Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica

DISEÑO Y CONSTRUCCIÓN DE UN SISTEMA DE CONTROL Y MONITORIZACIÓN PARA LA CARGA DE BATERIAS EN UN VEHICULO ELÉCTRICO

Alumno: Juan Caño Navas

Tutor: Prof. D. Miguel de la Fuente Ruz

Dpto: Ingeniería de Sistemas y Automática

1. ÍNDICE GENERAL

Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica

MEMORIA DESCRIPTIVA

Alumno: Juan Caño Navas

Tutor: Prof. D. Miguel de la Fuente Ruz

Dpto: Ingeniería de Sistemas y Automática

Diciembre, 2018

ÍNDICE

ÍNDICE DE FIGURAS

1. Encargo y redacción

El presente proyecto, titulado “Diseño y construcción de un sistema de control y monitorización para la carga de baterías en un vehículo eléctrico”, se ha realizado como Trabajo Fin de Grado en Ingeniería Electrónica Industrial, y a petición del equipo EPSJAÉN UJATEAM.

Dicho proyecto ha sido realizado por el alumno Juan Caño Navas, cuyo tutor es D. Miguel de la Fuente Ruz, profesor del Área de Ingeniería de Sistemas y Automática en la Escuela Politécnica Superior de la Universidad de Jaén.

2. Antecedentes

Previo a la exposición de la solución adoptada para el control y monitorización de la carga y descarga de las baterías de un vehículo eléctrico, hemos de realizar un estudio acerca de los siguientes temas:

- Las baterías de ión-litio: Consideraciones.
- Estado del arte.

Partiendo de estos estudios, se procede al diseño del sistema, que se encuentra compuesto por:

- Solución adoptada
- Sistema esclavo
- Sistema maestro

3. Objetivos

El objetivo final de este proyecto será la creación de un sistema de monitorización para el prototipo de motocicleta eléctrico diseñado por la Universidad de Jaén con motivo de la competición MotoStudent Electric, cumpliendo las funciones de vigilancia de tensión y temperatura, y actuando como medida de seguridad en caso de detectar valores extremos de dichos parámetros.

4. Estudio previo

Antes de pasar a la exposición del diseño objeto de este documento, se debe establecer un contexto previo al análisis, para conocer exactamente el problema y desde que perspectiva abordarlo, así como un conocimiento previo sobre qué soluciones existen a problemas similares. Para entender completamente el marco en el que se encuadra nuestra aplicación, se debe realizar un estudio acerca del mercado actual de baterías, de tal forma que se comprenda por qué se escoge concretamente una tecnología basada en litio y qué problemas necesidades implica esta decisión. Posteriormente debemos analizar las posibles problemáticas a las que nos enfrentamos en caso de no darle un correcto uso, mantenimiento y vigilancia a nuestros dispositivos. Finalmente se presentará un estudio sobre qué existe para solventar esta problemática y por qué la necesidad de desarrollar una solución adaptada a nuestro sistema.

4.1. Las baterías de ION-Litio

Es innegable que el paradigma de la automoción ha sufrido un vuelco en los últimos años con la aparición del vehículo eléctrico, y ello ha presentado nuevos e importantes retos en esta industria, y el más inmediato es el de almacenamiento de energía. En este marco de necesidad de impulso, han aparecido multitud de tecnologías con el propósito de dar solución a este problema y hacerse un hueco en este mercado en expansión, pero, ¿son todas ellas buenas alternativas? La respuesta no es algo trivial, sino que precisa de un estudio profundo de la aplicación concreta a la que se destina cada acumulador. A continuación se adjunta una tabla en la que se detallan las prestaciones de las tecnologías actualmente más usadas en automoción, y posteriormente se pasará a un análisis del mismo:

Tecnología Parámetros	Pb-ácido	Ni-Cd	Ni-MH	Li-ión [$LiCoO_2$]	LiFe	LI-PO
Voltaje (V/celda)	2v	1.2v	1.2v	3.6/3.7v	3.3v	3.7v
Autodescarga (%/mes)	3%-20%	10%	30%	8%	-	5%
Descarga en continua		10c	8c	1c	26c	20-45c
Descarga por picos	-	-	-	-	52c	30-90c
Mantenimiento	Bueno	Malo	Regular	Fácil	Bueno	Fácil
Ciclos de vida	500-800	1500-2000	300-500	400-1200	2000	>1000
Densidad energética [wh/l]	60-75	50-150	140-300	250-360	220	300
Energía específica [Wh/kg]	30-40	40-60	30-80	100-250	90-110	130-200
Potencia específica [W/Kg]	180	150	250-1000	250-340	3000	7100
Corriente carga rápida [C]	0.4	1- 2	1-2	1	4	1-2
Eficiencia. Carg/Desca	50%-92	70%-90%	66%	80%-90%	-	99.8%
Tolerancia a sobrecargas	-	M. buena	Media	M. mala	Mala	M. mala
Robustez a impactos	Buena	M. buena	Buena	M. mala	Media	M. mala
Altas temperaturas	Media	M. buena	Media	M. mala	Mala	M. Mala
Problemas de ecualización	No	No	No	Si	Si	Si
Seguridad	M. buena	M. buena	M. buena	M. buena	M.buena	Buena
Formato	-	Cilíndrico	Cilíndrico	Prisma	Pris/Cilin	Prisma

Tabla 1: Comparación de tecnologías en acumuladores

Tras poner en relación los diferentes aspectos a tener en cuenta podemos concluir que el uso de una u otra tecnología dependerá fundamentalmente de lo que se busque en la aplicación, por tanto se pasa a hacer un desglose acerca de las ventajas e inconvenientes que entraña el uso de cada una de ellas:

- **Polímero de litio:**

Esta tecnología es actualmente la que más prestaciones nos brinda, ya que su tecnología genera mayor tensión por elemento individual, y una gran capacidad de descarga. Moviéndonos al plano de competición, también aportan ventajas respecto otras tecnologías, por su gran densidad energética y capacidad. Por otro lado si pensamos en un vehículo cotidiano, podrían ser idóneas por la velocidad de su carga y el elevado número de ciclos de vida que tienen. Además, se trata de una tecnología económica.

Sin embargo, si se determina a utilizar esta tecnología, se debe tener en cuenta que no son adecuadas si se han de instalar en un entorno mecánico que entrañe posibilidad de impactos, se debe vigilar la sobredescarga con especial cuidado en esta tecnología y además no están indicadas para trabajar con temperaturas elevadas.

- **$LiFePO_4$**

Esta tecnología es una solución bastante equilibrada entre prestaciones, seguridad, rendimiento y durabilidad. En contraposición también precisan de un sistema de ecualización, y su respuesta frente a temperaturas y sobredescargas no es buena.

- **Ion de Litio**

Las soluciones basadas en iones de litio resultan ser las que mejores resultados arrojan en cuanto a autonomía, gracias a sus valores en densidad energética, lo que además se traduce en un ahorro de peso. Son por tanto, una buena solución en aplicaciones donde el espacio, el peso y la seguridad sean un factor limitante.

- **NiMH**

Se trata de una solución intermedia entre la tecnología de litio y tecnologías anteriores como el plomo ácido y el NiCd. Destaca por ser una tecnología mucho más madura que el litio, por lo cual resulta ser una alternativa bastante segura, sin embargo su potencia de descarga, densidad energética y eficiencia resultan ser insuficientes cuando se precisa de aplicaciones con mayor demanda.

- **NiCd**

Su principal virtud es la robustez, es la tecnología que menos se degrada por malas condiciones físicas, sobredescargas o cortocircuitos, además de trabajar relativamente bien a altas temperaturas. Además destacan por su número de ciclos de vida. Sin embargo, sus capacidades eléctricas están por debajo de la mayoría de soluciones aquí presentadas

- **Acumuladores de Pb**

Se trata de la tecnología que más se ha utilizado tradicionalmente en automoción, aunque actualmente su mantenimiento en el mercado se esté cuestionando por sus indudablemente inferiores capacidades. Destacan por lo económicas que resultan y lo accesibles al usuario, pero resultan inservibles en aplicaciones de demandas exigentes por su escasa densidad energética.

- **Conclusiones**

El contexto en el que se encuadra este proyecto requiere una baja densidad energética, ya que en una motocicleta de competición el espacio que ocupen los acumuladores resulta determinante para el correcto diseño estructural. Además se precisa de una capacidad razonablemente buena de descarga para obtener la mayor potencia posible. Ello nos hace desechar las tecnologías que no estén basadas en litio. Por su densidad energética, y lo equilibrado de su compromiso eficiencia-potencia-seguridad, las baterías basadas en iones de litio son las más adecuadas para nuestra aplicación.

4.2. Riesgos de las baterías ION-Litio

Todas las baterías independientemente de la tecnología que usen van a regir su comportamiento en función de dos parámetros clave: temperatura y tensión. Estos parámetros limitan entre que valores es seguro que la batería trabaje, y el salirnos de estos valores implica riesgos para el equipo y para la seguridad, en función de en qué zonas nos movamos. En el gráfico mostrado en la figura 1 podemos observar cuales son los riesgos en cada momento.

- **Efectos en la tensión**

Debido a la tecnología química empleada en estas baterías, los valores máximos de tensión que deben alcanzarse son 4.2V, y el mínimo fijado suele ser

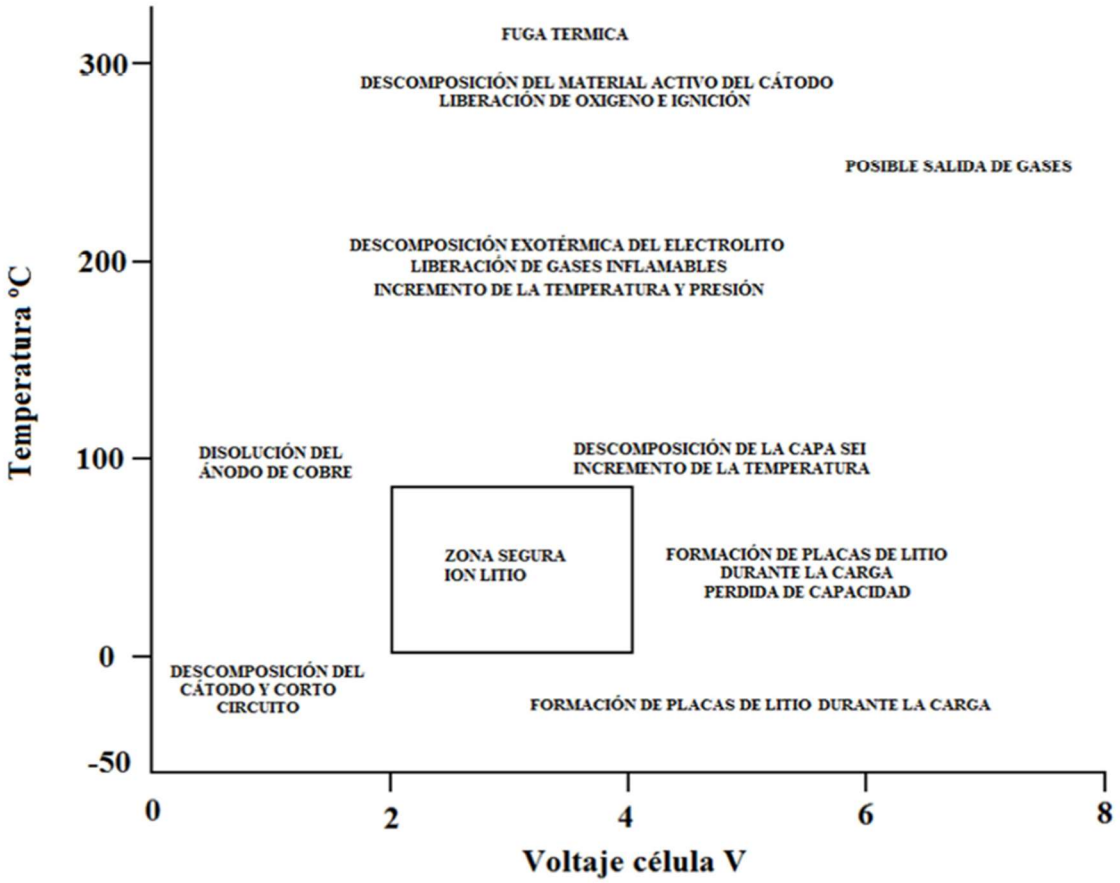


Figura 1: Esquema V-T; Degradación del dispositivo en función de la zona de trabajo.

3.7V. A continuación analizaremos los problemas que provoca movernos fuera de estos límites:

- **Efectos de sobrecarga**

Si se incrementa en exceso la tensión durante la carga, ello provocará una corriente interna de carga excesiva, lo que provocará:

- Formación de placas de litio: esto ocurre debido a que para valores de intensidad de carga muy altos, los iones de litio no tienen tiempo de acomodarse entre los estratos del ánodo, si no que se acumulan en la superficie del mismo formando una placa de litio. Este efecto es conocido como “Lithium planting”. Esto implica que los iones solidificados pasan a no estar libres durante la descarga, lo que implica una disminución de la

capacidad de descarga. En el caso más extremo, esta capa podría crecer hasta cortocircuitar los electrodos de la batería.

- Sobrecalentamiento: Una cantidad excesiva de intensidad producirá calentamiento por efecto Joule, lo que genera problemas sobre los que hablaremos a continuación en el apartado de temperatura.

➢ **Efectos de sobredescarga**

La sobredescarga también puede resultar igualmente perjudicial para el equipo, fundamentalmente es muy dañino para los electrodos, cuya descomposición puede verse fuertemente acelerada.

- Ánodo: el colector de cobre puede disolverse en el electrolito. Ello provoca una mayor autodescarga de la batería, y además, cuando se recargue la batería y el valor de tensión aumente por encima de 2 voltios, los iones de cobre precipitarán como cobre sólido y no tienen por qué volver a depositarse en el depósito de cobre del ánodo. Ello podría provocar un cortocircuito.
- Cátodo: los valores inferiores a 2 voltios durante un tiempo excesivo provocarán la descomposición gradual del cátodo, durante la cual se libera oxígeno del óxido de litio-cobalto y del óxido de litio-manganeso. Esto provocará un exceso de la presión en el interior del equipo que puede llegar a incluso a estallar si no se le facilita una salida controlada a estos gases.

- **Efectos de la temperatura**

La reacción de nuestro equipo será diferente ante bajas temperaturas que ante valores altos. A continuación se describirán los problemas que entrañan cada una de estas casuísticas:

➢ **Baja temperatura**

El índice de reacción química está relacionado con la temperatura debido a la ley de Arrhenius. Por lo tanto, un valor bajo de temperatura implica una disminución del número de reacciones químicas que ocurren dentro de la célula, y por tanto el equipo verá reducido el valor de corriente máxima que puede soportar durante la carga y la descarga. Esto implica una pérdida de potencia útil, y tendremos un mayor riesgo de generar el efecto “lithium planting”, tal y como se ha descrito en el anterior apartado, ya que al no soportar la corriente de carga, los iones de litio volverían a formar placas en la superficie del ánodo.

- **Alta temperatura**

Un exceso de temperatura suele desembocar en una reducción de la vida útil de la célula, e incluso en su destrucción en un caso extremo. Un aumento de la temperatura implica un aumento de las reacciones químicas, tal y como se describe en el anterior apartado. Esto provoca una mayor disipación de potencia en la resistencia interna de nuestra célula, lo que se traduce en generación de calor que provoca un aumento de la temperatura. Este bucle podría provocar que sin el control y la refrigeración adecuados terminaría generando una fuga térmica.

- **Fuga térmica**

No existe una única causa que produzca la fuga térmica, que puede iniciarse debido a una perforación del ánodo, o a un sobrecalentamiento del mismo. Éste a su vez puede estar provocado por una sobredescarga, un exceso de corriente o una temperatura externa excesiva.

Esto produce inicialmente la destrucción del estrato SEI (Solid Electrolyte Interface), que comienza a deteriorarse a partir de los 80º. Una vez dañada la capa, el electrolito reacciona con el carbono del ánodo de una manera descontrolada y dando lugar a una reacción exotérmica, lo que producirá un mayor aumento de la temperatura.

El aumento de temperatura producirá la descomposición de los disolventes orgánicos del electrolito liberando gases inflamables como etano y metano, pero no oxígeno. Esta descomposición comienza típicamente en torno a los 110ºC, pero puede llegar a ocurrir incluso a 70ºC. El gas producirá un aumento de presión, pero no se produce combustión debido a la ausencia de oxígeno. Las células suelen estar equipadas con mecanismos para extraer el gas sin poner la celda en riesgo, evitando que la misma explote. Sin embargo, estos gases a la temperatura a la que se expulsan comienzan a arder ante la presencia de oxígeno.

En torno a los 130ºC el separador de los electrodos se fundirá, provocando un cortocircuito, y el calor que genera el mismo provocara la descomposición del óxido del cátodo, liberando oxígeno que provocará la combustión de los gases que queden en el interior. La temperatura y presión se elevan tanto debido a la

combustión que provoca la explosión de la célula. La descomposición del cátodo suele ocurrir en torno a los 200ºC.

- **Conclusiones**

Dado el entorno de competición en el que el sistema va a ser instalado, resulta primordial realizar un control exhaustivo de temperatura, ya que de ello depende no sólo obtener el máximo rendimiento del equipo instalado, sino la seguridad del piloto y cuantos manipulen el prototipo durante la competición. También es fundamental una correcta instalación que permita la máxima refrigeración posible, y un control de carga que evite el deterioro de las celdas.

4.3. Estado del arte

La revolución electrónica que se ha vivido en las últimas décadas en la industria y electrónica de consumo en general ha exigido una evolución paralela de los sistemas de almacenamiento de energía, dando lugar a baterías que ofrecen mayores prestaciones. Es el caso de las baterías de litio, que presentan mayor densidad energética, menor efecto memoria, menor autodescarga y en consecuencia una mayor vida útil. Pero como inconveniente, este tipo de dispositivos poseen un margen de funcionamiento seguro bastante limitado, y fuera del cual pueden dañarse e incluso incurrir en riesgo de incendio ante las condiciones más adversas. Por ello, es vital un sistema de protección y control de estas celdas.

A grandes rasgos, estas son las funciones de un sistema de gestión de batería BMS (“Battery Management System”), monitorizar aquellos parámetros de los cuales depende el correcto funcionamiento de la batería, y posteriormente tomando decisiones sobre si los parámetros que ha recogido son admisibles y se encuentran dentro del límite del SOA, o por el contrario se han de tomar medidas sobre el sistema, como desconectarlo por seguridad.

Adicionalmente, un BMS también contempla otras funciones, pero la complejidad del mismo viene marcada por la aplicación para la que se contemple el mismo, siempre ofreciendo las funcionalidades necesarias para el usuario.

Nótese que el sistema BMS de un teléfono móvil compuesto por una sola celda de energía nunca alcanzará la complejidad del BMS de un vehículo compuesto por hasta cientos de celdas. Sin embargo, independientemente de la aplicación, los fabricantes de baterías estiman un aumento de la vida útil de

nuestras baterías cuando se instala un sistema BMS, incluso en sus variantes más simples.

- **Funciones**

Las funciones que típicamente puede realizar un sistema de gestión como el descrito son las siguientes:

- Adquisición de datos
- Determinación del estado de la batería
- Control de carga y descarga de la batería
- Equilibrado de las celdas
- Gestión térmica
- Gestión de seguridad
- Interfaz HMI con el usuario

- **Adquisición de datos**

En este aspecto podemos optar por dos soluciones. La primera es una solución centralizada, en la que todas las medidas van a un único bloque que procesa todas las señales. Es la solución en principio más simple y económica por el ahorro de material, sin embargo para instalaciones de muchas celdas puede implicar un aumento considerable del cableado, con los riesgos y aumento de costo que esto implica.

La segunda solución consiste en descentralizar la medida. Varios bloques tomarán la medida de cada celda de forma separada, y mediante un bus de comunicaciones transmitirán los datos recogidos al bloque general, el cual se encargará de procesar la información y de actuar sobre el circuito en caso necesario. La ventaja de este sistema es la simplicidad a la hora de agregar nuevas medidas, y el ahorro de cableado en instalaciones con muchas celdas, además de mejorar la robustez general del sistema, ya que en caso de fallar uno de los bloques, el resto pueden continuar funcionando sin ninguna anomalía.

El principal problema al que nos enfrentamos a la hora de medir las celdas en este tipo de sistemas es referenciar las tensiones de una cadena de celdas en serie. La solución más frecuente es utilizar canales de sondeo conmutado con tierras virtuales de referencia, pero este sistema ralentiza la

adquisición si buscas una medida precisa. Otra solución, que es la finalmente utilizada es aislar el circuito de medida del circuito general mediante optoacopladores, teniendo una tierra física diferente para cada bloque de medida. Esto tiene el problema de que necesitas un bloque por celda.

Para la medida de temperatura, el propio microcontrolador empleado en el bloque de adquisición podría ser aprovechado empleando su sensor de temperatura, pero si este bloque de medida no está adherido a la celda que pretendemos medir, esta solución no es viable. Otra opción es optar por sensores digitales que finalmente tienen la curva más proporcional de entre todas las opciones de sensores que podemos elegir en el mercado. Ello sumado a que la medida ya está calibrada hace que sean muy adecuados. En caso de no emplear esta solución, es muy importante aplicar coeficientes de corrección para evitar la ausencia de linealidad de los sensores analógicos convencionales, además de tener siempre presente el offset que estos pueden acarrear.

- **Detección de fuga**

Para esta función suele emplearse soluciones como los vigilantes de aislamiento comerciales que ya existen en el mercado. Es muy importante, ya que una derivación a tierra en un vehículo alimentado a una tensión nominal de entre 100 a 400V, con corrientes que alcanzan hasta 1000 A en algunos casos sería muy peligrosa.

La filosofía de estos sensores consiste en medir la resistencia entre la batería y el chasis. Si esta baja de 500Ω, el vigilante automáticamente actuará sobre el circuito de potencia abriéndolo, aislando las baterías del chasis.

- **Gestión de la temperatura**

Uno de los principales problemas que encontramos, sobre todo en vehículos eléctricos, es la refrigeración de los acumuladores. Una de las funciones que puede implementar nuestro BMS es gestionar cuándo se ha de activar la refrigeración (ventilador, refrigeración líquida…).

- **Bloque de señal**

Finalmente, la información recogida por el sistema BMS ha de ser transferida para que el usuario pueda interpretarla. Para este fin normalmente se conecta mediante Bus CAN el sistema BMS a la ECU del vehículo.

- **Conclusiones**

Hoy en día, en un mercado donde lo eléctrico se impone, es más que nunca fundamental desarrollar sistemas que controlen los parámetros que pueden hacer que nuestras baterías se degraden, y controlar el sistema de tal manera que el vehículo sea un entorno siempre seguro para el usuario. El mercado demanda diferentes soluciones para diferentes aplicaciones, de ahí la necesidad de desarrollar soluciones propias para cada tipo de sistema

5. Aplicación práctica

5.1. Solución adoptada

5.1.1. Aspectos generales

El proyecto consta de dos partes bien diferenciadas por la arquitectura que lo caracteriza. A fin de obtener una mayor robustez en el diseño, se opta por una solución descentralizada en la que cada microcontrolador individual realiza la monitorización de cada una de las 26 celdas que componen el sistema de alimentación del prototipo. Esta configuración además permite extrapolar la solución a prácticamente cualquier sistema de baterías con independencia de cuántos acumuladores disponga el mismo, simplemente cambiando el identificador de cada microcontrolador para el bloque de comunicaciones. Todas ellas se comunicarán mediante un bus de comunicaciones con un sistema maestro, que se encargará del control de la apertura y cierre del circuito de potencia, el almacenamiento y la comunicación con el usuario. Así pues, los materiales empleados para las placas esclavas y maestra quedan definidos en el **Anexo 1 y 2** respectivamente, correspondiente a los “Bill of Materials (BOM)” de ambos diseños.

5.1.2. Requerimientos normativos

Inicialmente se pensó en ubicar cada uno de las placas esclavas adheridas al lateral de cada celda que iba a medir. Sin embargo, las restricciones que impone el reglamento de la organización MotoStudent al diseño de la motocicleta implicaban una limitación al ancho del prototipo, lo que se traducía en una limitación en el diseño del cajón de baterías, y por ende, del chasis.

Estos motivos llevaron a la fabricación de la solución final, unas placas que aúnan la medición de seis celdas de forma simultánea e independiente, manteniendo el diseño original del bus de comunicaciones.

5.1.3. Software y herramientas empleadas

El diseño de las PCB ha sido diseñado mediante el software Eagle, y la programación de los microcontroladores se ha realizado mediante el entorno MPLAB, con el compilador XC8. Para la configuración de registros y módulos externos del microcontrolador se han utilizado las librerías de MCC para mayor facilidad de interpretación de la programación. La fabricación de los prototipos inicial se realizó a una cara mediante ataque químico. Todos los componentes empleados en este proyecto son de tecnología SMD, tal y como se detallará posteriormente.

5.1.4. Mejoras

La placa maestra del proyecto cuenta con un módulo Bluetooth RN4020, con el objetivo de comunicarse vía este protocolo con una aplicación de PC o móvil, para poder obtener los datos de tensión y temperatura guardados durante la actividad del prototipo. Para tal fin se desarrolló una GUI en MATLAB, que aprovecha la posibilidad de Windows de emular un puerto serie para un periférico bluetooth y compara los datos, muestra la curva obtenida en el dominio del tiempo y guarda en formato “.csv” para su posterior análisis.

5.2. Sistema Esclavo

Tal y como hemos expuesto de forma genérica, la solución adoptada pasa por el diseño de un sistema descentralizado compuesto por tantos microcontroladores como celdas vayan a muestrearse, todos ellos gobernados por un sistema maestro

que se encargará de actuar sobre el circuito en caso de ser necesario. A continuación se expondrán los elementos que conforman el sistema esclavo, y se desglosará su funcionamiento. Para una mayor profundidad de análisis acerca de los cálculos empleados para el diseño de la solución consúltese el Anexo de cálculos. Así mismo, en este apartado se expondrá qué realiza el microcontrolador, Pero para analizar en profundidad su programación, remítase el lector al Anexo 1.

Tal y como se muestra en la figura 1, se necesita que este sistema realice la adquisición de la tensión de las celdas, realice un pequeño procesamiento de los datos recibidos y los comunique al sistema maestro.

Figura 1: Diagrama de bloques del Sistema esclavo

5.2.1. Bloque de adquisición

En el bloque de adquisición, se debe realizar un acondicionamiento de la señal, adaptando su amplitud a valores admisibles para el módulo ADC de nuestro controlador. Además se dotará al sistema de un circuito de retención y muestreo como el expuesto en la figura 2 para favorecer la correcta lectura.

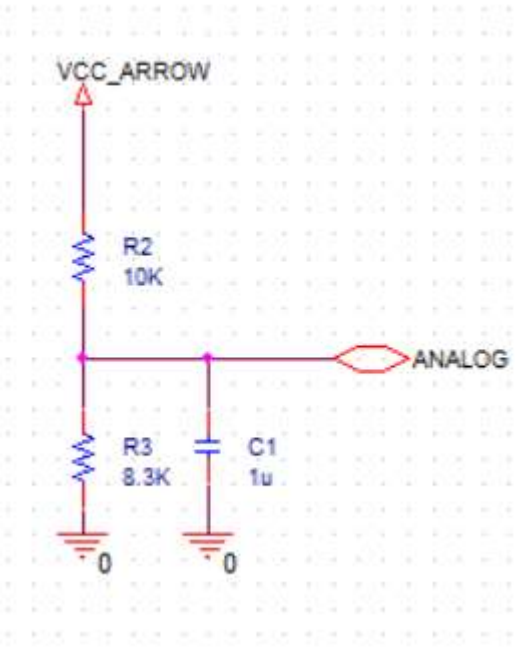


Figura 1: Acondicionamiento de la señal para su lectura

Se realiza una adaptación con un factor de atenuación a la señal de 19/42 mediante el uso de un divisor de tensión. Además se emplea un circuito RC que nos da una constante de tiempo de ζ=10ms.

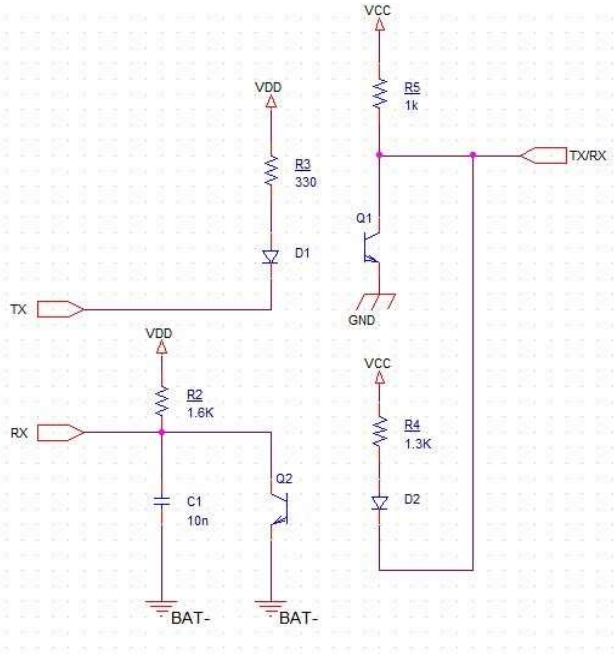
La necesidad de atenuar la señal radica en el inconveniente de que cada sistema se encuentra alimentado por la misma tensión que debe medir. Ello implica que se debe establecer una tensión de referencia, que además debe quedar por debajo de la mínima tensión que pueda alcanzar cada celda durante el tiempo de operación. Para tal fin se emplea el módulo FVR (Fixed Voltage Reference) del microcontrolador. Mediante este módulo se fija la tensión de referencia del módulo ADC en 2,048V.

5.2.2. Bloque de comunicación

Figura 3: Diagrama de bloques sobre el bloque de comunicaciones

Además de realizar la adquisición, el sistema esclavo debe ser capaz de transmitir la información obtenida en tiempo real al controlador maestro para que éste gestione la apertura del circuito en caso de ser necesario. Para ello se emplea comunicación mediante el módulo UART del controlador, pero con la peculiaridad de que en nuestra solución no se tratará de una conexión punto a punto entre dos dispositivos. En nuestro caso, tal y como se observa en la figura 3, contamos con un bus en el cual todos los sistemas esclavos permanecerán a la escucha, a la espera de ser llamados por el sistema maestro para establecer la comunicación y transmitir.

El principal problema que plantea esta configuración es que los esclavos se encuentran referenciados a un nivel de tensión diferente entre sí y respecto de la placa maestra. Ello nos obliga a diseñar un circuito que aísle galvánicamente la señal de comunicación respecto de los puntos de referencia de las placas esclavas. Para tal fin, se ha empleado el circuito expuesto en la figura 4:



(Figura 4: Esquema de comunicaciones en el bloque esclavo

Las corrientes del esquema son dimensionadas en mayor profundidad en el Anexo de cálculos. Para la transmisión, el fotodiodo se encuentra normalmente conduciendo, lo que establece que el receptor se encuentre en estado de saturación, enviando un estado bajo al bus. Para el caso de que el módulo EUSART del microcontrolador envíe un estado alto, se impedirá la conducción del fotodiodo, por lo que el receptor entrará en un estado de corte, transmitiendo un estado alto por el bus de comunicaciones. Para la recepción ocurrirá algo similar; el fotodiodo se encuentra en conducción para un estado bajo en el bus. Cuando se genere un estado alto en el bus, este fotodiodo dejará de conducir provocando un estado de corte en el receptor, lo que generará un estado alto en la patilla de recepción de nuestro módulo de comunicaciones. Nótese que este diseño permite al dispositivo “escuchar” cualquier señal que se envíe por el bus, incluida la suya propia. Ello es útil para obtener una retroalimentación y verificar que la señal es correcta. Se debe prestar especial

atención a la gestión de la comunicación, para no saturar el bus ya que sólo debe transmitir un dispositivo a la vez.

5.2.3. Listado de componentes

Una vez planteado el sistema, los diferentes bloques que lo conforman y la solución adoptada en cada caso, debemos justificar los materiales empleados para la fabricación de la solución hardware en cuestión. A continuación se detalla el BOM de nuestro sistema:

Qty	Value	Package	Parts
6	LED	CHIPLED_1206	LED1, LED2, LED3, LED4, LED5, LED6
6	1.6kΩ	R1206	R2, R10, R18, R26, R34, R42
6	100nF	C1206	C1, C5, C9, C13, C17, C21
6	100uH	L2012C	L1, L2, L3, L4, L5, L6
12	10kΩ	R1206	R1, R8, R9, R16, R17, R24, R25, R32, R33, R40, R41, R48
6	10nF	C1206	C4, C8, C12, C16, C20, C24
12	1kΩ	R1206	R4, R5, R12, R13, R20, R21, R28, R29, R36, R37, R44, R45
6	1nF	C1206	C2, C6, C10, C14, C18, C22
6	1uF	C1206	C3, C7, C11, C15, C19, C23
12	330Ω	R1206	R3, R7, R11, R15, R19, R23, R27, R31, R35, R39, R43, R47
6	50KΩ	R1206	R6, R14, R22, R30, R38, R46
6	CDSOD323-T15SC	SOD-323	D1, D2, D3, D4, D5, D6
6	MOCD217M3D	SOIC8	IC1, IC2, IC5, IC7, IC9, IC11
6	PIC16F18313-E_SN3D	SOIC8	IC3, IC4, IC6, IC8, IC10, IC12

Tabla 2: Listado de material para fabricación de PCB esclava

5.3. Sistema maestro

Los sistemas esclavos presentados anteriormente no son independientes, y han sido diseñados para trabajar gestionados por otro sistema, un maestro que realice las peticiones de envío de información en el bus y que gestione la misma, con el objetivo final de controlar el sistema de potencia en caso necesario. Además realizará la adquisición de los valores de temperatura de cada zona de nuestro sistema mediante tres sensores de temperatura. Por tanto, el sistema va a contar con varios bloques bien diferenciados, como se aprecia en la Figura 5:

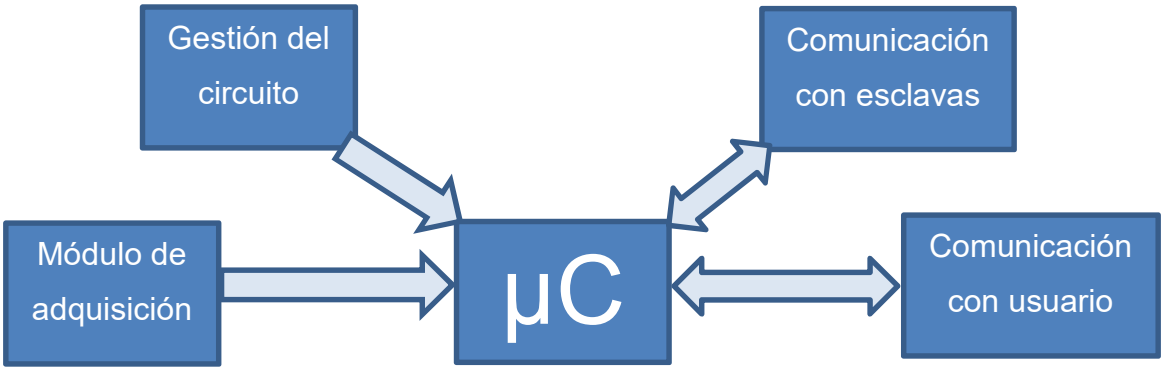


Figura 5: Diagrama de bloques del sistema maestro

5.3.1. Módulo de adquisición

Dado que los sensores se encuentran introducidos en el contenedor de los acumuladores, la señal que generan se encontrará sometida a posibles ruidos. Por ello, la señal es tratada mediante el circuito mostrado en la Figura 6. Dicho circuito se encarga de filtrar la señal y además realiza las veces de un circuito de retención y muestreo, facilitando la correcta adquisición.

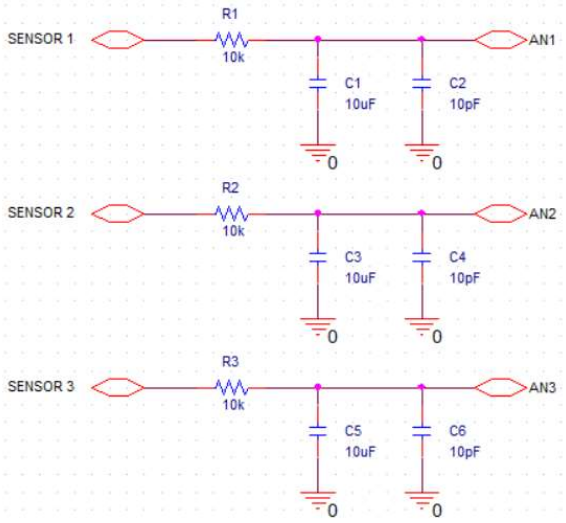


Figura 2: Circuito de filtrado, muestreo y retención

5.3.2. Gestión del circuito

El sistema maestro cuenta con una salida digital que se encuentra conectada a un relé tal como se muestra en la Figura 6:

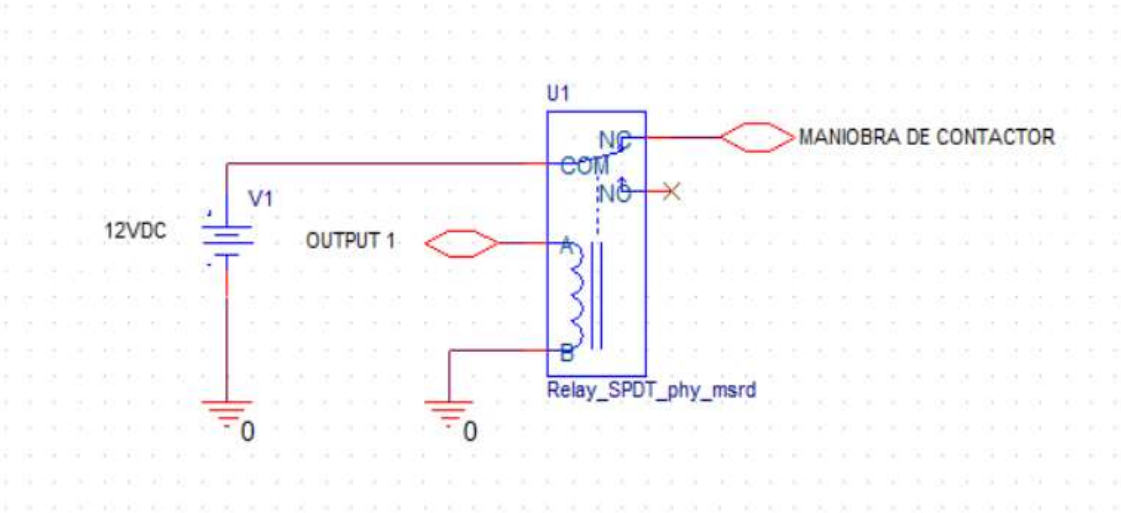


Figura 7: Esquema de mando circuito maestro

5.3.3. Comunicación con esclavas

El sistema debe suministrar potencia suficiente para que la señal enviada a las placas esclavas sea suficiente para que todos los dispositivos que permanecen a la escucha reciban la señal. Por ello se diseña una etapa de

amplificación no inversora para la transmisión, tal como se puede observar en la Figura 8:

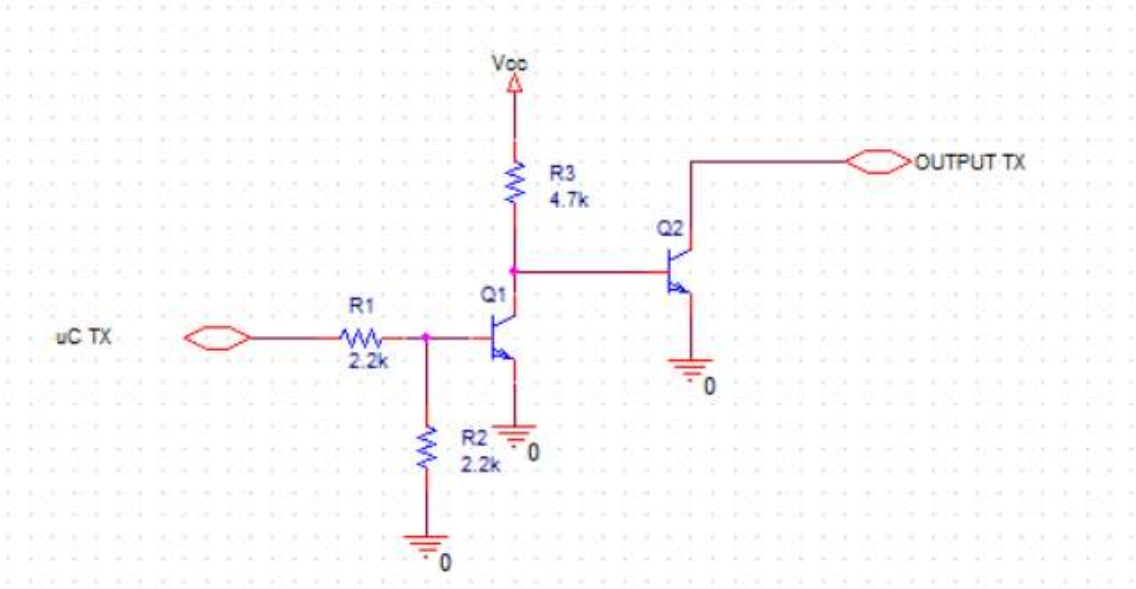


Figura 8: Esquema de transmisión en placa maestra

Para un análisis sobre el dimensionado de los componentes del sistema, por favor remítase al Anexo de Cálculos.

5.3.4. Comunicación con usuario

La comunicación del usuario, pese a no terminar siendo implantada completamente en la primera versión de este sistema, queda planteada para posibles mejoras del mismo. Para ello, se dota a la placa maestra de un módulo Bluetooth, que contará con la posibilidad de una configuración como puerto COM. Mediante un COM virtual enlazado al módulo bluetooth periférico de nuestro ordenador o teléfono, se enlaza una aplicación que gestione la consulta de datos y su procesamiento. A modo de ejemplo en el anexo de programación del sistema maestro se adjunta el código de una interfaz en Matlab, así como una breve explicación sobre su funcionamiento

5.4. Implantación física del sistema

El sistema según la normativa ha de ser estanco. Para conseguirlo, el sistema fue implantado en un cajetín comercial con IP67, con prensaestopas con el objetivo de aislar lo más posible el sistema. En cuanto a las placas, finalmente tienen un

diseño pensado para ser instaladas en estantería, ahorrando el máximo espacio posible. En cuanto a las conexiones con las baterías, con objeto de evitar cortocircuitos indeseados, se aislaron debidamente mediante material aislante termofusible.

6. Presupuesto

El presupuesto Total de Ejecución por Contrata de la implementación del proyecto es de **ONCE MIL TRESCIENTOS OCHENTA Y UN EUROS CON CINCUENTA Y CUATRO CÉNTIMOS.**

Presupuesto total de ejecución por contrata: **11.381,54€**

El presupuesto se encuentra desglosado en el apartado Presupuesto.

7. Condiciones de ejecución

En el Pliego de Condiciones quedan recogidas las condiciones de ejecución del proyecto, así como la normativa necesaria para la misma. También se exponen las prescripciones técnicas y otros aspectos concretos a tener en cuenta en este proyecto.

Grado en Ingeniería Electrónica Industrial

Escuela Politécnica Superior de Jaén

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica

CÁLCULOS

Alumno: Juan Caño Navas

Tutor: Prof. D. Miguel de la Fuente Ruz

Dpto: Ingeniería de Sistemas y Automática

Diciembre, 2018

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE ECUACIONES

1. Introducción

El objeto del presente documento es la justificación mediante cálculos de la solución alcanzada en el diseño final. A continuación se especifican las siguientes constantes con las cuales se trabajará a lo largo de todo el documento:

- $V_{dd1(max)}$=4.2V. Tensión de alimentación máxima del microcontrolador.
- $V_{dd2(min)}$=3.7V. Tensión de alimentación mínima del microcontrolador.
- V_{cc}= 5V. Tensión auxiliar.
- V_{fvr1}=2.048V. Tensión de referencia positiva analógica. Provista por el módulo FVR del microcontrolador.
- V_{led}=2.2V
- $V_{f(emisor)}$=1.05V Tensión consumida en activa-directa por el emisor del optoacoplador.
- $V_{ce(sat)(emisor)}$=0.35V Tensión consumida en saturación por el receptor del optoacoplador.
- V_{be}=1V Tensión base-emisor del transistor de la etapa de potencia (máster)
- β_{min}=63 Ganancia del transistor
-

2. Cálculos referentes a la placa esclava

2.1. Cálculos sobre dimensionamiento de hardware

Dentro de la placa, debemos tratar diferentes puntos sensibles al cálculo: alimentación del sistema, adquisición, actuación y comunicación.

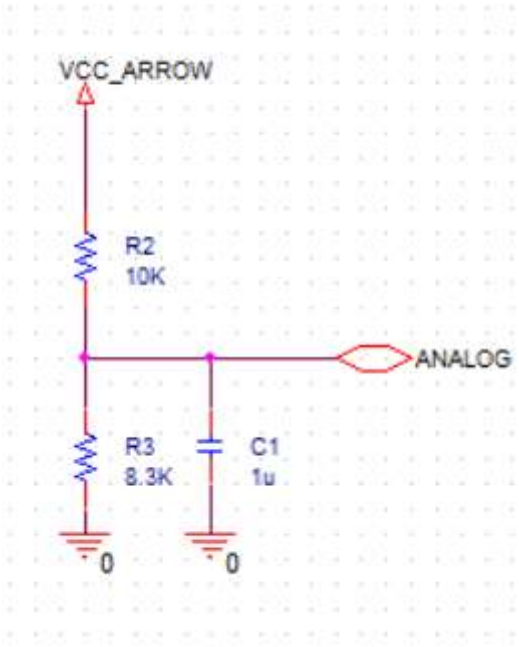
- **Alimentación del sistema**

Para la alimentación del sistema debemos garantizar una entrada de tensión estable, que rechace picos y resista ruidos e interferencias tanto a alta como a baja frecuencia. Los condensadores idealmente se comportan como un circuito abierto frente a corriente continua, mientras que su resistencia disminuye a medida que la frecuencia aumenta. Si incluimos condensadores en paralelo a una fuente de continua con un ruido en alterna, estos condensadores rechazarán la componente continua de la señal, y absorberán la parte alterna. Típicamente se han colocado los condensadores con unos valores de 10nF y 100nF para el desacoplo, y por lo tanto estos valores han sido escogidos en nuestro diseño.

En cuanto a la bobina empleada a la entrada, se emplea para evitar los picos de descarga iniciales que aparecen al cerrar un circuito alimentado mediante baterías.

El diodo TVS empleado en este circuito debe soportar picos de tensión de al menos la mitad más de la tensión de alimentación.

- **Adquisición**



(Anexo 1) Figura 1: Circuito de adquisición empleado para la tensión

Se emplea para la adquisición un circuito de muestreo y retención, formado por una resistencia y un condensador. Además, para acondicionar la amplitud de la señal se añade una resistencia adicional con el fin de realizar un divisor de tensión a la entrada de la analógica. Para el esquema 1 tenemos que se cumple lo siguiente:

$$\zeta = R_1 \cdot C_1$$

Ecuación 1

Fijamos el valor del condensador en un valor comercial, 1µF, y el valor de R_1 ha de ser tal que el adc tenga tiempo de tomar hasta cuatro muestras durante el tiempo de retención. El periférico está configurado para muestrear cada 46µs. Fijamos un tiempo de retención de al menos 10 veces más que el tiempo de muestreo, en este caso se ha fijado un valor de 10ms. Por lo tanto:

$$R_1 = \frac{\zeta}{C_1} = 10k\Omega$$

Ecuación 2

A continuación, debemos adaptar la amplitud de la señal para que no sature la analógica. La tensión de referencia para dicho módulo viene dada por el módulo FVR que incorpora el propio microcontrolador. Dicho módulo ofrece tres posibles referencias: 1.024V, 2.048V o 4.096V. Ante la posibilidad de que la alimentación llegue a ser inferior de 4.096V debido a la descarga de la celda, se emplea una referencia de 2.048V. por lo tanto, la tensión de entrada de la analógica no debe superar este valor. Para acondicionar la señal realizamos los siguientes cálculos, tomando para ellos la consideración de que el condensador se encuentra cargado:

$$V_{dd} = I \cdot (R_1 + R_2) \rightarrow I = \frac{V_{dd}}{R_1 + R_2}$$
$$V_{analog} = I \cdot R_2 = \frac{Vdd}{R_1 + R_2} \cdot R_2$$
$$Si\ R_2 = 8{,}3K \rightarrow V_{analog} = 1{,}9V$$

Ecuación 3

Se puede comprobar por tanto que para los valores de resistencia escogidos, la tensión no saturará el canal ADC. Para el supuesto inicial de que el condensador estuviera completamente descargado, perderíamos la primera muestra pero en cualquier caso la tensión suministrada no dañaría el canal ADC. Esto sólo ocurrirá en la primera medida que realice el canal al conectar la batería a la PCB.

- **Actuación**

En el caso de las placas esclavas, éstas no tendrán ningún poder de mando sobre el sistema, ya que el que debe tomar la determinación de abrir el sistema es la paca maestra. Por lo tanto, las únicas actuaciones que tiene esta placa es la de mostrar información visual mediante un LED y una señal digital. Para ello, se debe dimensionar el valor de corriente del LED para que trabaje en torno a los 10mA o menos. La señal digital proveerá una tensión igual a V_{dd}.

$$V_{dd} = I \cdot R + V_{led} \rightarrow I = \frac{V_{dd} - V_{led}}{R}$$
$$Si\ R = 330\Omega \rightarrow I = 6.36mA$$

Ecuación 4

Para el valor de resistencia propuesto, la salida digital nos proporcionará una corriente de 6.36mA, suficiente para nuestro propósito.

- **Comunicación**

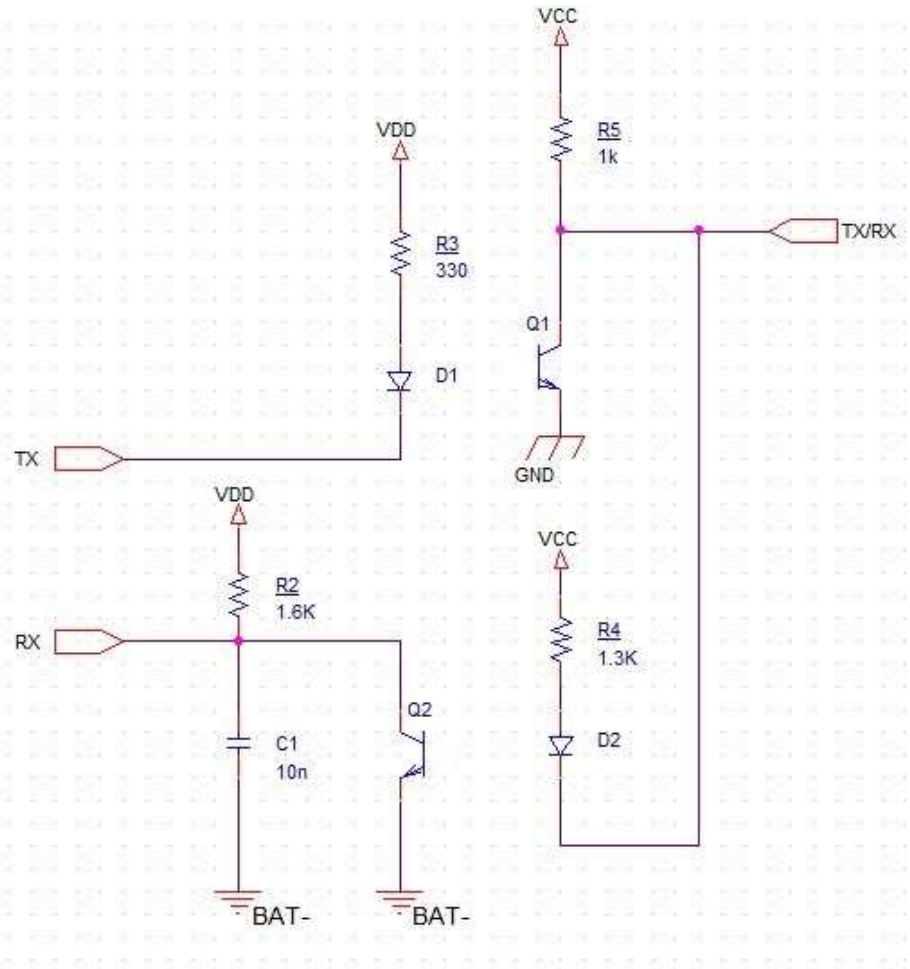


Figura 2: Diseño para aislamiento galvánico entre las señales generadas por el microcontrolador y la señal que procesa el microcontrolador maestro.

Para la comunicación, básicamente encontramos dos partes, transmisión y recepción. En ambas debemos limitar la corriente que circula por los optoaclopadores, de tal forma que los dispositivos no resulten dañados y el consumo de potencia no sea excesivo, con el objetivo de que el circuito de amplificación situado en el microcontrolador maestro disponga de potencia suficiente para comunicarse con todos. La naturaleza de la communicación de este bus ha sido descrita previamente en el Anexo 1, por lo que en este documento nos limitaremos a exponer el dimensionado de sus componentes.

$$V_{dd} = I \cdot R_3 + V_{f(emisor)}$$
$$Si\ R_3 = 330 \rightarrow I \cong 8{,}8mA$$

$$I_{\max(emisor)} = 60mA; I_{typ(emisor)} = 10mA$$

Ecuación 5

Como se puede comprobar, estamos dentro de los márgenes de funcionamiento del dispositivo. Para el caso de que la transmisión se inicie, el optoacoplador Q1 entrará en corte, por lo tanto, la corriente de transmisión será:

$$V_{cc} = I \cdot R_1 \rightarrow I = \frac{V_{cc}}{R_5}$$
$$Si\ R_5 = 1k\Omega \rightarrow I = 5mA$$

Ecuación 6

El valor de I en este caso se corresponde con la corriente de transmisión que tendría nuestra señal.

En la recepción la filosofía es similar, solo que además agregamos un circuito de muestreo y retención a la señal recibida.

$$V_{dd} = R_2 \cdot C_1 \rightarrow Para\ R_2 = 1.6k\Omega, C1 = 10nF$$
$$\zeta = 16\mu$$
$$Cuando\ Q_2 \rightarrow saturado, I = \frac{V_{dd} - V_{ce(sat)(receptor)}}{R_2} = 2.9mA$$

Finalmente, para el emisor D_2:

$$V_{cc} = I \cdot R_4 + V_{f(emisor)} \rightarrow I = \frac{V_{cc} - V_{f(emisor)}}{R_4} \cong 3mA$$

Ecuación 7

2.2. Cálculos referentes a la programación

En este tipo de aplicaciones suele ser habitual realizar una conversión desde el valor en bits del ADC a tensión, para una mejor comprensión del programa. Sin embargo en este caso, era más útil mandar este dato por el bus de comunicaciones sin modificar, ya que ello nos permite trabajar en variables enteras sin perder precisión. Por lo tanto, el único tratamiento q se le realiza a la lectura es el de un filtro de media móvil. Este filtro troma cuatro valores muestreados y los filtra siguiendo la siguiente ecuación:

$$filtrado = (0.45 * lectura_4 + 0.35 * lectura_3 + 0.15 * lectura_2 + 0.05 * lectura_1);$$

Ecuación 8

3. Cálculos referentes a la placa maestra

3.1. Cálculos sobre dimensionamiento de hardware

En este caso nos encontramos con una placa que precisa de una adaptación a la entrada, de una adaptación de señal analógica, un circuito de comunicación y dos señales digitales. La UART que se comunica con el módulo bluetooth, así como las señales digitales que necesita, no precisan de adaptación hardware alguna, por lo tanto se obviarán en este documento.

- **Alimentación**

Para el caso de esta placa, se ha provisto de una fuente integrada que convierte la tensión de entrada a 5Vdc. Sin embargo, esta fuente puede no limpiar la señal de ruido, por lo tanto se incluyen los condensadores de desacoplo al igual que en el anterior diseño.

- **Adquisición**

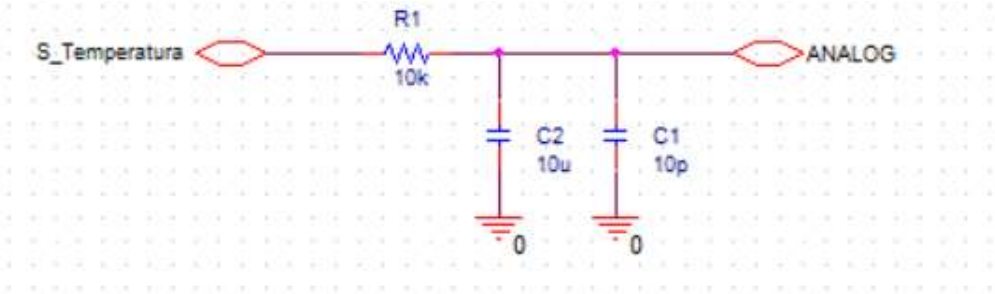


Figura 3: Circuito para la adquisición de la señal analógica del sensor de temperatura

La señal del sensor de temperatura tiene mucho riesgo de recibir ruidos electromagnéticos del cajón de baterías, por estar muy cerca. Para evitar estos ruidos, se realiza un filtrado de pequeñas y altas frecuencias, incorporando para ello un condensador de un valor muy bajo y otro de un valor muy alto.

$$f_{corte} = \frac{1}{RC}$$

Ecuación 9

- **Comunicación**

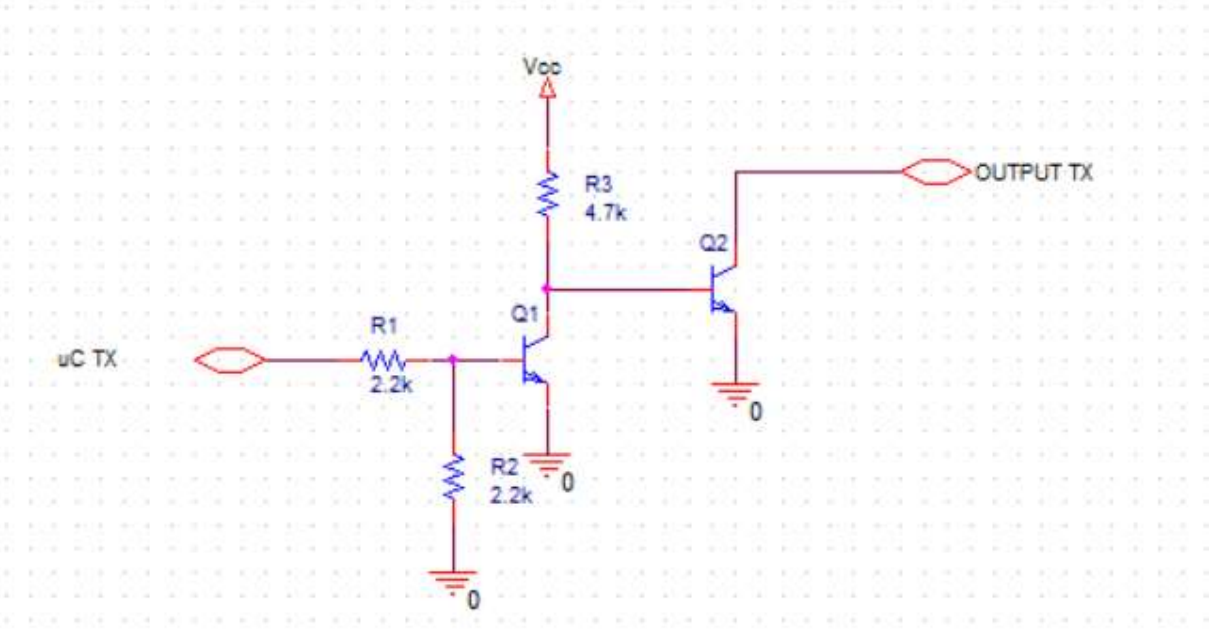


Figura 4: Esquema de amplificación de la salida de transmisión del microcontrolador maestro hacia el bus de comunicaciones

Para la transmisión podemos observar la figura mostrada arriba. Pasaremos la señal por una etapa de amplificación para posteriormente invertirla, para no alterar su polaridad con la primera etapa.

$$V_{uC\,TX} = I_1 \cdot R_1 + V_{be}$$

$$I_1 = \frac{V_{uC\,TX} - V_{be}}{R_1} = 1.81mA$$

$$I_1 = I_2 + I_b \rightarrow I_b = I_1 - I_2 = 4.35mA$$

$$I_2 = \frac{V_{be}}{R_2} = 454.54\mu A$$

$$V_{cc} = I_c \cdot R_3 + V_{ce}$$

$$I_c = \beta \cdot I_b = 274.4mA$$

Ecuación 10

En cuanto a la recepción, no tiene ninguna adaptación física remarcable

- **Señales digitales**

Existe en este diseño una señal digital con un led completamente igual al anterior diseño, y además la señal del relé está conectada directamente a una salida digital.

3.2. Cálculos referentes a la programación

En este microcontrolador tampoco se realizan operaciones para transformar los valores recibidos, ya que esto sólo es interesante para la depuración, sin embargo si conocemos los valores límite no es necesario q la programación realice estas conversiones, perdiendo tiempo y memoria.

También se realizará el mismo filtrado explicado en el apartado del microcontrolador esclavo, a fin de filtrar valores anómalos en la ADC de los sensores de temperatura.

4. Conclusiones

Para esta aplicación, las premisas que más han de ocupar el estudio y diseño de la parte hardware, son la adquisición, debiendo filtrar de manera conveniente todo lo que nuestro sistema reciba, pues se encuentra en un entorno electromagnético muy hostil, y la comunicación, teniendo en cuenta el coste energético de cada sistema que enlazamos al bus.

Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén
Grado en Ingeniería Electrónica

PLANOS

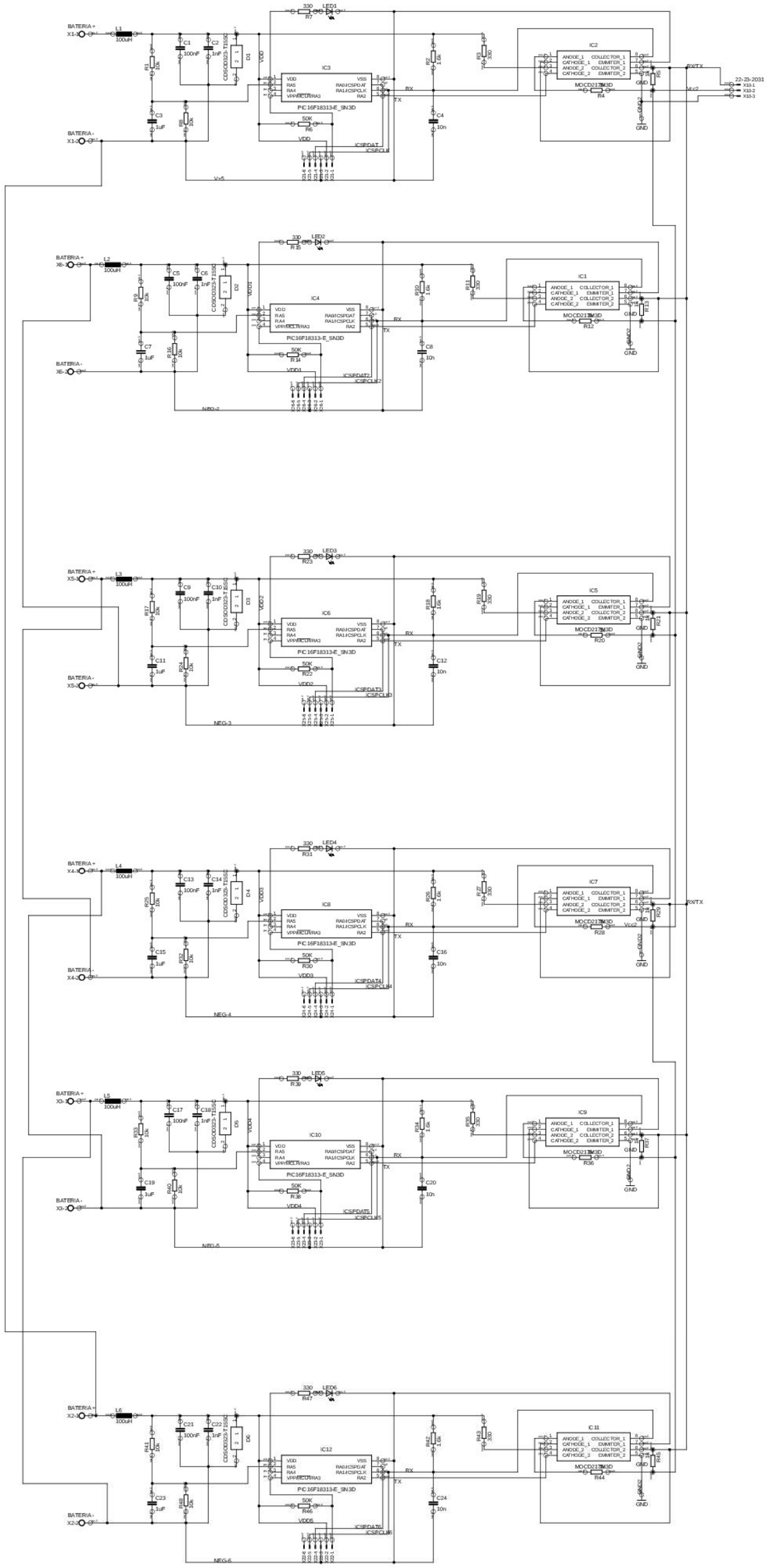
Alumno: Juan Caño Navas

Tutor: Prof. D. Miguel de la Fuente Ruz

Dpto: Ingeniería de Sistemas y Automática

Diciembre, 2018

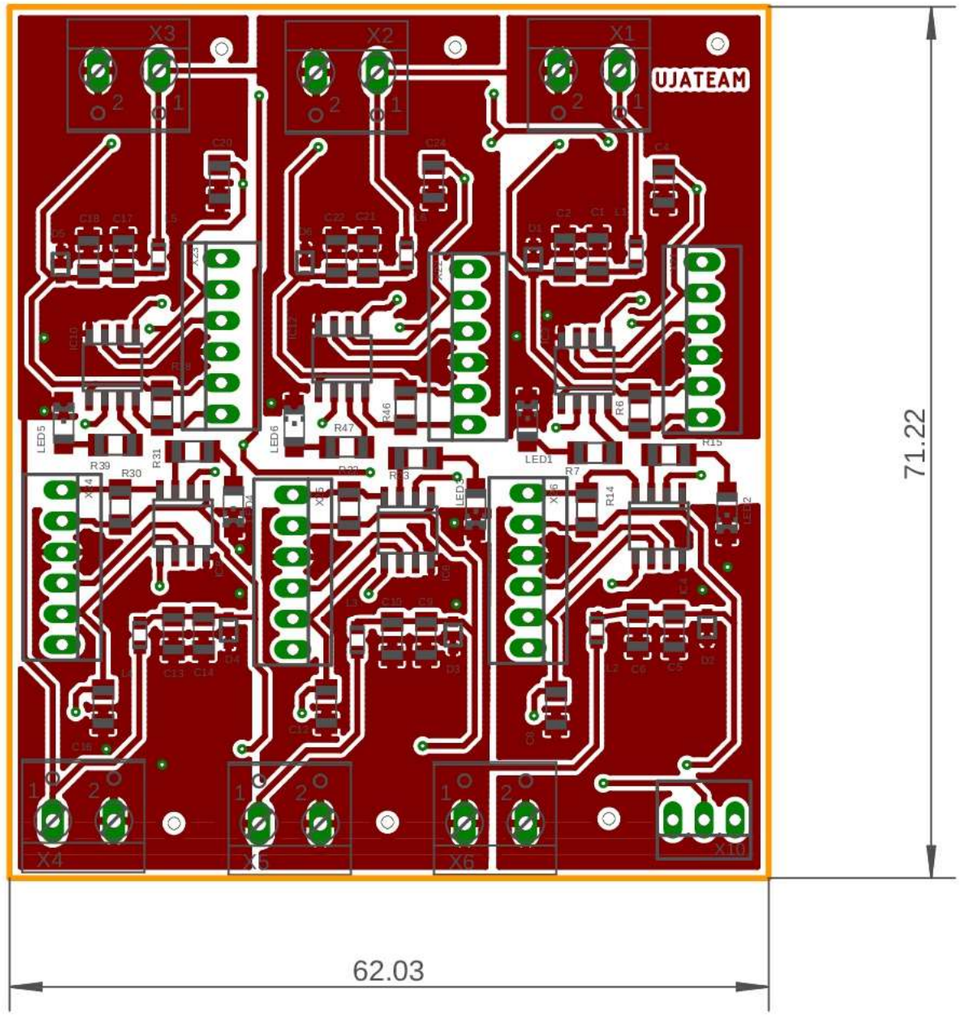
INDICE



	Fecha	Nombre	Firmas
Dibujado	05/09/2018	JCN	CAÑO NAVAS JUAN - 26253397R Firmado digitalmente por CAÑO NAVAS JUAN - 26253397R Fecha: 2018.10.18 21:52:09 +02'00'
Comprobado	05/09/2018	M. De la Fuente	

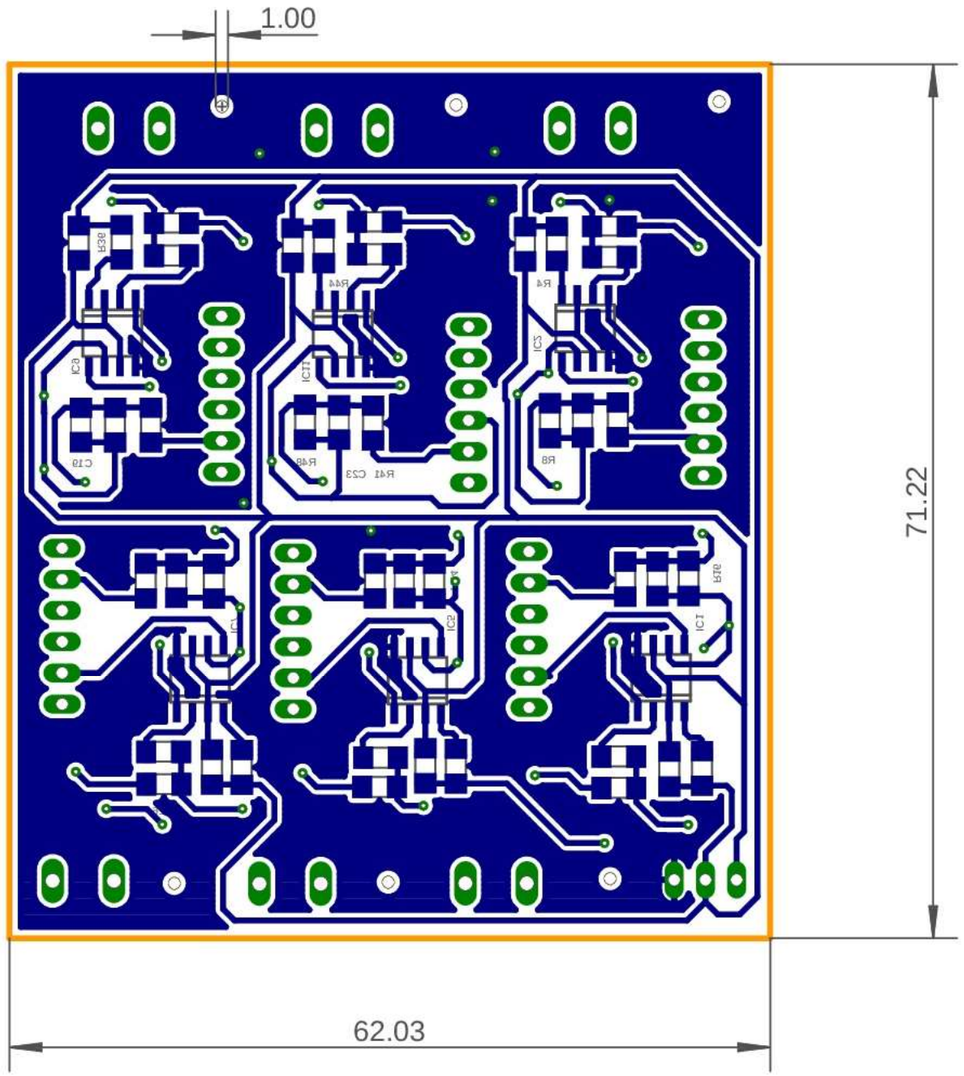
EPSJ Escuela Politécnica Superior de Jaén

Escala S/E	Esquemático PCB ESCLAVA	Número 1/6
		Sustituye a
		Sustituido por

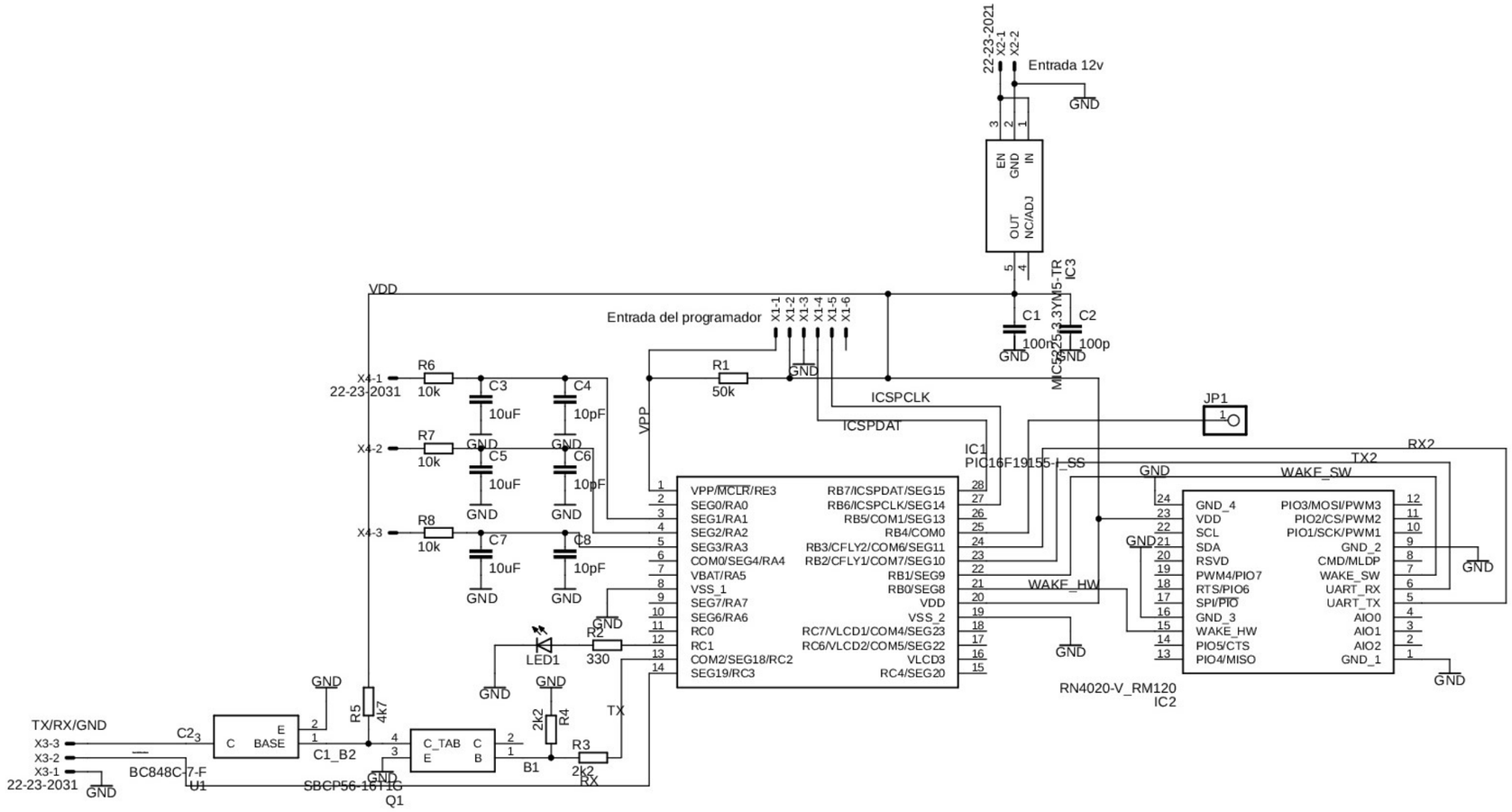


	Fecha	Nombre	Firmas	
Dibujado	05/09/2018	JCN		EPSJ Escuela Politécnica Superior de Jaén
Comprobado	05/09/2018	M. de la Fuente		

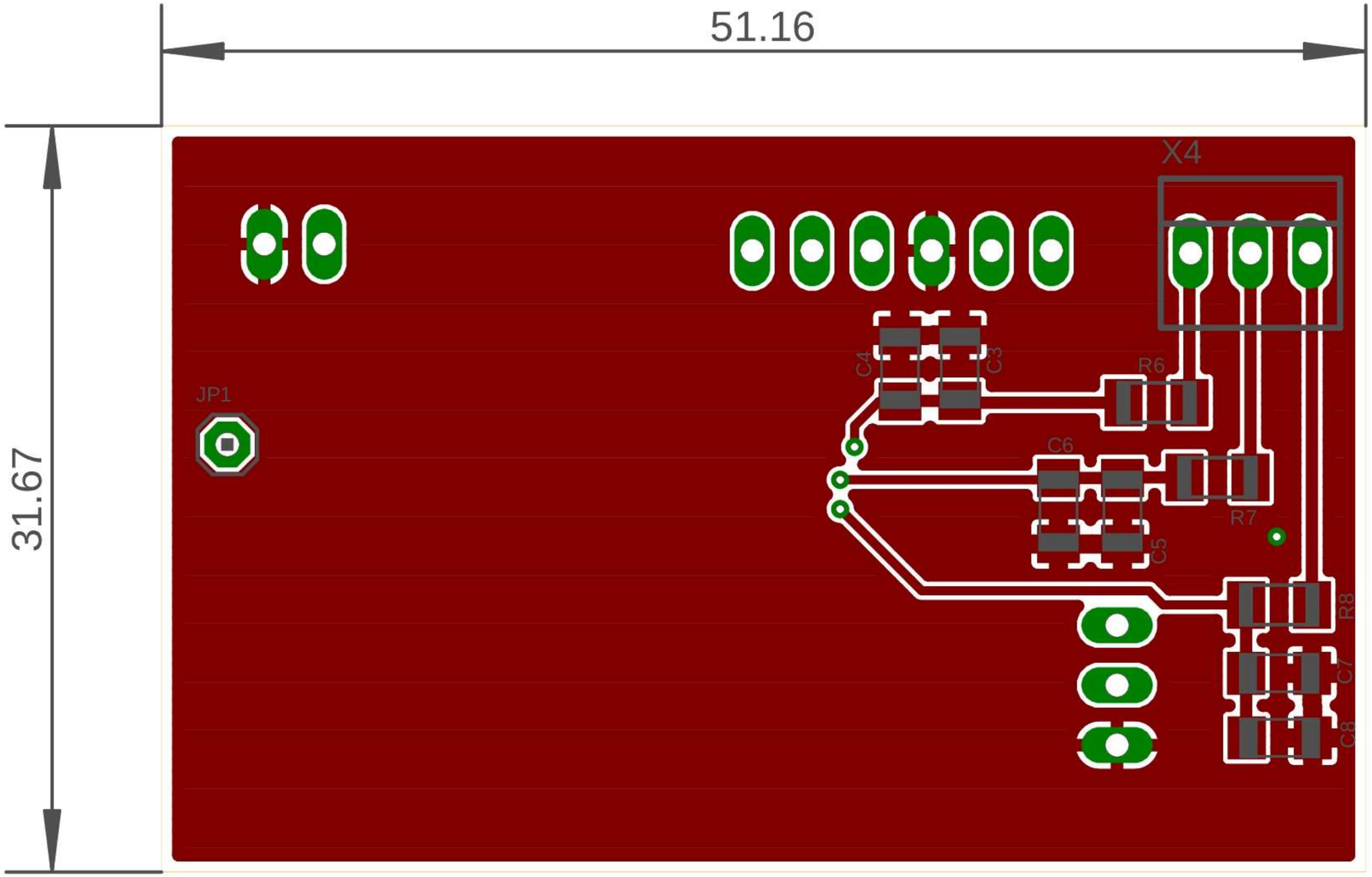
Escala 2:1	Capa top placa Esclava	Número 2/6
		Sustituye a
		Sustituido por



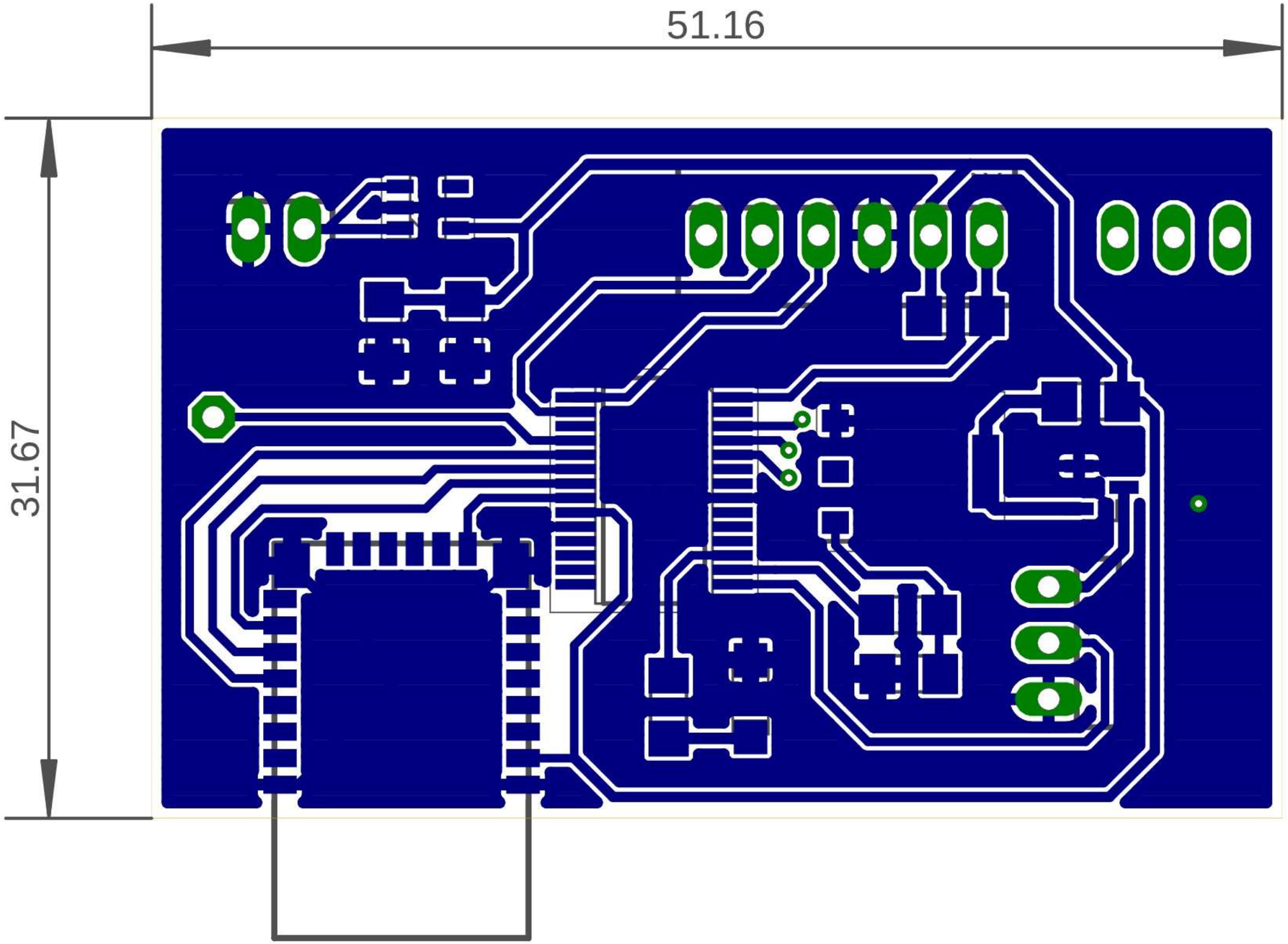
	Fecha	Nombre	Firmas	
Dibujado	05/09/2018	JCN		EPSj Escuela Politécnica Superior de Jaén
Comprobado	05/09/2018	M. de la Fuente		
Escala 2:1	Capa bottom placa Esclava			Número 3/6
				Sustituye a
				Sustituido por



	Fecha	Nombre	Firmas	
Dibujado	10/09/2018	JCN		EPSJ Escuela Politécnica Superior de Jaén
Comprobado	10/09/2018	M. De la Fuente		
Escala S/E	PLACA MAESTRA			Número 4/6
				Sustituye a
				Sustituido por



	Fecha	Nombre	Firmas	
Dibujado	10/09/2018	JCN		EPSJ Escuela Politécnica Superior de Jaén
Comprobado	10/09/2018	M. de la Fuente		
Escala 5:1	Capa top placa maestra			Número 5/6
				Sustituye a
				Sustituido por



	Fecha	Nombre	Firmas	
Dibujado	10/09/2018	JCN		EPSJ Escuela Politécnica Superior de Jaén
Comprobado	10/09/2018	M. de la Fuente		

Escala		
5:1	Capa bottom placa maestra	Número 6/6
		Sustituye a
		Sustituido por

Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén
Grado en Ingeniería Electrónica

PLIEGO DE CONDICIONES TÉCNICAS

Alumno: Juan Caño Navas

Tutor: Prof. D. Miguel de la Fuente Ruz

Dpto: Ingeniería de Sistemas y Automática

Diciembre, 2018

ÍNDICE

1. Objetivo del Pliego

El presente documento trata sobre las prescripciones de todo tipo (generales, particulares, de obligado cumplimiento, económicas, técnicas, facultativas y legales) que serán vinculantes en este proyecto durante la ejecución del mismo.

2. Normativa de obligado cumplimiento

La dirección de ejecución debe tener en cuenta tanto en los equipos como en el montaje, la normativa respectiva que afecte a la naturaleza del trabajo realizado. En este proyecto las más significativas son:

- Reglamento de Seguridad e Higiene en el Trabajo.
- Reglamento Electrotécnico de Baja Tensión (REBT).
- Normativa DIN 43.539; CEI/IEC 95-1, sobre la normativa a seguir en mediciones y métodos de carga.
- Normas de Compatibilidad Electromagnética. UNE-EN 6100.
- Norma ANSI-IPC 2221 sobre el diseño de circuitos impresos.

También será de obligado cumplimiento la normativa medioambiental sobre equipos electrónicos, a tener en cuenta:

- Real Decreto 208/2005, del 25 de Febrero.
- Directiva 2002/96/CE, Residuos de Aparatos Eléctricos y Electrónicos.

Será responsabilidad del contratista cumplir cualquier ordenanza social o de seguridad que esté estipulada, así como obtener permisos y licencias pertinentes a las autoridades, renunciando a exigir indemnizaciones por este concepto.

La gestión de la responsabilidad laboral sobre la seguridad, accidentes y derechos recaerá sobre el contratista en todo momento, y por tanto la responsabilidad ante accidentes y los derechos que ellos puedan acompañar también recaerán sobre el contratista.

Será responsabilidad completa del Director de Ejecución el revisar el proyecto antes de comenzar con la ejecución y denunciar en caso de que fuera pertinente a la Dirección y Propietario, de cualquier elemento que estuviese en contra de la correspondiente reglamentación.

3. Prescripciones económicas

3.1. Contrato

En caso de existir dudas entre las partes sobre la prioridad de las condiciones del contrato, se reserva el derecho de determinar la misma a La Dirección Técnica.

Téngase en cuenta que una reglamentación oficial tendrá siempre prioridad frente a cualquier prescripción del presente Pliego.

3.2. Abonos y precios

El abono total del proyecto, así como sus liquidaciones parciales, quedarán fijadas previamente en contrato, y deberán ser verificadas por la administración en el plazo y forma establecidos, y deberán haber sido previamente certificadas por la Dirección Técnica.

Los precios unitarios que se muestran en el proyecto tendrán carácter contractual, siendo los mismos los que presentará el contratista durante la formalización del proyecto.

3.3. Rescisión

En caso de que el contratista no reuniera los requisitos exigidos, La Dirección Técnica tendrá el derecho de rescindir el contrato

Si este caso ocurriera, el contratista no percibirá liquidación alguna por los acopios que hubiese realizado, y La Dirección Técnica fijará plazos para las actuaciones pertinentes, de tal manera que la ejecución no se vea afectada por este motivo.

4. Prescripciones Facultativas

El Director de Ejecución deberá manifestar expresamente que el proyecto ha sido debidamente estudiado, y que está de acuerdo a lo planteado en el mismo, entendiendo así que es acorde a la normativa vigente. Así mismo, no estará habilitado a introducir modificaciones de ningún tipo al proyecto, ciñéndose por tanto a las prescripciones y normas de La Dirección Técnica.

En el supuesto de encontrar algún error durante la ejecución del proyecto, estará obligado a comunicar dicha incidencia a La Dirección Técnica.

Será responsable en todo caso de realizar la instalación siguiendo la legislación vigente, así como de la confección, gestión y entrega de la documentación necesaria, así como de la obtención de las aprobaciones de dicha documentación ante los organismos oficiales.

También será responsable de efectuar las pruebas exigidas por la legislación y por el presente documento, y además deberá efectuar aquellas que La Dirección Técnica estimara oportunas, asumiendo los costes de dichas pruebas.

Será también su cometido asegurar que se cumple la garantía especificada en el presente proyecto, realizando las reparaciones o sustituciones en el plazo mínimo posible.

Finalmente, debe ser responsable del correcto uso del proyecto, de no divulgarlo sin autorización previa, respetando así la propiedad intelectual.

5. Pliego de Condiciones Técnicas

5.1. Estado de los materiales

Los materiales empleados en el presente proyecto deberán cumplir las prescripciones determinadas por los diferentes documentos del proyecto. Téngase en cuenta que los componentes además de cumplir las prescripciones descritas, deben ajustarse a los footprints del diseño, por lo tanto deben ceñirse a lo mostrado en los planos y anexos correspondientes. Pasamos a enumerar las prescripciones:

a. Diodo TVS: ha de ser bipolar, que soporte una tensión de pico de al menos 12.0V, y un pico de potencia de hasta 500W durante un período no superior a 8µs. Se empleará para este dispositivo un encapsulado SOD-323.

b. Sensor de Temperatura: ha de ser un sensor lineal, que devuelva una tensión proporcional a la temperatura medida, con un error no superior a ±0.5ºC y cuya tensión generada a 60ºC no supere los 2.048V. De otra forma, este sensor saturaría el canal ADC del microcontrolador. Este sensor se empleará a modo de sonda, se aconseja un encapsulado TO-92 o TO-220.

c. Fuente integrada: Debe poder trabajar de forma continua a 12VDC, y su salida debe ser de 5VDC. Debe garantizar una corriente estable de al menos

150mA en la salida, y debe soportar picos mayores de 12VDC. Se montará para este dispositivo un encapsulado SOT23 de 5 pines.

d. Optoaclopadores: Deben poder trabajar con las corrientes calculadas en el **Anexo 1**, y téngase en cuenta que el integrado debe constar de dos optoacopladores para la correcta instalación. El emisor no debe tener consumir tensiones mayores a 1.3V durante su período activo, y el detector tendrá una tensión inferior a 0.4V durante la saturación entre el colector y el emisor. Así mismo, los tiempos de ON, OFF, de subida y caída de la señal en estos dispositivos no superarán en ningún caso los 10µs. su encapsulado será un SOIC de 8 patillas

e. Microcontrolador Esclavo: Debe ser un dispositivo de 8bits, con 8 patillas y un conversor ADC, así como un módulo FVR, al menos un pin digital y una UART. Sus patillas han de ser de propósito general para configurarlas libremente, y su alimentación debe soportar ser alimentado a 5VDC. Su encapsulado será un SOIC.

f. Microcontrolador Maestro: Debe ser un dispositivo de 8 bits, con un módulo ADC, FVR y al menos dos UART. Además contará con memoria EEPROM, al menos dos pines digitales y una alimentación de 5VDC. Será adquirido en un encapsulado SSOP-28. Las patillas han de ser de propósito general también.

g. Módulo bluetooth: Debe poder ser configurado mediante interface API vía puerto serie, contar con antena propia, y trabajar a 5VDC. Además debe ser configurable para emular el comportamiento de un canal puerto serie. El footprint debe adaptarse al descrito en el Plano 6 del presente proyecto, pero está sujeto a modificaciones siempre que no se comprometa el resto del diseño.

h. Transistor de potencia: Se corresponde con el transistor Q1 del Anexo 2. Debe soportar un consumo de potencia de al menos 0.65W, por lo que se elegirá un encapsulado SOT223, y se prestará especial atención a la unión de la patilla 4, de mayor superficie, que deberá estar completamente en contacto con su pad, con el fin de disipar la mayor cantidad de calor posible. Debe ser un transistor NPN

i. Se corresponde con el elemento U1 del Anexo 2. Debe ser un transistor NPN, con un encapsulado SOT23.

j. Todas las resistencias montarán un encapsulado 1206.

k. El inductor montará un encapsulado 0806.

l. Los condensadores montarán un encapsulado 1206.

m. La placa fenólica tendrá un espesor de 1.6mm, de doble cara. Las dimensiones del enrutado deberán ser respetadas tal y como se adjuntan en los Planos 2, 3, 5 y 6.

n. Las conducciones deberán ir apantalladas con una malla conectada a masa siempre y cuando tengan cercanía con las conducciones de potencia, pero siempre deberá evitarse esta circunstancia.

5.2. Ejecución del proyecto

Para la ejecución del proyecto, lo fundamental será realizar una correcta verificación posterior al montaje de las placas, comprobando continuidad y realizando una inspección visual exhaustiva.

5.3. Disposiciones finales

5.3.1. Montaje y mantenimiento

El montaje se deberá hacer en un entorno limpio y con las medidas de seguridad apropiadas. Se empleará un soldador de estaño convencional para el montaje de los componentes, y el pisteado de la placa se realizará mediante una subcontrata externa.

5.3.2. Pruebas y recepción

El Director de Ejecución debe asegurar tras finalizar la ejecución de los trabajos, que el sistema sea completamente funcional y operativo, de acuerdo con los términos legales.

Para velar porque se cumpla esta premisa, el Director de Ejecución deberá realizar de forma obligatoria los siguientes ensayos:

- Comprobación de continuidad y fidelidad con el esquemático al recepcionar las placas.
- Comprobación de las dimensiones de los componentes adquiridos.
- Comprobar la idoneidad de las características de los valores escogidos.
- Examen visual general.

- Comprobación de las conexiones.
- Pruebas de funcionamiento y puesta en marcha.

5.3.3. Garantía

Será tarea del Director de Ejecución garantizar que todos los materiales utilizados en este proyecto sean nuevos y libres de defectos.

Así mismo, deberá garantizar el funcionamiento del sistema por un período de dos años a partir de la recepción, comprometiéndose a reemplazar de manera gratuita cualquier defecto.

Jaén a 20 de Octubre de 2018

Juan Caño Navas

Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén
Grado en Ingeniería Electrónica

PRESUPUESTO

Alumno: Juan Caño Navas

Tutor: Prof. D. Miguel de la Fuente Ruz

Dpto: Ingeniería de Sistemas y Automática

Diciembre, 2018

ÍNDICE

1. Mediciones

Se procede a realizar la medición de cada tarjeta del proyecto:

1.1. Medición placa máster

Qty	Valor	Componente	Designación	Descripción
1		22-23-2061	X1	Conector 6 pines
1		LEDCHIPLED_1206	LED1	Led naranja
1		PINHD-1X1	JP1	Conector 1 pin
1	100n	C-EUC1206	C1	Condensador 100nF
1	100p	C-EUC1206	C2	Condensador 100pF
3	10k	R-EU_M1206	R6, R7, R8	Resistor 10kΩ
3	10pF	C-EUC1206	C4, C6, C8	Condensador 10pF
3	10uF	C-EUC1206	C3, C5, C7	Condensador 10uF
1	22-23-2021	22-23-2021	X2	Conector 2 pines
2	22-23-2031	22-23-2031	X3, X4	Conector 3 pines
2	2k2	R-EU_R1206	R3, R4	Resistores 2,2kΩ
1	330	R-EU_R1206	R2	Resistor 330Ω
1	4k7	R-EU_R1206	R5	Resistor 4,7kΩ
1	50k	R-EU_R1206	R1	Resistor 50kΩ
1	Transistor NPN	BC848C-7-F	U1	Transistor de potencia
1	Fuente regulada 5V	MIC5225-5.0YM5-TR	IC3	Fuente integrada
1	Microcontrolador 8bits	PIC16F19155-I_SS	IC1	Microcontrolador 8bits
1	Módulo Bluetooth	RN4020-V_RM120	IC2	Módulo Bluetooth
1	Transistor NPN de potencia	SBCP56-16T1G	Q1	Transistor
1		PCB	Placa 52x32mm	Placa doble cara

1.2. Medición placa esclava

Qty	Valor	Componente	Designación	Descripción
6		22-23-2061	X21, X22, X23, X24, X25, X26	Conector 6 pines
6		CHIPLED_1206	LED1, LED2, LED3, LED4, LED5, LED6	Led Naranja
6		W237-102	X1, X2, X3, X4, X5, X6	Conector 2 pines
6	1.6kΩ	R1206	R2, R10, R18, R26, R34, R42	Resistor 1,6kΩ
6	100nF	C1206	C1, C5, C9, C13, C17, C21	Condensador 100nF
6	100uH	L2012C	L1, L2, L3, L4, L5, L6	Inductor 100µH
6	10kΩ	R1206	R1, R9 , R17, R25, R33, R41	Resistor 10kΩ

6	8.3kΩ	R1206	R8, R16, R24, R32, R40, R48	Resistor 10kΩ
6	10nF	C1206	C4, C8, C12, C16, C20, C24	Condensador 10nF
12	1kΩ	R1206	R4, R5, R12, R13, R20, R21, R28, R29, R36, R37, R44, R45	Resistor 1kΩ
6	1nF	C1206	C2, C6, C10, C14, C18, C22	Condensador 1nF
6	1uF	C1206	C3, C7, C11, C15, C19, C23	Condensador 1µF
1		22-23-2031	X10	Conector 3 pines
12	330Ω	R1206	R3, R7, R11, R15, R19, R23, R27, R31, R35, R39, R43, R47	Resistor 330Ω
6	50KΩ	R1206	R6, R14, R22, R30, R38, R46	Resistor 50kΩ
6	Diodo TVS	SOD-323	D1, D2, D3, D4, D5, D6	Diodo bidireccional supresor de transitorios
6	Optoacoplador	SOIC8	IC1, IC2, IC5, IC7, IC9, IC11	Integrado 2 optoacopladores
6	Microcontrolador 8bits	SOIC8	IC3, IC4, IC6, IC8, IC10, IC12	Microcontrolador 8bits
1		PCB	Placa 72x62mm	Placa doble cara

2. Cuadro de precios

2.1. Precios unitarios

2.1.1. Precios unitarios placa máster

Designación	Precio letra	Precio cifra (€)
LED1	Cuatro coma veintiocho céntimos	0,0428
C1	Cinco coma seis céntimos	0,056
C2	Ocho coma cinco céntimos	0,085
R6, R7, R8	Cero coma cuatro céntimos	0,004
C4, C6, C8	Seis coma ocho céntimos	0,068
C3, C5, C7	Cuatro coma cuatro céntimos	0,044
R3, R4	Dos coma ocho céntimos	0,028
R2	Cinco coma tres céntimos	0,053
R5	Uno coma siete céntimos	0,017
R1	Tres coma dos céntimos	0,032
U1	Quince con un céntimos	0,151
IC3	Treinta y cuatro céntimos	0,34
IC1	Ochenta y tres coma dos céntimos	0,832
IC2	Ocho euros con cuarenta y seis céntimos	8,46
Q1	Treinta y cinco con cuatro céntimos	0,354
Placa 52x32mm	Ochenta y siete coma setenta y tres céntimos	0,8773

2.1.2. Precios unitarios placa esclava

Designación	Precio letra	Precio cifra (€)
LED1, LED2, LED3, LED4, LED5, LED6	Cuatro coma veintiocho céntimos	0,0428
R2, R10, R18, R26, R34, R42	Cinco coma tres céntimos	0,053
C1, C5, C9, C13, C17, C21	Cinco coma seis céntimos	0,056
L1, L2, L3, L4, L5, L6	Once coma cuatro céntimos	0,114
R1, R9, R17, R25, R33, R41	Cero coma cuatro céntimos	0,004
R8, R16, R24, R32, R40, R48	Cero coma seis céntimos	0,006
C4, C8, C12, C16, C20, C24	Dos coma ocho céntimos	0,028
R4, R5, R12, R13, R20, R21, R28, R29, R36, R37, R44, R45	Cero coma ocho céntimos	0,008
C2, C6, C10, C14, C18, C22	Siete céntimos	0,07
C3, C7, C11, C15, C19, C23	Tres coma nueve céntimos	0,039
R3, R7, R11, R15, R19, R23, R27, R31, R35, R39, R43, R47	Uno coma siete céntimos	0,017
R6, R14, R22, R30, R38, R46	Seis coma tres céntimos	0,063
D1, D2, D3, D4, D5, D6	Once coma cinco céntimos	0,115
IC1, IC2, IC5, IC7, IC9, IC11	Cuarenta y ocho coma dos céntimos	0,482
IC3, IC4, IC6, IC8, IC10, IC12	Sesenta coma cuatro céntimos	0,604
Placa 72x62mm	Ochenta y siete coma setenta y tres céntimos	0,8773

2.2. Aplicación de precios

2.2.1. Aplicación de precios a placa máster

Designación	Valor	Qty	Precio unitario (€)	Precio (€)
LED1		1	0,0428	0,0428
C1	100n	1	0,056	0,056
C2	100p	1	0,085	0,085
R6, R7, R8	10k	3	0,004	0,012
C4, C6, C8	10pF	3	0,068	0,204
C3, C5, C7	10uF	3	0,044	0,132

R3, R4	2k2	2	0,028	0,056
R2	330	1	0,053	0,053
R5	4k7	1	0,017	0,017
R1	50k	1	0,032	0,032
U1	Transistor NPN de potencia	1	0,151	0,151
IC3	Fuente Integrada 5V	1	0,34	0,34
IC1	Microcontrolador 8bits	1	0,832	0,832
IC2	Módulo Bluetooth	1	8,46	8,46
Q1	Transistor NPN	1	0,354	0,354
Placa 52x32mm		1	0,8773	0,8773
	PRECIO TOTAL :			**11,70**

PLACA MAESTRA		
Coste componentes		11,70€
Mano de obra	2 horas técnico – 20€/h	40€
Maquinaria	Soldador	10€
Coste unitario placa maestra:		61,70€

2.2.2. Aplicación de precios a placa esclava

Designación	Valor	Qty	Precio unitario (€)	Precio (€)
LED1, LED2, LED3, LED4, LED5, LED6		6	0,0428	0,2568
R2, R10, R18, R26, R34, R42	1.6kΩ	6	0,053	0,318
C1, C5, C9, C13, C17, C21	100nF	6	0,056	0,336
L1, L2, L3, L4, L5, L6	100uH	6	0,114	0,684
R1, R9, R17, R25, R33, R41	10kΩ	6	0,004	0,024
R8, R16, R24, R32, R40, R48	8.3 kΩ	6	0.006	0.036
C4, C8, C12, C16, C20, C24	10nF	6	0,028	0,168
R4, R5, R12, R13, R20, R21, R28, R29, R36, R37, R44, R45	1kΩ	12	0,008	0,096

C2, C6, C10, C14, C18, C22	1nF	6	0,07	0,42
C3, C7, C11, C15, C19, C23	1uF	6	0,039	0,234
R3, R7, R11, R15, R19, R23, R27, R31, R35, R39, R43, R47	330Ω	12	0,017	0,204
R6, R14, R22, R30, R38, R46	50KΩ	6	0,063	0,378
D1, D2, D3, D4, D5, D6	Diodo TVS	6	0,115	0,69
IC1, IC2, IC5, IC7, IC9, IC11	Optoacoplador	6	0,482	2,892
IC3, IC4, IC6, IC8, IC10, IC12	Micro 8bits	6	0,604	3,624
Placa 72x62mm		1	0,8773	0,8773
PRECIO TOTAL:				**11,34**

PLACA ESCLAVA		
Coste componentes		11,34€
Mano de obra	3 horas técnico – 20€/h	60€
Maquinaria	Soldador	10€
Coste unitario placa maestra:		81,34€

3. Presupuesto total

3.1. Costes de control de calidad

El control de calidad debe:

- **Antes del montaje:**
  - Certificar el estado de los componentes
  - Comprobar la manufactura de las placas
- **Después del montaje:**
  - Examen visual del resultado
  - Comprobación de continuidad
  - Comprobación del estado de las soldaduras

El control de calidad queda fijado en un 1,5% del total.

3.2. Costes de diseño

Costes de diseño	
Actividad	**Tiempo (horas)**
Estudio previo	50
Simulaciones	10
Diseño placa maestra	80
Diseño placa esclava	80
Redacción del proyecto	50
Total horas	**270**
TOTAL (20€/h)	**5.400,00€**

3.3. Presupuesto de Ejecución material

Presupuesto de Ejecución Material			
Elemento	**Cantidad**	**Precio unitario**	**Total**
Placa esclava	26	81,3461€	2.114,99€
Placa maestra	1	61,7041€	61,70€
Costes de diseño			5.400,00€
Total			7.576,69€
Costes de control de calidad			200,00€
TOTAL		**7.776,69€**	

3.4. Presupuesto de Ejecución por Contrata

Presupuesto de Ejecución por Contrata	
Presupuesto de ejecución material	7.776,69€
Gastos generales (15%)	1.166,04€
Beneficio industrial (6%)	466,42€
Total sin IVA	9.406,04€
IVA (21%)	1.975,27€
PRESUPUESTO TOTAL DE EJECUCIÓN POR CONTRATA	**11.384,65€**

Asciende el Presupuesto Total de Ejecución por Contrata del Sistema de gestión de baterías “BMS” a la cantidad de **ONCE MIL TRESCIENTOS OCHENTA Y CUATRO EUROS CON SESENTA Y CINCO CÉNTIMOS.**

Jaén, a 10 de Diciembre de 2018

Juan Caño Navas

Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén
Grado en Ingeniería Electrónica

ANEXO 1: PLACAS ESCLAVAS

Alumno: Juan Caño Navas

Tutor: Prof. D. Miguel de la Fuente Ruz

Dpto: Ingeniería de Sistemas y Automática

Diciembre, 2018

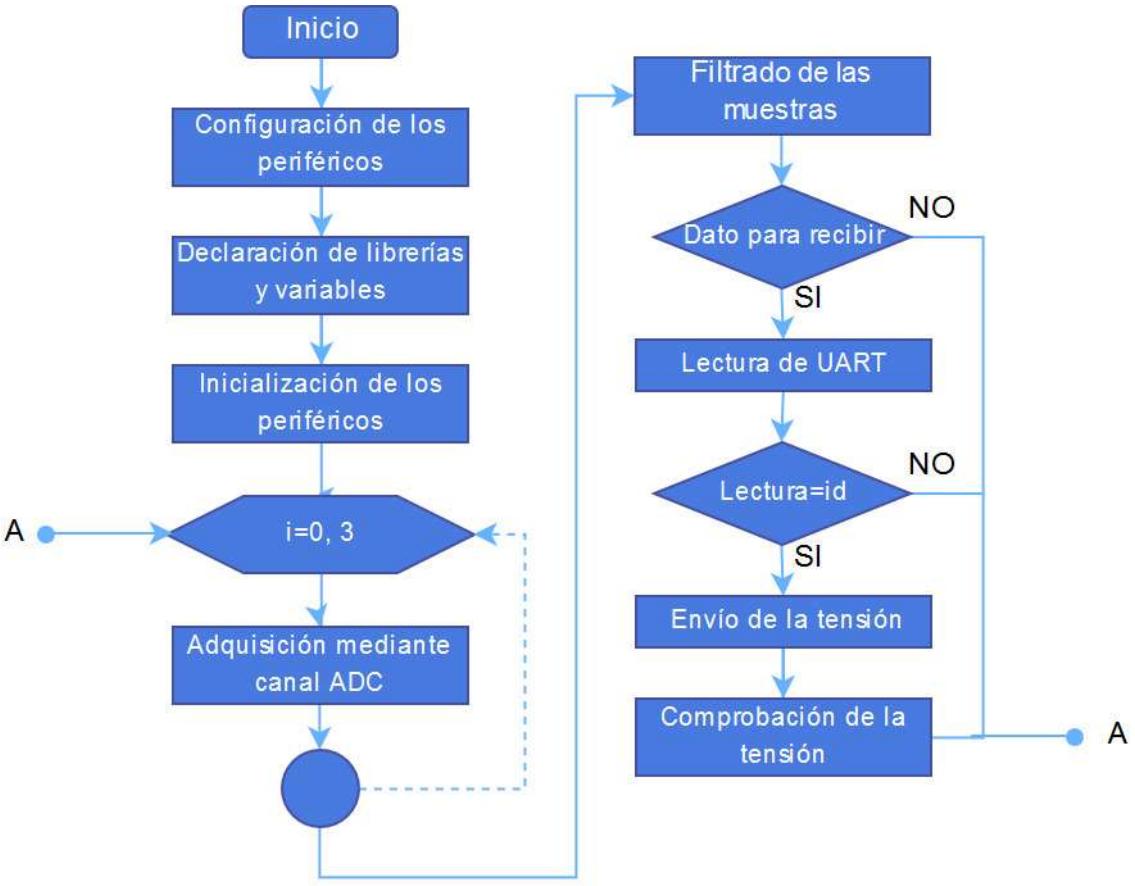
ÍNDICE

1. Introducción

El objeto del presente documento es la descripción detallada de la programación de las placas esclavas del sistema BMS. El documento expone al principio el flujo de programación que se ha seguido, y posteriormente se desglosa la programación, así como la configuración de cada uno de los módulos empleados.

2. Flujograma

En el siguiente apartado se adjunta la programación estas placas comentando cada parte. Para facilitar la comprensión, se dividirá este apartado en subapartados con la configuración de cada periférico el programa principal. Además, previo al desglose de la programación, se va expondrá el método de funcionamiento mediante un flujograma del programa principal (main).



3. Configuración ADC

El ADC ha sido configurado para que su reloj sea igual $F_{osc}/64$, lo que se traduce en un tiempo de conversión de 46µs, con un alineamiento del byte de resultado a la izquierda, una referencia positiva dada por el módulo FVR, y una referencia negativa igual a V_{SS}. Así mismo, se configura el pin 2 con entrada analógica (ANA5).

3.1. adc.h

```
#ifndef _ADC_H
#define _ADC_H

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

typedef uint16_t adc_result_t;

/**
 *  Resultado de una conversión ADC doble
 */
typedef struct{
    adc_result_t adcResult1;
    adc_result_t adcResult2;
} adc_sync_double_result_t;

/** Definición de canales ADC
 @Summary
   Define los canales disponibles para realizar la conversión.
 */
typedef enum{
    channel_ANA0 =  0x0,
    channel_ANA5 =  0x5,
    channel_AVSS =  0x3C,
    channel_Temp =  0x3D,
    channel_DAC1 =  0x3E,
    channel_FVR =  0x3F
} adc_channel_t;



/**
```

```
  @Summary
    Inicia el ADC

  @Description
    Esta rutina inicializa el módulo ADC
    Esta rutina debe ser llamada antes de llamar a cualquier otra rutina ADC.
    Se recomienda llamar a esta rutina sólo durante la inicialización.
*/
void ADC_Initialize(void);
/**
  @Summary
    Permite seleccionar el canal ADC
  @Description
    Esta rutina es usada para seleccionar el canal deseado para la conversión.
  @Preconditions
    ADC_Initialize() debe haber sido llamada previamente.
  @Returns
    None
  @Param
    Pase el número de canal requerido
    "Para consultar canales disponibles revise el enum disponible en adc.h"
*/
void ADC_SelectChannel(adc_channel_t channel);
/**
  @Summary
    Inicia la conversión
  @Description
    Esta rutina se emplea para iniciar la conversión del canal deseado.
  @Preconditions
    ADC_Initialize() debe haber sido llamada antes de utilizar esta rutina.
  @Returns
    None

  @Param
    None
*/
void ADC_StartConversion();

/**
  @Summary
```

```
    Devuelve "true" cuando finaliza la conversión, "false" si no.
  @Description
    Esta rutina es usada para determinar si la conversión ha finalizado.
    Cuando la conversión está completa la rutina devuelve un "true". Devuelve falso  en cualquier
    otro caso.
  @Preconditions
    ADC_Initialize() y ADC_StartConversion(adc_channel_t channel) deben ser llamadas antes de
    llamar a esta función.
  @Returns
   true  - Si la conversión es completa
   false - Si la conversión no es completa
  @Param
    None
*/
bool ADC_IsConversionDone();

/**
  @Summary
    Devuelve el valor de la conversión.

  @Description
    Esta rutina es usada para obtener el valor analógico en valor de conversión
    digital. Esta rutina obtiene valores convertidos del canal especificado.

@Preconditions
Esta rutina devuelve el valor de conversión sólo después de que la conversión se haya completado.
El estatus de completado puede ser revisado utilizando la rutina  ADC_IsConversionDone().

  @Returns
    Devuelve el valor convertido.

  @Param
    None
 */
adc_result_t ADC_GetConversionResult(void);
```

3.2. adc.c

```
#include <xc.h>
#include "adc.h"
#include "mcc.h"
```

```
#define ACQ_US_DELAY 5

void ADC_Initialize(void){
    // Configura el módulo con los parámetros de nuestra aplicación
    // ADGO stop; ADON enabled; CHS ANA0;
    ADCON0 = 0x01;

    // ADFM left; ADNREF VSS; ADPREF FVR; ADCS FOSC/64;
    ADCON1 = 0x63;

    // ADACT no_auto_trigger;
    ADACT = 0x00;

    // ADRESL 0;
    ADRESL = 0x00;

    // ADRESH 0;
    ADRESH = 0x00;

}

void ADC_SelectChannel(adc_channel_t channel){
    // Selecciona el canal A/D
    ADCON0bits.CHS = channel;
    // Activa el módulo ADC
    ADCON0bits.ADON = 1;
}

void ADC_StartConversion(){
    // Inicia la conversión
    ADCON0bits.ADGO = 1;
}

bool ADC_IsConversionDone(){
    // Comprueba si se ha realizado la conversión
    return (!ADCON0bits.ADGO);
}

adc_result_t ADC_GetConversionResult(void){
```

```
    // Devuelve el resultado de la conversión
    return ((ADRESH << 8) + ADRESL);
}
```

4. Configuración EUSART

El módulo EUSART ha sido configurado para un ratio de 9600 baudios, 8 bits de transmisión, 8 bits de recepción e interrupción de este módulo.

4.1. eusart.h

```
#ifndef _EUSART_H
#define _EUSART_H
/**
  Section: Includes
*/

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

/**
  Section: Macros
*/

#define EUSART_DataReady  (eusartRxCount)

/**
 Section: Global variables
 */

extern volatile uint8_t eusartTxBufferRemaining;
extern volatile uint8_t eusartRxCount;

/**
  Section: EUSART APIs
*/
/**
  @Summary
    Rutina de inicialización del módulo EUSART.
```

```
  @Description
    Esta rutina inicializa el driver EUSART.
    Esta rutina debe ser llamada antes que otra rutina EUSART.
*/
void EUSART_Initialize(void);



/**
  @Summary
    Lee un byte de datos desde el EUSART.
  @Description
    Esta rutina lee un byte de datos desde el EUSART.
  @Preconditions
    EUSART_Iniatialize() debe haber sido llamada antes de llamar a esta función. El estatus de
      transferencia debe ser revisado para ver que el receptor no esté vacío antes de llamar a esta
      función
    EUSART_DataReady es una macro que verifica si algún byte es recibido. LLamar a esa macro
      antes de usar esta función
  @Returns
    Un byte de datos recibido por el driver.
*/
uint8_t EUSART_Read(void);

 /**
  @Summary
    Escribe un byte de datos para el EUSART
  @Description
    Esta rutina escribe un byte de datos para el EUSART
  @Preconditions
    EUSART_Initialize() debe ser llamada antes de llamar a esta función. El estatus de envío debe
      ser llamado antes de llamar a esta función. El estatus de envío debe ser revisado para ver si el
      transmisor no está ocupado antes de llamar a esta función.
  @Param
    txData  - Byte de datos a escribir en el EUSART
  @Returns
    None
*/
void EUSART_Write(uint8_t txData);
```

```
/**
  @Summary
    Mantiene el estado del driver receptor e implementa su ISR.
  @Description
    Esta rutina es usada para mantener la máquina de estado receptora del driver interno. Este
      servicio de interrupción es llamado cuando el estado del receptor necesita ser revisado.
  @Preconditions
    EUSART_Initialize() debe ser llamada para que el ISR se ejecute correctamente
  @Param
    None
  @Returns
    None
*/
void EUSART_Receive_ISR(void);

#endif
/**
 End of File
*/
```

4.2. eusart.c

```
/**
  Section: Included Files
*/
#include "eusart.h"
#include "adc.h"
#include "mcc.h"

/**
  Section: Macro Declarations
*/
#define EUSART_TX_BUFFER_SIZE 8
#define EUSART_RX_BUFFER_SIZE 8

/**
  Section: Global Variables
*/

volatile uint8_t eusartTxHead = 0;
volatile uint8_t eusartTxTail = 0;
```

```
volatile uint8_t eusartTxBuffer[EUSART_TX_BUFFER_SIZE];
volatile uint8_t eusartTxBufferRemaining;

volatile uint8_t eusartRxHead = 0;
volatile uint8_t eusartRxTail = 0;
volatile uint8_t eusartRxBuffer[EUSART_RX_BUFFER_SIZE];
volatile uint8_t eusartRxCount;

/**
  Section: EUSART APIs
*/

void EUSART_Initialize(void){
    // Deshabilita las interrupciones antes de cambiar estados
    PIE1bits.RCIE = 0;
    PIE1bits.TXIE = 0;

    // Habilita el módulo EUSART con la configuración deseada

    // ABDOVF no_overflow; SCKP Non-Inverted; BRG16 16bit_generator; WUE disabled; ABDEN
       disabled;
    BAUD1CON = 0x08;

    // SPEN enabled; RX9 8-bit; CREN enabled; ADDEN disabled; SREN disabled;
    RC1STA = 0x90;

    // TX9 8-bit; TX9D 0; SENDB sync_break_complete; TXEN enabled; SYNC asynchronous; BRGH
       hi_speed; CSRC slave;
    TX1STA = 0x24;

    // Baud Rate = 9600; SP1BRGL 160;
    SP1BRGL = 0xA0;

    // Baud Rate = 9600; SP1BRGH 1;
    SP1BRGH = 0x01;


    // Inicialización del estado del driver
    eusartTxHead = 0;
    eusartTxTail = 0;
```

```
    eusartTxBufferRemaining = sizeof(eusartTxBuffer);

    eusartRxHead = 0;
    eusartRxTail = 0;
    eusartRxCount = 0;

    // Habilita la interrupción de RX
    PIE1bits.RCIE = 1;
}

uint8_t EUSART_Read(void){
    uint8_t readValue  = 0;

    while(0 == eusartRxCount);

    readValue = eusartRxBuffer[eusartRxTail++];
    if(sizeof(eusartRxBuffer) <= eusartRxTail){
        eusartRxTail = 0;
    }
    PIE1bits.RCIE = 0;
    eusartRxCount--;
    PIE1bits.RCIE = 1;

    return readValue;
}

void EUSART_Write(uint8_t txData){
    while(0 == eusartTxBufferRemaining);

    if(0 == PIE1bits.TXIE){
        TX1REG = txData;
    }
    else{
        PIE1bits.TXIE = 0;
        eusartTxBuffer[eusartTxHead++] = txData;
        if(sizeof(eusartTxBuffer) <= eusartTxHead){
            eusartTxHead = 0;
        }
        eusartTxBufferRemaining--;
    }
```

```
    PIE1bits.TXIE = 1;
}

void EUSART_Receive_ISR(void){
    recepcion=true;
    if(1 == RC1STAbits.OERR){
        // EUSART error - restart

        RC1STAbits.CREN = 0;
        RC1STAbits.CREN = 1;
    }

    // buffer overruns are ignored
    eusartRxBuffer[eusartRxHead++] = RC1REG;
    if(sizeof(eusartRxBuffer) <= eusartRxHead){
        eusartRxHead = 0;
    }
    eusartRxCount++;
}
/**
  End of File
*/
```

5. Configuración FVR

El módulo FVR se ha configurado para ser referencia del módulo ADC, con una ganancia de 2x, lo que se traduce en una tensión de referencia de 2.048V, y se habilita el sensor de temperatura interno.

5.1. fvr.h

```
#ifndef _FVR_H
#define _FVR_H

/**
  Section: Included Files
*/

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

```
/**
  Section: FVR APIs
*/

/**
  @Summary
    Inicializa el FVR
  @Description
    Esta rutina inicializa el FVR.
    Esta rutina debe ser llamada antes que cualquier otra rutina FVR.
    Esta rutina sólo debe ser llamada una vez durante la inicialización del sistema.
  @Preconditions
    None
  @Param
    None
  @Returns
    None
  */
 void FVR_Initialize(void);

/**
  @Summary
    Obtine el estatus de salida del FVR.
  @Description
    Esta rutina obtiene el estatus de la salida FVR.

  @Preconditions
    FVR_Initialize() debe haber sido llamada previamente de usar esta rutina.
  @Param
    None
  @Returns
     true  - El módulo FVR está listo para usarse.
     false - El módulo FVR no está listo para usarse.

*/
bool FVR_IsOutputReady(void);

#endif // _FVR_H
/**
```

```
 End of File
*/
```

5.2. fvr.c

```
/**
  Sección: Includes
*/

#include <xc.h>
#include "fvr.h"

/**
  Sección: FVR APIs
*/

void FVR_Initialize(void){
    // CDAFVR off; FVREN enabled; TSRNG Hi_range; ADFVR 2x; TSEN enabled;
    FVRCON = 0xB2;
}

bool FVR_IsOutputReady(void){
    return (FVRCONbits.FVRRDY);
}
/**
 End of File
*/
```

6. Configuración I/O

El módulo de I/O, ha sido configurado para que tener como entrada digital RA4, como entrada analógica RA5, con pin de transmisión RA2 y como de recepción RA1.

6.1. pin_manager.h

```
#ifndef PIN_MANAGER_H

#define PIN_MANAGER_H
```

```
#define INPUT   1
#define OUTPUT  0

#define HIGH    1
#define LOW     0

#define ANALOG      1
#define DIGITAL     0

#define PULL_UP_ENABLED      1
#define PULL_UP_DISABLED     0

// DEFINES para channel_ANA0
#define channel_ANA0_TRIS               TRISAbits.TRISA0
#define channel_ANA0_LAT                LATAbits.LATA0
#define channel_ANA0_PORT               PORTAbits.RA0
#define channel_ANA0_WPU                WPUAbits.WPUA0
#define channel_ANA0_OD                 ODCONAbits.ODCA0
#define channel_ANA0_ANS                ANSELAbits.ANSA0
#define channel_ANA0_SetHigh()          do { LATAbits.LATA0 = 1; } while(0)
#define channel_ANA0_SetLow()           do { LATAbits.LATA0 = 0; } while(0)
#define channel_ANA0_Toggle()           do { LATAbits.LATA0 = ~LATAbits.LATA0; } while(0)
#define channel_ANA0_GetValue()         PORTAbits.RA0
#define channel_ANA0_SetDigitalInput()    do { TRISAbits.TRISA0 = 1; } while(0)
#define channel_ANA0_SetDigitalOutput()   do { TRISAbits.TRISA0 = 0; } while(0)
```

```
#define channel_ANA0_SetPullup()      do { WPUAbits.WPUA0 = 1; } while(0)
#define channel_ANA0_ResetPullup()    do { WPUAbits.WPUA0 = 0; } while(0)
#define channel_ANA0_SetPushPull()    do { ODCONAbits.ODCA0 = 1; } while(0)
#define channel_ANA0_SetOpenDrain()   do { ODCONAbits.ODCA0 = 0; } while(0)
#define channel_ANA0_SetAnalogMode()  do { ANSELAbits.ANSA0 = 1; } while(0)
#define channel_ANA0_SetDigitalMode() do { ANSELAbits.ANSA0 = 0; } while(0)

// DEFINES para RA1
#define RA1_SetHigh()    do { LATAbits.LATA1 = 1; } while(0)
#define RA1_SetLow()   do { LATAbits.LATA1 = 0; } while(0)
#define RA1_Toggle()   do { LATAbits.LATA1 = ~LATAbits.LATA1; } while(0)
#define RA1_GetValue()         PORTAbits.RA1
#define RA1_SetDigitalInput()   do { TRISAbits.TRISA1 = 1; } while(0)
#define RA1_SetDigitalOutput()  do { TRISAbits.TRISA1 = 0; } while(0)
#define RA1_SetPullup()     do { WPUAbits.WPUA1 = 1; } while(0)
#define RA1_ResetPullup()   do { WPUAbits.WPUA1 = 0; } while(0)
#define RA1_SetAnalogMode() do { ANSELAbits.ANSA1 = 1; } while(0)
#define RA1_SetDigitalMode()do { ANSELAbits.ANSA1 = 0; } while(0)

// DEFINES para RA2
#define RA2_SetHigh()    do { LATAbits.LATA2 = 1; } while(0)
#define RA2_SetLow()   do { LATAbits.LATA2 = 0; } while(0)
#define RA2_Toggle()   do { LATAbits.LATA2 = ~LATAbits.LATA2; } while(0)
#define RA2_GetValue()         PORTAbits.RA2
#define RA2_SetDigitalInput()   do { TRISAbits.TRISA2 = 1; } while(0)
#define RA2_SetDigitalOutput()  do { TRISAbits.TRISA2 = 0; } while(0)
```

```
#define RA2_SetPullup()     do { WPUAbits.WPUA2 = 1; } while(0)
#define RA2_ResetPullup()   do { WPUAbits.WPUA2 = 0; } while(0)
#define RA2_SetAnalogMode() do { ANSELAbits.ANSA2 = 1; } while(0)
#define RA2_SetDigitalMode()do { ANSELAbits.ANSA2 = 0; } while(0)

// DEFINES PARA IO_RA4
#define IO_RA4_TRIS              TRISAbits.TRISA4
#define IO_RA4_LAT               LATAbits.LATA4
#define IO_RA4_PORT              PORTAbits.RA4
#define IO_RA4_WPU               WPUAbits.WPUA4
#define IO_RA4_OD                ODCONAbits.ODCA4
#define IO_RA4_ANS               ANSELAbits.ANSA4
#define IO_RA4_SetHigh()          do { LATAbits.LATA4 = 1; } while(0)
#define IO_RA4_SetLow()           do { LATAbits.LATA4 = 0; } while(0)
#define IO_RA4_Toggle()           do { LATAbits.LATA4 = ~LATAbits.LATA4; } while(0)
#define IO_RA4_GetValue()         PORTAbits.RA4
#define IO_RA4_SetDigitalInput()    do { TRISAbits.TRISA4 = 1; } while(0)
#define IO_RA4_SetDigitalOutput()   do { TRISAbits.TRISA4 = 0; } while(0)
#define IO_RA4_SetPullup()      do { WPUAbits.WPUA4 = 1; } while(0)
#define IO_RA4_ResetPullup()    do { WPUAbits.WPUA4 = 0; } while(0)
#define IO_RA4_SetPushPull()    do { ODCONAbits.ODCA4 = 1; } while(0)
#define IO_RA4_SetOpenDrain()   do { ODCONAbits.ODCA4 = 0; } while(0)
#define IO_RA4_SetAnalogMode()  do { ANSELAbits.ANSA4 = 1; } while(0)
#define IO_RA4_SetDigitalMode() do { ANSELAbits.ANSA4 = 0; } while(0)

// DEFINES para channel_ANA5
```

```
#define channel_ANA5_TRIS               TRISAbits.TRISA5
#define channel_ANA5_LAT                LATAbits.LATA5
#define channel_ANA5_PORT               PORTAbits.RA5
#define channel_ANA5_WPU                WPUAbits.WPUA5
#define channel_ANA5_OD                 ODCONAbits.ODCA5
#define channel_ANA5_ANS                ANSELAbits.ANSA5
#define channel_ANA5_SetHigh()          do { LATAbits.LATA5 = 1; } while(0)
#define channel_ANA5_SetLow()           do { LATAbits.LATA5 = 0; } while(0)
#define channel_ANA5_Toggle()           do { LATAbits.LATA5 = ~LATAbits.LATA5; } while(0)
#define channel_ANA5_GetValue()         PORTAbits.RA5
#define channel_ANA5_SetDigitalInput()    do { TRISAbits.TRISA5 = 1; } while(0)
#define channel_ANA5_SetDigitalOutput()   do { TRISAbits.TRISA5 = 0; } while(0)
#define channel_ANA5_SetPullup()      do { WPUAbits.WPUA5 = 1; } while(0)
#define channel_ANA5_ResetPullup()    do { WPUAbits.WPUA5 = 0; } while(0)
#define channel_ANA5_SetPushPull()    do { ODCONAbits.ODCA5 = 1; } while(0)
#define channel_ANA5_SetOpenDrain()   do { ODCONAbits.ODCA5 = 0; } while(0)
#define channel_ANA5_SetAnalogMode()  do { ANSELAbits.ANSA5 = 1; } while(0)
#define channel_ANA5_SetDigitalMode() do { ANSELAbits.ANSA5 = 0; } while(0)

/**
  @Param
    none
  @Returns
    none
  @Description
    Inicialización GPIO y periféricos I/O
```

```
 */
void PIN_MANAGER_Initialize (void);

#endif // PIN_MANAGER_H
/**
 End of File
*/
```

6.2. pin_manager.c

```
#include <xc.h>
#include "pin_manager.h"
#include "stdbool.h"

void PIN_MANAGER_Initialize(void){
    LATA = 0x00; //LATx registro

    TRISA = 0x23; //TRISx registro
    ANSELA = 0x21; //ANSELx registro
    WPUA = 0x00; //WPUx registro
    ODCONA = 0x00; //ODx registro

    bool state = GIE;
    GIE = 0;
    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x00; // desbloquea PPS
```

```
    RA2PPSbits.RA2PPS = 0x14;   //RA2->EUSART:TX;

    RXPPSbits.RXPPS = 0x01;   //RA1->EUSART:RX;


    PPSLOCK = 0x55;

    PPSLOCK = 0xAA;

    PPSLOCKbits.PPSLOCKED = 0x01; // bloquea PPS


    GIE = state;

}

/**

 End of File

*/
```

7. Interrupciones

La gestión de interrupciones ha sido configurada para que sólo se atienda a la interrupción de la recepción del puerto serie.

7.1. interrupt_manager.h

```
#ifndef INTERRUPT_MANAGER_H
#define INTERRUPT_MANAGER_H

/**
 * @Param
   none
 * @Returns
   none
 * @Description
   Esta macro habilitará las interrupciones globales.
 */
#define INTERRUPT_GlobalInterruptEnable() (INTCONbits.GIE = 1)

/**
 * @Param
   none
 * @Returns
```

```
    none
 * @Description
    Esta macro deshabilitará las interrupciones globales.
 */
#define INTERRUPT_GlobalInterruptDisable() (INTCONbits.GIE = 0)

/**
 * @Param
    none
 * @Returns
    none
 * @Description
    Esta macro habilitará las interrupciones periféricas.
 */
#define INTERRUPT_PeripheralInterruptEnable() (INTCONbits.PEIE = 1)

/**
 * @Param
    none
 * @Returns
    none
 * @Description
   Esta macro deshabilitará las interrupciones globales.
 */
#define INTERRUPT_PeripheralInterruptDisable() (INTCONbits.PEIE = 0)

/**
 * @Param
    none
 * @Returns
    none
 * @Description
    Rutina de servicio a la interrupción principal. Llama a los handlers de los módulos.
 */
void interrupt INTERRUPT_InterruptManager(void);


#endif  // INTERRUPT_MANAGER_H
/**
 End of File*/
```

7.2. interrupt_manager.c

```
#include "interrupt_manager.h"
#include "mcc.h"

void interrupt INTERRUPT_InterruptManager (void){
    // interrupt handler
    if(INTCONbits.PEIE == 1 && PIE1bits.RCIE == 1 && PIR1bits.RCIF == 1)
        EUSART_Receive_ISR();
}
/**
 End of File
*/
```

8. Configuración del sistema

El sistema ha sido configurado para trabajar con el oscilador interno del dispositivo, siendo este configurado para que el reloj interno sea de 16MHz.

8.1. Mcc.h

```
#ifndef MCC_H
#define MCC_H
#include <xc.h>
#include "pin_manager.h"
#include <stdint.h>
#include <stdbool.h>
#include "interrupt_manager.h"
#include "adc.h"
#include "fvr.h"
#include "eusart.h"

#define _XTAL_FREQ  16000000
bool recepcion;


/**
 * @Param
   none
 * @Returns
   none
 * @Description
   Inicializa el dispositivo con las configuraciones de la aplicación
 */
void SYSTEM_Initialize(void);

/**
 * @Param
   none
 * @Returns
   none
 * @Description
   Inicializa el oscilador
 * @Example
   OSCILLATOR_Initialize(void);
 */
void OSCILLATOR_Initialize(void);

/**
 * @Param
   none
 * @Returns
   none
 * @Description
   Inicializa el Watchdog con la configuración deseada
 */
void WDT_Initialize(void);

#endif /* MCC_H */
/**
 End of File
*/
```

8.2. Mcc.c

```
// Configuration bits:

// CONFIG1
```

```
#pragma config FEXTOSC = OFF    // FEXTOSC External Oscillator mode Selection bits->Oscillator not enabled

#pragma config RSTOSC = HFINT1    // Power-up default value for COSC bits->HFINTOSC (1MHz)

#pragma config CLKOUTEN = OFF    // Clock Out Enable bit->CLKOUT function is disabled; I/O or oscillator function on OSC2

#pragma config CSWEN = ON    // Clock Switch Enable bit->Writing to NOSC and NDIV is allowed

#pragma config FCMEN = ON    // Fail-Safe Clock Monitor Enable->Fail-Safe Clock Monitor is enabled

// CONFIG2

#pragma config MCLRE = ON    // Master Clear Enable bit->MCLR/VPP pin function is MCLR; Weak pull-up enabled

#pragma config PWRTE = OFF    // Power-up Timer Enable bit->PWRT disabled

#pragma config WDTE = SLEEP    // Watchdog Timer Enable bits->WDT enabled while running and disabled while in SLEEP/IDLE; SWDTEN is ignored

#pragma config LPBOREN = OFF    // Low-power BOR enable bit->ULPBOR disabled

#pragma config BOREN = ON    // Brown-out Reset Enable bits->Brown-out Reset enabled, SBOREN bit ignored

#pragma config BORV = LOW    // Brown-out Reset Voltage selection bit->Brown-out voltage (Vbor) set to 2.45V

#pragma config PPS1WAY = ON    // PPSLOCK bit One-Way Set Enable bit->The PPSLOCK bit can be cleared and set only once; PPS registers remain locked after one clear/set cycle

#pragma config STVREN = ON    // Stack Overflow/Underflow Reset Enable bit->Stack Overflow or Underflow will cause a Reset

#pragma config DEBUG = OFF    // Debugger enable bit->Background debugger disabled

// CONFIG3

#pragma config WRT = OFF    // User NVM self-write protection bits->Write protection off

#pragma config LVP = ON    // Low Voltage Programming Enable bit->Low voltage programming enabled. MCLR/VPP pin function is MCLR. MCLRE configuration bit is ignored.

// CONFIG4
```

```
#pragma config CP = OFF     // User NVM Program Memory Code Protection bit->User NVM code protection disabled

#pragma config CPD = OFF    // Data NVM Memory Code Protection bit->Data NVM code protection disabled

#include "mcc.h"


void SYSTEM_Initialize(void){

    PIN_MANAGER_Initialize();

    OSCILLATOR_Initialize();

    WDT_Initialize();

    FVR_Initialize();

    // Espera a que el módulo FVR esté listo

    while(FVR_IsOutputReady()==false);

    ADC_Initialize();

    EUSART_Initialize();

}


void OSCILLATOR_Initialize(void){

    // NOSC HFINTOSC; NDIV 1;

    OSCCON1 = 0x60;

    // CSWHOLD may proceed; SOSCPWR Low power; SOSCBE crystal oscillator;

    OSCCON3 = 0x00;

    // LFOEN disabled; ADOEN disabled; SOSCEN disabled; EXTOEN disabled; HFOEN disabled;

    OSCEN = 0x00;

    // HFFRQ 16_MHz;

    OSCFRQ = 0x06;

    // HFTUN 0;
```

```
        OSCTUNE = 0x00;

    }


    void WDT_Initialize(void){

        // WDTPS 1:65536; SWDTEN OFF;

        WDTCON = 0x16;

    }

    /**

     End of File*/
```

9. Main

Puede consultarse el flujograma de la programación para facilitar la comprensión al lector.

```
#include "mcc_generated_files/mcc.h"
#define id 0
//VARIABLES GLOBALES
  uint16_t vectorV[4];
  uint16_t v;
  uint16_t t;
  char caracter;
  uint8_t trama;
  float tension, tension2;
  adc_channel_t adc = channel_ANA5;
//-----------------
//DECLARACIÓN DE FUNCIONES
  uint16_t LeeTension(adc_channel_t channel);
  uint16_t FiltroMedia(uint16_t vector[]);
void main(void)
{
    // Inicializa el dispositivo y sus periféricos
    SYSTEM_Initialize();
    // Habilita las interrupciones globales
    INTERRUPT_GlobalInterruptEnable();
    while (1){
        /*-----------------------
         * Muestreo de la tensión
         *-----------------------*/
        for (int i=0;i<4;i++){
            vectorV[i]=LeeTension(adc);
        }
        /*---------------------------------------------------------------
         *  Tratamiento de la señal. Se filtra la señal recibida mediante un
         *filtro de media móvil para descartar valores anómalos.
         *---------------------------------------------------------------*/
```

```
        v=FiltroMedia(vectorV);
        tension=v*2.096/65536;
        //Cálculo de la tensión. Sólo para depuración de código.
        /*
        tension2=tension/8.333333*10;
        tension=tension+tension2;
        */
        /*-----------------------------------------------------------------
         *  Cuando se recibe algo en la UART, EUSART_DataReady se incremen-
         * ta. Cuando existe más de un mensaje, se realiza
         * la recepción y se responde si procede
         *-----------------------------------------------------------------*/
        if(EUSART_DataReady>0){
           trama=EUSART_Read();
           recepcion=false;
           /*-----------------------------------------------------------
            *  Si el valor recibido por el puerto serie se corresponde con
            * el identificador del microcontrolador significa que se está
            * realizando una petición desde el master, por lo que enviará
            * una trama con el carácter 'v' para indicar envío de tensión,
            * posteriormente el valor guardado, y el carácter 'f' para in-
            * dicar envío completado
            *-----------------------------------------------------------*/
           if(trama==id){
              EUSART_Write('v');
              EUSART_Write(v);
              EUSART_Write('f');
           }
           trama=EUSART_Read();
           /*-----------------------------------------------------------
            *  Si no se recibe correctamente el paquete, el master entende-
            * rá que el envío no ha sido correcto y devolverá un '$'. Ello
            * le indica a nuestro micro que debe tratar de enviar de nuevo
            * el dato.
            *-----------------------------------------------------------*/
           if(trama=='$'){
              EUSART_Write(v);
              EUSART_Write('f');
           }
        }
     }
}
/*-------------------------------------------------------------------------
 *  Función que realiza la lectura del ADC. Cuenta con una entrada y una sa-
 * lida. Inicialmente selecciona el canal a leer, realiza la conversión, es-
 * pera a que finalice y devuelve el valor obtenido.
 * Input: adc_channel_t channel -> Canal de ADC
 * Output: uint16_t q -> valor de lectura del canal
 *-------------------------------------------------------------------------*/
uint16_t LeeTension(adc_channel_t channel){
   uint16_t q;
   ADC_SelectChannel(channel);
   ADC_StartConversion();
   while(ADC_IsConversionDone()!=true);
   q=ADC_GetConversionResult();
   return q;
}
/*-------------------------------------------------------------------------
 *  Función que realiza un filtro de media móvil de 4 valores. Concede más
 * peso al último valor que reciba (último valor del array de entrada), y
```

```
 * se incrementa el peso en los siguientes, asignando 5% al primer valor, 15%
 * al segundo, 35% al tercero y 45% al más reciente.
 * Input: uint16_t vector[] -> array de 4 valores
 * Output: uint16_t filtrado -> Valor filtrado
 *-----------------------------------------------------------------------------*/
uint16_t FiltroMedia(uint16_t vector[]){
  uint16_t filtrado;
  if((sizeof(vector)/sizeof(vector[0]))!=4)
     return 0;
  filtrado=(0.45*vector[3]+0.35*vector[2]+0.15*vector[1]+0.05*vector[0]);
  return filtrado
```

Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén
Grado en Ingeniería Electrónica

ANEXO 2: PLACA MAESTRA

Alumno: Juan Caño Navas

Tutor: Prof. D. Miguel de la Fuente Ruz

Dpto: Ingeniería de Sistemas y Automática

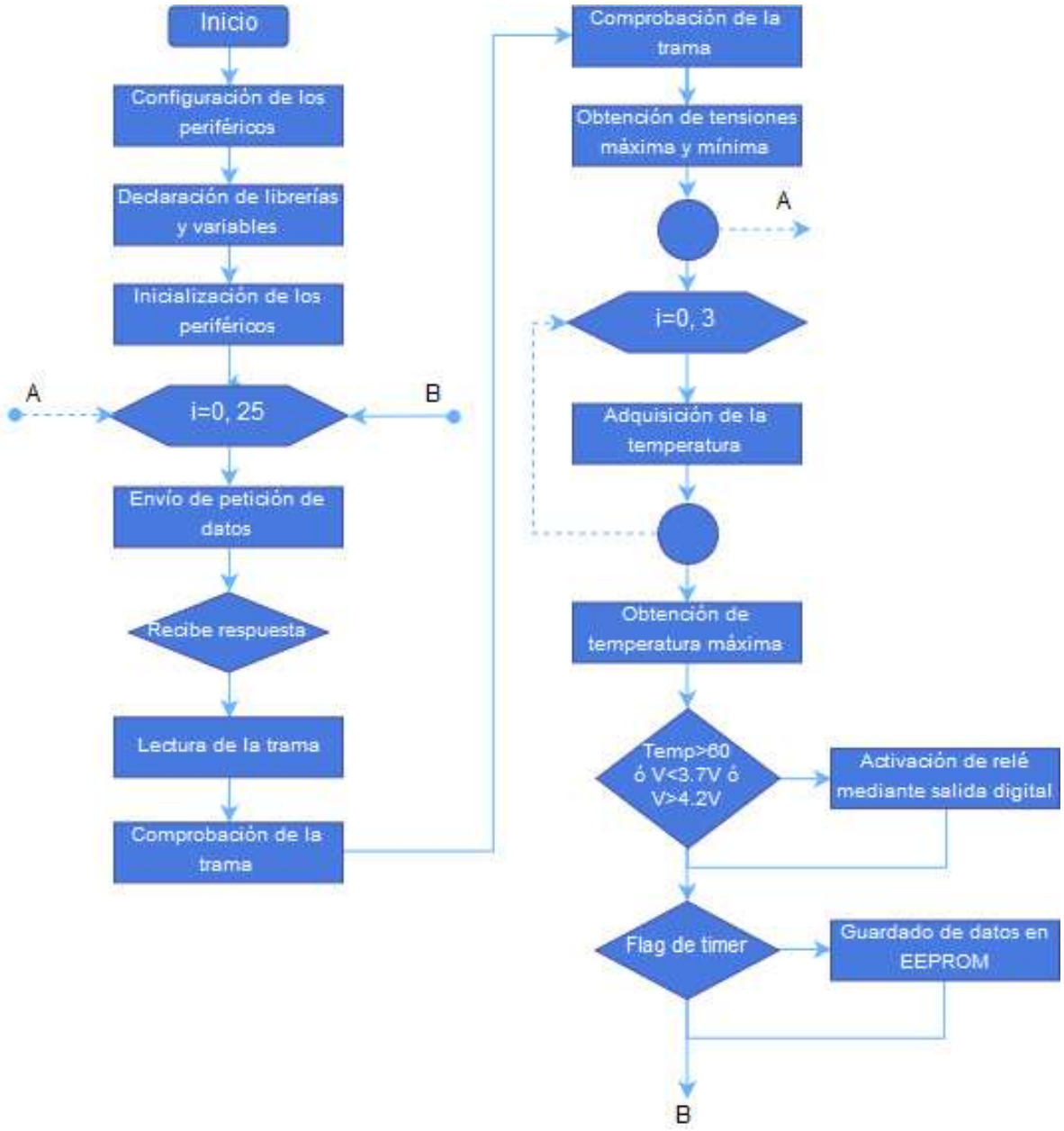
Diciembre, 2018

INDICE

1. Introducción

El objeto de este documento es la exposición de la programación empleada para la tarjeta maestra del sistema BMS. Previo al desarrollo del programa, se expone un flujograma del programa principal que realizará el controlador, y posteriormente se divirá la programación en apartados, exponiendo la configuración de cada uno de los módulos empleados en esta solución. Finalmente, se expondrá la programación principal mediante código.

2. Flujograma



3. Configuración ADC²

El módulo ADC ha sido configurado para usar como reloj el oscilador interno con un postescalado de 1/16. Ello se traduce en un tiempo de conversión de 11.5µs. El alineamiento del resultado es a la izquierda, y la referencia positiva será tomada del módulo FVR. Se habilitan como pines de entrada analógica RA1 (ANA1), RA2 (ANA2) y RA3 (ANA3). Además se podrá consultar el sensor de temperatura interno.

3.1. Adcc.h

```
#ifndef _ADCC_H
#define _ADCC_H

/**
 Sección: Includes
*/

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

/**
 *  Resultado de conversión A/D
 */

typedef uint16_t adc_result_t;
#ifndef int24_t
typedef signed short long int int24_t;
#endif

/** ADC Channel Definition

 @Summary
   Define los canales disponibles para la conversión.
 @Description
   Esta rutina define los canales que están disponibles para usar por el módulo.
 Remarks:
   None
 */

typedef enum
{
    channel_ANA1 =  0x1,
    channel_ANA2 =  0x2,
    channel_ANA3 =  0x3,
    channel_VBAT_div_3 =  0x39,
    channel_VLCD3_div_4 =  0x3A,
    channel_Vss =  0x3B,
    channel_Temp_Sensor =  0x3C,
    channel_DAC1_Output =  0x3D,
    channel_FVR_Buffer1 =  0x3E,
    channel_FVR_Buffer2 =  0x3F
} adcc_channel_t;

/**
```

```
  Section: ADC Module APIs
*/

/**
  @Summary
    Inicializa el ADCC.

  @Description
    Esta rutina inicializa el ADCC.
    Esta rutina debe ser llamado antes que cualquier otra rutina ADCC.
    Esta rutina debe ser sólo llamada una vez durante la inicialización del sistema.
  @Preconditions
    None
  @Param
    None
  @Returns
    None
  @Comment
 */
void ADCC_Initialize(void);

/**
  @Summary
    Permite la selección de canal para la conversión
  @Description
    Esta rutina es usada para seleccionnar el canal deseado para la conversión.
  @Preconditions
    ADCC_Initialize() debe haber sido llamada antes de llamar a esta función.
  @Returns
    None
  @Param
    Pasar el número del canal requerido
    "Para ver canales disponibles refiérase al enum adcc_channel_t"
*/
void ADCC_StartConversion(adcc_channel_t channel);

/**
  @Summary
    Devuelve 'true' cuando la conversión está completa, falso en otro caso.
  @Description
    Esta rutina es usada para determinar si la conversión está completada.
    Cuando la conversión está completa la rutina devuelve true. False en cualquier otro caso.
  @Preconditions
    ADCC_Initialize() y ADCC_StartConversion(adcc_channel_t channel) deben haber sido
llamados previamente.
  @Returns
    true  - Si la conversión está completa
    false - Si la conversión no está completa
  @Param
    None
 */
bool ADCC_IsConversionDone();

/**
  @Summary
    Devuelve el valor de conversión del ADCC.
  @Description
    Esta rutina es usada para obtener el valor de conversión de analógico a digital. Esta rutina
obtiene los valores de conversión del canal especificado.
  @Preconditions
```

```
    Esta rutina devuelve el valor de conversión sólo despues de completar la misma.
    El estatus de completado puede ser verificado usando la rutina ADCC_IsConversionDone().
  @Returns
    Devuelve el valor convertido.
  @Param
    None
 */
adc_result_t ADCC_GetConversionResult(void);


#endif          //_ADCC_H
/**
 End of File
*/
```

3.2. Adcc.c

```
/**
  Sección: Includes
*/
#include <xc.h>
#include "adcc.h"
#include "mcc.h"
/**
  Section: ADCC APIs
*/
void ADCC_Initialize(void)
{
    // Configuración del ADCC.
    // ADLTH 0;
    ADLTHL = 0x00;
    // ADLTH 0;
    ADLTHH = 0x00;
    // ADUTH 0;
    ADUTHL = 0x00;
    // ADUTH 0;
    ADUTHH = 0x00;
    // ADSTPT 0;
    ADSTPTL = 0x00;
    // ADSTPT 0;
    ADSTPTH = 0x00;
    // ADACC 0;
    ADACCU = 0x00;
    // ADRPT 0;
    ADRPT = 0x00;
    // ADPCH ANA0;
    ADPCH = 0x00;
    // ADACQ 0;
    ADACQL = 0x00;
    // ADACQ 0;
    ADACQH = 0x00;
    // ADCAP Additional uC disabled;
    ADCAP = 0x00;
    // ADPRE 0;
    ADPREL = 0x00;
    // ADPRE 0;
    ADPREH = 0x00;
    // ADDSEN disabled; ADGPOL digital_low; ADIPEN disabled; ADPPOL Vss;
    ADCON1 = 0x00;
    // ADCRS 0; ADMD Basic_mode; ADACLR disabled; ADPSIS FLTR;
```

```
    ADCON2 = 0x00;
    // ADCALC First derivative of Single measurement; ADTMD disabled; ADSOI ADGO not cleared;
    ADCON3 = 0x00;
    // ADMATH registers not updated;
    ADSTAT = 0x00;
    // ADPREF FVR;
    ADREF = 0x03;
    // ADACT disabled;
    ADACT = 0x00;
    // ADCS FOSC/16;
    ADCLK = 0x07;

    // ADGO stop; ADFM right; ADON enabled; ADCS FOSC/ADCLK; ADCONT disabled;
    ADCON0 = 0x84;


}

void ADCC_StartConversion(adcc_channel_t channel)
{
    // Selecciona el canal A/D.
    ADPCH = channel;

    // Activa el módulo ADC
    ADCON0bits.ADON = 1;

    // Inicia la conversión
    ADCON0bits.ADGO = 1;
}

bool ADCC_IsConversionDone()
{
    // Inicia la conversión
    return ((unsigned char)(!ADCON0bits.ADGO));
}

adc_result_t ADCC_GetConversionResult(void)
{
    // Devuelve el resultado
    return ((adc_result_t)((ADRESH << 8) + ADRESL));
}



/**
 End of File
*/
```

4. Configuración EUSART1

El módulo EUSART1 ha sido configurado para tener un ratio de 9600 baudios, con 8 bits de transmisión, 8 bits de recepción y polaridad de reloj no invertida.

4.1. Eusart.h

```
#ifndef _EUSART1_H
#define _EUSART1_H
```

```
/**
 Sección: Includes
*/

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

/**
  Sección: Macros
*/

#define EUSART1_DataReady  (eusart1RxCount)

/**
 Sección: Variables Globales
 */
extern volatile uint8_t eusart1TxBufferRemaining;
extern volatile uint8_t eusart1RxCount;

/**
  Sección: EUSART1 APIs
*/

/**
  @Summary
    Rutina de inicialización que toma las entradas de la configuración deseada.
  @Description
    Esta rutina inicializa el driver de EUSART1.
    Esta rutina debe ser llamada antes de cualquier otra rutina de EUSART1.
  @Preconditions
    Non
  @Param
    None
  @Returns
    None
*/
void EUSART1_Initialize(void);
```

```
/**
  @Summary
    Lee un byte de datos desde el EUSART1.
  @Description
    Esta rutina lee un byte de datos desde el EUSART1.
  @Preconditions
    EUSART1_Initialize() debe haber sido llamada antes de esta función. El estatus de
transferencia debe ser revisado para ver si el recepto no está vación antes de llamar a esta función.
            EUSART1_DataReady es una macro que verifica si algún byte es recibido. Llamar a
esta macro antes de usar esta función.
  @Param
    None
  @Returns
    Un byte de datos recibido por el driver.
*/
uint8_t EUSART1_Read(void);

 /**
  @Summary
    Escribe un byte de datos en el EUSART1.
  @Description
    Esta rutina escribe un byte de datos en el EUSART1.
  @Preconditions
    EUSART1_Initialize() debe haber sido llamada antes de llamar a esta función. El estatus de
transferencia debe ser revisado para ver si el transmisor no está ocupado antes de llamar a esta
función.
  @Param
    txData  - Byte de datos para escribir en EUSART1
  @Returns
    None
*/
void EUSART1_Write(uint8_t txData);

/**
  @Summary
    Mantiene la maquina de estado del receptor del driver e implementa su ISR.
  @Description
```

```
    Esta rutina es usada para mantener la máquina de estado del receptor interno del driver.Esta
rutina de interrupción es llamada cuando el estado del receptor necesita ser mantenido.
    @Preconditions
     EUSART1_Initialize() debe haber sido llamada para una ejecución correcta del ISR.
    @Param
     None
    @Returns
     None
*/
void EUSART1_Receive_ISR(void);

#endif  // _EUSART1_H
/**
 End of File
*/
```

4.2. Eusart.c

```
**
  Sección: Includes
 */
#include "mcc.h"
#include "eusart1.h"
/**
  Sección: Macros
*/
#define EUSART1_TX_BUFFER_SIZE 8
#define EUSART1_RX_BUFFER_SIZE 8
/**
  Sección: Variables Globales
*/
volatile uint8_t eusart1TxHead = 0;
volatile uint8_t eusart1TxTail = 0;
volatile uint8_t eusart1TxBuffer[EUSART1_TX_BUFFER_SIZE];
volatile uint8_t eusart1TxBufferRemaining;

volatile uint8_t eusart1RxHead = 0;
volatile uint8_t eusart1RxTail = 0;
volatile uint8_t eusart1RxBuffer[EUSART1_RX_BUFFER_SIZE];
volatile uint8_t eusart1RxCount;
/**
```

```
  Sección: EUSART1 APIs
*/


void EUSART1_Initialize(void){
    // Desactiva las interrupciones antes de cambiar estados.
    PIE3bits.RC1IE = 0;
    PIE3bits.TX1IE = 0;

    // Configura el módulo EUSART1 para las opciones seleccionadas para nuestra aplicación.

    // ABDOVF no_overflow; SCKP Non-Inverted; BRG16 16bit_generator; WUE disabled; ABDEN disabled;
    BAUD1CON = 0x08;

    // SPEN enabled; RX9 8-bit; CREN enabled; ADDEN disabled; SREN disabled;
    RC1STA = 0x90;

    // TX9 8-bit; TX9D 0; SENDB sync_break_complete; TXEN enabled; SYNC asynchronous; BRGH hi_speed; CSRC slave;
    TX1STA = 0x24;

    // Baud Rate = 9600; SP1BRGL 160;
    SP1BRGL = 0xA0;

    // Baud Rate = 9600; SP1BRGH 1;
    SP1BRGH = 0x01;

    // Inicialización del estado del driver
    eusart1TxHead = 0;
    eusart1TxTail = 0;
    eusart1TxBufferRemaining = sizeof(eusart1TxBuffer);

    eusart1RxHead = 0;
    eusart1RxTail = 0;
    eusart1RxCount = 0;

    // Habilita la interrupcion de recepción
    PIE3bits.RC1IE = 1;
}
```

```
uint8_t EUSART1_Read(void){
    uint8_t readValue  = 0;

    while(0 == eusart1RxCount);
    readValue = eusart1RxBuffer[eusart1RxTail++];
    if(sizeof(eusart1RxBuffer) <= eusart1RxTail){
        eusart1RxTail = 0;
    }
    PIE3bits.RC1IE = 0;
    eusart1RxCount--;
    PIE3bits.RC1IE = 1;

    return readValue;
}

void EUSART1_Write(uint8_t txData){
    while(0 == eusart1TxBufferRemaining);
    if(0 == PIE3bits.TX1IE)
        TX1REG = txData;
    else{
        PIE3bits.TX1IE = 0;
        eusart1TxBuffer[eusart1TxHead++] = txData;
        if(sizeof(eusart1TxBuffer) <= eusart1TxHead){
            eusart1TxHead = 0;
        }
        eusart1TxBufferRemaining--;
    }
    PIE3bits.TX1IE = 1;
}

void EUSART1_Receive_ISR(void){
    rx1=true;
    if(1 == RC1STAbits.OERR){
        // EUSART1 error - restart

        RC1STAbits.CREN = 0;
        RC1STAbits.CREN = 1;
    }
```

```
        // buffer overruns are ignored
        eusart1RxBuffer[eusart1RxHead++] = RC1REG;
        if(sizeof(eusart1RxBuffer) <= eusart1RxHead)
            eusart1RxHead = 0;
        eusart1RxCount++;
    }
    /**
      End of File
    */
```

5. Configuración EUSART2

El módulo EUSART2 comparte los mismos parámetros de configuración que el módulo EUSART1, y sus funciones serán homólogas por lo que por simplicidad se omitirán las funciones del mismo. Las funciones que atañen a este módulo tendrán como cabecera en todas sus funciones el identificador EUSART2.

6. Configuración FVR

El módulo FVR ha sido configurado para que el módulo ADCC se sirva de él como tensión de referencia positiva, configurándose el mismo con una ganancia de 1x, lo que se traducirá en una tensión de referencia de 1.024V.

6.1. Fvr.h

```
#ifndef _FVR_H
#define _FVR_H

/**
 Sección: Includes
*/

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

/**
  Sección: FVR APIs
*/

/**
  @Summary
```

```
    Inicializa el FVR

  @Description
    Esta rutina inicializa el FVR.
    Esta rutina debe ser llamada antes que cualquier otra rutina FVR.
    Esta rutina debe ser llamada sólo una vez duranta la inicialización del sistema.
  @Preconditions
    None
  @Param
    None
  @Returns
    None
*/
 void FVR_Initialize(void);

#endif // _FVR_H
/**
 End of File
*/
```

6.2. Fvr.c

```
/**
  Sección: Includes
*/
#include <xc.h>
#include "fvr.h"
/**
  Sección: FVR APIs
*/
void FVR_Initialize(void){
    // CDAFVR off; FVREN enabled; TSRNG Lo_range; ADFVR 1x; TSEN disabled;
    FVRCON = 0x81;
}
/**
 End of File
*/
```

7. Configuración Interrupciones

Las interrupciones han sido configuradas para tener habilitada la gestión de la interrupción de los módulos TMR0, EUSART1 Y EUSART2, dándole prioridad a la comunicación con la interfaz HMI (EUSART2), después a la comunicación entre dispositivos del BMS (EUSART1) y finalmente al Timer 0 en tercer lugar.

7.1. Interrupt_manager.h

```
#ifndef INTERRUPT_MANAGER_H
#define INTERRUPT_MANAGER_H

/**
 * @Param
   none
 * @Returns
   none
 * @Description
   Esta macro habilitará las interrupciones globales.
 */
#define INTERRUPT_GlobalInterruptEnable() (INTCONbits.GIE = 1)

/**
 * @Param
   none
 * @Returns
   none
 * @Description
   Esta macro deshabilitará las interrupciones globales.
 */
#define INTERRUPT_GlobalInterruptDisable() (INTCONbits.GIE = 0)

/**
 * @Param
   none
 * @Returns
   none
 * @Description
   Esta macro habilitará las interrupciones de periféricos.
```

```
 */
#define INTERRUPT_PeripheralInterruptEnable() (INTCONbits.PEIE = 1)

/**
 * @Param
   none
 * @Returns
   none
 * @Description
   Esta macro deshabilitará las interrupciones de periféricos.
 */
#define INTERRUPT_PeripheralInterruptDisable() (INTCONbits.PEIE = 0)

/**
 * @Param
   none
 * @Returns
   none
 * @Description
   Rutina principal de servicio a la interrupción. Llama a los handlers de las interrupciones de los
módulos
 * @Example
   INTERRUPT_InterruptManager();
 */
void interrupt INTERRUPT_InterruptManager(void);


#endif  // INTERRUPT_MANAGER_H
/**
 End of File
*/
```

7.2. Interrupt_manager.c

```
/**
  Sección: Includes
*/

#include <xc.h>
#include "memory.h"
```

```
/**
  Sección: Módulo Data EEPROM APIs
*/

void DATAEE_WriteByte(uint16_t bAdd, uint8_t bData){
    uint8_t GIEBitValue = INTCONbits.GIE;

    NVMADRH = ((bAdd >> 8) & 0xFF);
    NVMADRL = (bAdd & 0xFF);
    NVMDATL = bData;
    NVMCON1bits.NVMREGS = 0;
    NVMCON1bits.WREN = 1;
    INTCONbits.GIE = 0;     // Deshabilita interrupciones
    NVMCON2 = 0x55;
    NVMCON2 = 0xAA;
    NVMCON1bits.WR = 1;
    // Wait for write to complete
    while (NVMCON1bits.WR);

    NVMCON1bits.WREN = 0;
    INTCONbits.GIE = GIEBitValue;   // restablece habilitación de interrupciones
}

uint8_t DATAEE_ReadByte(uint16_t bAdd){
    NVMADRH = ((bAdd >> 8) & 0xFF);
    NVMADRL = (bAdd & 0xFF);
    NVMCON1bits.NVMREGS = 0;
    NVMCON1bits.RD = 1;
    NOP();  // NOPs puede ser requerido para latencias a altas frecuencias
    NOP();

    return (NVMDATL);
}

/**
 End of File
*/
```

8. Configuración Pineado

El pineado ha sido configurado para habilitar RA1, RA2 y RA3 como entradas analógicas, siendo identificados por el módulo ADCC, como ANA1, ANA2 y ANA3 respectivamente. Se han configurado como salidas digitales RB0, RB1, RB4 y RC1. Para el módulo EUSART1 se han habilitado las patillas RC2 como patilla de transmisión y RC3 como patilla de recepción. Para el módulo EUSART2, se han habilitado RB2 para la transmisión y RB3 para recepción.

8.1. Pin_manager.h

```
#ifndef PIN_MANAGER_H
#define PIN_MANAGER_H

#define INPUT   1
#define OUTPUT  0

#define HIGH    1
#define LOW     0

#define ANALOG      1
#define DIGITAL     0

#define PULL_UP_ENABLED      1
#define PULL_UP_DISABLED     0

// Configura alias de channel_ANA1
#define channel_ANA1_TRIS               TRISAbits.TRISA1
#define channel_ANA1_LAT                LATAbits.LATA1
#define channel_ANA1_PORT               PORTAbits.RA1
#define channel_ANA1_WPU                WPUAbits.WPUA1
#define channel_ANA1_OD                 ODCONAbits.ODCA1
#define channel_ANA1_ANS                ANSELAbits.ANSA1
#define channel_ANA1_SetHigh()          do { LATAbits.LATA1 = 1; } while(0)
#define channel_ANA1_SetLow()           do { LATAbits.LATA1 = 0; } while(0)
#define channel_ANA1_Toggle()           do { LATAbits.LATA1 = ~LATAbits.LATA1; } while(0)
#define channel_ANA1_GetValue()         PORTAbits.RA1
#define channel_ANA1_SetDigitalInput()    do { TRISAbits.TRISA1 = 1; } while(0)
#define channel_ANA1_SetDigitalOutput()   do { TRISAbits.TRISA1 = 0; } while(0)
#define channel_ANA1_SetPullup()      do { WPUAbits.WPUA1 = 1; } while(0)
```

```
#define channel_ANA1_ResetPullup()    do { WPUAbits.WPUA1 = 0; } while(0)
#define channel_ANA1_SetPushPull()    do { ODCONAbits.ODCA1 = 1; } while(0)
#define channel_ANA1_SetOpenDrain()   do { ODCONAbits.ODCA1 = 0; } while(0)
#define channel_ANA1_SetAnalogMode()  do { ANSELAbits.ANSA1 = 1; } while(0)
#define channel_ANA1_SetDigitalMode() do { ANSELAbits.ANSA1 = 0; } while(0)

// Configura alias de channel_ANA2
#define channel_ANA2_TRIS              TRISAbits.TRISA2
#define channel_ANA2_LAT               LATAbits.LATA2
#define channel_ANA2_PORT              PORTAbits.RA2
#define channel_ANA2_WPU               WPUAbits.WPUA2
#define channel_ANA2_OD                ODCONAbits.ODCA2
#define channel_ANA2_ANS               ANSELAbits.ANSA2
#define channel_ANA2_SetHigh()         do { LATAbits.LATA2 = 1; } while(0)
#define channel_ANA2_SetLow()          do { LATAbits.LATA2 = 0; } while(0)
#define channel_ANA2_Toggle()          do { LATAbits.LATA2 = ~LATAbits.LATA2; } while(0)
#define channel_ANA2_GetValue()        PORTAbits.RA2
#define channel_ANA2_SetDigitalInput()    do { TRISAbits.TRISA2 = 1; } while(0)
#define channel_ANA2_SetDigitalOutput()   do { TRISAbits.TRISA2 = 0; } while(0)
#define channel_ANA2_SetPullup()      do { WPUAbits.WPUA2 = 1; } while(0)
#define channel_ANA2_ResetPullup()    do { WPUAbits.WPUA2 = 0; } while(0)
#define channel_ANA2_SetPushPull()    do { ODCONAbits.ODCA2 = 1; } while(0)
#define channel_ANA2_SetOpenDrain()   do { ODCONAbits.ODCA2 = 0; } while(0)
#define channel_ANA2_SetAnalogMode()  do { ANSELAbits.ANSA2 = 1; } while(0)
#define channel_ANA2_SetDigitalMode() do { ANSELAbits.ANSA2 = 0; } while(0)

// Configura alias de channel_ANA3
#define channel_ANA3_TRIS              TRISAbits.TRISA3
#define channel_ANA3_LAT               LATAbits.LATA3
#define channel_ANA3_PORT              PORTAbits.RA3
#define channel_ANA3_WPU               WPUAbits.WPUA3
#define channel_ANA3_OD                ODCONAbits.ODCA3
#define channel_ANA3_ANS               ANSELAbits.ANSA3
#define channel_ANA3_SetHigh()         do { LATAbits.LATA3 = 1; } while(0)
#define channel_ANA3_SetLow()          do { LATAbits.LATA3 = 0; } while(0)
#define channel_ANA3_Toggle()          do { LATAbits.LATA3 = ~LATAbits.LATA3; } while(0)
#define channel_ANA3_GetValue()        PORTAbits.RA3
#define channel_ANA3_SetDigitalInput()    do { TRISAbits.TRISA3 = 1; } while(0)
#define channel_ANA3_SetDigitalOutput()   do { TRISAbits.TRISA3 = 0; } while(0)
#define channel_ANA3_SetPullup()      do { WPUAbits.WPUA3 = 1; } while(0)
```

```
#define channel_ANA3_ResetPullup()    do { WPUAbits.WPUA3 = 0; } while(0)
#define channel_ANA3_SetPushPull()    do { ODCONAbits.ODCA3 = 1; } while(0)
#define channel_ANA3_SetOpenDrain()   do { ODCONAbits.ODCA3 = 0; } while(0)
#define channel_ANA3_SetAnalogMode()  do { ANSELAbits.ANSA3 = 1; } while(0)
#define channel_ANA3_SetDigitalMode() do { ANSELAbits.ANSA3 = 0; } while(0)

// Configura alias de IO_RA6
#define IO_RA6_TRIS            TRISAbits.TRISA6
#define IO_RA6_LAT             LATAbits.LATA6
#define IO_RA6_PORT            PORTAbits.RA6
#define IO_RA6_WPU             WPUAbits.WPUA6
#define IO_RA6_OD              ODCONAbits.ODCA6
#define IO_RA6_ANS             ANSELAbits.ANSA6
#define IO_RA6_SetHigh()          do { LATAbits.LATA6 = 1; } while(0)
#define IO_RA6_SetLow()           do { LATAbits.LATA6 = 0; } while(0)
#define IO_RA6_Toggle()           do { LATAbits.LATA6 = ~LATAbits.LATA6; } while(0)
#define IO_RA6_GetValue()         PORTAbits.RA6
#define IO_RA6_SetDigitalInput()    do { TRISAbits.TRISA6 = 1; } while(0)
#define IO_RA6_SetDigitalOutput()   do { TRISAbits.TRISA6 = 0; } while(0)
#define IO_RA6_SetPullup()      do { WPUAbits.WPUA6 = 1; } while(0)
#define IO_RA6_ResetPullup()    do { WPUAbits.WPUA6 = 0; } while(0)
#define IO_RA6_SetPushPull()    do { ODCONAbits.ODCA6 = 1; } while(0)
#define IO_RA6_SetOpenDrain()   do { ODCONAbits.ODCA6 = 0; } while(0)
#define IO_RA6_SetAnalogMode()  do { ANSELAbits.ANSA6 = 1; } while(0)
#define IO_RA6_SetDigitalMode() do { ANSELAbits.ANSA6 = 0; } while(0)

// Configura alias de IO_RB0
#define IO_RB0_TRIS            TRISBbits.TRISB0
#define IO_RB0_LAT             LATBbits.LATB0
#define IO_RB0_PORT            PORTBbits.RB0
#define IO_RB0_WPU             WPUBbits.WPUB0
#define IO_RB0_OD              ODCONBbits.ODCB0
#define IO_RB0_ANS             ANSELBbits.ANSB0
#define IO_RB0_SetHigh()          do { LATBbits.LATB0 = 1; } while(0)
#define IO_RB0_SetLow()           do { LATBbits.LATB0 = 0; } while(0)
#define IO_RB0_Toggle()           do { LATBbits.LATB0 = ~LATBbits.LATB0; } while(0)
#define IO_RB0_GetValue()         PORTBbits.RB0
#define IO_RB0_SetDigitalInput()    do { TRISBbits.TRISB0 = 1; } while(0)
#define IO_RB0_SetDigitalOutput()   do { TRISBbits.TRISB0 = 0; } while(0)
#define IO_RB0_SetPullup()      do { WPUBbits.WPUB0 = 1; } while(0)
```

```
#define IO_RB0_ResetPullup()    do { WPUBbits.WPUB0 = 0; } while(0)
#define IO_RB0_SetPushPull()    do { ODCONBbits.ODCB0 = 1; } while(0)
#define IO_RB0_SetOpenDrain()   do { ODCONBbits.ODCB0 = 0; } while(0)
#define IO_RB0_SetAnalogMode()  do { ANSELBbits.ANSB0 = 1; } while(0)
#define IO_RB0_SetDigitalMode() do { ANSELBbits.ANSB0 = 0; } while(0)

// Configura alias de IO_RB1
#define IO_RB1_TRIS              TRISBbits.TRISB1
#define IO_RB1_LAT               LATBbits.LATB1
#define IO_RB1_PORT              PORTBbits.RB1
#define IO_RB1_WPU               WPUBbits.WPUB1
#define IO_RB1_OD                ODCONBbits.ODCB1
#define IO_RB1_ANS               ANSELBbits.ANSB1
#define IO_RB1_SetHigh()           do { LATBbits.LATB1 = 1; } while(0)
#define IO_RB1_SetLow()            do { LATBbits.LATB1 = 0; } while(0)
#define IO_RB1_Toggle()            do { LATBbits.LATB1 = ~LATBbits.LATB1; } while(0)
#define IO_RB1_GetValue()          PORTBbits.RB1
#define IO_RB1_SetDigitalInput()    do { TRISBbits.TRISB1 = 1; } while(0)
#define IO_RB1_SetDigitalOutput()   do { TRISBbits.TRISB1 = 0; } while(0)
#define IO_RB1_SetPullup()      do { WPUBbits.WPUB1 = 1; } while(0)
#define IO_RB1_ResetPullup()    do { WPUBbits.WPUB1 = 0; } while(0)
#define IO_RB1_SetPushPull()    do { ODCONBbits.ODCB1 = 1; } while(0)
#define IO_RB1_SetOpenDrain()   do { ODCONBbits.ODCB1 = 0; } while(0)
#define IO_RB1_SetAnalogMode()  do { ANSELBbits.ANSB1 = 1; } while(0)
#define IO_RB1_SetDigitalMode() do { ANSELBbits.ANSB1 = 0; } while(0)

// Configura procedimientos de IO_RB2
#define RB2_SetHigh()    do { LATBbits.LATB2 = 1; } while(0)
#define RB2_SetLow()   do { LATBbits.LATB2 = 0; } while(0)
#define RB2_Toggle()   do { LATBbits.LATB2 = ~LATBbits.LATB2; } while(0)
#define RB2_GetValue()         PORTBbits.RB2
#define RB2_SetDigitalInput()   do { TRISBbits.TRISB2 = 1; } while(0)
#define RB2_SetDigitalOutput()  do { TRISBbits.TRISB2 = 0; } while(0)
#define RB2_SetPullup()     do { WPUBbits.WPUB2 = 1; } while(0)
#define RB2_ResetPullup()   do { WPUBbits.WPUB2 = 0; } while(0)
#define RB2_SetAnalogMode() do { ANSELBbits.ANSB2 = 1; } while(0)
#define RB2_SetDigitalMode()do { ANSELBbits.ANSB2 = 0; } while(0)

// Configura procedimientos de IO_RB3
#define RB3_SetHigh()    do { LATBbits.LATB3 = 1; } while(0)
```

```
#define RB3_SetLow()   do { LATBbits.LATB3 = 0; } while(0)
#define RB3_Toggle()   do { LATBbits.LATB3 = ~LATBbits.LATB3; } while(0)
#define RB3_GetValue()         PORTBbits.RB3
#define RB3_SetDigitalInput()   do { TRISBbits.TRISB3 = 1; } while(0)
#define RB3_SetDigitalOutput()  do { TRISBbits.TRISB3 = 0; } while(0)
#define RB3_SetPullup()     do { WPUBbits.WPUB3 = 1; } while(0)
#define RB3_ResetPullup()   do { WPUBbits.WPUB3 = 0; } while(0)
#define RB3_SetAnalogMode() do { ANSELBbits.ANSB3 = 1; } while(0)
#define RB3_SetDigitalMode()do { ANSELBbits.ANSB3 = 0; } while(0)

// Configura alias de IO_RB4
#define IO_RB4_TRIS              TRISBbits.TRISB4
#define IO_RB4_LAT               LATBbits.LATB4
#define IO_RB4_PORT               PORTBbits.RB4
#define IO_RB4_WPU               WPUBbits.WPUB4
#define IO_RB4_OD               ODCONBbits.ODCB4
#define IO_RB4_ANS               ANSELBbits.ANSB4
#define IO_RB4_SetHigh()           do { LATBbits.LATB4 = 1; } while(0)
#define IO_RB4_SetLow()            do { LATBbits.LATB4 = 0; } while(0)
#define IO_RB4_Toggle()           do { LATBbits.LATB4 = ~LATBbits.LATB4; } while(0)
#define IO_RB4_GetValue()          PORTBbits.RB4
#define IO_RB4_SetDigitalInput()    do { TRISBbits.TRISB4 = 1; } while(0)
#define IO_RB4_SetDigitalOutput()   do { TRISBbits.TRISB4 = 0; } while(0)
#define IO_RB4_SetPullup()      do { WPUBbits.WPUB4 = 1; } while(0)
#define IO_RB4_ResetPullup()    do { WPUBbits.WPUB4 = 0; } while(0)
#define IO_RB4_SetPushPull()    do { ODCONBbits.ODCB4 = 1; } while(0)
#define IO_RB4_SetOpenDrain()   do { ODCONBbits.ODCB4 = 0; } while(0)
#define IO_RB4_SetAnalogMode()  do { ANSELBbits.ANSB4 = 1; } while(0)
#define IO_RB4_SetDigitalMode() do { ANSELBbits.ANSB4 = 0; } while(0)

// Configura alias de IO_RC1
#define IO_RC1_TRIS              TRISCbits.TRISC1
#define IO_RC1_LAT               LATCbits.LATC1
#define IO_RC1_PORT               PORTCbits.RC1
#define IO_RC1_WPU               WPUCbits.WPUC1
#define IO_RC1_OD               ODCONCbits.ODCC1
#define IO_RC1_ANS               ANSELCbits.ANSC1
#define IO_RC1_SetHigh()           do { LATCbits.LATC1 = 1; } while(0)
#define IO_RC1_SetLow()            do { LATCbits.LATC1 = 0; } while(0)
#define IO_RC1_Toggle()           do { LATCbits.LATC1 = ~LATCbits.LATC1; } while(0)
```

```
#define IO_RC1_GetValue()           PORTCbits.RC1
#define IO_RC1_SetDigitalInput()    do { TRISCbits.TRISC1 = 1; } while(0)
#define IO_RC1_SetDigitalOutput()   do { TRISCbits.TRISC1 = 0; } while(0)
#define IO_RC1_SetPullup()      do { WPUCbits.WPUC1 = 1; } while(0)
#define IO_RC1_ResetPullup()    do { WPUCbits.WPUC1 = 0; } while(0)
#define IO_RC1_SetPushPull()    do { ODCONCbits.ODCC1 = 1; } while(0)
#define IO_RC1_SetOpenDrain()   do { ODCONCbits.ODCC1 = 0; } while(0)
#define IO_RC1_SetAnalogMode()  do { ANSELCbits.ANSC1 = 1; } while(0)
#define IO_RC1_SetDigitalMode() do { ANSELCbits.ANSC1 = 0; } while(0)

// Configura procedimientos de RC2
#define RC2_SetHigh()    do { LATCbits.LATC2 = 1; } while(0)
#define RC2_SetLow()   do { LATCbits.LATC2 = 0; } while(0)
#define RC2_Toggle()   do { LATCbits.LATC2 = ~LATCbits.LATC2; } while(0)
#define RC2_GetValue()         PORTCbits.RC2
#define RC2_SetDigitalInput()   do { TRISCbits.TRISC2 = 1; } while(0)
#define RC2_SetDigitalOutput()  do { TRISCbits.TRISC2 = 0; } while(0)
#define RC2_SetPullup()     do { WPUCbits.WPUC2 = 1; } while(0)
#define RC2_ResetPullup()   do { WPUCbits.WPUC2 = 0; } while(0)
#define RC2_SetAnalogMode() do { ANSELCbits.ANSC2 = 1; } while(0)
#define RC2_SetDigitalMode()do { ANSELCbits.ANSC2 = 0; } while(0)

// Configura procedimientos de RC3
#define RC3_SetHigh()    do { LATCbits.LATC3 = 1; } while(0)
#define RC3_SetLow()   do { LATCbits.LATC3 = 0; } while(0)
#define RC3_Toggle()   do { LATCbits.LATC3 = ~LATCbits.LATC3; } while(0)
#define RC3_GetValue()         PORTCbits.RC3
#define RC3_SetDigitalInput()   do { TRISCbits.TRISC3 = 1; } while(0)
#define RC3_SetDigitalOutput()  do { TRISCbits.TRISC3 = 0; } while(0)
#define RC3_SetPullup()     do { WPUCbits.WPUC3 = 1; } while(0)
#define RC3_ResetPullup()   do { WPUCbits.WPUC3 = 0; } while(0)
#define RC3_SetAnalogMode() do { ANSELCbits.ANSC3 = 1; } while(0)
#define RC3_SetDigitalMode()do { ANSELCbits.ANSC3 = 0; } while(0)

/**
  @Param
    none
  @Returns
    none
  @Description
```

```
    Inicialización de GPIO and periféricos I/O
 */
void PIN_MANAGER_Initialize (void);

#endif // PIN_MANAGER_H
/**
 End of File
*/
```

8.2. Pin_manager.c

```
#include <xc.h>
#include "pin_manager.h"
#include "stdbool.h"

#if (__XC8_VERSION <= 1410)

typedef union {
    struct {
        unsigned T2INPPS                :6;
    };
    struct {
        unsigned T2INPPS0               :1;
        unsigned T2INPPS1               :1;
        unsigned T2INPPS2               :1;
        unsigned T2INPPS3               :1;
        unsigned T2INPPS4               :1;
        unsigned T2INPPS5               :1;
    };
} T2INPPSbits_t;
extern volatile T2INPPSbits_t T2INPPSbits @ (&T2AINPPS);

typedef union {
    struct {
        unsigned T4INPPS                :6;
    };
    struct {
        unsigned T4INPPS0               :1;
        unsigned T4INPPS1               :1;
        unsigned T4INPPS2               :1;
        unsigned T4INPPS3               :1;
```

```
        unsigned T4INPPS4       :1;
        unsigned T4INPPS5       :1;
    };
} T4INPPSbits_t;
extern volatile T4INPPSbits_t T4INPPSbits @ (&T4AINPPS);
#endif

void PIN_MANAGER_Initialize(void){
    /**
    LATx registros
    */
    LATA = 0x00;
    LATB = 0x00;
    LATC = 0x00;

    /**
    TRISx registros
    */
    TRISA = 0x9F;
    TRISB = 0xE8;
    TRISC = 0xD9;

    /**
    ANSELx registros
    */
    ANSELC = 0x91;
    ANSELB = 0xA0;
    ANSELA = 0xDF;

    /**
    WPUx registros
    */
    WPUE = 0x00;
    WPUB = 0x00;
    WPUA = 0x00;
    WPUC = 0x00;

    /**
    ODx registros
    */
```

```
    ODCONA = 0x00;
    ODCONB = 0x00;
    ODCONC = 0x00;


    bool state = (unsigned char)GIE;
    GIE = 0;
    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x00; // desbloquea PPS

    RX2PPSbits.RX2PPS = 0x0B;   //RB3->EUSART2:RX2;
    RB2PPS = 0x0F;   //RB2->EUSART2:TX2;
    RC2PPS = 0x0D;   //RC2->EUSART1:TX1;
    RX1PPSbits.RX1PPS = 0x13;   //RC3->EUSART1:RX1;

    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x01; // bloquea PPS

    GIE = state;
}
/**
 End of File
*/
```

9. Configuración Timer0

El timer ha sido configurado para tener un pre-escalado de 1:32 y un post-escalado de 1:10, todo ello para un reloj seleccionado como el oscilador interno del microcontrolador. El contador de 8 bits nos da una horquilla para seleccionar el periodo comprendida entre 20µs y 5.12ms. El valor seleccionado ha sido de 2ms. También se ha configurado el módulo para contar con una gestión del timer mediante interrupción.

9.1. Tmr0.h

```
#ifndef _TMR0_H
#define _TMR0_H
```

```
/**
 Sección: Includes
*/

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

/**
  Sección: TMR0 APIs
*/
/**
  @Summary
    Inicializa el TMR0.
  @Description
    Esta función inicializa los registros del TMR0.
    Esta función debe ser llamada antes que cualquier otra rutina de TMR0.
  @Preconditions
    None
  @Param
    None
  @Returns
    None
 */
void TMR0_Initialize(void);

/**
  @Summary
    Esta función arranca el TMR0.
  @Description
    Esta función inicia la operatividad del TMR0.
    Esta función debe ser llamada despues de inicializar el TMR0.
  @Preconditions
    Inicaializa el TMR0 antes de llamar a esta función.
  @Param
    None
  @Returns
    None
*/
void TMR0_StartTimer(void);
```

```
/**
  @Summary
    Rutina de servicio a la interrupción del Timer
  @Description
    La rutina de servicio a la interrupción del timer es llamada por la gestión de interrupciones.
  @Preconditions
    Iniciar el módulo TMR0 con interrupción antes de llamar a esta isr.
  @Param
    None
  @Returns
    None
 */
void TMR0_ISR(void);


/**
  @Summary
    Configura el Interrupt Handler del Timer
  @Description
    Configura la función para ser llamada durante la ISR.
  @Preconditions
    Inicializa el TMR0 con interrupción antes de llamar a esta función.
  @Param
    Dirección de la función a configurar
  @Returns
    None
*/
 void TMR0_SetInterruptHandler(void (* InterruptHandler)(void));

/**
  @Summary
    Timer Interrupt Handler
  @Description
    Esta es una función puntero a la función que será llamada durante la ISR
  @Preconditions
    Inicializar el módulo TMR0 con interrupción antes de llamar a esta ISR.
  @Param
    None
  @Returns
    None
```

```
*/
extern void (*TMR0_InterruptHandler)(void);

/**
  @Summary
    Timer Interrupt Handler por defecto
  @Description
    Esta es la función de Interrupt Handler por defecto
  @Preconditions
    Inicializar el módulo TMR0 con interrupción antes de llamar a esta ISR.
  @Param
    None
  @Returns
    None
*/
void TMR0_DefaultInterruptHandler(void);

#endif // _TMR0_H
/**
 End of File
*/
```

9.2. Tmr0.c

```
/**
  Sección: Includes
*/

#include <xc.h>
#include "tmr0.h"
#include "pin_manager.h"
#include "mcc.h"
/**
  Sección: TMR0 APIs
*/

void (*TMR0_InterruptHandler)(void);

void TMR0_Initialize(void){
    // Configura el TMR0 con la configuración de nuestra aplicación
```

```
    // T0OUTPS 1:1; T0EN disabled; T016BIT 8-bit;
    T0CON0 = 0x00;

    // T0CS HFINTOSC; T0CKPS 1:128; T0ASYNC synchronised;
    T0CON1 = 0x67;

    // TMR0H 249;
    TMR0H = 0xF9;

    // TMR0L 0;
    TMR0L = 0x00;

    // Limpia el flag de interrupción antes de inicializar la interrupción
    PIR0bits.TMR0IF = 0;

    // Habilita la interrupción de TMR0.
    PIE0bits.TMR0IE = 1;

    // Configura el Interrupt Handler por defecto
    TMR0_SetInterruptHandler(TMR0_DefaultInterruptHandler);

    // Start TMR0
    TMR0_StartTimer();
}

void TMR0_StartTimer(void){
    // Arranca el TMR0 escribiendo en el bit TMR0ON
    T0CON0bits.T0EN = 1;
}
void TMR0_ISR(void){
    static int reloj=0;
    static int espera=0;
    static int espera2=0;
    // Limpia el flag de interrupción de TMR0
    PIR0bits.TMR0IF = 0;
    if(TMR0_InterruptHandler)
        TMR0_InterruptHandler();
    reloj++;
    if(reloj>1000){
        IO_RC1_Toggle();
```

```
            reloj=0;
            espera2++;
            if(corte)
                espera++;
        }
        if(espera==3){
           corte=false;
           espera=0;
        }
        if(espera2>=2000){
            espera2=0;
            guardar=true;
        }
    }
    void TMR0_SetInterruptHandler(void (* InterruptHandler)(void)){
        TMR0_InterruptHandler = InterruptHandler;
    }
    void TMR0_DefaultInterruptHandler(void);

    /**
      End of File
    */
```

10. Configuración de EEPROM

En este caso, no se precisa de ninguna configuración adicional, por lo tanto lo expuesto en este apartado se limita simplemente a las funciones de escritura y lectura de esta zona particular de memoria, empleada con el fin de obtener un registro remanente de la información procesada.

10.1. memory.h

```
#ifndef _MEMORY_H
#define _MEMORY_H

/**
  Section: Included Files
*/
```

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

/**
  Section: Data EEPROM Module APIs
*/

/**
  @Summary
    Escribe un byte de datos en Data EEPROM
  @Description
    Esta rutina escribe un byte de datos para la localización dada de Data EEPROM
  @Preconditions
    None

  @Param
    bAdd  - Localización de Data EEPROM en la que los datos serán escritos
    bData - Datos para ser escrito es la dirección de Data EEPROM
  @Returns
    None
*/
void DATAEE_WriteByte(uint16_t bAdd, uint8_t bData);



/**
  @Summary
    Lee un byte de datos desde la Data EEPROM
  @Description
    Esta rutina lee un byte de datos de la localización de Data EEPROM dada
  @Preconditions
    None
  @Param
    bAdd  - Localización de Data EEPROM desde la que los datos tienen que ser leídos
  @Returns
    Byte de datos leído de la localización de Data EEPROM dada
*/
uint8_t DATAEE_ReadByte(uint16_t bAdd);

#endif // _MEMORY_H
```

```
/**
 End of File
*/
```

10.2. memory.c

```
/**
  Sección: Includes
*/
#include <xc.h>
#include "memory.h"
/**
  Sección: Módulo Data EEPROM APIs
*/

void DATAEE_WriteByte(uint16_t bAdd, uint8_t bData){
    uint8_t GIEBitValue = INTCONbits.GIE;

    NVMADRH = ((bAdd >> 8) & 0xFF);
    NVMADRL = (bAdd & 0xFF);
    NVMDATL = bData;
    NVMCON1bits.NVMREGS = 0;
    NVMCON1bits.WREN = 1;
    INTCONbits.GIE = 0;     // Deshabilita interrupciones
    NVMCON2 = 0x55;
    NVMCON2 = 0xAA;
    NVMCON1bits.WR = 1;
    // Wait for write to complete
    while (NVMCON1bits.WR);
    NVMCON1bits.WREN = 0;
    INTCONbits.GIE = GIEBitValue;   // restablece habilitación de interrupciones
}

uint8_t DATAEE_ReadByte(uint16_t bAdd){
    NVMADRH = ((bAdd >> 8) & 0xFF);
    NVMADRL = (bAdd & 0xFF);
    NVMCON1bits.NVMREGS = 0;
    NVMCON1bits.RD = 1;
    NOP();  // NOPs puede ser requerido para latencias a altas frecuencias
    NOP();
```

```
    return (NVMDATL);
}

/**
 End of File
*/
```

11. Configuración Sistema

El sistema se ha configurado para seguir al reloj interno, configurado para una frecuencia de 16 MHz, sin ningún divisor adicional.

11.1. mcc.h

```
#ifndef MCC_H
#define        MCC_H
#include <xc.h>
#include "pin_manager.h"
#include <stdint.h>
#include <stdbool.h>
#include "interrupt_manager.h"
#include "memory.h"
#include "eusart1.h"
#include "tmr0.h"
#include "eusart2.h"
#include "fvr.h"
#include "adcc.h"

#define _XTAL_FREQ  16000000
bool rx1=false;
bool rx2=false;
bool guardar= false;
char orden;
bool corte=false;

/**
 * @Param
    none
 * @Returns
```

```
    none
 * @Description
    Inicializa el dispositivo para los estados configurados
 */
void SYSTEM_Initialize(void);

/**
 * @Param
    none
 * @Returns
    none
 * @Description
    Inicializa el oscilador a los estados configurados
 */
void OSCILLATOR_Initialize(void);

#endif          /* MCC_H */
/**
 End of File
*/
```

11.2. mcc.c

```
// Bits de configuración

// CONFIG1
#pragma config FEXTOSC = OFF    // External Oscillator mode selection bits->Oscillator not enabled
#pragma config RSTOSC = HFINT1    // Power-up default value for COSC bits->HFINTOSC (1MHz)
#pragma config CLKOUTEN = OFF     // Clock Out Enable bit->CLKOUT function is disabled; i/o or oscillator function on OSC2
#pragma config VBATEN = OFF     // VBAT Pin Enable bit->VBAT functionality is disabled
#pragma config LCDPEN = ON    // LCD Charge Pump Mode bit->LCD Charge Pump is enabled
#pragma config CSWEN = ON     // Clock Switch Enable bit->Writing to NOSC and NDIV is allowed
#pragma config FCMEN = ON     // Fail-Safe Clock Monitor Enable bit->FSCM timer enabled

// CONFIG2
#pragma config MCLRE = ON     // Master Clear Enable bit->MCLR pin is Master Clear function
#pragma config PWRTE = OFF     // Power-up Timer selection bits->PWRT disable
#pragma config LPBOREN = OFF     // Low-Power BOR enable bit->ULPBOR disabled
#pragma config BOREN = ON    // Brown-out reset enable bits->Brown-out Reset Enabled, SBOREN bit is ignored
```

```
#pragma config BORV = LO    // Brown-out Reset Voltage Selection->Brown-out Reset Voltage (VBOR) set to 1.9V on LF, and 2.45V on F Devices
#pragma config ZCD = OFF    // Zero-cross detect disable->Zero-cross detect circuit is disabled at POR.
#pragma config PPS1WAY = ON    // Peripheral Pin Select one-way control->The PPSLOCK bit can be cleared and set only once in software
#pragma config STVREN = ON    // Stack Overflow/Underflow Reset Enable bit->Stack Overflow or Underflow will cause a reset

// CONFIG3
#pragma config WDTCPS = WDTCPS_31    // WDT Period Select bits->Divider ratio 1:65536; software control of WDTPS
#pragma config WDTE = OFF    // WDT operating mode->WDT Disabled, SWDTEN is ignored
#pragma config WDTCWS = WDTCWS_7    // WDT Window Select bits->window always open (100%); software control; keyed access not required
#pragma config WDTCCS = HFINTOSC    // WDT input clock selector->WDT reference clock is the 31.25 kHz HFINTOSC

// CONFIG4
#pragma config BBSIZE = 512    // Boot Block Size Selection bits->Boot Block Size (Words) 512
#pragma config BBEN = OFF    // Boot Block Enable bit->Boot Block disabled
#pragma config SAFEN = OFF    // SAF Enable bit->SAF disabled
#pragma config WRTAPP = OFF    // Application Block Write Protection bit->Application Block NOT write-protected
#pragma config WRTB = OFF    // Boot Block Write Protection bit->Boot Block NOT write-protected
#pragma config WRTC = OFF    // Configuration Register Write Protection bit->Configuration Words NOT write-protected
#pragma config WRTD = OFF    // Data EEPROM Write Protection bit->Data EEPROM NOT write-protected
#pragma config WRTSAF = OFF    // Storage Area Flash Write Protection bit->SAF NOT write-protected
#pragma config LVP = ON    // Low Voltage Programming Enable bit->Low Voltage programming enabled. MCLR/Vpp pin function is MCLR.

// CONFIG5
#pragma config CP = OFF    // UserNVM Program memory code protection bit->UserNVM code protection disabled

#include "mcc.h"
```

```
void SYSTEM_Initialize(void){

    PIN_MANAGER_Initialize();
    OSCILLATOR_Initialize();
    FVR_Initialize();
    ADCC_Initialize();
    TMR0_Initialize();
    EUSART1_Initialize();
    EUSART2_Initialize();
}

void OSCILLATOR_Initialize(void){
    // NOSC HFINTOSC; NDIV 1;
    OSCCON1 = 0x60;
    // CSWHOLD may proceed; SOSCPWR Low power;
    OSCCON3 = 0x00;
    // MFOEN disabled; LFOEN disabled; ADOEN disabled; SOSCEN disabled; EXTOEN disabled;
HFOEN disabled;
    OSCEN = 0x00;
    // HFFRQ 16_MHz;
    OSCFRQ = 0x05;
    // MFOR not ready;
    OSCSTAT = 0x00;
    // TUN 0;
    OSCTUNE = 0x00;
}
/**
 End of File
*/
```

12. Main

```
#include "mcc_generated_files/mcc.h"


//VARIABLES GLOBALES

uint16_t tensiones[26];

uint16_t temperaturas[3];

uint16_t aux[4];
```

```
float grados[3];
uint16_t temp_max,v_min,v_max;
float temp_max2;
uint8_t k;
uint8_t intentos=0;
uint8_t lectura;
uint16_t PWM;
uint16_t adress=0xF000;
uint16_t adress2=0xF000;
uint16_t word;
/*
                    Main application
 */
uint16_t LeeTemperatura(uint8_t canal);
uint16_t FiltroMedia(uint16_t vector[]);


void main(void)
{
    // Inicializa el dispositivo y sus drivers
    SYSTEM_Initialize();
    // Habilita las interrupciones globales
    INTERRUPT_GlobalInterruptEnable();
    IO_RB4_SetLow();


    while (1)
    {
```

```
v_min=0xFFFF;
v_max=0;
/*-----------------------------------------------------------------
 *   Se realiza una petición de envío de tensión a cada esclavo. El Id de
 * cada uno de ellos será el valor de i más el valor del caracter '1'.
 * Inicialmente se envía un mensaje con el identificador del micro al
 * que se le realiza la petición de envío y quedamos a la espera de recibir
 * respuesta. Si no se recibe la misma se pasa a realizar la siguiente
 * petición para no bloquear el micro en este punto. La respuesta serán
 * 3 mensajes. El primero el carácter 'v', dará inicio a la trama, el
 * segundo sera el dato, y el tercero será una 'f' para indicar que la
 * trama ha llegado completa. En caso de no ser así, se intentará otra
 * lectura notificando al esclavo que debe repetir el envío. Posteriormente
 * se comprueba si el dato recibido es un dato de tensión mínimo o máximo,
 * y en caso afirmativo se almacena en la variable correspondiente.
 *-----------------------------------------------------------------*/
for (int i=0;i<26;i++){
   k=i+'1';
   intentos=0;
   EUSART1_Write(k);
   while((EUSART1_DataReady<4)&&(intentos<100)){
      intentos++;
   }
   if(intentos<100){
      lectura=EUSART1_Read();
      if(lectura=='v'){
```

```
            tensiones[i]=EUSART1_Read()<<8;
            tensiones[i]=EUSART1_Read()||tensiones[i];
            lectura=EUSART1_Read();
            if(lectura!='f'){
               EUSART1_Write('$');
               tensiones[i]=EUSART1_Read();
               lectura=EUSART1_Read();
            }
         }
      }
      if(tensiones[i]<v_min)
         v_min=tensiones[i];
      if(tensiones[i]>v_max)
         v_max=tensiones[i];
   }
   /* ------------------------------------------------------------------
    *  Se realiza la adquisición de la temperatura. Se toman 4 muestras de
    * cada uno de las tres zonas que se miden, y se les realiza un filtrado
    * de media móvil. Posteriormente se comparan y se haya la máxima.
    * ------------------------------------------------------------------*/
   temp_max=0;
   for (char j=0;j<3;j++){
      for(int l=0;l<4;l++){
         aux[l]=LeeTemperatura(j);
      }
      temperaturas[j]=FiltroMedia(aux);
```

```
        grados[j]=1.024*temperaturas[j]/(4096*0.01);
         if(temperaturas[j]>temp_max){
            temp_max=temperaturas[j];
            temp_max2=grados[j];
        }
    }
    /* ------------------------------------------------------------------
     *  Se verifica si las temperatura máxima, la tensión mínima y la máxima
     * se encuentran dentro de rango. En caso de no cumplirse, se acciona
     * la salida digital que actúa sobre el relé.
     * ------------------------------------------------------------------*/
    if(v_max>60957||v_min<53700||temp_max>2400){
        IO_RB4_SetHigh();
    }
    /* ------------------------------------------------------------------
     *  Programación para implementación del módulo Bluetooth. Si recibe el
     * comando '$', devolverá las tensiones actuales, con una trama que será
     * 'V'+dato+'F'. En caso de recibir el comando 'H', enviará el histórico
     * almacenado en la EEPROM. La trama para el envío de datos de la EEPROM
     * será del tipo 'L'+v_min+'M'+v_max+'T'+t_max
     * ------------------------------------------------------------------*/

    if(EUSART2_DataReady>0){
        if(orden=='$'){
            for(int m=0;m<26;m++){
                EUSART2_Write('V');
```

```
            EUSART2_Write(tensiones[m]>>8);
            EUSART2_Write((uint8_t)tensiones[m]);
        }
        for(int n=0;n<3;n++){
            EUSART2_Write('T');
            EUSART2_Write(temperaturas[n]>>8);
            EUSART2_Write((uint8_t)temperaturas[n]);
        }
        EUSART2_Write('F');
    }
    if(orden=='H'){
        while(adress2>=adress){
            EUSART2_Write('L');
            EUSART2_Write(DATAEE_ReadByte(adress2)>>8);
            EUSART2_Write((uint8_t)DATAEE_ReadByte(adress2));
            adress2=adress2+0x0010;
            EUSART2_Write('M');
            EUSART2_Write(DATAEE_ReadByte(adress2)>>8);
            EUSART2_Write((uint8_t)DATAEE_ReadByte(adress2));
            adress2=adress2+0x0010;
            EUSART2_Write('T');
            EUSART2_Write(DATAEE_ReadByte(adress2)>>8);
            EUSART2_Write((uint8_t)DATAEE_ReadByte(adress2));
        }
        adress2=0xF000;
    }
```

```
        }
        /* ------------------------------------------------------------------
         *  El timer ha sido calculado para activar un flag cada 2 minutos que
         * notifica al sistema que debe guardar en la EEPROM los valores que
         * tenga en ese momento. Por limitaciones de la memoria EEPROM se ha
         * optado por almacenar sólo valores extremos.
         * ------------------------------------------------------------------*/

        if(guardar){
                word=v_min;
                DATAEE_WriteByte(adress, word);
                word=v_max;
                adress=adress+0x0010;
                DATAEE_WriteByte(adress,word);
                word=temp_max;
                adress=adress+0x0010;
                DATAEE_WriteByte(adress,word);
                adress=adress+0x0010;
                guardar=false;
                if(adress>0x0F0F0)
                   adress=0xF000;
        }
    }
}
/*------------------------------------------------------------------------
 *  Función que realiza la lectura del ADC. Cuenta con una entrada y una sa-
```

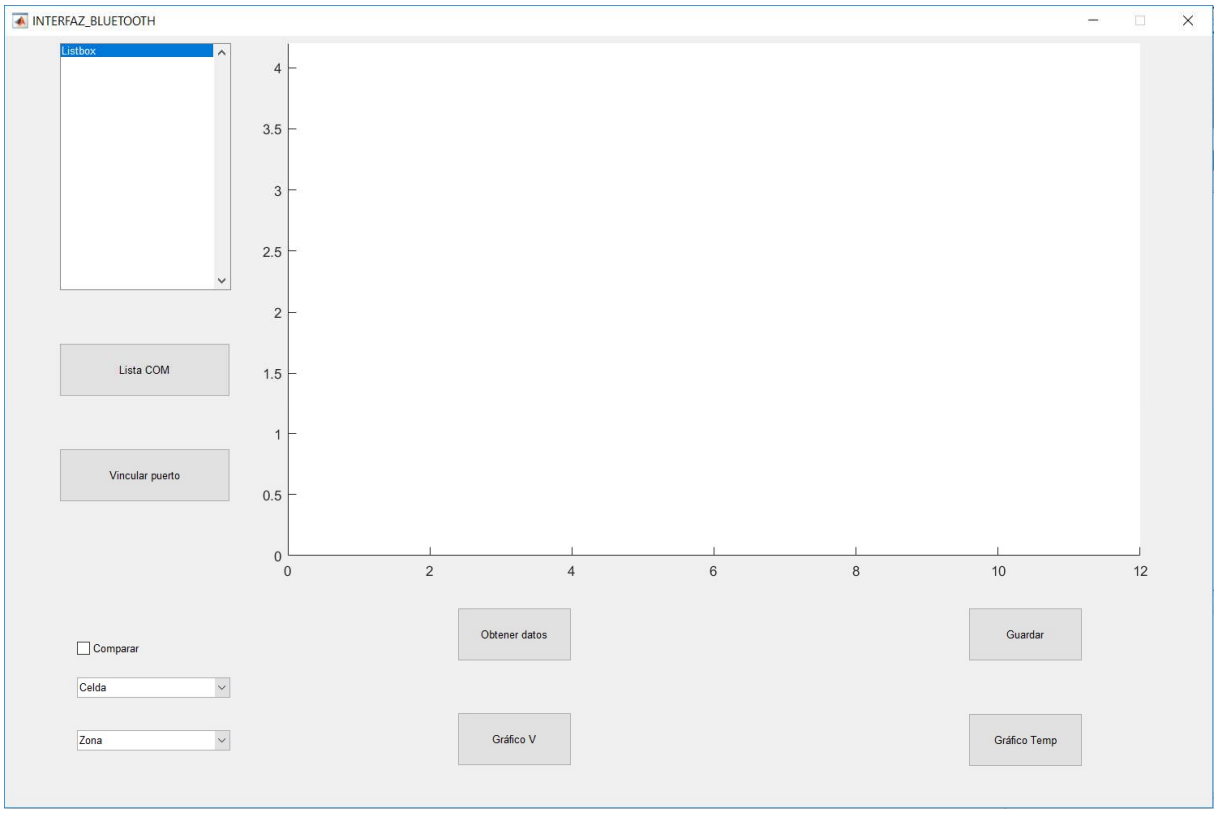
```
 * lida. Inicialmente selecciona el canal a leer, realiza la conversión, es-
 * pera a que finalice y devuelve el valor obtenido.
 * Input: adc_channel_t channel -> Canal de ADC
 *        canal =  1 -> channel_ANA1
 *        canal =  2 -> channel_ANA2
 *        canal =  3 -> channel_ANA3
 * Output: uint16_t q -> valor de lectura del canal
 *---------------------------------------------------------------------------*/


uint16_t LeeTemperatura(uint8_t canal){
  uint16_t q;
  uint8_t channel;
  ADCC_Initialize();
  __delay_us(250);
  switch(canal){
     case 0:
        channel=channel_ANA1;
        break;
     case 1:
        channel=channel_ANA2;
        break;
     case 2:
        channel=channel_ANA3;
        break;
  }
  ADCC_StartConversion( channel );
```

```
  while(ADCC_IsConversionDone()==false);
  q=ADCC_GetConversionResult();
  return q;
}
/*------------------------------------------------------------------------
 *  Función que realiza un filtro de media móvil de 4 valores. Concede más
 * peso al último valor que reciba (último valor del array de entrada), y
 * se incrementa el peso en los siguientes, asignando 5% al primer valor, 15%
 * al segundo, 35% al tercero y 45% al más reciente.
 * Input: uint16_t vector[] -> array de 4 valores
 * Output: uint16_t filtrado -> Valor filtrado
 *------------------------------------------------------------------------*/
uint16_t FiltroMedia(uint16_t vector[ ]){
  uint16_t filtrado;
  filtrado=(0.45*vector[3]+0.35*vector[2]+0.15*vector[1]+0.05*vector[0]);
  return filtrado;
}
/**
 End of File
*/
```

13. Interfaz para Bluetooth

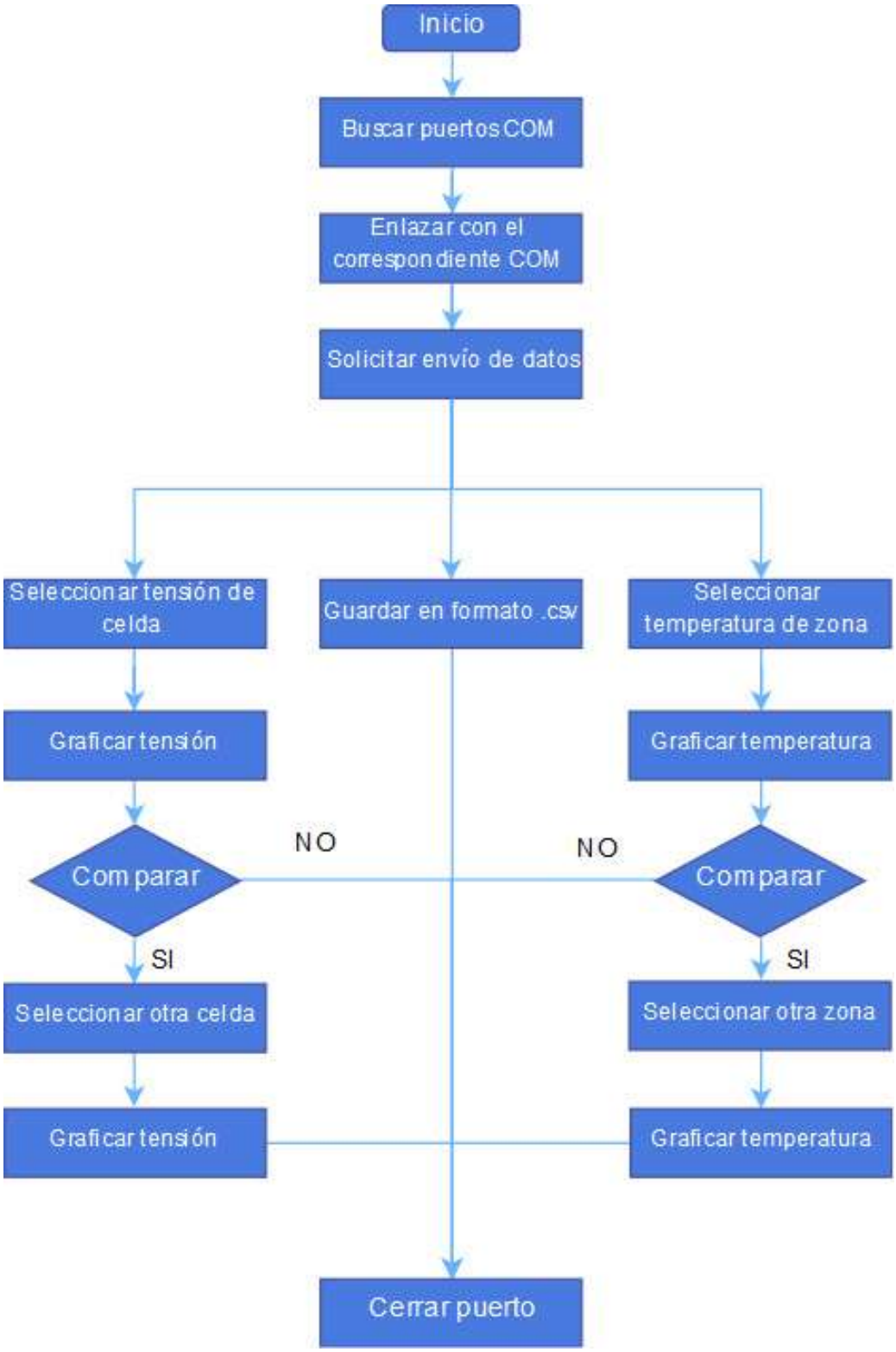
Como se ha expuesto en anteriores apartados, la implantación bluetooth se descartó para realizarse durante la competición. Sin embargo, la interfaz está diseñada y es posible comunicarse con ella mediante puerto serie a modo de prueba, y queda como una posible modificación de mejora el sustituir el módulo Bluetooth por un modelo que cumpla con la funcionalidad deseada.

La interfaz que se expone ha sido diseñada mediante Matlab, con la herramienta GUIDE. Dicha interfaz rastrea en busca de puertos series al pulsar “Lista COM”, el usuario debe seleccionar puerto virtual que enlace con el módulo bluetooth de su dispositivo, y posteriormente pulsar en el botón Vincular puerto.



Una vez completado el enlace, debemos pulsar el botón “Obtener datos”, que nos proporcionará una matriz con toda la información de todas las baterías y las tres zonas de temperatura. Tras esto, podemos seleccionar la casilla de comparar si queremos plotear dos señales o más, y posteriormente, se selecciona la celda cuya tensión vamos a graficar, o la zona de la caja cuya temperatura queremos que nos muestre, y pulsamos “Gráfico V” o “Gráfico Temp” según corresponda. Finalmente, mediante el botón “Guardar”, podemos guardar en formato .csv para un posterior análisis la información de las 26 celdas obtenida anteriormente de la EEPROM del microcontrolador máster y la temperatura de las 3 regiones del cajón de baterías.

13.1. Flujograma



13.2. Programación

```
1. function varargout = INTERFAZ_BLUETOOTH(varargin)
2. % INTERFAZ_BLUETOOTH MATLAB code for
INTERFAZ_BLUETOOTH.fig
3. %      INTERFAZ_BLUETOOTH, by itself, creates a new
INTERFAZ_BLUETOOTH or raises the existing
4. %      singleton*.
5. %
6. %      H = INTERFAZ_BLUETOOTH returns the handle to a
new INTERFAZ_BLUETOOTH or the handle to
7. %      the existing singleton*.
8. %
9. %
INTERFAZ_BLUETOOTH('CALLBACK',hObject,eventData,handles,
...) calls the local
10.     %      function named CALLBACK in
INTERFAZ_BLUETOOTH.M with the given input arguments.
11.     %
12.     %      INTERFAZ_BLUETOOTH('Property','Value',...)
creates a new INTERFAZ_BLUETOOTH or raises the
13.     %      existing singleton*.  Starting from the
left, property value pairs are
14.     %      applied to the GUI before
INTERFAZ_BLUETOOTH_OpeningFcn gets called.  An
15.     %      unrecognized property name or invalid
value makes property application
16.     %      stop.  All inputs are passed to
INTERFAZ_BLUETOOTH_OpeningFcn via varargin.
17.     %
18.     %      *See GUI Options on GUIDE's Tools menu.
Choose "GUI allows only one
19.     %      instance to run (singleton)".
20.     %
21.     % See also: GUIDE, GUIDATA, GUIHANDLES
22.
23.     % Edit the above text to modify the response to
help INTERFAZ_BLUETOOTH
24.
25.     % Last Modified by GUIDE v2.5 26-Sep-2018
22:59:33
26.
27.     % Begin initialization code - DO NOT EDIT
28.     gui_Singleton = 1;
29.     gui_State = struct('gui_Name',       mfilename,
...
30.                        'gui_Singleton',
gui_Singleton, ...
```

```
31.                          'gui_OpeningFcn',
@INTERFAZ_BLUETOOTH_OpeningFcn, ...
32.                          'gui_OutputFcn',
@INTERFAZ_BLUETOOTH_OutputFcn, ...
33.                          'gui_LayoutFcn',  [] , ...
34.                          'gui_Callback',   []);
35.     if nargin && ischar(varargin{1})
36.         gui_State.gui_Callback =
str2func(varargin{1});
37.     end
38.
39.     if nargout
40.         [varargout{1:nargout}] =
gui_mainfcn(gui_State, varargin{:});
41.     else
42.         gui_mainfcn(gui_State, varargin{:});
43.     end
44.     % End initialization code - DO NOT EDIT
45.
46.
47.     % --- Executes just before INTERFAZ_BLUETOOTH is
made visible.
48.     function INTERFAZ_BLUETOOTH_OpeningFcn(hObject,
eventdata, handles, varargin)
49.     % This function has no output args, see
OutputFcn.
50.     % hObject    handle to figure
51.     % eventdata  reserved - to be defined in a future
version of MATLAB
52.     % handles    structure with handles and user data
(see GUIDATA)
53.     % varargin   command line arguments to
INTERFAZ_BLUETOOTH (see VARARGIN)
54.
55.     % Choose default command line output for
INTERFAZ_BLUETOOTH
56.     handles.output = hObject;
57.
58.     % Update handles structure
59.     guidata(hObject, handles);
60.     axes(handles.axes1);
61.     ylim([0 4.2]);
62.     xlim([0 12]);
63.     set(handles.B_Cerrar,'Visible','off');
64.     set(handles.B_Cerrar,'Enable','off');
65.     % UIWAIT makes INTERFAZ_BLUETOOTH wait for user
response (see UIRESUME)
```

```
66.     % uiwait(handles.figure1);
67.
68.
69.     % --- Outputs from this function are returned to
the command line.
70.     function varargout =
INTERFAZ_BLUETOOTH_OutputFcn(hObject, eventdata,
handles)
71.     % varargout  cell array for returning output args
(see VARARGOUT);
72.     % hObject    handle to figure
73.     % eventdata  reserved - to be defined in a future
version of MATLAB
74.     % handles    structure with handles and user data
(see GUIDATA)
75.
76.     % Get default command line output from handles
structure
77.     varargout{1} = handles.output;
78.
79.
80.     % --- Executes on button press in B_DATOS.
81.     function B_DATOS_Callback(hObject, eventdata,
handles)
82.     % hObject    handle to B_DATOS (see GCBO)
83.     % eventdata  reserved - to be defined in a future
version of MATLAB
84.     % handles    structure with handles and user data
(see GUIDATA)
85.
86.
87.     datos=struct;
88.     k=1;
89.     m=1;
90.     s=get(handles.B_StartCOM,'UserData');
91.     fopen(s);
92.     pause(1);
93.     fwrite(s,'$');
94.     lectura=0;
95.     l=1;
96.     n=1;
97.     while(lectura~='F')
98.         if(s.BytesAvailable>=3)
99.
100.            lectura=fscanf(s,'%s');
101.            if(lectura=='V')
102.                datos.v(k,l)=fscanf(s,'%u');
```

```
103.                 l=l+1;
104.                 if(l>=27)
105.                     l=1;
106.                     k=k+1;
107.                 end
108.             end
109.             if(lectura=='T')
110.                 datos.t(m,n)=fscanf(s,'%u');
111.                 n=n+1;
112.                 if(n>=4)
113.                     n=1;
114.                     m=m+1;
115.                 end
116.             end
117.         end
118.     end
119.     fclose(s);
120.     set(handles.B_DATOS,'UserData',datos);
121.
122.
123.     % --- Executes on button press in B_COM.
124.     function B_COM_Callback(hObject, eventdata,
handles)
125.     % hObject    handle to B_COM (see GCBO)
126.     % eventdata  reserved - to be defined in a future
version of MATLAB
127.     % handles    structure with handles and user data
(see GUIDATA)
128.     lista=get_port_list;
129.     COM=lista(:,2);
130.     set(handles.listbox1,'String',COM);
131.     set(handles.B_COM,'UserData',COM);
132.     % --- Executes on selection change in listbox1.
133.     function listbox1_Callback(hObject, eventdata,
handles)
134.     % hObject    handle to listbox1 (see GCBO)
135.     % eventdata  reserved - to be defined in a future
version of MATLAB
136.     % handles    structure with handles and user data
(see GUIDATA)
137.
138.     % Hints: contents =
cellstr(get(hObject,'String')) returns listbox1 contents
as cell array
139.     %        contents{get(hObject,'Value')} returns
selected item from listbox1
140.     a=get(handles.listbox1,'Value');
```

```
141.    set(handles.listbox1,'UserData',a);
142.
143.    % --- Executes during object creation, after
setting all properties.
144.    function listbox1_CreateFcn(hObject, eventdata,
handles)
145.    % hObject    handle to listbox1 (see GCBO)
146.    % eventdata  reserved - to be defined in a future
version of MATLAB
147.    % handles    empty - handles not created until
after all CreateFcns called
148.
149.    % Hint: listbox controls usually have a white
background on Windows.
150.    %       See ISPC and COMPUTER.
151.    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
152.        set(hObject,'BackgroundColor','white');
153.    end
154.
155.
156.    % --- Executes on button press in B_StartCOM.
157.    function B_StartCOM_Callback(hObject, eventdata,
handles)
158.    % hObject    handle to B_StartCOM (see GCBO)
159.    % eventdata  reserved - to be defined in a future
version of MATLAB
160.    % handles    structure with handles and user data
(see GUIDATA)
161.    b=get(handles.listbox1,'UserData');
162.    COM=get(handles.B_COM,'UserData');
163.    port=COM(b);
164.    port2=char(port);
165.    s=serial(port2);
166.    s.BaudRate=9600;
167.    set(handles.B_StartCOM,'UserData',s);
168.    set(handles.B_StartCOM,'Visible','off');
169.    set(handles.B_StartCOM,'Enable','off');
170.    set(handles.B_Cerrar,'Visible','on');
171.    set(handles.B_Cerrar,'Enable','on');
172.
173.
174.    % --- Executes on button press in B_Guardar.
175.    function B_Guardar_Callback(hObject, eventdata,
handles)
176.    % hObject    handle to B_Guardar (see GCBO)
```

```
177.    % eventdata  reserved - to be defined in a future
version of MATLAB
178.    % handles    structure with handles and user data
(see GUIDATA)
179.    fileID=fopen('archivo.csv','a');
180.    datos=get(handles.B_DATOS,'UserData');
181.    [i,j]=size(datos.v);
182.    [k,l]=size(datos.t);
183.    for m=1:4
184.        fprintf(fileID,'TENSIONES %1.0f\r\n',m);
185.        for n=1:26
186.            fprintf(fileID,'Celda %1.0f;',n);
187.
fprintf(fileID,'%3.4f;V\r\n',datos.v(m,n));
188.        end
189.        fprintf(fileID,'TEMPERATURAS %1.0f\r\n',m);
190.        for p=1:3
191.            fprintf(fileID,'Zona %1.0f;',p);
192.
fprintf(fileID,'%3.4f;°C\r\n',datos.t(m,p));
193.        end
194.    end
195.
196.
197.    % --- Executes on button press in B_PlotT.
198.    function B_PlotT_Callback(hObject, eventdata,
handles)
199.    % hObject    handle to B_PlotT (see GCBO)
200.    % eventdata  reserved - to be defined in a future
version of MATLAB
201.    % handles    structure with handles and user data
(see GUIDATA)
202.    axes(handles.axes1);
203.    estado=get(handles.checkbox1,'Value');
204.    datos=get(handles.B_DATOS,'UserData');
205.    ylim([0 60]);
206.    xlim([0 12]);
207.    select=get(handles.popupmenu2,'Value');
208.    lista=get(handles.popupmenu2,'String');
209.    select2=str2num(char(lista(select)));
210.    tiempo=[3; 6; 9; 12];
211.    temp=datos.t(:,select2);
212.    if(estado==1)
213.        hold(handles.axes1,'on')
214.    else
215.        hold(handles.axes1,'off')
216.    end
```

```
217.    plot(tiempo,datos.t(:,select2));
218.
219.    % --- Executes on button press in B_PlotV.
220.    function B_PlotV_Callback(hObject, eventdata,
handles)
221.    % hObject    handle to B_PlotV (see GCBO)
222.    % eventdata  reserved - to be defined in a future
version of MATLAB
223.    % handles    structure with handles and user data
(see GUIDATA)
224.    axes(handles.axes1);
225.    estado=get(handles.checkbox1,'Value');
226.    datos=get(handles.B_DATOS,'UserData');
227.    select=get(handles.popupmenu1,'Value');
228.    lista=get(handles.popupmenu1,'String');
229.    select2=str2num(char(lista(select)));
230.    tiempo=[3; 6; 9; 12];
231.    tension=datos.v(:,select2);
232.    if(estado==1)
233.        hold(handles.axes1,'on')
234.    else
235.        hold(handles.axes1,'off')
236.    end
237.    plot(tiempo,datos.v(:,select2));
238.
239.    % --- Executes on button press in B_Cerrar.
240.    function B_Cerrar_Callback(hObject, eventdata,
handles)
241.    % hObject    handle to B_Cerrar (see GCBO)
242.    % eventdata  reserved - to be defined in a future
version of MATLAB
243.    % handles    structure with handles and user data
(see GUIDATA)
244.    fclose('all');
245.
246.
247.    % --- Executes on selection change in popupmenu1.
248.    function popupmenu1_Callback(hObject, eventdata,
handles)
249.    % hObject    handle to popupmenu1 (see GCBO)
250.    % eventdata  reserved - to be defined in a future
version of MATLAB
251.    % handles    structure with handles and user data
(see GUIDATA)
252.
```

```
253.    % Hints: contents =
cellstr(get(hObject,'String')) returns popupmenu1
contents as cell array
254.    %        contents{get(hObject,'Value')} returns
selected item from popupmenu1
255.
256.
257.    % --- Executes during object creation, after
setting all properties.
258.    function popupmenu1_CreateFcn(hObject, eventdata,
handles)
259.    % hObject    handle to popupmenu1 (see GCBO)
260.    % eventdata  reserved - to be defined in a future
version of MATLAB
261.    % handles    empty - handles not created until
after all CreateFcns called
262.
263.    % Hint: popupmenu controls usually have a white
background on Windows.
264.    %       See ISPC and COMPUTER.
265.    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
266.        set(hObject,'BackgroundColor','white');
267.    end
268.
269.
270.    % --- Executes on selection change in popupmenu2.
271.    function popupmenu2_Callback(hObject, eventdata,
handles)
272.    % hObject    handle to popupmenu2 (see GCBO)
273.    % eventdata  reserved - to be defined in a future
version of MATLAB
274.    % handles    structure with handles and user data
(see GUIDATA)
275.
276.    % Hints: contents =
cellstr(get(hObject,'String')) returns popupmenu2
contents as cell array
277.    %        contents{get(hObject,'Value')} returns
selected item from popupmenu2
278.
279.
280.    % --- Executes during object creation, after
setting all properties.
281.    function popupmenu2_CreateFcn(hObject, eventdata,
handles)
```

```
282.    % hObject    handle to popupmenu2 (see GCBO)
283.    % eventdata  reserved - to be defined in a future
version of MATLAB
284.    % handles    empty - handles not created until
after all CreateFcns called
285.
286.    % Hint: popupmenu controls usually have a white
background on Windows.
287.    %       See ISPC and COMPUTER.
288.    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
289.        set(hObject,'BackgroundColor','white');
290.    end
291.
292.
293.    % --- Executes on button press in checkbox1.
294.    function checkbox1_Callback(hObject, eventdata,
handles)
295.    % hObject    handle to checkbox1 (see GCBO)
296.    % eventdata  reserved - to be defined in a future
version of MATLAB
297.    % handles    structure with handles and user data
(see GUIDATA)
298.
299.    % Hint: get(hObject,'Value') returns toggle state
of checkbox1
```

13.3. Funciones

Para el rastreo de puertos COM se utiliza la función get_port_list.m, la cual adjunto a continuación:

```
if nargin < 1, verbose = 1; end
ttyList={};
portOrder =[];

    % Get COM port info with system (Windows) mode
command
    if verbose >= 3
        [stat, result] = system('mode', '-echo');
    else
        [stat, result] = system('mode');
    end
    if stat ~= 0
```

```
        fprintf(1,'get_port_list: Unable to get device
list (error %d)\n',stat);
        return
    end

    % identify the COM ports in the output string and
format in cell array
    portList = regexp(result,'COM\d+:','match');
    k=0;
    for p = portList
        k=k+1;
        portNum = regexp(p{1},'\d+','match');
        portName = regexp(p{1},'COM\d+','match');
        ttyList = [ttyList; {portNum{1}, portName{1}}];
        if verbose >= 2, fprintf(1,'  Serial Device at
%s\n',ttyList{k,2}); end
        portOrder = [portOrder str2num(portNum{1})];
    end
    % return the list sorted with largest number first
    [p2,i]=sort(portOrder,2,'descend');
    ttyList = ttyList(i,:);
```

Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén
Grado en Ingeniería Electrónica

BIBLIOGRAFÍA

Alumno: Juan Caño Navas

Tutor: Prof. D. Miguel de la Fuente Ruz

Dpto: Ingeniería de Sistemas y Automática

Diciembre, 2018

- “Battery Management Systems (BMS) for Increasing Battery Life Time” by A. Jossen, V. Spath, H. Doring, J. Garche, Center for Solar Energy and Hydrogen Research (ZSW Ulm) IEEE
- Alberto Martín Pernía. (2007). MANUAL DE MICROCONTROLADORES PIC. 10/05/2018, de Universidad de Oviedo Sitio web: https://www.unioviedo.es/ate/alberto/manualPic.pdf
- Barrie Lawson. (2005). Lithium Battery Failures. 03/08/2018, de Barrie Lawson Sitio web: https://www.mpoweruk.com/lithium_failures.htm
- Introducción a los microcontroladores, José Adolfo González V., McGraw Hill Isuskiza CREACIONES COPYRIGHT 2011
- Juan J. Guevara. (2014). Guías para principiantes. 15/06/2018, de Juan J. Guevara Sitio web: https://unasguiasmas.com/tag/xc8/
- Microchip. (2012). MPLAB XC8 C Compiler User's Guide. 09/06/2018, de Microchip Sitio web: http://ww1.microchip.com/downloads/en/devicedoc/50002053g.pdf
- Microcontroladores PIC, Tavernier, Editorial Paraninfo
- ‘‘Microcontroladores PIC teoría y práctica” por Mikel Etxebarria