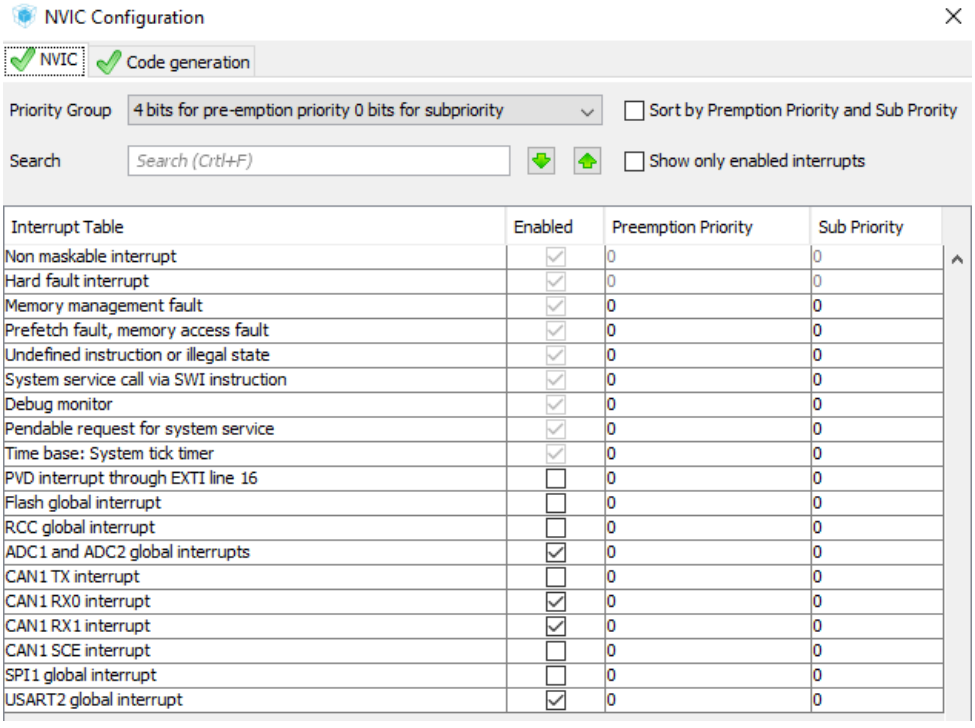


Figura 46. Configuración de las interrupciones

Una vez introducidas estas configuraciones se habrá finalizado con el proceso de configuración, por tanto, se podrá pasar a generar el código. Esta función se podrá realizar desde el propio software STM32CubeMX, pulsando sobre "*Generate source code based on our settings*" y seleccionando el entorno de programación en el que se desea generar dicho código.

6.3 Archivos generados en el Firmware

Cuando se haya generado el código fuente por parte del software STM32CubeMX, se podrá añadir el código en C que se crea oportuno para que la placa de desarrollo realice la aplicación deseada.

Para lograr el objetivo propuesto, en el presente Trabajo Fin de Máster se han desarrollado los siguientes archivos, los cuales se van a comentar para entender un poco mejor qué funciones contienen en su interior.

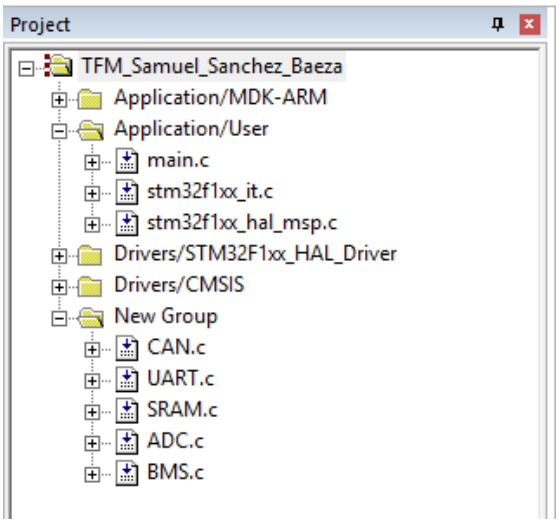


Figura 47. Archivos presentes en el proyecto Keil uVision4

Como se puede observar en la Figura 47, en el proyecto contamos con los siguientes ficheros o archivos:

Main: lugar en el que tendremos almacenado el código principal de la aplicación, desde este archivo se llamará a cada una de las funciones presentes en los demás archivos. El algoritmo que tiene introducido sigue los diagramas de flujo que se han visto en el apartado anterior.

Observando el código, se puede observar cómo se sigue una máquina de estados, donde se compara en todo momento cuál es el estado actual en el que se encuentra, siguiendo la variable "Estado_Actual". Dependiendo del estado en el que se encuentre en el proceso se harán unas llamadas a función determinadas u otras, cumpliendo así con el algoritmo necesario para el funcionamiento de la aplicación.

```
switch(Estado_Actual)
{
  // --- CAN ---
  case(Estado_Tratar_CAN):
    Leer_CAN (datos_leidos);
    Tratar_CAN (datos_leidos);
    Estado_Actual = Estado_Funcionamiento_Normal;
    break;
  // --- SPI ---
  case(Estado_Tratar_SPI):
    break;
  // --- UART ---
  case(Estado_Tratar_UART):
    Leer_UART();
    Estado_Actual = Estado_Funcionamiento_Normal;
    break;
  // --- LED's Funcionamiento ---
  case(Estado_Funcionamiento_Normal):
    // --- Existe un error de comunicaciones?? ---
    if(Errores_Presentes == NO_Error)
    {
      // --- No existe ningún tipo de error ---
      if (BatteryPack.BatteryStatus == READY)
      {
      // --- encendemos y apagamos el LED PE13 para indicar un funcionamiento correcto ---
        Encender_Led_Funcionamiento_Correcto();
      }
      // --- Existe un error en los mensajes obtenidos del BMS ---
      else
      {
        // --- encendemos y apagamos el LED PE9 para indicar un fallo en el BMS ---
        Encender_Led_MAL_Funcionamiento_BMS();
      }
    }
    else
    {
      // --- Parapadeo del LED PE8 para indicar que existe un error de funcionamiento ---
      Encender_Led_MAL_Funcionamiento_Comunicaciones();
    }
    //HAL_Delay(1500);
    break;
}
```

Figura 48. Máquina de estados del bucle principal

En la Figura 48 se puede observar la máquina de estados que se ha mencionado, esta lleva a cabo una consulta por polling, en la cual dependiendo de la variable “Estado_Actual” se llevan a cabo una serie de funciones.

En primer lugar se cuenta con una comunicación con el PC por medio de CAN, con esta comunicación se podrá tanto leer como escribir parámetros de las baterías.

En segundo lugar, se cuenta con una comunicación SPI, esta se ha añadido para futuras ampliaciones del proyecto, ya que existen otras placas de Texas Instruments capaces de realizar las mismas funciones que la seleccionada pero con otro tipo de baterías (Niquel-Cadmio, Plomo-Ácido, etc.), siendo la comunicación de algunas de ellas por medio de SPI. Puesto que está planteado para futuras aplicaciones no se ha desarrollado todavía las funciones para la realización de la comunicación SPI.

En tercer lugar, aparece una comunicación vía UART, esta será necesaria para establecer comunicación entre la placa MCBSTM32C y el bq76PL455. El funcionamiento es sencillo, se trata de un funcionamiento petición-respuesta, por tanto, cada vez que se desee actualizar los parámetros de las celdas se debe

realizar una petición de lectura (previa correcta configuración para saber qué parámetros se deben solicitar) y acto seguido se recibirá la respuesta con los datos correspondientes.

Tanto este, como los ficheros siguientes, se podrán encontrar dentro de la carpeta del proyecto: TrabajoFinMasterSamuelSanchezBaeza.zip.

stm32f1xx_it: fichero en el que se encuentra la gestión de las interrupciones, como ya se ha comentado anteriormente, se cuenta con dos interrupciones, una para la recepción de mensajes CAN y otra para la recepción de mensajes por UART. Sin embargo, la gestión de la interrupción de la UART se encuentra también dentro del fichero main.c, haciendo uso de la función "HAL_UART_RxCpltCallback".

Como se puede observar en el código de este fichero, véase dentro de TrabajoFinMasterSamuelSanchezBaeza.zip, cada vez que se debe gestionar alguna de las interrupciones se actualizará la variable "Estado_Actual" dependiendo de la interrupción que se haya generado, de esta forma se podrá llevar a cabo la consulta por polling que se ha planteado.

CAN: dentro de este fichero se contará con cada una de las funciones que se han desarrollado para la comunicación vía CAN. Se contará con las siguientes funciones, recepción de mensajes vía CAN y almacenado de estos mensajes en cola FIFO, desencolado de los mensajes de la cola FIFO y procesado de esos mensajes y, por último, mensajes de respuesta a los mensajes CAN que se han recibido y cuentan con una estructura acordada previamente.

En el proyecto se ha optado por la utilización del protocolo de comunicaciones estándar CANOpen, de tal forma que cada dispositivo debe implementar un perfil de este protocolo. Esta estructura de la que se habla, se basa en la normativa CiA 418 y CiA 419, en ella se indica cuál es la estructura que deben seguir los mensajes de comunicaciones CANOpen para módulos de baterías. Se detallarán a continuación cada uno de los mensajes que se han empleado para la realización del proyecto; donde se puede observar el índice y subíndice estipulado para cada mensaje CAN.

Índce	SubIndice	Nombre	Descripción	Tipo	Acceso	Valor por defecto
0x1A03	2	Dirección de la corrriente	Dirección de la corriente (carga o descarga)	Unsigned 32	ro	0
	3	SOH	Estado de Salud de la batería	Unsigned 32	ro	0

Índce	SubIndice	Nombre	Descripción	Tipo	Acceso	Valor por defecto
	4	Operating current	Corriente que circula por las baterías	Unsigned 32	ro	0

Índce	SubIndice	Nombre	Descripción	Tipo	Acceso	Valor por defecto
0x2000	1	1ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	2	2ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	3	3ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	4	4ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	5	5ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	6	6ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	7	7ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	8	8ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	9	9ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	10	10ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	11	11ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	12	12ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0

Índce	SubIndice	Nombre	Descripción	Tipo	Acceso	Valor por defecto
	13	13ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	14	14ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	15	15ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
	16	16ª Temperatura medida en las celdas	Temperatura de celda	Unsigned 32	ro	0
0x2001	1	1ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	2	2ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	3	3ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	4	4ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	5	5ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	6	6ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	7	7ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	8	8ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	9	9ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	10	10ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	11	11ª celda Tensión	Tensión de celda	Unsigned 32	ro	0

Índce	SubIndice	Nombre	Descripción	Tipo	Acceso	Valor por defecto
	12	12ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	13	13ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	14	14ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	15	15ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
	16	16ª celda Tensión	Tensión de celda	Unsigned 32	ro	0
0x2002	1	Dirección de la corrriente	0: sin corriente; 1:batería en descarga; 2: batería en carga	Unsigned 8	ro	0
	2	Operating current (calculada por shunt)	Corriente que circula por la batería	Unsigned 32	ro	0
0x2003	1	1ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	2	2ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	3	3ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	4	4ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	5	5ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	6	6ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	7	7ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0

Índce	SubIndice	Nombre	Descripción	Tipo	Acceso	Valor por defecto
	8	8ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	9	9ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	10	10ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	11	11ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	12	12ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	13	13ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	14	14ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	15	15ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
	16	16ª celda SOH en porcentaje	SOH por celda	Unsigned 8	ro	0
0x2005	1	1ª celda SOC	SOC por celda	Unsigned 8	ro	0
	2	2ª celda SOC	SOC por celda	Unsigned 8	ro	0
	3	3ª celda SOC	SOC por celda	Unsigned 8	ro	0
	4	4ª celda SOC	SOC por celda	Unsigned 8	ro	0
	5	5ª celda SOC	SOC por celda	Unsigned 8	ro	0
	6	6ª celda SOC	SOC por celda	Unsigned 8	ro	0

Índce	SubIndice	Nombre	Descripción	Tipo	Acceso	Valor por defecto
	7	7ª celda SOC	SOC por celda	Unsigned 8	ro	0
	8	8ª celda SOC	SOC por celda	Unsigned 8	ro	0
	9	9ª celda SOC	SOC por celda	Unsigned 8	ro	0
	10	10ª celda SOC	SOC por celda	Unsigned 8	ro	0
	11	11ª celda SOC	SOC por celda	Unsigned 8	ro	0
	12	12ª celda SOC	SOC por celda	Unsigned 8	ro	0
	13	13ª celda SOC	SOC por celda	Unsigned 8	ro	0
	14	14ª celda SOC	SOC por celda	Unsigned 8	ro	0
	15	15ª celda SOC	SOC por celda	Unsigned 8	ro	0
	16	16ª celda SOC	SOC por celda	Unsigned 8	ro	0
0x2006	3	Tensión máxima de celda	Tensión máxima de celda	Unsigned 32	rw	4,2V
	4	Tensión de celda nominal	Tensión de celda nominal	Unsigned 32	rw	3,7V
	5	Tensión mínima de celda	Tensión mínima de celda	Unsigned 32	rw	3V
	7	Temperatura operación máxima	Temperatura operación máxima	Unsigned 32	rw	60ºC
	8	Temperatura operación mínima	Temperatura operación mínima	Unsigned 32	rw	0ºC

Índce	SubIndice	Nombre	Descripción	Tipo	Acceso	Valor por defecto
	9	Corriente máxima carga	Corriente máxima carga	Unsigned 32	rw	0A
	11	Corriente máxima descarga	Corriente máxima descarga	Unsigned 32	rw	0A
	15	SOC mínimo apagado	SOC mínimo apagado	Unsigned 16	rw	20%
0x2008	0	Registro de alarma de bateria	Registro de alarma de bateria	Unsigned 32	ro	0

Índce	SubIndice	Nombre	Descripción	Tipo	Acceso	Valor por defecto
0x6000	0	Battery-Status	Indica si la batería está preparada para recibir carga: 0 : no preparada 1 : preparada	Unsigned 8	ro	0
0x6010	0	Temperature	Temperatura interna del pack	Unsigned 16	ro	0
0x6020	0	Highest sub-index supported	Máximo número de subíndice	Unsigned 8	ro	4
	1	Battery Type	Tipo de batería empleada	Unsigned 8	ro	0x00
	2	Ah capacity	Capacidad de la batería	Unsigned 16	ro	75 Ah
	3	Maximum charge current	Amperios	Unsigned 16	ro	225A
	4	Number of cells	Número de celdas en el pack de baterías	Unsigned 16	ro	16
0x6060	0	Battery Voltage	Tensión en el pack de baterías	Unsigned 32	ro	0

Índce	SubIndice	Nombre	Descripción	Tipo	Acceso	Valor por defecto
0x6081	0	Battery state of charge	Medición de la cantidad de energía contenida en la batería por parte de la batería en porcentaje, con una resolución del 1%.	Unsigned 8	ro	0x00

Dentro del archivo “CAN.c” se cuenta con las siguientes funciones, las cuales podrán ser llamadas desde el resto de los archivos incluyendo previamente el fichero .h correspondiente.

- CanHW_Init():inicialización del puerto de comunicaciones CAN, configuración del filtro CAN para solo recibir mensajes con ID 601 y habilitación de las interrupciones para recepción de mensajes CAN.
- Init_FIFO (void): inicialización de los valores de la cola FIFO.
- Recibir_CAN (): recepción del mensaje CAN y encolado en la cola FIFO.
- Leer_CAN (uint8_t New_Dato[MAXBUF]): desencola un mensaje de los mensajes de la cola FIFO.
- Tratar_CAN (uint8_t Datos[MAXBUF]): procesa el mensaje recibido por CAN y actúa dependiendo del mensaje que sea.
- Enviar_CAN(uint8_t Command_Byte, uint16_t Indice, uint8_t Subindice, uint32_t Bytes_Datos): escritura de mensajes por el puerto de comunicaciones CAN.

UART: Dentro de este fichero se contará con las funciones necesarias para el tratamiento de los mensajes recibidos por UART, así como las peticiones de lectura hacia el BMS de Texas Instruments. Las funciones con las que se contará serán las siguientes:

- UART_Init(): se inicializa la UART y se habilitan las interrupciones.
- Leer_UART(): cada vez que se recibe un mensaje por la UART se procesa el mensaje recibido por medio de esta función.
- Peticion_Lectura_UART(): se procede al montaje del mensaje con el que se producirá la petición de lectura al BMS; una vez montado se procederá a su envío por medio de UART.

SRAM: Desde este fichero se accederá a la memoria, pudiendo almacenar en ella cada uno de los parámetros que sean necesarios. En un principio se planteaba almacenar los parámetros en la memoria SRAM, ya que en este tipo de

memoria se puede escribir y leer directamente sobre una posición de memoria, mientras que para una memoria flash se debe actuar sobre bloques de memoria para proceder a la escritura. Sin embargo, finalmente se ha tenido que escoger la flash para el almacenado de parámetros. La decisión de grabar los parámetros en la memoria flash viene motivada porque esta es una memoria no volátil, es decir, aunque se produzca una pérdida de alimentación en el microcontrolador, los parámetros quedan grabados para poder utilizarlos cuando se realimente la placa de nuevo, la memoria SRAM no cuenta con esta característica y por tanto fue descartada.

La capacidad de poder grabar datos en memoria no volátil, es necesaria para los datos modificados desde la aplicación de escritorio, ya que estos deben ser accesibles siempre que se encienda de nuevo la placa MCBSTM32C. En la Figura 49, se puede observar el mapa de memoria perteneciente al microcontrolador de la placa empleada en el presente proyecto, como se puede observar, la dirección de inicio de la memoria flash es 0x08000000, sin embargo, en el Firmware se ha establecido un margen de maniobra y se ha seleccionado como la dirección de inicio de la memoria flash la dirección 0x0803F800, evitando de esta forma sobrescribir posiciones de memoria que ya se encontraran en uso.

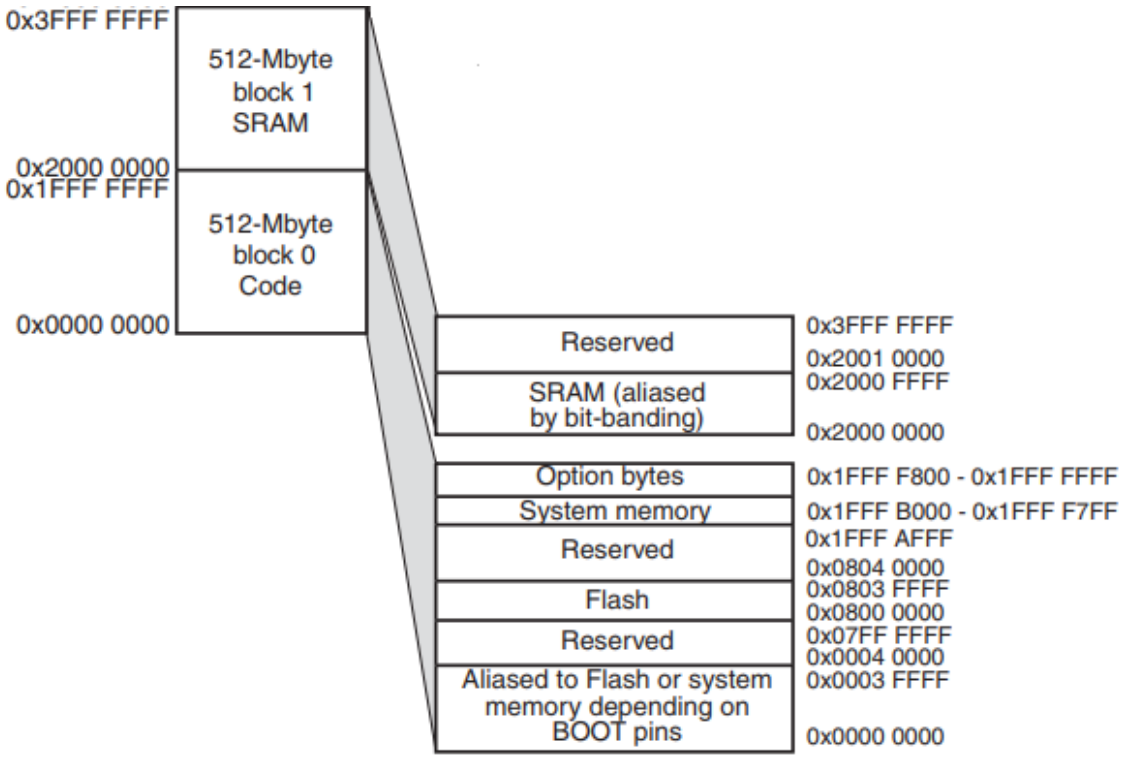


Figura 49. Mapa de memoria del STM32F107VC

En este fichero se contará con las siguientes funciones:

- Escribir_FLASH (uint32_t Pos_mem, uint32_t Valor): se escribe sobre la posición de memoria (indicada como parámetro) el valor que se introduce también como parámetro. Es importante escribir cada parámetro con la posición de memoria adecuada, ya que para proceder a la lectura del parámetro se accederá a esa posición de memoria.
- Leer_memoria (uint32_t Pos_mem): esta función devuelve el valor almacenado en la posición de memoria indicada como parámetro.

ADC: desde este fichero se podrá manejar el sensor de corriente. En la realización del presente proyecto el sensor de corriente empleado es el siguiente: LEM HASS 200-S.

El sensor cuenta con las siguientes características:

- Medida siguiendo el principio del efecto Hall.
- Aislamiento galvánico entre el circuito primario y secundario.
- Bajo consumo de potencia.
- Alimentación de 5 V.
- Cuenta con offset.

En cuanto al reparto de pines, se deberá seguir la siguiente distribución:

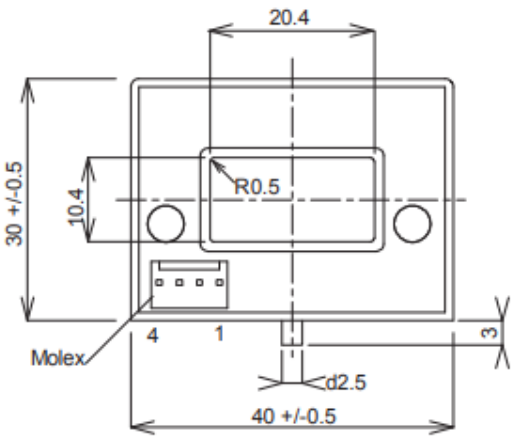


Figura 50. Distribución de pines del sensor de corriente LEM HASS 200-S

Retomando los aspectos del fichero “ADC.c”, en este fichero tan solo se cuenta con una función:

- Lectura_ADC(): mediante esta función se obtiene el valor de cuenta leído por el ADC y se realizan las conversiones necesarias para poder interpretar ese valor en Amperios. Este valor será el correspondiente a la intensidad que circula por el pack de baterías, siendo negativo para los momentos en

los que se encuentra en descarga y positivo para los momentos que se encuentra en carga.

BMS: en este fichero se encuentran las funciones encargadas de hacer los cálculos de los parámetros a obtener, encontrándose la mayoría de las funciones del proyecto, enumerando éstas se tendrán:

- BMS_InitData(): se inicializan los datos del BMS.
- BMS_BQ_UpdateData(): actualización de los parámetros del BMS.
- BMS_Analog_UpdateData(): actualiza el valor de la corriente que circula por las baterías.
- Evaluar_Posibles_Error(): se comprueba la existencia de errores en el sistema.
- Param_Calc (float temp_actual, tMOD_temps temp_min, tMOD_temps temp_max, tMOD_current current0, tMOD_current current1, tMOD_param param, uint8_t Cell): calcula y devuelve el valor interpolado de cada uno de los parámetros que se introducen.
- BMS_Lookup_Table (float * Table, float index): calcula y devuelve el valor interpolado para un índice introducido.
- BMS_SOC_Celda (uint8_t Cell): calcula el SOC de la celda indicada.
- BMS_SOH_Celda (uint8_t Cell): calcula el SOH de la celda indicada.
- BMS_Actualizar_SOC_SOH_Celdas(): actualiza el SOC y SOH de las celdas.
- BMS_Obtener_SOC_VOC (float * Table, float Vop): devuelve el SOC asociado a la tensión en abierto de la celda.
- BMS_SOH_Estimacion_SOH (uint8_t Cell): estimación del estado de salud (SOH) de la celda.

Los ficheros que se han mencionado anteriormente son los ficheros principales, acompañados cada uno de ellos de su fichero .h. Sin embargo, para poder poner en funcionamiento la placa MCBSTM32C es necesario contar con otros archivos. El proyecto compilado completo se encuentra dentro de la carpeta del proyecto (TrabajoFinMasterSamuelSanchezBaeza.zip), en ella se podrá observar todos los ficheros que se han mencionado en este apartado junto con el código empleado en cada una de las funciones nombradas. Además, en el código de los principales archivos .c se puede observar que el código dispone de comentarios para entender qué se realiza en cada línea del código.

7. APLICACIÓN DE ESCRITORIO PARA PC

Con el objetivo de visualizar los parámetros de las baterías, se desarrolla una aplicación de escritorio para PC, esta se ha desarrollado mediante la herramienta de programación Visual Studio bajo el lenguaje de programación C#.

A diferencia de la programación del microcontrolador, no se puede elaborar un diagrama de flujo a seguir, ya que la programación de la aplicación de PC está desarrollada bajo una programación orientada a objetos, dependiendo cada función que se vaya a realizar del evento que se haya generado[6].

Para entender cómo funciona esta aplicación de escritorio, se hará una breve descripción de cada una de las pantallas de esta, además se puede visualizar el código de la misma accediendo a TrabajoFinMasterSamuelSanchezBaeza.zip, lugar donde se encuentra almacenada.

Sin más dilación se pasará a la descripción de las distintas ventanas. En primer lugar, puede observarse cómo al iniciarse la aplicación, se dispondrá de una ventana donde se visualizará el logo de la Universidad Politécnica de Valencia durante un período de 2 segundos gracias a la utilización de un timer, que una vez transcurridos hará saltar una función encargada de cambiar de pantalla.

Figura 51. Logo mostrado al iniciar la aplicación de PC

La siguiente ventana que se mostrará será la pantalla de inicio de sesión, mediante la cual se procederá al acceso al menú principal de la aplicación.

[6] En Visual Studio existe una gran variedad a la hora de generar eventos. En este proyecto, la mayoría de los eventos son generados a través de pulsación de botones o temporizadores.

Figura 52. Pantalla de inicio de sesión

Existen dos tipos de usuario. Por un lado, un modo súper-usuario que cuenta con todas las posibles funciones de la aplicación y, por otro lado, un segundo usuario que tendría restringidas ciertas funciones. Para distinguir entre cada uno de estos usuarios, el acceso en el inicio de la sesión tendrá una contraseña[7] determinada para cada perfil. Una vez introducida la contraseña, se pulsará sobre el botón "iniciar sesión" y si esta es válida se accederá a la siguiente ventana, de lo contrario aparecerá un mensaje de error indicando que la contraseña introducida no es válida, evitando de esta forma el acceso a la aplicación de personal no autorizado.

La siguiente ventana a la que se accederá será la ventana principal, desde ella se pulsará sobre ajustes y se seleccionará "dispositivo". Una vez seleccionado se accederá a la ventana "conexión de dispositivos", desde esta se podrá conectar un dispositivo CAN contando con dos variantes. La primera de ellas sería una conexión WiFi-CAN y la segunda una conexión vía USB-CAN. Tanto para una como para la otra, los posibles dispositivos conectados a nuestro PC aparecerán en pantalla; se deberá seleccionar el adecuado y pulsar sobre el botón "conectar dispositivo" para el caso de USB-CAN o sobre el botón "conectar", previa introducción de la contraseña (si fuera necesario), para la conexión WiFi-CAN.

[7] La contraseña para el modo con restricciones será: 1111

Figura 53. Pantalla de conexión de dispositivos CAN

Una vez establecida la conexión CAN, por cualquiera de los dos medios y nunca ambos al mismo tiempo, se podrá hacer un uso normal de la aplicación ya que se habrá terminado con los pasos previos de configuración (inicio de sesión acorde al tipo de usuario y conexión del dispositivo CAN).

En la Figura 54 se visualiza la ventana principal de la ventana de escritorio, desde ella se podrá acceder a cada una de las ventanas. Siendo estas:

- Modificar parámetros: modificación de parámetros del BMS.
- Visualizar parámetros: visualización de parámetros del BMS.
- Visualizar gráficos: gráficas de tensión y temperatura por celda.
- Visualizar errores: visualización y log (almacenado) de los errores producidos.
- Modo súper-usuario: envío de todos los comandos mediante el montaje de la estructura de los mensajes CAN. Esta ventana solo será accesible si se ha iniciado sesión desde el modo súper-usuario, en el caso de que se inicie sesión desde otro usuario, este botón no será visible.
- Usuario: acceso a la ventana de inicio de sesión.
- Dispositivo: acceso a la ventana de conexión de dispositivos CAN.
- Acerca de: información acerca de la aplicación de PC.
- Salir: botón desde el que cerrar la sesión.

Modificar parámetros

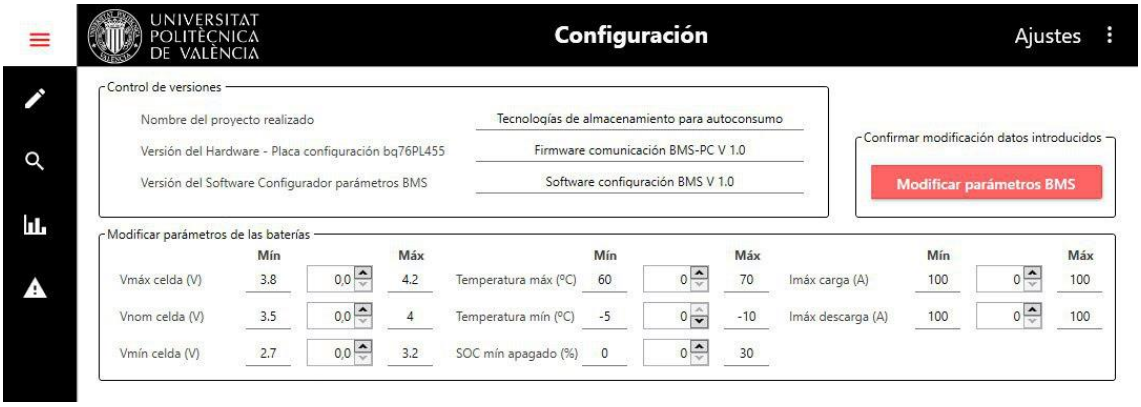


Figura 54. Pantalla de modificación de parámetros

Desde esta pantalla se procederá a la modificación de los parámetros del BMS. Para ello, se deberá introducir en el recuadro habilitado el valor deseado (siempre dentro de los límites marcados como mínimo y máximo). Una vez introducido se pulsará sobre el botón “modificar parámetros BMS”, de esta forma se confirmará la acción y estos datos serán grabados en la memoria flash del microcontrolador.

Desde esta ventana, también se podrá visualizar el control de versiones, indicando cuál es la versión más reciente de los dispositivos presentes en el sistema.

Visualizar parámetros

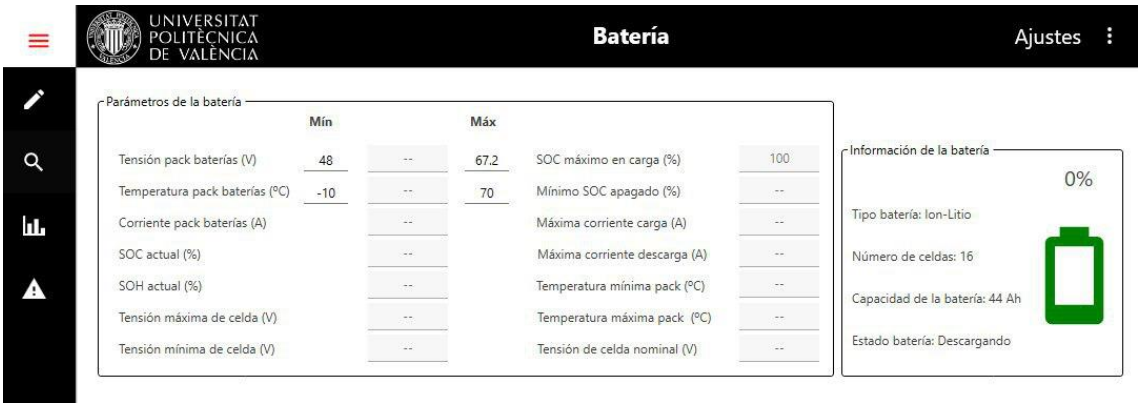


Figura 55. Pantalla de visualización de parámetros

Desde esta ventana es posible visualizar cada uno de los parámetros procedentes del BMS (los que hayan sido tratados). En ella, se encuentra información correspondiente al pack de baterías. Como se puede observar en la figura anterior, se cuenta con dos campos. El primero de ellos dará valores numéricos de los distintos parámetros del BMS (tensión en el pack de baterías, temperatura en el pack, corrientes, etc.). En el segundo de ellos, se tiene

información acerca de las celdas de baterías, dando información sobre el estado actual, además del porcentaje de carga actual.

Visualizar gráficos

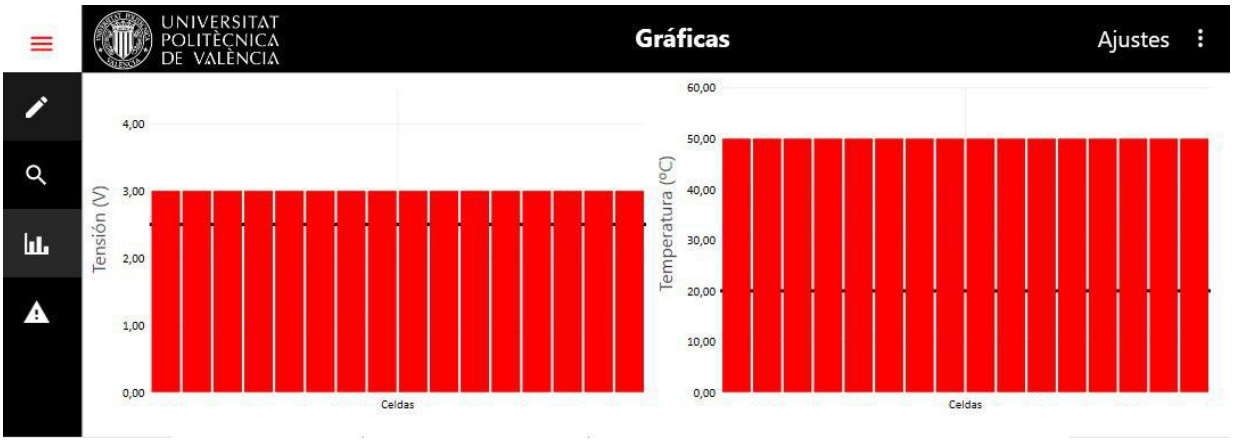


Figura 56. Pantalla de visualización de gráficos

Mediante esta ventana será posible visualizar gráficas en las que se indica el nivel de tensión y temperatura para cada una de las celdas, de esta forma será fácil visualizar si una de ellas se encuentra fuera de los rangos límite para su correcto funcionamiento, además de una interesante comparación entre los niveles de cada una de las celdas.

Visualizar errores

Figura 57. Pantalla de visualización de errores

Desde esta ventana se podrá visualizar los errores presentes en el sistema, añadiéndose automáticamente al entrar dentro de esta ventana. En el caso de que una vez dentro de la ventana se desee determinar si se ha producido algún otro error adicional, se pulsará sobre el botón “actualizar errores”, actualizándose de esta forma los errores presentes en el sistema.

En el caso de que se desee eliminar los errores que ya han sido visualizados, se podrá pulsar sobre el botón “vaciar lista”, permitiendo de esta forma el borrado de los errores presentes.

Otra característica interesante de esta pantalla es que se podrá proceder a un log de los errores que se hayan producido generando para ello un archivo.csv en el que quedarán registrados cada uno de los errores que se hayan producido desde ese instante.

Modo súper-usuario

Figura 58. Pantalla modo super-usuario

Desde esta ventana se podrá acceder a cada uno de los parámetros por medio de comunicación CAN. Es importante que esta ventana tan solo sea utilizada por personal autorizado o, en su defecto, personal con conocimientos acerca del proyecto, ya que de lo contrario se pueden producir daños sobre alguno de los dispositivos electrónicos. Es por esta razón por la que esta pestaña únicamente se encontrará visible en el caso de que se inicie sesión desde el modo súper-usuario.

Para proceder a la modificación de cada uno de los parámetros, se deberá rellenar cada uno de los bytes de datos que aparecen en esta ventana, indicar el DLC (longitud en bytes del mensaje) e ID del mensaje CAN (601 para el caso del BMS). Una vez introducidos cada uno de estos datos será posible acceder al envío del mensaje CAN. Para facilitar este proceso se han añadido el resto de campos que se pueden visualizar en la Figura 58, desde ellos se podrá seleccionar, mediante desplegables, la opción deseada, montando de esta forma la estructura del mensaje CAN a enviar.

Finalizado el montaje del mensaje CAN, ya sea de forma manual o con ayuda de los desplegables, se pulsará sobre el botón “enviar mensaje CAN manual”, enviándose así el mensaje CAN con los parámetros bytes de datos, DLC e ID rellenados.

En este apartado se ha podido visualizar cada una de las ventanas que componen la aplicación de escritorio para PC y cuáles son las acciones que se pueden llevar a cabo desde cada una de ellas, pero detrás de todo el aspecto visual hay una programación bajo el lenguaje de programación C#, dicho código se puede encontrar en TrabajoFinMasterSamuelSanchezBaeza.zip. De igual forma que el código desarrollado para Keil uVision 4, este código no se ha añadido en los anexos, ya que se trata de un código demasiado extenso. Cuenta con una parte de código en XAML dedicado a la programación de la interfaz gráfica y otra parte desarrollada en C# desde la cual se maneja el control de la interfaz gráfica, además del resto de funciones internas que se deben realizar para el correcto funcionamiento de la aplicación.

Por tanto, en el caso de que fuera de interés la programación de la aplicación de escritorio, se deberá acudir a la carpeta del proyecto en la que se encuentra la aplicación de PC desarrollada en Visual Studio.

8. RESULTADOS OBTENIDOS Y CONCLUSIONES

En el presente apartado se recopilan los resultados obtenidos en la realización del Trabajo Fin de Máster. Para mostrar los resultados se han tomado diferentes capturas de pantalla de cada uno de las herramientas de programación en modo depuración, en ellas se podrá visualizar los parámetros que se han obtenido en las pruebas de funcionamiento, además se harán aclaraciones acerca de estos resultados.

En primer lugar, se va a ver cómo a la hora de realizar las comunicaciones, ya sea a través de CAN como a través de UART, los dispositivos son capaces de comunicarse entre ellos. Para demostrar este hecho, se puede observar en la Figura 59 y Figura 60 cómo cada vez que se recibe un mensaje de comunicación salta la interrupción correspondiente. Es decir, se ha conseguido realizar una comunicación estable entre Aplicación de PC - MCBSTM32C (CAN) y MCBSTM32C – bq76PL455 (UART), ya que en el caso de que existiera algún error[8] no saltaría la interrupción, se tendría un fallo en la comunicación y por tanto no se podría enviar o recibir mensajes, no logrando nunca el salto de la interrupción.

```
398  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart2)
399  {
400    __NOP();
401    Estado_Actual = Estado_Tratar_UART;
402  }
```

Figura 59. Salto de interrupción en la recepción de mensaje por UART

```
220  void CAN1_RX0_IRQHandler(void)
221  {
222    /* USER CODE BEGIN CAN1_RX0_IRQn 0 */
223
224    /* USER CODE END CAN1_RX0_IRQn 0 */
225    HAL_CAN_IRQHandler(&hcan1);
226    /* USER CODE BEGIN CAN1_RX0_IRQn 1 */
227
228    //Leer_Mensaje_CAN();
229    Recibir_CAN();
230
231    /* USER CODE END CAN1_RX0_IRQn 1 */
232  }
233
```

Figura 60. Salto de interrupción en la recepción de mensaje por CAN

[8] Los errores en las comunicaciones CAN o UART se deben a mala asignación de pines, baudrate distinto en los diferentes dispositivos empleados, etc.

Esto también demostrará que es posible realizar envíos por medio de la UART, ya que para poder recibir los mensajes por UART se debe haber realizado antes una petición de lectura, enviando para ello un mensaje determinado por UART, véase Figura 61, en la cual se podrá observar la función mediante la cual se procede al envío de la petición de lectura de los parámetros del bq76PL455.

```
void Peticion_Lectura_UART()
{
  uint32_t Mensaje_Montado;

  //--- Single device write with response, 8 bit-addresing, 1 data byte, device address 0,
  char bytes_to_send[Tamanyo_Mensaje_Peticion];

  // --- Montamos el mensaje de petición de lectura ---
  bytes_to_send[0] = 0x81; // single device write with respone
  bytes_to_send[1] = 0x00; // Device address: 0
  bytes_to_send[2] = 0x02; // Register 02
  bytes_to_send[3] = 0x20; // Read sampled values command
  bytes_to_send[4] = 0x28;
  bytes_to_send[5] = 0x84;

  //--- Transmitimos la petición de lectura ---
  HAL_UART_Transmit(&huart2, (uint8_t*)bytes_to_send, Tamanyo_Mensaje_Peticion, 1000);

  // --- Comprobamos si existe algun error en la comunicacion UART ---
  if (huart2.ErrorCode == HAL_UART_ERROR_NONE)
  {
    Errores_Presentes = NO_Error;
  }
  // --- Existe un error en la comunicación ---
  else
  {
    Errores_Presentes = ERROR_UART;
  }

}
```

Figura 61. Envío de la petición de lectura de los parámetros del bq76PL455

Una vez comprobado que se cuenta con una comunicación correcta, se puede ver si los mensajes que se reciben de cada uno de los dispositivos son correctos. Cada vez que se hace una petición de lectura desde la aplicación de escritorio (esta se puede distinguir porque el primer byte de datos es 0x40), se recibe un mensaje CAN, este se procesa y se envía la respuesta dependiendo del mensaje que sea.

En la Figura 62, se puede ver un ejemplo de recepción de una petición de lectura del SOC, en la que se observa cómo se ha procesado el mensaje que se había recibido y se ha determinado que el índice que compone el mensaje es el índice 0x6081, el cual pertenece al SOC del pack de baterías, como ya se ha visto con anterioridad.

```
612
613         //-------------------------------------------------------------------------
614         //  0x6081
615         case  (0x6081):
616         {
617           Bytes_Datos_CAN = BatteryPack.SOC_Pack;
618         }
619       }
620       // --- Enviamos la respuesta a la petición de lectura que hemos recibido ---
621       Enviar_CAN(Command_Byte_CAN, Indice_CAN, Subindice_CAN, Bytes_Datos_CAN);
622   }
```

Figura 62. Petición de lectura del SOC del pack de baterías

En la Figura 62 se puede observar como una vez procesado el mensaje y sabiendo cual es la petición de lectura recibida, se procede al envío de la respuesta al mensaje, siendo la función “Enviar_Can” la encargada de enviar esta respuesta. En esta función, se envía el valor numérico que se tiene almacenado en el parámetro solicitado (SOC en este ejemplo).

Las respuestas a las peticiones de lectura que envía el MCBSTM32C serán recibidas por la aplicación de escritorio, obteniendo así los parámetros que se pueden ver representados en las siguientes figuras, donde se observa la aplicación de escritorio en funcionamiento, otorgando cada uno de los parámetros que se pueden visualizar del pack de baterías.

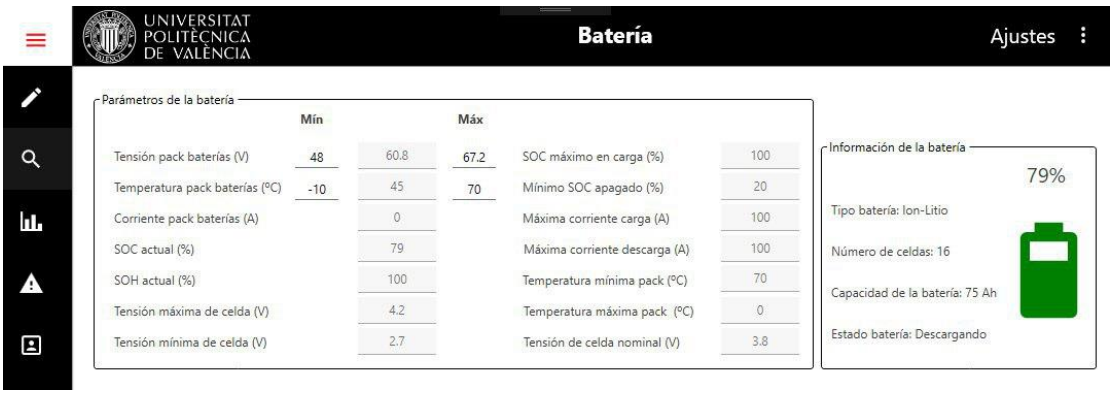


Figura 63. Representación de los parámetros en la aplicación de escritorio

Figura 64. Representación de los parámetros en la aplicación de escritorio

Por otro lado, desde la aplicación de PC se podrán enviar mensajes de escritura, tal y como se puede observar en la Figura 65. Con estos mensajes se pueden modificar los parámetros de las baterías. Principalmente, estos servirán para configurar las alarmas de error, estableciendo los límites de las baterías para que salte el error de mal funcionamiento en el BMS si se supera dicho límite. Los datos que se tienen rellenados en los campos de la aplicación de escritorio, en el momento en el que se realiza el pulsado del botón “Modificar parámetros BMS”, son escritos sobre la memoria FLASH.

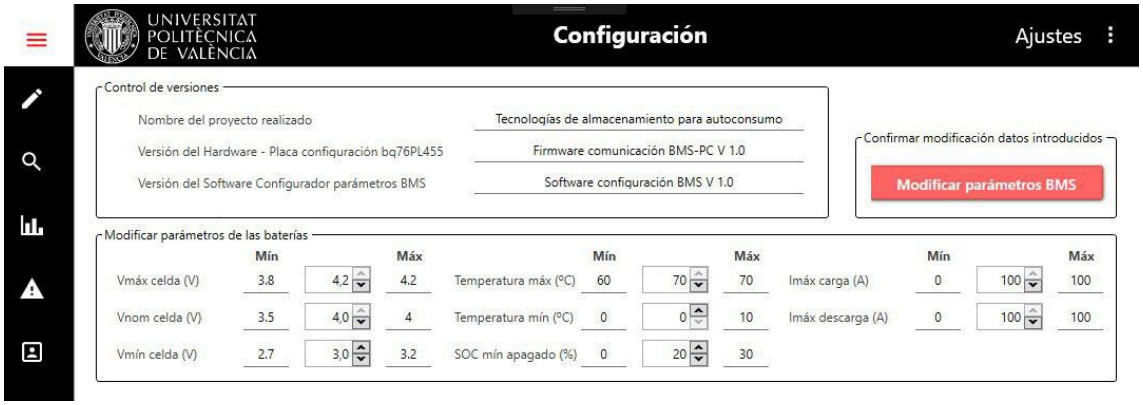


Figura 65. Ventana modificar parámetros

Como se puede observar en la Figura 66, estos mensajes de escritura (primer byte de datos, command byte, rellenado con 0x23) serán procesados por el MCBSTM32C, de igual forma que para las peticiones de lectura, con la diferencia de que ahora en vez de contestar con un mensaje de respuesta por CAN, el dato que se ha recibido por medio de CAN será almacenado en la memoria FLASH, estableciendo de esta forma los límites que se han indicado desde la aplicación de escritorio para cada parámetro. Guardando estos parámetros en FLASH se asegura que estos se encuentren presentes en el MCBSTM32 siempre, independientemente de si se pierde la alimentación de la placa o se realiza un reset, ya que al iniciarse el algoritmo se lee de la FLASH el valor de cada uno de estos parámetros.

```
623   // --- si es de escritura ---
624   else if (Command_Byte_CAN == 0x23)
625   {
626     switch (Indice_CAN)
627     {
628       case  (0x2006):
629       {
630         switch (Subindice_CAN)
631         {
632           case(0x03):
633           {
634             // --- Guardamos en las estructuras de datos ---
635             Conf_Celdas.TensionMaxima = Bytes_Datos_CAN;
636             BatteryPack.V_Max_Celda = Conf_Celdas.TensionMaxima;
637
638             // --- Guardamos en la memoria los datos ---
639             Escribir_FLASH(FLASH_Tension_Max_Celda, Conf_Celdas.TensionMaxima);
640             break;
641           }
642           case(0x04):
643           {
644             Conf_Celdas.TensionNominal = Bytes_Datos_CAN;
645             BatteryPack.V_Celda_nom = Bytes_Datos_CAN;
646
647             // --- Guardamos en la memoria los datos ---
648             Escribir_FLASH(FLASH_Tension_Nom_Celda, BatteryPack.V_Celda_nom);
649             break;
650           }
```

Figura 66. Escritura sobre la FLASH del MCBSTM32C

Desde el entorno de programación Keil uVision se puede acceder al mapa de memoria del microcontrolador, si accedemos a este se podrá ver cómo los parámetros que se han enviado desde la aplicación de escritorio como un mensaje de escritura, se han guardado en la memoria FLASH del microcontrolador. En la Figura 67 se puede observar cómo se ha almacenado en la FLASH los parámetros que se habían escrito desde el PC en la Figura 65. Es importante mencionar que los datos que se envían desde la aplicación tienen magnitud x10, es decir, se enviarán los valores deseados multiplicados por 10, de esta forma se elimina el valor decimal del numero enviado. Este fenómeno se realiza para poder almacenar y enviar por CAN los valores de cada parámetro de forma más sencilla, sin embargo, será imprescindible dividir este valor entre 10 en el microcontrolador para así disponer del valor float necesario.

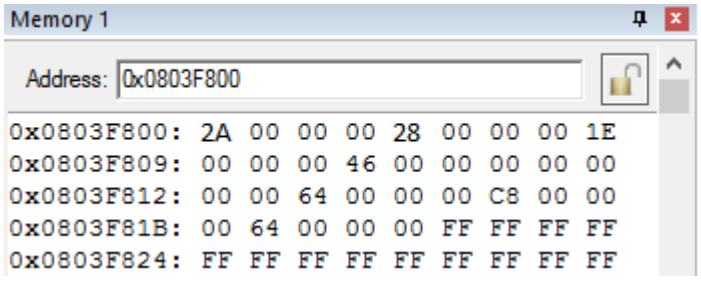


Figura 67. Acceso a la memoria FLASH del MCBSTM32C

Como se puede observar, el guardado sobre la FLASH se realiza correctamente, poniendo un ejemplo práctico, se puede observar cómo la tensión máxima por celda “Vmáx celda (V)” (Figura 65), tiene un valor de 4.2 V, siendo el valor de 42 tras aplicarle la magnitud por 10 mencionada anteriormente. Este valor en hexadecimal sería 2A, por tanto, en la FLASH se debería almacenar 2A. Tal y como se puede observar en la Figura 67, este es el valor almacenado en FLASH, indicando por tanto que el almacenado de datos en la FLASH se realiza correctamente.

Otro aspecto interesante a comentar en este apartado de resultados, es la actuación de la estructura BatteryPack, dentro de ella se almacenan cada uno de los parámetros que serán enviados por CAN (puede verse la definición de la estructura en BMS.h, véase TrabajoFinMasterSamuelSanchezBaeza.zip). En la siguiente imagen, se pueden observar rellenados los parámetros que se han mencionado, los cuales se rellenan mediante los datos que se reciben del bq76PL455 (UART) y los cálculos que se realizan mediante la obtención de estos datos. El hecho de que cada uno de estos parámetros se rellene adecuadamente, determina que se ha realizado correctamente la comunicación y el procesado de estos datos.

Watch 1

Name	Value	Type
BatteryPack	0x20000214 &BatteryP...	struct sPack
BatteryStat...	0x01 READY	enum (eStatus)
SOH_Pack	0x00000064	int
Temp_Cel...	0x2000021C	float[16]
V_Celdas	0x2000025C	float[16]
Direccion_...	0x00000000	int
I_Pack	0	float
SOH_Celdas	0x200002A4	int[16]
SOC_Celdas	0x200002E4	int[16]
V_Max_Cel...	4.19999981	float
V_Celda_n...	3.79999995	float
V_Min_Cel...	2.70000005	float
V_Min_Des...	0	float
Temp_Op_...	70	float
Temp_Op_...	5	float
I_Max_Carga	100	float
I_Max_Car...	0	float
I_Max_Des...	100	float
I_Max_Des...	0	float
SOC_Min_...	20	float
Temp_Pack	35	float
Tipo_Bater...	0x00000001	int
Capacidad...	0x0000004B	int
V_Pack	62.4000015	float
SOC_Pack	0x0000004F	int

Figura 68. Parámetros de las baterías representados en la estructura BatteryPack

Por poner un ejemplo de los parámetros que se visualizan en la figura anterior, se puede observar cómo se rellenan los parámetros de acuerdo a los datos recibidos por UART del bq76PL455. En el instante en el que se tomó la captura anterior, se tenía por ejemplo una tensión del pack de baterías de 62.4 V o un SOC del pack del 79 % (4F en hexadecimal).

Otra de las funciones presentes en el presente proyecto es la alerta al usuario en el caso de que se produzca algún error en el sistema. En las siguiente figuras, se podrá visualizar que para la placa MCBSTM32C los avisos acerca del funcionamiento se realizan por medio de una serie de leds presentes en la misma placa. Sin embargo, este aviso es para pruebas funcionales, ya que más adelante se pueden sustituir estos leds por leds de mayor tamaño situados en el exterior de la placa o incluso se puede plantear la inclusión de zumbadores, que sean capaces de proporcionar al usuario un aviso acústico de que se está produciendo un error en el sistema, otra opción podría ser una mezcla de ambos, ya que si el lugar de trabajo se encuentra en una zona ruidosa como podría ser una industria, una señal solo acústica no sería suficiente.

En el sistema se puede distinguir entre dos tipos de errores, uno de ellos sería un error producido por un fallo en las comunicaciones, ya sea UART o CAN, mientras que el segundo error vendría provocado por unos parámetros de las baterías anómalo, es decir, fuera de los limites que se han marcado desde la aplicación de PC como valores máximos y mínimos.

```
450   // --- Si no existe ningun error quiere de
451   if (BatteryPack.Reg_Alarmas == 0)
452   {
453     BatteryPack.BatteryStatus = READY;
454     Errores_Presentes = NO_Error;
455   }
456   else
457   {
```

Figura 69. Sin existencia de errores en el sistema

Tal y como se observa en la figura anterior, cuando todos los parámetros están dentro de los límites, se tendrá el registro de alarmas a 0, es decir, no habrá errores y por tanto denominaremos nuestro estado de baterías como READY, preparadas para su funcionamiento. Este hecho será visible desde la propia placa del microcontrolador, ya que existirá un parpadeo en el led PE13, tal y como se puede observar en la Figura 70.

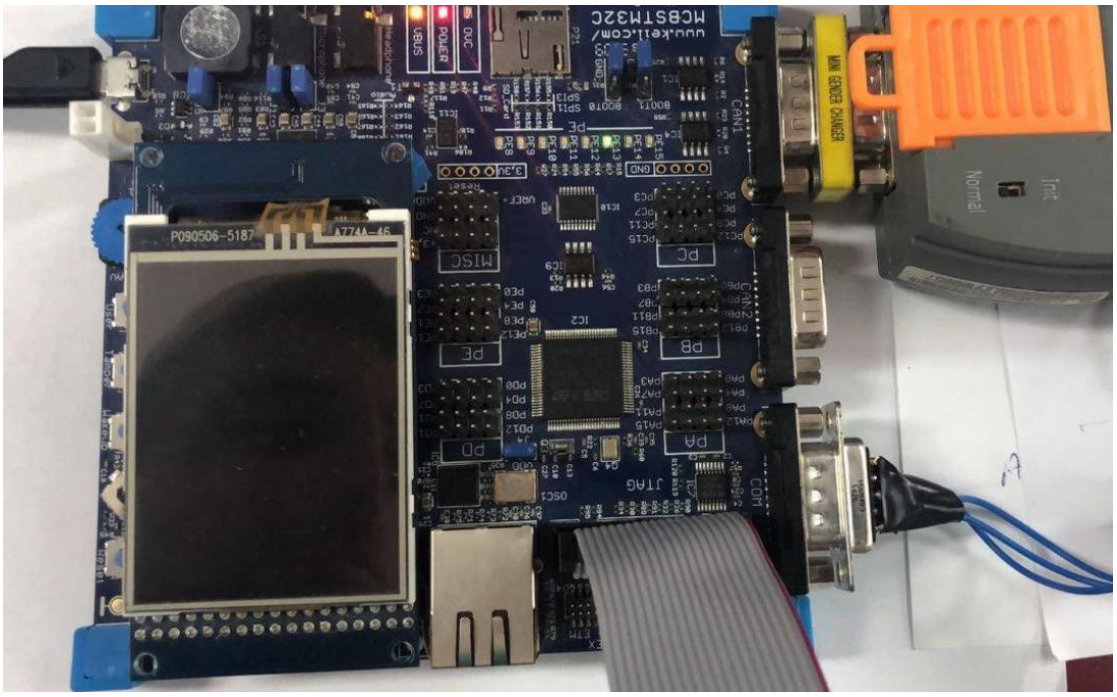


Figura 70. Encendido del led de correcto funcionamiento

Uno de los errores posibles, es un error por fallo en las comunicaciones, cada vez que se procede al envío de un mensaje por medio de CAN o UART se compara si existe algún código de error, “ErrorCode” tal y como se puede apreciar en la siguiente figura:

```
97   // --- Comprobamos si existe algun error en la comunicacion UART ---
98   if (huart2.ErrorCode == HAL_UART_ERROR_NONE)
99   {
100    Errores_Presentes = NO_Error;
101  }
102  // --- Existe un error en la comunicación ---
103  else
104  {
105    Errores_Presentes = ERROR_UART;
106  }
```

Figura 71. Error en las comunicaciones

En el caso de que se haya producido un fallo en las comunicaciones, tendremos un valor distinto a “HAL_UART_ERROR_NONE” o “HAL_CAN_ERROR_NONE” dependiendo del tipo de comunicación que se esté empleando en ese momento (CAN o UART). En la figura anterior se ve un ejemplo en el que se produce un error por fallo en las comunicaciones UART, tras poner la variable “Errores_Presentes” a “ERROR_UART”, se encenderá en la placa el led de error por comunicaciones, led PE8.

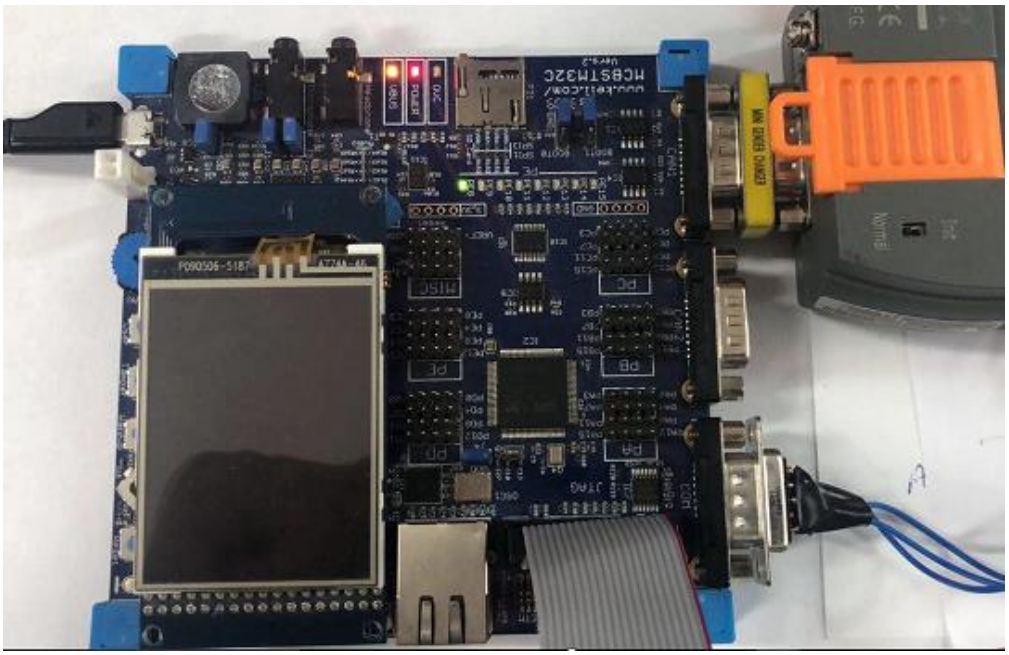


Figura 72. Led de error en las comunicaciones

Para el caso de error en el BMS, este se producirá cuando se obtenga un parámetro fuera de los límites. La comprobación de si los parámetros están dentro de los límites se realiza en la función “Evaluar_Posibles_Error”, cuando existe un error, el registro de alarmas será distinto de 0 y por tanto se pondrá el estado de la batería a “NOTREADY”, tal y como se puede observar en la Figura 73.

```
449   // --- Si no existe ningun error quiere decir que todos los parámetros son correctos,
450   if (BatteryPack.Reg_Alarmas == 0)
451   {
452     BatteryPack.BatteryStatus = READY;
453     //Errores_Presentes = NO_Error;
454   }
455   else
456   {
457     BatteryPack.BatteryStatus = NOTREADY;
458     //Errores_Presentes = ERROR_Parametros;
459   }
460 }
461
```

Figura 73. Error en los parámetros del BMS

Como consecuencia, se tendrá un parpadeo continuo del led PE9 hasta que desaparezca el error.

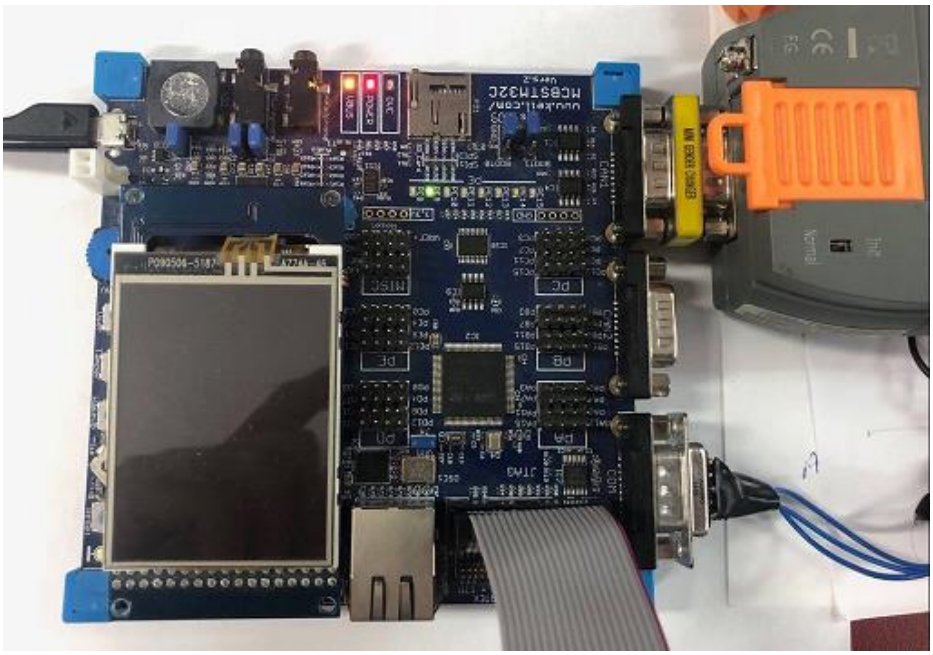


Figura 74. Led de error por parámetros anómalos en el BMS

Analizando los objetivos que se habían propuesto para el presente proyecto y observando los resultados que se han introducido en este apartado, se puede determinar que se ha cumplido con el objetivo propuesto para el presente Trabajo Fin de Máster, ya que se ha conseguido realizar un correcto modelado de las baterías de Litio, una correcta comunicación entre cada uno de los dispositivos presentes en el proyecto y un correcto procesado y cálculo de los parámetros que se comparten entre ellos.

Además, se ha obtenido una herramienta para la visualización de los parámetros de las baterías que se emplearán para cualquier uso que se les quiera dar y que mediante los avisos por medio de la aplicación de PC y los leds de la placa MCBSTM32C, permitirá determinar si alguna de las baterías estuviera funcionando con un comportamiento anómalo, pudiendo proceder de esta forma a la sustitución de esta antes de que pueda ocurrir un accidente que conlleve consecuencias negativas, ya sean materiales o físicas.

9. REFERENCIAS BIBLIOGRÁFICAS

1. ArmKeil (s.f.). MCBSTM32C. Technical Specifications. Recuperado de: http://www.keil.com/mcbstm32c/specs.asp

2. ArmKeil (s.f.). MCBSTM32C Evaluation Board. Recuperado de: http://www.keil.com/mcbstm32c/

3. ArmKeil (s.f.). MCBSTM32C Evaluation Board. User's Guide. Recuperado de: http://www.keil.com/support/man/docs/mcbstm32c/mcbstm32c_intro.htm

4. CAN (s.f.). Bit Time Calculation. Recuperado de: http://www.bittiming.can-wiki.info/

5. Dubarry, M. y Yann, B. (2007). Development of a universal modeling tool for rechargeable lithium batteries, vol. 174, núm. 2 pp. 856-860. Recuperado de: https://www.sciencedirect.com/science/article/pii/S037877530701381X

6. D.H. Doughty, P.C. Butler, R.G. Jungst, E.P. Roth (2002). Lithium battery thermal models, vol. 110, núm. 2, pp. 357-363. Recuperado de: https://www.sciencedirect.com/science/article/pii/S0378775302001982

7. Fabio Codecà, Sergio M. Savaresi, Giorgio Rizzoni (2008). On battery State of Charge estimation: a new mixed algorithm. Recuperado de: https://es.scribd.com/document/325212597/On-battery-State-of-Charge-estimation-a-new-mixed-algorithm-pdf

8. ITE (s.f.). ALHACENA: Tecnologías de almacenamiento e hibridación basadas en nuevos materiales que faciliten el autoconsumo. Recuperado de: http://www.ite.es/project/alhacena/

9. LEM (s.f.). Current Transducer HASS 200-S. Recuperado de: https://www.lem.com/sites/default/files/products_datasheets/hass_50_600-s.pdf

10. Microsoft (2018). Información general sobre Visual Studio. Recuperado de: https://docs.microsoft.com/es-es/visualstudio/ide/visual-studio-ide?view=vs-2017
11. Microsoft Visual Studio (2018). Wikipedia, Enciclopedia libre. Recuperado de: https://es.wikipedia.org/w/index.php?title=Microsoft_Visual_Studio&oldid=109806291

12. STMicroelectronics (2016). ST-LINK/V2 debugger. Recuperado de: https://www.st.com/resource/en/data_brief/st-link-slsh-v2.pdf

13. STMicroelectronics (2017). STM32F107xx Recuperado de: https://www.st.com/resource/en/datasheet/stm32f107vc.pdf

14. Suleiman, A. y Dennis, D. (2003). Rapid test and non-linear model characterisation of solid-state lithium-ion batteries. Recuperado de: http://cgi.ddoerffel.force9.co.uk/_publications/Published%20Paper.pdf

15. Texas instruments (2016). Bq76PL455A-Q1-Cell EV/HEV. Integrated Battery Monitor and Protector. Recuperado de: http://www.ti.com/lit/ds/symlink/bq76pl455a-q1.pdf

16. Texas instruments (2015). Bq76PL4554A-Q1. ExampleCode. Recuperado de: http://www.ti.com/lit/an/slva635/slva635.pdf

17. Universidad de Córdoba (2010). Prontuario del entorno de desarrollo Keil uVision 4. Recuperado de: http://www.uco.es/~el1mofer/Docs/IntPerif/P_KuV4_v1.0_s.pdf

18. Yongsheng, He, Wei Liu y Brain J. Koch (2010). Battery algorithm verification and development using hardware-in-the-loop testing, vol. 195, núm. 9, pp. 2969-2974. Recuperado de: https://www.sciencedirect.com/science/article/pii/S0378775309020448