MÁSTER UNIVERSITARIO EN

INGENIERÍA EN TECNOLOGÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

BMS (BATTERY MANAGEMENT SYSTEM) PARA UNA MOTOCICLETA ELÉCTRICA

Estudiante	*Lizarralde Bilbao Markel*
Director/Directora	*Zuloaga Izaguirre Aitzol*
Departamento	**Tecnología electrónica**
Curso académico	**2019-2020**

Bilbao, 14 de septiembre de 2020

RESUMEN TRILINGÜE

RESUMEN

El propósito de este documento es presentar un proyecto que es de utilidad para el equipo MotoStudent. Este proyecto consiste en el desarrollo, fabricación y puesta a punto de un BMS (battery management system), software necesario para controlarlo y software adicional para diagnosis desde el ordenador. Se muestra el diseño de un BMS de manera económica para posteriormente utilizarlo en la competición de MotoStudent. Se trabajan apartados tales como los objetivos, el alcance, estudio de alternativas y soluciones, desarrollo teórico y práctico completo para la consecución del trabajo y las conclusiones del mismo.

ABSTRACT

The purpose of this document is to introduce a project that could be useful for the MotoStudent team. The project consists in the development of a BMS (battery management system) and its corresponding software to be able to control it. The aim of this document is to present the possibility of designing a BMS in an economical way to be able to use it in the MotoStudent competition. The proposed study is described in the following text. Related to the document, we will find out that it is divided in diferent parts, including introduction, context, goals and scope, working plan, resources, and conclusions.

LABURPENA

Dokumentu honen helburua MotoStudent taldearentzat erabilgarria den proiektu bat garatzea da. Lan hau BMS (battery management system) baten garapen teknikoa, fabrikazioa, programazioa eta probak egitean datza. MotoStudent lehiaketan erabili ahal izango den BMS ekonomiko bat diseinatu da. Garatuko diren atalen artean helburua, irispena, aukeren eta konponbideen azterketa, lana aurrera eramateko garapen tekniko osoa eta ondorioak aurkitzen dira.

PALABRAS CLAVE

BMS, battery management system, MotoStudent, placa electrónica, software, programación, diseño esquemático, arduino.

1. Contenido

TABLA DE ILUSTRACIONES

1. MEMORIA

INTRODUCCIÓN

Este documento contiene un proyecto de diseño e implantación del BMS (battery management system) para una moto eléctrica de competición, así como el software necesario para dicha placa. Este software servirá para el posterior análisis del funcionamiento del BMS y las baterías de la motocicleta por parte del equipo de MotoStudent Bizkaia.

En una primera parte se presenta el contexto del proyecto, objetivos del mismo y los beneficios técnicos y económicos que se esperan conseguir. Para ello, se estudiarán distintas alternativas y se valorarán ambas mediante diferentes criterios, eligiendo aquella opción que más se ajuste a las necesidades.

Una vez seleccionada la alternativa, se realiza una descripción en la que se muestran las diferentes partes de las que constará el BMS y se da una breve explicación de cada una de ellas. También se detallarán todos los detalles del Software diseñado para su control.

A continuación, se planifican todos los trabajos a realizar, viendo las diferentes relaciones entre las tareas a llevar a cabo. Principalmente hay dos etapas, una en la que se realiza el diseño y fabricación de las placas electrónicas y otra en la que se programa el Software.

Este proyecto cuenta también con un análisis detallado de todos los costes. En este último se enumeran las ventajas obtenidas, dando una explicación de las consecuencias de las mismas.

CONTEXTO

Como alumno de la escuela participo voluntariamente en la sexta edición de la competición de MotoStudent (2018-2020) [1]. MotoStudent es un desafío que nace de la Ciudad del motor de Aragón, siendo una competición entre universidades de ingeniería de todo el mundo. El objetivo consiste en el diseño y construcción de una motocicleta de competición eléctrica. Las distintas universidades se enfrentarán en la competición a unas pruebas técnicas durante unas jornadas que se llevan a cabo en las instalaciones de Motorland Aragón durante un fin de semana en el mes de octubre 2020, teniendo como prueba final la carrera. Las pruebas planteadas tendrán como fin evaluar el comportamiento dinámico y las prestaciones de la moto fabricada, para comprobar la seguridad y funcionalidad, resultado dependiente de la destreza de los miembros del equipo.

Uno de los objetivos principales por parte de la escuela de ingeniería de Bilbao es construir la mayor cantidad de elementos de la motocicleta que sea posible. Uno de los componentes básicos y más importantes en un vehículo eléctrico es el BMS:

Las siglas de BMS proviene de Battery management system en inglés, o sistema de gestión de baterías en español. Las baterías de Ion de Litio son peligrosas ya que un uso indebido de las mismas puede llevarlas un sobrecalentamiento o incuso a la explosión. Con el fin de evitar cualquier problema se han desarrollado distintas alternativas. Entre ellas, y la más utilizada sin duda, es el BMS. Es la placa de control encargada de que las baterías del vehículo eléctrico permanezcan en su rango seguro de operación (*safe operation area*). Para ello debe monitorizar distintos parámetros eléctricos y térmicos de las baterías como voltajes, intensidades de carga y descarga, temperaturas… y asegurar la desconexión de las baterías del sistema de tracción en caso de fallo de alguno de los mismos.

Por lo tanto, debido a la importancia del BMS en el vehículo eléctrico, así como la necesidad utilizar uno para la próxima motocicleta de la universidad se ha decidido llevar a cabo el proyecto de diseño, ensamblaje y pruebas reales en circuito.

ESTADO DEL ARTE

BMS comerciales

Las funciones mínimas que cumple el BMS más sencillo son las siguientes:

- Protección ante sobretensiones: Si el voltaje de alguna de las celdas llega a los 4.2V (voltaje máximo de las celdas de Li-Ion), el BMS debe impedir que se siga cargando esa celda. Para impedirlo abre el contactor o mosfet que regule la entrada de corriente y deja de cargar toda la batería. En caso de que se use un BMS sin la capacidad de balancear las celdas no se habrá cargado la batería al completo ya que el resto de celdas no habrán llegado al voltaje de 4.2V. Además, los BMS más sencillos no permiten regular el voltaje al que se corta la carga, con lo que solo se podría utilizar ese dispositivo para una tecnología específica de celdas.
- Protección ante sobrecargas de intensidad: Si la intensidad de descarga supera cierto valor abre el circuito desconectando a la batería de la carga. Puede que no exista esta protección cuando se cargue la batería ya que suele ser el propio cargador quien lleva esa limitación de corriente. Al igual que el voltaje, esta intensidad de corte no suele ser regulable en los BMS comerciales. En los BMS más compactos suele usarse una resistencia shunt y en los de mayor potencia un sensor

tipo Hall para medir la intensidad (véase el apartado "Grupo de alternativas 3: Topología de medición de intensidades"). Los segundos son más complicados de conseguir y mucho más caros.

- Protección ante baja tensión: Al igual que la protección ante sobretensiones se corta el suministro cuando una de las celdas ha llegado a su voltaje mínimo. Cuando esto ocurre el resto de celdas puede que no hayan llegado hasta dicho voltaje, con lo que no se habrá aprovechado toda la energía disponible en la batería. Tampoco se suele poder cambiar el valor del voltaje de corte en el rango inferior.

En el caso de que el BMS posea más funciones suele tener las siguientes:

- Balanceo de las celdas: Generalmente mediante el uso de resistencia conmutada. Gracias a esto se evita el problema que se explica en mayor detalle en el apartado "Grupo de alternativas 2: Tipo de balanceo".
- Valores de protecciones configurables: Posibilidad de modificar todos los parámetros de protección y corte.
- Comunicaciones con otros BMS del mismo tipo para monitorizar una batería con más celdas de las que puede monitorizar un único BMS (véase el apartado "Grupo de alternativas 1: Tipología del BMS").
- Monitorización del estado de los parámetros en un ordenador, móvil u otro dispositivo. Para ello se suele utilizar protocolos de comunicación como Bluetooth, CAN, Ethernet, etc.

OBJETIVOS Y ALCANCE

A continuación se muestra tanto el objetivo del proyecto como como el alcance del mismo. El objetivo principal es el **diseño, ensamblaje y puesta a punto de un BMS para la motocicleta eléctrica de Bizkaia ESI Bilbao 2020**. El BMS a diseñar contendrá todas las funciones que se han explicado en el apartado anterior.

Adicionalmente se pueden destacar los siguientes objetivos secundarios:

- Posibilitar la adquisición de datos de los parámetros de funcionamiento del BMS y la batería de la motocicleta.
- Comunicar el BMS con distintos elementos de la motocicleta para extraer datos útiles de la misma.
- Diseñar un programa PC o aplicación móvil que permita extraer los datos adquiridos para posibilitar un posterior análisis.

- Extraer los datos a tiempo real mediante el uso de la tecnología de telemetría desarrollada por el equipo de la anterior edición.

El alcance de este proyecto parte desde el estudio de los distintos tipos de BMS hasta el uso en circuito del mismo. Se van a diseñar las placas electrónicas y el software necesario que permita maximizar las funciones del BMS. Una vez diseñado y ensamblado se realizarán distintas pruebas en laboratorio y finalmente se efectuarán pruebas en circuito.

BENEFICIOS

A continuación se detallan los diferentes beneficios del proyecto, los cuales se pueden clasificar en tres categorías: beneficios técnicos, económicos y medioambientales:

Beneficios Técnicos

Entre los beneficios técnicos que produce el proyecto destacan los siguientes:

- Diseño a medida de uno de los elementos cruciales de la motocicleta. Esto permitirá una mayor optimización de los recursos y espacio, reduciendo así el coste y peso del conjunto.
- Adquisición de datos de los parámetros de funcionamiento del BMS y la batería de la motocicleta. Con BMS comerciales no es tan sencillo o barato de lograrlo y esto permitirá realizar un análisis más profundo con el fin de poder mejorar la motocicleta para futuras ediciones.
- Facilidad de comunicar el BMS con el resto de componentes de la moto como el dashboard, inversor, sensores… ya que, al estar la mayoría diseñado por miembros del equipo, es más sencillo utilizar un protocolo de comunicación común para todos los elementos.
- En caso de conseguir implantar la telemetría todos los datos serán a tiempo real, lo que permite un análisis y previsión de fallos de una manera más rápida. Por ejemplo, en caso de detectar algo, bastará con comunicarle al piloto en el dashboard que debe volver a boxes para corregir algo en vez de esperar a que se termine el tiempo de la tanda y vuelva tarde.

Aunque se trate de diseñar el BMS a medida para la competición, se va a tener en cuenta la posibilidad de diseñar la placa electrónica de tal forma que pueda ser modular, esto es, que si en un futuro hiciera falta ampliar el rango de voltaje o intensidad fuera posible mediante el uso de más de un BMS del mismo tipo. De esta manera se obtienen dos grandes beneficios:

- No tener que diseñar un nuevo BMS ante un cambio de requisitos en la competición.

- Adaptabilidad del BMS para distintas aplicaciones fuera de la competición: Vehículos eléctricos como coches, motocicletas, bicicletas e incluso patinetes.

Beneficios Económicos

Atendiendo a los beneficios económicos, destaca el ahorro en el coste de diseño y montaje del BMS dentro del equipo frente a comprar uno comercial. Dado que los alumnos que participan en la universidad lo hacen de forma voluntaria, lo que quiere decir que la mano de obra y de ingeniería es gratuita y las piezas y componentes fabricadas para la moto tienden a ser más baratas que comprarlas directamente.

Beneficios Sociales

Como beneficio social, demostrar la posibilidad de que una motocicleta eléctrica es tan viable como una de gasolina, actuando como ejemplo, concienciando e impulsando que la movilidad eléctrica es posible mejorando así su entorno.

Beneficios Medioambientales

Proyectos como estos ayudan a que se descubran nuevas tecnologías en el sector eléctrico/renovable, utilizándose en empresas de gran envergadura para que puedan utilizarlo en sus propios productos y concienciar así a la sociedad.

DESCRIPCIÓN DE REQUERIMIENTOS

Según los distintos elementos de la motocicleta como batería, motor, inversor… se establecen los siguientes requerimientos para el BMS:

Rango de voltaje	67.5- 113.4V
nº de celdas	hasta 27 celdas
Tensión de cada celda	2.5-4.2V
Corriente continua de descarga	250A
Corriente pico máximo de descarga	500A
Corriente máxima de carga	70A
Corriente pico máximo de carga	150A
Sensores de temperatura	4
Temperatura de la placa	20-60 ºC
Temperatura de las baterías	hasta 75 ºC
Comunicaciones	CAN, Bluetooth

Tabla 1. Requerimientos del BMS

El BMS debe controlar diversos parámetros eléctricos y térmicos para asegurar que las celdas trabajan en el área de operación segura (SOA), como ya se ha comentado anteriormente. Concretamente deberán especificarse y controlar los siguientes parámetros:

- Tensión nominal.
- Tensión máxima de carga.
- Tensión mínima en descarga.
- Corriente máxima de carga.
- Corriente mínima de carga.
- Corriente de pico máximo de carga.
- Corriente máxima de descarga.
- Corriente de pico máximo de descarga.
- Tiempo máximo del pico de carga y descarga.
- Estado de balanceo durante la carga.
- Temperatura máxima de las celdas.
- Temperatura máxima ambiental.
- Temperatura máxima en el circuito de potencia.

Los parámetros anteriores son medidos en algunos BMS comerciales, pero se van a incluir las siguientes funcionalidades adicionales como requerimiento:

- Control de contactores de la motocicleta.
- Supervisión del estado de los contactores.
- Supervisión del IMD (véase apartado “Diseño esquemático: Supervisión del IMD”).
- Comunicación Bluetooth y CAN.
- Almacenamiento de los datos en una micro SD.

Las placas electrónicas del BMS se van a soldar en la propia universidad con el procedimiento mostrado en el apartado “Ensamblaje”. Todos los componentes electrónicos que se vayan a soldar tendrán una separación mínima de los pines de los mismos. Por lo tanto, para los componentes pasivos como resistencias, condensadores, inductores… se va a escoger como mínimo el encapsulado estándar “1206” y el resto de componentes de varias patillas se va a exigir que tengan una separación de al menos 1mm entre las mismas. Es un factor importante a la hora de tener en cuenta el encapsulado o componente a escoger. Se tratará de minimizar el tamaño de estos siempre y cuando se respeten los límites inferiores. Si se escogieran encapsulados más pequeños a los comentados generarían problemas a la hora de soldarlos en las placas.

ANÁLISIS Y SELECCIÓN DE ALTERNATIVAS

Para cumplir los requerimientos especificados se debe tomar correctas decisiones a la hora de diseñar o escoger los componentes electrónicos. Estas decisiones se detallan a continuación en diferentes grupos funcionales.

Grupo de alternativas 1: Topología del BMS

Los BMS se pueden clasificar en función de cómo están instalados, esto es, el número de placas electrónicas que lo componen y el porqué de las mismas. Se pueden dividir en 4 tipos principalmente:

- Centralizado: Se trata de un único BMS encargado de realizar todas las tareas de medición y comunicación con otros elementos. Es el tipo más compacto y barato al tener todos los componentes sobre la misma placa. Resulta útil cuando el número de celdas a controlar es pequeño, ya que de lo contrario la placa sería demasiado grande y convendría utilizar otra tipología. Además, en caso de avería, los costes de reparación serían mayores ya que habría que sustituir el BMS completo y no solo una parte como en las siguientes tipologías.
- Modular: El BMS modular consiste en varios BMSs centralizados comunicados entre sí. Cada uno de ellos se encarga de realizar las mediciones del grupo de celdas que tiene asignadas y cada una de ellas tiene la posibilidad de realizar comunicaciones. Disminuye la desventaja de tener que sustituir todo el sistema en caso de avería como en el centralizado ya que solo se sustituiría la placa afectada minimizando costes. También soluciona el problema de que si la batería a controlar tiene muchas celdas y la placa quedaría grande ya que las mediciones se reparten entre el número de placas instaladas. Sin embargo, resulta un poco ineficiente económicamente y en cuanto a volumen ya que se está dotando la capacidad de realizar las comunicaciones a todas las placas sin que sea completamente necesario. Esto se soluciona con las siguientes dos tipologías.
- Maestro-esclavo: El BMS está compuesto por varias placas, pero a diferencia del modular en el que todas las placas eran iguales, ahora hay dos tipos de placas: Esclavo y maestro. El esclavo realiza las medidas de un grupo de celdas y el maestro es el que trata esta información y se encarga de las comunicaciones. Es similar al modular, pero se consigue que los esclavos sean más baratos ya que no necesitarán la electrónica correspondiente a las comunicaciones.

- Distribuido: En esta tipología cada celda tiene su pequeño circuito que se encarga de las mediciones y cada una de las placas se unen mediante uno o dos cables para realizar las comunicaciones hasta un circuito final que es el controlador. Es más caro que las anteriores tipologías debido al alto número de placas que se utilizan y ocupa más espacio.

A continuación se puede ver una representación de cada una de las tipologías [2]:

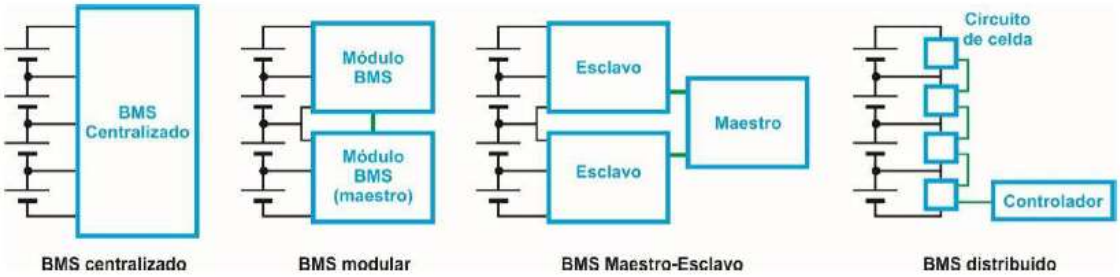


Ilustración 1. Topologías de BMS

En la motocicleta hay demasiadas celdas a medir como para hacerlo en una placa utilizando un BMS centralizado, con lo que esta sería demasiado grande y poco sustituible. El BMS modular se descarta también por el ahorro de componentes que supone utilizar un BMS maestro-esclavo frente al modular. El BMS distribuido se ha descartado también porque no es necesario que cada placa de mediciones comparta datos con las otras placas. Simplemente se harán comunicaciones de cada placa de mediciones a la placa que gestione las comunicaciones y otros controles. Por lo tanto, la topología maestro-esclavo resulta la más adecuada.

Grupo de alternativas 2: Tipo de balanceo

Se denomina el balanceo de una batería como la acción de asegurar que todas las celdas conectadas en serie mantengan el mismo nivel de carga, o lo que es lo mismo, el voltaje. Por factores relacionados con la fabricación o configuración de las mismas celdas siempre habrá ligeras diferencias en la capacidad relativa entre ellas. El BMS corta el suministro de energía cuando detecta que una de las celdas ha llegado a su voltaje mínimo en caso de descarga o a su voltaje máximo en caso de carga. Supongamos estas dos situaciones:

- El BMS cortará la carga de la batería cuando detecte que una de las celdas ha llegado a su voltaje máximo de carga. Debido a las imperfecciones de la batería puede que el resto tengan un voltaje menor en ese mismo instante. Eso significa que una o

varias de las celdas no se han cargado por completo y el BMS habrá dado por finalizada la carga.

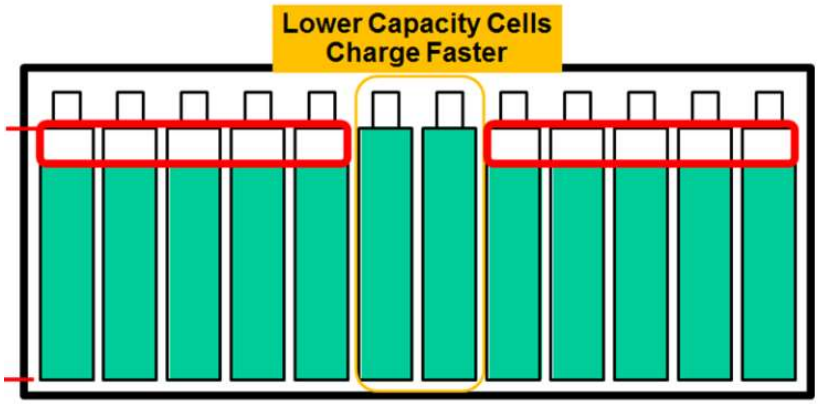


Ilustración 2. Celdas desbalanceadas en carga máxima [3]

- El BMS corta la descarga de la batería cuando detecta que una celda ha llegado a su voltaje mínimo. Partiendo del caso anterior en el que una celda estaba menos cargada que la otra, es sencillo ver que la celda menos cargada será la primera en llegar al voltaje mínimo. Esto llevará a no haber aprovechado toda la energía que tenía en su interior.

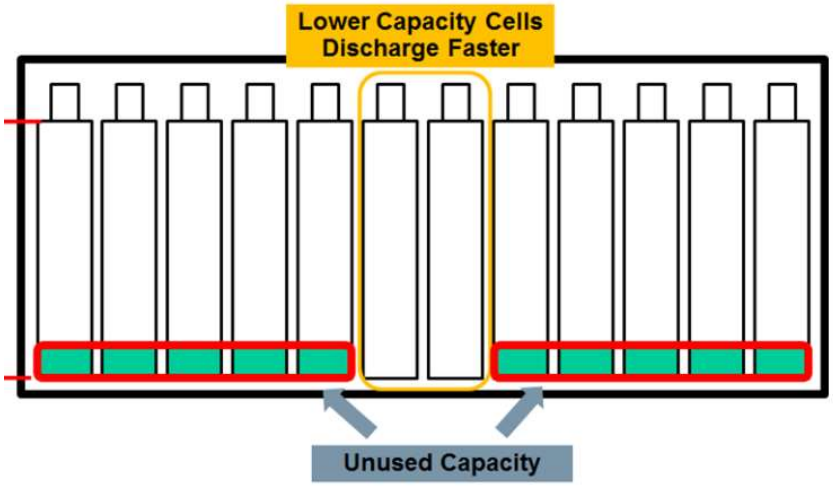


Ilustración 3. Celdas desbalanceadas en carga mínima

Aumentando el número de ciclos de carga y descarga el desbalanceo de las celdas sería cada vez mayor y, por lo tanto, el rendimiento cada vez peor. Con el fin de evitar este fenómeno se hace uso de uno de los métodos de balanceo que se explican a continuación. Estos métodos se pueden dividir en dos grandes grupos: Balanceo activo y pasivo.

Los métodos de balanceo activo tratan de pasar el exceso de carga de algunas celdas a las de menor carga tratando de desperdiciar la mínima energía posible. Como gran desventaja tienen que son mucho más costosos y necesitan más espacio físico. Como estos últimos dos puntos son muy importantes en el caso de la motocicleta se descarta directamente el estudio de los mismos en este proyecto. Además, requieren generalmente un control más complejo que con el balanceo pasivo.

El balanceo pasivo disipa la energía sobrante de las celdas de mayor carga directamente en forma de calor. Es un método más ineficiente que el balanceo activo, pero suficiente para la competición. Como la motocicleta está diseñada para desempeñar el máximo rendimiento en carrera, no resulta tan importante disponer de un método de balanceo algo más ineficiente a la hora de cargarlo en boxes si este hace que la placa sea más pequeña y pese menos para la carrera. Además, el control del balanceo pasivo es muy sencillo.

Ambos métodos pueden dividirse en subgrupos como se puede ver en la siguiente imagen [4]. Solamente se van a analizar los métodos pasivos en este proyecto por las razones anteriormente comentadas.

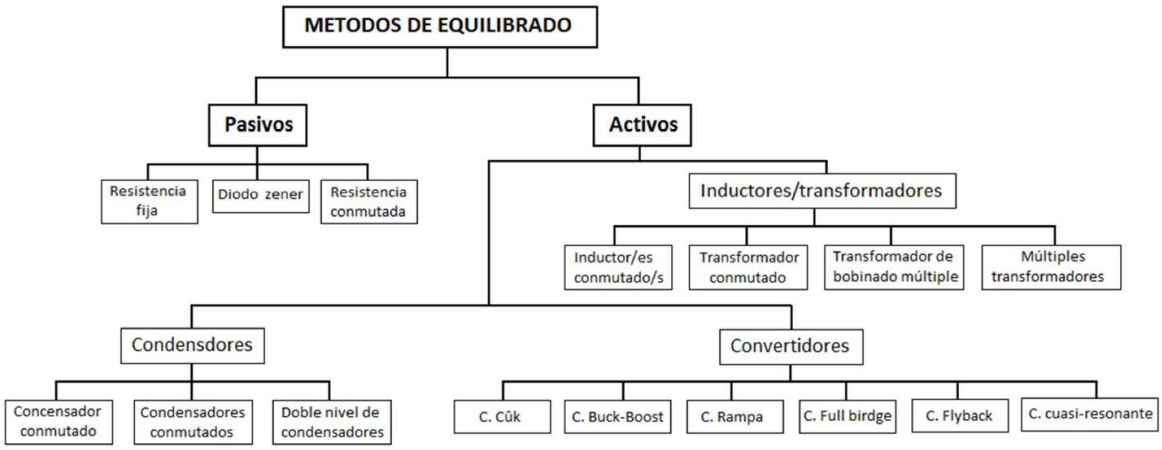


Ilustración 4. Tipos de balanceo

Equilibrado pasivo: Resistencia fija

Consiste en colocar una resistencia en paralelo a cada celda de tal forma que la resistencia limite la tensión a la que puede llegar la celda. Se mantiene el balanceo entre las celdas ya que, si se da el caso de que una celda tiene mayor tensión, la descarga que esta sufrirá a causa de su resistencia también será mayor y volverá al mismo voltaje que el resto. Es el método más sencillo, pero también obsoleto debido a la alta ineficiencia energética.

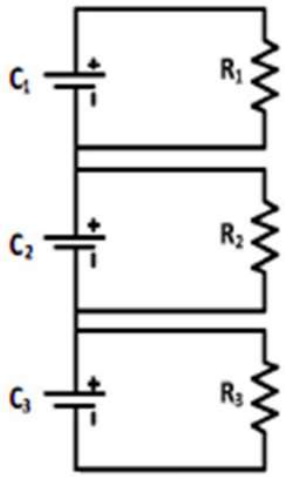


Ilustración 5. Balanceo por resistencia fija

Equilibrado pasivo: Diodo zener

El uso de un diodo zener en lugar de una resistencia fija reduce el problema de la gran pérdida de energía del caso anterior. Protege a las celdas de sobrecargas ya que estas no sobrepasarán el voltaje del diodo. Sin embargo, no protege ante sobredescargas y los diodos deben soportar la corriente de carga de las celdas suponiendo esto problemas económicos y térmicos.

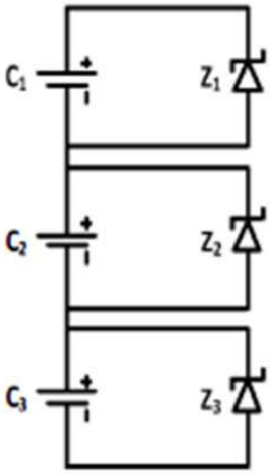


Ilustración 6. Balanceo por diodo zener

Equilibrado pasivo: Resistencia conmutada

Se trata de una versión mejorada a la resistencia fija. Se sigue utilizando una resistencia fija para disipar la energía sobrante de cada celda, pero solo cuando es necesario. Para ello se coloca un interruptor en serie a cada resistencia y se decide cuando se cierra y cuando no.

Estos interruptores pueden ser transistores, mosfets u otros elementos dependiendo de la intensidad que tengan que soportar. El esquema es el siguiente:

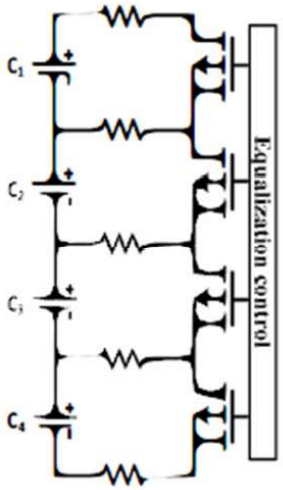


Ilustración 7. Balanceo por resistencia conmutada

La velocidad de balanceo depende del paso de corriente por el mosfet y la resistencia. El control de los mosfets se realiza con el microcontrolador, que es el encargado de medir los voltajes y decidir en cada instante cuales de los mosfets deben estar conduciendo.

Existen diferentes algoritmos de control. El más sencillo consiste en activar el balanceo de todas las celdas salvo la del voltaje más bajo. De esta forma se estará cargando más rápido la celda que menor voltaje tenga para que se iguale con el resto. Otra opción es activar solamente los que estén por encima de un rango de voltaje (10mV por ejemplo) sobre la celda de voltaje más bajo. La elección de uno u otro influye básicamente en la velocidad de balanceo, calentamiento del BMS y complejidad del algoritmo de control entre otros factores. Es un sistema de balanceo más caro que los dos anteriores, pero sigue siendo más barato que lo métodos de balanceo activos ya descartados. Por todos estos motivos se ha decidido utilizar la topología de resistencia conmutada

Además, tiene una gran ventaja frente a muchos BMS comerciales ya que permite empezar a balancear la batería en cualquier nivel de carga de la batería. Muchos de los comerciales solo empiezan a balancear las celdas que llegan a su nivel máximo de carga. Si se supone el siguiente estado de carga de una batería:

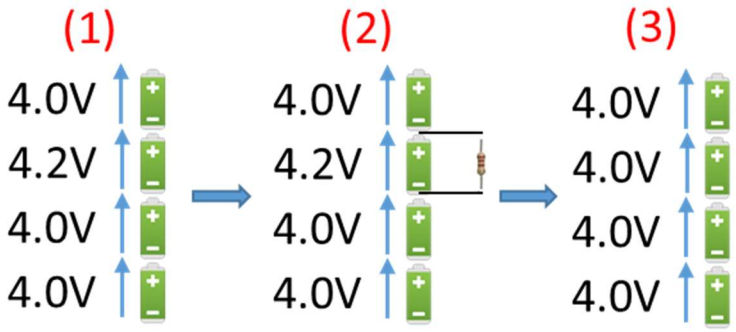


Ilustración 8. Balanceo de celdas

En el estado de carga (1) en la imagen, el BMS comenzaría el balanceo de la celda de 4.2V (2) hasta que recuperara el mismo voltaje que las otras (3), esto es, a 4.0V. En este caso se ha supuesto que no está circulando intensidad por ninguna de las otras celdas.

Ahora se plantea el caso anterior, pero cargando simultáneamente todas las celdas, incluida la del balanceo. La intensidad máxima a la que se podrá cargar esta será la intensidad de balanceo de la celda ya que, si se le suministra una intensidad mayor a la que se está disipando en el balanceo, esta seguirá cargándose hasta que se estropee o se incendie por sobrecargarla. Esto limita la intensidad de carga de toda la batería cuando una de las celdas llega a su voltaje máximo. Los BMS comerciales son capaces de disipar unos 100mA, con lo que ese sería el límite de intensidad de carga cuando se está llegando al tope. Los hay hasta más ineficientes, que cortan directamente la carga hasta que las celdas se balancean, lo que hace la carga aún más lenta.

Si se utilizara un BMS habitual para la batería de la motocicleta, suponiendo que hubiera una diferencia de carga del 5% entre la celda más cargada y el resto, el tiempo de balanceo sería el siguiente:

$$Capacidad\ de\ cada\ rama\ de\ la\ motocicleta\ en\ paralelo = 60\ Ah$$

$$Capacidad\ de\ la\ rama\ a\ disipar\ en\ balanceo = 60x0.05 = 3\ Ah$$

$$Intensidad\ de\ balanceo = 100\ mA = 0.1\ A$$

$$Tiempo\ de\ balanceo = \frac{3\ Ah}{0.1\ A} = 30\ h$$

Como se puede observar se necesita una intensidad de balanceo mayor a la ofrecida por la mayoría de los BMS del mercado. Hay que buscar un compromiso entre la velocidad de balanceo y el calor a disipar en cada resistencia del balanceo. Se ha decidido escoger una intensidad cercana a 0.3A, reduciendo 3 veces el tiempo necesario.

Por lo tanto, en la situación planteada se necesitarían 30h para completar la carga de la batería con un BMS comprado mientras que con el de este proyecto se necesitarían 10h. Además, es muy importante tener en cuenta que el BMS comprado haría este balanceo a partir del instante en que la celda más cargada llegue a los 4.2V como se ha planteado. En el BMS del proyecto se podrá empezar a balancear las celdas en cualquier momento, antes de que esto ocurra, gracias a que se tendrá control absoluto con el microcontrolador, aprovechando así todo el tiempo de carga. Esto quiere decir que, si por ejemplo hicieran falta 2h cargando la batería para que se diera la situación planteada anteriormente, con el BMS diseñado ya se habría estado balanceando la batería 2h, reduciendo aún más el tiempo de balanceo. Esto le da una ventaja sustancial frente a otros BMS.

Grupo de alternativas 2: Medición de voltajes

Hay que adaptar los voltajes de las celdas para poder medirlos con un microcontrolador. Esto se debe a que la medición de la tensión de cada celda está referenciada a su propio borne negativo, con lo que, aunque cada celda solo tenga un voltaje del rango 2.5-4.2V, el microcontrolador no medirá lo mismo (el microcontrolador estará referenciado al borne negativo de la primera celda y no la que quiere medir). Este fenómeno se puede ver a continuación:

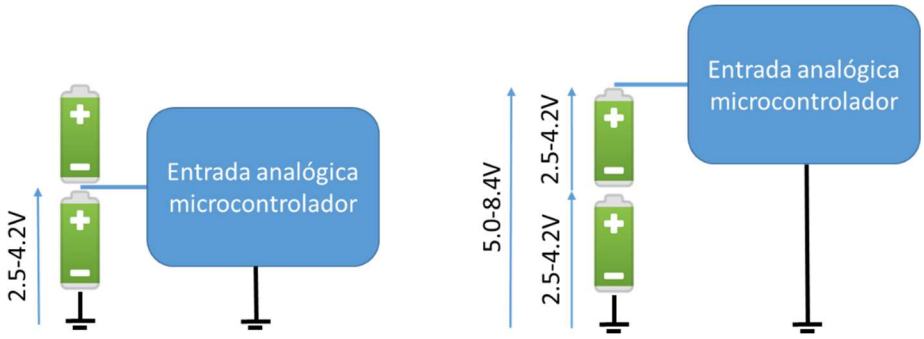


Ilustración 9. Medición de voltajes de celdas en serie

Se puede como el microcontrolador al medir el voltaje de la segunda celda verá una tensión entre 5.0 y 8.4V, valor resultante de suma de las dos primeras celdas. Este fenómeno imposibilita la medida directa de las tensiones ya que si así se hiciera se dañaría el microcontrolador por alimentarlo con tensiones mayores de las permitidas (generalmente funcionan a 3.3 0 5V). Para evitar esto se debe medir el voltaje de cada celda de forma diferencial, esto es, tomar la medida del borne negativo y positivo, obtener el diferencial de ambos y referenciarlo al mismo punto que el microcontrolador. A esta etapa se le llamará la etapa de acondicionamiento.

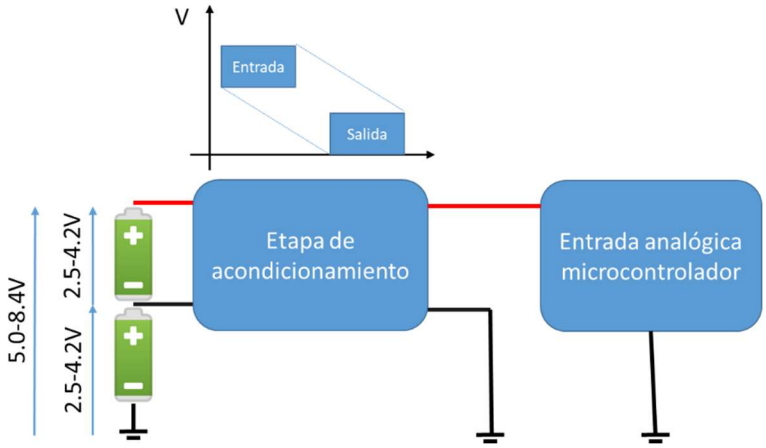


Ilustración 10. Etapa de acondicionamiento

Otro problema importante a resolver es que hay que medir 27 tensiones diferentes y el microcontrolador no tendrá tantas entradas analógicas. En consecuencia, no queda más remedio que multiplexar las medidas de las celdas. Los multiplexores son dispositivos que permiten controlar un mayor número de señales con un número menor de entradas [5]. Básicamente funcionan como un interruptor de señales. Son capaces de juntar 2,4,8,16 o incluso 32 señales en una única. Para ello requieren cierto control de señales, ya sea utilizando entradas digitales o con algún protocolo de comunicación. Dicho esto, se pueden utilizar dos topologías diferentes:

- Colocar una etapa de acondicionamiento para cada conjunto serie de celdas y, una vez acondicionados todos los voltajes, multiplexarlos hasta la entrada o entradas analógicas. La elección del multiplexor se hace más sencilla ya que los voltajes están ya acondicionados. Sin embargo, se utiliza una mayor cantidad de componentes que en la siguiente tipología ya que se utilizan tantas etapas de acondicionamiento como celdas en serie.
- Colocar el multiplexor o los multiplexores directamente a las celdas y una única o un par de etapas de acondicionamiento entre la salida del multiplexor y las entradas analógicas del microcontrolador. Son necesarios menos componentes electrónicos, pero los multiplexores deben soportar la alta tensión compuesta por el conjunto de celdas, ya que se están pasando las señales sin acondicionar por el mismo

Las dos tipologías se pueden ver en la siguiente imagen:

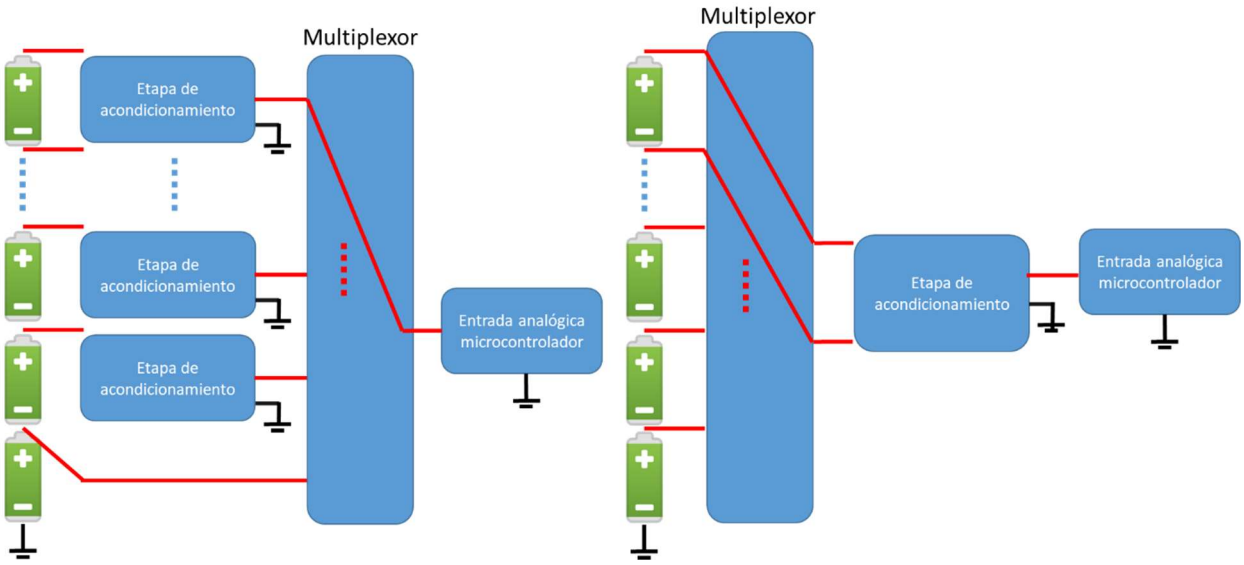


Ilustración 11. Distintos tipos de etapas de acondicionamiento

La alternativa adoptada es la segunda, la de multiplexar primero las señales para así reducir el número de etapas de acondicionamiento. Como se necesita un menor número de componentes para obtener el mismo resultado no solo el coste total de la placa será menor, sino que el espacio necesario también lo será. Respecto a la desventaja de hacer circular señales de altos voltajes por el multiplexor, se reduce el número de multiplexores del mercado que se puedan utilizar, pero sigue habiendo un amplio abanico a elegir.

Grupo de alternativas 3: Topología de medición de intensidades

Como ya se ha comentado anteriormente en el apartado de descripción de requerimientos es necesario medir la intensidad que circula por la batería, tanto en carga como en descarga. Esta intensidad es de corriente continua ya que se mide directamente en el borne positivo de la batería. Se podría medir también la corriente alterna circulada en los bornes del motor, pero resultaría útil ya que no se estarían midiendo las pérdidas que hay aguas arriba. Existen diferentes maneras de medir la corriente continua:

- Resistencia shunt: Consiste en colocar una resistencia en serie a la salida de la batería y medir la caída de voltaje en la resistencia. Conocido el valor de la resistencia y la caída de voltaje del mismo (se mediría mediante el microcontrolador) se puede calcular la intensidad que circula por él mediante la ley de Ohm. Es una forma muy sencilla de medir la intensidad, pero conlleva las siguientes desventajas: Se trata de un método de medición invasivo no aislado ya que se actúa directamente sobre el cable de alimentación de la batería provocando

una caída de voltaje en la salida y, lo peor de todo, a intensidades tan altas resulta completamente ineficiente por la potencia que disiparía en forma de calor (las pérdidas de calor van en relación de al cuadrado respecto a la intensidad que circula por la resistencia).

- Efecto Hall: Este tipo de sensores son sensibles a las variaciones del campo magnético que lo atraviesa, generando un voltaje en su salida equivalente a la influencia magnética que lo circula [6]. Son capaces de medir corrientes DC y AC de una forma lineal, precisa y sin bajo consumo. No son invasivos y están completamente aislados del circuito principal.

El método a utilizar es el de efecto Hall por su sencillez, robustez y eficiencia. Son capaces de medir corrientes en ambas direcciones, con lo que teniendo un único sensor se podrán medir las intensidades tanto en carga como en descarga de la batería. Habrá que tener esto en cuenta a la hora de leer el voltaje proporcionado por el sensor para leerlo en el microcontrolador.

Grupo de alternativas 4: Tecnología de medición de temperaturas

Por normativa de la competición se deben colocar, al menos, 4 sensores de temperatura en la batería con el fin de garantizar su seguridad [7]. Adicionalmente, aunque no sea obligatorio, es recomendable monitorizar la temperatura del propio BMS para asegurar que este tampoco sobrepase su límite de temperatura. El tipo de sensor de temperatura a utilizar no está restringido por la competición, con lo que queda a elección libre. Por lo tanto, se pueden utilizar los siguientes tipos de sensores de temperatura [8]:

- Termopar: Están compuestos por dos hilos metálicos de diferente tipo unidos en uno de los extremos. El efecto de la temperatura hace que se cree una diferencia de tensión en el otro extremo y esta tensión es leída para obtener la temperatura. Son los más utilizados al ser económicos, de sencilla instalación y precisos en un amplio rango de medición. Sin embargo, tienen una respuesta lenta en comparación a otros tipos y necesitan una entrada analógica y una etapa de amplificación por cada sensor a instalar ya que la tensión que pueden ser capaces de generar es del orden de milivoltios.
- Termistor: Este tipo de sensor contiene un material semiconductor en la que varía su resistencia en función de la temperatura a la que esté sometida. Existen dos tipos de termistores: Los NTC, que disminuyen su resistencia al aumentar la temperatura

y los PTC que aumentan su resistencia junto con la temperatura. Al igual que con los termopares necesitan una entrada analógica por cada sensor termistor a utilizar.

- Dallas: Este tipo de sensores de temperatura están compuestos por 3 cables: Dos para alimentación y uno de datos. Este último es un bus de datos que, mediante el uso de un protocolo de comunicación específico, está unido a cada uno de los sensores tipo Dallas que se utilicen. Permiten realizar la medición de hasta 2^48 sensores mediante un único cable, esto es, un número prácticamente ilimitado de dispositivos. Son sencillos de leer en un microcontrolador con una única entrada digital, baratos y además provocan un uso menor de cables al compartir el bus de datos [9].

El sensor a utilizar para las mediciones de temperatura tanto en la batería como en el BMS es el de tipo Dallas, principalmente por el ahorro de cableado en la motocicleta. Adicionalmente habrá un termistor dentro del motor eléctrico ya que es el que viene preinstalado internamente. Por lo tanto, habrá que añadir también la posibilidad de medir este tipo de sensores.

Grupo de alternativas 5: Protocolos de comunicación

En la mayoría de microcontroladores del mercado se incluye la posibilidad de utilizar los siguientes dos protocolos maestro-esclavo: SPI e I2C. Las diferencias son las siguientes

El protocolo SPI es síncrono, Full Duplex y necesita 4 cables para realizar la comunicación. El maestro decide con qué esclavo comunicarse a través de uno de los 4 cables que utiliza. Por cada pulso de la señal de reloj se puede enviar un bit del maestro al esclavo y viceversa. Se pueden mandar secuencias de bit de cualquier tamaño y a velocidades de transmisión de hasta 8MHz. Las desventajas son que necesita hasta 4 cables, funciona para cortas distancias, no tiene mecanismo de control y detección de errores y la longitud de mensajes debe conocerse tanto por los esclavos como por los maestros [10].

El I2C es un protocolo síncrono que solamente necesita 2 cables para realizar la comunicación. Uno para la señal de reloj y otro para el envío de datos. Cada dispositivo tiene una dirección y el maestro accede a los esclavos escribiendo en el propio bus la dirección del esclavo del que quiere obtener datos. Es posible disponer más de un maestro, pero aumenta la complejidad de la comunicación y tampoco es necesario en este proyecto. A diferencia del SPI contiene un bit de validación en la comunicación dándole un grado mayor de fiabilidad. Además, tiene una velocidad de comunicación mucho mayor, de hasta 400MHz [11].

En el BMS se utilizará el I2C por la sencillez y robustez utilizando solamente dos pines del microcontrolador. La comunicación deberá hacerse de forma aislada ya que cada esclavo estará referenciada a distintas masas.

Grupo de alternativas 6: Alimentación del BMS

Se dispone de una alimentación en la motocicleta de 12V aislada respecto de la masa de la batería. Esto se debe a que distintos dispositivos de la motocicleta, incluido el BMS, deben ir aislados eléctricamente de la parte de alto voltaje (batería y controlador del motor eléctrico) por motivos de seguridad. Estos 12V se obtienen de un convertidor reductor aislado DC/DC a partir de los 113.4V máximos de la batería.

Como las medidas realizadas en los esclavos del BMS están referenciados a la celda de menor voltaje de la que obtienen medidas, no se pueden alimentar sin aislamiento galvánico. Si así se hiciera se provocarían cortocircuitos entre los distintos esclavos por interferencias de masas. Por lo tanto, de los 12V del circuito de bajo voltaje se deben obtener 5V para la electrónica de cada esclavo de forma aislada.

No obstante, no será necesario aislar la placa en el caso del maestro. Como no tiene que estar referenciado a ninguna tensión intermedia de la parte de alto voltaje como los esclavos se pueden obtener directamente 5V de los 12V sin aislamiento galvánico. De esta forma el maestro formará parte del circuito de bajo voltaje y los esclavos del alto voltaje.

A continuación se puede ver un esquema de la alimentación del BMS:

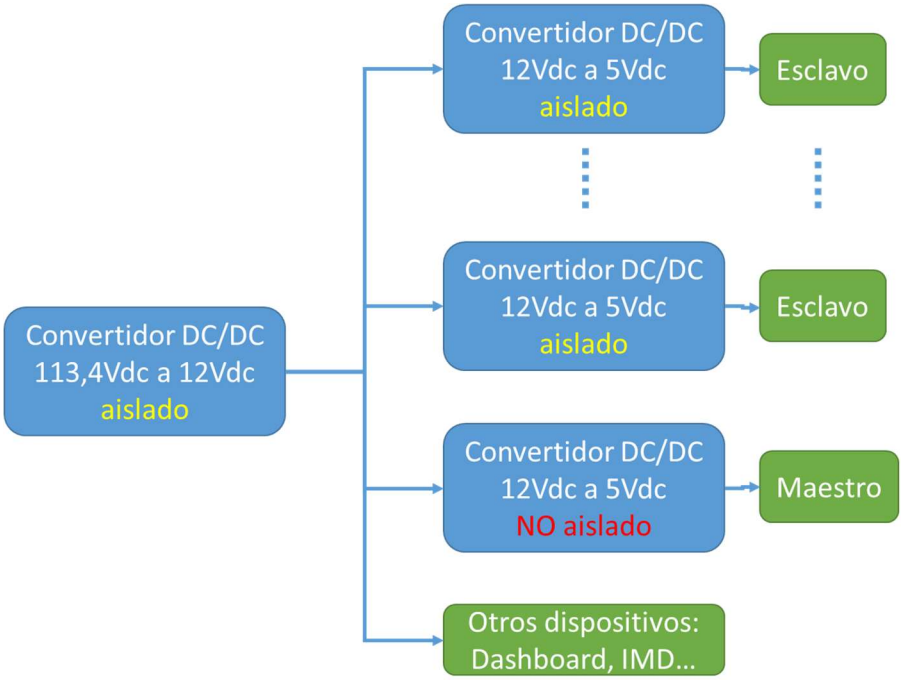


Ilustración 12. Convertidores de alimentación

Grupo de alternativas 7: Microcontrolador

Para elegir el microcontrolador es necesario tener en cuenta los pines que necesita, comunicaciones, alimentación y procesador. Cada placa esclava tiene las siguientes necesidades:

- Señales analógicas:
  - Al menos 1 entrada analógica para medir la tensión de las celdas. Solamente hará falta 1 entrada si todas las celdas se multiplexan hasta una única entrada. También se ve la posibilidad de multiplexar las celdas hasta 2 entradas. Las celdas tendrán una tensión de hasta 4.2V, con lo que, salvo que se cambie la ganancia de la tensión en la etapa de acondicionamiento hay que escoger un microcontrolador que funcione a 5V.
  - Otra entrada analógica para medir la intensidad a través del sensor Hall.
- Señales digitales: Hasta 7 pines. Se dividen en las siguientes funciones:
  - 2 o 3 pines de salida para controlar las señales de los multiplexores en función de si se usan 2 o 1 entradas analógicas respectivamente.
  - 3 pines de salida para controlar el balanceo.
  - 1 pin de entrada para leer la temperatura de los sensores Dallas.
- Comunicaciones: Hasta 4 pines de la siguiente forma:

  - USB: 2 pines digitales para programar el microcontrolador desde el ordenador.
  - I2C: 2 pines.
- Alimentación: 5V a ser posible para no tener que adaptar la señal de medida de las celdas y así poder utilizar el BMS para monitorizar otro tipo de celdas que tengan otro voltaje máximo.
- Frecuencia del reloj: Cualquier microchip de 8Mhz será capaz de cumplir el tiempo necesario para las medidas con una correcta programación.

La placa maestra necesita:

- Entradas analógicas:
  - 2 para poder leer dos termistores.
- Entradas digitales:
  - 1 para la supervisión del IMD.
  - 3 para la supervisión de 3 contactores.
- Salidas digitales:
  - 3 para el control de los 3 contactores.
- Comunicaciones:
  - 2 pines para bluetooth.
  - 2 pines para I2C.
  - 4 pines para SPI.
  - 2 pines para USB.

El hardware a utilizar podría ser alguno de los típicos microcontroladores PIC, Arduino, ESP, Attiny, etc. Viendo las necesidades del Hardware se ha optado por utilizar un chip Atmel como los que se utilizan en distintos dispositivos Arduino. Concretamente, el microchip Atmega328P. Se ha elegido este no solo porque cumple los requisitos de Hardware, sino porque ya se ha utilizado anteriormente en otros proyectos de la competición y se tiene una buena base de conocimientos acerca del mismo. Sus características son las siguientes:

Voltaje de funcionamiento	**5V**
E/S digitales	14 (pines PWM incluidos)
Pines PWM	6
Entradas analógicas	6

Corriente E/S por pin	20 mA
Memoria Flash	32 kB
SRAM	2 kB
EEPROM	1 kB

Tabla 2. Arduino Nano

Se puede insertar el Atmel principalmente de dos formas:

- Directamente en la placa: Sería la opción más compacta, pero implicaría insertar otros componentes necesarios para el funcionamiento del mismo: Condensadores, reguladores de tensión, driver que realiza las comunicaciones UART TTL y algunos más.
- A través del arduino Nano: Todos los elementos mencionados anteriormente están ya soldados en el arduino Nano, con lo que soldando el arduino ya sería suficiente.

Soldar el Atmel directamente con el resto de componentes supone un diseño más compacto y profesional. Sin embargo, apenas se ganaría espacio respecto a soldar el arduino y complicaría mucho el proceso de soldadura, ya que algunos de los elementos que habría que soldar poseen un espaciado entre patillas menor al que se puede soldar en el taller de la universidad. En conclusión, se ha optado por insertar directamente el arduino Nano.

SOLUCIÓN PROPUESTA. DISEÑO DE ALTO NIVEL

Una vez analizados todos los aspectos funcionales se explica en el presente apartado la solución propuesta de forma global. No es más que la conclusión de todos los apartados del análisis de alternativas de forma conjunta:

El BMS se va a dividir en dos placas distintas: El esclavo y el maestro. Las características de cada uno son las siguientes:

- Esclavo:
    - Microcontrolador arduino Nano.
    - Monitorización configurable desde 1 hasta 8 celdas.
    - Medición de voltaje realizadas mediante multiplexación de las señales.
    - Medición de temperaturas a través de un número prácticamente ilimitado de sensores Dallas.

- Medición de intensidad mediante el uso de 1 sensor tipo Hall.
- Balanceo de todas las celdas monitorizadas mediante resistencia conmutada.
- I2C para la comunicación con el maestro.
- Programación mediante USB.

- Maestro:
  - Microcontrolador arduino Nano.
  - Capacidad de controlar hasta 128 tarjetas esclavas.
  - Control de apertura y cierre de hasta 3 contactores.
  - Posibilidad de conectar un circuito de precarga a cada contactor controlado.
  - Supervisión del estado de apertura o cierre de hasta 3 contactores.
  - Medición de la temperatura de hasta dos termistores NTC o PTC.
  - I2C para la comunicación con los esclavos.
  - Bluetooth, CAN y USB para la comunicación con un ordenador o móvil en boxes, dashboard de la motocicleta o distintos elementos.
  - Programación mediante USB

2. METODOLOGÍA SEGUIDA EN EL DESARROLLO DEL TRABAJO

PLAN DE PROYECTO Y PLANIFICACIÓN

En el presente capítulo se describe la planificación del proyecto que comienza el 30 de septiembre de 2019 y se estima que finalice el 14 de septiembre de 2020 (es el tiempo total disponible en el último curso del máster hasta la presentación del trabajo). En primer lugar, se realiza un desglose de las distintas actividades o fases e hitos que componen el trabajo. Después se detallan los equipos o recursos técnicos utilizados. Finalmente se muestra mediante un diagrama de Gantt la planificación de forma gráfica.

Fases del trabajo

Búsqueda de información

En esta fase o plan de trabajo se realizará la búsqueda de información necesaria para adquirir los conocimientos en las diferentes fuentes bibliográficas disponibles. Es el primer paso para el desarrollo del proyecto.

Análisis de alternativas

Una vez encontrada la información y obtenida una base sólida de conocimientos en el tema se procede a estudiar las distintas alternativas existentes para la realización del proyecto. Se busca información sobre los distintos sensores, tipos de BMS, estudio del mercado, etc.

Diseño del esquema eléctrico del esclavo

Una vez elegidas las soluciones entre todas las alternativas se sabe qué tipo de dispositivos electrónicos que se van a utilizar. Se comienza a diseñar el esquema eléctrico de la placa del esclavo para posteriormente diseñar el circuito impreso. En esta etapa se trata de dejar el esquema lo más invariable posible para no tener que volver atrás en los siguientes apartados.

Diseño del circuito impreso del esclavo

Cuando el esquema eléctrico está completamente acabado se empieza con la tarea del desarrollo del circuito impreso. Es la etapa en la que se pasa del diseño esquemático al diseño real de la placa. Todos los componentes se colocan sobre la placa electrónica en la disposición que más convenga y posteriormente se realizan todas las conexiones.

Diseño del esquema eléctrico del maestro

Al igual que con el esclavo se diseña el esquema eléctrico del maestro.

Diseño del circuito impreso del maestro

Se diseña el PCB para poder dar por terminada la parte de diseño de ambas placas.

Periodo de envío de los circuitos impresos

Tras haber terminado de diseñar el circuito completo se encarga a un fabricante de circuitos impresos para que envíe el prototipo a la universidad. Es una fase en la que habrá que esperar a que la placa llegue y el tiempo depende de los fabricantes contratados.

Diseño del software del esclavo y maestro

Se diseña el programa necesario para controlar ambas placas. Para ello se utilizará el software proporcionado por arduino, el Arduino IDE.

Desarrollo del software de monitorización y análisis

Para mostrar los valores en el ordenador se diseñará utilizando el lenguaje de programación Processing. El fin es crear un programa que permita extraer los datos del BMS y graficarlos con el fin de analizar el funcionamiento de la motocicleta en la carrera.

Búsqueda de los materiales y componentes

Se buscan los componentes electrónicos, materiales y sensores necesarios en distintos proveedores para realizar todo el ensamblaje y pruebas. La búsqueda deberá hacerse una vez terminado el diseño por completo, para así evitar buscar componentes innecesariamente si se dieran cambios en el diseño. No obstante, esto no quiere decir que no se vaya a buscar ninguna información mientras se diseña la placa sino todo lo contrario, ya que la elección del componente es un punto crucial en el diseño de la placa. Esta fase trata de encontrar el precio más económico para encargar todo el material.

Periodo de envío del material

En esta fase toca esperar a que los componentes lleguen al taller. Al igual que con el encargo del circuito impreso, la duración de esta fase depende de la velocidad de los distribuidores.

Ensamblaje de las placas

Obtenidas las placas electrónicas y todos los componentes se procede a ensamblar el conjunto.

Pruebas funcionales

Una vez ensambladas las placas se comienza a hacer pruebas de cada función específica, tanto del esclavo como del maestro. Se probará a medir el voltaje, intensidad y temperatura de las celdas, así como la comunicación entre esclavo-maestro y el ordenador. Todas estas pruebas se harán por separado con el fin de facilitar la detección de errores en alguna de las etapas. Ni siquiera hará falta que los voltajes intensidades y temperaturas provengan de la batería que se vaya a usar en la motocicleta, ya que el fin de esta fase es comprobar el funcionamiento del diseño electrónico. La prueba definitiva en el entorno real es la siguiente fase.

Pruebas reales del BMS

Tras haber comprobado que todos los componentes funcionan y que la placa mide las variables como debería se montará definitivamente el BMS en la motocicleta. Se hará funcionar la motocicleta ya sea en un banco de potencia o en un circuito real. De esta forma se podrá dar por válido el principal objetivo del proyecto.

Redacción del trabajo

Esta fase se irá trabajando a lo largo de todo el curso, a medida que el proyecto vaya avanzando. Se anotará las cosas más importantes del trabajo que se vaya realizando para una mejor redacción final del trabajo.

Hitos

El cumplimiento de estos hitos es fundamental para poder llevar a cabo el proyecto a tiempo. Los hitos marcados son los siguientes:

Elección del trabajo de fin de máster

Es el primer hito y el más importante. Simboliza que para esa fecha el tema del trabajo de fin de máster ya habrá sido escogido y que se comienza a trabajarlo.

Elección de la solución

Es el momento en que tras profundizar en los conceptos teóricos se ha escogido entre cada una de las alternativas estudiadas. Llegado a este punto se empieza a diseñar tanto el esquema eléctrico como el Software. Se da comienzo a la parte práctica del proyecto.

Encargo circuitos impresos

Es el hito que marca que tanto la placa esclava como maestra ya ha sido diseñada y puede encargarse a una empresa para su fabricación. El trabajo que sigue a continuación será buscar los componentes eléctricos, pruebas funcionales, puesta en marcha en la motocicleta, etc.

Compra de material y componentes electrónicos

Se hace la compra para conseguir los componentes electrónicos, sensores y demás cosas necesarias. Todo el trabajo que queda después de este hito es mano pura mano de obra.

Entrega del trabajo

Es el último hito que marca que el trabajo ha sido terminado cumpliendo cada una de las fases planificadas. Significa que el proyecto ha sido redactado completamente y está listo para ser entregado.

Equipo y recursos utilizados

El equipo a utilizar para el diseño, programación y redacción del trabajo es el portátil personal y un ordenador ya disponible en el taller. Para el ensamblaje de las placas se utilizará un soldador, horno, estaño, pinzas y una lupa ya disponible en el taller también.

Se van a utilizar 4 softwares distintos: Arduino IDE para la programación del Arduino, Processing para la programación del software de análisis y monitorización, Autodesk Eagle para el diseño de los PCB y el MS Project para la planificación y el Gantt.

Arduino IDE y Processing son gratuitos, mientras que Autodesk Eagle y MS Project no. Sin embargo, MS Project está a disponibilidad de los alumnos en los ordenadores de la escuela de forma gratuita y Autodesk proporciona licencias gratuitas a estudiantes para sus softwares, entre ellos Autodesk Eagle.

El equipo necesario para realizar el ensamblaje de la placa esta disponible en el taller de la universidad y es el siguiente:

- Soldador de estaño.
- Bomba de calor.
- Pasta de soldar.
- Pinzas para colocar los componentes.
- Horno para la soldadura SMD.
- Lupa.

Para las pruebas de puesta punto:

- Multímetro.
- Osciloscopio.

Los recursos humanos a destinar al trabajo son los del propio autor del proyecto. La medida de la misma se va a realizar haciendo una equivalencia del valor del trabajo de fin de máster. Como este trabajo equivale a 24 créditos ECTS en el máster y cada crédito equivale a 25h, el tiempo dedicado a este será de 600h a lo largo de todas las tareas.

DIAGRAMA DE GANTT

Una vez explicadas todas las tareas a realizar e hitos a alcanzar se representan en un Gantt para tener una visión más gráfica de toda la planificación del trabajo:

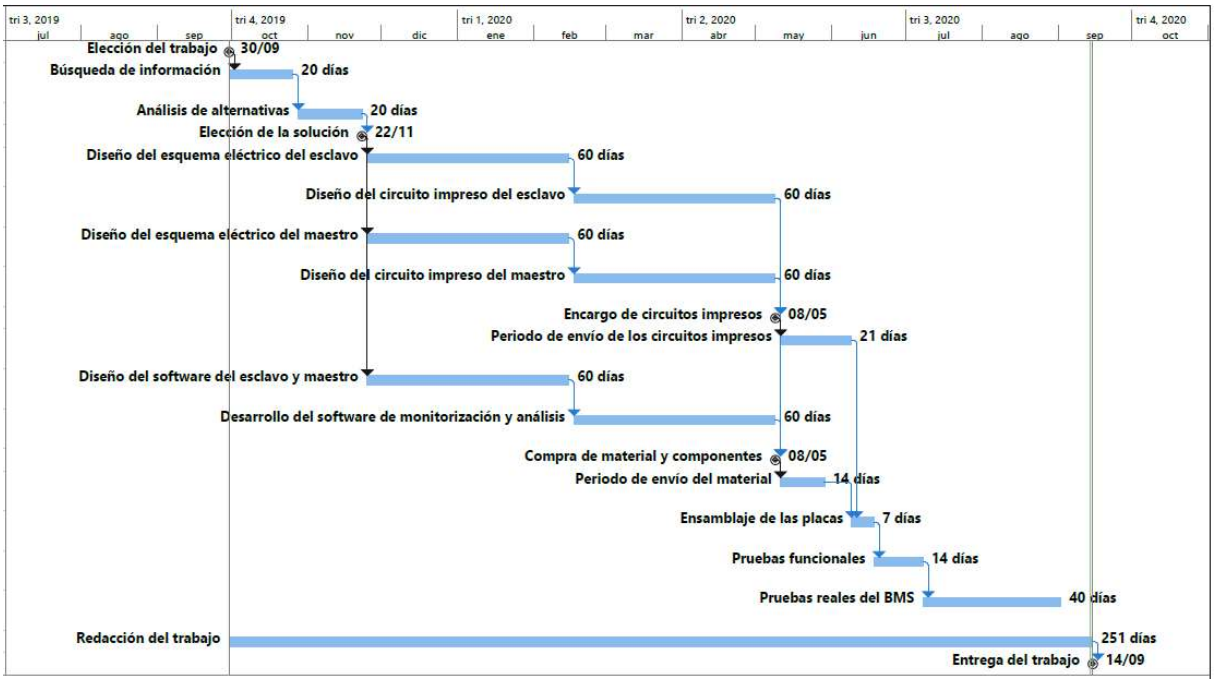


Ilustración 13. Diagrama de Gantt

DISEÑO DE PLACA ESCLAVA

En el presente apartado se aborda todo lo correspondiente al diseño esquemático de cada una de las partes del esclavo. La explicación se divide por bloques funcionales:

Diseño esquemático: Arduino Nano

El uso de pines digitales y analógicos es el siguiente:

- Entradas digitales:

- D8: Medición de temperaturas con sensor Dallas.

- Salidas digitales:
  - D3 y D4: Control de multiplexores para medición de voltajes.
  - D5 a D7: Control de balanceo.
- Entradas analógicas:
  - A3: Sensor de intensidad
  - A6 y A7: Medición de voltajes de las celdas.
- Comunicaciones:
  - D0 y D1: Comunicaciones serie. Tienen que ser estos dos pines ya que van internamente conectados en el arduino al conector micro USB.
  - D2, D10, D11, D12, D13: Comunicaciones SPI para la micro SD. Son los pines que la placa tiene reservados para tal fin.
  - A4 y A5: Comunicaciones I2C. Al igual que con el SPI, estos dos pines son los habilitados para realizar las comunicaciones I2C.

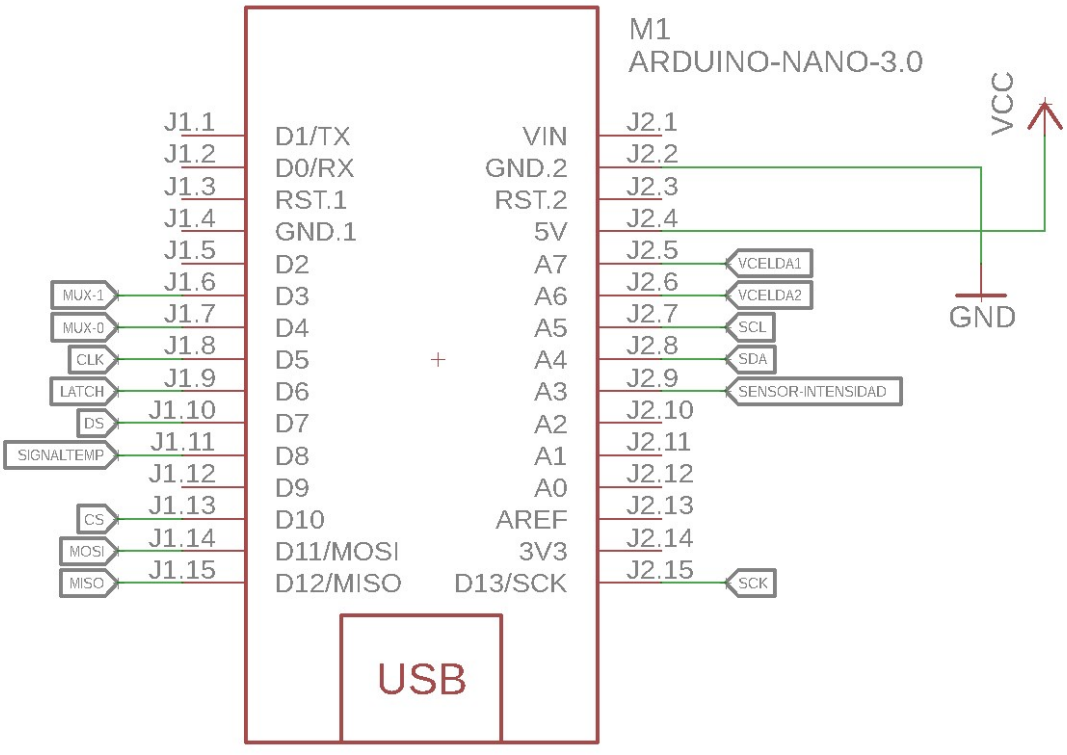


Ilustración 14. Arduino Nano en el esclavo

Como se puede observar hay pines que van a quedar inutilizados. Los que no se utilizan podrían haber sido escogidos para realizar alguna de las funciones que se realiza en otro

pin. Sin embargo, se ha decidido esta disposición de pines ya que era la que permitía un ruteado más sencillo a la hora de diseñar el PCB.

Cada uno de los pines y sus funciones se detallan en los próximos apartados:

Diseño esquemático: Balanceo

El método de balanceo a utilizar es el de resistencia conmutada. Como ya se ha explicado anteriormente se necesita controlar la conmutación de una resistencia en serie por cada celda a monitorizar. Estas conmutaciones se pueden realizar con un mosfet o transistor. Se ha optado por utilizar mosfet ya que tienen mejor capacidad de conducción de corriente que los transistores, siendo así mejor opción para el balanceo.

Serán necesarias tantas señales digitales como mosfets, esto es, 8. El arduino tiene más de 8 salidas digitales, en concreto 13. Sin embargo, se necesitan más de 5 pines para el resto de funciones que debe cumplir la placa. Por lo tanto, al no tener el número de salidas necesarias se recurre a la opción de utilizar un *shift register* o registro de desplazamientos. Los *shift register* pertenecen a una familia de chips que aceptan una entrada de bits en serie y los sacan en 8 pines paralelos. Son utilizados para ampliar el número de salidas digitales que se dispone [12]. Funciona mediante la comunicación serie síncrona. Es decir que usa un pin para enviar los bits en serie (el Data pin) y un segundo pin (el Clock pin) para indicar cuando hay que leer el bit.

Cuando los 8 bits se han leído en el registro un tercer pin (Latch pin) escribe estos bits en los pines de salida del chip y los mantiene hasta que se reciban nuevos datos.

Se necesita un shift register con las siguientes características: Voltaje de funcionamiento 5V, 8 salidas digitales y capacidad de conducir una corriente de al menos 10mA por cada salida digital para que el balanceo funcione correctamente. Un shift register típico con estas características y coste económico es el SN74HC595D [13].

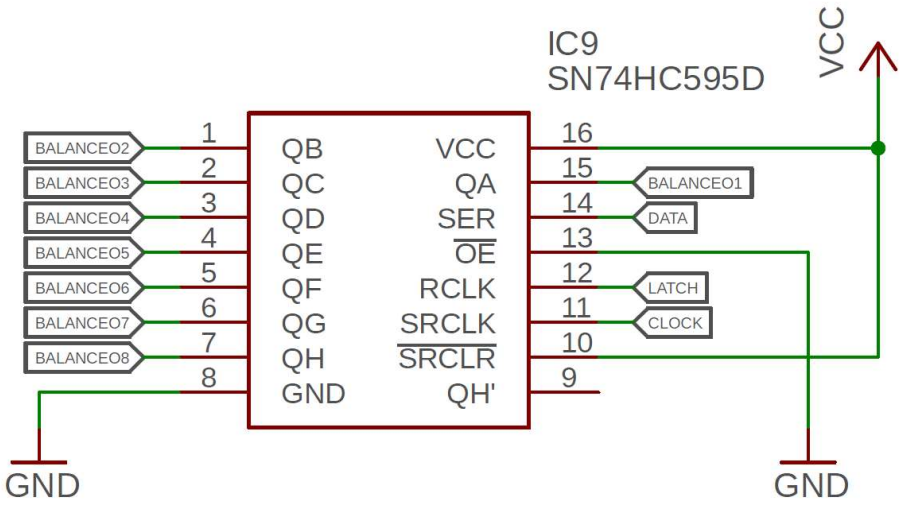


Ilustración 15. Shift register

Los pines numerados desde "BALANCEO1" hasta "BALANCEO8" son las salidas digitales del shift register. El resto son pines de alimentación y de control ya comentados. El pin 10 debe conectarse a 5V también para activar el funcionamiento del shift register.

Cada una de estas salidas digitales no se puede conectar directamente al mosfet correspondiente. Al igual que con la medición de voltajes de las celdas no se puede hacer una conexión directa, ya que los 5V de la salida digital están referenciados a la referencia del arduino y no a la de cada celda. En consecuencia, se va a utilizar el siguiente esquema [14]:

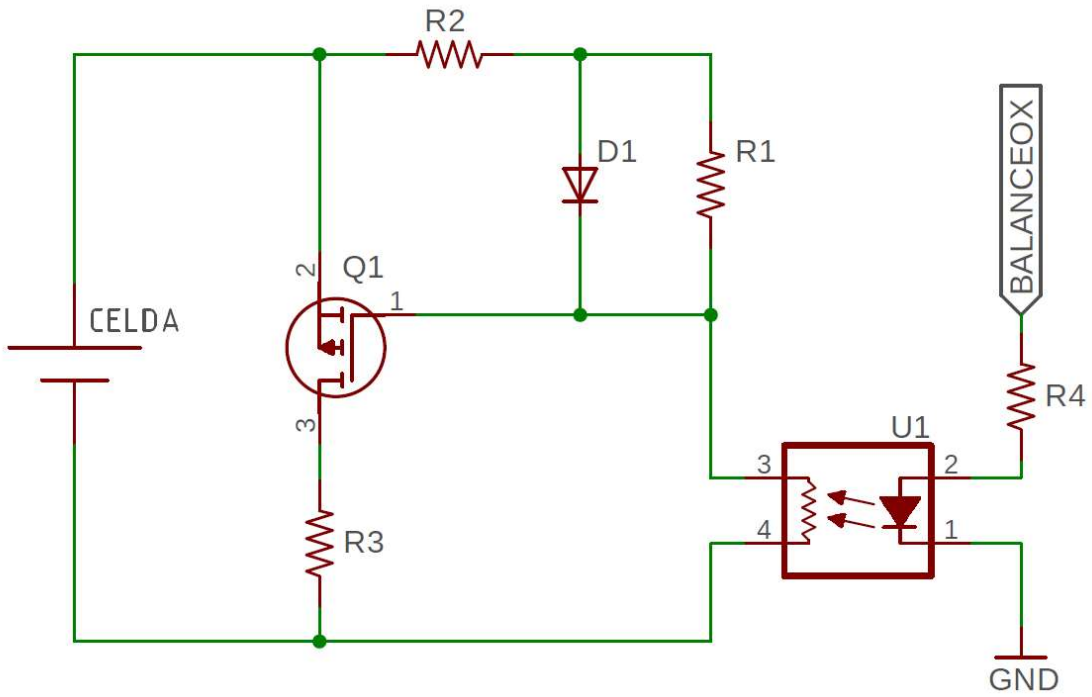


Ilustración 16. Circuito de balanceo

Cuando el voltaje en la salida digital "BALANCEOX" (representa a cada una de las salidas del multiplexor) es nulo el optoacoplador no conduce. En consecuencia, tampoco circula corriente en el lado izquierdo del circuito y tanto el led como el mosfet están en corte.

Cuando el voltaje de la salida digital es 5V el optoacoplador conduce una corriente que activa al led D1 y genera un voltaje suficiente para que el mosfet de canal P entre en saturación. Al estar este en saturación la caída de voltaje en el mismo será prácticamente nula y casi toda la potencia de balanceo la disipará la resistencia R3.

El led D1 sirve para que indique cuando la celda está balanceándose y, según su datasheet, una corriente de 7mA será suficiente para iluminarlo [15].

Gracias a la resistencia R1 las corrientes parásitas de Q1 conseguirán retornar cuando el transistor del optoacoplador esté en corte. También aumenta la fiabilidad del sistema ya que permite seguir funcionando el balanceo aun dándose el caso de que el led se estropeara. Se escoge un valor de resistencia alto para poder cumplir su función, pero no excederse en el consumo de corriente (560 kΩ, por ejemplo). Partiendo de los siguientes datos se realizan los siguientes cálculos:

$$I_{D1} = 7\ mA\ (datasheet)$$

$$V_{D1} = 1.25\ V\ (datasheet)$$

$$V_{R1} = V_{D1} \rightarrow I_{R1} = \frac{V_{D1}}{R1} = \frac{1.25}{560.000} = 2.23x10^{-6}\ mA \approx 0\ \text{mA}$$

$$I_{R2} = I_{R1} + I_{D1} = 7\ mA \rightarrow R2 = \frac{V_{R2}}{I_{R2}} = \frac{4.2 - 1.9}{0.007} = 330\ \Omega$$

El valor de R2 se ha calculado suponiendo un voltaje de 4.2V en la celda ya que es el momento en el que causará una mayor circulación de corriente. También se ha calculado la intensidad que circulará por el opto y será de 7mA aproximadamente. El optoacoplador a utilizar es el "TLP293-4(GB-TP,E)" por disponer de 4 canales en un encapsulado compacto, buena calidad precio y un *"current transfer ratio"* mínimo de 100% [16], lo que garantiza que si se hacen circular 7 mA por el lado derecho del optoacoplador se obtendrán al menos los 7 mA necesarios en el lado izquierdo (según el esquema mostrado).

Para garantizar esa corriente de 7mA se ha colocado una resistencia R4. La caída de voltaje en el optoacoplador es de 1.25 V para una corriente de 7 mA según el datasheet. Por lo tanto:

$$V_{balanceoX} = 5\ V = V_{opto} + V_{R4} \rightarrow V_{R4} = 5 - 1.25 = 3.75\ V$$

$$R4 = \frac{V_{R4}}{I_{R4}} = \frac{3.75}{0.00} = 535\ \Omega$$

Solamente queda por calcular R3. La caída de tensión en esta resistencia será como máximo 4.2 V simplificando que el mosfet tiene una caída de voltaje prácticamente nula. Además, se ha establecido como requisito una potencia de disipación de calor máxima de 2W. Por lo tanto:

$$P_{R3} = V_{R3}xI_{R3} \rightarrow I_{R3} = \frac{P_{R3}}{V_{R3}} = \frac{2}{4.2} = 0.476\ A$$

$$R3 = \frac{V_{R3}}{I_{R3}} = \frac{4.2}{0.476} = 8.82\ \Omega$$

Buscando distintas resistencias que sean capaces de disipar esa potencia entre los distintos tamaños y valores se ha llegado a la conclusión de utilizar una resistencia de valor 15 Ω con una capacidad de disipación de hasta 5W. Es de encapsulado tipo radial. Por lo tanto, la potencia a disipar e intensidad de balanceo serán:

$$V_{R3} = R_3xI_{R3} \rightarrow I_{R3} = \frac{V_{R3}}{R_3} = \frac{4.2}{15} = 0.28\ A$$

$$P_{R3} = \frac{{V_{R3}}^2}{R_3} = \frac{4.2^2}{15} = 1.176\ W$$

El mosfet debe ser de canal P para funcionar en la configuración del esquema. Además, deberá ser capaz de soportar 0.28 A de corriente por el mismo, con lo que interesa que tenga

una resistencia de conducción lo más baja posible. También deberá necesitar un voltaje bajo para entrar en saturación ya que el voltaje de la celda oscilará entre 2.5 y 4.2V. Un modelo adecuado que reúne estas características es el FDN304P.

Como aclaración cabe comentar que el circuito de la anterior figura sirve solamente para el balanceo de cada celda. Como en cada placa esclava habrá posibilidad de balancear hasta 8 celdas simultáneamente, este circuito estará repetido 8 veces en cada placa.

Diseño esquemático: Medición de voltajes

Como ya se ha explicado en el análisis de alternativas se ha optado por multiplexar los voltajes de las celdas para reducir el número de entradas analógicas y etapas de acondicionamiento necesarias. Como se tienen que medir hasta 8 voltajes y, teniendo en cuenta que el arduino Nano dispone de 6 entradas analógicas (ya que 2 se utilizan para las comunicaciones I2C porque comparten pin), surgen las siguientes posibilidades:

- Multiplexar las 8 señales hasta 4 y utilizar 4 etapas de acondicionamiento con 4 entradas analógicas.
- Multiplexar las 8 señales hasta 2 y utilizar 2 etapas de acondicionamiento con 2 entradas analógicas.
- Multiplexar las 8 señales hasta 1 y utilizar 1 única etapa con una sola entrada analógica.

Como las 3 soluciones son factibles a nivel electrónico se ha tomado la decisión en función del coste económico y espacio necesario en la placa para cada opción. Tras una búsqueda en el mercado para las distintas posibilidades se ha visto que aproximadamente los componentes de la etapa de acondicionamiento cuestan el doble que cada multiplexor. Por lo tanto, la primera opción ha sido descartada por su coste y necesitar algo más de espacio. Entre la segunda y la tercera opción se ha decidido escoger la segunda ya que ambas tenían un coste similar, pero la segunda opción ha resultado más sencilla para optimizar el tamaño del PCB.

Para escoger el modelo del multiplexor se debe tener en cuenta principalmente los siguientes dos factores:

- El voltaje que son capaces de soportar ya que el voltaje de las celdas se suma.
- El número de canales que tiene. Existen multiplexores que multiplexan 8 señales en 1 en un solo encapsulado. También hay modelos con el mismo encapsulado que el

anterior, pero ahora lo que hacen es tener dos circuitos para multiplexar 4 canales en 1. La diferencia se puede ver a continuación:

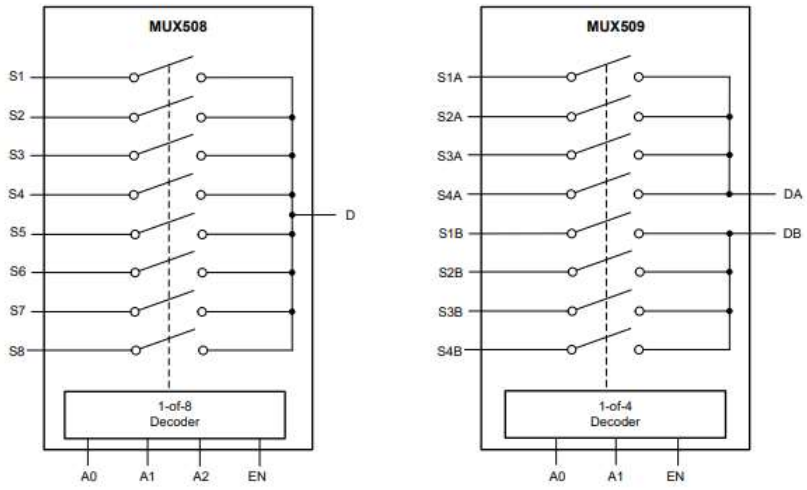


Ilustración 17. Multiplexor MUX508 y MUX509

Para decantarse por un multiplexor u otro se ha probado a colocar los dos a la hora de diseñar el PCB y se ha concluido que el de dos canales ofrece un enrutamiento más sencillo. Además, aumenta el número de multiplexores entre los que se puede elegir ya que el de dos canales deberá soportar una tensión de 16.8V (4.2V x 4 celdas) mientras que el de un solo canal 33.6V (4.5V x 8celdas). El modelo que se ha decidido utilizar por lo descrito anteriormente, por el coste y tipo de encapsulado sencillo de soldar es el MUX509. El diagrama de funcionamiento es el siguiente:

EN	A1	A0	STATE
0	X(1)	X(1)	All channels are off
1	0	0	Channels 1A and 1B on
1	0	1	Channels 2A and 2B on
1	1	0	Channels 3A and 3B on
1	1	1	Channels 4A and 4B on

Ilustración 18. Tabla de verdad del multiplexor [17]

Como se pueden conectar hasta 4 celdas por multiplexor se van a conectar las primeras 4 celdas a un multiplexor y las otras 4 al segundo. Si se conectan entre sí los pines A0 de ambos multiplexores y lo mismo para el A1 se logra controlar ambos multiplexores simultáneamente. De esta forma se acondicionarán dos celdas a la vez en todo momento para medirlas con las dos entradas analógicas. El esquema de conexiones seguido es el siguiente:

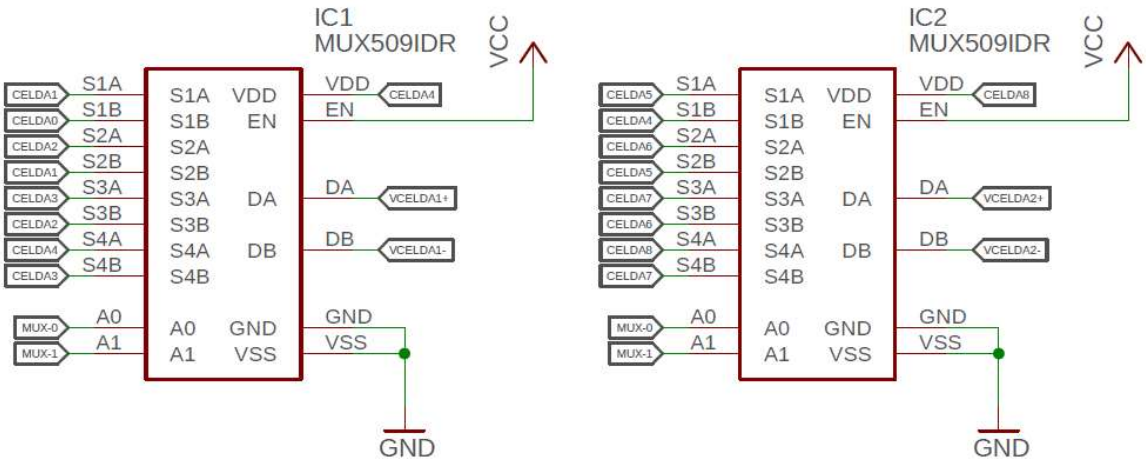


Ilustración 19. Conexiones del multiplexor

Como se puede observar el primer multiplexor multiplexa las primeras 4 celdas y el segundo las 4 restantes. Sus señales multiplexadas salen por los pines de salida "DA" y "DB" de cada multiplexor. Estos pines de salida se conectan con la etapa de amplificación que a continuación se explica.

Cada multiplexor debe ser alimentado en el pin "VDD" al voltaje máximo que se vaya a circular por el dispositivo. El voltaje positivo de la celda 4 para el primero y el de la celda 8 para el segundo. Si se alimentara el multiplexor a VCC (5V) no se transmitirían bien las señales ya que, según el datasheet, el voltaje máximo que se puede multiplexar es VDD+2V. Si se alimentara a 5V solo se podrían multiplexar señales de hasta 7V. Alimentándolo al voltaje máximo del grupo de celdas se soluciona el problema.

Ambos multiplexores están referenciados al mismo punto, esto es, a la referencia del arduino nano. Así se pueden controlar los multiplexores mediante salidas digitales sin adaptar las señales. Si la referencia fuera diferente habría que adaptar las señales A0 y A1 mediante el uso de optoacopladores. Es por eso que se simplifica el circuito referenciándolo al mismo punto. El inconveniente que puede surgir a causa usar la misma referencia es que en el segundo multiplexor, aunque se mida el voltaje de 4 celdas, la tensión a multiplexar será la de 8 celdas ya que se ha referenciado al borne negativo de la primera celda y se está midiendo hasta la octava. Esto crea una diferencia de tensión de 33.6V, pero no supone un problema para el MUX509 ya que soporta voltajes de hasta 40V. Es uno de los criterios de selección que se ha comentado antes.

El pin EN se ha conectado directamente a 5V ya que, como se ha visto en la tabla de verdad, es necesario poner en estado de HIGH ese pin para activar el multiplexor. La parte de control solo necesita un voltaje mínimo de 2V para cambiar a estado HIGH, con lo que 5V serán suficientes. Los pines A0 y A1 de control están conectados mediante la etiqueta "MUX-0" y "MUX-1" respectivamente a dos salidas digitales del arduino.

La etapa de acondicionamiento consiste en adaptar el voltaje de cada celda para referenciarlo a la masa del arduino. Para ello debe ser capaz de admitir hasta 33.6V y tener una ganancia unitaria en el acondicionamiento, ya que el arduino puede medir voltajes de hasta 5V. Para ello se ha decidido utilizar el dispositivo AD628ARZ. Se trata de un amplificador operacional en modo diferencial con un amplio rango de voltajes de entrada, ganancia configurable y mínimo desfase en el voltaje de salida teórico. Su esquema es el siguiente:

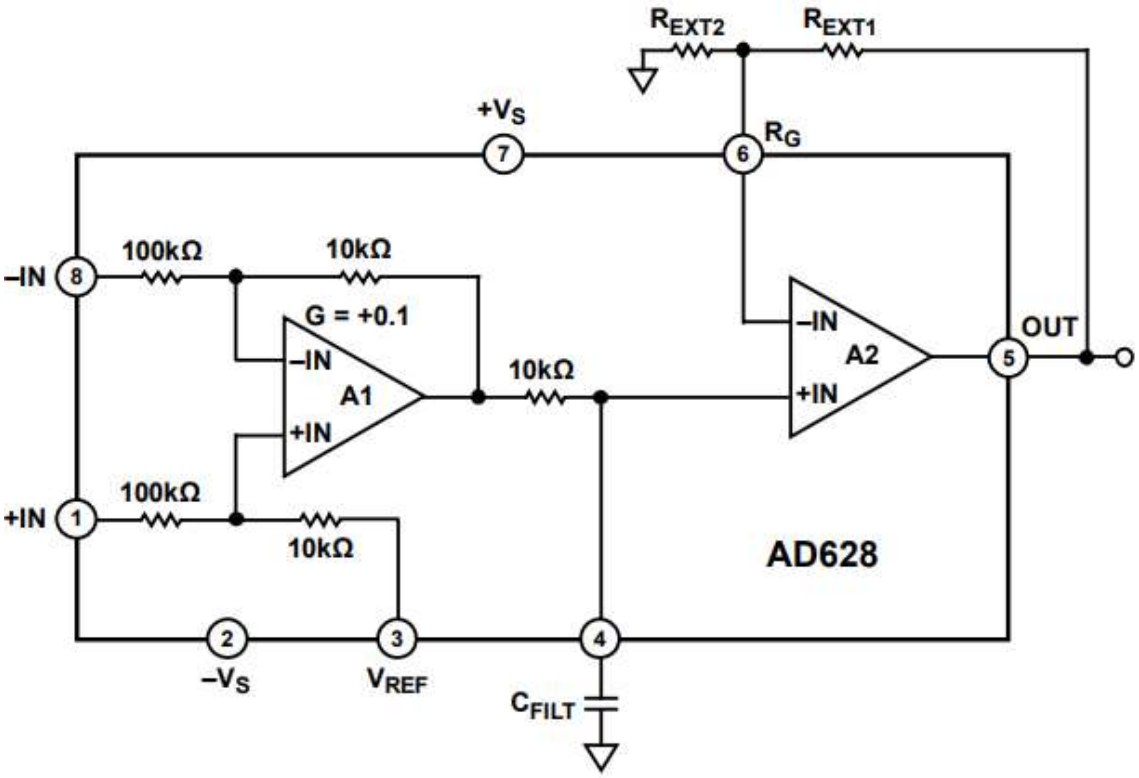


Ilustración 20. Esquema interno del AD628 [18]

Como se puede observar el dispositivo se divide en 3 etapas: Un primer amplificador operacional en modo restador-inversor, un filtro RC de paso bajo y un amplificador operacional en modo no inversor.

El voltaje de salida del A1 sigue la siguiente ecuación:

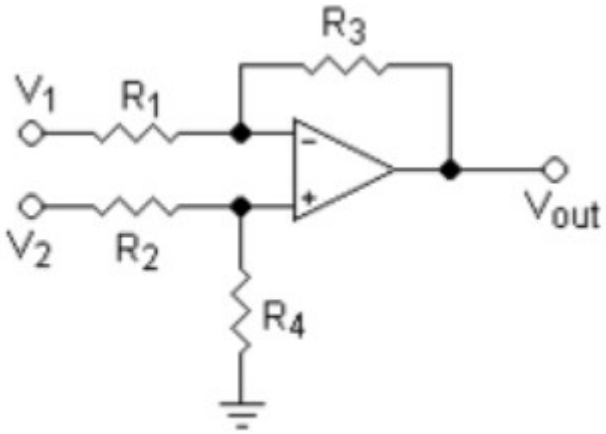


Ilustración 21. OP-AMP en modo no inversor

$$V_{out} = V_2 \frac{(R_3 + R_1)R_4}{(R_4 + R_2)R_1} - V_1 \frac{R_3}{R_1}$$

Comparándola con los valores que ya vienen preestablecidos en el AD628ARZ se obtiene el siguiente valor:

$$V_{out} = V_2 \frac{(10 + 100)10}{(10 + 100)100} - V_1 \frac{10}{100} = (V_2 - V_1)\frac{1}{10}$$

Nótese que para que esto ocurra el pin 3 del dispositivo debe ir conectado a la referencia del arduino. Además, la ganancia de salida de la primera etapa es de 0.1 y se necesita una ganancia de 1, con lo que la ganancia del segundo amplificador deberá ser de 10. Esta ganancia se ajusta variando los valores de R_{ext1} y R_{ext2} a la salida del pin 6. En el datasheet se recomiendan ciertos valores de resistencias para configurar la ganancia que se desee. En este caso, para una ganancia de 10 en el segundo amplificador o una ganancia global unitaria recomienda un valor de $100k\Omega$ y $11k\Omega$ para R_{ext1} y R_{ext2} respectivamente.

El valor de C_{filt} se escoge en función de la frecuencia de corte que se desee para el filtro RC de paso bajo. Se ha escogido un valor de $1kHz$ ya que escogiendo una frecuencia menor puede que se interfiera con el switching de los multiplexores y esto destruiría por completo la veracidad de la señal a medir. El datasheet aconseja un condensador de $15nF$ para esa frecuencia de corte. En el esquema eléctrico queda de la siguiente forma:

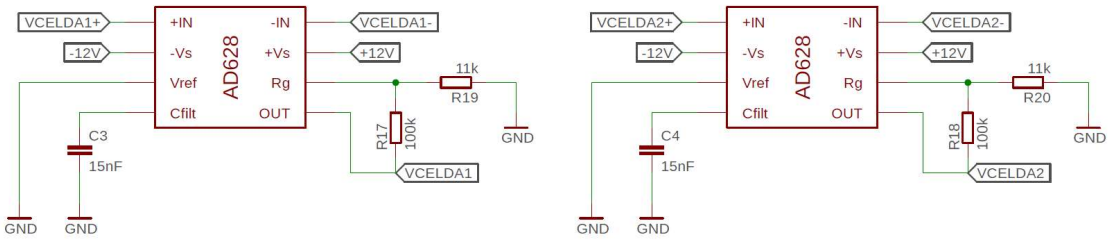


Ilustración 22. AD628 en el esquema eléctrico

Se pueden ver que a las dos etapas idénticas le llegan las señales de salida de cada multiplexor. Los valores de resistencia y condensador son los concluidos anteriormente. Se ha decidido alimentar a $\pm 12V$ para garantizar que todo el rango de tensiones a acondicionar se realiza con la mínima distorsión posible. Esta alimentación se obtiene a partir de los 5V.

La señal ya acondicionada y lista para medir en las entradas analógicas del arduino se han etiquetado como “VCELDA1” y “VCELDA2”. Estas van conectadas a la entrada analógica 6 y 7 respectivamente.

Diseño esquemático: Comunicaciones

I2C

Las comunicaciones I2C entre esclavo-maestro debe ser galvánicamente aislada ya que ambas placas están a distinta referencia. Por lo tanto, se necesita usar un aislador digital de comunicaciones I2C para conectar ambas placas.

El chip elegido es el SI8600AC-B-ISR de Silicon Labs por su calidad-precio y empaquetado SOIC adecuado para soldarlo en el taller.

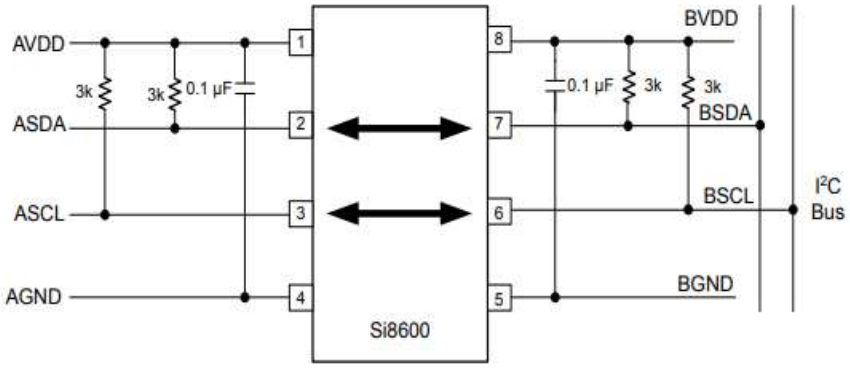


Ilustración 23. Aislador Si8600 [19]

Como se puede ver en el diagrama la comunicación puede ser bidireccional y el propio fabricante recomienda los valores de resistencia Pull-Up y condensador para filtrar el ruido en la alimentación. Los pines de "AVDD"," ASDA"," ASCL" y "AGND" hacen referencia al bus I2C por la parte del maestro y los pines restantes al esclavo.

Puerto serie

Arduino nano ofrece facilidades para comunicarse con otro arduino, ordenador u otro dispositivo gracias a la comunicación serie UART TTL que ofrece el microcontrolador Atmega328.

Para realizar comunicaciones USB con el ordenador tiene el chip FTDI FT232RL. Este será utilizado para programar el arduino desde el ordenador en el entorno de Arduino IDE. Simplemente habrá que conectar el arduino al ordenador mediante el conector mini USB que tiene ya instalado.

Diseño esquemático: Módulo Micro SD

Para guardar los datos de las distintas variables se hará uso de una tarjeta micro SD. Esta irá integrada tanto en cada placa esclava como la maestra y será el usuario quien decida qué tarjeta querrá usar. Por lo tanto, si se quieren ahorrar costes, no hará falta soldar los componentes correspondientes al módulo micro SD en las tarjetas esclavas. Esto se debe a que con tener el módulo en la placa maestra ya es suficiente. Se ha optado por integrarlo en los esclavos por dar una mayor versatilidad y funcionalidad al BMS.

El módulo consiste en el propio conector que contendrá a la tarjeta, un regulador de voltaje y un adaptador de nivel. El regulador de voltaje convierte los 5V del arduino en 3.3V para la micro SD ya que es el voltaje que necesita para su funcionamiento. El adaptador de nivel adapta las comunicaciones entre el arduino y la tarjeta, esto es, de 5V a 3.3V y viceversa.

El regulador de voltaje utilizado es el LP2985-30DBVR. Ofrece una capacidad de corriente de salida de 150mA a 3V, mucho más de lo necesario a un bajo coste y un encapsulado sencillo de soldar.

El adaptador de nivel es el CD74HC4050M96. Tiene la capacidad de adaptar el nivel de 6 hilos diferentes, suficientes para esta aplicación.

El esquemático es el siguiente:

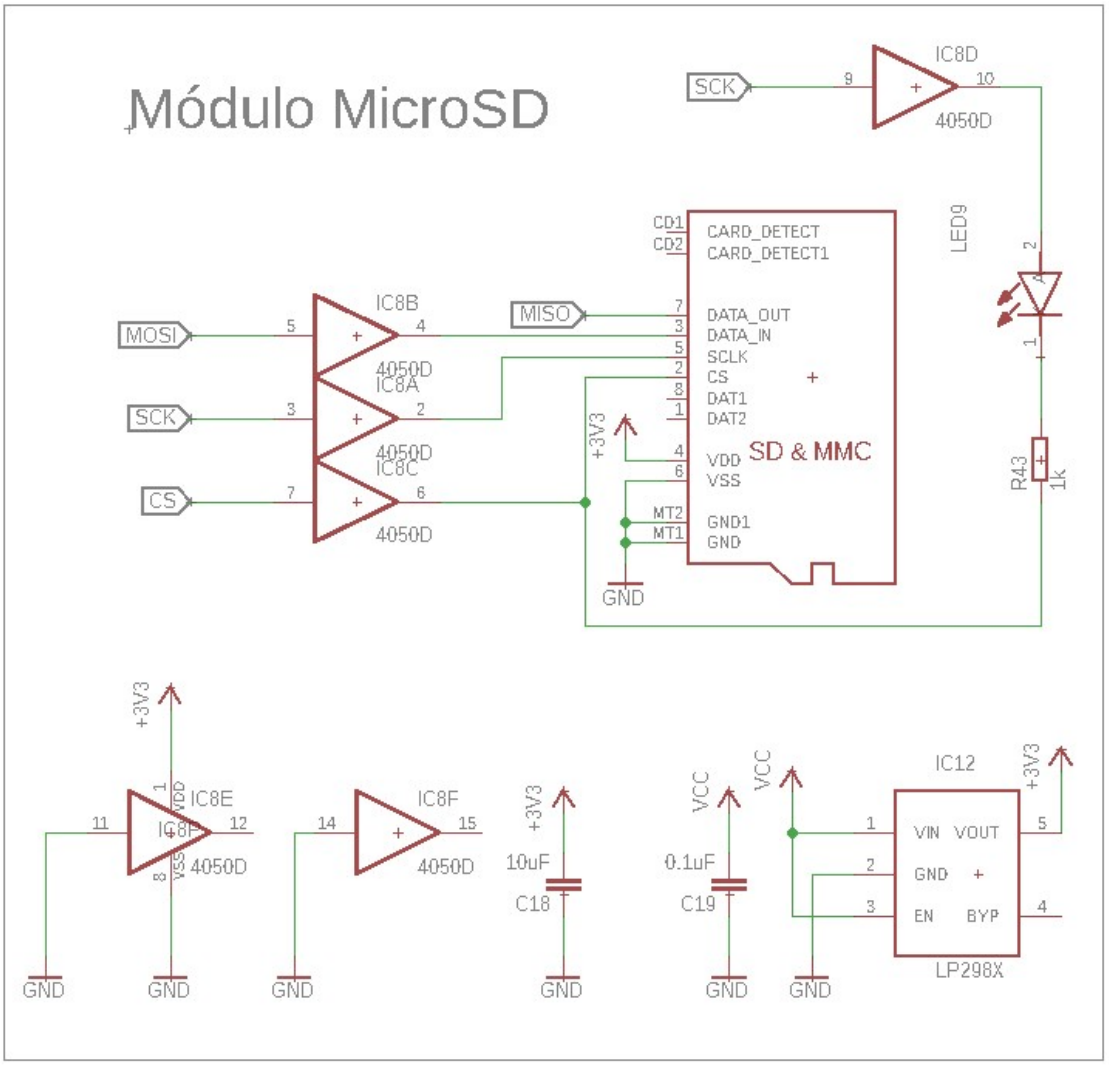


Ilustración 24. Esquema del módulo Micro SD

Diseño esquemático: Pantalla OLED

El módulo OLED a utilizar será una pantalla de 0.96" típica utilizada en proyectos de arduino, dado su bajo coste, consumo y facilidad de uso. Existen principalmente dos tipos de módulos: Con comunicaciones SPI y con I2C. Se ha escogido el de I2C por reducir el número de pines y por simplicidad a la hora de programarlo. La alimentación es a 5V y su conexión y apariencia es la siguiente:

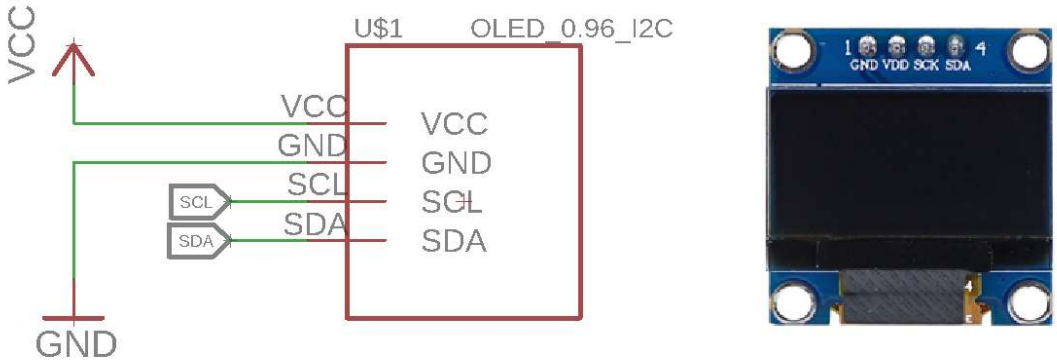


Ilustración 25. Pantalla OLED

Diseño esquemático: Alimentaciones

Surge la necesidad de alimentar los componentes de la placa a 2 voltajes diferentes. Por un lado está el AD628ARZ de la etapa de acondicionamiento que necesita alrededor de $\pm 12V$, mientras que el resto de componentes se alimentan a $5V$.

Estas tensiones deben lograrse a partir de los 12V disponibles en la motocicleta y hay que tener en cuenta que la conversión debe hacerse aislada galvánicamente, ya que cada placa tiene su referencia. Por lo tanto, surgen dos posibilidades:

- Utilizar dos reguladores DC/DC aislados para obtener los $\pm 12V$ y $5V$ a partir de los 12V de entrada.
- Utilizar un regulador DC/DC aislado para obtener $5V$ a partir de los $12V$ de entrada y otro regulador no aislado para obtener los $\pm 12V$ a partir de los $5V$.

Para ambos casos se necesitan dos reguladores, a diferencia de que en el segundo caso solo hace falta que uno de los reguladores sea aislado. Esto ofrece una ventaja económica ya que los reguladores aislados tienden a ser más caros. Electrónicamente hablando apenas hay una ventaja de una frente a la otra, con lo que se opta por la segunda opción.

Una vez conocido el tipo de regulador necesario hace falta estimar el consumo de corriente para elegir de la forma más eficiente los componentes. Para ello se van a enumerar todos los componentes que consuman corriente de cada regulador y se estimará su consumo a partir del datasheet correspondiente:

- La fuente de alimentación de $\pm 12V$ necesita alimentar a:
    - Amplificador diferencial AD628ARZ: $1.6mA$ de consumo por cada amplificador. Como se utilizan 2 el consumo total asciende a $3.2mA$.

Siendo el consumo tan pequeño en los $\pm 12V$ se ha decidido utilizar el conversor MAX680CSA+. Es capaz de suministras hasta $10mA$, con lo que tiene capacidad de sobra para su función. Acepta voltajes entre 1.5 y $6V$, con lo que la alimentación a $5V$ no le supone ningún problema. El voltaje de salida es el doble al voltaje de entrada, con lo que en realidad se obtendrán $\pm 10V$, aunque eso no va a suponer ningún problema a los amplificadores diferenciales que alimenta. Su conexión es la siguiente:

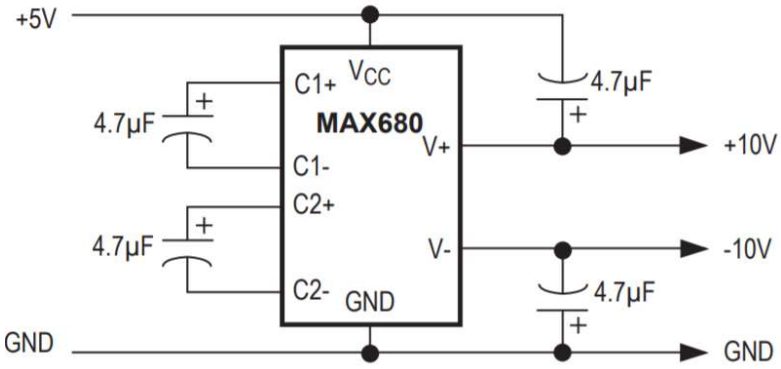


Ilustración 26, MAX680

Es necesario colocar 4 condensadores para poder duplicar el voltaje. El valor recomendado para estos condensadores es de $4.7\mu F$ según el datasheet [26], con lo que se colocarán esos mismos valores.

- La fuente de alimentación de 5V necesita alimentar a:
  - Arduino Nano: Como máximo puede llegar a consumir unos $200mA$, aunque realmente en este proyecto no se vaya a llegar a dichos valores puesto que no se utilizan todos los pines. A este valor se le está sumando el consumo del módulo micro SD.
  - Pin Enable de los multiplexores: $0.15\mu A$, prácticamente despreciable.
  - Aislador digital: $7.6mA$ de consumo máximo.
  - Registro de desplazamiento de la etapa del balanceo: Se va a consumir aproximadamente $7mA$ por cada salida que se active. Suponiendo que se activan todas las salidas a la vez supondría un consumo de $56mA$.
  - Sensores de temperatura Dallas: Aproximadamente unos $5mA$ por cada sensor conectado. Suponiendo que se conectan 4 sensores por placa, algo muy cercano a la realidad, supondría un consumo de $20mA$.
  - Pantalla OLED: 16mA según su datasheet.

- Conversor DC/DC de $5V$ a $\pm 12V$: Para el consumo que tendrá el datasheet estima que la eficiencia del conversor ronda el 90%. Esto supone el siguiente consumo a $5V$:

$$P_{10V} = V * I = 10 * 0.0032 = 0.032\ W$$

$$P_{5V} = \frac{P_{10}}{\eta} = \frac{0.032}{0.9} = 0.0355\ W$$

$$I_{5V} = \frac{P_{5V}}{V} = \frac{0.0355}{5} = 0.007A = 7mA$$

Sumando todos los consumos máximos se llega a un valor de $306.6mA$. En la realidad probablemente tenga un valor menor, pero siempre hay que considerar el peor caso para evitar sobrecargas. Tras una búsqueda entre diferentes fabricantes se opta por el conversor TMH 1205S de Traco Power. Ofrece un tamaño compacto para ser un conversor aislado, además de a un bajo precio y con una capacidad de suministro de hasta $400mA$. Se ha conectado de la siguiente forma:

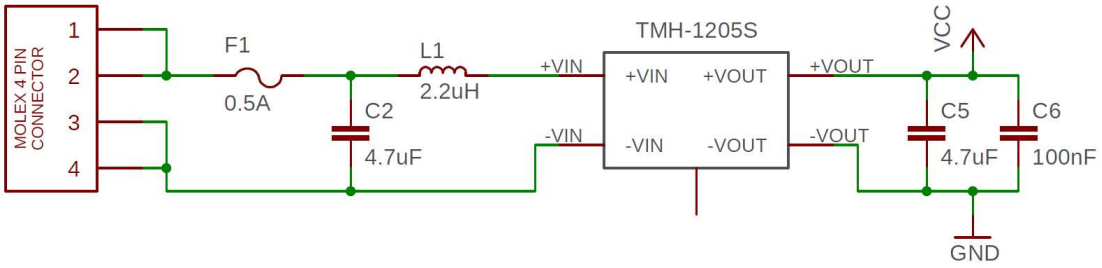


Ilustración 27. Alimentación de la placa esclava

Se han introducido condensadores y una inductancia para reducir el ruido que pueda haber del entorno. También se ha colocado un fusible de 0.5A como medida de protección. La alimentación a la placa se introduce mediante el uso del conector 43650-0400 de Molex.

Diseño del PCB

Para el diseño del circuito impreso se ha utilizado el mismo programa que para el esquema eléctrico, el Autodesk Eagle. Una vez terminado el esquema hay que importar todos los componentes a un nuevo archivo que contendrá el PCB. Cuando se importa el esquema completo aparece lo siguiente:

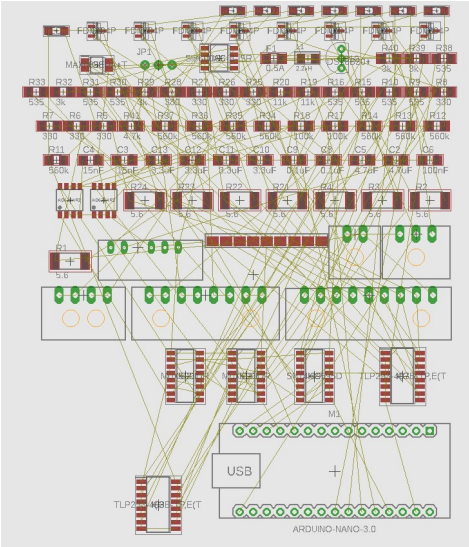


Ilustración 28. Diseño del PCB esclavo

Las líneas amarillas simbolizan todas las conexiones que hay entre dichos componentes. El siguiente paso es colocar todos los componentes en el sitio que más les convenga. Para ello se siguen los siguientes criterios:

- Todos los componentes se colocarán en la misma cara para facilitar el ensamblaje de los mismos. Esto se debe a que todos los componentes, salvo los Through Hole, serán soldados a la vez en un horno. Este proceso se explica en detalle en el apartado "Ensamblaje".
- El resto de componentes pertenecientes a la misma referencia se van a colocar lo más cerca posible para reducir el tamaño del PCB y ordenados de tal forma que faciliten las conexiones. Para tal fin se agrupan los componentes en función del papel que desempeñan en la placa: Conectores, alimentación, comunicaciones, balanceo, medida de celdas, sensores de temperatura y Arduino.
- No se necesita un ancho de pista excesivo ya que la mayor corriente no será de más de 400mA. El criterio a seguir será colocar un grosor de pista que permita circular la corriente sin que apenas se caliente la placa. Contrastando en diferentes bibliografías y experiencia personal se concluye que un ancho de pista de 0.5mm es suficiente para corrientes de hasta 2A. Por lo tanto, si se cumple esta anchura quedará descartado el sobrecalentamiento de la placa.
- Las resistencias que disipan la energía del balanceo se colocan ordenados en una fila por si se diera la necesidad de instalar un disipador de calor. De esta forma colocando un único disipador rectangular se puede refrigerar todas las resistencias. Adicionalmente se colocan dos agujeros en los extremos izquierdo y derecho para

facilitar el amarre del disipador. En caso de que fuera necesario se podría aprovechar los agujeros para colocar un ventilador.

- Se colocan al menos tres agujeros más para facilitar el amarre entre las placas o a la propia motocicleta.
- La organización de MotoStudent indica en el reglamento la siguiente norma a cumplir: *"En el caso de que ciertos componentes pertenecientes al HVS y GLVS se instalen en una misma placa base, se colocarán en zonas claramente diferenciadas y marcadas a tal efecto sobre la placa. La separación entre ambas será de al menos, 6,4 mm sobre superficie, 3,2 mm a través del aire y de 2 mm si están bajo recubrimiento (estas distancias pueden no respetarse para el caso de optoacopladores cuya tensión nominal sea igual o mayor que la tensión del HVS) ".* Puesto que no se va a utilizar ningún tipo de recubrimiento habrá que respetar las distancias de 6.4 y 3.2mm. Se prestará especial atención a la hora de colocar los dos conectores que introducen las comunicaciones I2C y alimentación y el aislador digital de las comunicaciones, ya que son los únicos componentes que van referenciados a otra masa. Para aumentar la seguridad ante arcos eléctricos se va a realizar un corte en la placa en las zonas que separan elementos de distintas referencias. De esta forma se aumenta la distancia de separación de componentes sobre superficie sin haberlos alejados entre ellos.

Teniendo en cuenta lo anteriormente comentado el resultado es el siguiente:

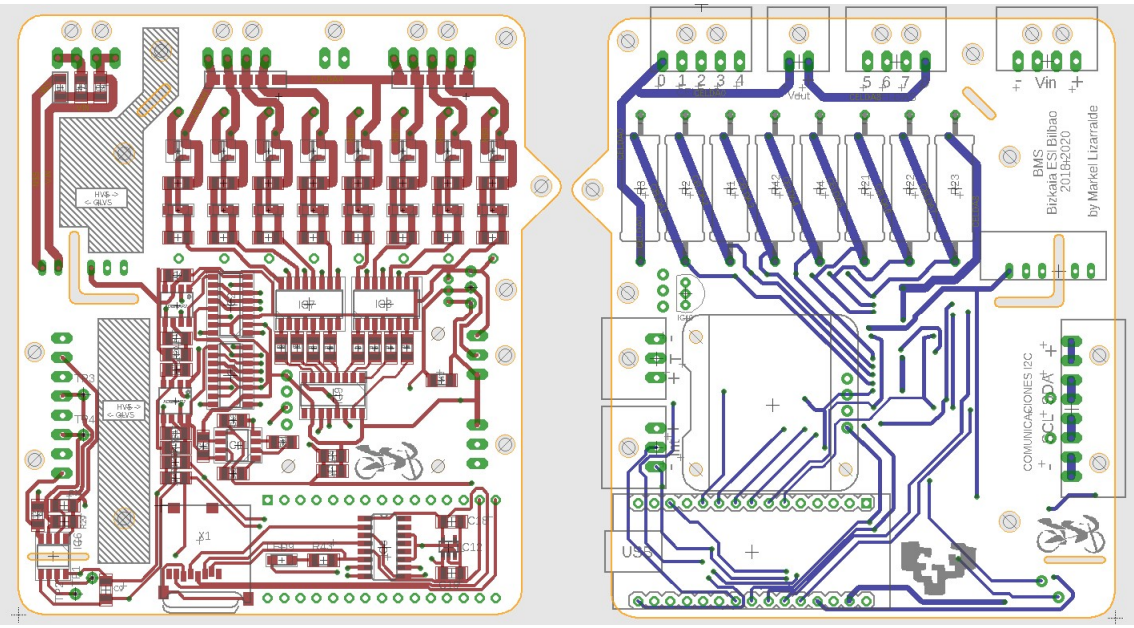


Ilustración 29. PCB esclavo

Se ha logrado colocar un gran número de componentes, elementos que consumen potencia y distintas referencias que obligan a aumentar la separación de los componentes en un tamaño reducido. El PCB se resume en unas dimensiones finales de $85x94mm$.

A continuación se enumera cada uno de los bloques funcionales de la placa y se muestra en qué parte están colocados:

1. Conexión de las celdas a la placa.
2. Alimentación de la placa esclava.
3. Etapa de balanceo.
4. Comunicaciones I2C.
5. Medición de voltaje de las celdas.
6. Etapa de multiplexación del balanceo.
7. Alimentación de ±12V.
8. Almacenamiento en micro SD.
9. Sensores Dallas.
10. Pantalla OLED.
11. Sensor de intensidad.
12. Arduino Nano.

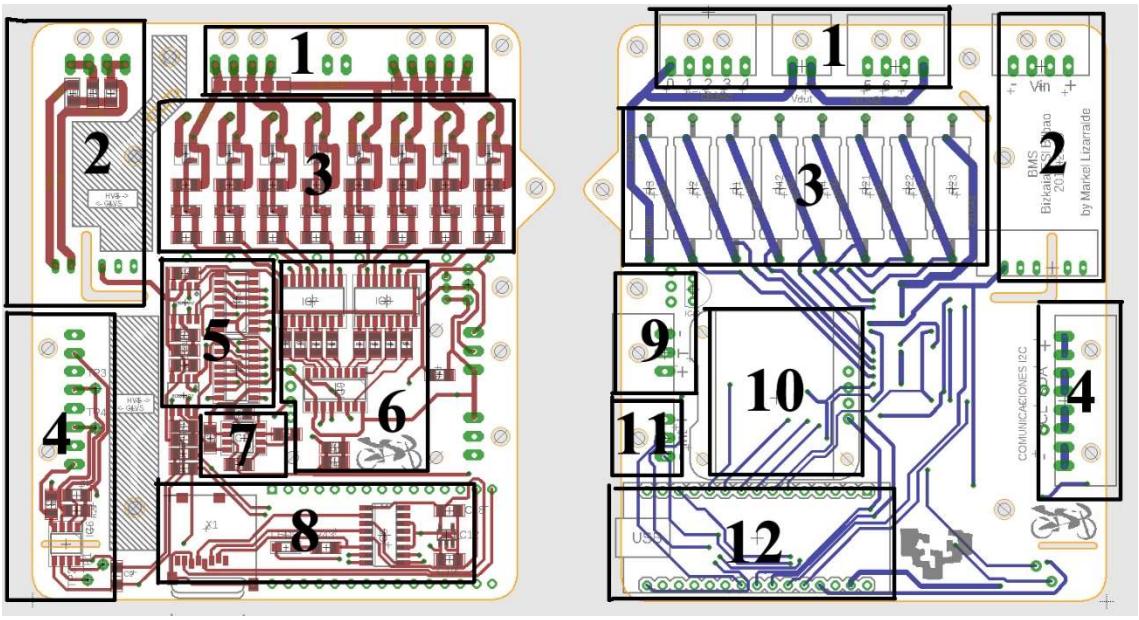


Ilustración 30. Bloques funcionales del esclavo

DISEÑO DE PLACA MAESTRA

Diseño esquemático: Arduino Nano

El uso de pines digitales y analógicos es el siguiente:

- Entradas digitales:
  - D6 hasta D8: Supervisión de apertura y cierre de hasta 3 contactores.
  - A0: Supervisión del IMD. En realidad es una entrada analógica, pero se puede usar como entrada digital.
- Salidas digitales:
  - A1 hasta A3: Apertura y cierre de contactores. Las entradas analógicas también se pueden utilizar como salidas digitales.
- Entradas analógicas:
  - A6 y A7: Medición de termistores.
- Comunicaciones:
  - D0 y D1: Comunicaciones serie. No hay nada conectado ya que va internamente en el arduino al conector micro USB.
  - D3 hasta D5: Comunicaciones bluetooth.
  - D2, D10, D11, D12, D13: Comunicaciones SPI para la micro SD y el CAN.
  - A4 y A5: Comunicaciones I2C.

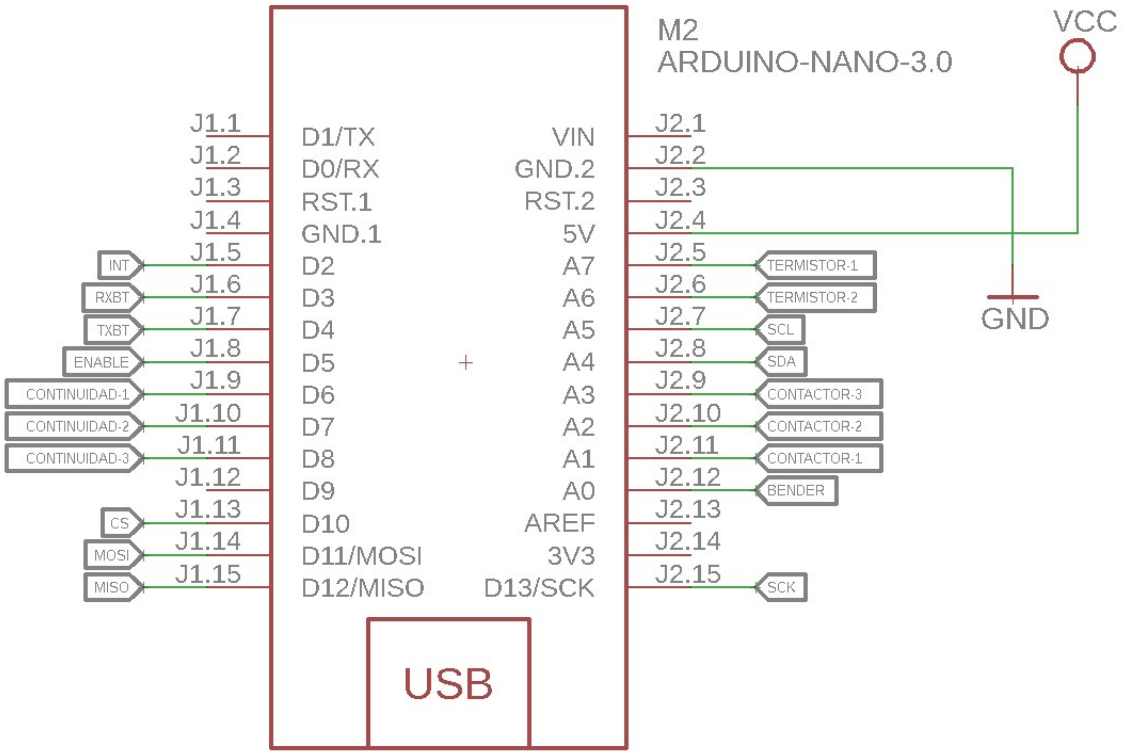


Ilustración 31. Arduino Nano en el maestro

Al igual que con la placa esclava, cada uno de los pines que se ha utilizado o dejado de utilizar se han elegido con el fin de facilitar el ruteado a la hora de diseñar el PCB.

Diseño esquemático: Control de contactores

El control de contactores se puede realizar de dos formas: Con o sin circuito de precarga. Los circuitos de precarga se utilizan como elemento de seguridad en contactores que alimentan cargas capacitivas a un alto voltaje. La carga capacitiva requeriría un pico de intensidad tan alto al cerrar el contactor que podría dañar el contactor.

A fin de evitar este fenómeno no deseado, el circuito de precarga adecúa el voltaje de ambos terminales del contactor para que sea por debajo de un rango seguro. Existen diferentes topologías para precargar un contactor, pero no hace falta detallarlo en este proyecto, simplemente saber que se debe añadir la posibilidad de utilizarlos.

Se va a explicar primero el circuito del control de contactor sin precarga y posteriormente con precarga para finalmente mostrar cómo se ha implementado.

Control de contactor sin precarga

"CONTACTOR-1" es la salida digital del arduino que controla el contactor. Como los contactores pueden llegar a consumir más intensidad que la que el arduino puede dar por pin y el voltaje necesario es de 12V en lugar de 5V se debe diseñar un esquema que cumpla las necesidades. Ese esquema es el siguiente:

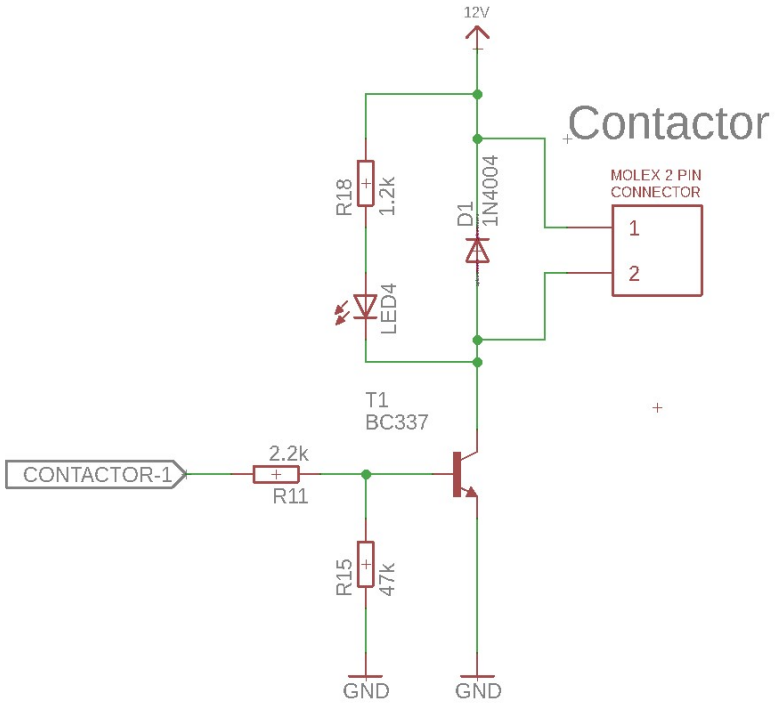


Ilustración 32. Circuito del contactor [20]

El control del contactor se realiza a través del transistor NPN "T1". Este cierra el circuito alimentando a 12V directamente el contactor ya que pone a masa el terminal del contactor que no está a los 12V. De esta forma se soluciona el problema de alimentar el contactor a una tensión diferente a los 5V del arduino. Los 12V se suministran directamente de la alimentación de la placa maestra.

El valor de R11 se ha calculado para garantizar que el transistor deja pasar la suficiente corriente como para alimentar el relé. Se ha supuesto que el contactor que se va a conectar consumirá como mucho 100mA a 12V. Por lo tanto, el valor de R11 será:

$$V_{"CONTACTOR-"} = V_{R11} + V_{BE} \ (1)$$

$$\left.\begin{matrix} V_{R11} = R11 x I_{R11} \\ I_{R11} = I_B = \dfrac{I_C}{h_{fe}} \end{matrix}\right\} \rightarrow V_{R11} = R11 x \frac{I_C}{h_{fe}} \ (2)$$

$$V_{BE} = 0.6\,V \ (3)$$

$$V_{"CONTACTOR-1"} = 5V\ (4)$$

Sustituyendo (2), (3) y (4) en (1) y despejando el valor de R11:

$$R11 = \frac{(5 - 0.6)xh_{fe}}{I_C}$$

Suponiendo un h_{fe} pesimista de valor 50 y los 100mA de I_C:

$$R11 = \frac{(5 - 0.6)x50}{0.1} = 2200\Omega$$

R15 se ha colocado entre la base del transistor y la masa para evitar que el transistor se active de forma errática en caso de que la salida digital "CONTACTOR-1" se encuentra en un estado indefinido. Esto puede ocurrir cuando el BMS está en fase de inicialización y se debe evitar. Con esta resistencia la tensión de la base del emisor quedará siempre bien definida. Su valor no debe ser exacto, pero es conveniente que tenga un valor alto para que consuma poca intensidad y no afecte a los cálculos realizados anteriormente. Se ha escogido un valor de 47kΩ.

El diodo "D1" evita la sobretensión producida al desconectar el contactor. Esto se debe a las inductancias creadas por el bobinado del mismo. El diodo se coloca en paralelo e inversamente polarizado para absorber esos picos de tensión. Se ha optado por poner el diodo 1N4004 por ser uno de los más comunes del mercado.

El "LED4" es un led indicador que se ilumina cuando el contactor está cerrado. Para calcular el valor de la resistencia "R18" se han realizado los siguientes cálculos:

$$12V = V_{R18} + V_{LED4}$$

$$12V = R18xI_{R18} + V_{LED4}$$

$$R18 = \frac{12 - 1.3}{0.009} = 1188.8\ \Omega\ \approx 1.2k\Omega$$

Control de contactor con precarga

El esquema de control del contactor sigue siendo el mismo que en el caso de no tener el circuito de precarga. Sin embargo, se introduce el circuito de precarga entre la salida digital del arduino y la base del transistor:

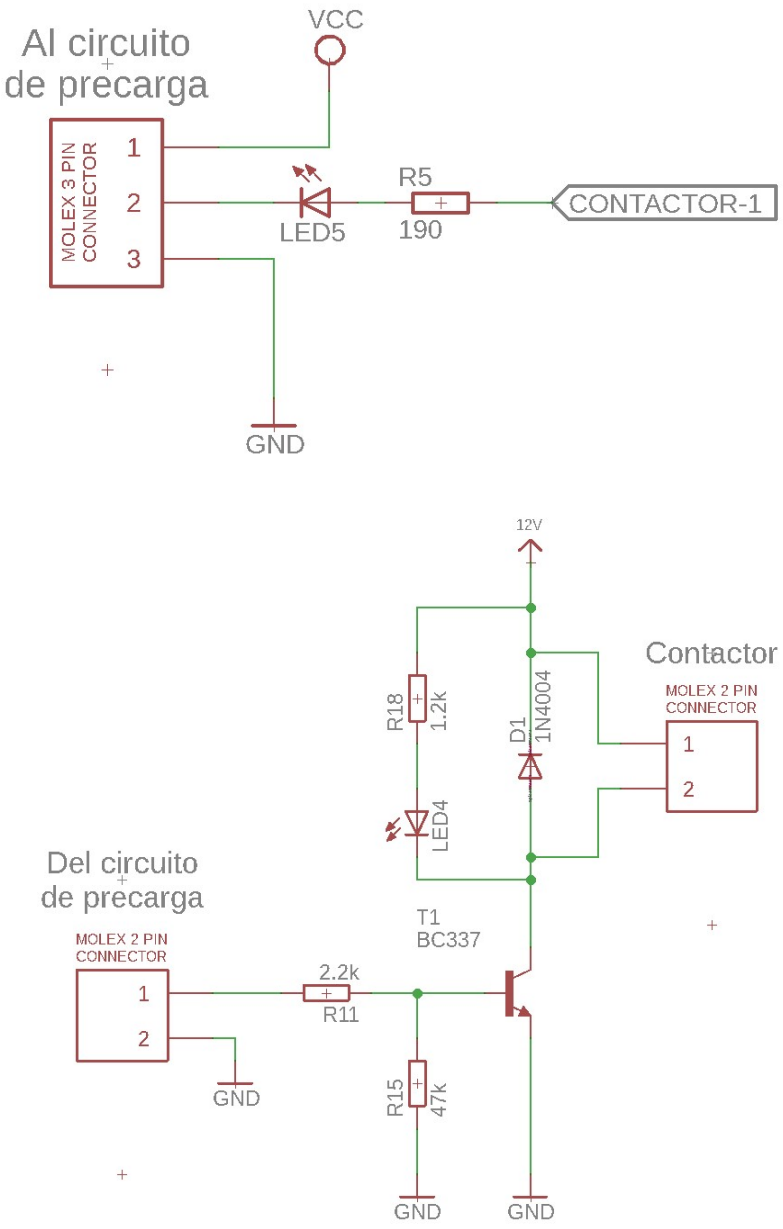


Ilustración 33. Esquema de control de contactor

El conector de 3 pines de Molex sale de la placa maestra y va al circuito de precarga. Dos cables son de alimentación y el tercero es la señal de activación del contactor. El circuito de precarga procesa esa señal, realiza la precarga del contactor y cuando está listo envía de vuelta la señal de activación definitiva al conector de 2 pines de Molex que está conectado a la base del transistor. En resumen, el transistor no se va a cerrar para cerrar el contactor hasta que el circuito de precarga diga que pueda hacerlo.

Control de contactor implementado

Como en la realidad no se sabe cuántos contactores se van a utilizar y si todos esos contactores van a necesitar un circuito de precarga (puede que algunos de esos contactores no manejen altos voltajes), hay que añadir la posibilidad de utilizar circuito de precarga o no.

Para ello se ha realizado una combinación de los dos métodos anteriores:

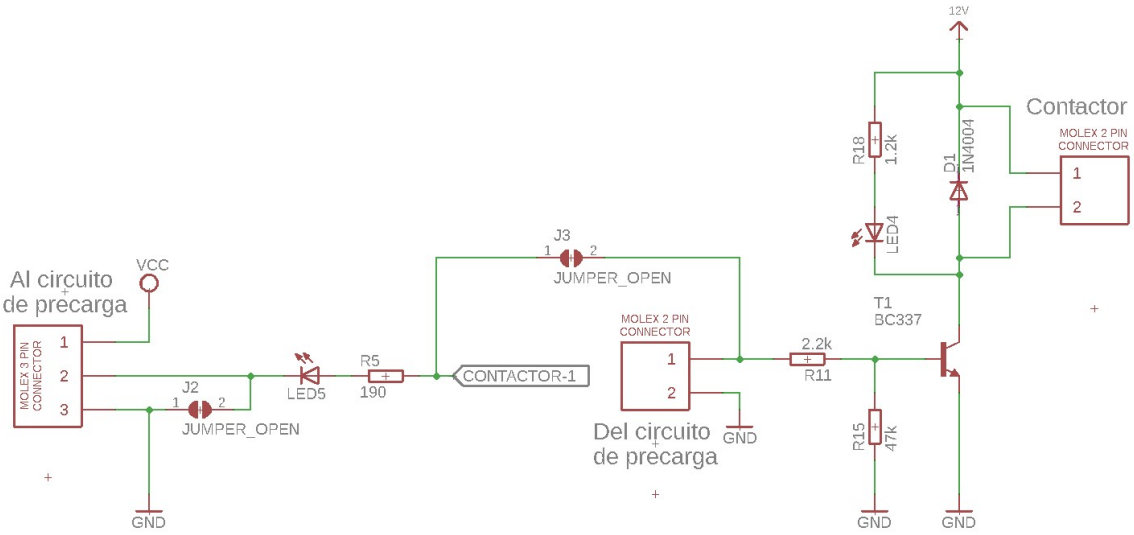


Ilustración 34. Control de contactor completo

Se han añadido dos jumpers "J2" y "J3" para decidir el uso o no de la precarga. Es el usuario quien decidirá cómo utilizarlo:

- Si se cierran los dos jumpers, el circuito de precarga sería puenteado y no haría falta utilizarlo.
- Si se dejan abiertos los dos jumpers se usaría con circuito de precarga.

Como se va a incorporar la posibilidad de controlar 3 contactores distintos, este circuito estará triplicado en la placa maestra.

Diseño esquemático: Supervisión de contactores

La supervisión de contactores monitoriza si el contactor está abierto o cerrado. Es una herramienta útil para saber el estado de contactores que no controlan el BMS, como un interruptor de emergencia, por ejemplo.

Por normativa de la competición, si se pulsa el interruptor manual de emergencia de la motocicleta el BMS no podrá volver a reenganchar el contactor principal hasta que se

reinicie todo el sistema. Por lo tanto, se ve necesario supervisar al menos el interruptor de emergencia. Como función adicional se van a colocar otros dos circuitos idénticos a fin de poder supervisar otros contactores, para futuras ampliaciones.

El circuito es el siguiente:

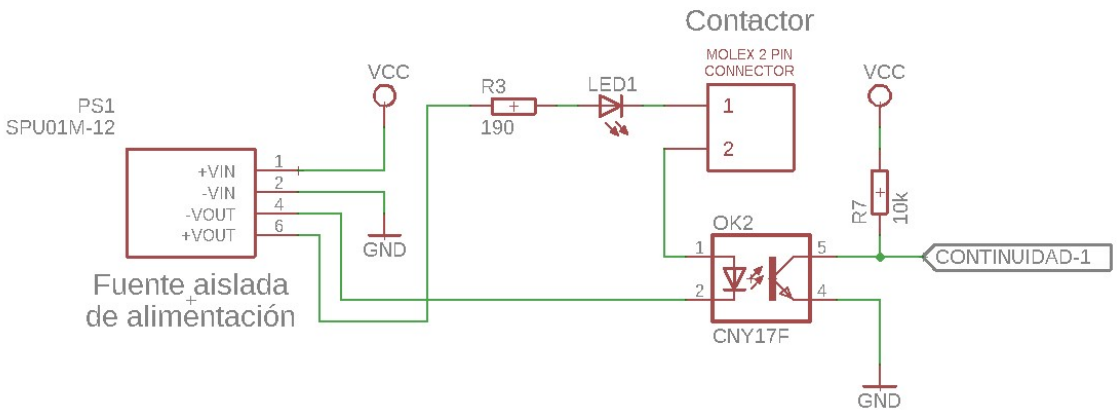


Ilustración 35. Supervisión de contactor

El bloque "PS1" es una fuente de alimentación aislada que alimenta a 12V aislados galvánicamente respecto a los 12V de alimentación de la placa maestra. Este voltaje debe ser aislado ya que los contactores que se van a supervisar no tienen por qué estar referenciados a la misma masa.

Los terminales del contactor se conectan al conector de dos pines mostrado. En el caso de que el contactor esté cerrado el optoacoplador entrará en conducción y cambiará el estado del pin "CONTINUIDAD-1" que va directamente como entrada digital al arduino. De esta forma, cuando el contactor está cerrado el opto conduce y al arduino le llega una señal LOW y viceversa.

El valor de R3 se ha calculado de la siguiente forma:

$$12V = V_{R3} + V_{LED1} + V_{opto}$$

$$12V = R3xI_{R3} + V_{LED1} + V_{opto}$$

$$R3 = \frac{12 - 1.39 - 1.4}{0.01} = 921\ \Omega$$

El voltaje de 1.39V e intensidad de conducción de 10mA se ha obtenido del datasheet del optoacoplador CNY17F [21].

La resistencia R10 no es más que una resistencia pull-up y se ha puesto el típico valor de 10k Ω.

Diseño esquemático: Termistores

Se va a colocar la posibilidad de conectar dos termistores, tanto NTC como PTC. Las conexiones son las mismas, lo único que cambia es la calibración de los mismos mediante software.

Los termistores varían su resistencia en función de la temperatura. Si se quiere medir con el arduino nano no hay más que convertir esa variación de resistencia en voltaje. Para ello se alimenta el termistor con los 5V del arduino y se coloca a una resistencia fija en serie. El voltaje a medir en el arduino será el voltaje del termistor.

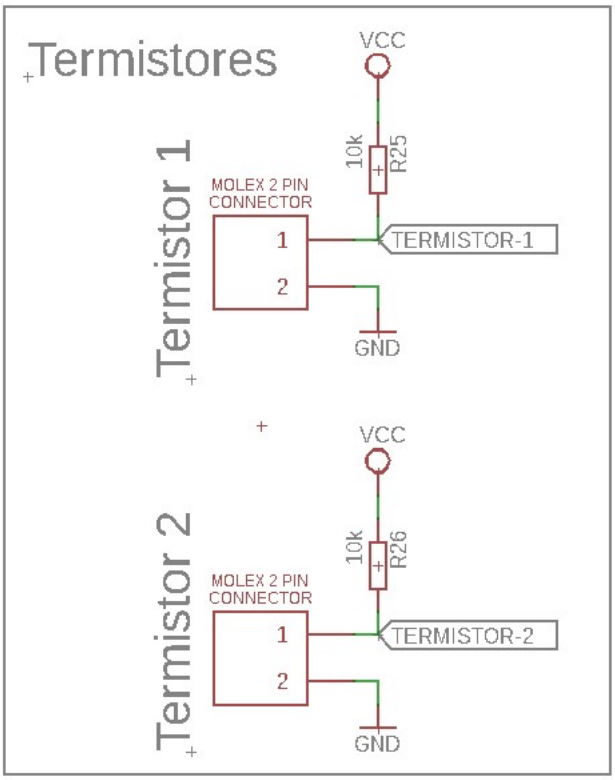


Ilustración 36. Esquema de conexión de termistores

La conversión de voltaje medido a temperatura se detalla en el apartado "Medición de termistores".

Diseño esquemático: Supervisión del IMD

El IMD tiene como salida digital una señal de unos 10V aproximadamente. Esta indica el estado de aislamiento de las distintas referencias de la motocicleta. Si la resistencia entre la referencia de bajo voltaje y alto voltaje es mayor al valor estipulado el IMD envía una señal

continua de 10V. Si la resistencia tiene un valor menor al permitido, lo que indicaría que hay un problema de aislamiento, el BMS debe detectarlo y abrir el contactor correspondiente.

Como el IMD y la placa maestra están referenciados a la misma masa, lo único que habrá que hacer para leer esa señal de 10V es adaptar su voltaje para poder leerlo en el arduino. Como arduino admite un voltaje de hasta 5V habrá que dividir ese voltaje por la mitad utilizando un puente resistivo:

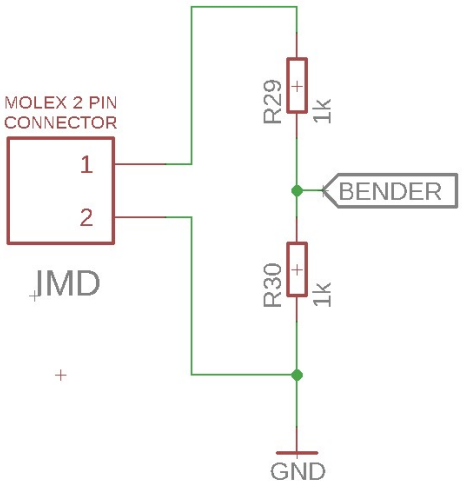


Ilustración 37. Esquema de conexión del IMD

La señal "BENDER" es la entrada digital que va directa al arduino y valdrá 0V (LOW) o 5V (HIGH) por lo anteriormente comentado. Los valores de las resistencias de 1k Ω garantizan un bajo consumo de corriente.

Diseño esquemático: Comunicaciones

I2C

Las comunicaciones I2C son idénticas a la placa esclava, a diferencia de que en la placa maestra no hace falta añadir el aislador de comunicaciones SI8600AC-B-ISR. Simplemente se sacan los cables de comunicación SDA y SCL y los de alimentación para llevarlos a cada una de los esclavos. El aislamiento se realiza en cada esclavo.

Puerto serie

Como se utiliza el mismo microcontrolador en el esclavo y en el maestro las comunicaciones de puerto serie entre placa y ordenador son las mismas, con lo que no hace falta volver a explicarlo. Para más detalles véase el mismo apartado en el diseño de la placa esclava.

CAN

Las comunicaciones CAN entre el arduino y otros dispositivos se realizan mediante los controladores MCP2515 y MCP2551 [22]. Ambos chips realizan la conversión del protocolo CAN a SPI para que el arduino pueda comunicarse. El esquema de conexiones es el siguiente:

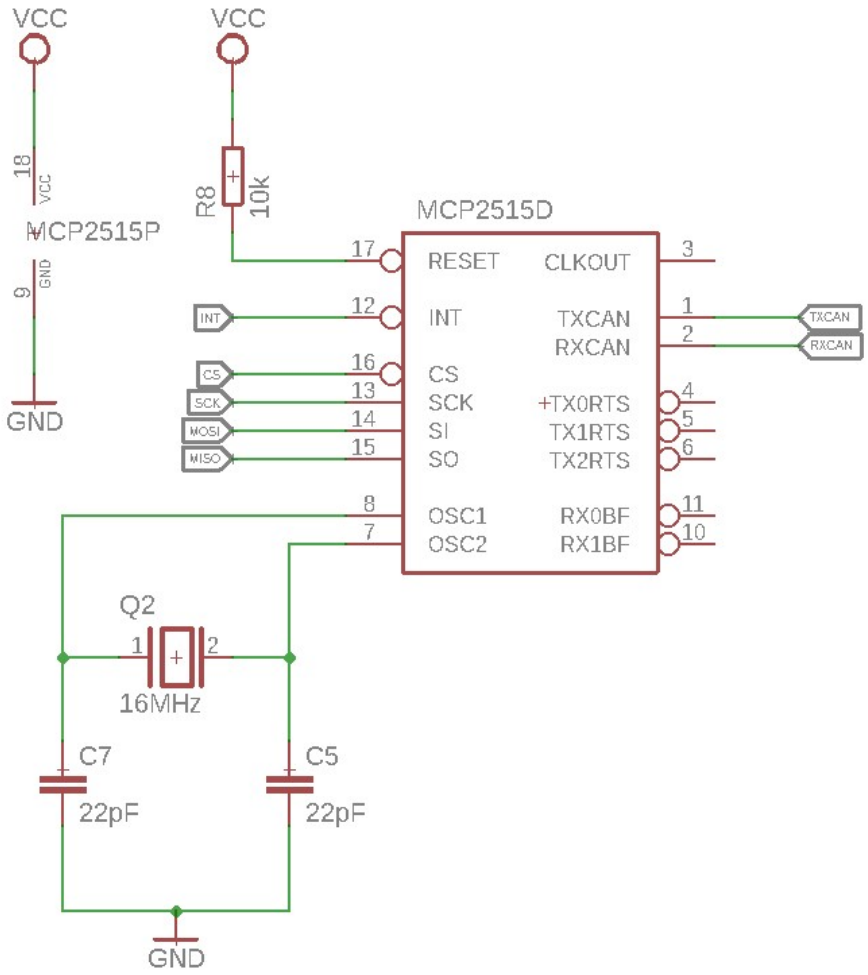


Ilustración 38. Esquema del MCP2515D

Para el correcto funcionamiento del protocolo CAN es necesario añadir un oscilador de 16MHz junto con dos condensadores de 22pF. Esta parte se encarga de generar la señal de reloj necesaria en el módulo.

Las conexiones CS, SCK, MOSI y MISO son las necesarias para las comunicaciones SPI con el arduino. El MCP2515D traduce las comunicaciones SPI a dos únicas señales, TXCAN y RXCAN y las envía directamente al MCP2551:

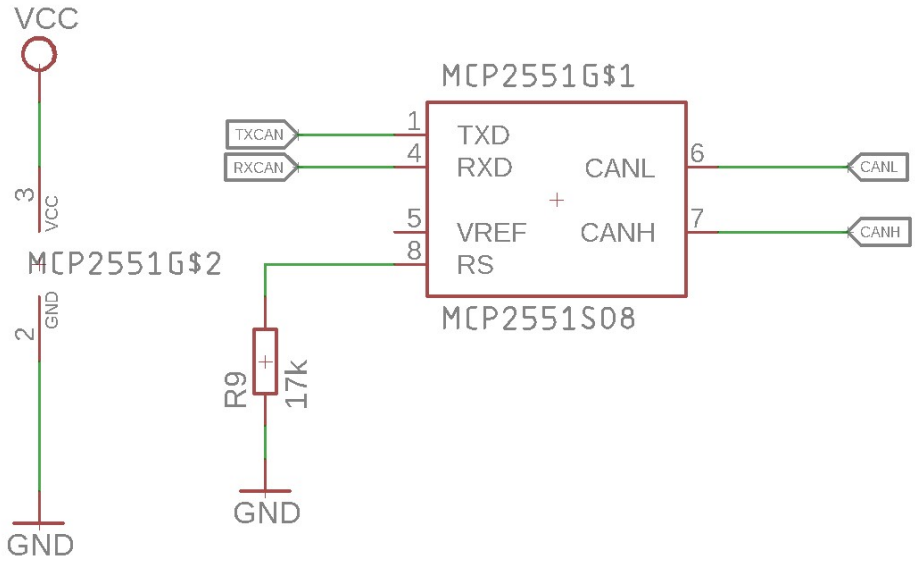


Ilustración 39. Esquema del MCP2551

El MCP2551 es el transmisor encargado de cumplir los requisitos de la capa física del protocolo CAN. CANH y CANL son las señales ya en el protocolo CAN que irán de un dispositivo a otro.

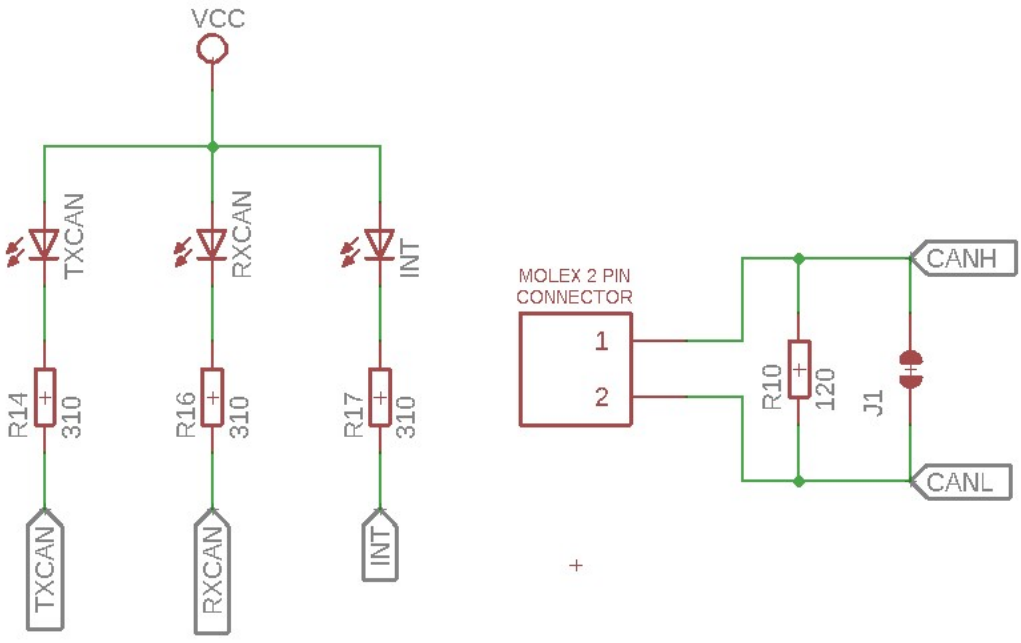


Ilustración 40. Resto de esquema del CAN

La resistencia R10 de 120Ω es la resistencia terminal de las comunicaciones. Sirve para facilitar la tarea de averiguar averías y se coloca al lado de CANH y CANL [23].

Adicionalmente se han colocado 3 leds como indicadores de lo que está ocurriendo en las comunicaciones.

Bluetooth

Por simplicidad a la hora de realizar las soldaduras y conocimientos por haber realizado comunicaciones bluetooth en otros proyectos se va a utilizar un módulo HC-05. Tiene la siguiente apariencia:

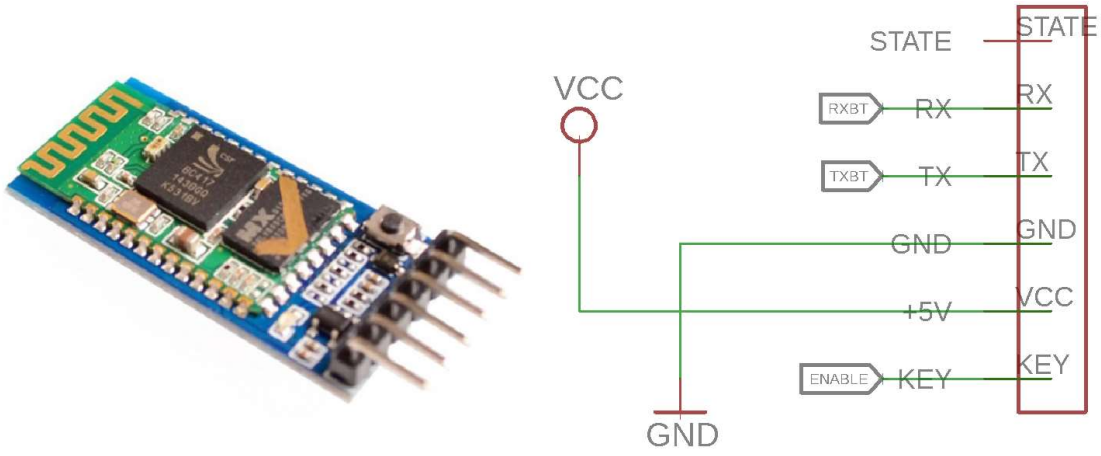


Ilustración 41. Módulo bluetooth HC-05

El módulo bluetooth se encarga de realizar el enlace entre el bluetooth y el protocolo UART TTL del arduino. Se alimenta a 5V y el pin "ENABLE" se conecta con el arduino para posibilitar la configuración del bluetooth mediante comandos AT. El pin "STATE" no es necesario conectarlo.

Diseño esquemático: Módulo Micro SD

El módulo micro SD es exactamente idéntico al de la placa esclava, con lo que no hace falta detallarlo en el presente apartado.

Diseño esquemático: Pantalla OLED

La pantalla OLED es exactamente idéntica al de la placa esclava, con lo que no hace falta detallarlo en el presente apartado.

Diseño esquemático: Alimentaciones

La alimentación de la placa maestra proviene de 12V de corriente continua de la motocicleta. Como el maestro estará referenciado a la misma masa que estos 12V la

conversión no debe ser aislada. Esos 12V deben adaptarse a 5V y el consumo que habrá es el siguiente:

- Arduino Nano: Como mucho 200mA según el fabricante para todas sus funcionalidades. En este consumo se ha incluido el módulo de la micro SD y el del CAN.
- Pantalla OLED: 16mA según su datasheet.
- Bluetooth: 50mA según su datasheet.

En total suman 266mA de máximo consumo a 5V. Se va a utilizar la fuente TSR1-2450, que tiene una capacidad de hasta 1A. Suponiendo una eficiencia de 92% supondría un consumo a 12V de:

$$0.92 = \frac{P_{5V}}{P_{12V}} = \frac{5VxI_{5V}}{12VxI_{12V}} \rightarrow I_{12V} = \frac{5x266}{0.92x12} = 120.47mA$$

Cada uno de los circuitos de supervisión de contactores tiene una fuente de alimentación aislada para suplir los $10mA$ que se estima que consumirán. Se ha escogido SPU01M-12 con una capacidad de 84mA. Es más que de sobra, pero se ha escogido por ser la opción más económica que se ha encontrado en el momento. Según su datasheet tiene una eficiencia aproximada de 84%, lo que supondría en total un consumo de 11.90mA por circuito de supervisión.

Cada uno de los contactores puede llegar a consumir hasta 100mA y, como pueden conectarse 3, el consumo total puede llegar a 300mA. Si se le suman los 3 circuitos de supervisión y la parte que alimenta a la fuente no aislada de 5V el consumo total llega a:

$$I_{total} = I_{fuente\ no\ aislada} + I_{supervisión} + I_{contactores} = 120.47 + 3x11.9 + 300$$
$$= 456.17mA$$

Este consumo habrá que tenerlo en cuenta a la hora de escoger la fuente de alimentación de 12V que alimentará a todo el BMS.

Diseño del PCB

Para el diseño de la placa maestra se han seguido los mismos criterios de ancho de pista y separación de referencias de la placa esclava. Las dimensiones finales son 84x95mm, idénticas a la placa esclava. Se ha hecho el esfuerzo de encajar todos los componentes en el mismo pequeño de placa para aumentar la compactación de toda la electrónica. El resultado es el siguiente:

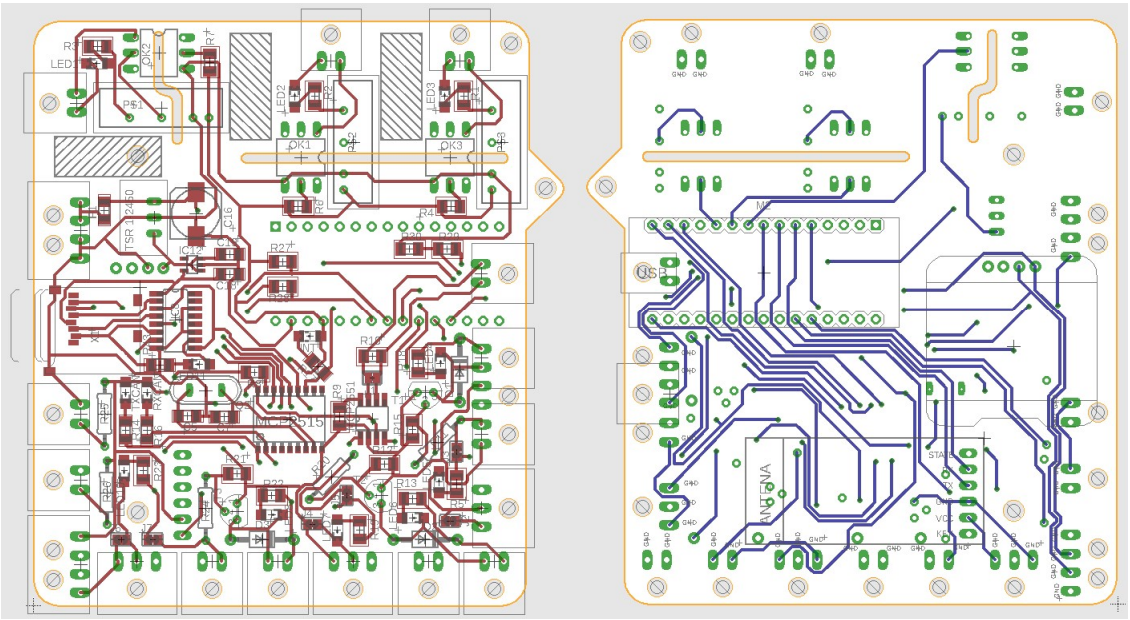


Ilustración 42. Diseño de placa maestra

A continuación se enumera cada uno de los bloques funcionales de la placa y se muestra en qué parte están colocados:

1. Supervisión de contactores.
2. Alimentación de la placa maestra.
3. Almacenamiento en micro SD.
4. Termistores.
5. Comunicaciones CAN.
6. Comunicaciones I2C.
7. Control de contactores.
8. Arduino Nano.
9. Pantalla OLED.
10. Supervisión del IMD.
11. Módulo Bluetooth.

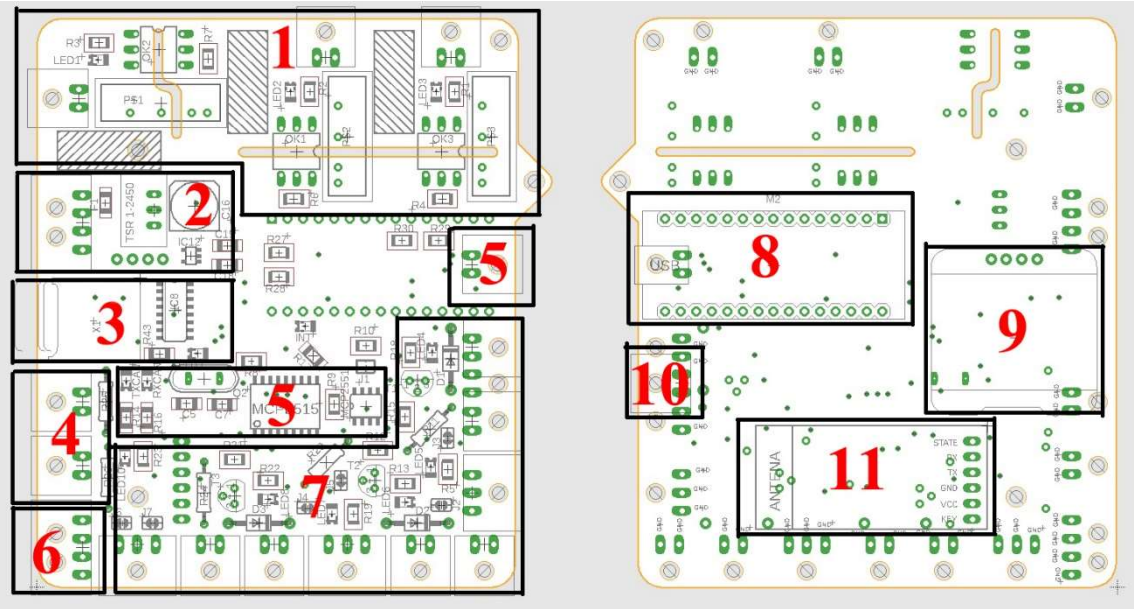


Ilustración 43. Bloques funcionales de placa maestra

SOFTWARE DEL BMS

Ambos softwares, tanto el del esclavo como el del maestro, se programarán utilizando el entorno Arduino IDE. Como ambas placas deben realizar distintas funciones y cada una de ellas con sus propias restricciones temporales se hará un estudio previo de cada una de las tareas por separado con el fin de asegurar que el software final funcionará sin problemas.

Se va a programar el arduino como si se tratara de un sistema operativo de tiempo real o RTOS (Real Time Operative System). Un RTOS es cualquier actividad de procesamiento de la información que debe responder dentro de un intervalo de tiempo especificable, finito y breve. Para realizar la actividad organiza las tareas de forma concurrente, esto es, un planificador decide qué tarea se ejecuta en cada instante en base a ciertos atributos, en vez de ejecutar el código de forma secuencial. Se utilizan los sistemas RTOS en aplicaciones críticas en las que un fallo del sistema puede llevar a un accidente, catástrofe o cualquier consecuencia grave no deseable.

Aunque el procesador del arduino no posea los mecanismos Hardware necesarios para funcionar como un RTOS existe una librería que mediante software ofrece las herramientas necesarias para conseguirlo. Para que el planificador de tareas gestione el procesador y las comunicaciones deben especificarse ciertos atributos a cada una de las tareas. Son los siguientes:

- Tipo de tarea: Periódica si se ejecuta cíclicamente o esporádica si debe cumplirse alguna condición para que la tarea se ejecute.
- Prioridad de la tarea: Es un atributo numérico que se le da a cada tarea. La prioridad de cada tarea ayuda al planificador de tareas a decidir que se ejecuta en cada momento.
- Tiempo de ejecución (C): Tiempo que necesita la tarea para ejecutarse completamente. Se habla del tiempo máximo de ejecución cuando se da el caso en el que la tarea necesita el tiempo máximo posible para ejecutarse. Una tarea puede tener diferentes tiempos de ejecución a causa de bucles no acotados, condiciones que llevan a una sección de código, etc.
- Periodo de activación (T): Intervalo de tiempo que debe pasar entre dos activaciones consecutivas. Como es de esperar el periodo de activación deberá ser igual o mayor al tiempo de ejecución de la tarea.
- Plazo de respuesta o "*deadline*" (D): Tiempo máximo que debe transcurrir para que una tarea se ejecute completamente desde que se ha activado. Este tiempo debe ser igual o mayor que el tiempo de ejecución.

Como ya se ha especificado se debe conocer el tiempo máximo de ejecución. Existen dos formas de hacerlo: Experimentalmente midiendo el propio tiempo mediante el uso de algún algoritmo en el código o descomponiendo el código de la propia tarea línea a línea y calcularlo de forma precisa. El código a ejecutar para las tareas del BMS siempre es el mismo ya que son tareas estáticas y no dinámicas. Solo en algunas de ellas habrá bucles condicionales que varíen el tiempo de ejecución de la tarea. En esos casos se supondrá que se accede a todos los bucles condicionales y se calculará ese tiempo de ejecución. Por lo tanto, no resulta ningún inconveniente hacerlo de forma experimental ya que dará un tiempo preciso y requiere menos esfuerzo que realizar el análisis línea a línea.

Para medir de forma precisa el tiempo transcurrido se toma el instante temporal absoluto antes de comenzar y al terminar la tarea. La resta de ambos tiempos da el tiempo exacto que ha necesitado para ejecutarse.

Se estudia primero el software del esclavo y a continuación el del maestro.

Software del esclavo

El esclavo tiene 7 tareas a cumplir: Medición de voltajes, temperaturas, intensidades, balanceo, controlar la pantalla OLED, almacenamiento en micro SD y comunicaciones. A

continuación se detalla el funcionamiento de cada una de las tareas juntos con sus respectivos atributos temporales:

- Medición de voltajes (Tarea 1): Se encarga de leer el valor de todas las celdas conectadas a esa placa. Es la tarea que se necesita ejecutar con mayor frecuencia ya que es vital tener los voltajes de las celdas actualizados para garantizar la seguridad de las mismas. En caso de que ocurra un cortocircuito y el voltaje de una celda caiga este debe ser detectado en el mejor tiempo posible. Por lo tanto, se trata de una tarea periódica que debe cumplir un periodo de ejecución de 100ms. Se considera un tiempo de muestreo muy ajustado en comparación a otros BMS del mercado.
- Medición de temperaturas (Tarea 2): Esta tarea se encarga de medir la temperatura de todos los sensores Dallas conectados a la placa. La temperatura es una variable que cambia lentamente debido a los fenómenos físicos de la propia batería. En consecuencia, no se requiere medir el valor a una frecuencia tan alta y se considera adecuada un periodo de 1s para esta tarea periódica.
- Medición de intensidad (Tarea 3): Al igual que la medición de voltajes, esta es una tarea crítica ya que detectará sobrecargas en la batería y deberá actuar el contactor lo más rápido posible. Se establece un periodo de ejecución también de 100ms.
- Almacenamiento en la micro SD (Tarea 4): No es necesario guardar todos los datos muestreados ya que generaría muchísimos datos poco útiles y resultaría difícil asegurar los plazos de ejecución de las otras tareas por el tiempo que necesitaría esta para ejecutarse. Por lo tanto, se ha decidido establecer un periodo de 1s para guardar los valores instantáneos de las variables muestreadas.
- Pantalla OLED (Tarea 5): No requiere actualizarse cada poco tiempo ya que no hace falta tener tanta precisión a la hora de visualizar los datos en la pantalla. Esta simplemente se utilizará para ver de una forma rápida los valores instantáneos, no involucra ninguna parte vital de la placa. Periodo de 1s.
- Balanceo de las celdas (Tarea 6): Actúa sobre el registro de desplazamiento para así activar el balanceo de las celdas. Con el conocimiento de los voltajes de las celdas calcula cuales deben balancearse. Como la intensidad de balanceo es muy pequeña respecto a la intensidad de carga de la batería apenas se ve afectado el rendimiento del balanceo teniendo un periodo de ejecución muy corto. Por lo tanto, se considera correcto un periodo de 1s para esta tarea semiperiódica. No se considera como periódica a pesar de que no varíe su frecuencia y ejecución ya que esta tarea solo se ejecutará cuando la batería esté cargándose. Se puede catalogar como una tarea

esporádica que tiene como condición que la batería se esté cargando y que tiene una separación mínima de 1s entre dos activaciones consecutivas.

- Comunicaciones I2C (Tarea 7): Se encarga de escuchar al maestro y enviarle de vuelta las mediciones actualizadas (voltajes, temperaturas, intensidades y celdas que se están balanceando). No se puede considerar una tarea puramente periódica, aunque sea cíclica, ya que el inicio de la tarea depende del instante en el que el maestro establece la comunicación y esto a su vez depende del número total de placas esclavas con las que se comunica. A mayor número de placas esclavas mayor periodo de ejecución. Para el análisis se va a considerar un periodo estricto de 200ms entre dos comunicaciones consecutivas.

Conocidos los requisitos temporales de cada tarea se procede a definir el método de planificación, o lo que es lo mismo, la forma en la que el planificador repartirá los recursos del procesador para ejecutar el código.

En el peor de los casos el programa se puede interpretar como 7 tareas periódicas independientes. Se va a realizar una planificación con prioridades fijas y con desalojo: Una tarea que ha empezado a ejecutarse solo dejará de hacerlo si ha terminado su ejecución o porque una tarea más prioritaria quiere ejecutarse.

Para la asignación de prioridades en este tipo de planificadores se sigue el criterio RMS (Rate Monotonic Scheduling) ya que es óptima para dicho planificador y se cumple el requisito para utilizarla (que cada tarea tenga plazo igual al periodo). Consiste en dar mayor prioridad a las tareas que tengan menor periodo. Por lo tanto, las asignaciones de prioridades junto con el resto de atributos se resumen en la siguiente tabla:

Tarea	**P**	**C (ms)**	**T (ms)**	**D (ms)**
Medición de voltajes	3	10	40	40
Medición de intensidades	3	2	40	40
Comunicaciones	2	20	80	80
Almacenamiento micro SD	1	50	1000	1000
Pantalla OLED	1	20	1000	1000
Medición de temperaturas	1	10	1000	1000

Balanceo	1	0.12	1000	1000

Tabla 3. Atributos de las tareas del esclavo

La librería RTOS que se va a implementar en arduino solo permite asignar prioridades entre 1 y 3, siendo la mayor prioridad el valor de 3 y la menor el 1.

Según el teorema de Liu & Layland, para el modelo simple de tareas con asignación de prioridades RMS, los plazos de ejecución están garantizados si:

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N\left(2^{\frac{1}{N}} - 1\right)$$

Siendo N el número de tareas a controlar en el sistema:

$$\left.\begin{array}{c} U = \frac{10}{40} + \frac{2}{40} + \frac{20}{80} + \frac{50}{1000} + \frac{20}{1000} + \frac{10}{1000} + \frac{0.12}{1000} = 0.63 \\ N = 6 \rightarrow U = 6\left(2^{\frac{1}{6}} - 1\right) = 0.734 \end{array}\right\} \rightarrow 0.63 \leq 0.734$$

Como se puede observar se cumple el teorema, lo que quiere decir que queda garantizada la ejecución de las tareas por parte del procesador. A continuación se detalla cada una de las tareas:

Medición de voltajes

Los voltajes de las 8 celdas se miden a través de las entradas analógicas 6 y 7 del arduino. Para elegir que voltaje medir en cada una de las entradas hay que seleccionar el canal deseado en los multiplexores de la etapa de medición de voltajes. Estos multiplexores se controlan mediante 2 pines digitales, el 3 y el 4 concretamente. El diagrama funcional de los multiplexores es el siguiente:

Pines digitales de salida					Celdas medidas en las entradas analógicas
	Digital 4	Digital 3	Analógico 6	Analógico 7	
	LOW	LOW	5	1	
	LOW	HIGH	6	2	
	HIGH	LOW	7	3	
	HIGH	HIGH	8	4	

Ilustración 44. Tabla de verdad de los multiplexores

Definidos los pines la tarea tiene el siguiente bloque funcional:

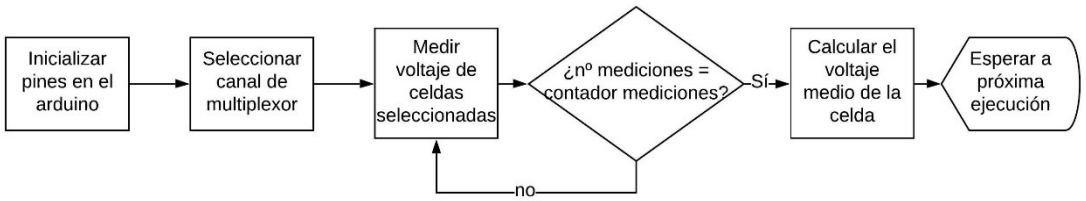


Ilustración 45. Bloque funcional de medición de voltajes

Como se puede observar se realiza una iteración en la medición de voltajes tantas veces como se estipule. Se ha considerado suficiente calcular el voltaje medio en la celda realizando 10 iteraciones, ya que aumentar el número de mediciones aumentaría proporcionalmente el tiempo de cómputo de la tarea, pero apenas aumentaría precisión en la medida.

Visualizando el esquema, la medición de voltajes se haría de la siguiente forma:

- Se configuran los pines digitales 3 y 4 como pines de salida. Esto solamente se ejecuta la primera vez que se ejecute la tarea. Cada reactivación consecutiva empezará desde el siguiente paso.
- Se coloca el pin digital 3 y 4 en un estado fijo para seleccionar dos canales a medir. Supóngase el primer caso en el que se coloca los dos pines a LOW.
- Se mide el voltaje en los pines analógicos 6 y 7, que corresponden a los voltajes de las celdas 5 y 1 en este caso. Esta medición se realiza 10 veces para calcular después el voltaje medio.
- Se modifica los pines digitales que controlan los multiplexores a la siguiente posición y se vuelve a empezar.
- Una vez realizadas todas las mediciones la tarea se suspende hasta que necesite volver a ejecutarse, esto es, 40ms después.

Medición de intensidades

Esta tarea es similar a la anterior, pero aún más sencilla. Utiliza un pin analógico para obtener la intensidad que circula por el sensor de Hall. El pin analógico utilizado es el 3. Por lo tanto, lo único que tiene que hacer cada vez que se ejecute es medir el voltaje en el pin analógico hasta 10 veces, como en la tarea anterior, y posteriormente obtener el voltaje medio. Con este voltaje medio se realiza la conversión de voltaje en la entrada analógica a los amperios que circulan por el sensor. Esta conversión se realiza una vez el sensor ha sido calibrado tal y como se explica en el apartado “Pruebas de la placa esclava”.

Comunicaciones I2C

Como ya se ha explicado anteriormente, la comunicación I2C es de tipo maestro-esclavo. El maestro enviará un mensaje al esclavo con el que se quiera comunicar y este le responderá cuando reciba el mensaje del maestro. Por lo tanto, la tarea de comunicación se ejecutará cuando el maestro establezca la comunicación con él, en vez de cada un tiempo fijo. Para implementar esta tarea en el arduino se hace uso de las interrupciones. Cuando se recibe un mensaje del maestro se interrumpe el hilo principal de ejecución del procesador para que la tarea de comunicaciones se lleve a cabo.

La respuesta del esclavo al maestro siempre será una trama del mismo tamaño, ya que siempre se envían los mismos datos, independientemente del valor de las mismas. Los datos a enviar son los siguientes:

- Voltajes de las 8 celdas: Cada uno de esos voltajes oscila entre 0 y 1025, esto es, una variable de tipo INT, lo que supone una longitud de 2 bytes por voltaje. En total 16 bytes.
- Intensidad medida en el Hall: Otro INT como en el caso de los voltajes. Otros dos bytes.
- Temperaturas de los Dallas: Da el resultado en coma flotante, pero se redondea y se envía como un INT. Cada una ocupa 2 bytes.

Por lo tanto, el número de bytes a enviar depende del número de celdas y sensores Dallas conectados a la placa. Por ejemplo, si se tienen conectadas 8 celdas y 4 sensores la longitud de la trama será de 26 bytes (8*2+2+4*2).

Como siempre se enviará esa cantidad de bytes, a fin de simplificar el protocolo de comunicación se va a indicar en una variable constante la longitud de la trama tanto en el esclavo como en el maestro.

Las comunicaciones I2C en arduino son muy sencillas gracias a la implementación de la librería “Wire.h”, que posee funciones sencillas de manejar. Entre ellas está la función “write()”, que es la encargada de enviar lo que se le indique dentro de las paréntesis. Se llamará a la función tantas veces como datos haya que enviar. Primero se enviarán los voltajes, después la intensidad y posteriormente las temperaturas.

Almacenamiento micro SD

Arduino se comunica con la micro SD mediante el protocolo SPI. Para ello hace uso de dos librerías: “SPI.h” y “SD.h”. El funcionamiento de la tarea es el siguiente:

Detecta si la tarjeta micro SD se ha introducido y se puede comunicar con ella o no. En caso de que pueda hacerlo la comunicación con la micro SD ya queda establecida. Si da un error por cualquier motivo el almacenamiento de datos no estará disponible.

Cada vez que se ejecute esta tarea se van a introducir los datos en la micro SD. En cada operación de escritura se debe abrir el archivo a utilizar, escribir los valores de las variables y cerrar el archivo al terminar. Solamente se puede acceder por una única tarea y un único archivo simultáneamente. Esto no supone ningún problema ya que solamente esta tarea estará accediendo a la micro SD.

Cada vez que se introduzcan nuevos datos se harán en el mismo orden que en las comunicaciones I2C: Voltajes, intensidad y temperaturas. Adicionalmente, antes de los voltajes se le sumará una variable más, que será el tiempo transcurrido desde el inicio del sistema. Todas estas variables serán delimitadas por comas, facilitando así la extracción de datos para el software de monitorización y análisis u otros programas como Excel, por ejemplo.

Pantalla OLED

La pantalla OLED se controla a través de I2C. La pantalla incorpora el controlador SDD1306 y tiene un tamaño muy reducido de 25mm x 14mm. Es monocroma y tiene una resolución de 128x64 pixeles [24].

La pantalla se programa mediante las librerías “Adafruit_SSD1306” y “Adafruit_GFX”. Ofrece funciones sencillas para insertar texto sin tener que ir dibujando los caracteres pixel a pixel. Se trata de presentar todos los datos en una única pantalla, esto es, sin transiciones.

Cada vez que se ejecute esta tarea deberá borrar la vista actual, volver a mostrar los datos actualizados y refrescar la pantalla para que los muestre. De no borrar los datos anteriores los nuevos se superpondrían a los anteriores y si no se refrescara la pantalla al finalizar la comunicación la pantalla no mostraría los datos, aunque estos hayan sido enviados.

Medición de temperaturas

Para interpretar el protocolo de comunicaciones propio de los sensores Dallas se utilizan las librerías “OneWire” y “DallasTemperature”. El arduino identifica a cada sensor a través de la dirección de los mismos. Cada sensor tiene una dirección única e irrepetible y debe ser identificada antes de medir las temperaturas. Para identificar las direcciones se utiliza un sketch proporcionado por el autor de la librería utilizada.

La inicialización de los sensores se realiza únicamente en la primera ejecución de la tarea.

Todas las direcciones están contenidas en un array. Cada vez que la tarea va a leer la temperatura de un sensor lee una de las direcciones del array y lee la temperatura. Para que lea todas las temperaturas en cada ejecución de la tarea el ciclo es el siguiente:

1. Coge la primera dirección.
2. Lee la temperatura.
3. Aumenta el índice del array.
4. Comprueba si hay un sensor para el índice actual. Si lo hay se vuelve a repetir el ciclo. Si no lo hay significa que ya ha leído todos los sensores y puede finalizar la tarea.

Balanceo

El balanceo se realiza a través del shift register SN74HC595D y la comunicación serie síncrona. El estado de los 8 pines se va a establecer a través de un único byte. Esto es posible ya que cada byte contiene 8 bits que pueden tener el valor de 0 y 1. Por lo tanto, cada bit va a representar a un pin del shift register y el valor de 0 o 1 especificará si ese pin debe estar a LOW o HIGH respectivamente. Se muestran un par de ejemplos a continuación:

- El bit 00010010 representaría el pin 4 y 7 en estado HIGH y el resto en LOW.
- El bit 10000000 representaría el pin 1 a HIGH y el resto a LOW.

Como se puede observar los pines van ordenados de izquierda a derecha en orden ascendente. El algoritmo de balanceo consiste en poner a balancear todas las celdas salvo la que tenga el menor voltaje, con el fin de que el resto de celdas se vayan igualando en tensión a esta.

A la hora de implantarlo en el arduino se hace de la siguiente manera: Se va a recorrer un bucle que comience con a primera celda y así hasta la última:

- Se compara el voltaje de la celda con el voltaje mínimo detectado.
    - Si el voltaje es menor que el voltaje mínimo previamente detectado:
        - Guarda tanto el voltaje de esta celda como el índice de la celda comparada.
        - Pasa a comparar la siguiente celda.
    - Si el voltaje es mayor:
        - No hace nada y pasa a comparar la siguiente ceda.

Como se puede observar se han necesitado 3 variables a lo largo del ciclo:

- El índice que indica en todo momento que celda se está comparando. Esta tendrá un valor entre 1 y 8, esto es, el número de celdas que haya conectadas.
- El voltaje mínimo para compararla en cada iteración.
- El valor del índice en el que esté el voltaje mínimo para saber cuál será el bit que se dejará a 0 a la hora de enviar el byte al shift register.

Por ejemplo, si se diera el caso de que la celda 6 tiene el menor voltaje habría que poner todos los bits salvo el sexto a 1 para que todas las celdas se balanceasen salvo esa.

La tarea sigue el siguiente diagrama:

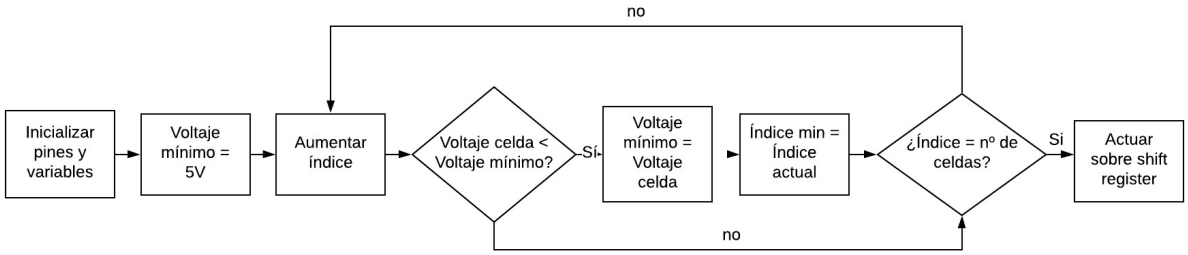


Ilustración 46. Diagrama funcional del balanceo

Software del maestro

El software del maestro tiene que realizar las siguientes 9 tareas:

- Control de contactores: Se encarga de abrir o cerrar los contactores en función de los valores muestreados de los esclavos. Debe comprobar si hay un motivo de abrir o cerrar un contactor y actuar en consecuencia. Es una tarea crítica ya que es la principal medida de seguridad ante un problema en la batería. Se establece un periodo de ejecución de 50ms.
- Supervisión de estado de contactores: Averigua el estado de los contactores que esté supervisando y en función de ello abre o mantiene cerrados los contactores que controla. Requiere un periodo de ejecución corto como la anterior tarea para mantener la seguridad del sistema, esto es, de 50ms.
- Medición de temperaturas con el termistor: Mide las temperaturas de los dos termistores que se le pueden conectar. Al igual que en el esclavo no hace falta tener un periodo corto por la dinámica de los sensores, con lo que se establece un periodo de 1s

- Supervisión IMD: Se trata de una medida de seguridad ya que mide el aislamiento entre las distintas partes de la motocicleta, con lo que se establece un periodo de 50ms.
- Pantalla OLED: Mismo criterio que para la placa esclava, 1s.
- Almacenamiento en micro SD: Mismo criterio que para la placa esclava, 1s.
- Comunicaciones I2C: Esta tarea se ejecutará cada vez que vaya a comunicarse con cada uno de los esclavos. Se va a estipular un periodo de 50ms.
- Cálculo del SOC: Calcula el porcentaje de batería disponible en ese instante. No requiere un periodo rápido puesto que el SOC apenas disminuye en intervalos de tiempo cortos. Periodo de 1s.
- Comunicaciones bluetooth: Se va utilizar para comunicarse o bien con un ordenador o bien con otros elementos de la motocicleta. Se busca un compromiso entre la cantidad de datos que se envían y el tiempo de procesador que se va a dedicar a ello. Se considera correcto un periodo de 1s, al igual que con el almacenamiento de datos de la micro SD.
- Comunicaciones CAN: Principalmente se utilizará para comunicarse con el dashboard, con lo que un periodo de 200ms es más que suficiente. Un valor mayor sería ineficaz ya que no hace falta refrescar los datos tan rápido en el dashboard.
- Comunicaciones vía serie con el ordenador: Mismo criterio que con la comunicación Bluetooth, 1s.

En el peor de los casos el programa se puede interpretar como 9 tareas periódicas independientes. Se va a realizar una planificación con prioridades fijas y con desalojo, al igual que con el esclavo.

Para la asignación de prioridades en este tipo de planificadores se sigue el mismo criterio de RMS (Rate Monotonic Scheduling. Las asignaciones de prioridades junto con el resto de atributos se resumen en la siguiente tabla:

Tarea	**P**	**C (ms)**	**T (ms)**	**D (ms)**
Control de contactores	3	5	50	50
Supervisión de estado de contactores	3	5	50	50
Supervisión IMD	3	2	50	50

Comunicaciones CAN	2	10	200	200
Comunicaciones I2C	2	5	50	50
Cálculo del SOC	1	2	1000	1000
Medición de termistores	1	4	1000	1000
Comunicaciones Bluetooth	1	5	1000	1000
Comunicaciones serie	1	5	1000	1000
Pantalla OLED	1	20	1000	1000
Almacenamiento micro SD	1	100	1000	1000

Tabla 4. Tareas del maestro.

Se procede a realizar el cálculo del teorema de Liu & Layland de la misma manera que en el caso anterior:

$$\left\{\begin{array}{c} U = \frac{5}{50}+\frac{5}{50}+\frac{2}{50}+\frac{10}{200}+\frac{5}{50}+\frac{2}{1000}+\frac{4}{1000}+\frac{5}{1000}+\frac{5}{1000}+\frac{20}{1000}+\frac{100}{1000} = 0.426 \\ N = 11 \rightarrow U = 11\left(2^{\frac{1}{11}}-1\right) = 0.715 \end{array}\right\}$$

$$\rightarrow 0.426 \leq 0.715$$

Se cumple el teorema, lo que quiere decir que queda garantizada la ejecución de las tareas por parte del procesador. A continuación se describe el código de cada una de las tareas:

Control de contactores

El software es mucho más sencillo que el hardware para el caso del control de contactores. En caso de que se quiera abrir o cerrar el contactor bastará con cambiar el estado del pin que controla dicho contactor. No es necesario atender a la precarga ya que de eso se encarga una placa externa al BMS.

Simplemente habrá que actuar sobre las salidas digitales mapeadas del A1 al A3 en función de qué contactor se quiere abrir o cerrar. Para cerrar un contactor hay que poner la salida digital a 1 y viceversa.

La decisión de abrir o cerrar contactor la tomará en función de los valores en todas las variables. Si estas son correctas el contactor se cerrará. En caso contrario ordenará que el contactor permanezca abierto.

Supervisión de estado de contactores

La supervisión del estado de contactores es similar a la del control del mismo, a diferencia que se lee una entrada digital en vez de controlar una salida. Cada vez que se ejecuta la tarea se lee el estado de los pines entre D6 y D8 y la secuencia es la siguiente: Si el contactor está cerrado llegará un 0 a la entrada digital y viceversa.

En caso de que se detecte un caso indeseado (contactor cerrado que no debería o contactor abierto que debería estar abierto) ejecutará la tarea de control de contactores para remediarlo.

Supervisión IMD

El estado del IMD se lee a través de la entrada digital A0. En caso de que el valor sea 1 significa que los valores medidos por el IMD son correctos, esto es, que el aislamiento entre el alto voltaje y el bajo voltaje en la motocicleta está entre los valores admitidos. Si no es así el arduino leerá un 0 en la entrada digital y el control de contactores deberá actuar.

Medición de termistores

Los termistores son leídos en los pines A6 y A7. Una vez medido el voltaje se debe hacer la conversión de voltios a grados centígrados. Para ello se parte del modelo de Steinhart-Hart [25]:

$$T\,(ºC) = \frac{1}{A + B * log(R) + C * (\log(R))^3}$$

Los parámetros de A, B y C generalmente vienen en el propio datasheet del termistor. En caso de que no fuera así se podría obtener experimentalmente en el taller.

El valor de R es la resistencia que ofrece el termistor cuando es sometida a esa temperatura. Conocido el voltaje del termistor se puede obtener el valor de R del mismo ya que este termistor está conectado en serie con una resistencia. Para ello primero se obtiene la intensidad que circula por el termistor:

$$5V = V_{resistencia\ seri} + V_{medido\ arduino} \rightarrow V_{resistencia\ serie} = 5V - V_{medido\ arduino} \rightarrow$$

$$10.000 * I = 5V - V_{medido\ arduino} \rightarrow I = \frac{5V - V_{medido\ arduino}}{10.000}$$

Se ha puesto un valor de 10.000Ω para la resistencia en serie ya que es la que se ha colocado en el circuito. El valor de R por lo tanto es:

$$R * I = V_{medido\ arduino} \rightarrow R = \frac{V_{medido\ arduino}}{\frac{5V - V_{medido\ arduino}}{10.000}} \rightarrow R = \frac{10.000 * V_{medido\ arduino}}{5V - V_{medido\ arduino}}$$

Como el voltaje se mide sobre una escala de 10 bits, en realidad los 5V debería ser un valor de 1025 para realizar la conversión correctamente. En conclusión:

$$R = \frac{10.000 * V_{medido\ arduino}}{1025 - V_{medido\ arduino}}$$

En conclusión, para obtener la temperatura del termistor se debe obtener el valor de R mediante la fórmula anterior para posteriormente sustituirla en el modelo de Steinhart-Hart.

Cálculo del SOC

El cálculo del SOC (State Of Charge) de una batería de ion de Litio se puede realizar de diversas formas. Muchas de ellas conllevan algoritmos matemáticos complejos en los que hay que modelizar la batería a través de rigurosos ensayos.

Existen algoritmos más sencillos que se obtienen a través de las curvas de funcionamiento a distintas temperaturas y descargas de la batería.

El más sencillo de todos es suponer que la batería tiene una capacidad conocida y se estima su capacidad restante a través del consumo en la misma. Esto es, si la capacidad de la batería fuera de 1Ah y se consumiera 1A durante 30 minutos, la batería restante sería del 50% (suponiendo que inicialmente estaba al 100%). El resultado se obtiene de la siguiente fórmula:

$$SOC_n = SOC_{n-1} - \frac{I * t}{C_{batería} * 3600} * 100 = SOC_{n-1} - \frac{I}{C_{batería} * 36}$$

El resultado del SOC se obtiene en % puesto que la intensidad se mide en amperios (A), el tiempo en segundos (s) y la capacidad en amperios hora (Ah). Se divide entre 3600 para realizar la conversión entre segundos y horas y se multiplica por 100 para ponerlo en porcentaje. El tiempo se ha sustituido como 1s ya que esa intensidad será la intensidad media medida en un periodo de 1s.

En caso de que el SOC llegara debajo del límite establecido se deberá abrir el contactor principal de la motocicleta.

Comunicaciones I2C

El maestro pedirá una cantidad de bytes específica a cada maestro en función del número de sensores que tenga conectados. Ese número es conocido y se indica en la configuración del programa, a través de una variable.

Tras un breve periodo recibirá la trama des esclavo y procederá a leerla. Para ello simplemente irá leyendo byte a byte y almacenándolo en las variables que le corresponde. Es una tarea sencilla ya que el orden es conocido de antemano. Se envían los voltajes, después las intensidades y finalmente las temperaturas.

Almacenamiento micro SD

Es exactamente idéntica a la tarea que tiene la tarjeta esclava, a diferencia de que esta guarda los datos de todos los esclavos, mientras que los esclavos solo guardan los datos de sus propias variables.

Comunicaciones serie

Las comunicaciones serie van a tener 3 funcionalidades en el proyecto:

- Programar las placas esclavas y maestras. No es necesario programar ningún software adicional ya que la programación se realiza mediante el entorno arduino IDE. Solamente hay que conectar el arduino al ordenador por USB y subir el programa al microcontrolador.
- Extraer datos en directo: El maestro enviará una trama de datos delimitados por coma para analizarlos en directo en el software de monitorización y análisis. Los datos se enviarán a través de la tarea de la micro SD. Esto se debe a que la trama que se guarda en la micro SD es la misma que se envía por el puerto serie. Gasta menos recursos enviar la trama por serie tras haberla guardado en la micro SD, ya que la trama que se envía sigue guardada en una variable local y no es necesario acceder a la micro SD a leer la última trama como si se hiciera desde una tarea independiente.
- Extraer datos de la micro SD: Existirá esta opción adicional para poder realizar un backup de todos los datos almacenados. Esta función se hará a través de un programa independiente con el fin de simplificar el funcionamiento y concurrencia de tareas. El programa consistirá en abrir cada uno de los archivos de texto almacenados en la micro SD y enviarlos al ordenador.

Comunicaciones Bluetooth

El funcionamiento y programación de esta tarea es exactamente idéntico al de la comunicación serie. Hace uso de las mismas funciones y se utilizan en el mismo momento que la comunicación serie. Cumple la misma función que el puerto serie, pero de forma inalámbrica.

Comunicaciones CAN

El protocolo CAN es un poco más complejo de programar que la comunicación serie o bluetooth. Esto se debe a que las funciones que utiliza requieren de mayor conocimiento sobre el protocolo que en los otros casos. Para enviar una trama en CAN se hace de la siguiente forma:

- CAN.sendMsgBuf(id, ext, len, data_buf);
    - **id**: Es el identificador de la trama. Sirve para establecer la prioridad de la trama.
    - **ext**: Representa el tipo de trama que se va a enviar. En CAN se pueden enviar 2 tipos diferentes: El estándar y el extendido. Para enviar una trama estándar se pone un 0 en este parámetro y un 1 para la extendida.
    - **len**: Se indica la longitud de la trama de datos. Por cada trama se pueden enviar hasta 8 bytes.
    - **data_buf**: Array de hasta 8 bytes que contiene los datos que se van a enviar.

Para recibir un mensaje se utiliza la siguiente función:

- CAN.readMsgBuf(len, buf);
    - **len**: Indica la longitud de la trama que se va a recibir.
    - **buf**: Es la variable en la que se guardarán la trama que se va a leer.

Pantalla OLED

El software de la pantalla del maestro es exactamente idéntico al de los esclavos, pero mostrando todos los parámetros, en vez de los de cada esclavo. La disposición de los datos es igual que en el caso del esclavo, esto es, muestra los datos recogidos de cada placa en una única pantalla. Sin embargo, como tiene que mostrar los datos de todos los esclavos, refrescará la pantalla en cada ejecución mostrando los datos de un esclavo.

La motocicleta, por ejemplo, tendrá 4 placas esclavas. Por lo tanto, serán necesarias 4 ejecuciones de la tarea para mostrar todos los datos. Como la tarea se ejecuta cada 4s, ese será el tiempo necesario para visualizar todos los datos.

SOFTWARE DE MONITORIZACIÓN Y ANÁLISIS DE FUNCIONAMIENTO

El software de monitorización y análisis es desarrollado mediante "processing". Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java y de fácil utilización. Se ha escogido este lenguaje por tener conocimientos previos de Java y ser una versión más "visual y sencilla" para realizar entornos gráficos. Además, ofrece una portabilidad plena a distintas plataformas.

Las funciones principales del software son las siguientes:

- Recibir los datos del BMS en directo.
- Extraer los datos guardados en la memoria del BMS de un tiempo anterior.
- Visualizar todos esos datos ya sea en directo o a través de los datos extraídos del BMS. Para ello se usarán distintos elementos como gráficos de dos dimensiones, de barras, diales, animaciones, etc.

Los distintos elementos de la interfaz para visualizar los datos que tienen en común los 3 modos de funcionamiento son:

- 4 gráficas: Los datos a graficar en cada una de ellas se seleccionan en las dos listas desplegables que hay en la parte superior izquierda. Para cargar el dato seleccionado en la gráfica se debe clicar en el botón que se encuentra a la derecha de las listas desplegables y que tiene un gráfico como icono.
- Un cuadro de texto en el que se exponen todos los datos de los distintos sensores. En el modo de conexión en directo se muestran los datos de la última trama recibida, mientras que en los otros dos modos se enseñan los datos del instante que se está visualizando en ese momento.
- 2 barras verticales que mostrarán los datos que se desee en la pestaña de configuración del programa.
- 2 diales que se configurarán de la misma forma que las 2 barras verticales.
- 1 mapa que indica la posición de la motocicleta en el circuito. Como la motocicleta siempre estará en un circuito solo hace falta cargar la imagen del circuito en el que va a circular y la posición de la moto en la imagen se logra interpolando las coordenadas reales de la moto con las que le correspondería en la imagen. Si se tratara de un vehículo que circulara fuera del circuito no habría más que precargar distintos mapas y realizar el mismo procedimiento.

- 1 gráfico que muestra el voltaje de cada una de las ramas en serie de la batería. Se le ha dado un buen tamaño a la gráfica ya que muestra claramente el estado de la batería, el nivel de balaceo entre las celdas y la carga de las mismas con un solo vistazo.
- 1 gráfico que muestra las temperaturas de los sensores que están colocados en la batería. Son 14 sensores y la figura tiene la forma de la batería con el fin de visualizar de una forma más clara el estado térmico de las baterías. Se podrá interpretar qué zonas se calientan más, siendo este un dato valioso para futuros diseño de baterías.
- 1 cuadro de texto que da información adicional del estado de las celdas, así como voltaje máximo, mínimo y medio.
- Otro cuadro de texto que da la misma información que el anterior, pero en este caso sobre las temperaturas de la batería.

Adicionalmente se van a colocar los siguientes elementos en función del modo de funcionamiento seleccionado en el programa:

- 1 botón para conectar y desconectar la conexión bluetooth en los dos modos que requiere conectarse con el BMS. También habrá dos campos en los que se indica que puerto y velocidad de trasmisión se utilizará para la comunicación.
- 1 botón para cargar el archivo deseado del ordenador en el modo de cargar datos desde el PC.
- 1 botón que permite cambiar de la visualización estándar de los gráficos a una animación de los mismos. En el modo estándar los gráficos, diales y barras muestran el dato del instante seleccionado del archivo mediante el uso de una barra deslizable que hay al lado del botón. En el caso de que se elija la animación se hace una rápida transición de la evolución temporal de los datos. La velocidad de transición entre un dato y otro se realiza con la misma barra. Esta función permite ver de una forma clara el desarrollo de los valores de los sensores a lo largo del tiempo.

La apariencia del software es la siguiente:

Ilustración 47. Software de análisis y monitorización

La interacción de los tres modos de funcionamiento con los distintos elementos se puede representar mediante el siguiente diagrama funcional:

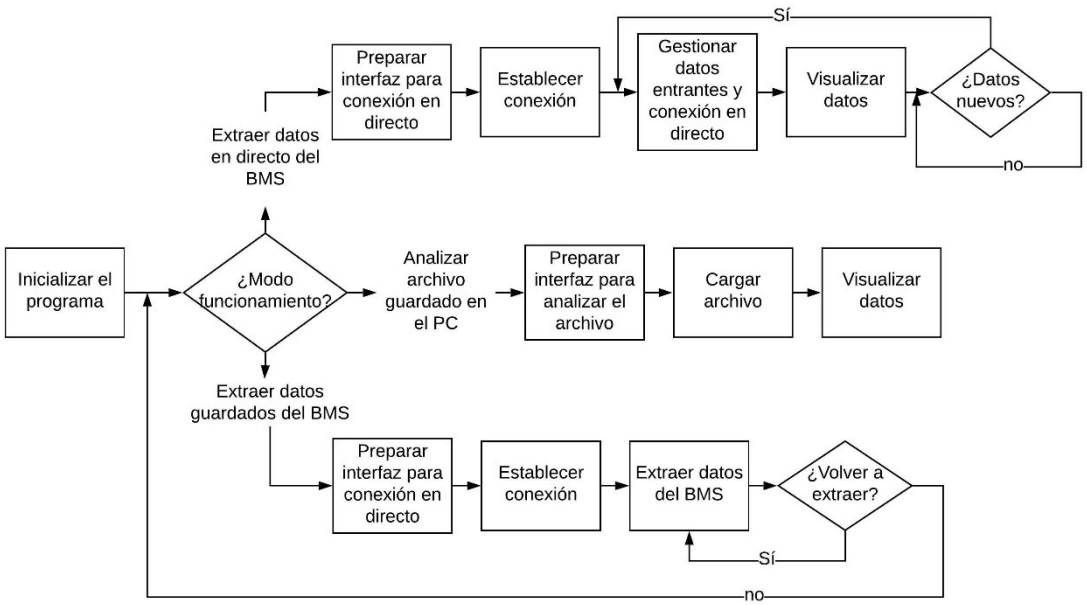


Ilustración 48. Diagrama funcional del software de análisis y monitorización

Como se puede observar, una vez abierto el programa para que este haga algo ha de indicarse qué tipo de visualización de datos se quiere realizar. El funcionamiento de cada uno se explica en los siguientes apartados:

Extraer datos en directo del BMS

Para cada uno de los tipos de análisis se cargan sus interfaces necesarias. En este modo se muestra el botón de conectar/desconectar y los parámetros de configuración de la conexión bluetooth y se esconde el botón y barra deslizable de la animación en caso de que anteriormente estuviera en alguno de los otros dos modos.

Posteriormente, se debe seleccionar el puerto "COM" y la velocidad de transmisión en la que se va a realizar la comunicación bluetooth. Para averiguar que puerto COM se va a utilizar, una vez emparejado por bluetooth el BMS al ordenador, se accede a la pestaña de administrador de dispositivos de Windows y se ve cuál es. La velocidad de trasmisión debe ser la misma a la que se haya programado el BMS desde el Arduino IDE. Ambos parámetros serán siempre los mismos, siempre y cuando no se elimine el emparejamiento entre el BMS y el ordenador.

Una vez configurados los dos parámetros se clica en el botón correspondiente para establecer la comunicación. Nada más clicar se abre una ventana del explorador de archivos pidiendo el lugar y nombre del archivo que se quiere utilizar o crear para realizar el guardado de todos los datos entrantes. Si estos parámetros son correctos y el BMS se encuentra en el rango de alcance del bluetooth se inicia la conexión.

A partir de aquí el software esperará a que haya datos entrantes del BMS en el buffer del puerto. El software procederá a leer la trama (string que contiene los valores de todos los sensores en un instante de muestreo), decodificar el mensaje y visualizar todos los datos en las distintas interfaces. Si detectara algún error en la trama de datos esta sería ignorada.

Los datos llegan ordenados según la configuración del BMS y están delimitados por comas entre ellos. Adicionalmente, cada vez que se envía una trama entera se imprime un salto de línea dejando así una trama por cada fila. Se envían de esta forma para facilitar su lectura y escritura como archivos de formato "csv".

Las gráficas pueden mostrar los datos de dos formas: Graficando todos los datos recibidos o los que se deseen utilizando los botones que tienen a su derecha. Estos botones permites realizar zooms tanto en el eje X como en el eje Y, desplazarse por la gráfica y restaurar la visualización a todos los datos. El resto de interfaces (gráfico de celdas, barras, diales…)

muestran solamente los datos de la última trama, con lo que no hay nada especial que comentar de ellas.

Mientras el software funciona del modo descrito existe la posibilidad de que ocurran dos eventos adicionales:

- Que se decida cerrar la conexión volviendo a clicar en el botón. Cerrará el puerto de comunicaciones limpiando el buffer previamente (por si se decide volver a establecer la conexión de nuevo) y guardará y cerrará el archivo csv en el que estaba haciendo todo el salvado de datos. Posteriormente esperará nueva orden del usuario.
- Que se decida cambiar de tipo de análisis de datos. Esto conllevará al cierre del puerto de comunicaciones bluetooth y se cambiará al modo seleccionado cambiando las interfaces que hagan falta.

Analizar un archivo guardado en el PC

En este modo no es necesario establecer la conexión con el BMS, con lo que no es necesario definir los parámetros de comunicación ni mostrar el botón de conexión bluetooth. En lugar de ello se muestra un nuevo botón para cargar el archivo que se quiere leer desde el PC y el botón y barra deslizable de la animación de datos.

Cuando se pulsa el botón de cargar se abre el explorador de Windows para seleccionar el archivo a cargar. Si el directorio y tipo de archivo es correcto este se abre y se colocan todos los datos en la interfaz según la configuración establecida.

La visualización de datos es la misma en los tres modos de funcionamiento disponibles, a diferencia del modo en directo que se da la posibilidad de realizar la animación de datos. Esto no es posible en mediante la conexión en directo ya que están llegando constantemente datos nuevos y, por lo tanto, no tiene mucho sentido realizar una animación.

Dos eventos adicionales pueden ocurrir mientras se está en este modo: Que se decida cambiar de tipo de análisis de datos o que se quiera cargar otro archivo del PC, lo que volvería a la situación inicial de este modo.

Extraer datos guardados del BMS

Este modo es una mezcla de los anteriores dos. Tiene la misma interfaz para visualizar los datos de un archivo del PC (con lo que mantiene el botón y barra deslizable de la animación),

pero necesita comunicarse por bluetooth para extraer el archivo. Por lo tanto, se muestra el botón de conectar/desconectar del bluetooth en lugar del botón de cargar archivo del PC.

Los eventos adicionales que pueden ocurrir son los mismos que en el modo anterior, teniendo como única diferencia que el archivo a extraer es desde el BMS y no el PC.

PUESTA EN MARCHA. RESULTADOS OBTENIDOS

Ensamblaje

Para el ensamblaje de las placas electrónicas se van a tener en cuenta las siguientes consideraciones:

Los componentes electrónicos se van a soldar a cada placa en la propia universidad. Para ello se hará uso de un horno convencional y el procedimiento es el siguiente:

1. Colocar la pasta de soldadura. Para ello se fija la placa que se quiere soldar sobre la mesa y cuidadosamente se coloca el stencil encima, dejándolo alineado con la placa. Posteriormente se aplica uniformemente la pasta con el uso de una espátula o similar. Al terminar se retira el stencil y ya se tiene la placa lista para colocar los componentes SMD.

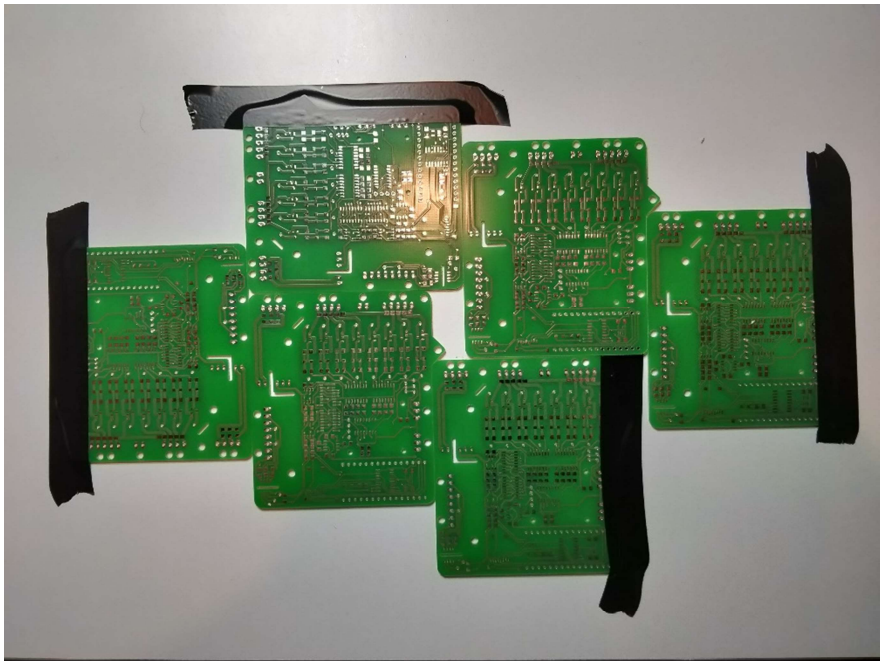


Ilustración 49. Fijación de las placas a soldar

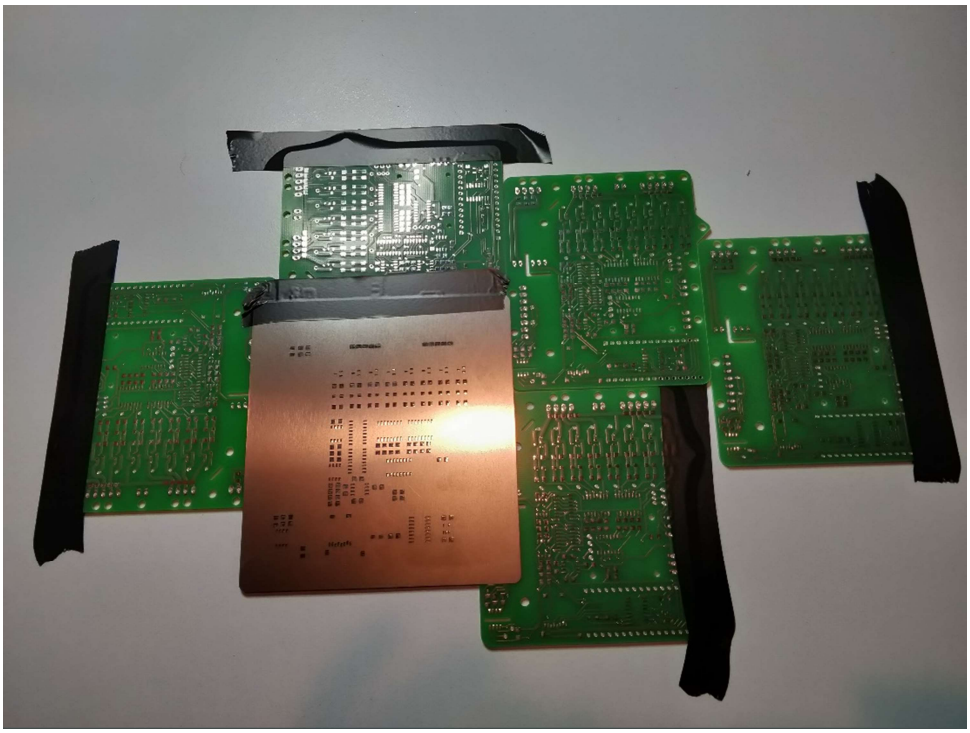


Ilustración 50. Colocación del Stencil

2. Se colocan los componentes SMD uno a uno con el uso de unas pinzas y mucho cuidado con el fin de no mover la pasta de soldadura a contactos cercanos y facilitar la aparición de cortocircuitos posteriormente. Esta es la principal razón de la restricción del tamaño de componentes; si se escogieran encapsulados muy pequeños aparecerían cortocircuitos entre pines, por la falta de precisión a la hora de colocar los componentes.

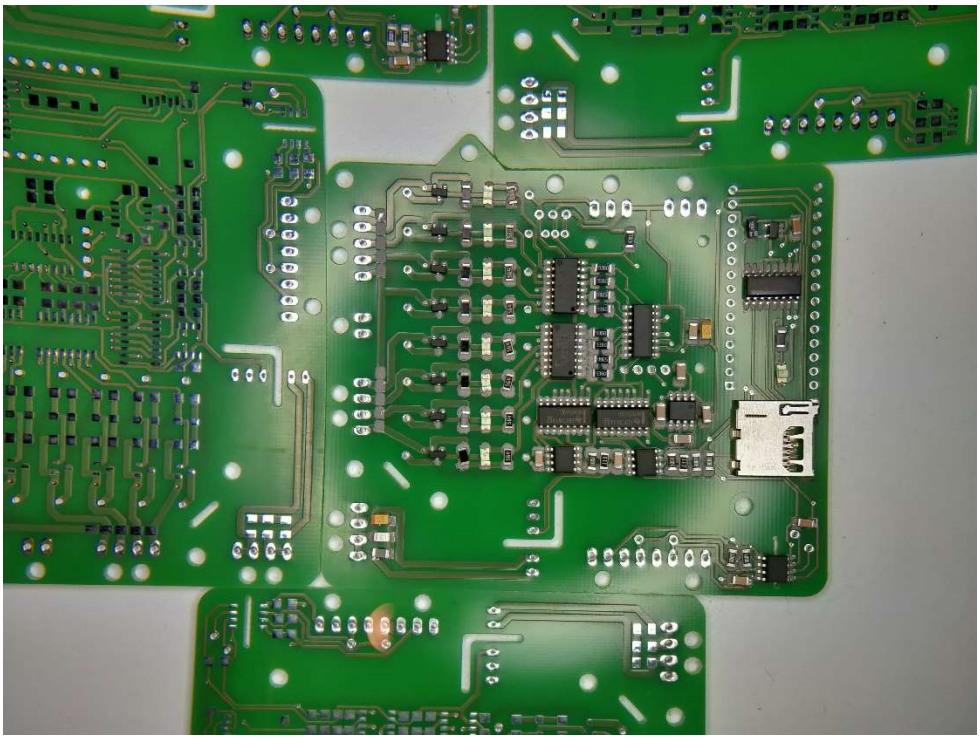


Ilustración 51. Colocación de componentes

3. Se debe tener en cuenta el perfil de temperatura de la pasta de soldar y la temperatura máxima a la que se pueden someter los componentes. Para seguir el perfil de temperatura necesario, las empresas hacen uso de un horno longitudinal en el que las placas van avanzando por él. Como el horno tiene la temperatura ajustada en sus distintas zonas, se logra aplicar el perfil de temperatura correcto. Con el fin de realizar lo mismo en la propia universidad se usará un horno de cocina convencional. Como solamente se puede aplicar una única temperatura en cada instante, se procede a calentar el horno a la temperatura necesaria en cada instante en vez de desplazar la placa por un horno con distintas temperaturas. Para seguir el perfil de soldadura en el horno convencional se necesita utilizar un sensor de temperatura y controlar la potencia del propio horno. La potencia se puede controlar con un simple relé y un microcontrolador que lee la temperatura actual con el sensor, la compara con la consigna y decide si calentar más o no. En este caso, como el horno daba un perfil de calentamiento muy similar al del datasheet de la pasta de soldar se ha obviado el control de temperatura y simplemente se ha metido la placa con el horno frío y se ha puesto a máxima potencia hasta lograr la máxima temperatura del perfil de soldadura. Por lo tanto, el único control a realizar es que

se haya alcanzado los XºC. Los resultados son realmente buenos y semejantes a los que se obtendrían en una empresa de ensamblaje.

Ilustración 52. Soldadura de los componentes en el horno

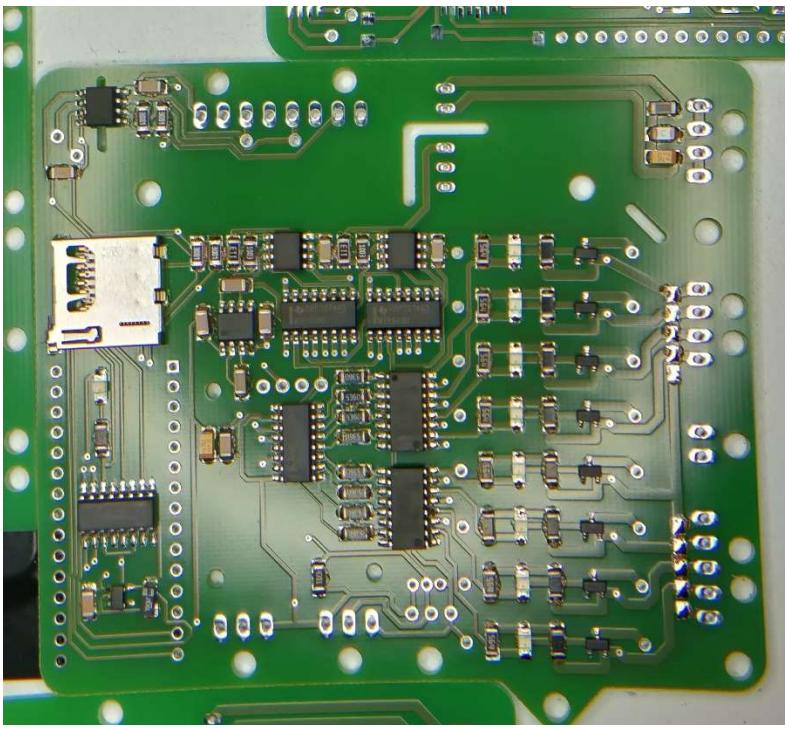


Ilustración 53. Resultado obtenido

4. Soldar los componentes THT con un soldador normal y corriente.
5. Asegurar que todas soldaduras están bien hechas y corregirlas manualmente en caso de que se haya hecho un cortocircuito tanto en el horno como con el soldador.

A continuación se muestran imágenes del ensamblaje completo:

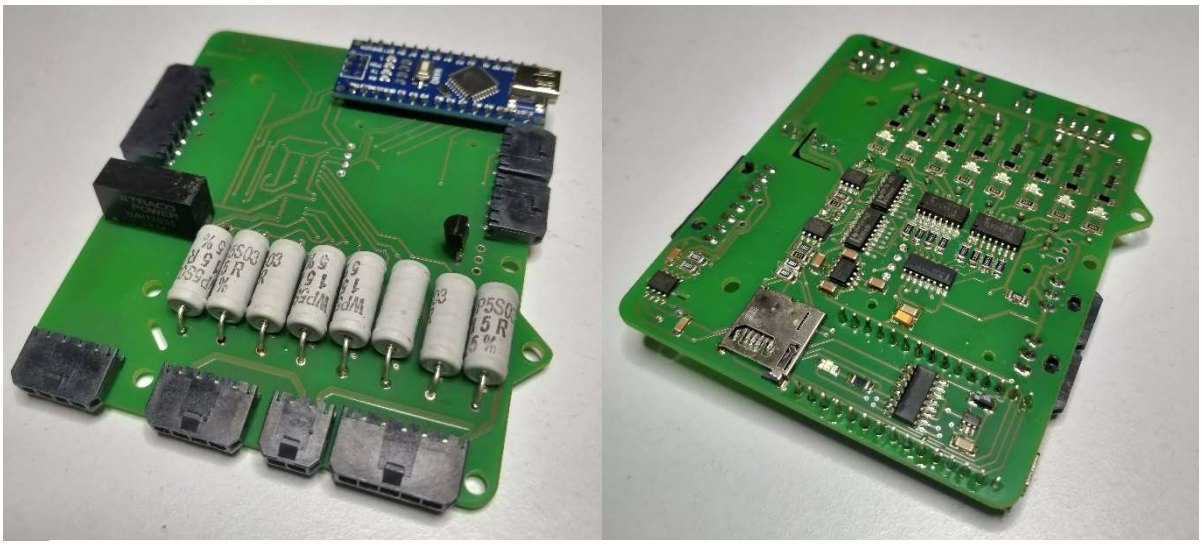


Ilustración 544. Placa esclava

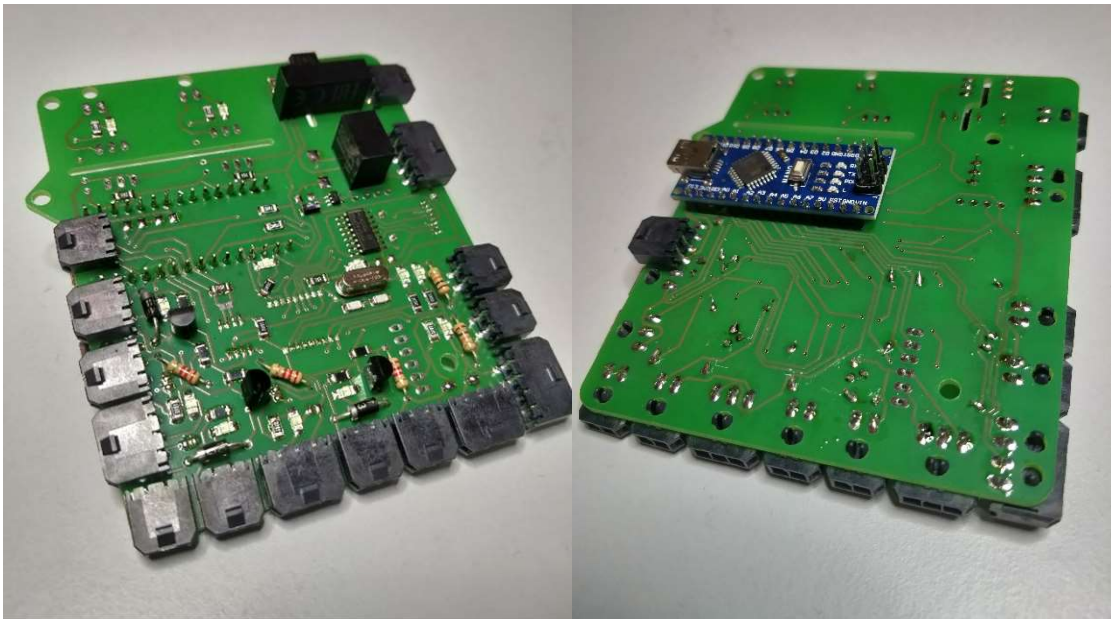


Ilustración 55. Placa maestra

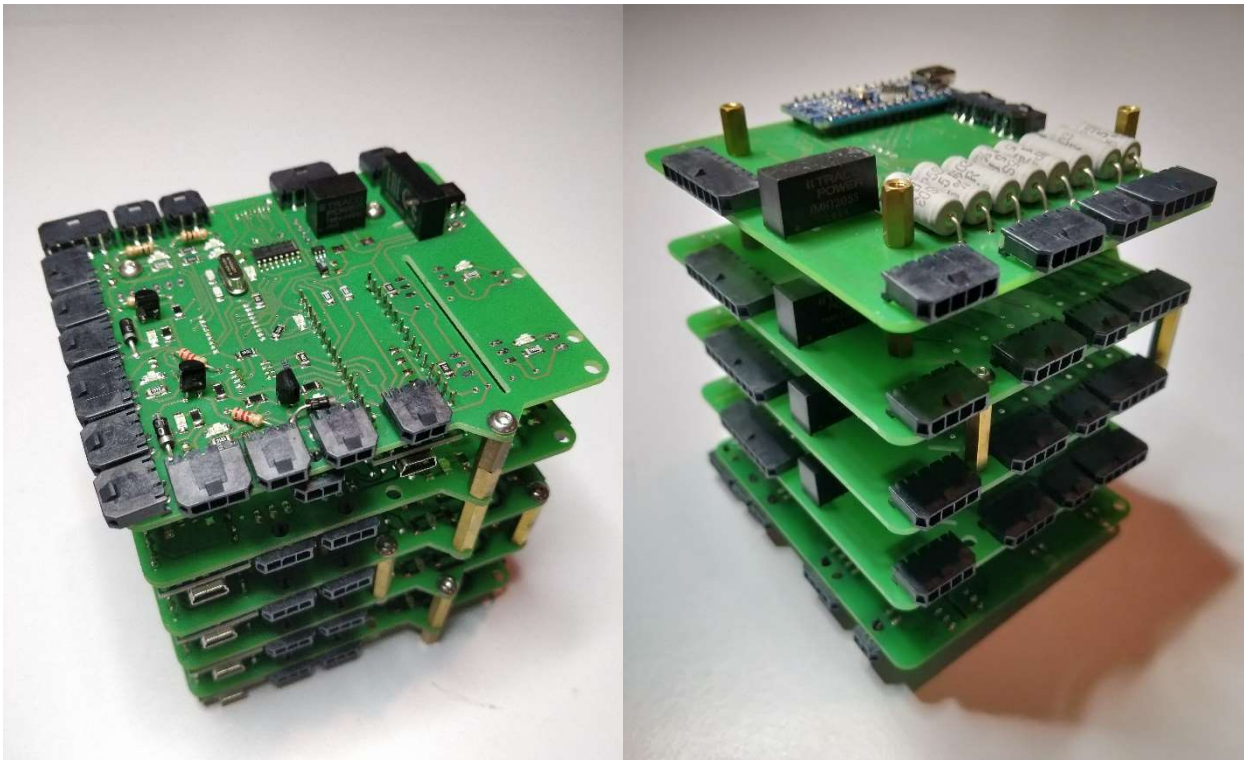


Ilustración 55. Ensamblaje del BMS al completo

Retrasos por fuerzas mayores

Como bien se sabe el presente año ha sufrido un golpe fuerte pro el covid-19. La pandemia ha obligado a cerrar la universidad desde mediados de marzo hasta mediados de junio, perdiendo 3 meses para poder hacer pruebas. A partir de junio, aunque se haya podido entrar al taller, no ha habido una gran flexibilidad de horarios.

Todo este tiempo se ha intentado aprovechar de la mejor forma: Realizando la mayor parte del trabajo desde el ordenador en el confinamiento, llevando todos los componentes para realizar el ensamblaje y pruebas desde casa, etc.

Sin embargo, todo esto no ha sido suficiente para concluir el proyecto con las expectativas esperadas. No se ha podido comprobar todas las partes del BMS de la forma en la que se habían planeado, ya sea por falta de tiempo o de recursos. Las partes que no se han podido comprobar son las siguientes:

- Placa esclava:
    - Medición de temperatura de las resistencias de la etapa de balanceo.
- Placa maestra:
    - Medición de temperatura mediante termistores.

- OLED con los datos de todos los esclavos.
- Supervisión de contactores. El control de contactores si se ha podido probar como se explicará más adelante.

- Comunicación del BMS con el software de monitorización y análisis: Todas las pruebas realizadas en este software se han hecho a través de datos simulados, con lo que no era una situación real. Sin embargo, los datos procedentes del BMS tienen el mismo formato, con lo que se espera que funcione sin problemas.
- Pruebas ante fallas: Medición de tiempos de apertura y cierre de contactor ante diferentes tipos de falla (caída de voltaje, sobrecargas, temperatura excesiva, etc). No se han tenido los medios necesarios en el tiempo disponible.
- Tampoco se ha podido hacer una prueba de campo del BMS al completo con la batería de la motocicleta.

A pesar de este gran inconveniente, a continuación se muestran las pruebas que si se han podido hacer:

Pruebas de la placa esclava

Balanceo

El funcionamiento del balanceo se va a comprobar colocando 8 celdas desbalanceadas y monitorizándolo en directo mientras se cargan. Se ha decidido cargar la batería con una intensidad de 3A y su curva de carga ha sido la siguiente:

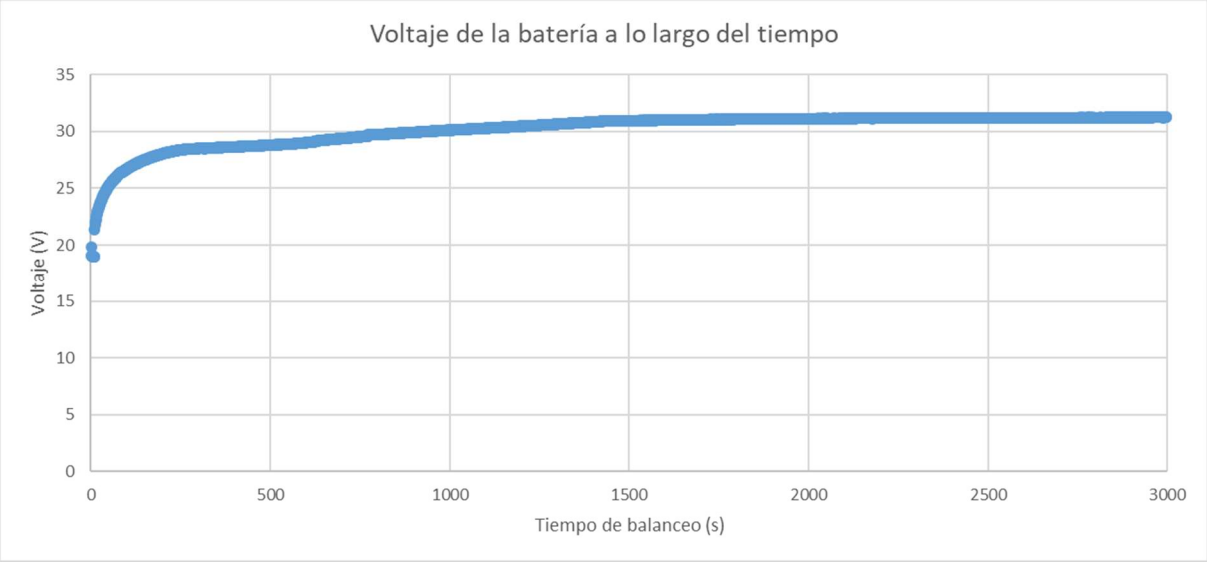


Ilustración 56. Curva de carga de las celdas

El voltaje graficado es el correspondiente a las 8 celdas en serie. Mientras se cargaba la batería se ha probado el algoritmo de balanceo. Para comprobar la efectividad del mismo se ha calculado la desviación entre el voltaje de las 8 celdas:

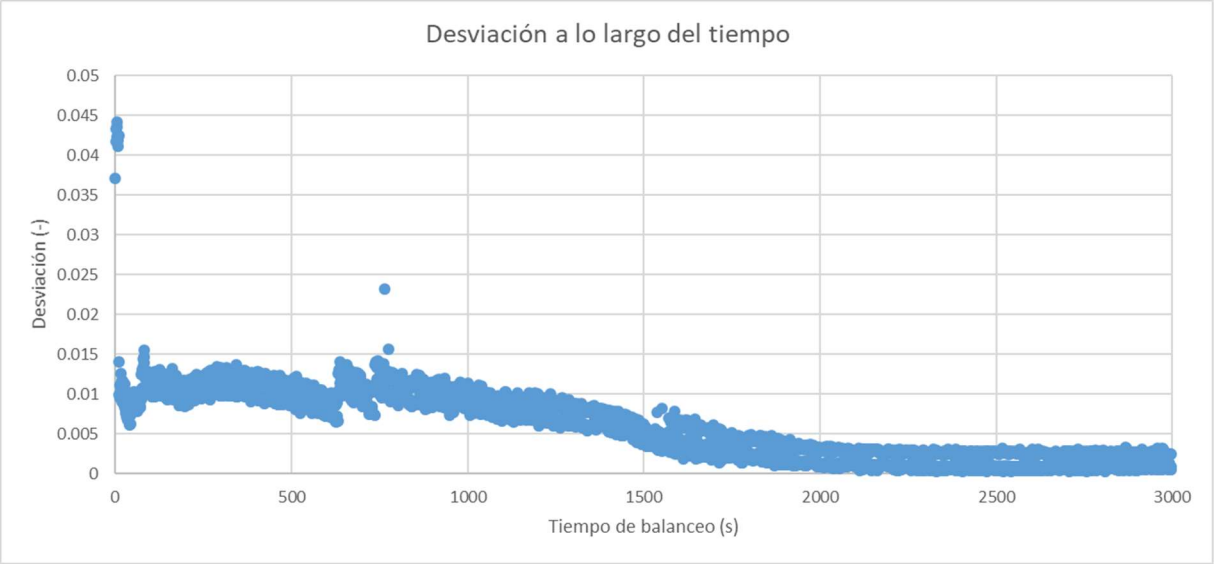


Ilustración 57. Desviación del voltaje en el balanceo de las celdas

Como se puede observar, la desviación se ha reducido significativamente. Inicialmente la batería mostraba una desviación de hasta 10 veces mayor que la que había tras finalizar la carga. Se puede ver también como ese balanceo resulta más eficaz en el comienzo de la carga que al final a causa de la curva de carga de las celdas; a menor tensión una diferencia de voltaje supone una diferencia de carga menor a la que habría en otro nivel de carga superior.

Medición de voltajes

La etapa de medición de voltajes debe ser calibrada para asegurar su correcto funcionamiento. El microcontrolador tiene una resolución de 10 bits en sus pines analógicos, esto es, valores entre 0 y 1025. Esta medición debe traducirse a voltaje de las celdas. Si la medición fuera perfecta bastaría con aplicar una regla de tres entre los valores de 0 y 1025 y los 0 y 5V.

Sin embargo, las medidas están sujetas a oscilaciones como el valor como el voltaje de alimentación del microcontrolador, imprecisiones en el valor de las resistencias de la etapa de medición, temperaturas, etc. A fin de tener en cuenta estas variaciones y obtener una medida lo más real posible se van a realizar una serie de mediciones que determinen la ecuación que relaciona el voltaje de las celdas con el valor obtenido en la entrada analógica.

Se va a utilizar 8 celdas conectadas en serie para simular una batería real. Cada una de las celdas será monitorizada por el BMS en sus distintos niveles de carga y se van a recoger los

datos obtenidos. Las celdas monitorizadas se colocan en serie a través de dos placas como la de la siguiente imagen. Esta placa ha sido diseñada para este tipo de tests:

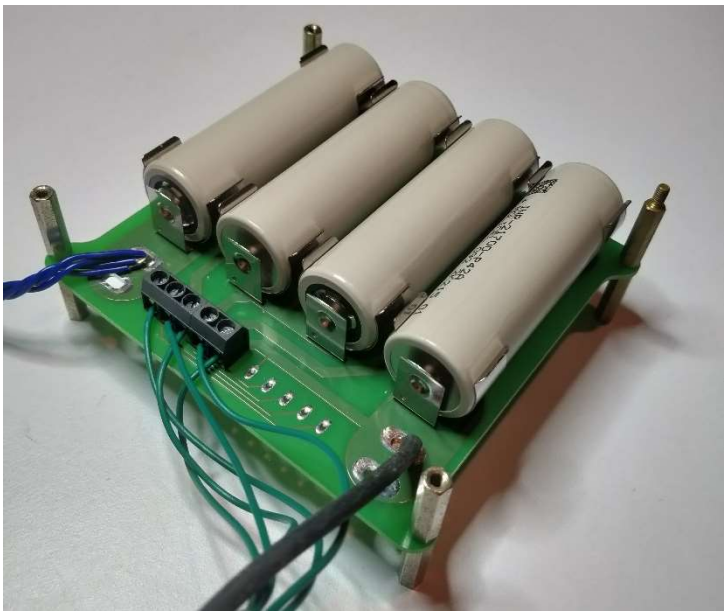


Ilustración 58. Placa de pruebas para las celdas

Para obtener la ecuación se va a comparar las medidas del microcontrolador con un multímetro que asegure una medida precisa. Los resultados son los siguientes:

	CELDA 1	CELDA 2	CELDA3	CELDA 4	CELDA 5	CELDA 6	CELDA 7	CELDA 8
Error (%)	-1.56%	-1.85%	-1.40%	-1.20%	-1.47%	-1.71%	-1.30%	-1.31%
Factor a multiplicar	1.0156	1.0185	1.0140	1.0120	1.0147	1.0171	1.0130	1.0131
Error medio (V)	0.003	0.007	0.003	0.005	0.004	0.005	0.005	0.006
Error medio total (V)	**0.0047**							

Tabla 5. Errores en la medición de voltajes.

En la primera fila se muestra el error medio medido en cada una de las combinaciones del esclavo. Aunque los esquemas de todas las entradas analógicas sean idénticas ofrecen resultados diferentes. Esto se debe a imprecisiones en elementos electrónicos como resistencias, longitud de pistas, etc.

Con el fin de minimizar el error de medición se pretende corregir la tendencia que tiene cada una de las mediciones. Para ello se multiplicará cada medida por una ganancia que contrarreste el error medio registrado en las pruebas. El factor por el que se multiplica cada medida es el que se muestra en la segunda fila.

Se vuelve a realizar las mismas pruebas aplicando el factor de corrección y se ha conseguido una precisión media de 0.0047V, o lo que es lo mismo, 4.7mV.

Esta calibración se va a realizar para cada una de las placas esclavas ya que no todas tendrán un comportamiento idéntico.

Medición de intensidades

Para obtener la ecuación que relaciona el valor analógico obtenido con la intensidad monitorizada se realiza un proceso similar al de la medición de voltajes. Se va a someter a las celdas a una descarga de distintas intensidades, medir estas intensidades con un multímetro y compararlas con el valor obtenido.

El sensor de intensidad se ha probado midiendo la intensidad que circula por una resistencia cuando se alimenta a unos 120V, ya que es el voltaje que puede alcanzar la motocicleta.

Para variar la intensidad que se quiere medir se pueden hacer dos cosas:

Variar la resistencia: Esto supondría tener tantas resistencias como valores intensidades se quiera medir.

Tener la misma resistencia siempre y pasar el cable varias veces por el sensor para multiplicar la corriente. Por ejemplo, si se hacen pasar 10A por el cable y se le dan 5 vueltas, el sensor de Hall medirá 50A.

La segunda opción resulta la más económica y sencilla y además no varía la precisión de la prueba, con lo que se decide hacer de esa forma. Los resultados son los siguientes:

Intensidad sensor (A)	Voltaje arduino (V)	Voltaje datasheet (V)	Error (%)
0	2.51	2.510	0.00%
8.64	2.54	2.537	0.12%
17.34	2.56	2.564	0.16%
26.07	2.59	2.591	0.06%
34.6	2.62	2.618	0.07%
43.4	2.64	2.646	0.21%
52.26	2.67	2.673	0.12%
60.69	2.7	2.700	0.01%
69.44	2.73	2.727	0.11%
78.21	2.75	2.754	0.16%
87.1	2.78	2.782	0.08%
121.66	2.89	2.890	0.01%

139.04	2.94	2.945	0.15%
156.6	3	2.999	0.02%
-156.24	2.02	2.022	0.09%
-130.65	2.1	2.102	0.08%
-104.16	2.18	2.185	0.21%
-77.67	2.26	2.267	0.32%
-52.02	2.35	2.347	0.11%
-25.98	2.43	2.429	0.05%
		Error medio (%)	**0.11%**

Tabla 6. Mediciones de intensidad

El error medio obtenido es del 0.11%. Suponiendo un rango de medida máximo de ±600A, el error máximo sería de 0.624A.

Medición de temperaturas

Para garantizar la precisión que el fabricante indica en su datasheet se van a contrastar las medidas de temperatura obtenidas con un termómetro preciso disponible en el taller. Se ha podido comprobar que las medidas eran correctas y se confía en la precisión a temperaturas más altas, ya que no se ha tenido los medios para probarlo en el taller.

Almacenamiento en micro SD

En esta prueba se van a muestrear las distintas variables y se van a tratar de guardar en la micro SD tal y como se ha estipulado en apartados anteriores. Se puede comprobar que los datos se han guardado correctamente de dos formas distintas:

- Se extrae la tarjeta micro SD de la placa y se introduce en el ordenador directamente mediante un lector de tarjetas micro SD.
- Se comunica por puerto serie el esclavo con el ordenador y el esclavo le envía los datos que tiene almacenados en la micro SD.

No obstante, no ha sido posible comprobar el funcionamiento y no se ha logrado saber por qué. Se sospecha que hay problemas en alguna de las soldaduras, especialmente en el conector de la micro SD, al ser el componente más difícil de soldar y menos accesible.

Se cree que el esquema de la micro SD está bien diseñado ya que se ha conectado un módulo idéntico de otro fabricante (pero que utiliza el mismo esquemático) puenteado en el arduino y ha funcionado correctamente.

Para futuras versiones habrá que tener en cuenta que resulta más sencillo conectar un módulo completo de cualquier fabricante que intentar integrarlo a la placa, debido a problemas en las soldaduras.

Pruebas de la placa maestra

Control de contactores

Se va a someter un contactor a diferentes activaciones y desactivaciones consecutivas. Para comprobar que funciona correctamente se va a medir la continuidad en los terminales del contactor y ver que coincide con los instantes en los que el contactor debe estar cerrado.

Las pruebas han concluido que el circuito funciona correctamente.

Almacenamiento en micro SD

En esta prueba se han encontrado los mismos problemas que con la placa esclava, con lo que no se ha podido testear la micro SD.

Comunicación bluetooth

Para saber que la comunicación bluetooth funciona correctamente se conecta el maestro con otro dispositivo que tenga tecnología bluetooth. Dos dispositivos muy sencillos de conectar son un teléfono móvil y un ordenador. Se va a realizar la prueba con los dos métodos para ver que no existe ningún problema.

Antes de establecer la comunicación es necesario configurar el módulo bluetooth. Sus parámetros configurables más importantes son el modo de funcionamiento (maestro o esclavo), contraseña, velocidad de transmisión y nombre del dispositivo. Para configurarlos se debe entrar en el modo de configuración o modo AT del bluetooth. Toda esta información se puede encontrar de forma fácil en internet.

Una vez configurado el sistema se establece conexión tanto con el móvil como con el ordenador. Se ha establecido la comunicación sin ningún problema, con lo que se da por válido el funcionamiento del bluetooth.

Comunicación serie con el ordenador

La comunicación serie se va a comprobar de la misma forma que se ha comprobado el bluetooth a través del ordenador. En este caso se utilizará la consola ofrecida por el arduino IDE. La comunicación a funcionado sin problemas.

Supervisión del IMD

Se va a forzar al IMD a dar valores de funcionamiento correcto e incorrecto en el sistema. Los cambios deberán observarse en la lectura del BMS. Efectivamente así ha sido. Cuando el IMD daba una señal de 10V al BMS le llegaba un HIGH en la entrada digital, y cuando el IMD estaba a masa le llegaba LOW.

Pruebas BMS al completo

Comunicación entre maestro-esclavos

Se ha configurado cada esclavo con una dirección diferente y se ha subido un sketch al maestro para que se comunique con los esclavos sin funciones adicionales, simplemente que muestre los datos leídos a través del puerto serie en el ordenador.

La comunicación se ha realizado con éxito con todos los esclavos simultáneamente.

FUTURAS MEJORAS

A causa de los problemas surgidos por el covid-19 u otros factores, el diseño tiene correcciones y mejoras a realizar. Se trata de un proyecto que no está acabado completamente, pero del trabajo que se ha realizado se han detectado los siguientes fallos:

- Placa esclava:
    - Se podría diseñar el esclavo sin el multiplexor en la etapa de balanceo y alimentar directamente a los optoacopladores desde las salidas digitales del arduino. Los pines del arduino serían suficientes para esta tarea y se ahorraría el multiplexor.
    - El condensador del filtro de paso bajo en la etapa de medición altera la medición que se quiera realizar. Aunque se haya escogido una frecuencia de corte superior al cambio de señal en el shift register no ha sido suficiente y se ha eliminado para el funcionamiento. Al quitar el condensador, también sobraría la resistencia de ese mismo filtro.
    - Añadir un algoritmo de balanceo que no solo tenga en cuenta las 8 celdas de la placa sino todas las que haya en la batería. Esto se podría hacer si el maestro le dijera a todos los esclavos el menor voltaje detectado y así los esclavos actuaran en consecuencia.
    - Se puede hacer un algoritmo más eficiente de enviar los datos de los esclavos al maestro en caso de no conectar todas las celdas. En vez de enviar siempre la misma longitud de bits con algunos datos vacíos se puede indicar en el

primer byte la longitud de la trama y en función de eso leer unas cosas u otras. O se puede especificar en el setup cuantas celdas, sensores, etc hay en cada esclavo para que sepa qué es lo que tiene que leer.

- En caso de que se conecten varios sensores Dallas en el esclavo no se mostrarían los resultados en la pantalla OLED, ya que está diseñada para un número específico de datos. Para solucionar esto se podría mejorar la tarea de forma adaptable, esto es, que mostrara los datos en el OLED independientemente del número de celdas y sensores que hubiera.
- La precisión en las mediciones analógicas depende del voltaje al que se esté alimentando el arduino. La precisión de 10 bits en las entradas analógicas se mide sobre ese voltaje de alimentación, esto es, el valor 0 corresponde a 0V y el valor 1023 corresponde a la alimentación. Oscilaciones en la tensión de alimentación genera perturbaciones proporcionales en la medida. Se podría buscar una fuente más estable para alimentar el arduino y así hacer más estable las mediciones de los pines analógicos.
- Añadir una línea i2c adicional en otros dos pines del arduino para conectar su pantalla. Con la disposición actual, como todas las pantallas tienen la misma dirección, al estar en el mismo bus de I2C se graficaría en todas las pantallas los mismos datos. Sería imposible tener cada pantalla con su interfaz.
- El footprint del OLED está mal tanto en el esclavo como en el maestro. Los pines VCC y GND están intercambiados, con lo que no se puede soldar directamente a la placa. Temporalmente se ha soldado la pantalla utilizando cables y así es como se ha probado su funcionamiento.
- Al poder conectar tantos sensores Dallas como se quieran existe la posibilidad de conectar tantos que el software no sea capaz de cumplir los plazos. Se podría hacer un cálculo que estime cuantos sensores se podrían conectar.

- Placa maestra:
  - Las resistencias de la parte de los termistores y del IMD se han colocado con un encapsulado SMD, pero habría sido mejor soldarlo con encapsulado THT. De esta forma, si se necesitara cambiar algún valor de resistencia se cambiaría de una forma mucho más rápida y sencilla que con SMD. También se podría haber optado por poner potenciómetros.

- Aún no se ha podido probar la supervisión de los contactores, pero se prevé un posible fallo en el diseño; Si el contactor que se supervisa estuviera abierto y el voltaje entre terminales fuera mayor a los 12V de la fuente de alimentación, este voltaje del contactor podría dañar a la fuente por hacer circular una corriente en sentido contrario. No se sabe realmente si ocurrirá esto, pero hay una posibilidad de que lo haga. Se comprobará cuando se realicen las pruebas restantes. Como alternativa se podría cambiar el diseño utilizando un sensor Hall que mida la corriente que circula por el contactor. Si midiera una corriente distinta a 0, significaría que el contactor está cerrado. Como contra está la situación en la que el contactor está cerrado, pero la carga no pide ninguna intensidad y, por lo tanto, este circuito lo mediría como contactor abierto.
- En el circuito de control de contactores se podría evitar uno de los leds, ya que hay 2 que cumplen la misma función.
- No pueden funcionar las comunicaciones CAN y el almacenamiento en micro SD simultáneamente porque se ha colocado el mismo pin CS para los dos casos. En las comunicaciones SPI los pines MOSI, MISO y CLK son comunes en todo el bus para todos los dispositivos. Sin embargo, el pin CS debe ser único para cada dispositivo. Esto es, al haber dos dispositivos que se comunican por SPI el arduino debería tener dos pines destinados en vez de 1 [27].
- La resistencia que limita la intensidad en la supervisión de contactores debería de ser 921Ω según los cálculos, pero se ha colocado una resistencia de 120Ω. Esto se debe a una errata a la hora de realizar el diseño.
- Se puede añadir un jumper en paralelo a cada resistencia del bus de comunicaciones I2C para cortocircuitarlas. Sería útil ya que tanto la placa esclava como la maestra tienen colocadas resistencias de pull-up. Con estos jumpers se podrían ignorar las resistencias que se considerasen oportunas.
- Se puede pulir más el código para mejorar la seguridad ante fallas, reducir tiempos de respuesta y añadir funcionalidades extra que no implicarían cambios en el hardware.

3. COSTES DEL PROYECTO

En este apartado se valora económicamente todos los elementos y acciones que han sido necesarias para la consecución del proyecto. Se muestran los costes y no el proyecto ya que se trata de un proyecto real que se ha llevado a cabo. A continuación se muestra una tabla en la que figura la inversión realizada desglosada por partidas. Este desglose de costes se encuentra detallado en el Anexo VII.

Nº Ref	Cant. (Ud)	Concepto	Precio unitario (€/ud.)	Precio total (€)
1		**SUBCONTRATACIÓN**		
1.1	1	Subcontratación fabricación PCBs	112.59 €	112.59 €
		TOTAL SUBCONTRATACIONES		**112.59 €**
2		**MATERIALES**		
2.1	4	Placa esclava	75.40 €	301.61 €
2.2	1	Placa maestra	75.06 €	75.06 €
		TOTAL MATERIALES		**376.67 €**
		COSTE TOTAL		**489.26 €**

Tabla 7. Costes del proyecto

Los costes se han desglosado en dos subgrupos:

- Subcontrataciones: Incluye el pedido que se hizo a la empresa alemana Multi Circuit Boards para la fabricación de las placas electrónicas. En este concepto está incluido tanto la fabricación como el envío.
- Materiales: Agrupa todos los componentes electrónicos del BMS, tanto del esclavo como del maestro. Todos los componentes han sido encargados a la empresa Mouser, a excepción de algunos, ya que no estaban disponibles o eran más económicos y se han conseguido a través de ebay.

No se han desglosado todos los costes en este apartado con el fin de simplificarlo. Todos los conceptos llevan ya incluidos los impuestos, con lo que no hay que sumarle el IVA y se trata del coste total.

Como se puede observar, el coste total del proyecto es de 425.71€. Comparándolo con BMS de características similares disponibles en el mercado, que pueden oscilar entre 700 y 1000€, supone un ahorro interesante para el equipo de MotoStudent Bizkaia. Más aún

cuando este BMS se podría reutilizar para futuras ediciones suponiendo este ahorro por cada edición que se compita.

La viabilidad económica del proyecto se debe a que las horas de ingeniería de desarrollo y ensamblaje del BMS son gratuitas por haber realizado el proyecto de forma voluntaria. De la misma forma, todas las herramientas necesarias como soldador, bomba de calor, horno, pasta de soldar… han sido prestadas por el equipo para llevar a cabo el proyecto. Esto ha supuesto un factor muy importante, ya que si se hubieran sumado todos estos conceptos como coste habría sido imposible realizar el proyecto, económicamente hablando. Esto tiene mucho sentido, ya que este BMS ha sido fabricado una única vez, a diferencia de los BMS comerciales que se fabrican en grandes cantidades y pueden sacar provecho de las economías de escala.

4. CONCLUSIONES

Diseñar y ensamblar un BMS al completo ha supuesto un auténtico reto a nivel personal. Ha sido necesaria mucha más investigación y trabajo del que se esperaba. Si se hubieran contabilizado las horas invertidas seguro que serían mucho mayores a las estipuladas por el plan de estudios (600h). Además, las circunstancias que se han vivido a causa del covid-19 no han facilitado la tarea, sino todo lo contrario. El cierre del taller durante meses ha obligado a buscar posibilidades para trabajar en el proyecto donde no las había.

Con el principal objetivo de terminar el proyecto en los plazos estipulados ha sido necesario llevar todo el material del taller a casa; horno, componentes electrónicos, placas, soldador, bomba de calor, etc. Se ha tenido que continuar sin tener todos los medios que se habrían tenido si nada hubiera ocurrido. Todo esto teniendo en cuenta que este proyecto se ha hecho de forma voluntaria, ya que las personas que participan en el proyecto de MotoStudent lo hacen sin esperar nada a cambio. A pesar de las circunstancias, se pueden sacar las siguientes conclusiones:

El objetivo principal era el diseño, ensamblaje y puesta a punto de un BMS para la motocicleta eléctrica de Bizkaia ESI Bilbao 2020. Se ha diseñado, ensamblado y quedan pocos pasos para terminar la puesta a punto del mismo. No se ha logrado cumplir el objetivo al 100%, pero por circunstancias ajenas al proyecto. Además, aunque el trabajo de fin de master se dé por terminado, el proyecto del BMS seguirá adelante hasta la fecha de la competición, que será en marzo de 2021. Por lo tanto, como el trabajo restante es mucho menor al que ya se ha realizado, se prevé terminar las pruebas de puesta a punto mucho antes de la competición. Se estima que pueda hacer falta 1 mes más para terminarlo por completo.

Los objetivos secundarios se han cumplido completamente. Se ha posibilitado la adquisición de datos de los parámetros de funcionamiento del BMS y la batería de la motocicleta a través de comunicaciones bluetooth, CAN y serie. Es cierto que no se ha logrado la adquisición a través de la micro SD, pero no supone ningún problema grave ya que simplemente era una función adicional del BMS.

Se ha conseguido comunicar el BMS con distintos elementos de la motocicleta a través del bluetooth y el CAN. El puerto serie no se ha utilizado para tal fin ya que ningún componente de la motocicleta dispone de ese protocolo.

Se ha diseñado el software para PC que permita extraer los datos adquiridos posibilitando su posterior análisis. Como estos datos se pueden obtener a tiempo real se cumple también el último objetivo secundario.

El coste total del proyecto ha sido menor a un BMS comercial de las mismas características y solamente se ha producido 1 unidad, siendo esto factor limitante contra las economías de escala. Si surgiera la necesidad de fabricar muchas más unidades, los costes del proyecto se reducirían.

En conclusión, el trabajo propuesto se ha cumplido a pesar de todas las circunstancias. Cierto es que existen mejoras a realizar, pero esto no supondrá ningún problema como ya se ha comentado en el apartado “Futuras mejoras”. El proyecto seguirá adelante a pesar de que este TFM se dé por concluido, ya sea por el autor del proyecto como por las futuras generaciones del equipo MotoStudent Bizkaia.

5. BIBLIOGRAFÍA

[1]. MotoStudent. *TechnoPark MotorLand*. Obtenido de Moto Engineering Foundation: http://www.motostudent.com/about-motostudent.html

[2]. Miranda, C. V. (2018). *Diseño de un BMS para baterías de tecnología Li-ion.* Barcelona: Universitat Oberta de Catalunya.

[3]. Scott, K. *Analog Devices*. Obtenido de https://www.analog.com/en/technical-articles/active-battery-cell-balancing.html]

[4]. Illera, G. H. (2014). Gestor de carga de baterías. Burgos: Escuela técnica superior Universidad de Burgos.

[5]. LLamas, L. (19 de noviembre de 2016). Obtenido de https://www.luisllamas.es/mas-salidas-y-entradas-en-arduino-con-multiplexor-cd74hc4067

[6]. *Medir corriente eléctrica con un sensor de efecto Hall*. Obtenido de https://cuningan.wordpress.com/2011/07/05/medir-corriente-electrica-con-un-sensor-de-efecto-hall/

[7]. MotoStudent. (2019). *Reglamento de la competición.*

[8]. SRC. SRC. Obtenido de http://www.ledoelectronics.com/Projects/current_hall_sensor.pdf

[9]. Llamas, L. (27 de Junio de 2016). *MEDIR TEMPERATURA DE LÍQUIDOS Y GASES CON ARDUINO Y DS18B20*. Obtenido de https://www.luisllamas.es/temperatura-liquidos-arduino-ds18b20/

[10]. Llamas, L. (18 de Mayo de 2016). *EL BUS SPI EN ARDUINO*. Obtenido de https://www.luisllamas.es/arduino-spi/

[11]. LLamas, L. (2018 de Mayo de 2016). *EL BUS I2C EN ARDUINO*. Obtenido de https://www.luisllamas.es/arduino-i2c/

[12]. *ARDUINO Y LOS SHIFT REGISTERS*. Obtenido de https://www.prometec.net/shift-registers/

[13]. Instruments, T. (2015). *SNx4HC595 8-Bit Shift Registers With 3-State Output Registers.*

[14]. Gago, G. E. (2017). *Diseño y desarrollo del BMS de un monoplaza de Formula Student Electric.* Sevilla: Universidad de Sevilla.

[15]. ELECTRONIK, W. (2019). *WL-SMRW SMT Mono-color Reverse mount Waterclear.*

[16]. Toshiba. (2014). *TLP293-4.* Toshiba Electronic Devices & Storage Corporation.

[17]. Texas Instruments. (2016). *MUX50x.*

[18]. Analog Devices. *AD628.*

[19]. Sillicon Labs. *Si860x Data Sheet.*

[20]. Gago, G. E. (2017). *Diseño y desarrollo del BMS de un monoplaza de Formula Student Electric.* Sevilla: Universidad de Sevilla.

[21]. Vishay. (2014). *Optocoupler, Phototransistor Output, with Base Connection CNY17.*

[22]. *Seeed*. Obtenido de https://wiki.seeedstudio.com/CAN-BUS_Shield_V1.2/

[23]. González, V. (9 de Septiembre de 2019). *Diagnosis Tips*. Obtenido de https://www.diagnosistips.com/can-bus/

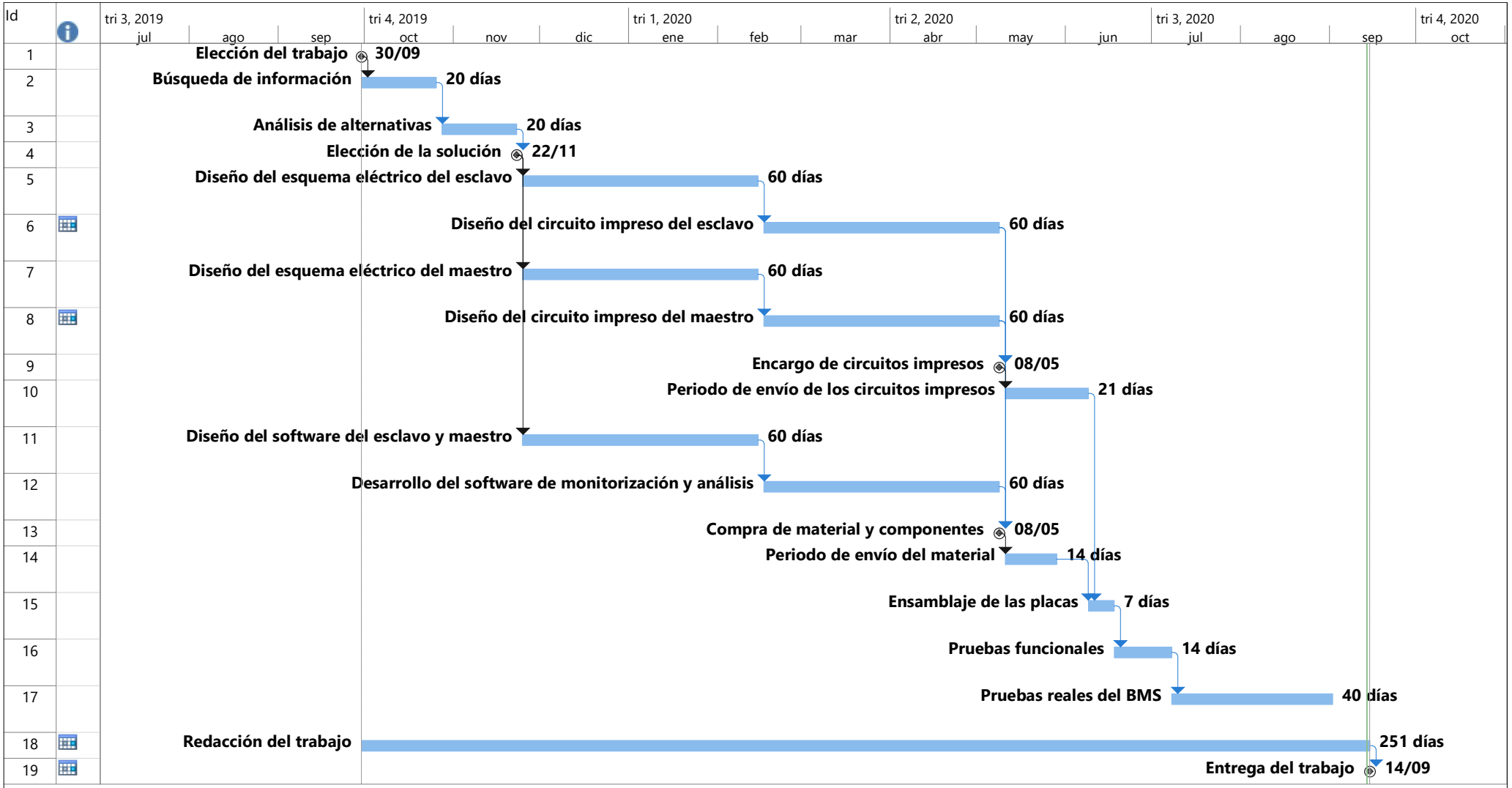
[24]. Llamas, L. (4 de Noviembre de 2016). *CONECTAR ARDUINO A UNA PANTALLA OLED DE 0.96"*. Obtenido de https://www.luisllamas.es/conectar-arduino-a-una-pantalla-oled-de-0-96/

[25]. Llamas, L. (10 de Abril de 2015). *MEDIR TEMPERATURA CON ARDUINO Y TERMISTOR (MF52)*. Obtenido de https://www.luisllamas.es/medir-temperatura-con-arduino-y-termistor-mf52/

[26]. Maxim. (2019). *MAX680/MAX681.*

[27]. circuits, L. *How to Connect Multiple SPI devices to an Arduino Microcontroller*. Obtenido de http://www.learningaboutelectronics.com/Articles/Multiple-SPI-devices-to-an-arduino-microcontroller.php

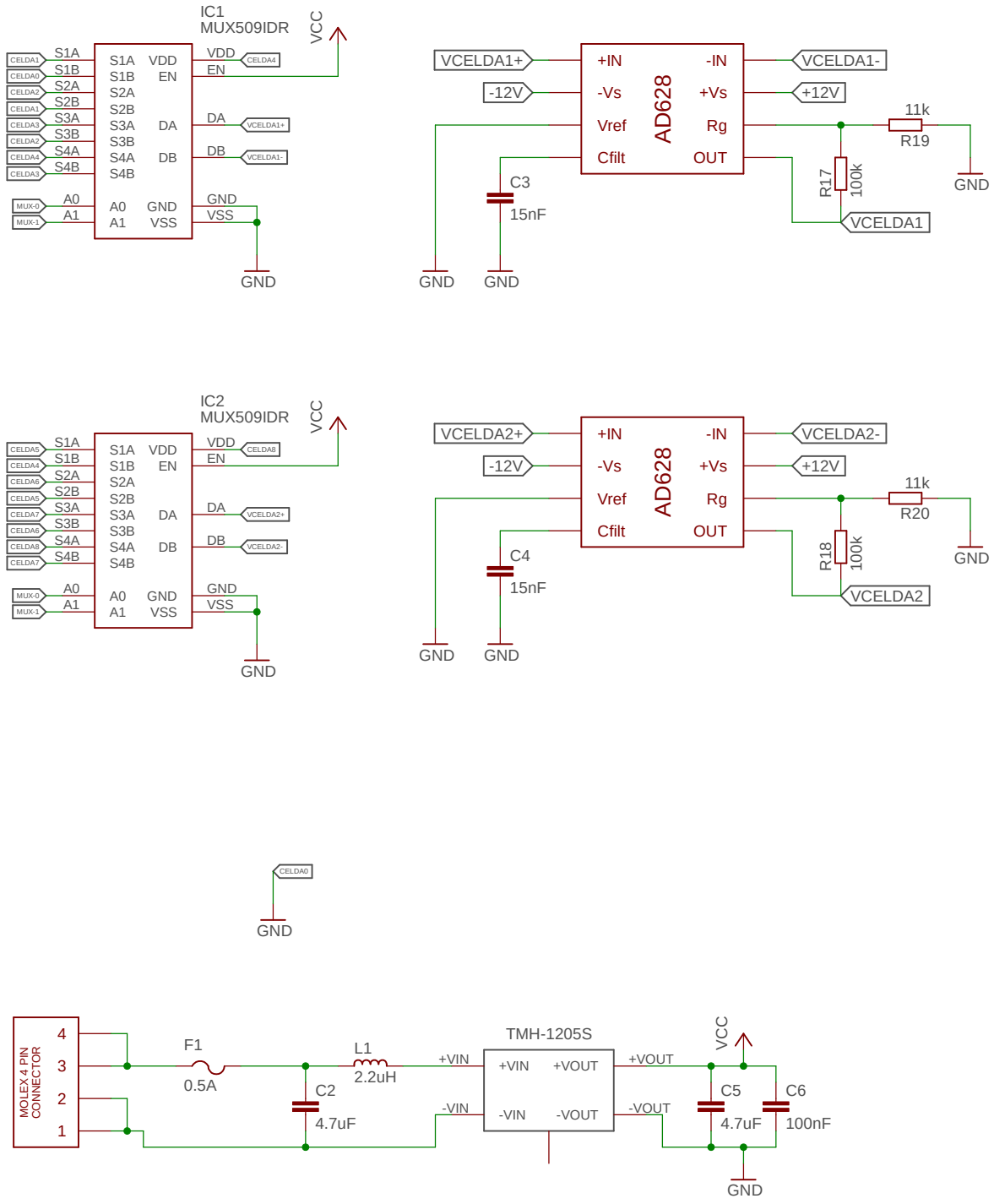
6. ANEXO I: Diagrama de Gantt



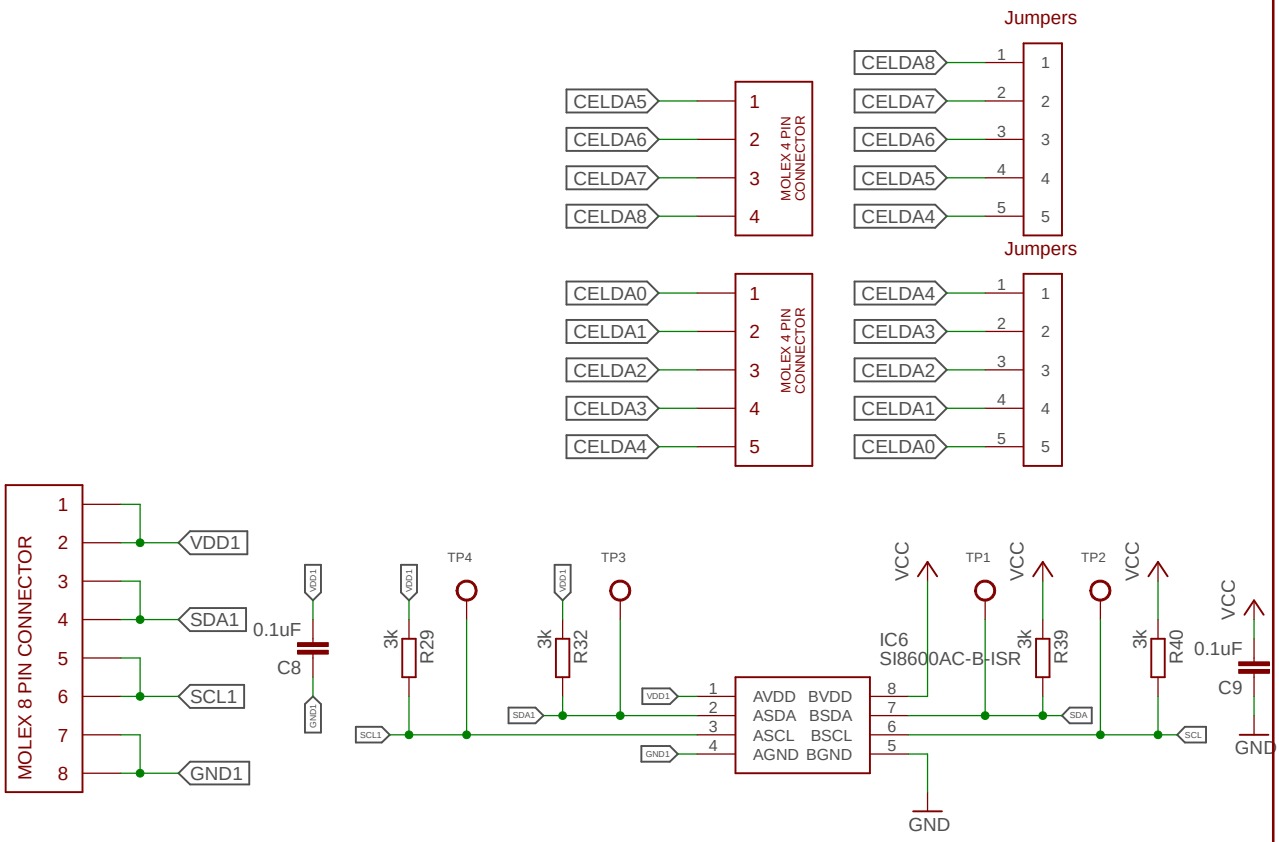
Proyecto: Gantt TFG
Fecha: sáb 04/04/20

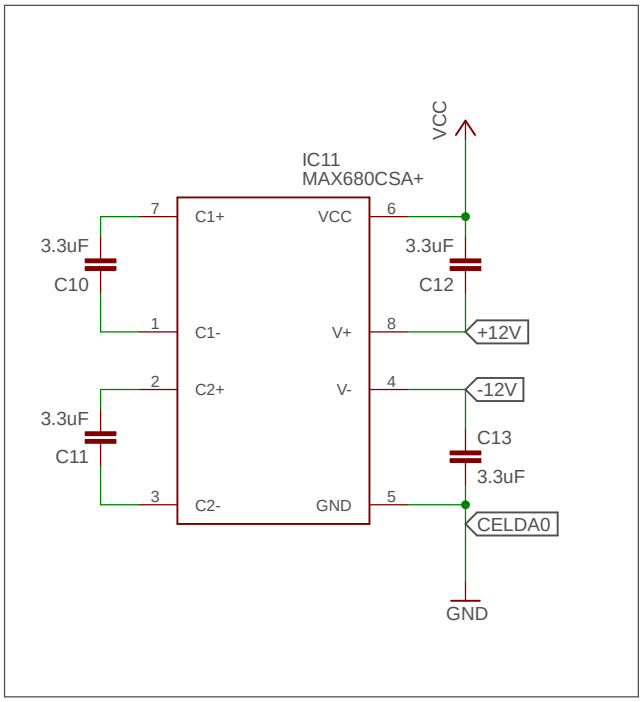
Tarea	Tarea inactiva	Informe de resumen manual	Hito externo
División	Hito inactivo	Resumen manual	Fecha límite
Hito	Resumen inactivo	solo el comienzo	Progreso
Resumen	Tarea manual	solo fin	Progreso manual
Resumen del proyecto	solo duración	Tareas externas	

7. ANEXO II: Esquema eléctrico de la placa esclava

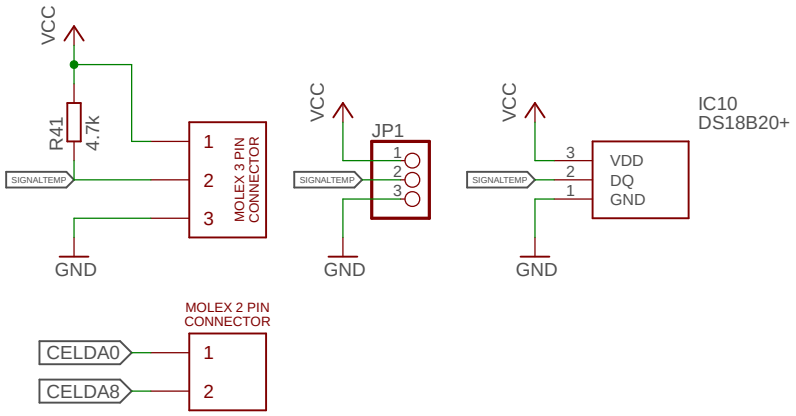


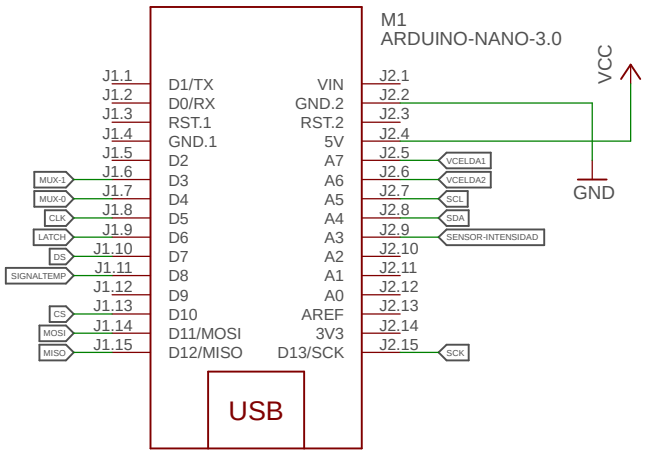
TITLE: BMS	
Document Number:	REV:
Date: 12/09/2020 16:56	Sheet: 1/1

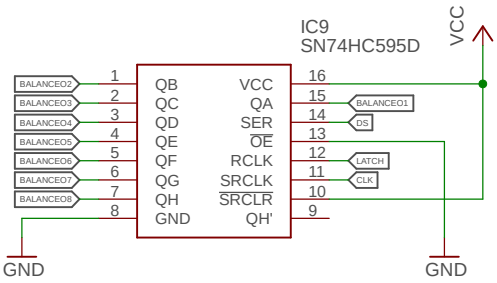




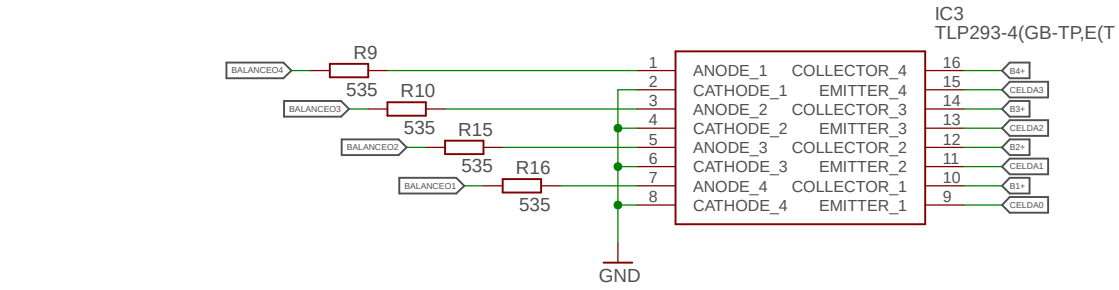
TITLE: BMS	
Document Number:	REV:
Date: 12/09/2020 16:56	Sheet: 1/1

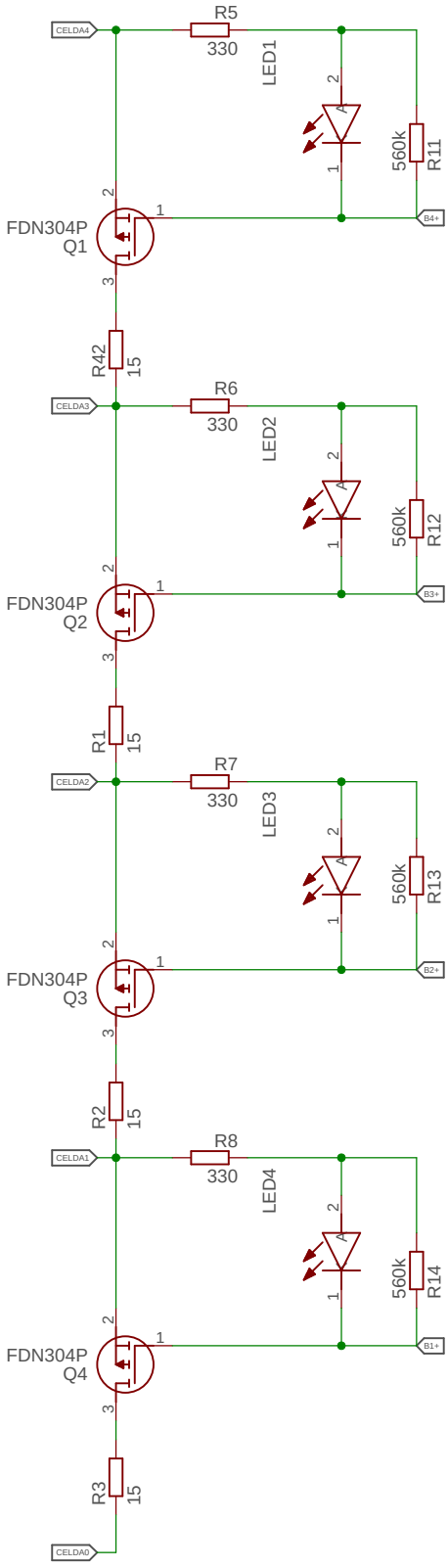






TITLE: BMS		
Document Number:		REV:
Date: 12/09/2020 16:56	Sheet: 1/1	



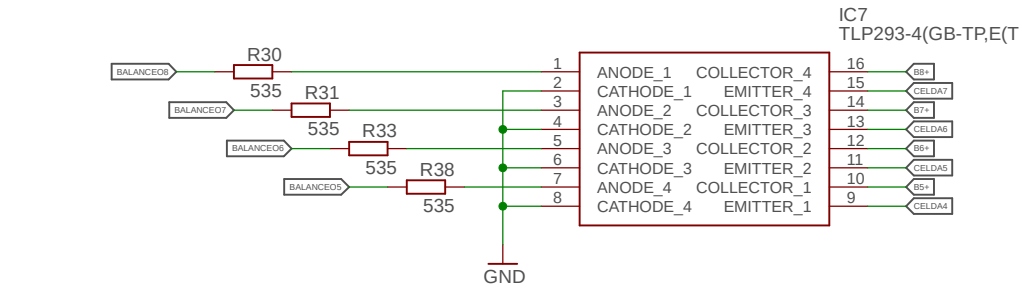


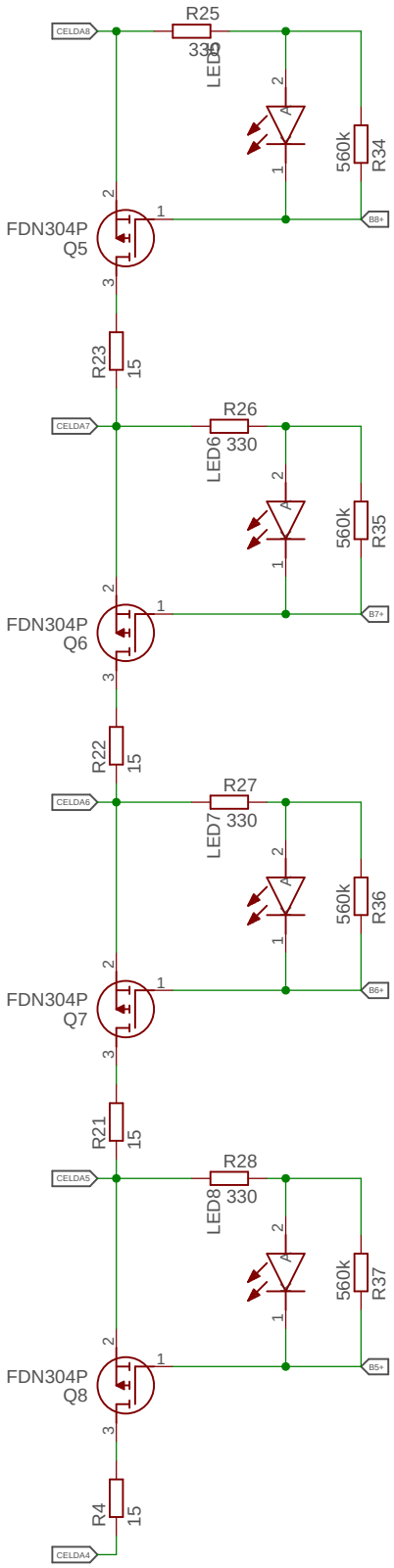
4.2V-1.9V(LED)=2.3V
2.3V/330ohm=7mA por el Led

Necesito que vayan al menos 7mA por el otro
lado del opto para cumplir el 100% de current transfer ratio minimo que me da el opto

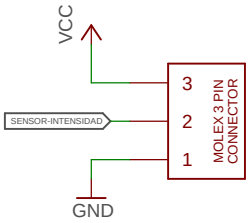
El opto chupa 1.25V con 7mA
5V-1.25V=3.75V tiene que caer en la R
3.75V/0.007A=535ohm

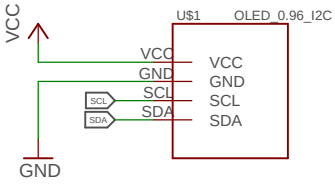
TITLE: BMS		
Document Number:		REV:
Date: 12/09/2020 16:56	Sheet: 1/1	

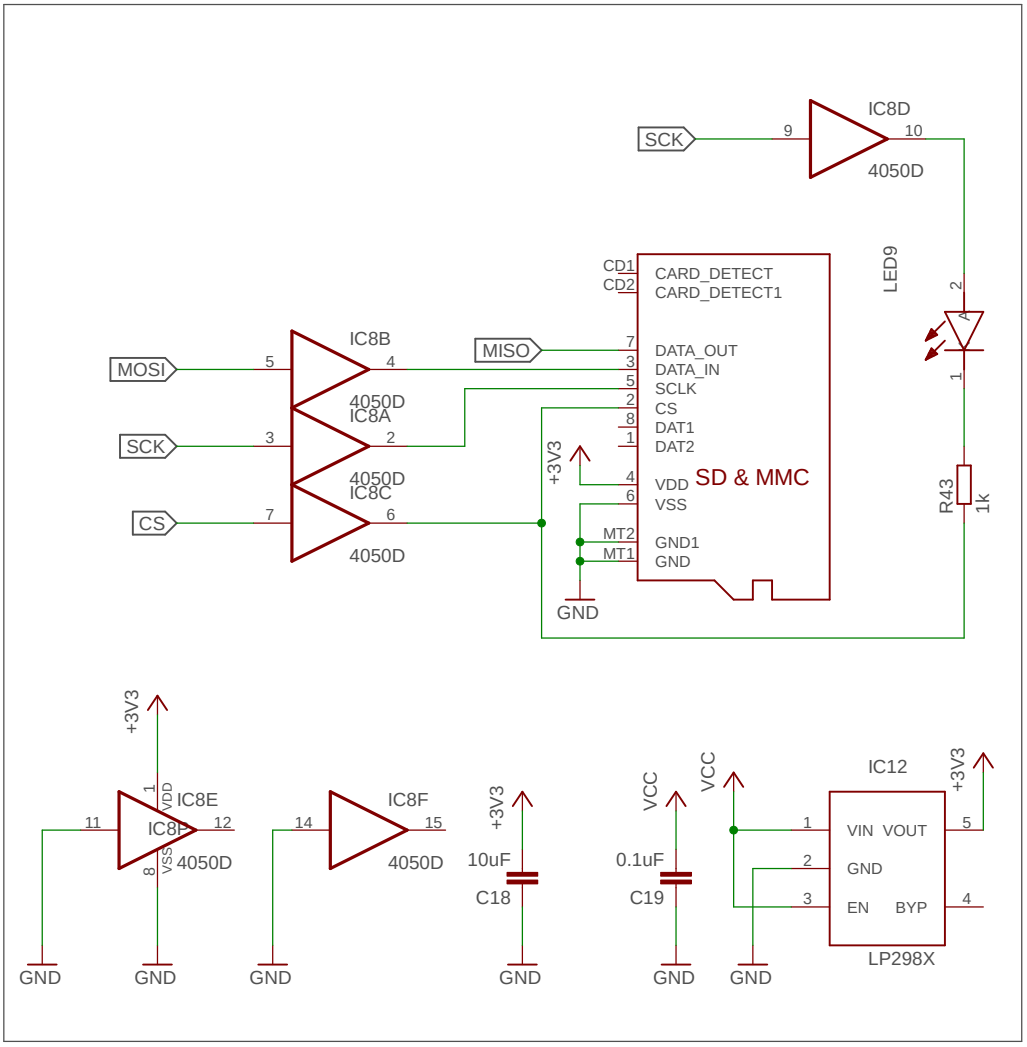




TITLE: BMS		
Document Number:		REV:
Date: 12/09/2020 16:56	Sheet: 1/1	



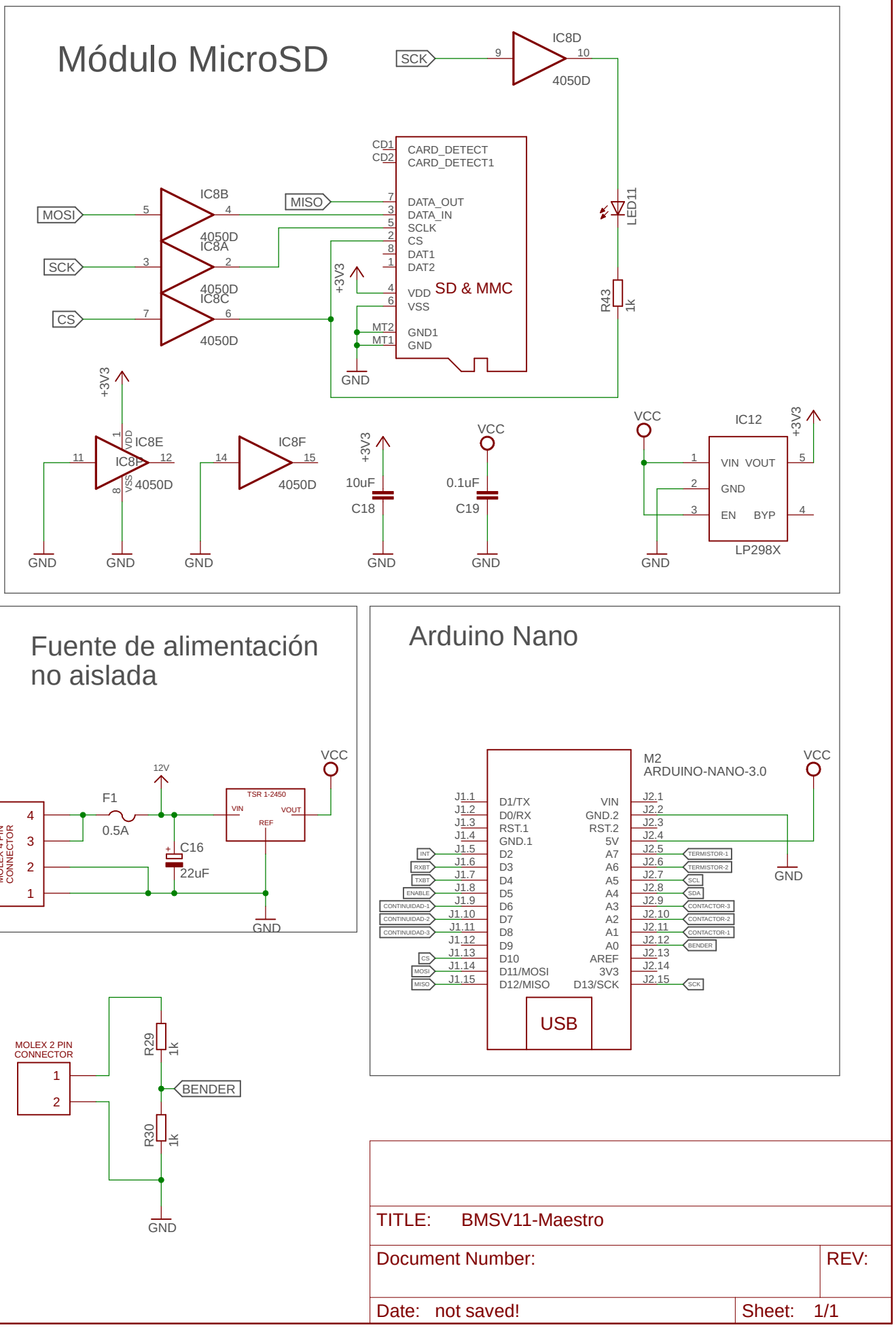




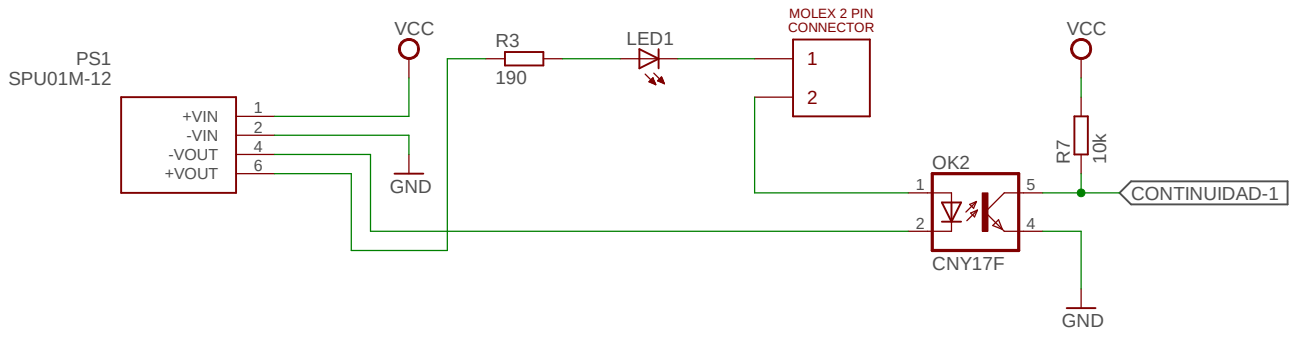
Módulo MicroSD

TITLE: BMS	
Document Number:	REV:
Date: 12/09/2020 16:56	Sheet: 1/1

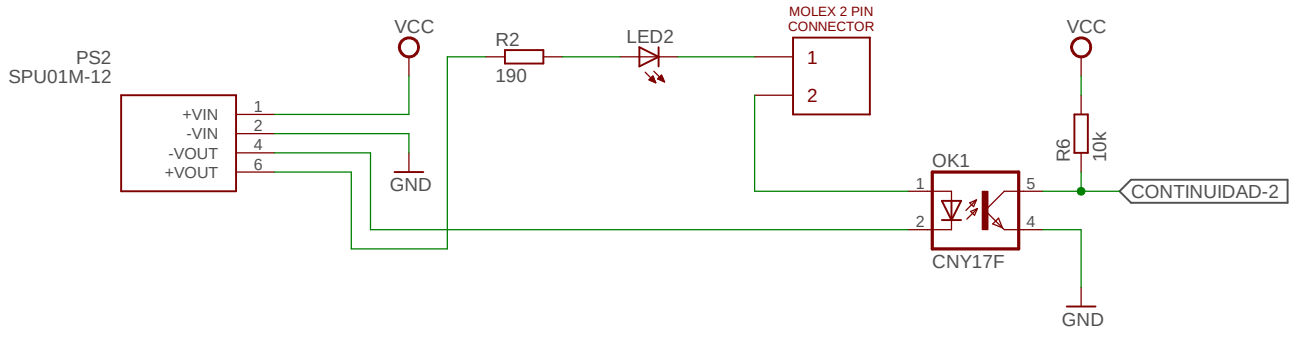
8. ANEXO III: Esquema eléctrico de la placa maestra



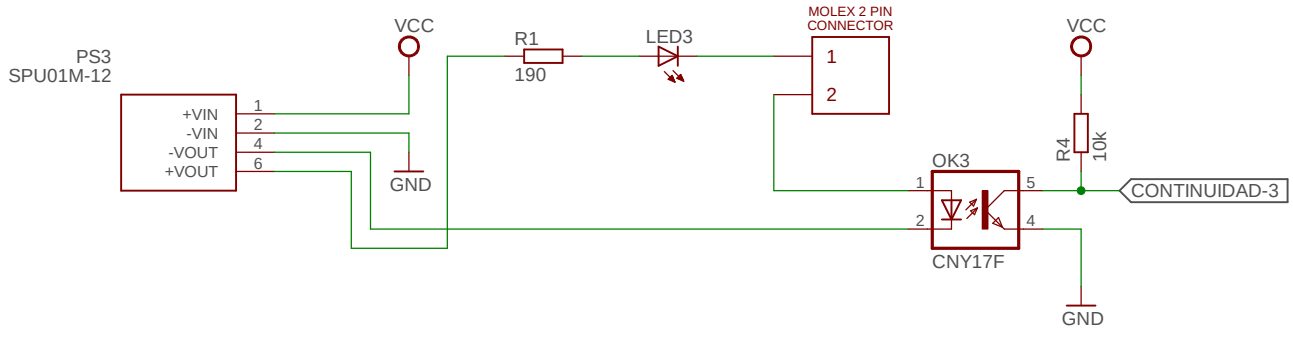
Control de continuidad de contactor nº1



Control de continuidad de contactor nº2

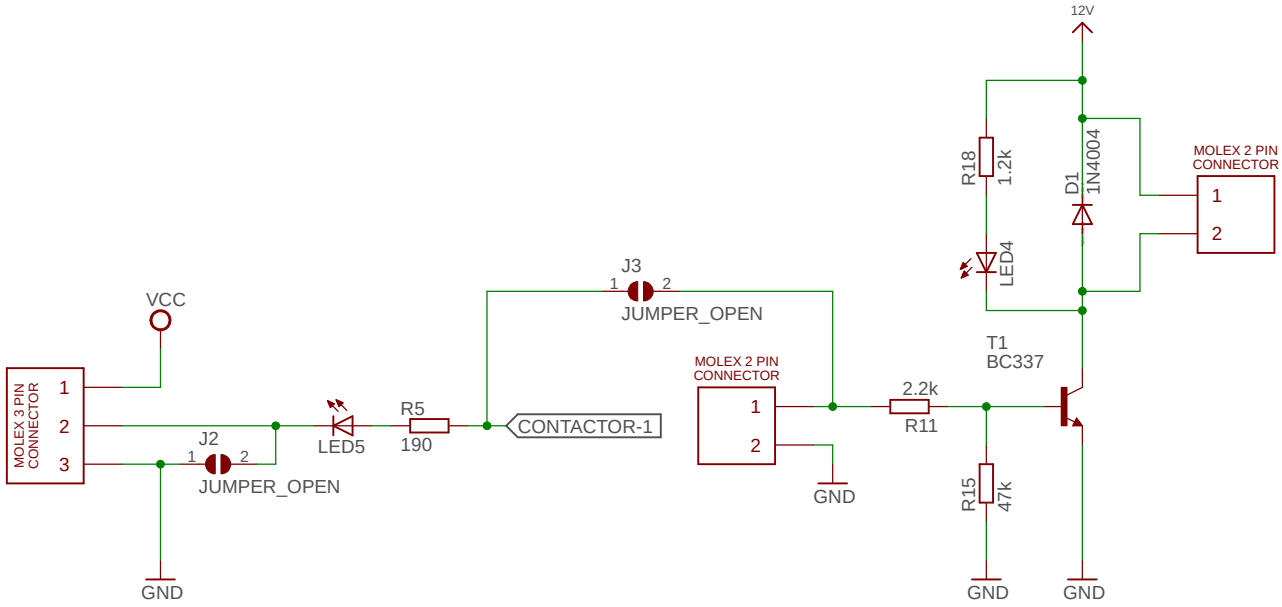


Control de continuidad de contactor nº3

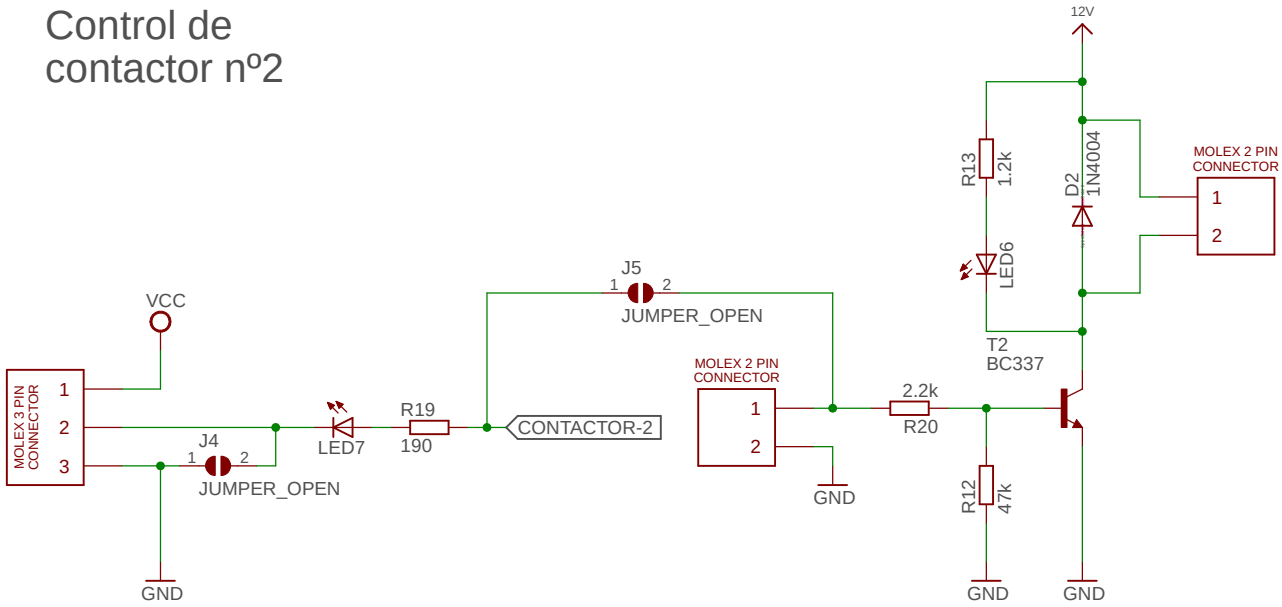


TITLE: BMSV11-Maestro		
Document Number:		REV:
Date: not saved!	Sheet: 1/1	

Control de contactor nº1

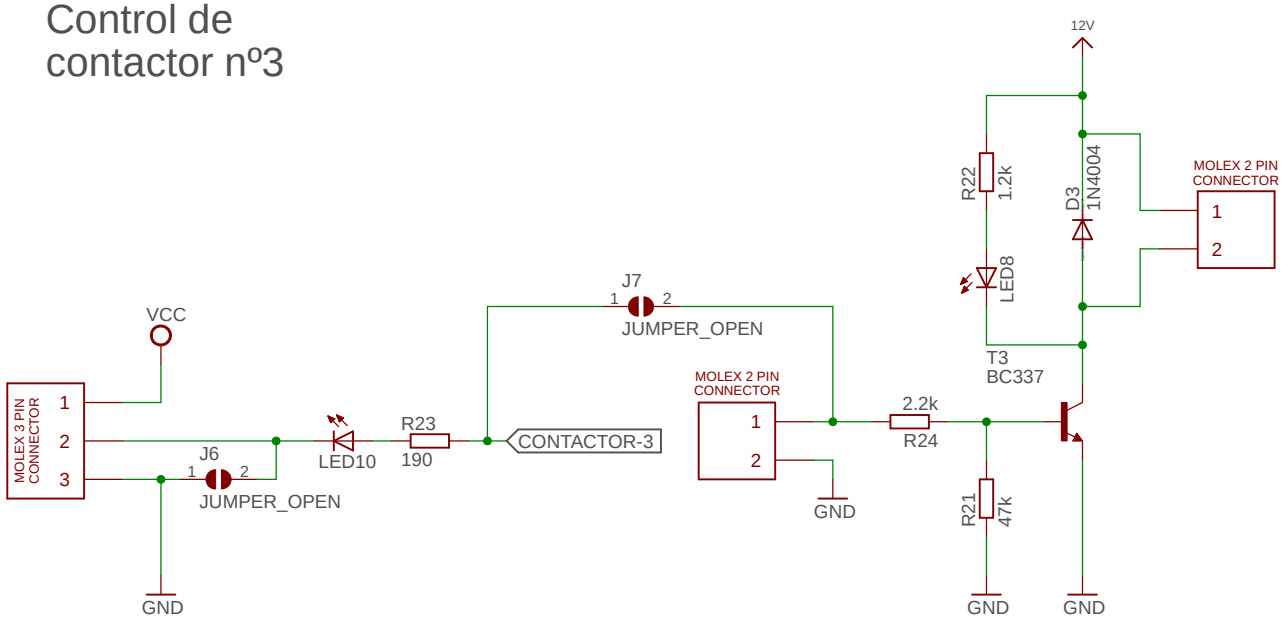


Control de contactor nº2

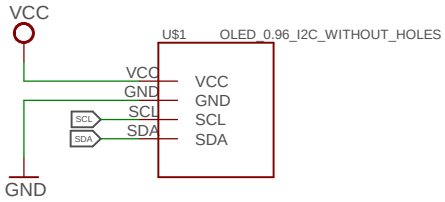


TITLE: BMSV11-Maestro	
Document Number:	REV:
Date: not saved!	Sheet: 1/1

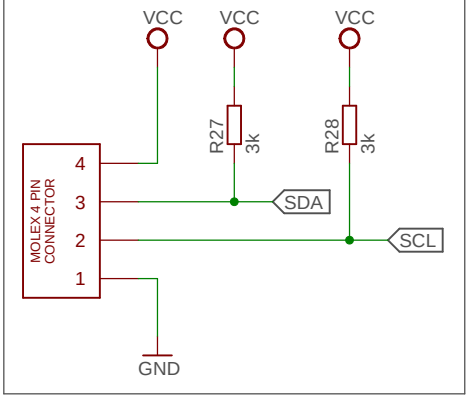
Control de contactor nº3



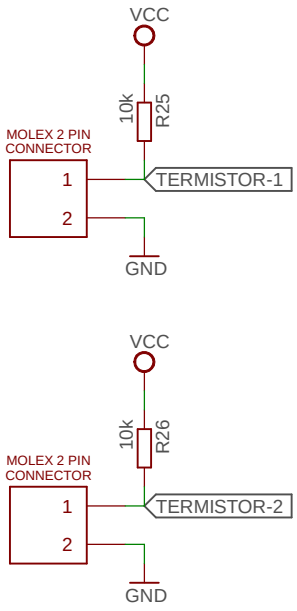
Pantalla OLED



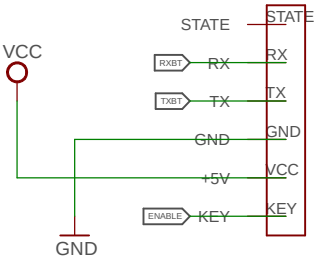
Comunicaciones I2C



Termistores

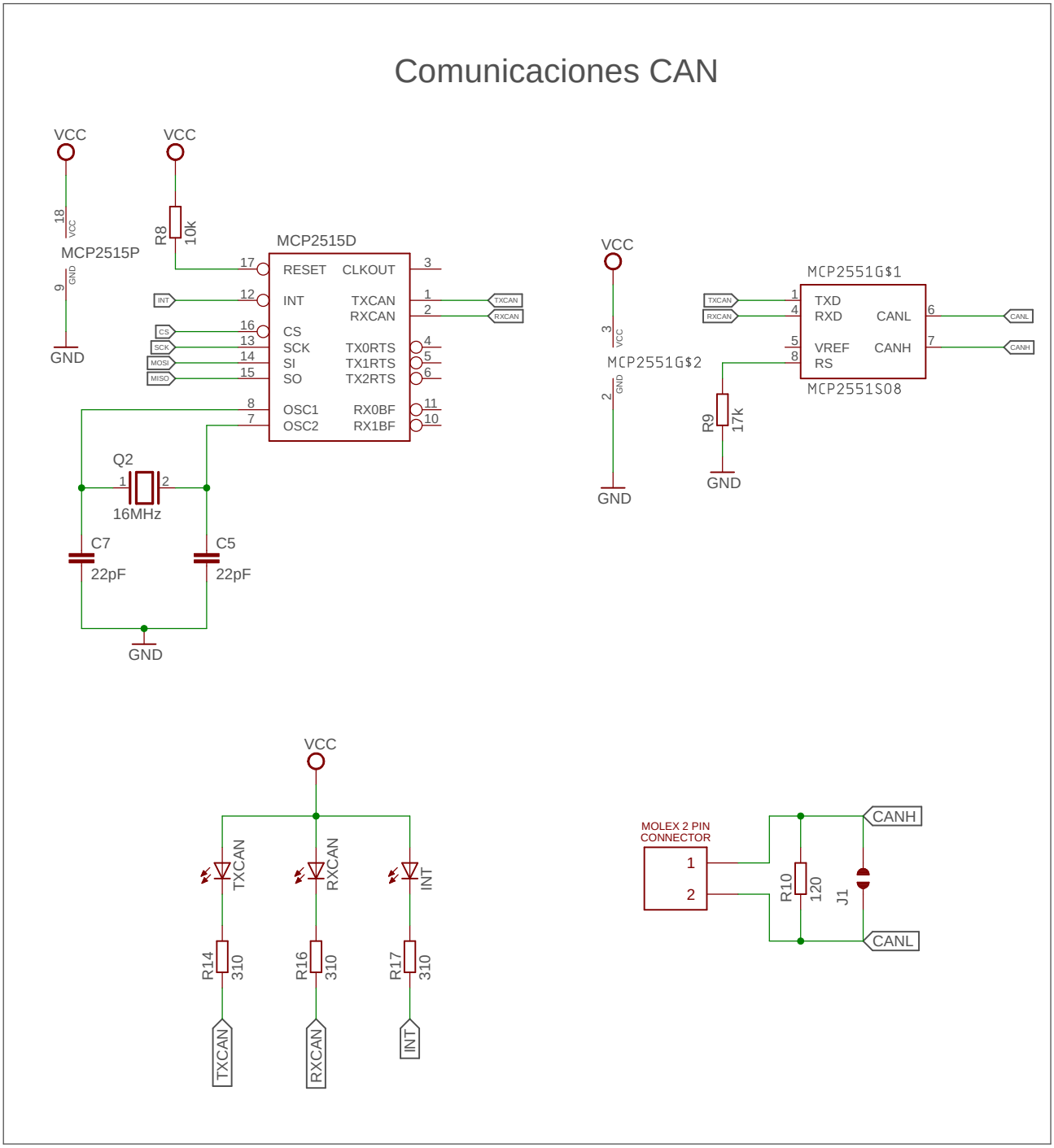


Módulo Bluetooth



TITLE: BMSV11-Maestro		
Document Number:		REV:
Date: not saved!	Sheet: 1/1	

Comunicaciones CAN



TITLE: BMSV11-Maestro	
Document Number:	REV:
Date: not saved!	Sheet: 1/1

9. ANEXO IV: Software de la placa esclava

```
//  Para el RTOS
#include <Arduino_FreeRTOS.h>
//  Para el Dallas
#include <OneWire.h>
#include <DallasTemperature.h>
//  Para la micro SD
#include <SPI.h>
#include <SD.h>
//  I2C
#include <Wire.h>

/*-----------------------------------------------------*/
/*---------------------- DALLAS-----------------------*/
/*-----------------------------------------------------*/
const int oneWirePin = 8;
float tempC = 0.0;

OneWire oneWireBus(oneWirePin);
DallasTemperature sensors(&oneWireBus);

DeviceAddress sensores[] = {{ 0x28, 0xAA, 0xC4, 0x28, 0x3D, 0x14,
0x01, 0x9B },
};

/*-----------------------------------------------------*/
/*-------------------- Voltajes----------------------*/
/*-----------------------------------------------------*/
const int PinVoltajeCelda1 = A6;
const int PinVoltajeCelda2 = A7;

const int mux0 = 4;
const int mux1 = 3;

//  Array que indica en qué posiciones monitorizo celdas. 1 si hay
celda y 0 si no lo hay
float aCeldaInsertada[8] = {1, 1, 1, 1, 1, 1, 1, 1};
//  Array de datos que contiene primero los voltajes, luego
intensidad, luego temperaturas y después bit de fallo
int aVoltajes[8] = {1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024};

// el clock y el latch estan al reves en el esquema
const int latchPin = 6 ;        //Pin conectado a ST_CP of 74HC595
const int clockPin = 5;        //Pin conectado a SH_CP of 74HC595
const int dataPin = 7;          //Pin connected to DS of 74HC595

int iVoltajeMin = 1024;
int iPosicionBalanceo = 0;
byte byteBalanceo = 255;

/*-----------------------------------------------------*/
/*-------------------- Intensidad-------------------*/
/*-----------------------------------------------------*/
int intensidad = 0;
const int PinIntensidad = A3;

/*-----------------------------------------------------*/
```

```
/*-------------------- Micro SD----------------------*/
/*---------------------------------------------------*/
const int chipSelect = 10;


/*---------------------------------------------------*/
/*-------------------- Tareas-----------------------*/
/*---------------------------------------------------*/
void MedicionVoltajes(void *param);
void MedicionTemperaturas(void *param);
void MedicionIntensidad(void *param);
void MicroSD(void *param);
void Balanceo(void *param);

TaskHandle_t Task_Handle_1;
TaskHandle_t Task_Handle_2;
TaskHandle_t Task_Handle_3;
TaskHandle_t Task_Handle_4;
TaskHandle_t Task_Handle_5;

void setup()
{
  Serial.begin(9600);

  // Unimos este dispositivo al bus I2C con dirección 1
  Wire.begin(1);

  // Registramos el evento al recibir datos
  Wire.onReceive(receiveEvent);

  xTaskCreate(MedicionVoltajes, "MedicionVoltajes", 100, NULL, 3,
&Task_Handle_1);
  xTaskCreate(MedicionTemperaturas, "MedicionTemperaturas", 100, NULL,
1, &Task_Handle_2);
  xTaskCreate(MedicionIntensidad, "MedicionIntensidad", 100, NULL, 3,
&Task_Handle_3);
  xTaskCreate(MicroSD, "MicroSD", 100, NULL, 1, &Task_Handle_4);
  xTaskCreate(Balanceo, "Balanceo", 100, NULL, 1, &Task_Handle_5);
}

//  Aunque no usemos el loop hay que añadirlo

void loop()
{
}

/*---------------------------------------------------*/
/*---------------------- Tasks ----------------------*/
/*---------------------------------------------------*/
void MedicionVoltajes(void *pvParameters)
{
  (void) pvParameters;

  TickType_t getTick;
  getTick = xTaskGetTickCount();

  pinMode(mux0, OUTPUT);
  pinMode(mux1, OUTPUT);

  while (1)
```

```
  {
    for (byte i = 0; i < 4; i++)
    {
      SetMuxChannel(i);

      for (int j = 1; j <= 10; j++)
      {
        if (aCeldaInsertada[i])
        {
          aVoltajes[i] = aVoltajes[i] +
float(analogRead(PinVoltajeCelda1));
        }
        else
        {
          aVoltajes[i] = 0.0;
        }

        if (aCeldaInsertada[i + 4])
        {
          aVoltajes[i + 4] = aVoltajes[i + 4] +
float(analogRead(PinVoltajeCelda2));
        }
        else
        {
          aVoltajes[i + 4] = 0.0;
        }
      }
      aVoltajes[i] = aVoltajes[i] * 5.0 / (1024.0 * 10.0);
      aVoltajes[i + 4] = aVoltajes[i + 4] * 5.0 / (1024.0 * 10.0);
    }

    for (byte i = 0; i < 8; i++)
    {
      Serial.print("   V");
      Serial.print(i + 1);
      Serial.print(":");
      Serial.print(aVoltajes[i]);
    }
    Serial.println();

    vTaskDelayUntil(&getTick, 40 / portTICK_PERIOD_MS);
  }
}

int SetMuxChannel(byte channel)
{
  digitalWrite(mux0, bitRead(channel, 0));
  digitalWrite(mux1, bitRead(channel, 1));
}

/*-----------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*-----------------------------------------------------*/
void MedicionTemperaturas(void *pvParameters)
{
  (void) pvParameters;

  TickType_t getTick;
  getTick = xTaskGetTickCount();
```

```
  sensors.begin();
  for (int i = 0; i < sizeof (sensores) / sizeof (sensores[0]); i++)
  {
    sensors.setResolution(sensores[i], 10);
  }

  while (1)
  {
    Serial.println("Leyendo temperaturas");

    for (int i = 0; i < sizeof (sensores) / sizeof (sensores[0]); i++)
    {
      for (int j = 0; j <= i; j++)
      {
        sensors.requestTemperatures();
        //      Serial.print("Temperatura sensor ");
        //      Serial.print(i+1);
        //      Serial.print(": ");
        tempC = sensors.getTempC(sensores[j]);
        if (tempC == -127.00)
        {
          Serial.print("Error getting temperature");
        }
        else
        {
          //    Serial.print(tempC);
          //    Serial.println(" °C");
        }
      }
    }
    vTaskDelayUntil(&getTick, 1000 / portTICK_PERIOD_MS);
  }
}

/*-----------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*-----------------------------------------------------*/
void MedicionIntensidad(void *pvParameters)
{
  (void) pvParameters;

  TickType_t getTick;
  getTick = xTaskGetTickCount();

  pinMode(PinIntensidad, OUTPUT);

  while (1)
  {
    intensidad = analogRead(PinIntensidad);
    vTaskDelayUntil(&getTick, 40 / portTICK_PERIOD_MS);
  }
}

/*-----------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*-----------------------------------------------------*/
void MicroSD(void *pvParameters)
{
  (void) pvParameters;
```

```
  TickType_t getTick;
  getTick = xTaskGetTickCount();

  while (!Serial)
  {
    ; // wait for serial port to connect. Needed for native USB port
only
  }
  Serial.print("Initializing SD card...");
  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect))
  {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    return;
  }
  Serial.println("card initialized.");

  while (1)
  {
    // make a string for assembling the data to log:
    String dataString = "";


    // open the file. note that only one file can be open at a time,
    // so you have to close this one before opening another.
    File dataFile = SD.open("datalog.txt", FILE_WRITE);

    // if the file is available, write to it:
    if (dataFile)
    {
      for (int j = 0; j < 8; j++)
      {
        dataString = dataString + String(aVoltajes[j]) + ",";
      }
      dataString = dataString + String(intensidad) + ",";
      dataString = dataString + String(tempC);

      dataFile.println(dataString);
      dataFile.close();
      // print to the serial port too:
      Serial.println(dataString);
    }
    // if the file isn't open, pop up an error:
    else
    {
      Serial.println("error opening datalog.txt");
    }
    vTaskDelayUntil(&getTick, 1000 / portTICK_PERIOD_MS);
  }
}

/*-----------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*-----------------------------------------------------*/
void Balanceo(void *pvParameters)
{
  (void) pvParameters;

  TickType_t getTick;
```

```
  getTick = xTaskGetTickCount();

  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  while (1)
  {
    byteBalanceo = 255;
    // aqui hay que hacer la conversión a 4.2V por seguridad
    iVoltajeMin = 1024;

    for (byte i = 0; i < 8; i++)
    {
      if (aVoltajes[i] < iVoltajeMin)
      {
        iPosicionBalanceo = i;
        iVoltajeMin = aVoltajes[i];
      }
    }

    bitClear(byteBalanceo, iPosicionBalanceo);
    digitalWrite(latchPin, LOW) ; // Latch a LOW para que no varíe la
salida
    shiftOut(dataPin, clockPin, LSBFIRST, byteBalanceo);  // Aqui va
Num
    digitalWrite(latchPin, HIGH) ; // Latch a HIGH fija valores en la
salida

    Serial.print("Byte balanceo: ");
    Serial.println(byteBalanceo);
    Serial.print("");

    vTaskDelayUntil(&getTick, 1000 / portTICK_PERIOD_MS);
  }
}

/*-----------------------------------------------------*/
/*---------------------- Tasks ----------------------*/
/*-----------------------------------------------------*/
// Función que se ejecuta siempre que se reciben datos del master
// siempre que en el master se ejecute la sentencia endTransmission
// recibirá toda la información que hayamos pasado a través de la
sentencia Wire.write
void receiveEvent(int howMany)
{
  for (byte i = 0; i < 8; i++)
  {
    // Enviamos un byte
    Wire.write(aVoltajes[i]);
  }
  Wire.write(intensidad);
  Wire.write(int(tempC));
  // Paramos la transmisión
  Wire.endTransmission();
}
```

10. ANEXO V: Software de la placa maestra

```
//  Para el RTOS
#include <Arduino_FreeRTOS.h>
//  Para el CAN
#include <mcp_can.h>
//  Para la micro SD
#include <SPI.h>
#include <SD.h>
//  I2C
#include <Wire.h>
//  Bluetooth
#include <SoftwareSerial.h>
//  Termistor
#include <math.h>
//  OLED
//#define __DEBUG__
//#include <Adafruit_GFX.h>
//#include <Adafruit_SSD1306.h>

/*-----------------------------------------------------*/
/*---------------------- TEMPS-----------------------*/
/*-----------------------------------------------------*/
float tempC = 0.0;

/*-----------------------------------------------------*/
/*---------------- CONTACTORES----------------------*/
/*-----------------------------------------------------*/
const int PinContactor1 = A1;
const int PinContactor2 = A2;
const int PinContactor3 = A3;
const int SupervisionContactor1 = 6;
const int SupervisionContactor2 = 7;
const int SupervisionContactor3 = 8;
boolean Contactor1 = false;
boolean Contactor2 = false;
boolean Contactor3 = false;
boolean Contactor1Supervisado = false;
boolean Contactor2Supervisado = false;
boolean Contactor3Supervisado = false;

/*-----------------------------------------------------*/
/*---------------------- TEMPS-----------------------*/
/*-----------------------------------------------------*/
int aTemps[6] = {0, 0, 0, 0};
const int PinTermistor1 = A7;
const int PinTermistor2 = A6;

/*-----------------------------------------------------*/
/*------------------- Voltajes---------------------*/
/*-----------------------------------------------------*/
//  Array de datos que contiene primero los voltajes, luego
intensidad, luego temperaturas y después bit de fallo
int aVoltajes[32] = {1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024};

/*-----------------------------------------------------*/
/*------------------- Intensidad-------------------*/
/*-----------------------------------------------------*/
```

```
int aIntensidad[4] = {0, 0, 0, 0};

/*-----------------------------------------------------*/
/*-------------------- SOC----------------------------*/
/*-----------------------------------------------------*/
float capacidadBateria=70.0;
float SOC=1.0;

/*-----------------------------------------------------*/
/*-------------------- Micro SD-----------------------*/
/*-----------------------------------------------------*/
const int chipSelect = 10;

/*-----------------------------------------------------*/
/*-------------------- IMD----------------------------*/
/*-----------------------------------------------------*/
const int PinIMD = A0;

/*-----------------------------------------------------*/
/*-------------------- CAN----------------------------*/
/*-----------------------------------------------------*/
const int SPI_CS_PIN = 10;

MCP_CAN CAN(SPI_CS_PIN);

/*-----------------------------------------------------*/
/*---------------- BLUETOOTH--------------------------*/
/*-----------------------------------------------------*/
SoftwareSerial BT1(3, 4);


/*-----------------------------------------------------*/
/*---------------- OLED-------------------------------*/
/*-----------------------------------------------------*/
//// Definir constantes
//#define ANCHO_PANTALLA 128 // ancho pantalla OLED
//#define ALTO_PANTALLA 64 // alto pantalla OLED
//
//// Objeto de la clase Adafruit_SSD1306
//Adafruit_SSD1306 display(ANCHO_PANTALLA, ALTO_PANTALLA, &Wire, -1);

/*-----------------------------------------------------*/
/*-------------------- Tareas-------------------------*/
/*-----------------------------------------------------*/
void Contactores(void *param);
void Termistores(void *param);
void SupervisionContactores(void *param);
void MicroSD(void *param);
void IMD(void *param);
void I2C(void *param);
void Comunicaciones(void *param);
//void OLED(void *param);
void soc(void *param);

TaskHandle_t Task_Handle_1;
TaskHandle_t Task_Handle_2;
TaskHandle_t Task_Handle_3;
TaskHandle_t Task_Handle_4;
TaskHandle_t Task_Handle_5;
TaskHandle_t Task_Handle_6;
```

```
TaskHandle_t Task_Handle_7;
//TaskHandle_t Task_Handle_8;
TaskHandle_t Task_Handle_9;

void setup()
{
  Serial.begin(9600);

  xTaskCreate(Contactores, "Contactores", 100, NULL, 3,
&Task_Handle_1);
  xTaskCreate(Termistores, "Termistores", 100, NULL, 1,
&Task_Handle_2);
  xTaskCreate(SupervisionContactores, "SupervisionContactores", 100,
NULL, 3, &Task_Handle_3);
  xTaskCreate(MicroSD, "MicroSD", 100, NULL, 1, &Task_Handle_4);
  xTaskCreate(IMD, "IMD", 100, NULL, 3, &Task_Handle_5);
  xTaskCreate(I2C, "I2C", 100, NULL, 2, &Task_Handle_6);
  xTaskCreate(Comunicaciones, "Comunicaciones", 100, NULL, 2,
&Task_Handle_7);
  //xTaskCreate(OLED, "OLED", 100, NULL, 1, &Task_Handle_8);
  xTaskCreate(soc, "soc", 100, NULL, 1, &Task_Handle_9);
}

//  Aunque no usemos el loop hay que añadirlo
void loop()
{
}

/*-----------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*-----------------------------------------------------*/
void Contactores(void *pvParameters)
{
  (void) pvParameters;

  TickType_t getTick;
  getTick = xTaskGetTickCount();

  pinMode(PinContactor1, OUTPUT);
  pinMode(PinContactor2, OUTPUT);
  pinMode(PinContactor3, OUTPUT);

  while (1)
  {
    if (Contactor1)
    {
      digitalWrite(PinContactor1, HIGH);
    }
    else
    {
      digitalWrite(PinContactor1, LOW);
    }

    if (Contactor2)
    {
      digitalWrite(PinContactor2, HIGH);
    }
    else
    {
      digitalWrite(PinContactor2, LOW);
```

```
    }

    if (Contactor3)
    {
      digitalWrite(PinContactor3, HIGH);
    }
    else
    {
      digitalWrite(PinContactor3, LOW);
    }

    vTaskDelayUntil(&getTick, 50 / portTICK_PERIOD_MS);
  }
}

/*-----------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*-----------------------------------------------------*/
void SupervisionContactores(void *pvParameters)
{
  (void) pvParameters;

  TickType_t getTick;
  getTick = xTaskGetTickCount();

  pinMode(SupervisionContactor1, INPUT);
  pinMode(SupervisionContactor2, INPUT);
  pinMode(SupervisionContactor3, INPUT);

  while (1)
  {
    if (!digitalRead(SupervisionContactor1) && Contactor1Supervisado)
    {
      Contactor1 = false;
    }
    else if (digitalRead(SupervisionContactor1) &&
Contactor1Supervisado)
    {
      Contactor1 = true;
    }
    else if (digitalRead(SupervisionContactor1) &&
!Contactor1Supervisado)
    {
      Contactor1 = false;
    }
    else if (!digitalRead(SupervisionContactor1) &&
!Contactor1Supervisado)
    {
      Contactor1 = true;
    }

    if (!digitalRead(SupervisionContactor2) && Contactor2Supervisado)
    {
      Contactor2 = false;
    }
    else if (digitalRead(SupervisionContactor2) &&
Contactor2Supervisado)
    {
      Contactor2 = true;
    }
```

```
    else if (digitalRead(SupervisionContactor2) &&
!Contactor2Supervisado)
    {
      Contactor2 = false;
    }
    else if (!digitalRead(SupervisionContactor2) &&
!Contactor2Supervisado)
    {
      Contactor2 = true;
    }

    if (!digitalRead(SupervisionContactor3) && Contactor3Supervisado)
    {
      Contactor3 = false;
    }
    else if (digitalRead(SupervisionContactor3) &&
Contactor3Supervisado)
    {
      Contactor3 = true;
    }
    else if (digitalRead(SupervisionContactor3) &&
!Contactor3Supervisado)
    {
      Contactor3 = false;
    }
    else if (!digitalRead(SupervisionContactor3) &&
!Contactor3Supervisado)
    {
      Contactor3 = true;
    }

    vTaskDelayUntil(&getTick, 50 / portTICK_PERIOD_MS);
  }
}

/*-----------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*-----------------------------------------------------*/
void IMD(void *pvParameters)
{
  (void) pvParameters;

  TickType_t getTick;
  getTick = xTaskGetTickCount();

  pinMode(PinIMD, INPUT);

  while (1)
  {
    if (digitalRead(PinIMD))
    {
      Contactor1 = true;
      Contactor2 = true;
      Contactor3 = true;
    }
    else
    {
      Contactor1 = false;
      Contactor2 = false;
      Contactor3 = false;
```

```
    }

    vTaskDelayUntil(&getTick, 50 / portTICK_PERIOD_MS);
  }
}

/*--------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*--------------------------------------------------*/
void Comunicaciones(void *pvParameters)
{
  (void) pvParameters;

  TickType_t getTick;
  getTick = xTaskGetTickCount();

  while (CAN_OK != CAN.begin(CAN_500KBPS))              // init can
bus : baudrate = 500k
  {
    Serial.println("CAN BUS Shield init fail");
    Serial.println(" Init CAN BUS Shield again");
    delay(100);
  }
  Serial.println("CAN BUS Shield init ok!");

  //inicializacion Bluetooth
  BT1.begin(38400); //Inicia la comunicacion Bluetooth

  unsigned char CanMSG[8] = {0, 0, 0, 0, 0, 0, 0, 0};

  int ciclos = 0;
  String dataString = "";

  while (1)
  {
    if (sizeof(aVoltajes) / 8 == 0)
    {
      ciclos = sizeof(aVoltajes) / 8;
    }
    else
    {
      ciclos = trunc(sizeof(aVoltajes) / 8) + 1;
    }
    for (int j = 0; j < ciclos; j++)
    {
      for (int i = 0; i < 8; i++)
      {
        dataString = dataString + String(aVoltajes[(j + 1) * 8 + i]) +
",";
        CanMSG[i] = aVoltajes[(j + 1) * 8 + i];
      }
      CAN.sendMsgBuf(0x00, 0, 8, CanMSG);
    }

    if (sizeof(aIntensidad) / 8 == 0)
    {
      ciclos = sizeof(aIntensidad) / 8;
    }
    else
    {
```

```
      ciclos = trunc(sizeof(aIntensidad) / 8) + 1;
    }
    for (int j = 0; j < ciclos; j++)
    {
      for (int i = 0; i < 8; i++)
      {
        dataString = dataString + String(aIntensidad[(j + 1) * 8 + i])
+ ",";
        CanMSG[i] = aIntensidad[(j + 1) * 8 + i];
      }
      CAN.sendMsgBuf(0x00, 0, 8, CanMSG);
    }

    if (sizeof(aTemps) / 8 == 0)
    {
      ciclos = sizeof(aTemps) / 8;
    }
    else
    {
      ciclos = trunc(sizeof(aTemps) / 8) + 1;
    }
    for (int j = 0; j < ciclos; j++)
    {
      for (int i = 0; i < 8; i++)
      {
        dataString = dataString + String(aTemps[(j + 1) * 8 + i]) +
",";
        CanMSG[i] = aTemps[(j + 1) * 8 + i];
      }
      CAN.sendMsgBuf(0x00, 0, 8, CanMSG);
    }

    BT1.println(dataString);
    Serial.println(dataString);

    vTaskDelayUntil(&getTick, 200 / portTICK_PERIOD_MS);
  }
}

/*-----------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*-----------------------------------------------------*/
void Termistores(void *pvParameters)
{
  (void) pvParameters;

  TickType_t getTick;
  getTick = xTaskGetTickCount();

  const int Rc = 10000; //valor de la resistencia
  const int Vcc = 5;
  const int SensorPIN = A0;
  const float A = 1.11492089e-3;
  const float B = 2.372075385e-4;
  const float C = 6.954079529e-8;
  float K = 2.5; //factor de disipacion en mW/C

  float raw = 0.0;
  float V =  0.0;
  float R = 0.0;
```

```
  float logR  = 0.0;
  float R_th = 0.0;
  float kelvin = 0.0;

  while (1)
  {

    raw = analogRead(PinTermistor1);
    V =  raw / 1024 * Vcc;
    R = (Rc * V ) / (Vcc - V);
    logR  = log(R);
    R_th = 1.0 / (A + B * logR + C * logR * logR * logR );
    kelvin = R_th - V * V / (K * R) * 1000;
    aTemps[5] = round(kelvin - 273.15);

    raw = analogRead(PinTermistor2);
    V =  raw / 1024 * Vcc;
    R = (Rc * V ) / (Vcc - V);
    logR  = log(R);
    R_th = 1.0 / (A + B * logR + C * logR * logR * logR );
    kelvin = R_th - V * V / (K * R) * 1000;
    aTemps[6] = round(kelvin - 273.15);

    vTaskDelayUntil(&getTick, 1000 / portTICK_PERIOD_MS);
  }
}

/*-----------------------------------------------------*/
/*---------------------- Tasks ----------------------*/
/*-----------------------------------------------------*/
//void OLED(void *pvParameters)
//{
//  (void) pvParameters;
//
//  TickType_t getTick;
//  getTick = xTaskGetTickCount();
//
//  #ifdef __DEBUG__
//    Serial.begin(9600);
//    delay(100);
//    Serial.println("Iniciando pantalla OLED");
//  #endif
//
//  // Iniciar pantalla OLED en la dirección 0x3C
//  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
//  {
//    #ifdef __DEBUG__
//        Serial.println("No se encuentra la pantalla OLED");
//    #endif
//    while (true);
//  }
//
//  while (1)
//  {
//    // Limpiar buffer
//    display.clearDisplay();
//
//    // Tamaño del texto
//    display.setTextSize(1);
//    // Color del texto
```

```
//    display.setTextColor(SSD1306_WHITE);
//    for (int j = 0; j < 2; j++)
//    {
//      for (int i = 0; i < 4; i++)
//      {
//        display.setCursor(j * 64, i * 10);
//        display.print("V");
//        display.print((i + 1) * (j + 1));
//        display.print(": ");
//      }
//    }
//    display.drawLine(0, 40, 128, 40, SSD1306_WHITE);
//
//    display.setCursor(0, 43);
//    display.print("I:");
//    display.setCursor(64, 43);
//    display.print("T:");
//
//    display.drawLine(0, 51, 128, 51, SSD1306_WHITE);
//
//    display.setCursor(0, 52);
//    display.print("Esclavo 1");
//
//    display.drawLine(60, 0, 60, 64, SSD1306_WHITE);
//
//    // Enviar a pantalla
//    display.display();
//    vTaskDelayUntil(&getTick, 40 / portTICK_PERIOD_MS);
//  }
//}
/*-----------------------------------------------------*/
/*---------------------- Tasks ----------------------*/
/*-----------------------------------------------------*/
void MicroSD(void *pvParameters)
{
  (void) pvParameters;

  TickType_t getTick;
  getTick = xTaskGetTickCount();

  while (!Serial)
  {
    ; // wait for serial port to connect. Needed for native USB port
only
  }
  Serial.print("Initializing SD card...");
  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect))
  {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    return;
  }
  Serial.println("card initialized.");

  while (1)
  {
    // make a string for assembling the data to log:
    String dataString = "";
```

```
    // open the file. note that only one file can be open at a time,
    // so you have to close this one before opening another.
    File dataFile = SD.open("datalog.txt", FILE_WRITE);

    // if the file is available, write to it:
    if (dataFile)
    {
      for (int j = 0; j < sizeof(aVoltajes); j++)
      {
        dataString = dataString + String(aVoltajes[j]) + ",";
      }
      for (int j = 0; j < sizeof(aIntensidad); j++)
      {
        dataString = dataString + String(aIntensidad[j]) + ",";
      }
      for (int j = 0; j < sizeof(aTemps); j++)
      {
        dataString = dataString + String(aTemps[j]) + ",";
      }

      dataFile.println(dataString);
      dataFile.close();
      // print to the serial port too:
      Serial.println(dataString);
    }
    // if the file isn't open, pop up an error:
    else
    {
      Serial.println("error opening datalog.txt");
    }
    vTaskDelayUntil(&getTick, 1000 / portTICK_PERIOD_MS);
  }
}

/*-----------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*-----------------------------------------------------*/
void I2C(void *pvParameters)
{
  (void) pvParameters;

  TickType_t getTick;
  getTick = xTaskGetTickCount();

  const byte I2C_SLAVE1_ADDR = 1;
  const byte I2C_SLAVE2_ADDR = 2;
  const byte I2C_SLAVE3_ADDR = 3;
  const byte I2C_SLAVE4_ADDR = 4;
  int contador=0;

  Wire.begin();

  while (1)
  {
    contador=0;
    Wire.requestFrom(I2C_SLAVE1_ADDR, 20); /* request & read data of
size 9 from slave */
    while (Wire.available())
    {
```

```
      if (contador<8)
      {
        aVoltajes[contador]=Wire.read();
      }
      else if (contador<9)
      {
        aIntensidad[0]=Wire.read();
      }
      else
      {
        aTemps[0]=Wire.read();
      }
      contador++;
    }

    contador=0;
    Wire.requestFrom(I2C_SLAVE2_ADDR, 20); /* request & read data of
size 9 from slave */
    while (Wire.available())
    {
      if (contador<8)
      {
        aVoltajes[contador]=Wire.read();
      }
      else if (contador<9)
      {
        aIntensidad[1]=Wire.read();
      }
      else
      {
        aTemps[1]=Wire.read();
      }
      contador++;
    }

    contador=0;
    Wire.requestFrom(I2C_SLAVE3_ADDR, 20); /* request & read data of
size 9 from slave */
    while (Wire.available())
    {
      if (contador<8)
      {
        aVoltajes[contador]=Wire.read();
      }
      else if (contador<9)
      {
        aIntensidad[2]=Wire.read();
      }
      else
      {
        aTemps[2]=Wire.read();
      }
      contador++;
    }

    contador=0;
    Wire.requestFrom(I2C_SLAVE4_ADDR, 20); /* request & read data of
size 9 from slave */
    while (Wire.available())
    {
```

```
      if (contador<8)
      {
        aVoltajes[contador]=Wire.read();
      }
      else if (contador<9)
      {
        aIntensidad[3]=Wire.read();
      }
      else
      {
        aTemps[3]=Wire.read();
      }
      contador++;
    }

    vTaskDelayUntil(&getTick, 80 / portTICK_PERIOD_MS);
  }
}

/*-----------------------------------------------------*/
/*---------------------- Tasks ----------------------*/
/*-----------------------------------------------------*/
void soc(void *pvParameters)
{
  (void) pvParameters;

  TickType_t getTick;
  getTick = xTaskGetTickCount();

  int intensidadTotal=0;

  while (1)
  {
    intensidadTotal=0;
    for (int i=0; i<sizeof(aIntensidad);i++)
    {
      intensidadTotal=intensidadTotal+aIntensidad[i];
    }
    //  SOC en tanto por 1
    //  aIntensidad en A y capacidadBatería en Ah
    SOC=SOC-float(intensidadTotal)/(capacidadBateria*3600.0);

    vTaskDelayUntil(&getTick, 1000 / portTICK_PERIOD_MS);
  }
}
```

11. ANEXO VI: Software de monitorización y análisis de funcionamiento

```
/*-------------------------------------------------------------------
----------------------SOFTWARE DE ANALISIS DE DATOS DE UN BMS---------
----------------------DISEÑADO Y CREADO POR MARKEL LIZARRALDE---------
-------------------------------------------------------------------
-------------*/

import grafica.*;
import interfascia.*;
// Para el bluetooth
import processing.serial.*;
// Para la lista desplegable
import controlP5.*;
import java.util.*;

/*-------------------------------------------------------------------
---------------
-----------------VARIABLES MODIFICABLES EN EL PROGRAMA-----------------
---------------
-------------------------------------------------------------------
-------------*/
int NumeroCeldas = 27;
int NumeroSensoresTemp=14;
int numeroDatos = 51;
String [] etiquetasDatos =  {"time", "V", "I", "P", "vel", "SOC",
"tempMotor", "rpm", "lat", "long",
  "c1", "c2", "c3", "c4", "c5", "c6", "c7", "c8", "c9", "c10",
  "c11", "c12", "c13", "c14", "c15", "c16", "c17", "c18", "c19",
"c20",
  "c21", "c22", "c23", "c24", "c25", "c26", "c27",
  "t1", "t2", "t3", "t4", "t5", "t6", "t7", "t8", "t9", "t10", "t11",
"t12",
  "t13", "t14"};
String [] etiquetasEjeY =  {"Tiempo (s)", "Voltaje bateria (V)",
"Intensidad (A)", "Potencia (kW)", "Velocidad (km/h)", "SOC (%)",
  "Temperatura motor (°C)", "Revoluciones motor (rpm)", "Latitud",
"Longitud",
  "celda 1 (V)", "celda 2 (V)", "celda 3 (V)", "celda 4 (V)", "celda 5
(V)", "celda 6 (V)", "celda 7 (V)", "celda 8 (V)", "celda 9 (V)",
"celda 10 (V)",
  "celda 11 (V)", "celda 12 (V)", "celda 13 (V)", "celda 14 (V)",
"celda 15 (V)", "celda 16 (V)", "celda 17 (V)", "celda 18 (V)", "celda
19 (V)", "celda 20 (V)",
  "celda 21 (V)", "celda 22 (V)", "celda 23 (V)", "celda 24 (V)",
"celda 25 (V)", "celda 26 (V)", "celda 27 (V)",
  "Sensor temperatura 1 (°C)", "Sensor temperatura 2 (°C)",
  "Sensor temperatura 3 (°C)", "Sensor temperatura 4 (°C)", "Sensor
temperatura 5 (°C)",
  "Sensor temperatura 6 (°C)", "Sensor temperatura 7 (°C)", "Sensor
temperatura 8 (°C)",
  "Sensor temperatura 9 (°C)", "Sensor temperatura 10 (°C)", "Sensor
temperatura 11 (°C)",
  "Sensor temperatura 12 (°C)", "Sensor temperatura 13 (°C)", "Sensor
temperatura 14 (°C)"};
```

```
/*-----------------------------------------------------------------------
--------------
----------------------RESTO DE VARIABLES DEL PROGRAMA-----------------
--------------
-----------------------------------------------------------------------
-------------*/
// Creamos los objetos para elegir el modo de visualizacion de datos
GUIController guiController;
IFRadioController seleccionModoController;
IFRadioButton modoOnline, modoAnalisis;

//  Se crea el logotipo del equipo y el fondo
PImage iconoETSIB, fondo;

//  Se crea el mapa de los arcos y las variables necesarias para coloc
arlo
PImage MapaLosArcos;
float mapWidth, mapHeight, origenMapaX, origenMapaY = 0.0;

// Creamos los objetos de graficos
GPlot plot1, plot2, plot3, plot4, plotC;
GPlot [] graficos = new GPlot [4];

// Creamos el puerto para la conexion bluetooth
Serial myBluetooth;
boolean ConexionBluetooth, conexionNueva, cerrarConexion,
datosBluetoothDisponibles=false;
int indiceBluetooth=0;

boolean seleccionArchivoCancelado=false;

// Creamos el objeto que contendrá el texto del cuadro de comandos
Textarea textoCuadro;

// Creamos los objetos de la interfaz
ControlP5 listaDesplegable1, listaDesplegable2, listaDesplegable3, bot
onGraficos;
ControlP5   SliderSOC, knobVelocimetro, SliderBMSTemp, toggleAnimacion
, sliderAnimacion,botonGraficosDefecto;
ControlP5 cuadroComunicaciones,botonCargar,botonBluetoothConnect, boto
nBluetoothDisconnect;

//  Array que contiene los botones de centrar graficos, zoom en X y zo
om en Y
//  El primer elemento del array es el boton que afecta a todos los gr
aficos
//  El resto de elementos van en orden con los graficos
ControlP5 [] botonesCentrar = new ControlP5[5];
ControlP5 [] botonesZoomX = new ControlP5[5];
ControlP5 [] botonesZoomY = new ControlP5[5];

//  Contiene el numero de veces que se ha hecho zoom en cada grafico
//  Sirve para calcular los puntos a graficar al pulsar en zoom
int[] ZoomsRealizadosX=new int[4];
int[] ZoomsRealizadosY=new int[4];

//  Valor del slider que mueve la animacion de los datos
int SliderAnimacionValor=0;
//  Contador que se utiliza junto con el slider para realizar
```

```
//  las animaciones
int contadorAnimaciones=0;
//  booleana que indica si en el modo de visualizacion se está
//  haciendo toda la animación de variables o está parado
boolean GraficarAnimaciones=false;

//  Se utiliza para saber que modo estaba elegido al cambiar a un nuev
o
//  modo en el programa
int seleccionAnterior=0;

// En esta variable se guarda la opción escogida en la lista
int seleccionLista1, seleccionLista2= 0;

// Creamos el objeto que grafica las temperaturas
TemperaturasBateria GraficoTemperaturas;
float [] temperaturas = new float[NumeroSensoresTemp];
float [] tempsColor = new float[2];
float [] celdas = new float[NumeroCeldas];
float R, G, B=0.0;

// Variables que controlan los degradados de las líneas
int colorContador=0;
boolean subiendo=true;
int velocidadContorno=15;

// Creamos el objeto velocimetro
Knob speedKnob, rpmKnob;

//Creamos el objeto que contendra el CSV que importemos del bluetooth
PrintWriter CSV;

//  numero de muestreos que hay en el archivo que se lee en el modo an
alisis
int numeroMuestreos=0;

//  Array que contiene todos los datos que se están procesando
GPointsArray [] datos= new GPointsArray[numeroDatos];
//  Array que contiene los datos que se grafican
GPointsArray [] datosaGraficar= new GPointsArray[4];
//  Array que indica qué gráficas hay que utilizar
boolean[] graficoSeleccionado = new boolean[4];
//  Array que indica qué array de datos[] hay que introducir en datosa
Graficar[]
int [] indicesDatos = new int [4];
//  Array que contiene el último mensaje bluetooth recibido
float [] datosBluetooth = new float[numeroDatos];
//  Booleana que indica si se ha cargado los datos de un archivo o no
boolean datosCargados=false;
//  Dirección y nombre del archivo a cargar o escribir
String nombreArchivo="";
//  Indica si se ha elegido ya un archivo o no
boolean archivoElegido=false;

/*--------------------------------------------------------------------
---------------
-----------------------------------SETUP--------------------------------
---------------
--------------------------------------------------------------------------
-------------*/
```

```
void setup()
{
  // Define the window size
  size(displayWidth, displayHeight);
  //surface.setResizable(true);

  /*-------------------------------------------------------------------
  --------------Inicializamos variables-----------------------------
  -------------------------------------------------------------------*/
  for (int i=0; i<numeroDatos; i++)
  {
    datosBluetooth[i]=0.0;
  }
  for (int i=0; i<4; i++)
  {
    ZoomsRealizadosX[i]=0;
    ZoomsRealizadosY[i]=0;
  }

  for (int i=0; i<4; i++)
  {
    graficoSeleccionado[i]=false;
  }

  /*-------------------------------------------------------------------
  ---------SE LLAMA A TODOS LOS CONSTRUCTORES DE LOS OBJETOS---------
  -------------------------------------------------------------------*/
  for (int j=0; j<numeroDatos; j++)
  {
    datos[j] = new GPointsArray();
  }

  for (int j=0; j<4; j++)
  {
    indicesDatos[j]=j;
    datosaGraficar[j] = new GPointsArray();
  }

  for (int j=0; j<5; j++)
  {
    botonesCentrar[j] = new ControlP5(this);
    botonesZoomX[j] = new ControlP5(this);
    botonesZoomY[j] = new ControlP5(this);
  }

  botonGraficosDefecto = new ControlP5(this);

  // Inicializa los graficos
  for (int i=0; i<=3; i++)
  {
    graficos[i]=new GPlot(this);
  }

  // Creamos el selector de modo de visualizacion de datos
  guiController = new GUIController(this);
  seleccionModoController = new IFRadioController("Mode Selector");

  modoAnalisis = new IFRadioButton("", width*52/100, height*2/100,
seleccionModoController);
```

```
  modoOnline = new IFRadioButton("", width*52/100, height*6/100,
seleccionModoController);

  guiController.add(modoOnline);
  guiController.add(modoAnalisis);

  modoAnalisis.setSize(width/60, width/60);
  modoOnline.setSize(width/60, width/60);

  // Cuadro de texto para escribir los mensajes del bluetooth
  cuadroComunicaciones = new ControlP5(this);

  // Añadimos las listas desplegables para escoger lo que se quiere
graficar
  listaDesplegable1 = new ControlP5(this);
  listaDesplegable2 = new ControlP5(this);

  botonGraficos = new ControlP5(this);
  botonCargar = new ControlP5(this);
  botonBluetoothConnect = new ControlP5(this);
  botonBluetoothDisconnect = new ControlP5(this);

  // Creamos el objeto grafico temperaturas
  GraficoTemperaturas = new TemperaturasBateria (width*33/40,
height*33/54, 0.5);

  // Creamos el objeto knob del velocimetro
  knobVelocimetro = new ControlP5(this);

  // Creamos los objetos de los sliders
  SliderSOC = new ControlP5(this);
  SliderBMSTemp = new ControlP5(this);
  sliderAnimacion = new ControlP5(this);

  // create a toggle and change the default look to a (on/off) switch
look
  toggleAnimacion = new ControlP5(this);
}

/*------------------------------------------------------------------
---------------
----------------------FUNCION QUE PARAMETRIZA TODOS LOS OBJETOS------
---------------
------------------------------------------------------------------
-------------*/
void crearObjetos()
{
  /*------------------------------------------------------------------
  ---------ZONA IZQUIERDA DEL PROGRAMA (GRAFICOS)--------------------
  ------------------------------------------------------------------*/
  // Colocamos el fondo de pantalla
  fondo = loadImage("fondo2.jpeg");
  fondo.resize(width, height);

  // Colocamos el icono del equipo y la firma
  iconoETSIB = loadImage("logoetsib.png");
  iconoETSIB.resize(0, height*10/100);

  //  Lista desplegable para configurar los gráficos
  listaDesplegable1.addScrollableList("dropdown1")
```

```
    //.setPosition(width/30, height/50)
    .setPosition(width/9, height/30)
    .setSize(width/15, height/10)
    .setBarHeight(height/50)
    .setItemHeight(height/50)
    .addItems(etiquetasEjeY)
    //.setType(ScrollableList.LIST) // currently supported DROPDOWN
and LIST
    ;

  //  Lista desplegable para configurar los gráficos
  List listaGraficos = Arrays.asList("Grafica 1", "Grafica 2",
"Grafica 3", "Grafica 4");
  /* add a ScrollableList, by default it behaves like a DropdownList
*/
  listaDesplegable1.addScrollableList("dropdown2")
    //.setPosition(width/30, height/50)
    .setPosition(width/30, height/30)
    .setSize(width/15, height/10)
    .setBarHeight(height/50)
    .setItemHeight(height/50)
    .addItems(listaGraficos)
    //.setType(ScrollableList.LIST) // currently supported DROPDOWN
and LIST
    ;

  //  Botón que coloca a todos los gráficos una variable por defecto
  botonGraficosDefecto.addButton("buttonGraficosDefecto")
    .setPosition(width*41/200, height*8/100)
    .setImages(loadImage("defecto1.png"), loadImage("defecto2.png"),
loadImage("defecto3.png"))
    .updateSize();

  //  Coloca todos los botones de zooms y centrar graficos
  for (int j=0; j<5; j++)
  {
    String nombreFuncionCallback="";

    nombreFuncionCallback="buttonZoomY"+j;
    botonesZoomY[j].addButton(nombreFuncionCallback)
      .setImages(loadImage("zoomY1.png"), loadImage("zoomY2.png"),
loadImage("zoomY3.png"))
      .updateSize()
      .onPress(new CallbackListener()
      { // a callback function that will be called onPress
        public void controlEvent(CallbackEvent theEvent)
        {
          String name = theEvent.getController().getName();
          ZoomY(int(name.substring(11)));
        }
      }
      );
    nombreFuncionCallback="buttonCentrar"+j;
    botonesCentrar[j].addButton(nombreFuncionCallback)
      .setImages(loadImage("expand1.png"), loadImage("expand2.png"),
loadImage("expand3.png"))
      .updateSize()
      .onPress(new CallbackListener()
      { // a callback function that will be called onPress
        public void controlEvent(CallbackEvent theEvent)
```

```
        {
          String name = theEvent.getController().getName();
          buttonCentrar(int(name.substring(13)));
        }
      }
      );
    nombreFuncionCallback="buttonZoomX"+j;
    botonesZoomX[j].addButton(nombreFuncionCallback)
      .setImages(loadImage("zoomX1.png"), loadImage("zoomX2.png"),
loadImage("zoomX3.png"))
      .updateSize()
      .onPress(new CallbackListener()
      { // a callback function that will be called onPress
        public void controlEvent(CallbackEvent theEvent)
        {
          String name = theEvent.getController().getName();
          ZoomX(int(name.substring(11)));
        }
      }
      );

    if(j==0)
    {
      botonesZoomY[j].setPosition(width*24/100, height*((19*j))/100);
      botonesCentrar[j].setPosition(width*24/100,
height*(4+(19*j))/100);
      botonesZoomX[j].setPosition(width*24/100,
height*(8+(19*j))/100);
    }
    else
    {
      botonesZoomY[j].setPosition(width*44/100, height*(-
2+(19*j))/100);
      botonesCentrar[j].setPosition(width*44/100,
height*(2+(19*j))/100);
      botonesZoomX[j].setPosition(width*44/100,
height*(6+(19*j))/100);
    }
  }

  //  Botón que se pulsa para añadir el dato seleccionada en la
gráfica deseada
  //  gracias a las listas desplegables
  botonGraficos.addButton("buttonGraficos")
    .setPosition(width*20/100, height*3/100)
    .setImages(loadImage("imgrafico1.png"),
loadImage("imgrafico2.png"), loadImage("imgrafico3.png"))
    .updateSize();



  // Inicializa los graficos
  for (int i=0; i<=3; i++)
  {
    graficos[i].setPos(width*3/200, height*(15+(19*i))/100);
    graficos[i].setDim(width*38/100, height*13/100);
  }

  /*-------------------------------------------------------------------
  ---------ZONA DERECHA SUPERIOR DE COMUNICACIONES------------------
```

```
  ------------------------------------------------------------------*/
  //  Botón para conectarse por bluetooth al bms
  botonBluetoothConnect.addButton("buttonBluetoothConnect")
    .setPosition(width*189/200, height*5/100)
    .setImages(loadImage("bluetooth1.png"),
loadImage("bluetooth2.png"), loadImage("bluetooth3.png"))
    .updateSize();

  botonBluetoothConnect.hide();

  //  Botón para desconectarse
  botonBluetoothDisconnect.addButton("buttonBluetoothDisconnect")
    .setPosition(width*189/200, height*5/100)
    .setImages(loadImage("bluetooth3.png"),
loadImage("bluetooth2.png"), loadImage("bluetooth1.png"))
    .updateSize();

  botonBluetoothDisconnect.hide();

  //  Botón para cargar los datos en el modo análisis
  botonCargar.addButton("buttonCargar")
    .setPosition(width*94/100, height*5/100)
    .setImages(loadImage("botoncargar1.png"),
loadImage("botoncargar2.png"), loadImage("botoncargar3.png"))
    .updateSize();

  // Cuadro de texto para escribir los mensajes del bluetooth
  textoCuadro = cuadroComunicaciones.addTextarea("txt")
    .setPosition(width*82/100, height*13/100)
    .setSize(width*14/100, height*20/100)
    .setFont(createFont("arial", 16))
    .setLineHeight(14)
    .setColor(color(0))
    .setColorBackground(color(255))
    .setColorForeground(color(0, 0, 255));
  ;
  textoCuadro.hide();


  /*------------------------------------------------------------------
  ---------ZONA DERECHA INFERIOR DONDE SE ANIMAN TODOS LOS DATOS-----
  ------------------------------------------------------------------*/
  speedKnob = knobVelocimetro.addKnob("speedKnob")
    .setRange(0, 200)
    .setValue(0)
    .setPosition(width*34/50, height*22/50)
    .setRadius(width/30)
    .setColorForeground(color(0, 76, 153))
    .setColorBackground(color(51, 153, 255))
    .lock()
    ;

  rpmKnob = knobVelocimetro.addKnob("rpmKnob")
    .setRange(0, 6000)
    .setValue(0)
    .setPosition(width*39/50, height*22/50)
    .setRadius(width/30)
    .setColorForeground(color(0, 76, 153))
    .setColorBackground(color(51, 153, 255))
    .lock()
```

```
  ;

SliderSOC.addSlider("sliderSOC")
  .setPosition(width*31/50, height*45/100)
  .setSize(width/50, height*5/50)
  .setRange(0, 100)
  .setColorBackground(color(51, 153, 255))
  .setValue(100.0)
  .lock()
  ;

SliderBMSTemp.addSlider("sliderBMSTemp")
  .setPosition(width*28/50, height*45/100)
  .setSize(width/50, height*5/50)
  .setRange(0, 100)
  .setColorBackground(color(51, 153, 255))
  .setValue(100.0)
  .lock()
  ;


// Colocamos el icono del mapa
mapWidth=displayWidth/8;
mapHeight=displayWidth/7;
origenMapaX=width*86/100;
origenMapaY=height*37/100;
MapaLosArcos = loadImage("LosArcosRecto.png");
MapaLosArcos.resize(round(mapWidth), round(mapHeight));
image(MapaLosArcos, origenMapaX, origenMapaY);

// Gráfico de los voltajes de las celdas
plotC = new GPlot(this);
plotC.setPos(round(displayWidth*101/200), height*23/38);
// Tamaño del grafico
plotC.setDim(round(displayWidth*3/10), round(displayHeight/5));
plotC.setYLim(2.4, 5.0);
plotC.getTitle().setText("Estado celdas en instante: " + str(0));
plotC.getTitle().setTextAlignment(LEFT);
plotC.getTitle().setRelativePos(0.1);
plotC.getXAxis().getAxisLabel().setText("Celda");
plotC.getYAxis().getAxisLabel().setText("Voltaje (V)");
plotC.drawGridLines(GPlot.BOTH);
plotC.drawLabels();
plotC.startHistograms(GPlot.VERTICAL);

//  Slider que controla las animaciones de todos los datos
sliderAnimacion.addSlider("sliderAnimacion")
  .setPosition(width*64/100, height*183/200)
  .setLabelVisible(false)
  .setSize(width*8/100, height*2/100)
  .setRange(1, 99)
  .setValue(50)
  .setDecimalPrecision(0)
  ;

sliderAnimacion.hide();

//  Botón que activa o desactiva la función de animación
toggleAnimacion.addToggle("toggleAnimacion")
  .setPosition(width*56/100, height*237/256)
```

```
    .setSize(50, 20)
    .setValue(true)
    .setLabelVisible(false)
    .setMode(ControlP5.SWITCH)
    ;

  toggleAnimacion.hide();
}

/*-------------------------------------------------------------------------
--------------
-----------------------FUNCION QUE COLOCA TODOS LOS TEXTOS, FONDO...--
--------------
-------------------------------------------------------------------------
-------------*/
void plantillaPorDefecto()
{
  background(255);

  image(fondo, 0, 0);

  image(iconoETSIB, width*30/100, height*1/100);
  textSize(16);
  fill(0);
  textAlign(CENTER);
  text("by Markel Lizarralde", width*34/100, height*13/100);
  textAlign(LEFT);
  text("Analizar archivo guardado", width*54/100, height*4/100);
  text("Analizar datos Online", width*54/100, height*8/100);
  fill(255);

  //  Funciones que se encargan de dibuar los contornos de la
aplicación
  contorno1(0, true);
  contorno2(0, true);
  contorno3(0, true);

  /* add a ScrollableList, by default it behaves like a DropdownList
*/
  listaDesplegable1.show();

  botonGraficos.show();

  speedKnob.show();

  rpmKnob.show();

  SliderSOC.show();

  SliderBMSTemp.show();

  textoCuadro.show();

  for (int j=0; j<5; j++)
  {
    botonesZoomY[j].show();
    botonesCentrar[j].show();
    botonesZoomX[j].show();
  }
```

```
  botonCargar.hide();
  botonBluetoothConnect.hide();
  botonBluetoothDisconnect.hide();

  plotC.beginDraw();
  plotC.drawBackground();
  plotC.drawBox();
  plotC.drawXAxis();
  plotC.drawYAxis();
  plotC.getTitle().setText("Estado celdas en instante: " +
str(contadorAnimaciones));
  plotC.drawTitle();
  plotC.drawHistograms();
  plotC.activatePointLabels();
  plotC.drawLabels();
  plotC.endDraw();

  for (int i=1; i<=NumeroSensoresTemp; i++)
  {
    temperaturas[i-1] = 0.0;
  }
  GraficoTemperaturas.Graficar(temperaturas);

  //text(str, x1, y1, x2, y2)
  textSize(16);
  fill(0);
  textAlign(CENTER, CENTER);
  text("Temperatura", width*57/100, height*41/100);
  text("motor (°C)", width*57/100, height*43/100);
  text("SOC (%)", width*63/100, height*42/100);
  text("RPM", width*244/300, height*42/100);
  text("Velocidad", width*214/300, height*40/100);
  text("(km/h)", width*214/300, height*42/100);
  fill(255);

  image(MapaLosArcos, origenMapaX, origenMapaY);

  //text(str, x1, y1, x2, y2)
  textSize(16);
  fill(0);
  textAlign(LEFT);
  text("Temperatura máxima:   ", width*86/100, height*90/100);
  text("Temperatura media:      ", width*86/100, height*92/100);
  text("Temperatura mínima:   ", width*86/100, height*94/100);
  text("Voltaje máximo:   ", width*75/100, height*90/100);
  text("Voltaje medio:   ", width*75/100, height*92/100);
  text("Voltaje mínimo:   ", width*75/100, height*94/100);
  fill(255);

  // Contornos de la zona de animacion y contorno interior de slider y
boton animacion
  strokeWeight(3);
  stroke(0);
  noFill();
  rect(width*74/100, height*88/100, width*24/100, height*7/100, 10);
}

/*-----------------------------------------------------------------------
---------------
```

```
----------------
FUNCION QUE COLOCA LOS TEXTOS Y ASPECTO DEL MODO ONLINE-------------
------------------------------------------------------------------------
-------------*/
void plantillaOnline()
{
  // Borramos todos los datos que había ya que puede que cambiemos de
modo de visualización
  // Si pasamos de online a analisis no queremos que se grafiquen los
datos del online y viceversa
  for (int i=0; i<numeroDatos; i++)
  {
    datos[i].removeRange(0, datos[i].getNPoints());
  }

  for (int i=0; i<4; i++)
  {
    datosaGraficar[i].removeRange(0, datos[i].getNPoints());
  }

  // Quitamos las interacciones con los graficos para el directo
  for (int i=0; i<4; i++)
  {
    graficos[i].deactivatePointLabels();
    graficos[i].deactivatePanning();
    graficos[i].deactivateZooming();
    graficos[i].updateLimits();
  }

  toggleAnimacion.hide();
  sliderAnimacion.hide();
  textoCuadro.show();
  botonGraficos.show();
  botonCargar.hide();
  botonBluetoothConnect.show();
  botonBluetoothDisconnect.hide();

  datosCargados=false;
}

/*------------------------------------------------------------------------
---------------
----------------
FUNCION QUE COLOCA LOS TEXTOS Y ASPECTO DEL MODO ANÁLISIS------------
------------------------------------------------------------------------
-------------*/
void plantillaAnalisis()
{
  // Borramos todos los datos que había ya que puede que cambiemos de
modo de visualización
  // Si pasamos de online a analisis no queremos que se grafiquen los
datos del online y viceversa
  for (int i=0; i<numeroDatos; i++)
  {
    datos[i].removeRange(0, datos[i].getNPoints());
  }
  for (int i=0; i<4; i++)
  {
    datosaGraficar[i].removeRange(0, datos[i].getNPoints());
  }
```

```
  // Ponemos a 0 este valor para que las animaciones no accedan a un
indice que no existe
  // Esto podría ocurrir porque no se ha cargado el archivo a leer aún
  numeroMuestreos=0;

  // Activamos las interacciones con los graficos ya que en el modo
online está desactivado
  for (int i=0; i<4; i++)
  {
    graficos[i].activatePointLabels();
    graficos[i].activatePanning();
    graficos[i].activateZooming(1.1, CENTER, CENTER);
    graficos[i].updateLimits();
  }


  toggleAnimacion.show();
  sliderAnimacion.show();
  textoCuadro.show();
  botonGraficos.show();
  botonCargar.show();
  botonBluetoothConnect.hide();
  botonBluetoothDisconnect.hide();

  // Contornos de la zona de animacion y contorno interior de slider y
boton animacion
  stroke(0);
  strokeWeight(2);
  rect(width*51/100, height*90/100, width*22/100, height*5/100, 10);
  strokeWeight(1);

  textSize(16);
  fill(0);
  textAlign(CENTER, CENTER);
  text("Visualización - Animación", width*57/100, height*117/128);
  fill(255);
}

/*-----------------------------------------------------------------------
---------------
----------------FUNCION QUE GESTIONA LA CONEXIÓN BLUETOOTH------------
---------------
-----------------------------------------------------------------------
-------------*/
void gestionarConexion()
{
  if (ConexionBluetooth)
  {
    //println("Bluetooth activado");
    if (conexionNueva)
    {
      println("nueva conexion");

      selectOutput("Selecciona directorio para guardar los datos:",
"fileToWrite");

      while (archivoElegido==false)
      {
        delay(1);
        //if (seleccionArchivoCancelado)
```

```
        //{
        //  selectOutput("Selecciona directorio para guardar los
datos:", "fileToWrite");
        //}
      }
      archivoElegido=false;

      CSV = createWriter(nombreArchivo);

      myBluetooth = new Serial(this, "COM4", 9600);
      myBluetooth.bufferUntil('\n');
      delay(1000);

      conexionNueva=false;

      botonBluetoothConnect.hide();
      botonBluetoothDisconnect.show();
    }

    if (cerrarConexion)
    {
      // Close the port
      myBluetooth.clear();
      myBluetooth.stop();
      CSV.flush(); // Writes the remaining data to the file
      CSV.close(); // Finishes the file
      ConexionBluetooth=false;
      cerrarConexion=false;
      delay(1000);
      println("cerrando conexion");

      botonBluetoothConnect.show();
      botonBluetoothDisconnect.hide();
    }
  }
}

/*-------------------------------------------------------------------
---------------
-----------------------
FUNCION QUE GESTIONA LOS DATOS EN EL MODO ANÁLISIS------------
-------------------------------------------------------------------
-------------*/
void AnimacionesBackup()
{
  //  Calcula el valor de contadorAnimaciones para elegir el instante
a mostrar
  //  en todas las gráficas
  if (numeroMuestreos>0)
  {
    //  Modo animación activado
    if (!GraficarAnimaciones)
    {
      contadorAnimaciones=contadorAnimaciones+SliderAnimacionValor;
      if (contadorAnimaciones>=numeroMuestreos)
      {
        contadorAnimaciones=0;
      }
    }
    //  Modo libre de visualización
```

```
    else
    {
      contadorAnimaciones=round(float(numeroMuestreos*SliderAnimacionV
alor)*0.01)-1;
    }

    for (int i=1; i<=NumeroSensoresTemp; i++)
    {
      temperaturas[i-1] = datos[i+36].getY(contadorAnimaciones);
    }

    //  Actualiza todos los gráficos, sliders...
    GraficoCeldasAnalisis();
    GraficoTemperaturas.Graficar(temperaturas);
    ColocarMapa(datos[8].getY(contadorAnimaciones), datos[9].getY(cont
adorAnimaciones));

    speedKnob.setValue(datos[4].getY(contadorAnimaciones));
    rpmKnob.setValue(datos[7].getY(contadorAnimaciones));
    SliderSOC.getController("sliderSOC").setValue(datos[5].getY(contad
orAnimaciones));
    SliderBMSTemp.getController("sliderBMSTemp").setValue(datos[6].get
Y(contadorAnimaciones));

    fill(255);
    rect(width*55/100, height*56/100, width*30/100, height*5/100);
    rect(width*74/100, height*87/100, width*49/200, height*8/100);

    //text(str, x1, y1, x2, y2)
    textSize(16);
    fill(0);
    textAlign(CENTER, CENTER);
    text(str(datos[6].getY(contadorAnimaciones))+ " °C", width*57/100,
height*58/100);
    text(str(datos[5].getY(contadorAnimaciones))+ " %", width*63/100,
height*58/100);
    text(str(datos[4].getY(contadorAnimaciones))+ " km/h",
width*214/300, height*58/100);
    text(str(datos[7].getY(contadorAnimaciones))+ " rpm",
width*244/300, height*58/100);

    textAlign(LEFT);
    text("Temperatura máxima:   ", width*86/100, height*90/100);
    text(str(GraficoTemperaturas.maximo(temperaturas))+ " °C",
width*94/100, height*90/100);
    text("Temperatura media:      ", width*86/100, height*92/100);
    text(str(GraficoTemperaturas.media(temperaturas))+ " °C",
width*94/100, height*92/100);
    text("Temperatura mínima:   ", width*86/100, height*94/100);
    text(str(GraficoTemperaturas.minimo(temperaturas))+ " °C",
width*94/100, height*94/100);

    text("Voltaje máximo:   ", width*75/100, height*90/100);
    text(str(max(celdas))+ " V", width*81/100, height*90/100);
    text("Voltaje medio:   ", width*75/100, height*92/100);
    text(str(media(celdas))+ " V", width*81/100, height*92/100);
    text("Voltaje mínimo:   ", width*75/100, height*94/100);
    text(str(min(celdas))+ " V", width*81/100, height*94/100);
    colorMode(RGB, 255);
    fill(255);
```

```
    // Contornos de la zona de animacion y contorno interior de slider
y boton animacion
    strokeWeight(3);
    stroke(0);
    noFill();
    rect(width*74/100, height*88/100, width*24/100, height*7/100, 10);

    textoCuadro.setText("");

    for (int i=0; i<numeroDatos; i++)
    {
      String str=etiquetasEjeY[i]+":
"+datos[i].getY(contadorAnimaciones)+'\n';

      textoCuadro.setText(textoCuadro.getText()+str);
    }
  }
}

/*-------------------------------------------------------------------
---------------
-----------------------
FUNCION QUE GESTIONA LOS DATOS EN EL MODO ANÁLISIS------------
-------------------------------------------------------------------
-------------*/
void AnimacionesOnline()
{
  for (int i=1; i<=NumeroSensoresTemp; i++)
  {
    temperaturas[i-1] = datosBluetooth[i+36];
  }

  for (int i=1; i<=NumeroCeldas; i++)
  {
    celdas[i-1] = datosBluetooth[i+9];
  }

  GraficoCeldasOnline(celdas);
  GraficoTemperaturas.Graficar(temperaturas);
  ColocarMapa(datosBluetooth[8], datosBluetooth[9]);

  speedKnob.setValue(datosBluetooth[4]);
  rpmKnob.setValue(datosBluetooth[7]);
  SliderSOC.getController("sliderSOC").setValue(datosBluetooth[5]);
  SliderBMSTemp.getController("sliderBMSTemp").setValue(datosBluetooth
[6]);

  fill(255);
  rect(width*55/100, height*56/100, width*30/100, height*5/100);
  rect(width*74/100, height*87/100, width*49/200, height*8/100);

  //text(str, x1, y1, x2, y2)
  textSize(16);
  fill(0);
  textAlign(CENTER, CENTER);
  text(str(datosBluetooth[6])+ " °C", width*57/100, height*58/100);
  text(str(datosBluetooth[5])+ " %", width*63/100, height*58/100);
  text(str(datosBluetooth[4])+ " km/h", width*214/300, height*58/100);
  text(str(datosBluetooth[7])+ " rpm", width*244/300, height*58/100);
```

```
  textAlign(LEFT);
  text("Temperatura máxima:    ", width*86/100, height*90/100);
  text(str(GraficoTemperaturas.maximo(temperaturas))+ " °C",
width*94/100, height*90/100);
  text("Temperatura media:      ", width*86/100, height*92/100);
  text(str(GraficoTemperaturas.media(temperaturas))+ " °C",
width*94/100, height*92/100);
  text("Temperatura mínima:    ", width*86/100, height*94/100);
  text(str(GraficoTemperaturas.minimo(temperaturas))+ " °C",
width*94/100, height*94/100);

  text("Voltaje máximo:    ", width*75/100, height*90/100);
  text(str(max(celdas))+ " V", width*81/100, height*90/100);
  text("Voltaje medio:    ", width*75/100, height*92/100);
  text(str(media(celdas))+ " V", width*81/100, height*92/100);
  text("Voltaje mínimo:    ", width*75/100, height*94/100);
  text(str(min(celdas))+ " V", width*81/100, height*94/100);
  colorMode(RGB, 255);
  fill(255);

  // Contornos de la zona de animacion y contorno interior de slider y
boton animacion
  strokeWeight(3);
  stroke(0);
  noFill();
  rect(width*74/100, height*88/100, width*24/100, height*7/100, 10);

  textoCuadro.setText("");

  for (int i=0; i<numeroDatos; i++)
  {
    String str=etiquetasEjeY[i]+": "+datosBluetooth[i]+'\n';
    textoCuadro.setText(textoCuadro.getText()+str);
  }
}
/*--------------------------------------------------------------------
---------------
----------------------FUNCIONES QUE DIBUJAN LOS CONTORNOS------------
---------------
--------------------------------------------------------------------
-------------*/
void contornos()
{
  if (colorContador-velocidadContorno<=255 && subiendo==true)
  {
    colorContador=colorContador+velocidadContorno;
  } else
  {
    subiendo=false;

    if (colorContador-velocidadContorno>=0)
    {
      colorContador=colorContador-velocidadContorno;
    } else
    {
      subiendo=true;
    }
  }
```

```
  contorno1(colorContador, true);
  contorno2(colorContador, false);
  contorno3(colorContador, false);
}

void contorno1(int Color, boolean fill)
{
  //  Vertices que dibujan el contorno de los graficos
  if (fill)
  {
    fill(255);
  } else
  {
    noFill();
  }
  strokeWeight(3);
  stroke(Color, 0, 0);
  beginShape();
  vertex(width*3/100, height/100);
  vertex(width*26/100, height/100);
  quadraticVertex(width*28/100, height/100, width*28/100,
height*3/100);
  vertex(width*28/100, height*12/100);
  quadraticVertex(width*28/100, height*14/100, width*31/100,
height*14/100);
  vertex(width*47/100, height*14/100);
  quadraticVertex(width*49/100, height*14/100, width*49/100,
height*16/100);
  vertex(width*49/100, height*94/100);
  quadraticVertex(width*49/100, height*96/100, width*47/100,
height*96/100);
  vertex(width*3/100, height*96/100);
  quadraticVertex(width*1/100, height*96/100, width*1/100,
height*94/100);
  vertex(width*1/100, height*3/100);
  quadraticVertex(width*1/100, height/100, width*3/100, height/100);
  endShape();
  strokeWeight(1);
}

void contorno2(int Color, boolean fill)
{
  //  Vertices que dibujan el contorno de el cuadro de comunicaciones
  if (fill)
  {
    fill(255);
  } else
  {
    noFill();
  }
  strokeWeight(3);
  stroke(Color, 0, 0);
  beginShape();
  vertex(width*52/100, height*10/100);
  vertex(width*90/100, height*10/100);
  quadraticVertex(width*92/100, height*10/100, width*92/100,
height*8/100);
  vertex(width*92/100, height*5/100);
  quadraticVertex(width*92/100, height*3/100, width*94/100,
height*3/100);
```

```
  vertex(width*97/100, height*3/100);
  quadraticVertex(width*99/100, height*3/100, width*99/100,
height*5/100);
  vertex(width*99/100, height*32/100);
  quadraticVertex(width*99/100, height*34/100, width*97/100,
height*34/100);
  vertex(width*52/100, height*34/100);
  quadraticVertex(width*50/100, height*34/100, width*50/100,
height*32/100);
  vertex(width*50/100, height*12/100);
  quadraticVertex(width*50/100, height*10/100, width*52/100,
height*10/100);
  endShape();
  strokeWeight(1);
}

void contorno3(int Color, boolean fill)
{
  // Contornos de la zona de animacion y contorno interior de slider y
boton animacion
  strokeWeight(3);
  if (fill)
  {
    fill(255);
  } else
  {
    noFill();
  }
  stroke(Color, 0, 0);
  rect(width*10/20, height*35/100, width*98/200, height*61/100,
height*3/100);
}


/*--------------------------------------------------------------------
---------------
-----------------------------------------------DRAW-------------------
---------------
--------------------------------------------------------------------
-------------*/
void draw()
{

  if (modoOnline.isSelected())
  {
    if (seleccionAnterior!=1)
    {
      println("modo online");
      plantillaPorDefecto();
      plantillaOnline();
      seleccionAnterior=1;
    }

    gestionarConexion();

    if (datosBluetoothDisponibles)
    {
      println("animando");
      AnimacionesOnline();
      datosBluetoothDisponibles=false;
```

```
      contornos();
    }
  } else if (modoAnalisis.isSelected())
  {
    if (seleccionAnterior!=2)
    {
      println("modo backup");
      plantillaPorDefecto();
      plantillaAnalisis();
      seleccionAnterior=2;
    }

    if (datosCargados)
    {
      AnimacionesBackup();
    }
  } else if (seleccionAnterior==0)
  {
    println("plantilla creada");
    crearObjetos();
    plantillaPorDefecto();
    seleccionAnterior=3;
    datosCargados=false;
  }

  Graficos();

  //  Herramienta cojonuda para sacar capturas y hacer un video
  //  Herramientas -> Creador de películas
  //  https://processing.org/reference/saveFrame_.html
  //saveFrame("frames/line-######.png");

  //  como crear un archivo csv a partir de los datos
  //  https://processing.org/reference/saveTable_.html
  //  Libreria de Table()
  //  https://processing.org/reference/Table.html
  //  Libreria loadTable()
  //  https://processing.org/reference/loadTable_.html
}


void Graficos()
{
  //  Parte que se encarga de refrescar las graficas en cada frame
  fill(255);
  stroke(255);
  rect(width*2/100, height*2/100, width*22/100, height*15/100);
  rect(width*2/100, height*17/100, width*42/100, height*78/100);

  for (int i=0; i<=3; i++)
  {
    if (graficoSeleccionado[i])
    {
      if (modoOnline.isSelected())
      {
        graficos[i].getXAxis().setAxisLabelText("Tiempo (s):
"+datosBluetooth[indicesDatos[0]]);
        graficos[i].getYAxis().setAxisLabelText(etiquetasEjeY[indicesD
atos[i]]+": "+datosBluetooth[indicesDatos[i]]);
```

```
      } else if (modoAnalisis.isSelected() && datosCargados)
      {
        graficos[i].getXAxis().setAxisLabelText("Tiempo (s): 
"+datosaGraficar[i].getX(contadorAnimaciones));
        graficos[i].getYAxis().setAxisLabelText(etiquetasEjeY[indicesD
atos[i]]+": "+datosaGraficar[i].getY(contadorAnimaciones));
        graficos[i].updateLimits();
      } else
      {
        graficos[i].getXAxis().setAxisLabelText("Tiempo (s): ");
        graficos[i].getYAxis().setAxisLabelText(etiquetasEjeY[indicesD
atos[i]]);
      }
      graficos[i].setPoints(datosaGraficar[i]);
    }
    // Draw the plot
    graficos[i].beginDraw();
    graficos[i].drawBox();
    graficos[i].drawXAxis();
    graficos[i].drawYAxis();
    graficos[i].drawTitle();
    graficos[i].drawGridLines(GPlot.BOTH);
    graficos[i].drawPoints();
    graficos[i].drawLabels();
    graficos[i].endDraw();
  }
}

/*
------------------------------------------------------------------------
-----------------------
 PROCEDIMIENTO QUE GRAFICA LOS VOLTAJES DE LAS CELDAS
 ------------------------------------------------------------------------
-----------------------
 */
void GraficoCeldasAnalisis()
{
  GPointsArray pointsC = new GPointsArray(NumeroCeldas);
  Table table = loadTable(nombreArchivo, "header");

  for (int i=1; i<=NumeroCeldas; i++)
  {
    String id = trim("c" + str(i));
    celdas[i-1] = table.getFloat(contadorAnimaciones, id);
    pointsC.add(i, celdas[i-1], str(celdas[i-1]));
  }

  plotC.beginDraw();
  plotC.drawBackground();
  plotC.drawBox();
  plotC.drawXAxis();
  plotC.drawYAxis();
  plotC.getTitle().setText("Estado celdas en instante: " +
str(contadorAnimaciones));
  plotC.drawTitle();
  plotC.drawHistograms();
  plotC.setPoints(pointsC);
  plotC.activatePointLabels();
  plotC.drawLabels();
  plotC.endDraw();
```

```
}

void GraficoCeldasOnline(float[] celdas)
{
  GPointsArray pointsC = new GPointsArray(NumeroCeldas);

  for (int i=1; i<=NumeroCeldas; i++)
  {
    pointsC.add(i, celdas[i-1], str(celdas[i-1]));
  }

  plotC.beginDraw();
  plotC.drawBackground();
  plotC.drawBox();
  plotC.drawXAxis();
  plotC.drawYAxis();
  plotC.getTitle().setText("Estado celdas en instante: " +
str(contadorAnimaciones));
  plotC.drawTitle();
  plotC.drawHistograms();
  plotC.setPoints(pointsC);
  plotC.activatePointLabels();
  plotC.drawLabels();
  plotC.endDraw();
}

void serialEvent(Serial myBluetooth)
{

  // read a string from the serial port:
  String BluetoothInput = myBluetooth.readStringUntil('\n');

  if (BluetoothInput!=null)
  {
    textoCuadro.setText(textoCuadro.getText()
      +BluetoothInput
      );

    CSV.print(BluetoothInput);

    if (indiceBluetooth>0)
    {
      // Crea un array que contiene todos los valores
      // Le extrae las comas y guarda los valores como string
      String [] listaDatos= split(BluetoothInput, ',');
      // Ahora pasamos los datos a float y los añadimos al
GPointsArray
      if (listaDatos.length==numeroDatos)
      {
        for (int i=0; i<numeroDatos; i++)
        {

          datosBluetooth [i]= float(trim(listaDatos[i]));
          println("hasta aqui");
          println( listaDatos [i]);
          println( datosBluetooth [i]);
          datos[i].add(datosBluetooth[0], datosBluetooth[i]);
        }
        datosBluetoothDisponibles=true;
      }
```

```
    }
    indiceBluetooth++;
  }


  //textoCuadro.setText(textoCuadro.getText() + BluetoothInput);
}

/*
-------------------------------------------------------------------------
-----------------------
 CLASE DIBUJO TEMPERATURAS
 -------------------------------------------------------------------------
-----------------------
 */

class TemperaturasBateria
{

  int posX, posY;
  //[filas][columnas][RGB]
  float [][] temperaturasRGB = new float [NumeroSensoresTemp][2];
  float escala;

  TemperaturasBateria(int x, int y, float esc)
  {
    escala=esc;
    posX=round(x/escala);
    posY=round(y/escala);
  }

  // Mapeo a RGB las temperaturas
  void Graficar(float[] temps)
  {
    colorMode(RGB, 1);
    stroke(0);
    scale(escala);

    for (int i=1; i<=NumeroSensoresTemp; i++)
    {
      colorear(temps[i-1], temperaturasRGB[i-1]);
    }

    pushMatrix();

    textAlign(CENTER, CENTER);

    int indice = 1;

    for (int j=0; j<=300; j=j+100)
    {

      for (int l=100; l<=300; l=l+100)
      {

        if (indice==4)
        {
          indice++;
        }
```

```
        beginShape(QUADS);

        fill(temperaturasRGB[indice][0], temperaturasRGB[indice][1],
0);
        vertex(posX+l+100, posY+j+100);


        fill(temperaturasRGB[indice][0], temperaturasRGB[indice][1],
0);
        vertex(posX+l+100, posY+j+200);

        fill(temperaturasRGB[indice][0], temperaturasRGB[indice][1],
0);
        vertex(posX+l+200, posY+j+200);

        fill(temperaturasRGB[indice][0], temperaturasRGB[indice][1],
0);
        vertex(posX+l+200, posY+j+100);

        endShape();

        fill(0, 0, 0);
        textSize(30);
        text(str(temps[indice]), posX+l+150, posY+j+150);

        //print("  (" + nf(temperaturasRGB[indice][0],1,2) + " , " +
nf(temperaturasRGB[indice][1],1,2) + ")");
        indice++;
      }
    }

    beginShape(QUADS);

    fill(temperaturasRGB[0][0], temperaturasRGB[0][1], 0);
    vertex(posX+100, posY+100);

    fill(temperaturasRGB[0][0], temperaturasRGB[0][1], 0);
    vertex(posX+100, posY+200);

    fill(temperaturasRGB[0][0], temperaturasRGB[0][1], 0);
    vertex(posX+200, posY+200);

    fill(temperaturasRGB[0][0], temperaturasRGB[0][1], 0);
    vertex(posX+200, posY+100);

    endShape();

    fill(0, 0, 0);
    textSize(30);
    text(str(temps[0]), posX+150, posY+150);

    beginShape(QUADS);

    fill(temperaturasRGB[4][0], temperaturasRGB[4][1], 0);
    vertex(posX+500, posY+100);

    fill(temperaturasRGB[4][0], temperaturasRGB[4][1], 0);
    vertex(posX+500, posY+200);

    fill(temperaturasRGB[4][0], temperaturasRGB[4][1], 0);
```

```
  vertex(posX+600, posY+200);

  fill(temperaturasRGB[4][0], temperaturasRGB[4][1], 0);
  vertex(posX+600, posY+100);

  endShape();

  fill(0, 0, 0);
  textSize(30);
  text(str(temps[4]), posX+550, posY+150);

  popMatrix();

  colorMode(RGB, 255);
  scale(1/escala);

  noStroke();
}

float media(float [] temperaturas)
{
  float media=0.0;

  for (int i=1; i<=NumeroSensoresTemp; i++)
  {
    media = media + temperaturas[i-1];
  }

  return round((media/float(NumeroSensoresTemp))*10.0)/10.0;
}

void valores(float [] temperaturas)
{
  Table table = loadTable(nombreArchivo, "header");

  for (int i=1; i<=NumeroSensoresTemp; i++)
  {
    String id = "t" + str(i);
    temperaturas[i-1] = table.getFloat(contadorAnimaciones, id);
  }
}

float maximo(float [] temperaturas)
{
  return max(temperaturas);
}

float minimo(float [] temperaturas)
{
  return min(temperaturas);
}

void colorear(float temp, float [] tRGB)
{
  if (temp<=50.0)
  {
    tRGB[0]=0.02*temp;
    tRGB[1]=1.0;
  } else if (temp<=100.0)
  {
```

```
      tRGB[0]=1.0;
      tRGB[1]=-0.02*temp+2.0;
    }
  }
}


void dropdown1(int n)
{
  seleccionLista1 = n;
}

void dropdown2(int n)
{
  seleccionLista2 = n;
}

// function buttonGraficos will receive changes from
// controller with name buttonGraficos
public void buttonGraficos(int theValue)
{

  graficoSeleccionado[seleccionLista2]=true;
  indicesDatos[seleccionLista2] = seleccionLista1;
  datosaGraficar[seleccionLista2]=datos[seleccionLista1];
  graficos[seleccionLista2].setPoints(datosaGraficar[seleccionLista2])
;

  graficos[seleccionLista2].getXAxis().setAxisLabelText("Tiempo (s)");
  graficos[seleccionLista2].getYAxis().setAxisLabelText(etiquetasEjeY[
seleccionLista1]);

  // Draw the plot
  graficos[seleccionLista2].beginDraw();
  graficos[seleccionLista2].drawBox();
  graficos[seleccionLista2].drawXAxis();
  graficos[seleccionLista2].drawYAxis();
  graficos[seleccionLista2].drawTitle();
  graficos[seleccionLista2].drawGridLines(GPlot.BOTH);
  graficos[seleccionLista2].drawPoints();
  graficos[seleccionLista2].drawLabels();
  graficos[seleccionLista2].endDraw();
}

void buttonGraficosDefecto(int thevalue)
{
  for (int i=0;i<4;i++)
  {
    graficoSeleccionado[i]=true;
    indicesDatos[i] = i;
    datosaGraficar[i]=datos[i];
    graficos[i].setPoints(datosaGraficar[i]);

    graficos[i].getXAxis().setAxisLabelText("Tiempo (s)");
    graficos[i].getYAxis().setAxisLabelText(etiquetasEjeY[i]);

    // Draw the plot
    graficos[seleccionLista2].beginDraw();
    graficos[seleccionLista2].drawBox();
    graficos[seleccionLista2].drawXAxis();
```

```
    graficos[seleccionLista2].drawYAxis();
    graficos[seleccionLista2].drawTitle();
    graficos[seleccionLista2].drawGridLines(GPlot.BOTH);
    graficos[seleccionLista2].drawPoints();
    graficos[seleccionLista2].drawLabels();
    graficos[seleccionLista2].endDraw();
  }
}

public void buttonCentrar(int numeroBoton)
{
  if(modoAnalisis.isSelected())
  {
    if (numeroBoton==0)
    {
      for (int i=0; i<4; i++)
      {
        centrarGraficos(i);
      }
    }
    else if (numeroBoton<=4 && numeroBoton>0)
    {
      centrarGraficos(numeroBoton-1);
    }
    println("centrando grafico "+numeroBoton);
  }

}

void centrarGraficos(int i)
{
  if (graficoSeleccionado[i])
  {
    int numDatos=datosaGraficar[i].getNPoints();
    float [] valoresX = new float [numDatos];
    float [] valoresY = new float [numDatos];

    for (int j=0; j<numDatos; j++)
    {
      valoresX[j] = datosaGraficar[i].getX(j);
      valoresY[j] = datosaGraficar[i].getY(j);
    }

    graficos[i].setXLim(min(valoresX), max(valoresX));
    graficos[i].setYLim(min(valoresY), max(valoresY));

    //println("Grafico "+i+": "+"limX: "+limitesX[0]+"
"+limitesX[1]+"   "+"limY: "+limitesY[0]+"  "+limitesY[1]);
  }
}

void ZoomX(int numeroBoton)
{
  if(modoAnalisis.isSelected())
  {
    if (numeroBoton==0)
    {
      for (int i=0; i<4; i++)
      {
        ZoomGraficoX(i);
```

```
      }
    } else if (numeroBoton<=4 && numeroBoton>0)
    {
      ZoomGraficoX(numeroBoton-1);
    }
  }
}


void ZoomGraficoX(int numGraf)
{
  float[] offsetZoom={0.05, 0.05, 0.05, 0.05};

  if (graficoSeleccionado[numGraf])
  {
    int numDatos=datosaGraficar[numGraf].getNPoints();

    if (key=='+')
    {
      if (round(float(numDatos)*(1-
offsetZoom[numGraf]*(ZoomsRealizadosX[numGraf]+1)))-
round(offsetZoom[numGraf]*(ZoomsRealizadosX[numGraf]+1)*float(numDatos
))>1)
      {
        ZoomsRealizadosX[numGraf]++;
      }
    } else if (key=='-')
    {
      if (ZoomsRealizadosX[numGraf]>0)
      {
        ZoomsRealizadosX[numGraf]--;
      }
    }

    offsetZoom[numGraf]=offsetZoom[numGraf]*ZoomsRealizadosX[numGraf];

    int offsetL=round(offsetZoom[numGraf]*float(numDatos));
    int offsetR=round(float(numDatos)*(1-offsetZoom[numGraf]));
    int numDatosNuevos=offsetR-offsetL;

    float [] valoresX = new float [numDatosNuevos];

    println("numero datos: "+numDatos+"
"+round(offsetZoom[numGraf]*float(numDatos))+"
"+round(float(numDatos)*(1-offsetZoom[numGraf])));
    for (int j=0; j<numDatosNuevos; j++)
    {
      valoresX[j] = datosaGraficar[numGraf].getX(j+offsetL);
    }

    graficos[numGraf].setXLim(min(valoresX), max(valoresX));

    //println("Grafico "+numeroBoton+": "+"limX: "+limitesX[0]+"
"+limitesX[1]+"   "+"limY: "+limitesY[0]+"  "+limitesY[1]);
  }
}

void ZoomY(int numeroBoton)
{
  if(modoAnalisis.isSelected())
```

```
  {
    if (numeroBoton==0)
    {
      for (int i=0; i<4; i++)
      {
        ZoomGraficoY(i);
      }
    } else if (numeroBoton<=4 && numeroBoton>0)
    {
      ZoomGraficoY(numeroBoton-1);
    }
  }
}

void ZoomGraficoY(int numGraf)
{

  float offsetPorcentaje=0.04;

  if (graficoSeleccionado[numGraf])
  {
    int numDatos=datosaGraficar[numGraf].getNPoints();

    float [] valoresY = new float [numDatos];
    float [] valoresX = new float [numDatos];

    for (int j=0; j<numDatos; j++)
    {
      valoresY[j] = datosaGraficar[numGraf].getY(j);
      valoresX[j] = datosaGraficar[numGraf].getX(j);
    }

    float offset=(max(valoresY)-min(valoresY))*offsetPorcentaje;

    if (key=='+' && (max(valoresY)-
offset*(ZoomsRealizadosY[numGraf]+1))>(min(valoresY)+offset*(ZoomsReal
izadosY[numGraf]+1)))
    {
      //println("max: "+max(valoresY)+"  min: "+min(valoresY));
      //println(max(valoresY)-offset*(ZoomsRealizadosY[numGraf]+1)+"
"+(min(valoresY)+offset*(ZoomsRealizadosY[numGraf]+1)));
      ZoomsRealizadosY[numGraf]++;
    } else if (key=='-')
    {
      if (ZoomsRealizadosY[numGraf]>0)
      {
        //println("max: "+max(valoresY)+"  min: "+min(valoresY));
        //println(max(valoresY)-offset*(ZoomsRealizadosY[numGraf]+1)+"
"+(min(valoresY)+offset*(ZoomsRealizadosY[numGraf]+1)));
        ZoomsRealizadosY[numGraf]--;
      }
    }

    graficos[numGraf].setYLim(min(valoresY)+offset*ZoomsRealizadosY[nu
mGraf], max(valoresY)-offset*ZoomsRealizadosY[numGraf]);
    graficos[numGraf].setXLim(min(valoresX), max(valoresX));
  }
}
```

```
// function buttonGraficos will receive changes from
// controller with name buttonCargar
public void buttonCargar(int theValue)
{

  selectInput("Selecciona archivo a cargar:", "fileToRead");

  while (archivoElegido==false)
  {
    delay(1);
    //if(seleccionArchivoCancelado)
    //{
    //  selectInput("Selecciona archivo a cargar:", "fileToRead");
    //}
  }
  archivoElegido=false;

  // Load the cvs dataset.
  Table table = loadTable(nombreArchivo, "header");

  numeroMuestreos=table.getRowCount();

  for (int row = 0; row < table.getRowCount(); row++)
  {
    for (int j=0; j<numeroDatos; j++)
    {
      datos[j].add(table.getInt(row, "time"), table.getFloat(row,
etiquetasDatos[j]), table.getString(row, etiquetasDatos[j]));
    }
  }

  datosCargados=true;
}

void ColocarMapa(float latitud, float longitud)
{

  image(MapaLosArcos, origenMapaX, origenMapaY);
  // Latitud es el eje y
  float mapaAltoLatitud=((42.565149+42.565154)/2.0)-
((42.555204+42.555198)/2.0);
  float mapaAnchoLongitud=((-2.159650-2.159566)/2.0)-((-2.170280-
2.170266)/2.0);

  //println("Alto y ancho en GPS: " + str(mapaAltoLatitud)+ "  " +
str(mapaAnchoLongitud));

  float origenX=((-2.170280-2.170266)/2.0);
  float origenY=((42.555204+42.555198)/2.0);

  // En coordenadas GPS
  float coordX=longitud-origenX;
  float coordY=latitud-origenY;

  //println("En coordenadas GPS: " + str(coordX)+ "  " + str(coordY));

  // Pasamos las coordenadas GPS a pixeles
  coordX = coordX*mapWidth/mapaAnchoLongitud;
  coordY = coordY*mapHeight/mapaAltoLatitud;
```

```
  //println("En pixeles: " + str(coordX)+ "  " + str(coordY));

  fill(0, 255, 0);
  ellipse(origenMapaX+coordX, origenMapaY+(mapHeight-coordY), 15, 15);
  fill(255);
}

void toggleAnimacion (boolean theValue)
{
  GraficarAnimaciones=theValue;
  //println(theValue);
}


// function buttonBluetoothConnect will receive changes from
// controller with name buttonBluetoothConnect
public void buttonBluetoothConnect(boolean theValue)
{
  println(theValue);
  ConexionBluetooth=theValue;
  conexionNueva=theValue;
}

public void buttonBluetoothDisconnect(boolean theValue)
{

  println(!theValue);
  cerrarConexion=true;
}


float media(float [] valores)
{
  float media=0.0;

  for (int i=0; i<=26; i++)
  {
    media = media + valores[i];
  }

  return round((media/float(NumeroCeldas))*100.0)/100.0;
}

void sliderAnimacion(int valor)
{
  SliderAnimacionValor = valor;
  //println(str(valor));
}


void fileToRead(File selection)
{
  if (selection == null)
  {
    println("Window was closed or the user hit cancel.");
    seleccionArchivoCancelado=true;
  } else
  {
    println("User selected " + selection.getAbsolutePath());
    nombreArchivo=selection.getAbsolutePath();
```

```
    archivoElegido=true;
    seleccionArchivoCancelado=false;
  }
}

void fileToWrite(File selection)
{
  if (selection == null)
  {
    println("Window was closed or the user hit cancel.");
    seleccionArchivoCancelado=true;
  } else
  {
    println("User selected " + selection.getAbsolutePath());
    nombreArchivo=selection.getAbsolutePath();
    archivoElegido=true;
    seleccionArchivoCancelado=false;
  }
}
```

12. ANEXO VII: COSTES DEL PROYECTO

Nº Ref	Cant. (Ud)	Concepto	Precio unitario (€/ud.)	Precio total (€)
1		**SUBCONTRATACIÓN**		
1.1	1	Subcontratación fabricación PCBs	112.59 €	112.59 €
		TOTAL SUBCONTRATACIONES		**112.59 €**
2		**MATERIALES**		
2.1	4	Placa esclava	75.40 €	301.61 €
2.2	1	Placa maestra	75.06 €	75.06 €
		TOTAL MATERIALES		**376.67 €**
		COSTE TOTAL		**489.26 €**

Subcontratación fabricación PCB			
Descripción	cantidad	€/unidad	coste
placa esclavo	4	12.38	49.52 €
stencil esclavo	1	9.9	9.90 €
placa maestro	1	22.87	22.87 €
stencil maestro	1	9.9	9.90 €
envío	1	20.4	20.40 €
		TOTAL	**112.59 €**

BMS V11 PLACA ESCLAVA					
Cantidad	Descripción	En el PCB	Mouser Part Number	Precio unitario	Coste total
9	Led	LED1, LED2, LED3, LED4, LED5, LED6, LED7, LED8, LED9	710-156120VS75000	0.16 €	1.47 €
1	0.5A	F1	652-SF-1206HI050M-2	0.72 €	0.72 €
2	100k	R17, R18	71-CRCW1206100KFKEAC	0.09 €	0.18 €
4	100nF	C6,C8,C9,C10	80-C1206C104K5R7210	0.09 €	0.36 €
1	10uF	C18	74-293D106X96R3A2TE3	0.26 €	0.26 €
2	11k	R19, R20	652-CR1206JW-113ELF	0.09 €	0.18 €
8	15	R1, R2, R3, R4, R21, R22, R23, R42	756-WP5S03-15RJT075	0.28 €	2.22 €
2	15nF	C3, C4	80-C1206C153K3RECAUT	0.28 €	0.56 €
1	1k	R43	603-RT1206FRE0771K5L	0.16 €	0.16 €
1	2.2uH	L1	81-LQM31PN2R2M00L	0.35 €	0.35 €
4	3.3uF	C10, C11, C12, C13	81-GCJ31CR71C335MA1L	0.49 €	1.94 €
8	330	R5, R6, R7, R8, R25, R26, R27, R28	71-CRCW1206-330-E3	0.07 €	0.58 €
4	3k	R29, R32, R39, R40	667-ERJ-8ENF3001V	0.07 €	0.29 €
1	4.7k	R41	71-CRCW12064K70FKEBC	0.09 €	0.09 €
2	4.7uF	C2, C5	581-TAJA475M016RNJV	0.26 €	0.52 €
1	4050D	IC8	595-CD74HC4050M96	0.42 €	0.42 €
8	535	R9, R10, R15, R16, R30, R31, R33, R38	71-CRCW1206-536-E3	0.07 €	0.58 €
8	560k	R11, R12, R13, R14, R34, R35, R36, R37	652-CR1206JW-564ELF	0.03 €	0.26 €
2	AD628ARZ	IC4, IC5	584-AD628ARZ	4.77 €	9.54 €
1	DS18B20+	IC10	700-DS18B20+	2.95 €	2.95 €
8	FDN304P	Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8	512-FDN304P	0.27 €	2.18 €
1	LP298XS	IC12	595-LP2985-30DBVR	0.46 €	0.46 €
1	MAX680CSA+	IC11	700-MAX680CSA	3.63 €	3.63 €
1	MICROSD	X1	517-2908-05WB-MG	3.39 €	3.39 €
1	MOLEX-2PIN-436500200	C15	538-43650-0200	0.82 €	0.82 €
2	MOLEX-3PIN-436500300	C14, C16	538-43650-0300	0.94 €	1.87 €
2	MOLEX-4PIN-436500400	C1, C17	538-43650-0400	1.00 €	2.00 €
1	MOLEX-5PIN-436500500	U$2	538-43650-0500	1.38 €	1.38 €
1	MOLEX-8PIN-436500800	C7	538-43650-0800	1.84 €	1.84 €
1	MOLEX-2PIN-436450200	-	538-43645-0200	0.29 €	0.29 €
2	MOLEX-3PIN-436450300	-	538-43645-0300	0.34 €	0.68 €
2	MOLEX-4PIN-436450400	-	538-43645-0400	0.41 €	0.81 €
1	MOLEX-5PIN-436450500	-	538-43645-0500	0.38 €	0.38 €
1	MOLEX-8PIN-436450800	-	538-43645-0800	0.50 €	0.50 €
58	Conectores crimpado	-	538-43030-0001-CT	0.07 €	3.77 €
2	MUX509IDR	IC1, IC2	595-MUX509IDR	2.31 €	4.62 €
1	SI8600AC-B-ISR	IC6	634-SI8600AC-B-ISR	3.00 €	3.00 €
1	SN74HC595D	IC9	595-SN74HC595D	0.55 €	0.55 €
2	TLP293-4(GB-TP,E(T	IC3, IC7	757-TLP293-4GB-TPE	1.18 €	2.36 €
1	TMH-1205S	PWR1	495-TMH-1205S	5.76 €	5.76 €
1	OLED_0.96_I2C	U$1	ebay	7.00 €	7.00 €
1	ARDUINO-NANO-3.0	M1	ebay	4.49 €	4.49 €
				COSTE TOTAL	75.40 €

BMS V11 PLACA MAESTRA					
Cantidad	Descripción	En el PCB	Mouser Part Number	Precio unitario	Coste total
13	LED	INT, LED1, LED2, LED3, LED4, LED5, LED6, LED7, LED8, LED9, LED10, RXCAN, TXCAN	710-156120VS75000	0.16 €	2.12 €
1	MCP2515	MCP2515	579-MCP2515-I/SO	1.64 €	1.64 €
1	0.1uF	C19	80-C1206C104K5R7210	0.09 €	0.09 €
1	0.5A	F1	652-SF-1206HI050M-2	0.72 €	0.72 €
3	1.2k	R13, R18, R22	71-CRCW1206-1.2K-E3	0.09 €	0.27 €
4	10k	R4, R6, R7, R8	71-CRCW120610K0JNEAC	0.09 €	0.36 €
1	10uF	C18	74-293D106X96R3A2TE3	0.26 €	0.26 €
1	120	R10	71-CRCW1206120RFKEAC	0.09 €	0.09 €
1	16MHz	Q2	73-XT49S1600-20	0.44 €	0.44 €
1	17k	R9	71-CRCW1206-16.9K-E3	0.09 €	0.09 €
6	190	R1, R2, R3, R5, R19, R23	667-ERJ-8ENF1910V	0.09 €	0.54 €
3	1N4004	D1, D2, D3	621-1N4004	0.17 €	0.51 €
3	1k	R43, R29, R30	603-RT1206FRE0771K5L	0.16 €	0.49 €
2	22pF	C5, C7	710-885012008011	0.10 €	0.20 €
1	22uF	C16	667-EEE-FK1H220P	0.44 €	0.44 €
3	310	R14, R16, R17	71-CRCW1206-309-E3	0.09 €	0.27 €
2	3k	R27, R28	667-ERJ-8ENF3001V	0.07 €	0.14 €
1	4050D	IC8	595-CD74HC4050M96	0.42 €	0.42 €
3	47k	R12, R15, R21	71-CRCW1206-47K-E3	0.09 €	0.27 €
3	BC337	T1, T2, T3	833-BC337-25-AP	0.16 €	0.49 €
1	BLUETOOTHHC05	U$2	ebay	5.78 €	5.78 €
3	CNY17F	OK1, OK2, OK3	782-CNY17-1.	0.59 €	1.76 €
1	LP298XS	IC12	595-LP2985-30DBVR	0.46 €	0.46 €
1	MCP2551SO8	MCP2551	579-MCP2551T-I/SN	0.92 €	0.92 €
1	MICROSD	X1	517-2908-05WB-MG	3.39 €	3.39 €
13	MOLEX-2PIN-436500200	C1, C2, C3, C4, C6, C12, C14, C15, C20, C21, C23, C24, C26	538-43650-0200	0.82 €	10.65 €
3	MOLEX-3PIN-436500300	C8, C22, C25	538-43650-0300	0.94 €	2.81 €
2	MOLEX-4PIN-436500400	C13, C17	538-43650-0400	1.00 €	2.00 €
13	MOLEX-2PIN-436450200	-	538-43645-0200	0.29 €	3.74 €
3	MOLEX-3PIN-436450300	-	538-43645-0300	0.34 €	1.03 €
2	MOLEX-4PIN-436450400	-	538-43645-0400	0.41 €	0.81 €
43	Conectores crimpado	-	538-43030-0001-CT	0.07 €	2.80 €
3	SPU01M-12	PS1, PS2, PS3	709-SPU01M-12	2.75 €	8.25 €
1	TSR-1-2450	U$10	495-TSR1-2450E	3.56 €	3.56 €
1	BLUETOOTHHC05	U$2	ebay	5.78 €	5.78 €
1	OLED_0.96_I2C_WITHOUT_HOLES	U$1	ebay	7.00 €	7.00 €
1	ARDUINO-NANO-3.0	M2	ebay	4.49 €	4.49 €
3	2.2k	R11, R20, R24	EN EL TALLER	- €	- €
2	10k THT	R25, R26	EN EL TALLER	- €	- €
				COSTE TOTAL	75.06 €