UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

DISEÑO E IMPLEMENTACIÓN DE UN HOLTER PARA LA MONITORIZACIÓN DEL ELECTROCARDIOGRAMA Y LA RESPIRACIÓN DE UN ANIMAL

AUTOR: Jesús Limens Pinaque

TUTOR: Julio Gomis-Tena Dolz

COTUTORA: Irene del Canto Serrano

Curso Académico: 2020-21

A Irene, que confió en mí desde el primer momento y me brindó la magnífica oportunidad de descubrir el campo de la Ingeniería Biomédica. Eres una investigadora, una profesora y una persona increíble.

A Julio, que ha conseguido sacar lo mejor de mí mismo y me ha brindado su ayuda cuando verdaderamente la necesitaba. Has demostrado que eres un ingeniero electrónico brillante.

Muchas gracias a ambos por regalarme un futuro que me apasiona. La Universitat Politècnica de València tiene la inmensa suerte de teneros a los dos.

A mis amigos, que han hecho de esta experiencia una de las mejores decisiones de mi vida. Ya sabéis que en Galicia tendréis siempre una casa.

A mis padres, que nunca dudaron de mí. Estoy muy orgulloso de vosotros.

Y por último, a Valencia, una ciudad maravillosa que me recibió con los brazos abiertos y que jamás me hizo sentir solo.

I. MEMORIA

RESUMEN

En la actualidad el uso de animales en investigación es una práctica muy extendida que demanda de equipos de monitorización especiales para estos. En este sentido, dispositivos que registren la información biológica de forma fiable y precisa, son de gran utilidad para el diagnóstico, tratamiento o prevención de estas enfermedades en seres humanos.

Las patologías cardíacas son la principal causa anual de fallecimiento en Europa y en España, lo que convierte en esencial la investigación en electrofisiología básica con el fin de dilucidar los mecanismos responsables de la alta mortalidad. Los modelos animales, tales como el conejo, son de uso frecuente en el estudio de las patologías cardíacas por su similitud con el sistema cardiovascular humano. Sin embargo, la disponibilidad de sistemas electrónicos de monitorización adaptados a su tamaño y condiciones es escasa.

En base a estas necesidades, se plantea en el presente TFM el diseño de un sistema electrónico que monitorice el electrocardiograma y la respiración de un animal de pequeño tamaño como es el conejo. Se garantizará su correcto funcionamiento respetando todas las limitaciones y las demandas de precisión, autonomía y dimensión. Mediante el uso de tan solo dos electrodos y de componentes altamente eficientes se logrará alcanzar dicho objetivo.

Para ello, se realizará la selección de los componentes electrónicos que cumplan con las necesidades de tamaño y funcionalidad requeridas, se realizarán diversos diseños de esquemáticos del dispositivo y se analizarán las posibles mejoras hasta obtener el definitivo. A continuación, se procederá a la elaboración del diseño de la PCB (placa de circuito impreso) siguiendo unas directrices establecidas que asegurarán la calidad de la medida y el acceso sencillo a los elementos. Una vez finalizado el diseño, se procederá a la fabricación y montaje de la placa, en el laboratorio mediante técnicas de soldadura. Finalmente, se introducirá el código de programación en el microcontrolador y se realizarán las pruebas necesarias para verificar su correcto funcionamiento.

En resumen, mediante el diseño del dispositivo objetivo del presente trabajo, se dispondrá de un dispositivo de pequeño tamaño con mayor adaptación al animal, de mayor precisión, el cual supondrá un ahorro de tiempo y recursos para la investigación.

PALABRAS CLAVE ECG, Respiración, Monitorización, Conejos, Investigación, PCB

RESUM

En l'actualitat l'ús d'animals en investigació és una pràctica molt estesa que demanda d'equips de monitorització especials per a estos. En este sentit, dispositius que registren la informació biològica de forma fiable i precisa, són de gran utilitat per al diagnòstic, tractament o prevenció d'estes malalties en sers humans.

Les patologies cardíaques són la principal causa anual de defunció a Europa i a Espanya, la qual cosa convertix en essencial la investigació en electrofisiologia bàsica a fi de dilucidar els mecanismes responsables de l'alta mortalitat. Els models animals, com ara el conill, són d'ús freqüent en l'estudi de les patologies cardíaques per la seua similitud amb el sistema cardiovascular humà. No obstant això, la disponibilitat de sistemes electrònics de monitorització adaptats a la seua dimensió i condicions és escassa.

Basant-se en estes necessitats, es planteja en el present TFM el disseny d'un sistema electrònic que monitoritze l'electrocardiograma i la respiració d'un animal de dimensió reduïda com és el conill. Es garantirà el seu funcionament correcte respectant totes les limitacions i les demandes de precisió, autonomia i dimensió. Per mitjà de l'ús de tan sols dos elèctrodes i de components altament eficients s'aconseguirà aconseguir dita objectiva.

Per a això, es realitzarà la selecció dels components electrònics que complisquen amb les necessitats de grandària i funcionalitat requerides, es realitzaran diversos dissenys d'esquemàtics del dispositiu i s'analitzaran les possibles millores fins a obtindre el definitiu. A continuació, es procedirà a l'elaboració del disseny de la PCB (placa de circuit imprés) seguint unes directrius establides que asseguraran la qualitat de la mesura i l'accés senzill als elements. Una vegada finalitzat el disseny, es procedirà a la fabricació i muntatge de la placa, en el laboratori per mitjà de tècniques de soldadura. Finalment, s'introduirà el codi de programació en el microcontrolador i es realitzaran les proves necessàries per a verificar el seu funcionament correcte.

En resum, per mitjà del disseny del dispositiu objectiu del present treball, es disposarà d'un dispositiu de dimensió reduïda amb major adaptació a l'animal, de més precisió, el qual suposarà un estalvi de temps i recursos per a la investigació.

PARAULES CLAU ECG, Respiració, Monitorització, Conills, Investigació, PCB

ABSTRACT

Currently the use of animals in research is a widespread practice that demands special monitoring equipment for them. In this sense, devices that record biological information in a reliable and accurate way are very useful for the diagnosis, treatment or prevention of these diseases in humans.

Cardiac diseases are the main annual cause of death in Europe and Spain, which makes research into basic electrophysiology essential in order to elucidate the mechanisms responsible for high mortality. Animal models, such as the rabbit, are frequently used in the study of cardiac pathologies because of their similarity to the human cardiovascular system. However, the availability of electronic monitoring systems adapted to their size and conditions is scarce.

Based on these needs, the present TFM proposes the design of an electronic system to monitor the electrocardiogram and breathing of a small animal such as the rabbit. Its correct functioning will be guaranteed, respecting all the limitations and demands of precision, autonomy and size. By using only two electrodes and highly efficient components this objective will be achieved.

To this end, the selection of electronic components that meet the size and functionality requirements will be made, various schematic designs of the device will be made, and possible improvements will be analyzed until the definitive one is obtained. Then, the PCB (printed circuit board) design will be developed following established guidelines that will ensure the quality of the measurement and easy access to the elements. Once the design has been completed, the board will be manufactured and assembled in the laboratory using soldering techniques. Finally, the programming code will be introduced in the microcontroller and the necessary tests will be carried out to verify its correct operation.

In summary, by designing the target device of this work, a small device will be available with greater adaptation to the animal, with greater precision, which will save time and resources for research.

KEY WORDS ECG, Breathing, Monitoring, Rabbits, Research, PCB

Documentos incluidos en este TFM:

- I. Memoria
- II. Presupuesto
- III. Planos
- IV. Anexos

ÍNDICE

I. Memoria

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

1. INTRODUCCIÓN

1.1 OBJETIVOS

El principal objetivo del presente TFM es el diseño de un sistema electrónico que monitorice el electrocardiograma y la respiración de un animal de pequeño tamaño, en nuestro caso de un conejo. Se garantizará su correcto funcionamiento respetando todas las limitaciones y las demandas de precisión, autonomía y dimensión. Mediante el uso de tan solo dos electrodos (el mínimo posible) y de componentes altamente eficientes se logrará alcanzar dicho objetivo.

Para alcanzar este diseño se han propuesto los siguientes objetivos específicos que conforman el trabajo:

- Selección de elementos electrónicos adecuados y estudio de su funcionamiento y sus características.
- Diseño del esquemático del sistema.
- Diseño de la PCB (placa de circuito impreso) y transformación del esquema en un circuito físico y real.
- Montaje del prototipo con todos los componentes en la PCB.
- Programación el microcontrolador y puesta en marcha del dispositivo.
- Comprobación del correcto funcionamiento del dispositivo Holter desarrollado en animales y presentación de las posibles mejoras.

1.2 JUSTIFICACIÓN

1.2.1 Médica

El conejo es un animal de experimentación ampliamente empleado en los estudios de electrofisiología cardíaca, y concretamente, en el estudio de arritmias post-infarto. El presente TFM surge de un modelo experimental de infarto crónico desarrollado en conejo, en el que la mortalidad es elevada y los motivos de esta se desconocen en muchos casos, ya que no existe una monitorización del ritmo cardíaco durante las horas nocturnas. Por ese motivo, surge la idea del diseño de un dispositivo Holter para animal de pequeño tamaño, capaz de recopilar los datos necesarios con el fin de realizar un estudio posterior de las señales adquiridas, lo que conllevaría una mejora en el modelo, además de un ahorro de tiempo y recursos empleados en el proyecto de investigación.

1.2.2 Académica

Este proyecto abarca los 12 ECTS que se necesitan superar para obtener el título de Ingeniero Industrial de la Universidad Politécnica de Valencia. Se requieren también unos conocimientos concretos en campos como el de la salud o la electrónica, los cuales he ido obteniendo a medida que he realizado el proyecto.

1.3 **CONTEXTO**

Las enfermedades cardiovasculares son la principal causa de muerte en el mundo. La Organización Mundial de la Salud (OMS) estima que cada año un tercio de los fallecimientos son producidos por esta causa, siendo el número total de alrededor de 18 millones de personas. En España, la distribución de las causas de mortalidad del año 2013 se presenta en la Figura 1. Datos publicados por Instituto Nacional de Estadística (INE).

Actualmente, se está incrementando la inversión en la investigación de este campo, debido a que, aunque la mortalidad por enfermedad cardiovascular ha descendido en los últimos años, la prevalencia no ha dejado de crecer y se espera un incremento de ésta debido al envejecimiento de la población y al incremento de enfermedades tales como la obesidad y la diabetes. En este sentido, la investigación de los mecanismos responsables de las muertes asociadas a las patologías cardiacas es esencial para el desarrollo de estrategias que permitan una correcta prevención, un diagnóstico precoz y la reducción de la mortalidad asociada.

El uso de animales de experimentación para la investigación cardiovascular está muy extendido y se necesitan recursos de diversa índole para su correcta manipulación y control. En un futuro, se estima que estos equipos sean muy demandados.

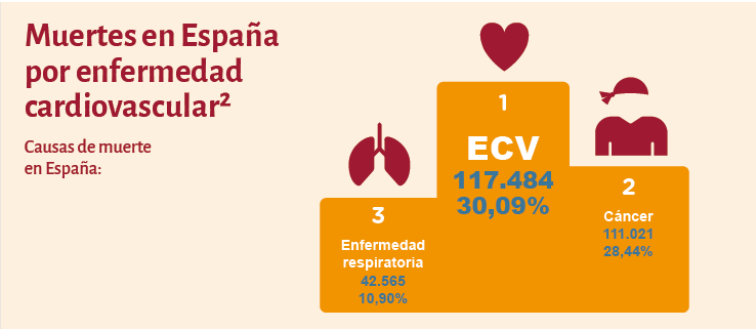

Figura 1 Causas de muerte en España 2013 (Fuente: Instituto Nacional de Estadística)

1.4 **FUNDAMENTOS TEÓRICOS**

1.4.1 CONEJO

El conejo es un mamífero que alcanza como máximo los 50cm de longitud y los 2,5 Kg de peso. Tienen una alta capacidad reproductora, siendo esta característica una de las principales razones que les convierte en un animal muy empleado en la investigación. Asimismo, su anatomía y electrofisiología cardíaca son similares a la humana (referencia: ¿Panfilov AV. Is heart size a factor in ventricular fibrillation? Or how close are rabbit and human hearts? Heart Rhythm 2006;3:862-864.).

Su anatomía se muestra en la Figura 2 y como podemos comprobar, el corazón se sitúa cercano a la cabeza, en la parte inferior. Está protegido por el esternón (se apoya en él) y los pulmones. Es idéntico al de los humanos en cuanto a anatomía y funcionamiento, difiriendo en la pulsación. En promedio, la pulsación cardíaca del conejo se encuentra entre 120 y 150 latidos por minuto, un 150% más rápidos que la pulsación del ser humano. Son animales con el corazón muy delicado, al ser una presa de los depredadores viven en constante alerta, está en su naturaleza. Este es uno de los principales motivos de su uso en investigación de problemas cardiovasculares. [2]

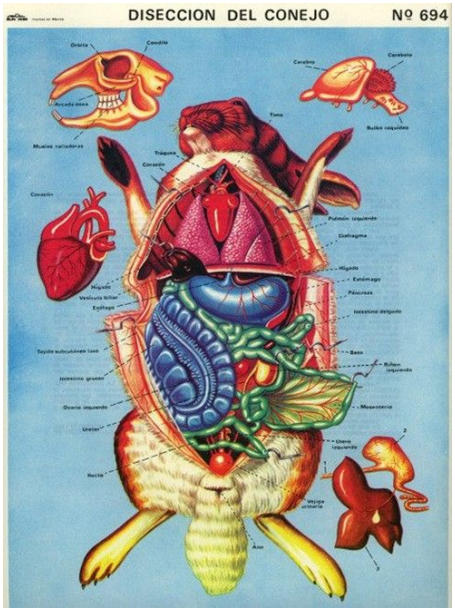

Figura 2 Anatomía del conejo (Fuente: Imagen de una clase de disección de conejos en Méjico, sin fecha e impreso por la empresa Sun Rise)

1.4.2 CORAZÓN

El corazón (Figura 3) es el órgano encargado de hacer circular la sangre por todo el cuerpo. Su funcionamiento es similar al de dos bombas en paralelo que operan a velocidad y presión variable según el momento. [3] Está compuesto de cuatro cámaras:

- Aurícula derecha
- Aurícula izquierda
- Ventrículo derecho
- Ventrículo izquierdo

Las aurículas se sitúan en la parte superior y los ventrículos en la inferior. También existen cuatro válvulas en el corazón, cada una con su función específica:

- Válvula tricúspide (aurícula derecha – ventrículo derecho)
- Válvula mitral (aurícula izquierda – ventrículo izquierdo)
- Válvula pulmonar (ventrículo derecho – arteria pulmonar)
- Válvula mitral (ventrículo izquierdo – arteria aorta)

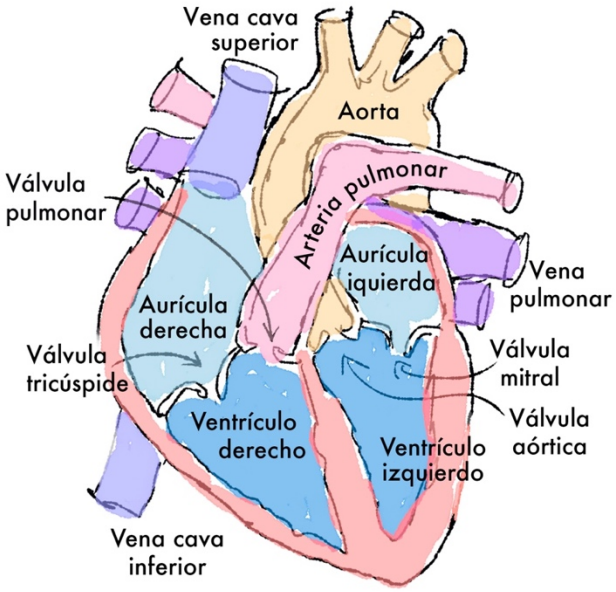

Figura 3 Esquema del corazón (Fuente: Brainly)

La sangre realiza el recorrido que se detalla en la Figura 4, donde aparecen descritas las fases del bombeo cardíaco. A modo general, la sangre entra por las venas cavas a las aurículas (diástole) y sale de los ventrículos por las arterias a todo el cuerpo (sístole). Las cuatro venas pulmonares que desembocan en la aurícula izquierda transportan sangre oxigenada procedente de los pulmones en cada ciclo.

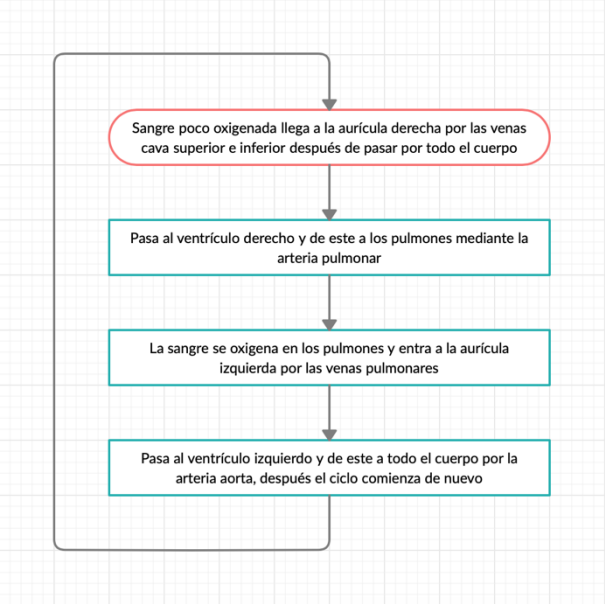

Figura 4 Recorrido de la sangre durante el ciclo cardiaco

Todo este proceso se produce mediante impulsos eléctricos y por tanto hay que hablar de la contracción de los músculos. Empezaremos por el músculo cardíaco que, a diferencia de otros, es un órgano autónomo capaz de iniciar la excitación eléctrica, y no necesita el control consciente del cerebro para contraerse. A continuación, se exponen las siguientes estructuras del sistema de conducción cardiaco:

- Nódulo sinoauricular (Nódulo sinusal)
- Nódulo auriculoventricular (Nódulo de Aschoff-Tawara)
- Fascículo auriculoventricular (Haz de His)
- Fibras de Purkinje

Estas generan y distribuyen el impulso por el corazón, que en resumen es lo que entendemos por pulso cardíaco, el cual queda reflejado en la señal de ECG o electrocardiograma. En la Figura 5 se expone el circuito seguido por este impulso.

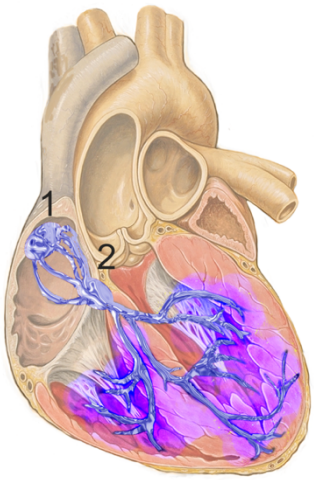

Figura 5 Representación del sistema de conducción cardiaco (1 - Nodo sinoauricular, 2 - Nodo auriculoventricular.) (Fuente: Wikipedia)

1.4.3 ELECTROCARDIOGRAMA

El electrocardiograma (ECG) (Figura 6) es la representación gráfica de la actividad eléctrica del corazón que consiste en representar variaciones de voltaje en función del tiempo. Es de gran utilidad tanto para monitorizar la actividad eléctrica como para diagnosticar patologías cardíacas, y es por ello por lo que su amplio uso es de gran importancia. A continuación, se detalla la secuencia de activación cardíaca y su traducción a las ondas visualizadas en el electrocardiograma. [4]

En primer lugar, el nódulo sinusal, que funciona como marcapasos natural, genera el impulso eléctrico que se transmite a todo el corazón por el sistema de conducción, a partir de las células auriculares hasta las células ventriculares. El estímulo sinusal despolariza las aurículas, dando lugar a la primera oscilación del gráfico, la onda P. Se caracteriza por una duración media de 0,1 segundos y por tener una forma redondeada.

Al llegar al nódulo auriculoventricular la onda de despolarización se ralentiza para facilitar la coordinación con el paso de la sangre. Aquí apenas se genera voltaje. A continuación, se despolariza el septo interventricular, que es la pared que separa ambos ventrículos. Esto genera la onda Q, la cual no es fundamental y en ocasiones puede no ser muy visible.

Al llegar a la parte distal del nodo, la onda de despolarización se acelera y entra en el haz de His, continuando a izquierda y a derecha por las dos ramas del haz. La despolarización ventricular comienza simultáneamente en 3 puntos: las regiones de inserción de los haces supero-anterior, infero-posterior y medio-septales de la rama izquierda. Una vez iniciada, comienza la despolarización de la gran masa ventricular izquierda y derecha, siendo la suma de los dos procesos la que produce la onda R, valor pico del ECG que suele rondar los 1,6 mV en humanos. La despolarización termina en las zonas menos ricas en fibras de Purkinje: las zonas basales y septales altas, resultando la onda S. Por tanto, el complejo QRS corresponde a la corriente eléctrica que causa la contracción de los ventrículos derecho e izquierdo

(despolarización ventricular), la cual es mucho más potente que la de las aurículas e implica a una mayor masa muscular, produciendo de este modo una mayor deflexión en el electrocardiograma.

Finalmente, los ventrículos están polarizados (cargados positivamente) y se produce una ligera pausa hasta que llega la repolarización de los ventrículos, representada por la onda T, con forma asimétrica y duración general de 0,2 segundos.

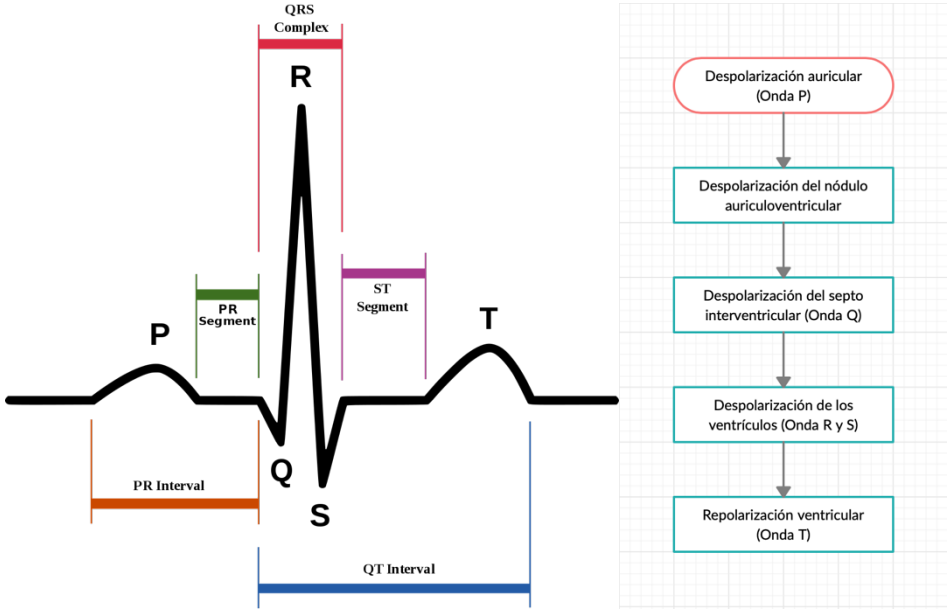

Figura 6 Señal ECG (Fuente: Universitat de Barcelona)

Como explicamos antes, el corazón del conejo suele operar a velocidades superiores a las humanas y los valores de voltaje producidos son inferiores. A la hora de la toma de datos debemos utilizar un rango de medida y frecuencia adecuados.

Para obtener el ECG, se emplean electrodos conectados estratégicamente en determinadas partes del cuerpo, estos producen las llamadas derivaciones que miden la diferencia de potencial entre puntos. [5] Existen tres grandes grupos de estas (Figura 7):

- Derivaciones estándar
- Derivaciones aumentadas
- Derivaciones precordiales

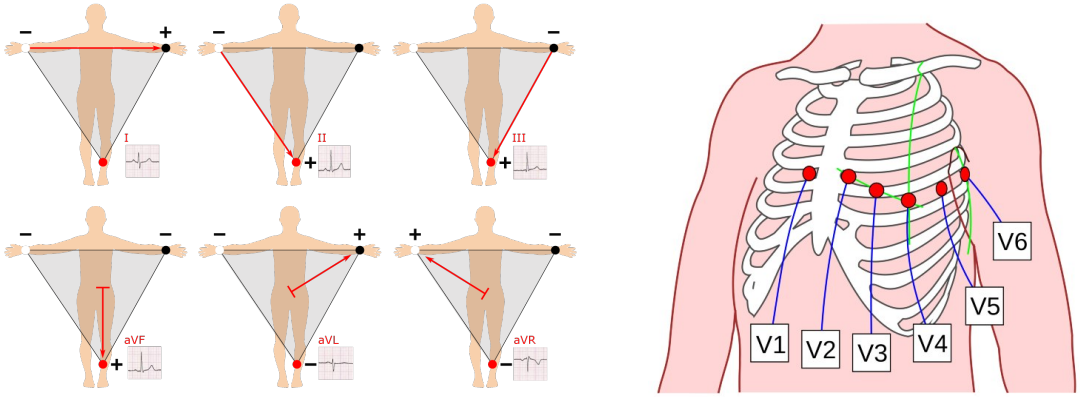

Figura 7 Representación de las derivaciones estándar, aumentadas y precordiales (Fuente: klipartz)

Pasamos ahora a definir cada una de ellas:

- Derivación I – Brazo derecho (-) con brazo izquierdo (+)
- Derivación II – Brazo derecho (-) con pie izquierdo (+)
- Derivación III – Brazo izquierdo (-) con pie izquierdo (+)
- Derivación aVR – Brazo derecho (+) con promedio pie izquierdo / brazo izquierdo (-)
- Derivación aVL – Brazo izquierdo (+) con promedio pie izquierdo / brazo derecho (-)
- Derivación aVF – Pie izquierdo (+) con promedio brazo izquierdo / brazo derecho (-)
- Derivaciones precordiales – Tórax (+) con promedio pie izquierdo / brazo derecho / brazo izquierdo (-)

Cuando se mide con respecto a un valor promedio, en realidad la intención es la de hacerlo con un electrodo situado en el infinito. Aproximando así se obtienen valores aceptables. En concreto, en las derivaciones precordiales se compara con el llamado Central Terminal de Wilson (CTW) (Figura 8), valor de referencia constante en un cuerpo.

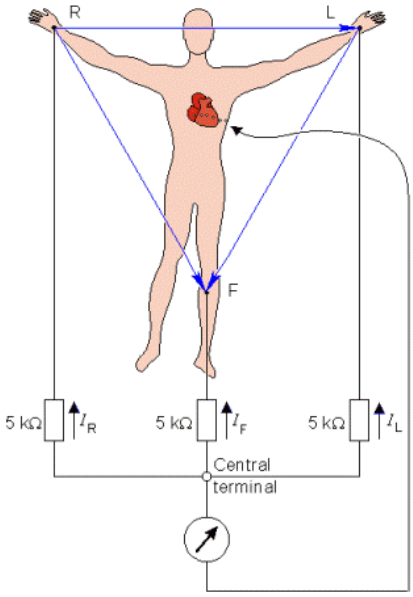

Figura 8 Central Terminal de Wilson

1.4.4 DETECCIÓN DEL RITMO CARDÍACO

La detección del ritmo cardiaco a partir de la señal de ECG se realiza mediante la detección y localización de los complejos QRS y la medida del intervalo RR. En la Figura 9 se representa una señal de electrocardiograma donde se observan 2 complejos QRS y el intervalo RR entre ambos complejos.

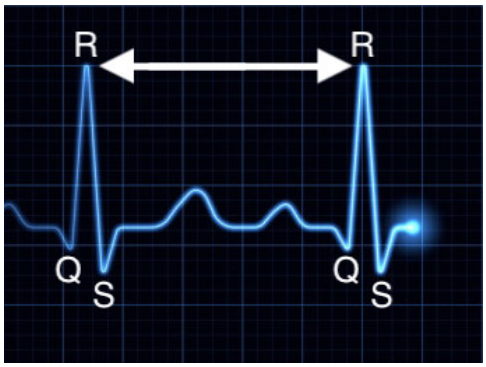

Figura 9 Complejo QRS e intervalo RR

Con el seguimiento del intervalo RR podremos observar cambios en el ritmo cardíaco y detectar arritmias o anomalías fuera de los parámetros normales como cardiopatías dependientes de estas variables. La detección de los intervalos RR permite estimar si el funcionamiento del corazón es en ritmo normal o sinusal (señal periódica y con un ritmo de frecuencia media), o, por el contrario, si se produce una alteración del mismo: aceleración de la activación cardiaca manteniendo la periodicidad (taquicardia), aceleración de la actividad presentando un patrón irregular (fibrilación), o enlentecimiento a frecuencias muy bajas (bradicardia).

1.4.5 RESPIRACIÓN

De igual forma que con el electrocardiograma, nuestro dispositivo también registrará la respiración del animal, ya la información que ésta nos proporciona es de gran utilidad. El reto es hacerlo de una forma cómoda y con tan solo dos electrodos conectados. Para ello utilizaremos un método muy eficaz basado en la variación de la impedancia en la inspiración y la espiración. Por uno de los electrodos se inyecta una pequeña corriente y esta encontrará mayor o menor resistencia dependiendo de la fase de la respiración. El espectro producido será similar al de la Figura 10.

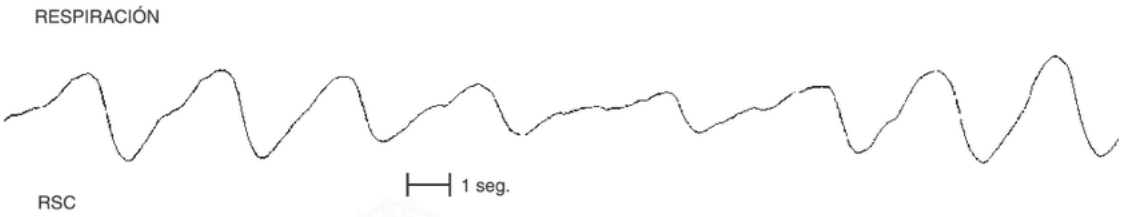

Figura 10 Onda tipo de una respiración normal

A mayor volumen de aire en el tórax, mayor amplitud de onda. Las subidas por tanto representan las inspiraciones, y las bajadas las espiraciones. Los conejos respiran entre sesenta y noventa veces por minuto, una frecuencia bastante menor que la de su pulsación. Será por ello importante ajustar bien el muestreo para ambos casos.

1.4.6 ELECTRODOS

Los electrodos son los encargados de transformar las corrientes iónicas en eléctricas mediante el contacto directo con la piel. Por motivos de diseño, utilizaremos tan solo dos electrodos que tomarán las dos medidas demandadas, el registro de ECG y la respiración. Esto supondrá una mayor complejidad, pero reducirá las dimensiones del dispositivo, factor fundamental para su viabilidad.

Mediante un multiplexor incluido en el circuito integrado ADS1292R se conmutarán dos tareas. Una de ellas es la de inyectar una pequeña corriente por los electrodos que servirá para polarizar los transistores del amplificador, detectar si un electrodo está suelto y medir la variación de la impedancia del tórax. La otra será la de obtención de la señal de ECG sin circulación de corriente, que se servirá de la capacidad parásita entre los dos conductores (carga y descarga continua) para polarizar el amplificador. El funcionamiento de este sistema ya ha sido verificado en anteriores proyectos.

En esta parte se presentan varias dificultades:

- Errores por mal contacto entre electrodo y piel.
- Comportamiento impredecible del animal.
- Errores derivados de la polarización del electrodo.
- Error de la tensión modo común.

- Suficiente corriente de polarización de los amplificadores.

Al trabajar con conejos deberemos ajustar bien el dispositivo para que no se suelte con facilidad. Además, el propio circuito integrado rechazará el modo común y su interferencia correspondiente. En la Figura 11 podemos ver a modo resumen el modelo equivalente de la interfaz electrodo-piel. [6]

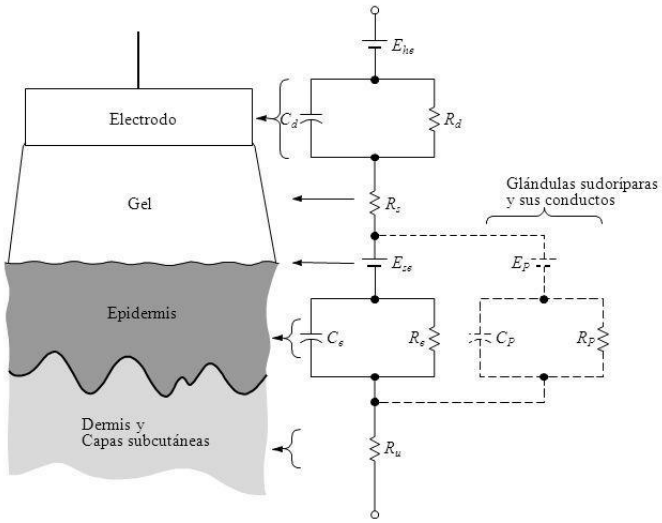

Figura 11 Circuito equivalente de la interfaz electrodo-piel (Fuente: "Protocolo para la adquisición de señales microeléctricas de los músculos inervados por los nervios ulnar, radial y medial para un a órtesis de mano" – Universidad de Santiago de Cali)

1. METODOLOGÍA DE DISEÑO

1.1 ESPECIFICACIONES INICIALES

Antes de comenzar con el diseño, existen una serie de especificaciones que nuestro dispositivo debe cumplir, descritas a continuación:

- Las dimensiones de la placa han de ser lo más reducidas posibles para su buen funcionamiento en el animal.
- El peso del dispositivo ha de ser mínimo.
- La autonomía de la batería ha de ser lo suficiente para trabajar toda la noche.
- Ha de existir una conexión entre la computadora y el dispositivo para recopilar e interpretar los datos de la memoria µSD.
- Será necesario un interruptor para facilitar su correcto uso.

1.2 ESTRUCTURA

La estructura del Holter será la habitual en dispositivos de pequeño tamaño. Por un lado, tenemos los dos electrodos que llevarán la señal eléctrica de ECG y de respiración hasta el circuito integrado ADS1292R, donde se realizará el filtrado, la amplificación y la conversión analógico digital a la señal. Los registros serán enviados al microprocesador PIC24FJ128GC006 que los organizará y almacenará en la memoria µSD. Finalmente se conectará a un ordenador mediante el puerto mini USB y se recopilarán e interpretarán los datos obtenidos. Todo esto estará alimentado siempre por la batería incorporada en la parte inferior. En la Figura 12 vemos un diagrama con el proceso descrito.

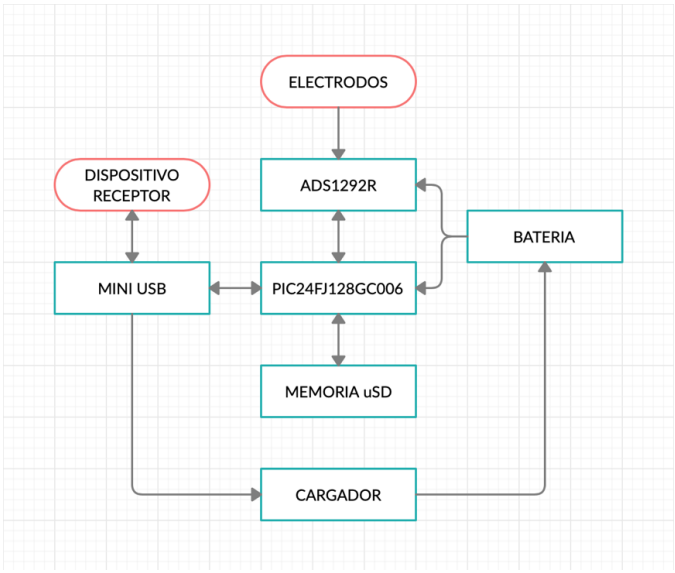

Figura 12 Diagrama del dispositivo

1.3 **COMPONENTES**

1.3.1 ADS1292R

El circuito integrado ADS1292R incorpora todas las características comúnmente requeridas para monitorizar la señal de electrocardiograma en equipos portátiles de baja potencia. Tiene un alto grado de integración y un excepcional rendimiento que se traduce en una reducción de tamaño, consumo y coste. De esta forma se posibilita el diseño de dispositivos médicos como el del presente documento. [7]

Además, el ADS1292R a diferencia de sus otras versiones ADS1291 y ADS1292, incorpora la posibilidad de monitorizar también la respiración mediante la variación de impedancia en el cuerpo, por lo que ofrece una simplificación mayor en la adquisición. En las Figuras 13 y 14 podemos ver dos imágenes del *data sheet* del componente: en la primera de ellas, el diagrama de bloques y en la segunda, la distribución de los pines.

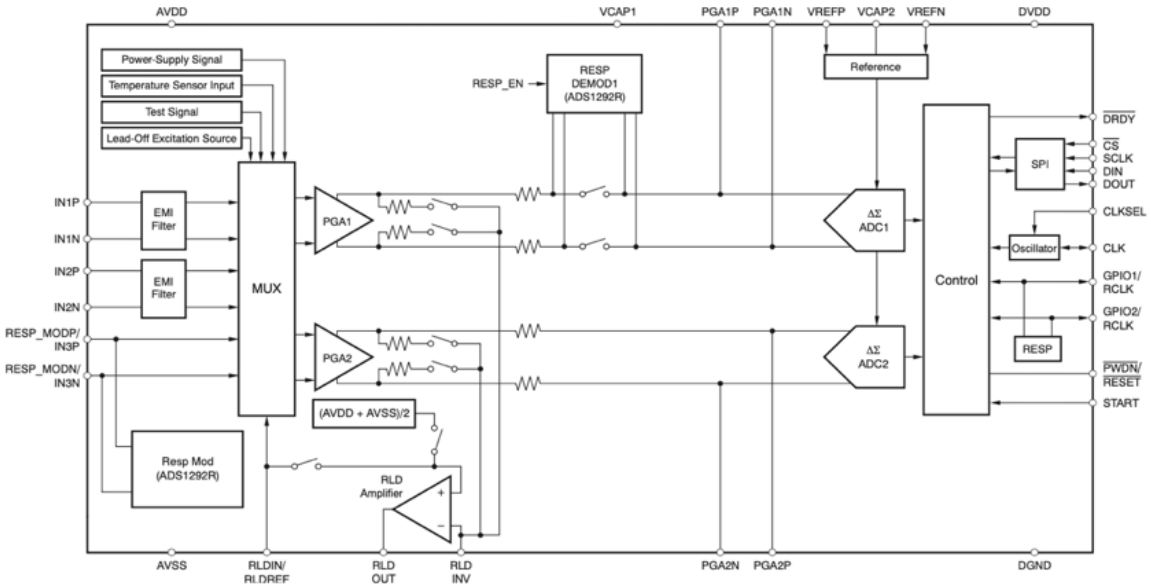

Figura 13 Diagrama de bloques del ADS1292R

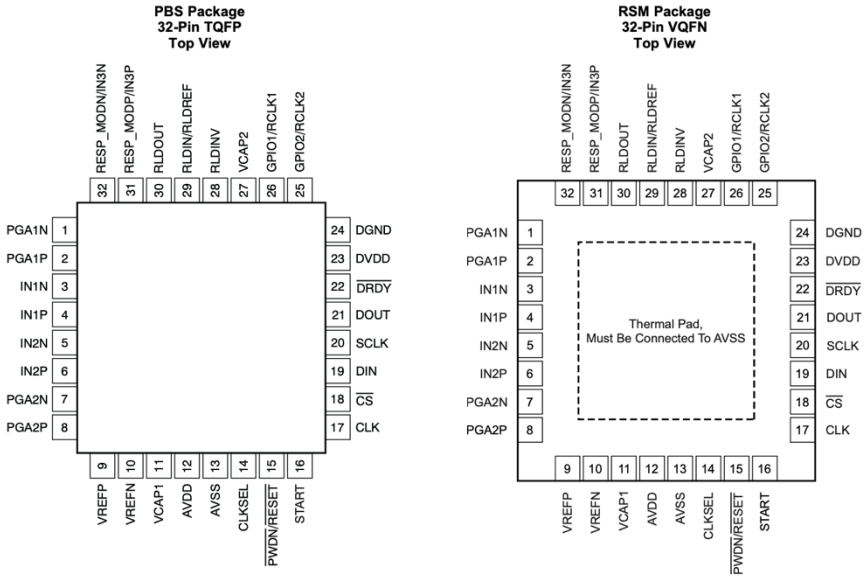

Figura 14 Distribución de los pines del ADS1292R

Una vez introducido el circuito pasaremos a explicar punto por punto su funcionamiento y las medidas adoptadas en este caso en particular.

- Adquisición de datos

El proceso de adquisición viene resumido en la Figura 15. La señal analógica se transmite por los electrodos y pasa por un amplificador diferencial de ganancia programable (PGA) cuyo valor estándar será seis (tiene otras siete posibles). Este primer paso se considera de gran importancia ya que las señales bioeléctricas se caracterizan por sus bajos niveles de tensión.

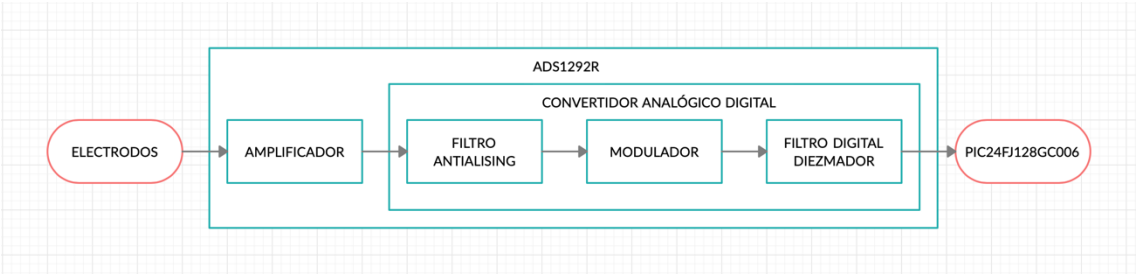

Figura 15 Esquema de funcionamiento del ADS1292R

A continuación, viene la conversión analógico digital que consta de tres subprocesos.

a) Filtro *antialising*

Filtro de paso bajo previo a la conversión para eliminar todas aquellas componentes de la señal con frecuencias superiores a la mitad de la frecuencia de muestreo que se haya programado. En el

ADSR1292R el filtro implementado para nuestro dispositivo está formado por una resistencia interna de 2kΩ y un condensador de 4,7 nF que cortocircuita los pines 1 y 2, y también los pines 7 y 8.

b) Modulador

Muestrea y cuantifica la señal, convierte una señal continua en una sucesión de valores discretos mediante la comparación con los niveles de cuantificación más próximos. Esta aproximación genera un pequeño error el cual se filtra mediante el sobremuestreo y la conformación de ruido. La primera técnica se basa en el aumento de la frecuencia de muestreo en comparación a la de Nyquist, la segunda agrupa la mayor parte de la potencia de las ondas en frecuencias altas que no demandan ningún tipo de interés.

c) Filtro diezmador

La frecuencia de la señal de salida será muy inferior a la de modulación, esta es la base del sobremuestreo. Esta reducción se conoce como diezmado. Teniendo en cuenta el mínimo recomendado para el ancho de banda del ECG, 100Hz, tenemos que la frecuencia de muestreo utilizada será de 250 muestras por segundo (*samples per second*, SPS) (>200 SPS demandados).

- Alimentación

El ADS1292R utiliza una fuente de alimentación para su parte analógica (AVDD) y otra para la digital (DVDD), ambas de 3,3V suministrados por la batería. Importante la colocación de condensadores de desacoplo entre los pines que reciben el voltaje, minimizan el ruido generado por la conmutación de las salidas del dispositivo mediante el aporte de una pequeña alimentación.

- Referencia

En un Holter habitual, la referencia tomada es el Central Terminal de Wilson (CTW) mediante la colocación de varios electrodos en el cuerpo. El problema es que, en nuestro caso, al solo disponer de dos, debemos establecer una referencia fija. Para el voltaje utilizado de 3,3V este valor es de 2,4V medido respecto a tierra analógica (AVSS). La forma de conexión de los pines 9 y 10 es similar a la de la Figura 16.

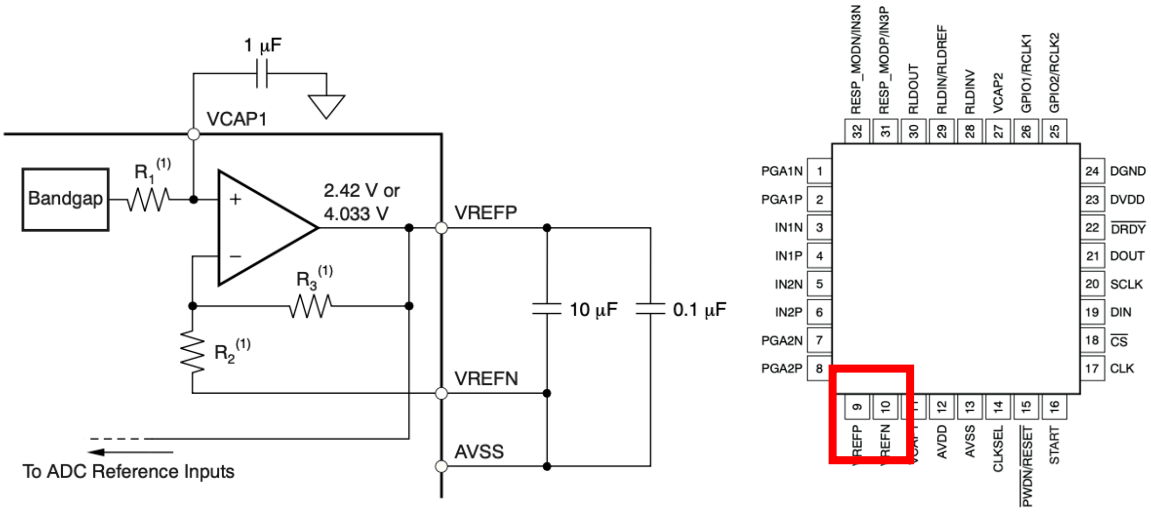

Figura 16 Diagrama de bloques de la referencia interna y localización de los pines 9 y 10 en el microcontrolador

- Reloj

Aunque en un principio se pensó en la adición de un oscilador, finalmente y para minimizar el espacio ocupado utilizaremos el reloj interno incorporado en el ADS1292R, que opera a 512 kHz. Esta opción se considera idónea para dispositivos de baja potencia portátiles que funcionan con baterías de alto rendimiento.

- Comunicación

El ADS1292R se comunicará con el microprocesador PIC24FJ128GC006 mediante los pines 16, 18, 19, 20, 21 y 22. La arquitectura de comunicación utilizada será maestro-esclavo, en la que el maestro puede enviar y recibir datos, así como generar una señal de reloj que mantiene a todos los dispositivos sincronizados. El esclavo por contra solo puede recibir información y depende del maestro para poder comunicarse. En nuestro diseño, el microprocesador actuará como maestro y el circuito integrado ADS1292R como esclavo. Esto se traduce en un estándar de comunicaciones conocido como Bus SPI, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos, controla casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj.

La comunicación constará de las cinco señales expuestas en la Figura 17. En la Figura 18 se presenta tenemos un esquema de los datos de salida.

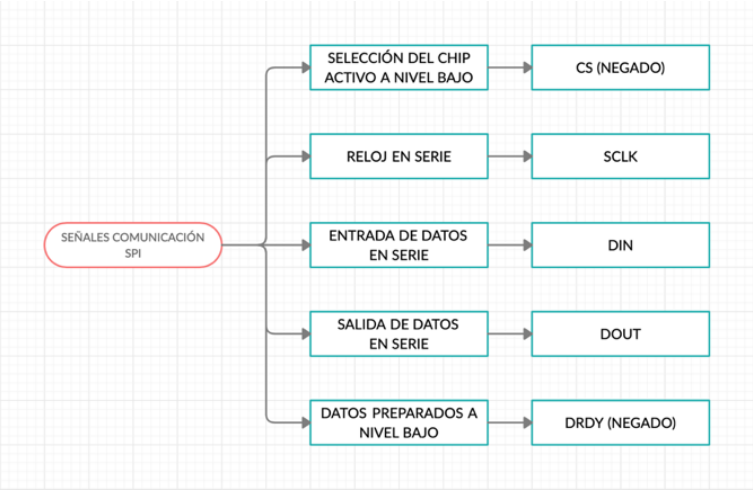

Figura 17 Señales de la comunicación SPI (Serial Peripheral Interface)

a) $\overline{\textbf{CS}}$ (chip select) Este pin controla la llegada datos, si está a nivel bajo el microprocesador recibirá la información del circuito integrado.
b) **SCLK** (serial clock) Marca la entrada/salida de datos con cada flanco de subida del oscilador del microprocesador, que se sincroniza con el del circuito integrado.
c) **DIN** (data in) (MOSI) El ADS1292R recibe información del microprocesador con cada flanco de bajada de la señal SCLK.
d) **DOUT** (data out) El ADS1292R envía información al microprocesador en cada flanco de subida de la señal SCLK.

$$Bits \text{ salida ADS1292R} = 24 \text{ bits estado} + 2 \text{ canales} * 24 \text{ bits canal} = 72 \text{ bits}$$

e) $\overline{\textbf{DRDY}}$ (data ready) Si recibe voltaje bajo (valor cercano a cero) indica que hay nuevos datos procedentes del conversor analógico digital listos para transmitirse al microprocesador. Si al contrario está a nivel alto, indica que el ADS1292R todavía está procesando los datos de los electrodos.

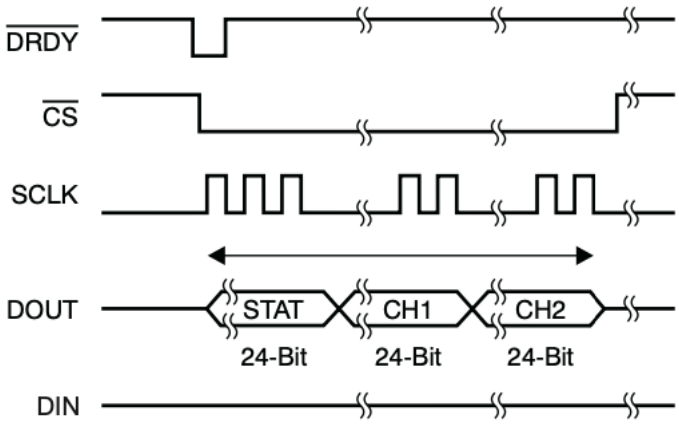

Figura 18 Datos de salida del ADS1292R

- Electrodo suelto

El ADS1292R tiene incorporado dos sistemas de detección de electrodo suelto, por corriente continua y por corriente alterna. Por simplificación utilizaremos el basado en corriente continua. Su funcionamiento es simple, se hace circular una pequeña corriente continua por los electrodos y gracias a ella podremos obtener los siguientes datos que sirven para evaluar la calidad de su colocación (Figura 19):

a) **No circula corriente** – Los electrodos no están bien colocados
b) **Sí circula corriente** – Los electrodos si están bien colocados
c) **Alta tensión** – Los electrodos están demasiado separados (circuito abierto)
d) **Baja tensión** – Los electrodos están demasiado juntos (cortocircuito)
e) **Tensión en el umbral del comparador** – La colocación y la posición es correcta

El umbral de comparador se puede programar y será lo suficientemente amplio para nuestro trabajo. En la Figura 19 vemos un diagrama de lo expuesto.

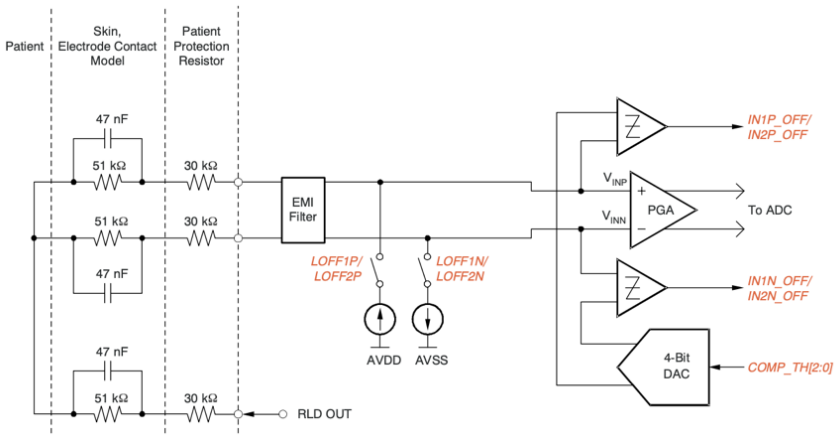

Figura 19 Diagrama de bloques para la detección del electrodo suelto

- Respiración

El ADS1292R lleva incorporado un sistema de medición de la respiración, su funcionamiento es simple, se inyecta una corriente por los electrodos y la variación de impedancia del tórax del animal produce modificaciones en la tensión, recopilando estas variaciones se obtiene el espectro de la respiración. En la Figura 20 podemos ver el diagrama de bloques implantado.

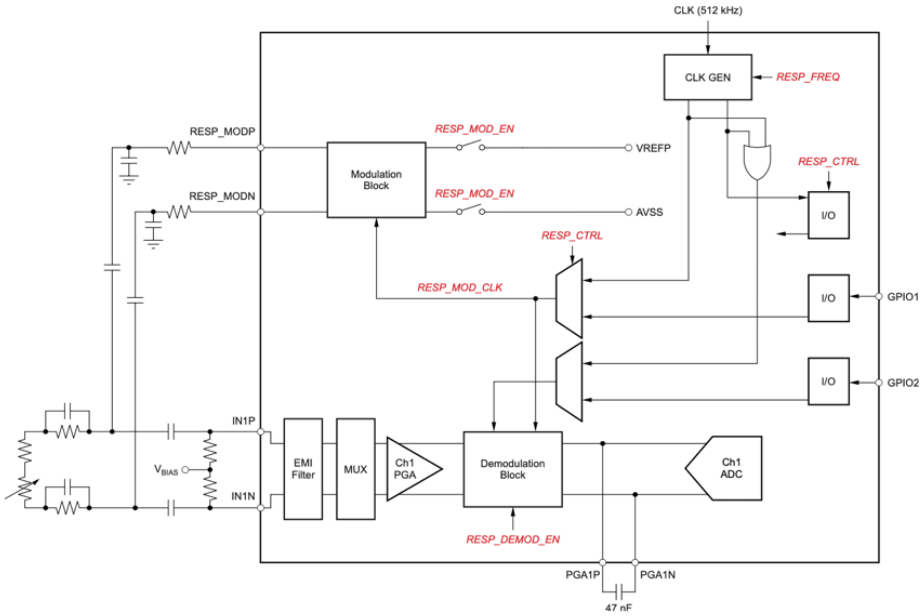

Figura 20 Diagrama de bloques del sistema de respiración

El principal inconveniente es que solo tenemos dos canales de entrada y por tanto si se decide monitorizar la respiración no podremos tomar la señal de electrocardiograma simultáneamente. Esto se soluciona con el esquema de la Figura 21, donde se conectan los dos electrodos a ambos canales, se dedica el primero a la respiración y el segundo a la señal de ECG. En la parte central izquierda se observan las dos entradas de señal, correspondientes a los electrodos *Left Arm Lead* y *Right Arm Lead*. En los extremos de la parte izquierda (arriba y abajo), se observan las dos señales referencia tanto digital como analógica. Con un multiplexor alternaremos estas dos funciones, inyectando corriente en un caso y recibiendo la señal bioleléctrica en el otro. La viabilidad de este modelo ya ha sido comprobada en anteriores proyectos, por lo que su funcionamiento será el adecuado. [8]

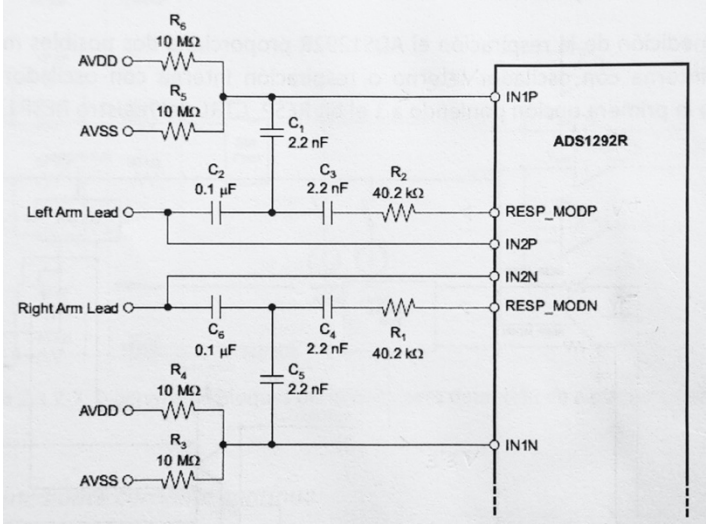

Figura 21 Circuito de entrada de datos

- Esquemático ADS1292R

Finalmente se obtiene el siguiente resultado (Figura 22).

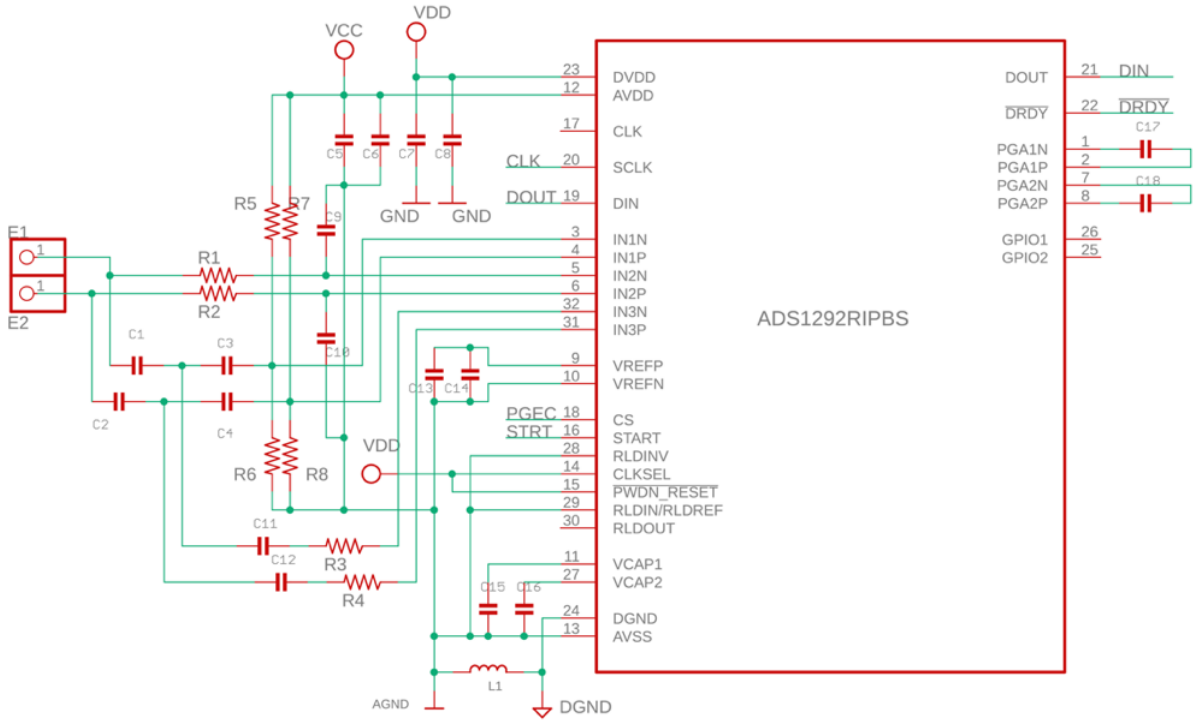

Figura 22 Esquemático final del ADS1292R

1.3.2 PIC24FJ128GC006

El microcontrolador PIC24FJ128GC006 (Figura 23) amplía las capacidades de la familia PIC24F al agregar una selección completa de periféricos analógicos avanzados a sus características digitales existentes. [9] Esta combinación, junto con sus características de ultra bajo consumo, acceso directo a memoria para periféricos, USB On-The-Go y un controlador y controlador LCD incorporado, hacen de esta familia el nuevo estándar para señal mixta en un paquete económico y de bajo consumo. Estas son algunas de sus características más notables:

- Memoria de programación flash de 128Kbytes
- 64 pines
- 29 pines programables
- Puertos de comunicación UART, SPI e Inter Integrated Circuit

Su función es la de operar el dispositivo. Recibirá órdenes por parte del programador y de los interruptores. Controlará la entrada de datos enviados por el circuito integrado ADS1292R y los recopilará en la memoria.

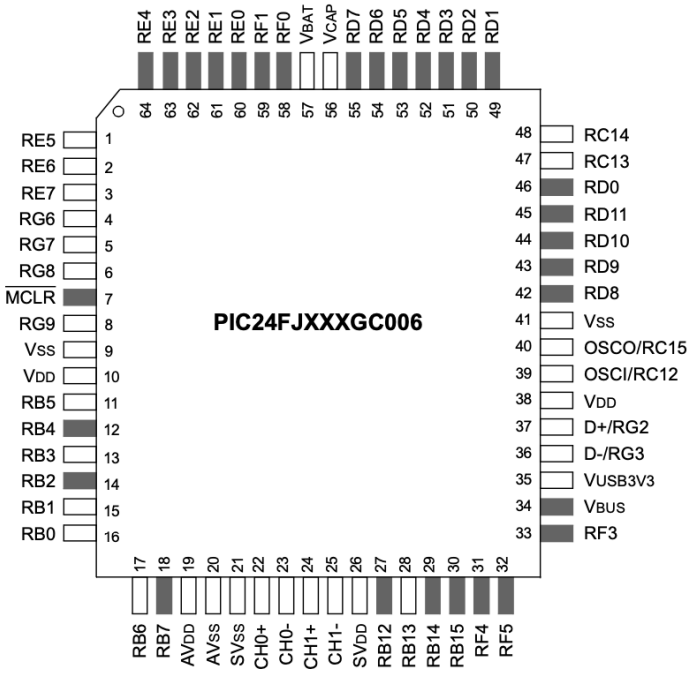

Figura 23 PIC24FJ128GC006

- Oscilador

Todos los dispositivos de la familia PIC24FJ128GCXXX ofrecen cinco opciones de osciladores: dos modos de cristal, dos modos de reloj externo, multiplicador de frecuencia Phase-Locked Loop (PLL), Fast Internal Oscillator (FRC) y Aseparate Low-Power Internal RC Oscillator (LPRC). En nuestro caso nos decantaremos por un oscilador externo que siempre estará conectado a la alimentación (Figura 24).

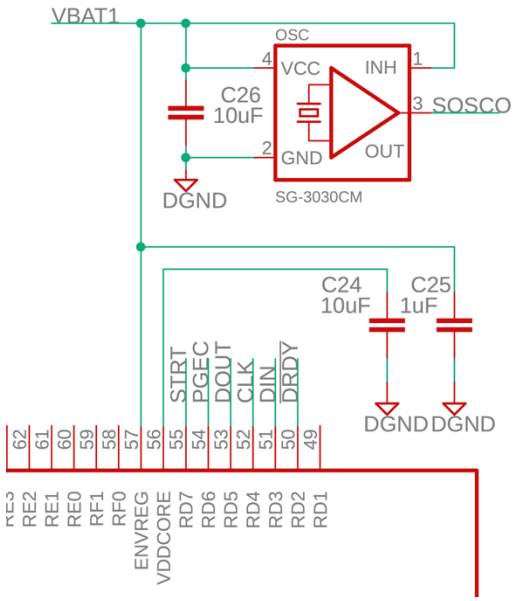

Figura 24 Oscilador externo

- Programación

Para su programación se utilizará la tecnología ICSP (In Circuit Serial Programming) cuya principal ventaja es que no requiere la extracción del microcontrolador de la placa cada vez que se decida realizar modificaciones. El cabezal implantado para ello será un Mini USB en la parte inferior del dispositivo. Se conectará mediante un cable apropiado a un dispositivo externo (ordenador) con el que se escribirá el código oportuno. En la Figura 25 se muestran los esquemáticos del puerto Mini USB y de los respectivos pines de entrada en el microcontrolador.

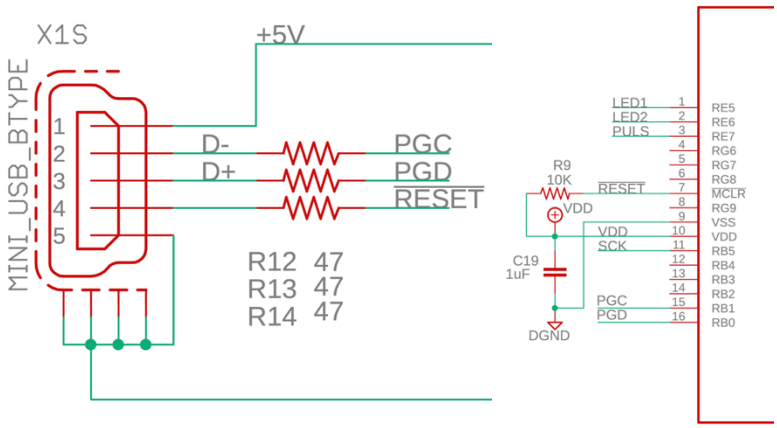

Figura 25 Tecnología ICSP

- Esquemático PIC24FJ128GC006

Basándonos en los modelos proporcionados por el data sheet (Figura 26) y nuestras especificaciones, finalmente se obtiene el resultado mostrado en la Figura 27. Entre cada pin de alimentación y la masa, se coloca un condensador lo más cercano posible al pin. Estos son conocidos como condensadores de desacoplo y sirven para dar estabilidad a la alimentación filtrando pequeños ruidos.

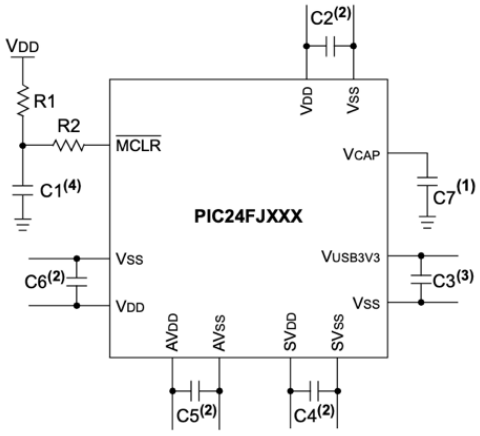

Figura 26 Modelo recomendado por el data sheet del PIC24FJ

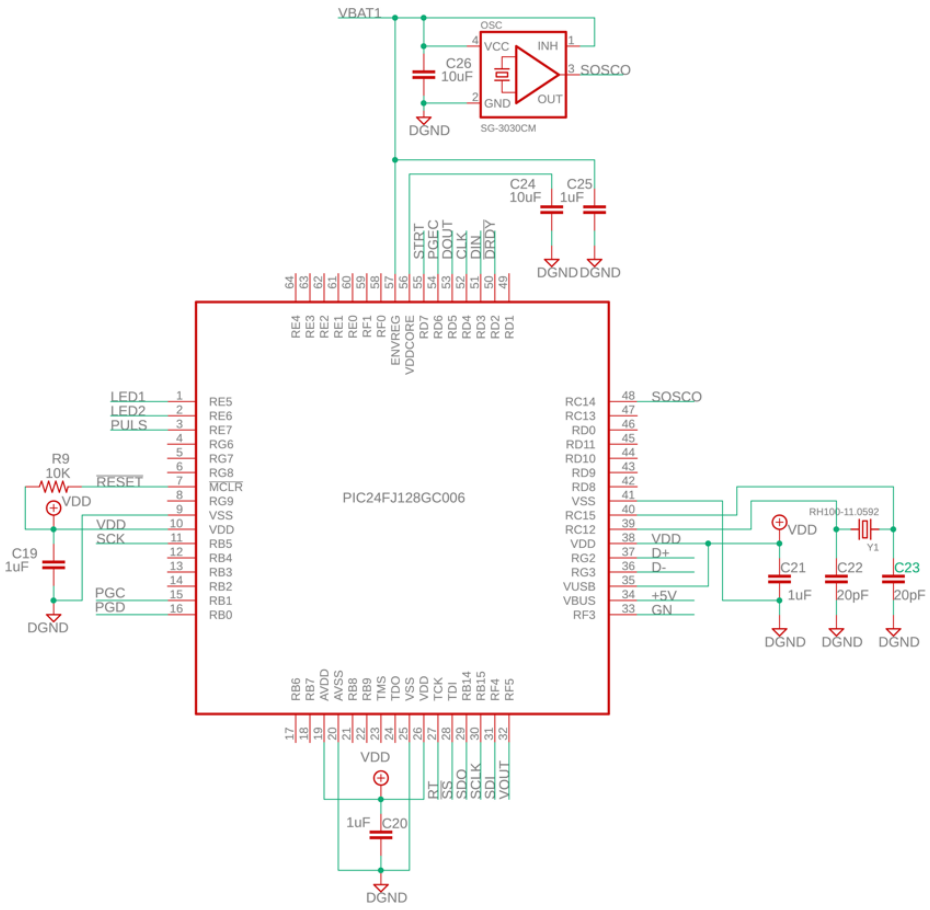

Figura 27 PIC24FJ128GC006

1.3.3 MEMORIA µSD

Para el almacenamiento de datos se utilizará una tarjeta µSD estándar. Es el método más empleado con microprocesadores/microcontroladores por su sencillez y por su amplio abanico de librerías. Su principal ventaja es que son memorias no volátiles, por lo que la información no se pierde cuando se desconecta de la fuente de alimentación, permitiendo que se puedan almacenar datos durante días sin problema. Además, destaca su pequeño tamaño, perfecta para dispositivos remotos portátiles como el desarrollado en el presente trabajo. Se obtiene el resultado mostrado en la Figura 28. [10]

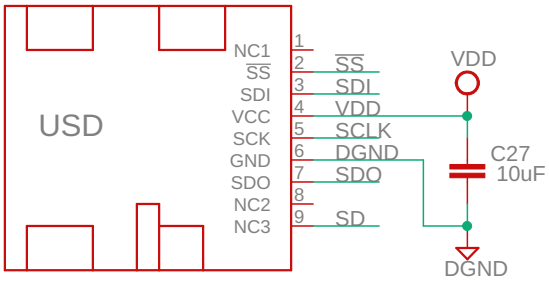

Figura 28 Memoria µSD

1.3.4 CIRCUITO DE CARGA

Para la carga de la batería se ha diseñado un circuito apto para aquellas baterías que disponen de dos y tres pines. Este circuito detectará que la batería se ha cargado completamente mediante una resistencia de 10kΩ que modifica su temperatura o mediante un cable conectado al MCP73833 que recibe una señal térmica directamente de la batería. De esta forma se evitan sobrecalentamientos y sobrecargas. Para alimentar el circuito se dispone de un puerto Mini USB que servirá tanto para cargar como para programar el Holter (Figura 29). [11]

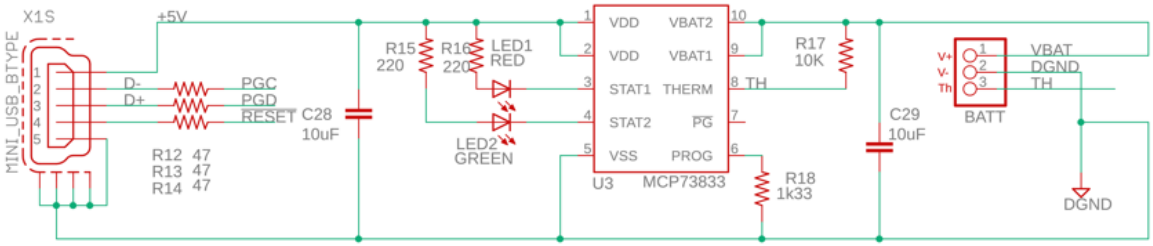

Figura 29 Circuito de carga

1.3.5 REGULADORES DE TENSIÓN

Dispondremos de tres reguladores de tensión NCP551SN30 que recibirán 3,7V de la batería (pin IN) y alimentarán todo el circuito con 3,3V nominales (pin OUT). Uno de ellos alimenta la parte analógica, otro la parte digital y el último el oscilador externo del microcontrolador. El pin EN (enable) funciona a modo de interruptor (en nuestro dispositivo siempre estarán activados si llega voltaje de la batería) del regulador y el pin GND se conecta a tierra analógica o digital dependiendo de su procedencia. Finalmente se obtiene el siguiente resultado (Figura 30). [12]

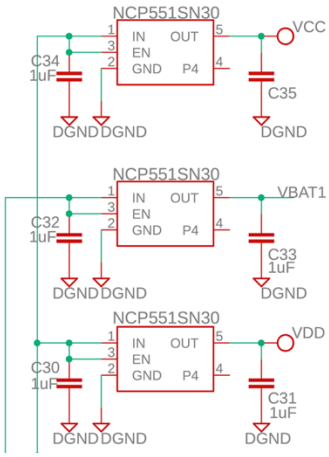

Figura 30 Reguladores de tensión

1.3.6 INTERRUPTOR

El Holter contará con un interruptor de tres posiciones (Figura 31) las cuales se describen a continuación:

- **Derecha (Holter)** – Funcionamiento en modo Holter con ECG y respiración.
- **Centro (VBat)** – Alimentación desconectada y batería dispuesta para ser cargada.
- **Izquierda (RTM)** – Funcionamiento en modo test del ADS1292R, adquiere una señal interna de calibración (onda cuadrada de amplitud 1mV y frecuencia 1Hz) con el fin de calibrar el programa de adquisición y confirmar el correcto funcionamiento).

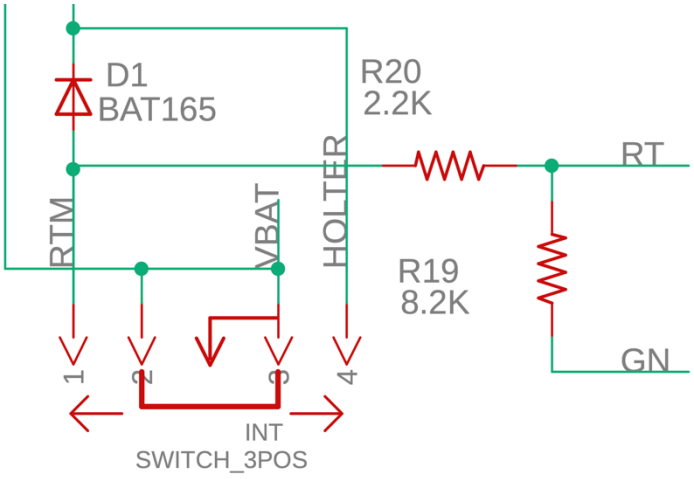

Figura 31 Interruptor

1.3.7 PULSADORES / LEDS

El aparato cuenta con dos LEDs y dos pulsadores. Los cuatro componentes pueden ser programados según nuestras necesidades provocando gran versatilidad para el usuario. En la Figura 32 se exponen estos componentes.

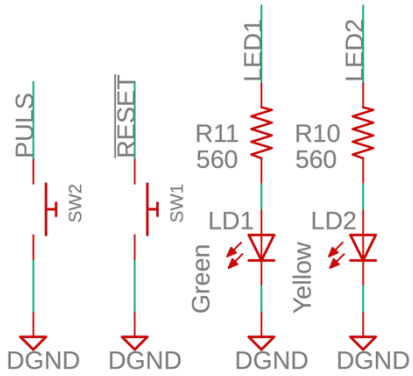

Figura 32 Esquemáticos Pulsadores / LEDs

1.3.8 ELECTRODOS

Nuestro dispositivo es compatible con casi cualquier electrodo, que serán los encargados de captar la señal bioleléctrica directamente del torso del conejo. Para el presente proyecto se utilizarán dos electrodos de placas metálicas, compuestos por un botón de cloruro de plata y un disco de hule espuma que lo fija al cuerpo.

1.3.9 BATERIA

La batería será una pieza clave en la viabilidad del dispositivo. No puede superar el tamaño de la placa 3,5x5cm y su autonomía ha de ser mayor de 12h (toda la noche funcionando). Como se ha descrito previamente, el circuito de carga se ha adaptado a la batería. Se escoge finalmente una batería de polímero de litio (LiPo) utilizada en ya en otros dispositivos como el iPod mini. [13] Sus propiedades son las siguientes:

- Voltaje - 3.7V
- Corriente - 500 mAh
- Potencia - 1.85Wh

1.4 ESQUEMÁTICO FINAL

Por último, se expone el esquemático de todo el diseño realizada (Figura 33 y III. Planos). En la parte superior, se observa el circuito de carga, el zócalo de la memoria y las luces LED. En la parte inferior, se ubican los reguladores de tensión (con el interruptor), el ADS1292R y el PIC24FJ128GC006.

Todos los planos se realizan con el programa EAGLE 9.5.1 de diseño electrónico. Los elementos se introducen desde la librería de SCR (sketches) o manualmente descargándolos desde la red. Una vez colocados se unen con el comando "net" (red) y se etiquetan mediante el comando "label" (etiqueta). Se pueden cambiar las características de cada elemento con la herramienta "name" (nombre) y "value" (valor). Finalmente, con la opción "move" (mover) se ordena todo el esquemático.

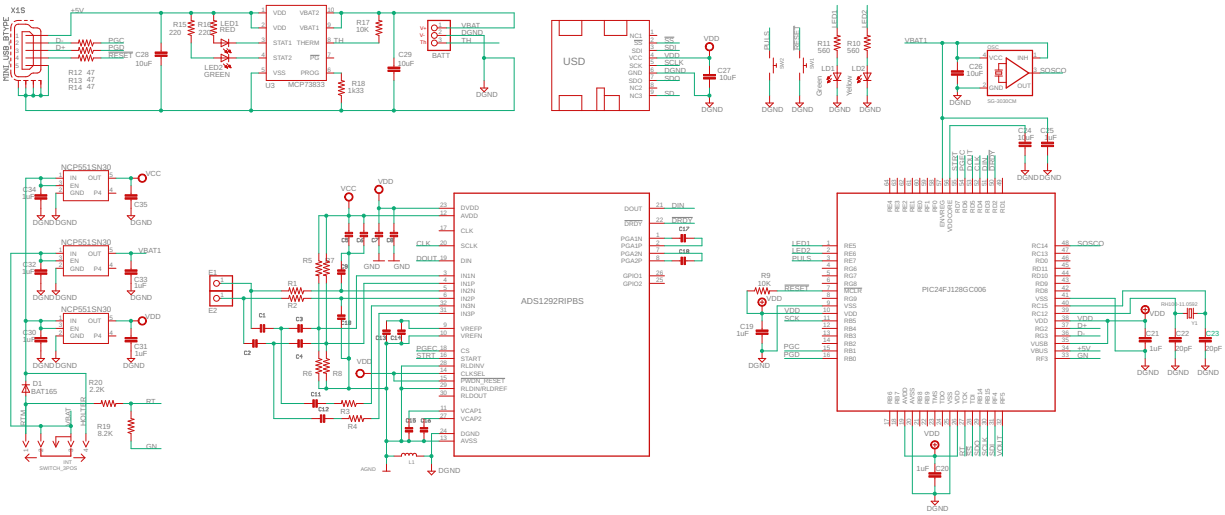

Figura 33 Esquemático final

1.5 DISEÑO DE LA PLACA (PCB)

Terminado el esquemático se procede al diseño de la placa PCB, también con el mismo programa EAGLE 9.5.1 utilizado anteriormente. Utilizamos la opción "switch to board" (cambiar a placa) para pasar al diseño del boardfile.

1.5.1 CONCEPTOS BÁSICOS

Empezamos definiendo una serie de conceptos clave para el diseño de PCB.

- **Capa *Top*** – Capa superior de la placa, sus trazas se muestran en color rojo.
- **Capa *Bottom*** – Capa inferior de la placa, sus trazas se muestran en color azul.
- ***Pads*** – Zona ocupada por un pin de un elemento, suelen ser rectangulares.

- **Huella** – Zona ocupada por el elemento dibujado en la placa.
- **Máscara de soldadura** – Capa protectora de las trazas de cobre, habitualmente de color verde.
- **Capa de serigrafía** – Textos impresos en la placa que ayudan a la colocación y aportan información al montaje.
- ***Net*** – Línea de comunicación que conecta componentes en la placa PCB.
- **Vía** – Agujero que conecta *pads* de una capa a otra.

1.5.2 TAMAÑO INICIAL

Lo primero que haremos será definir el tamaño inicial a 35x50mm, por lo que ajustaremos la placa a esas dimensiones antes de empezar a colocar los componentes.

1.5.3 COLOCACIÓN DE LOS COMPONENTES

Para el diseño de la placa se deben tener en cuenta varias directrices que tendremos que cumplir para el correcto funcionamiento del Holter. La principal de ellas es la de evitar que la parte digital induzca ruido en la analógica, ya que esta señal es muy sensible a estos cambios, y es por ello por lo que el ADS1292R estará ubicado en una esquina de la PCB.

Al convertir las señales analógicas en código digital, el conversor trabaja con unos niveles determinados que vienen definidos por unos valores determinados: VIHmax, VIHmin, VILmax, VILmin, VOHmax, VOHmin, VOLmax y VOLmin (Figura 34). Normalmente el nivel bajo (0) se asocia con 0V y el alto (1) con 3,3V. Al ser del orden de voltios, las pequeñas interferencias y ruidos apenas tendrán consecuencias.

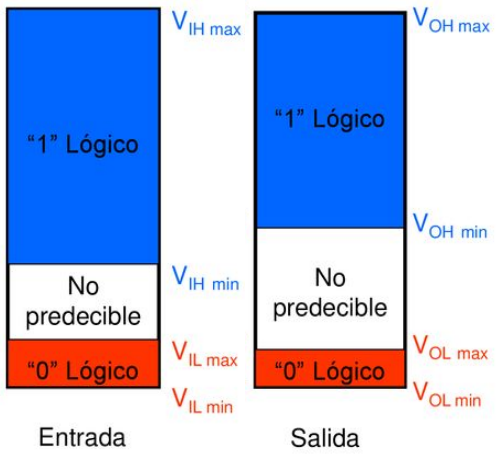

Figura 34 Diagrama de niveles de conversión

En la Figura 35 se puede apreciar la separación final de ambas zonas; en amarillo la parte analógica y en azul la digital.

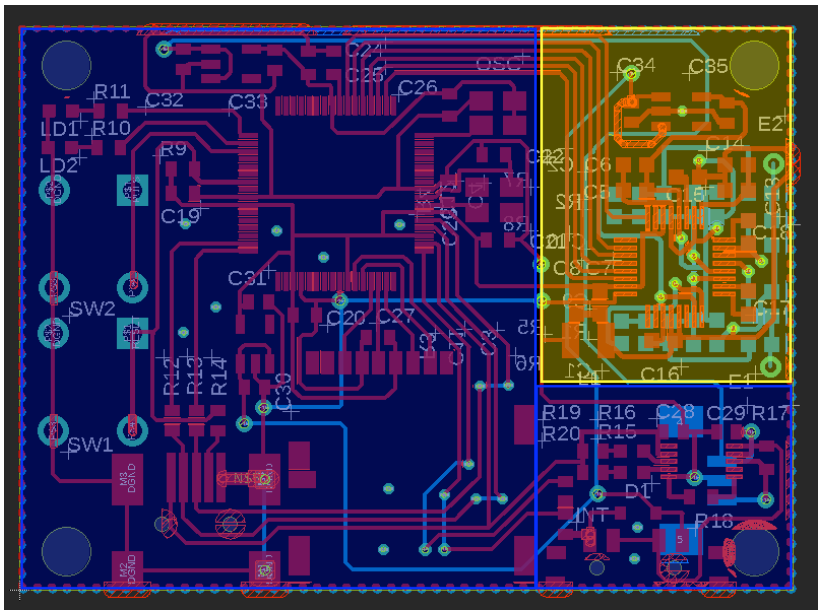

Figura 35 Distribución PCB parte analógica y digital

El microcontrolador estará situado en la parte central para facilitar su conexión a todos los elementos del dispositivo. Tanto la entrada Mini USB como la tarjeta SD estarán ubicadas cerca del borde de la PCB para su correcto funcionamiento. Todos los interruptores se colocan en puntos de fácil manejo cercanos también a los extremos. En cuanto a la masa, se decide diseñar una para cada parte (analógica y digital), cuya unión se realiza lo más cercana posible al pin de tierra de la batería, punto más alejado de los electrodos de medida.

Por último, la batería se situará pegada a la capa *bottom* y no entorpecerá en ningún momento el funcionamiento del Holter.

1.5.4 PISTAS Y VÍAS

A la hora de trazar las pistas de la PCB debemos atender las siguientes directrices:

- Evitar ángulos agudos en los cambios de dirección.
- Separación uniforme entre las pistas.
- Conexión radial entre pista y *pad* (nunca tangencial).
- Se pueden conectar como máximo cuatro pistas a cada *pad*.
- Evitar la conexión directa entre dos *pad* cercanos, se realiza con una pista, aunque sea de pequeño tamaño.

- El mínimo ancho de pista es 0,15mm, pero el mínimo recomendado por el fabricante es de 0,2032mm. Nosotros nos decantaremos por una anchura general de 0,254mm, suficiente gracias al bajo amperaje de las corrientes del dispositivo.
- Las vías son agujeros que conectan dos caras distintas de la PCB y en nuestro dispositivo, al ser de pequeño tamaño, abundan mucho. Se componen de dos *pad* y un tubo conductivo que los une por una perforación en la placa. En la Figura 36 se expone un esquema de esta configuración.

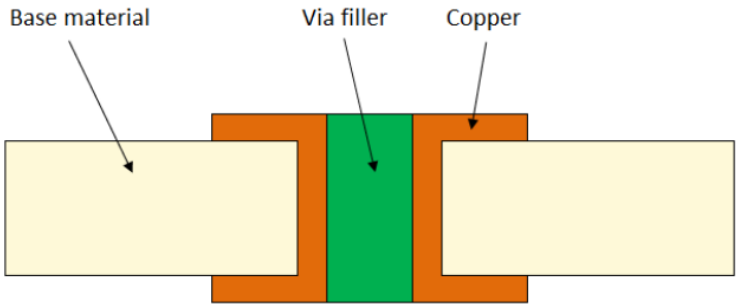

Figura 36 Esquema de una vía (Fuente: Eurocircuits)

1.5.5 ESPESOR

Los espesores disponibles en el programa van de 0,4mm a 2mm. Si la placa es muy delgada su coste será elevado, por el contrario, si es demasiado gruesa, su peso y tamaño aumentará perjudicando también nuestro objetivo de diseño. Por tanto, finalmente se decide elegir un espesor de 1mm, valor intermedio que soportará todos los elementos de la PCB.

1.5.6 PCB FINAL

La distribución final se expone en la Figura 37.

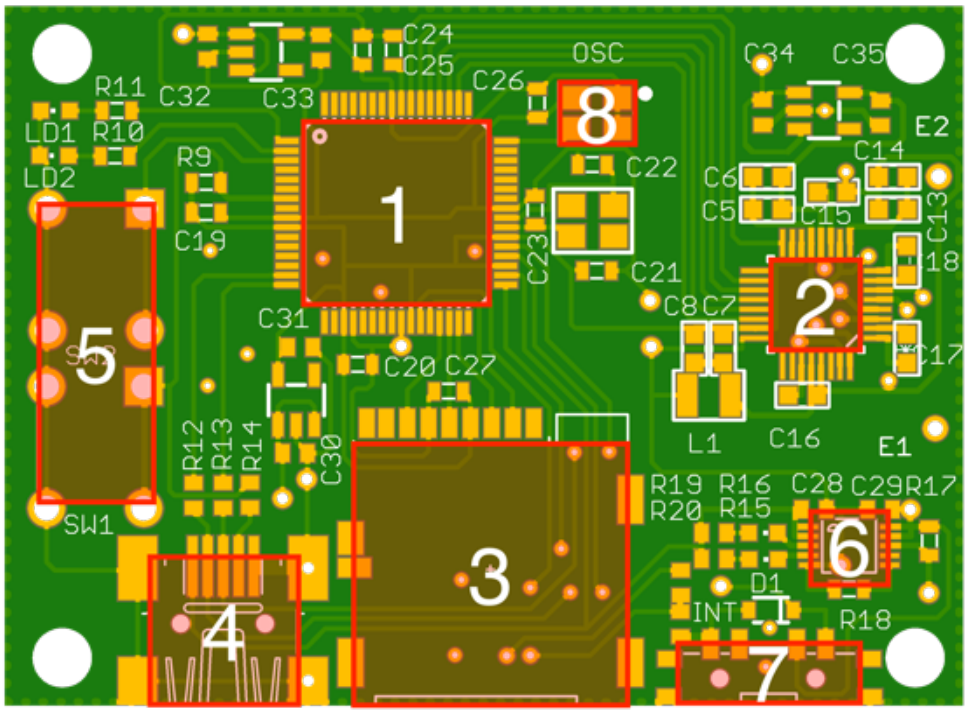

Figura 37 Distribución final de la PCB. 1-PIC24FJ128GC006. 2- ADS1292R. 3-Tarjeta SD. 4-Mini USB. 5-Pulsadores. 6-Circuito cargador. 7-Interruptor. 8-Oscilador

En la Figura 28, se muestran las capas *top* (arriba) y *bottom* (abajo) de la placa, mientras que la Figura 39 muestra una vista desde el entorno del programa EAGLE, donde en rojo aparecen los elementos de la capa *top* y en azul los de la *bottom*.

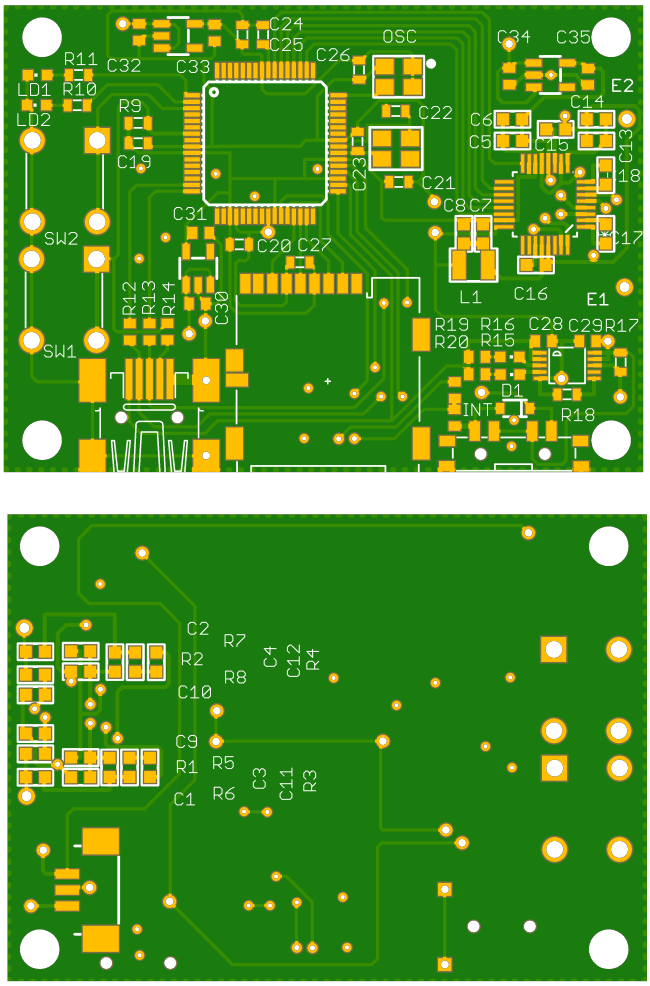

Figura 38 Capas TOP y BOT de la PCB

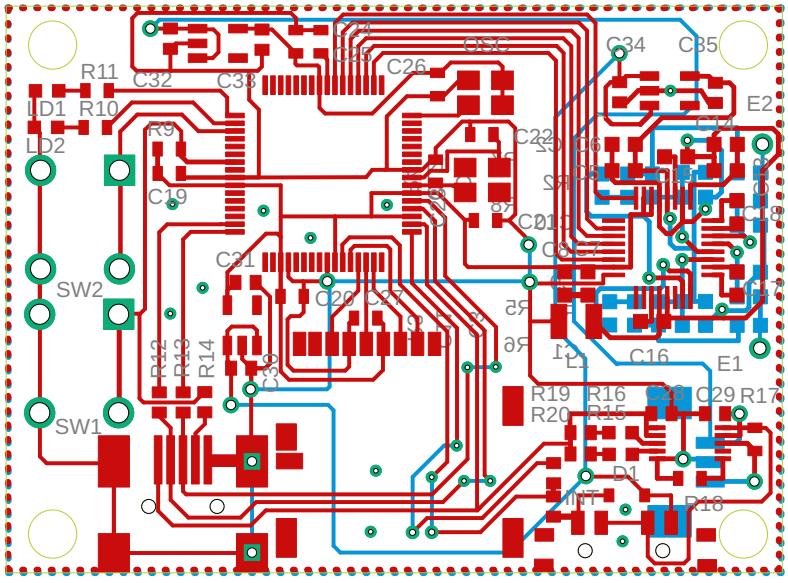

Figura 39 Vista desde EAGLE de la PCB

1.6 **FABRICACIÓN**

Para la etapa de fabricación, se encarga a una empresa la fabricación de la placa y se adquieren todos los elementos elegidos para el diseño. En la Figura 40, se presentan 2 fotografías de la PCB recibida.

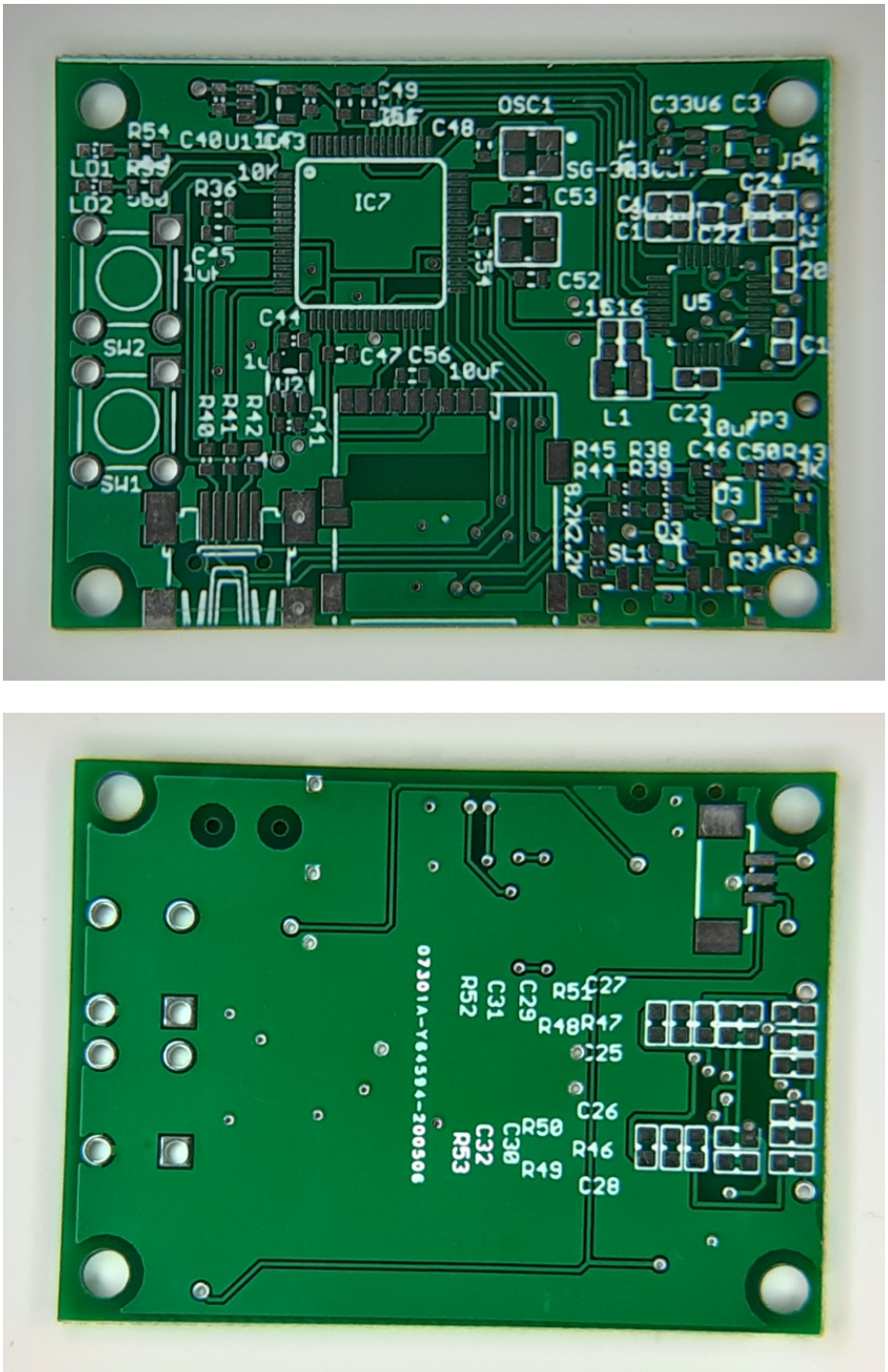

Figura 40 PCB proporcionada por el fabricante (cara superior y cara inferior, sin montar)

1.7 **CONSUMO Y AUTONOMÍA**

La autonomía del dispositivo es un factor clave, pues ha de ser suficiente para aguantar encendido las aproximadamente diez horas demandadas. Para comprobar dicha capacidad se realiza la siguiente tabla con los consumos medios de los componentes principales de la PCB, obtenidos de sus respectivas hojas de datos (Tabla 1).

COMPONENTE	CONSUMO (mA)
PIC24FJ128GC006 (microcontrolador)	26,000
ASD1292R (circuito integrado)	10,000
USD (tarjeta memoria)	0,600
MCP73833 (circuito de carga)	0,075
NCP551SN30 (reguladores lineales)	0,006
SG-3030CM (oscilador externo)	1,500
TOTAL	38,181 mA

Tabla 1 Consumo de los componentes principales del dispositivo

Sabemos que la batería elegida tiene una capacidad de 500 mAh y ante un funcionamiento de exigencias por encima de la media, pero por debajo de valores máximos, se obtiene que la autonomía del Holter (Ecuación 1) es de 13,1 horas. Por tanto, consideramos válida esta parte del diseño.

$$\frac{500\ mAh}{38{,}181\ mA} = 13{,}1\ h$$

Ecuación 1 Autonomía del Holter

1.8 **MONTAJE**

Una vez recibida la placa se lleva a cabo el montaje, el cual se realizará mediante soldadura y siguiendo la directiva RoHS ("Restriction of Hazardous Substances"). Los pasos que seguir son los siguientes:

- Comprobar la limpieza de los *pads* y de las pistas.
- Colocar los elementos de menor tamaño primero (para evitar desnivelaciones).
- Realizar soldadura aplicando estaño y calor en las uniones.
- Enfriar quedando la conexión realizada.

En la Figura 41 se muestra ya la placa terminada, con todos los elementos soldados.

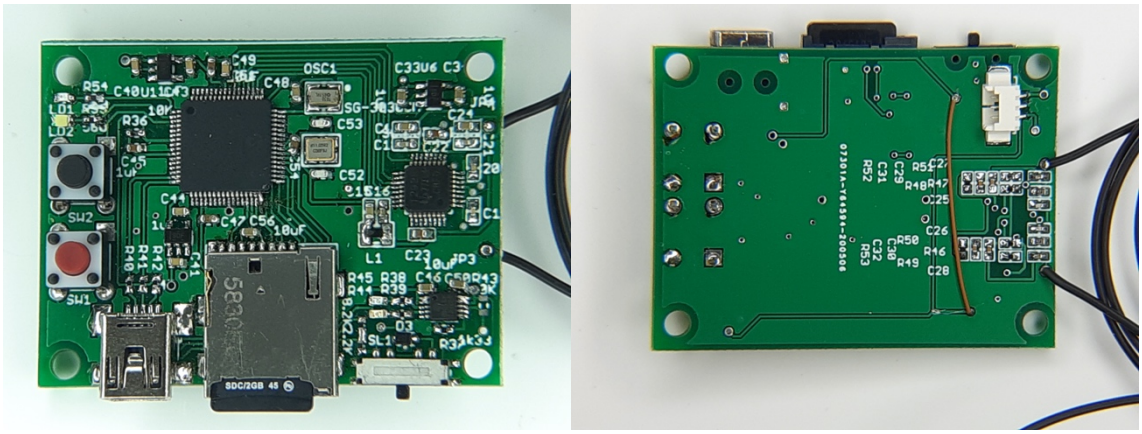

Figura 41 PCB montada (cara superior y cara inferior)

En la Figura 42 se muestra la PCB, con los electrodos, y la batería, que más tarde se acoplará por debajo para que no entorpezca el funcionamiento.

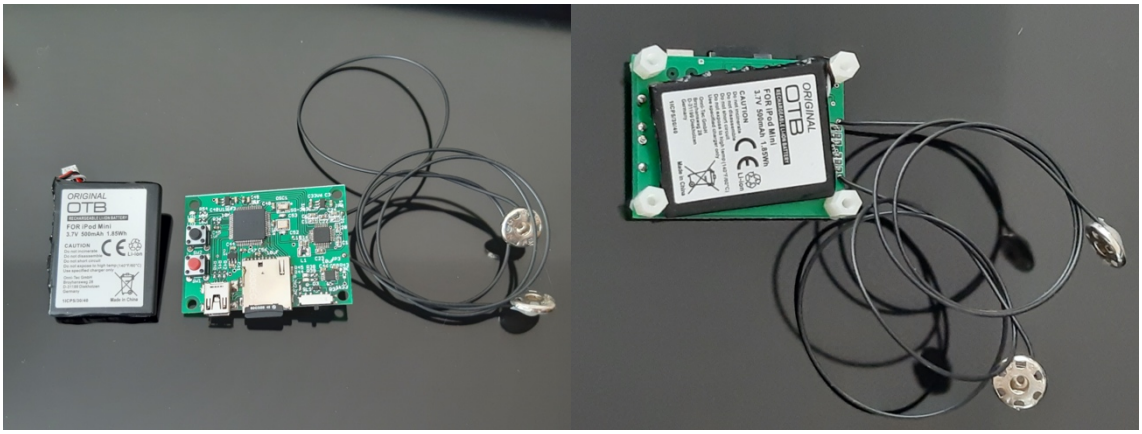

Figura 42 Conjunto completo de PCB, batería y cables de conexión (izquierda) y posición de la batería (derecha)

Por último, en la Figura 43 se muestra dispositivo montado con tornillos de nylon mediante los cuales se conecta a otra placa a modo de "sándwich" para fijar la batería. Este es el resultado final y al que solo le faltaría una posible carcasa de aluminio o plástico.

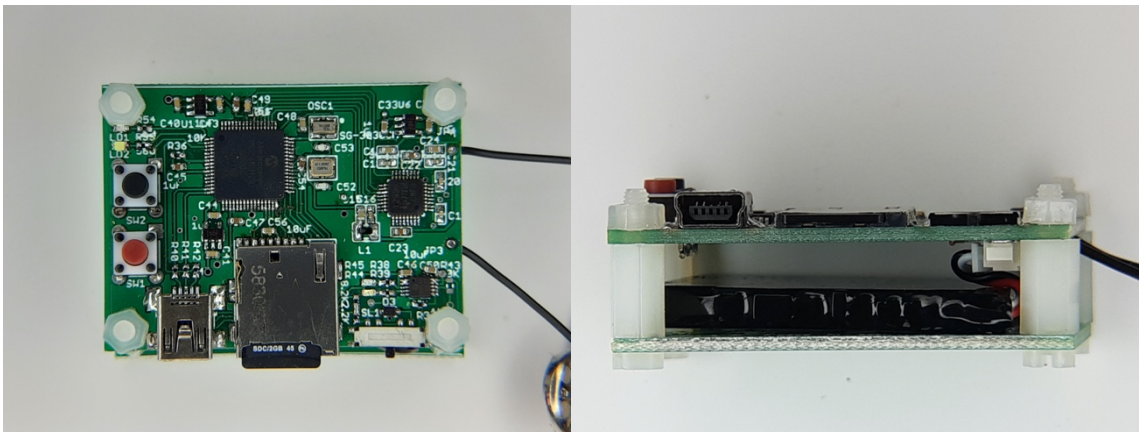

Figura 43 Montaje final

2. ADQUISICIÓN DE DATOS Y COMUNICACIÓN

2.1 TRAMA DE DATOS

El procesador recibe 72 bits que se distribuyen de la siguiente forma (Tabla 2).

ESTADO	ECG	RESPIRACIÓN
24 BITS	24 BITS	24 BITS

Tabla 2 Distribución del código de 72 bits

A continuación, se almacenan en la memoria los datos de las señales adquiridas, más un código de 8 bits de ID para cada señal (Tabla 3).

ECG		RESPIRACIÓN	
SEÑAL	ID (identidad)	SEÑAL	ID (identidad)
24 BITS	8 BITS	24 BITS	8 BITS

Tabla 3 Distribución del código de 64 bits

Los datos tienen formato DWORD (signo Ca2) y el último byte de cada señal de 32 bits se fijará en 1 o 2 dependiendo de si es señal ECG o de respiración. Cada fichero almacena valores recogidos durante una hora, creándose un fichero nuevo cada vez, cuyos nombres seguirán la siguiente nomenclatura: **D0_00_00.DAT**, **D0_01_00.DAT**, **D0_02_00.DAT**…

También hay que destacar que el nivel de batería no se controlará de ninguna forma por el programa, tendremos que hacerlo nosotros manualmente mediante los leds del circuito de carga que serán los encargados de suministrarnos dicha información. Si el LED rojo se ilumina deberemos cargar el dispositivo hasta que el LED verde se encienda.

En cuanto a la detección de electrodo suelto, si bien es cierto que algunos bits de la palabra ESTADO se dedican a ello, no se está teniendo en cuenta. Una posible mejora para un futuro sería la de utilizar parte del byte ID para optimizar esta función.

2.2 **PROGRAMACIÓN**

La programación del microcontrolador PIC24FJ128GC006 se realiza en lenguaje C mediante un compilador llamado CCS (Custom Computer Services, Inc.).

- **Línea 1 – 4** Encabezado del programa.

En la primera línea y mediante el comando #ifndef, se define la función main_h. Inmediatamente después se declara la librería del microcontrolador utilizado, esta contiene todas las etiquetas necesarias para nuestro modelo. Cada microcontrolador tiene una librería correspondiente.

También se declara el número de bits del convertidor analógico digital incorporado en el microcontrolador, en nuestro caso es de 12 bits.

- **Líneas 6 – 79** Definición de fusibles.

Se indica la configuración del microcontrolador y por tanto se define su funcionamiento. Los fusibles disponibles se indican en la librería del modelo utilizado. En caso de querer desactivar alguna función basta con poner NO delante de su nombre. A continuación, se exponen esas líneas del código.

  - #fuses **NOJTAG** // JTAG port is disabled
  - #fuses **NOPROTECT** // Code protection is disabled
  - #fuses **WRT** // Writes to program memory are allowed
  - #fuses **LVR** // Low-power, low-voltage/retention regulator is enabled and controlled in firmware - RETEN bit
  - #fuses **ICSP1** // Emulator functions are shared with PGEC1/PGED1
  - #fuses **NOWDT** // WDT is disabled; SWDTEN bit is disabled
  - #fuses **WINDIS** // Standard Watchdog Timer is enabled
  - #fuses **WDT128** // WDT Prescaler Ratio Select bit: Prescaler ratio of 1:128
  - #fuses **WPOSTS16** // Watchdog Timer Postscaler Select bits-> 1:32,768 (NOT USED)
  - //#fuses **NOIESO** // Internal External Switchover bit: IESO mode (Two-Speed Start-up) is disabled
  - #fuses **WDTCMX** // WDT Clock Multiplex Control bit: Enables WDT clock multiplexing
  - #fuses **CVREFNORM** // External CVREF+/CVREF- Location Select bit (NOT USED)
  - #fuses **VREFNORM** // External AVREF+/AVREF- Location Select bit (NOT USED)
  - #fuses **FRC_PS** // Initial Oscillator Select bits: Fast RC Oscillator with Postscaler (FRCDIV)
  - #fuses **CKSNOFSM** // Clock switching is enabled, Fail-Safe Clock Monitor is disabled
  - #fuses **OSCIO** // OSCO Pin Configuration bit: OSCO/CLKO/RC15 functions as port I/O (RC15)
  - #fuses **WDTCLK_SOSC** // WDT Clock Source Select bits: SOSC input
  - #fuses **WPEND** // Segment Write Protection End Page
  - #fuses **NOWPCFG** // Configuration Word Code Page Write Protection

- #fuses **WPDIS** // Segment Write Protection Disabled
- #fuses **NOBROWNOUT** // Brown-out Reset Disabled
- #fuses **WDTWIN_50%** // Watchdog Timer Window Width Select bits
- #fuses **SOSC_SEL** // SOSC Selection bit: SOSC circuit is selected
- #fuses **NOWPFP** // Write-Protected Code Segment Boundary Page bits
- #fuses **PLL1** // USB 96 MHz PLL Prescaler Select bits: Oscillator input used directly (4 MHz input)
- #fuses **RTCBAT** // RTCC operation continues when the device is in VBAT mode
- #fuses **DS_SW** // Deep Sleep operation is enabled and controlled by the DSEN bit
- #fuses **NODSWDT** // Deep Sleep Watchdog Timer Enable bit: Deep Sleep WDT is disabled
- #fuses **NODSBOR** // Deep Sleep Brown-out Reset
- #fuses **DSWDTCK_SOSC** // Deep Sleep Watchdog Timer Clock Select bit: Clock source is SOSC
- #fuses **DSWDT_25DAYS** // Deep Sleep Watchdog Timer Postscaler
- #fuses **DEBUG**

- **Línea 82 – 152** Definición de etiquetas.

Se definen todas las etiquetas del microcontrolador, es decir, se definen los nombres de los 64 pines del dispositivo. También se definen los estados ON (output_high) y OFF (output_low) del dispositivo.

- **Línea 153** Dirección de memoria.

Se define SPIBUF que constituye un registro del microcontrolador. La posición de la memoria RAM 0x0248 tiene ahora la etiqueta de SPI1BUF, esto permite tanto su escritura como su lectura.

- **Línea 155 – 157**

Se activa el oscilador primario y se declara su pulso, el cual es de 8000000Hz (8MHz)

- **Línea 159**

Se cierra el condicional #ifndef que abarca todo el main_h.

- **Línea 160 – 161**

Se establece otro #ifndef para definir si no está definida ya, la función INC_ADS1292R, que es la encargada de iniciar el circuito integrado ADS1292R.

- **Línea 163 – 177** Definición de los códigos de operación de los comandos del ADS.

La configuración del circuito integrado se realiza mediante comandos, que a su vez son manejados por códigos de operación de 8 bits que deben ser declarados. A continuación, se enuncian los utilizados en nuestro dispositivo.

- **WAKEUP** – Salir del modo de espera / sleep.
- **STANDBY** – Entrar en el modo de espera / sleep.
- **RESET** – Devolver la configuración del registro a sus valores por defecto.
- **START** – Comenzar la conversión de datos.
- **STOP** – Terminar la conversión de datos.
- **OFFSETCAL** – Ajustar a cero la tensión en cortocircuito de los electrodos.
- **RDATACT** – Habilitar el modo continuo de lectura de datos.
- **SDATACT** – Cancelar el modo continuo de lectura de datos.
- **RDATA** – Con el modo continuo deshabilitado, se emite con cada flanco de bajada de DRDY para poder leer la conversión realizada.
- **RREG0** – Leer registros de datos.
- **WREG0** – Escribir registros de datos.

- **Línea 179 – 655** Recordatorio de los registros de configuración del ADS1292R.

Descripción de los registros del ADS1292R para facilitar su entendimiento y programación.

- **Línea 657 – 738** Función de configuración del ADS1292R.

En estas líneas se establece la forma de comunicación entre el PIC24FJ128GC006 y el ADS1292R. Se gestiona por tanto el puerto serie o bus SPI. Lo primero a realizar es la inicialización de la función mediante el comando setup_SPI( ).

- **SPI_MASTER** – Configurar el PIC24FJ128GC006 como maestro.
- **SPI_SLAVE** – Configurar el ADS1292R como esclavo.
- **SPI_SS_DISABLED** – Desactivar el pin de selección al existir solo un esclavo.
- **SPI_L_TO_H** – Enviar datos cada flanco de subida.
- **SPI_XMIT_H_TO_L**
- **SPI_CLK_DIV_5** – Dividir frecuencia del oscilador.

También cabe destacar que los comandos se llevaban a cabo en dos ciclos de reloj, ergo su pulsación será la mitad de la del PIC24FJ128GC006 (8MHz / 2 = 4MHz).

Se procede a introducir los registros, pero para ello antes debemos parar el modo de lectura continuo (que se pone en marcha al iniciar el dispositivo) con el comando SDATACT, mediante WREG0. Este está compuesto por 16bits que incluyen la dirección de la memoria y el número de registros menos uno a escribir.

En el siguiente paso, se procede a la lectura de los datos escritos para asegurar la ausencia de errores. Para ello se utiliza el comando RREG0 mencionado anteriormente, que permite la lectura de los

registros. Mediante la comparación de estos valores con los del registro SPI1BUF sabremos si lo escrito es correcto. En ambos procesos se introducen pequeños retardos de tiempo para facilitar y asegurar que se completen adecuadamente.

Una vez comprobado, se activa la compensación con OFFSETCAL y se vuelve a instaurar el modo de lectura continuo con RDATACT, quedando ya a disposición del comando START que inicia la conversión de datos.

- Línea **739 – 767** Definición de interrupción y de su rutina de servicio.

Se define la rutina de la interrupción como int_EXT1 y se declaran las variables recibido, rsp_[16] y ecg_[16]. El microprocesador ejecutará una interrupción cada vez que haya datos listos a la salida del convertidor, que será de forma cíclica cada 125sps (4ms). Lo primero que debemos hacer es establecer la conexión con el esclavo seleccionado y en nuestro caso solo existe uno, el ADS1292R. Esto se hace poniendo a cero la señal CS_ADS, activando así el *chip select* y la comunicación SPI. A partir de ahí, lo único que interesa es leer los datos del esclavo, por lo que mediante el comando spi_read iremos recogiendo la información demandada.

La señal de reloj tiene una forma cuadrada y se caracteriza por tener ocho ciclos, en cada flanco de bajada escribe un cero en la señal MOSI, y en cada flanco de subida el microprocesador captura un bit de la señal mencionada anteriormente y lo incorpora al buffer SPI1BUF. Una vez obtenemos un byte, este se transfiere a una de las variables declaradas.

Esto se realiza durante 9 ciclos de reloj para cubrir los 72 bits enviados de cada vez a la memoria. Finalmente se incluye un pequeño retardo de 10µs, para asegurar que el proceso se termina adecuadamente, se cambia a uno la señal CS_ADS y se aumenta el contador de la variable recibida.

- **Línea 769 – 976** Función principal.

Controla y organiza el funcionamiento. Se encarga de actualizar la fecha y la hora, mediante el oscilador externo. También establece los patrones de los LED1 y LED2, si está en modo normal se encenderá el primero y si está en modo test el segundo. Si no se están introduciendo datos, no se enciende ninguno.

Se reciben muestras del ADS1292R y se van llenando los sectores de la memoria (1 Sector = 512 bits = 64 muestras de 1 byte). Cada hora se termina un fichero y se empieza con otro nuevo.

- **Línea 978 – 1299** Memoria.

Se empieza definiendo la función INC_SD_CARD con el comando #ifndef si aun lo estaba. Después se definen los comandos y los códigos de respuesta de la memoria.

  - #define **CMD0 0x40** // go to idle
  - #define **CMD1 0x41** // initialization process
  - #define **CMD8 0x48** // verify interface
  - #define **CMD17 0x51** // read single block
  - #define **CMD24 0x58** // write single block
  - #define **CMD55 0x77** // escape for app specific command

- #define **CMD58 0x7a** // read OCR
- #define **ACMD41 0x69** // poll operation range
- #define **R1_READY_STATE 0x00** // SD ready
- #define **R1_IDLE_STATE 0x01** // card in idle state
- #define **R1_ILLEGAL_COMMAND 0x04** // illegal command
- #define **DATA_START_BLOCK 0xFE** // start token for read or write
- #define **DATA_RES_MASK 0x1F** // mask for data response
- #define **DATA_RES_ACCEPTED 0x05** // write accepted

Se configura también fecha y hora, para que queden registrados con los datos y se facilite luego la interpretación de estos. Cuando la señal CS este en cero, la tarjeta estará habilitada para tanto recibir como para transmitir datos.

2.3 VISUALIZACIÓN DE RESULTADOS

Para una prueba inicial de su funcionamiento, primero se procede a la comprobación del circuito de carga mediante el puerto Mini USB. En la Figura 44 tenemos el LED rojo iluminado, indicando que la batería está agotada. En la Figura 45 vemos ya el LED verde iluminado, informando que se ha cargado por completo.

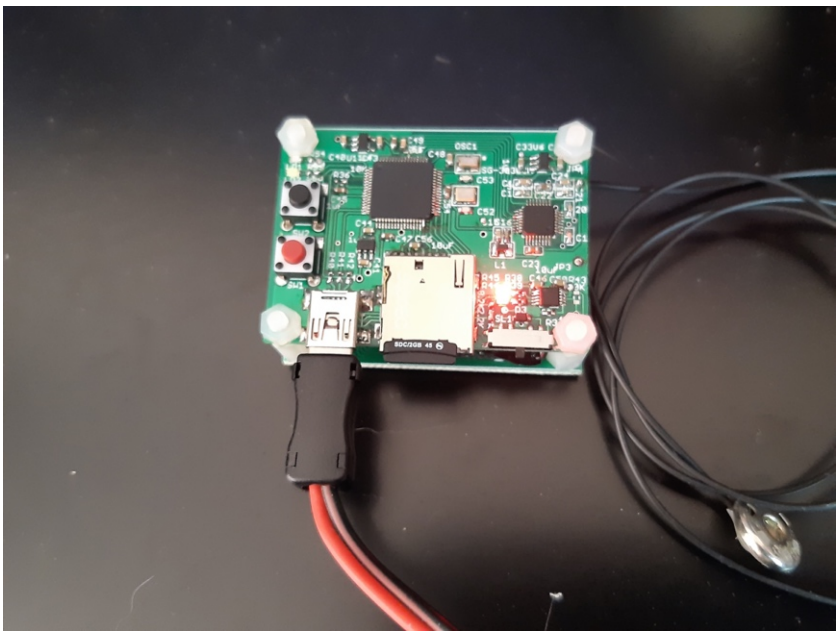

Figura 44 Batería baja

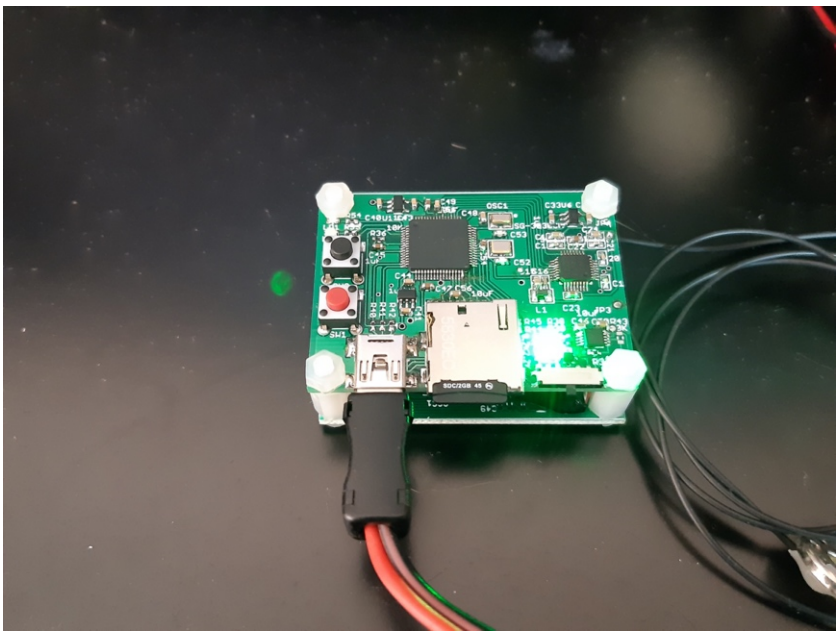

Figura 45 Batería cargada

Con la alimentación asegurada, se realiza un montaje inicial para la implementación del firmware (Figura 46), en el que se distinguen los siguientes elementos de izquierda a derecha:

- El programador PICKit 3 que conecta el microprocesador con la computadora.
- Analizador Lógico conectado a la resistencia que alimenta al LED Blanco para medir duraciones.
- Convertidor para comunicar vía serie con PC (conectado a un pin libre y GND).
- Simulador de ECG para comprobar funcionamiento del AFE (*Active Front End*).

Una vez concluida la implementación del programa se desconectan los cables y se elimina la parte del código empleado en la comprobación del correcto funcionamiento.

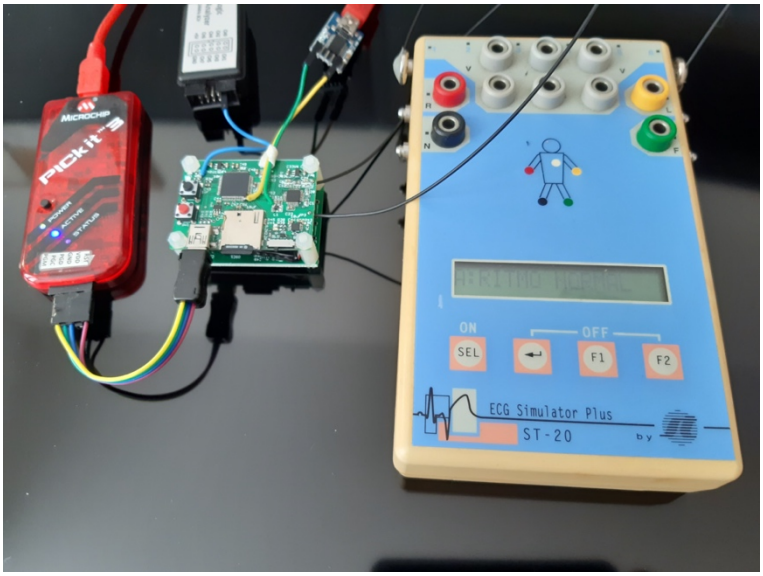

Figura 46 Montaje inicial

A la hora de implementar el firmware se produce una comunicación serie con un PC. Esto permitió emplear el programa SIOW (del entorno de desarrollo CCS PICC) para poder visualizar las señales muestreadas en tiempo real (Figura 47) lo que permitió comprobar que funcionaba correctamente (al menos el ECG).

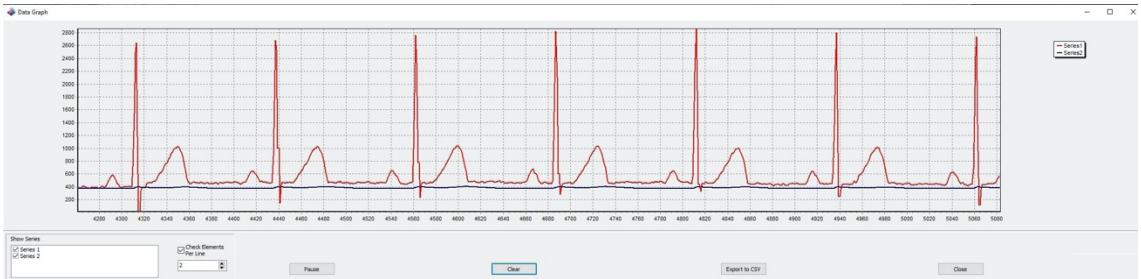

Figura 47 Señal ECG muestreada en tiempo real

A continuación, en la Figura 48, se muestra el entorno de desarrollo de la implementación del programa (MPLAB IDE v.5.92 + Compilador CCS).

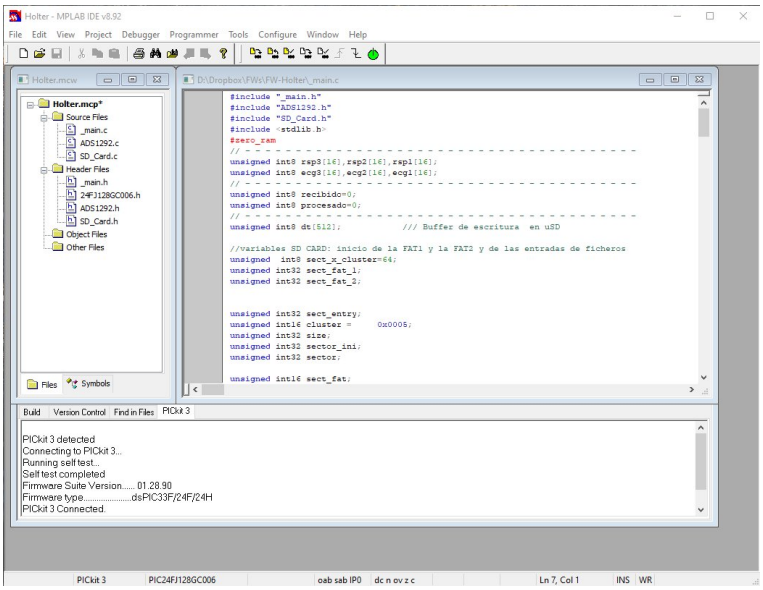

Figura 48 Captura del programa MPLAB y CCS

La escritura de los datos en la tarjeta de memoria, tal y como se comentó anteriormente, es en formato binario de palabras DWORD (32 bit con signo Ca2) alternando ECG y Respiración. A esta medida se le dedican 24 bits, por lo que el byte restante se destina para identificar la señal en cuestión (1 o 2 dependiendo si es Respiración o ECG). Cada fichero almacena los valores recogidos durante 1 hora, creándose un fichero nuevo cada hora transcurrida de adquisición. En la Figura 49 se recoge una captura de algunas líneas de estos datos.

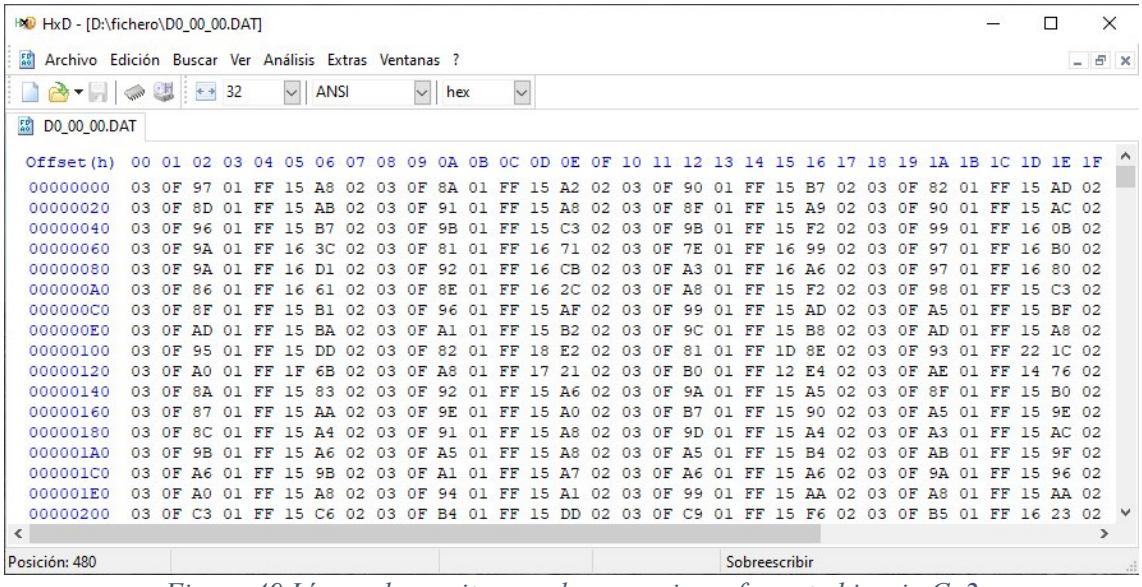

Figura 49 Líneas de escritura en la memoria en formato binario Ca2

Como los ficheros guardados son de tipo binario, es necesario convertir en texto para poder filtrar, procesar (p.ej. detectar el ritmo cardiaco y el ritmo de respiración) y representar las señales. Por tanto, conviene escribir otro código de apoyo para el PC en entorno GCC (Visual Studio Code) para realizar este trabajo. En la Figura 50 tenemos una captura de dicho programa.

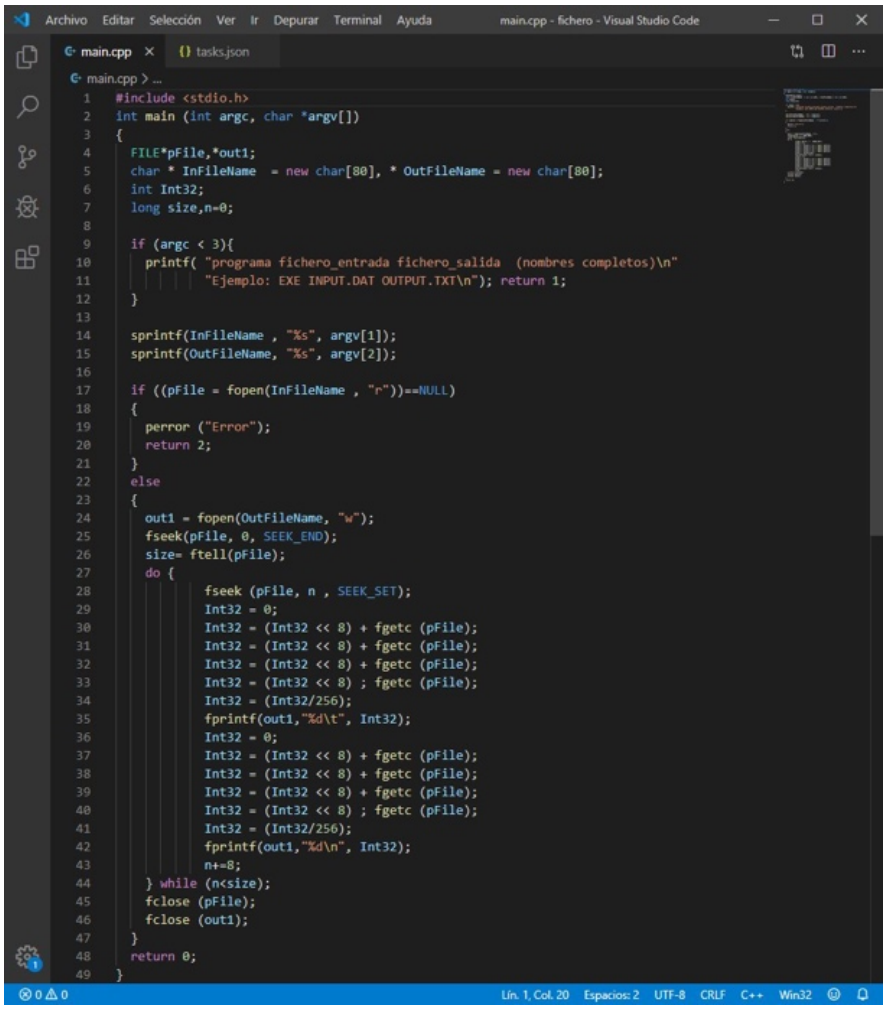

Figura 50 Programa para la conversión de datos

De esta forma se comprueba que el dispositivo está operativo y por tanto se podría comenzar la fase de pruebas en los animales, que permitiría concluir si efectivamente el diseño es correcto.

3. PRUEBAS

Ante la imposibilidad de realizar las pruebas con los conejos debido a las restricciones actuales en los hospitales e institutos de investigación por motivo del COVID-19 (aprobación de los procedimientos experimentales y acceso a las instalaciones del animalario de la Universitat de València), nos vimos obligados a posponer las mismas, y por ese motivo, no se presentan adquisiciones de señales en el presente trabajo de fin de máster.

4. CONCLUSIONES Y LÍNEAS FUTURAS

El objetivo principal de este proyecto era el diseño y puesta en funcionamiento un Holter de pequeño tamaño y bajo consumo, para la monitorización del electrocardiograma y de la respiración de un animal de pequeño tamaño.

Para ello, primero se han seleccionado los componentes principales y se ha procedido a la lectura de sus respectivas hojas de datos. Con toda la información necesaria se realiza el esquemático del circuito y posteriormente el diseño de la PCB, siguiendo las normas establecidas para su correcta distribución. Se ha realizado su montaje y a continuación se ha programado el microcontrolador, para la realización de las pruebas de adquisición.. Se ha comprobado su funcionamiento mediante simulación y se han procesado los datos monitorizados. Por último, se han convertido estos datos en valores gráficos que permiten su interpretación.

Por tanto, se ha obtenido un dispositivo que almacena en una memoria SD el ritmo cardíaco y la respiración, con una autonomía que dura toda la noche, permitiendo al usuario la monitorización del animal en todo momento. Estos resultados son muy interesantes para la investigación de ciertas patologías y el estudio de patrones para la detección y prevención de arritmias.

Su uso podría extenderse a otros campos, como el del deporte de alto rendimiento o como recurso habitual para la monitorización del ritmo cardíaco en mascotas, gracias a su pequeño tamaño y a su bajo coste. En la actualidad, la monitorización de la actividad cardíaca en tiempo real de un paciente es algo fundamental y con un rango de explotación elevado, debido al amplio número de personas afectadas por enfermedades cardiovasculares y trastornos del ritmo cardíaco. Este proyecto ha propuesto una solución sólida a este problema, aunque cabe mencionar la necesidad de validación del dispositivo diseñado.

Como líneas futuras, existen posibilidades de mejora entre las que destacan las siguientes:

- Carcasa de Aluminio – Su implementación aportaría resistencia mecánica y térmica al *Holter*. Además, actuaría como un tercer electrodo que proporcionaría mayor estabilidad a la polarización del amplificador diferencial, definiendo la tierra con precisión, y que se traduciría en menos ruido y menos interferencias (mejor calidad de señal).

- Conexión inalámbrica – La conexión inalámbrica sería una de las mejoras que facilitaría la accesibilidad y la monitorización. El problema es que, con nuestro tamaño de placa y autonomía de batería, su implementación eficiente es muy complicada. En un futuro quizá estas limitaciones no existan.

- Software personalizado – La programación de un *firmware* a medida de nuestro dispositivo es otro punto pendiente. De esta forma se aprovecharían al máximo todas las funciones disponibles y por tanto una optimización del *Holter* en su conjunto.

- Post-procesado - Para facilitar su interpretación, es conveniente realizar un post-procesado de las señales obtenidas, eliminando datos irrelevantes y destacando los puntos de interés. En la

señal de respiración no será necesario debido a su simplicidad, ya que cualquier anomalía se expondrá sin problemas al no tener la señal particularidades especiales, caracterizada por una subida y una bajada que corresponden con la inspiración y la espiración respectivamente. Sin embargo, para el análisis de la señal de ECG, se propone la implementación de un algoritmo de detección del intervalo RR, para la evaluación de las alteraciones en el ritmo cardiaco.

5. REFERENCIAS

[1] https://www.who.int/es – OMS (2017)

[2] "Estudio de las modificaciones farmacológicas de los efectos electrofisiológicos producidos por el estiramiento local miocárdico a partir de técnicas dinámicas de cartografía eléctrica en un modelo experimental de corazón aislado de conejo" – Irene del Canto Serrano (2014)

[3] https://www.texasheart.org – Texas Heart Institute (2020)

[4] https://fundaciondelcorazon.com – Fundación Española del Corazón (2020)

[5] https://www.my-ekg.com – My EKG (2020)

[6] "Diseño e implementación de un sistema portátil de adquisición de potenciales evocados auditivos del tronco cerebral mediante estimulación acústica" – Joaquín Tomás Valderrama Valenzuela (2010)

[7] "Low-Power, 2-Channel, 24-Bit Analog Front-End for Biopotential Measurements" – Texas Instruments (2011)

[8] "Diseño e implementación de un monitor de respiración y electrocardiografía de peso y dimensiones reducidas y elevada autonomía" – Gloria Requena Ramos (2017)

[9] "16-Bit Flash Microcontrollers with 12-Bit Pipeline A/D, Sigma-Delta A/D, USB On-The-Go and XLP Technology" – Microchip (2016)

[10] "SD Specifications Part 1, Physical Layer Simplified Specification, Version 4.10" – SD Group (Panasonic, SanDisk, Toshiba) (2013)

[11] "Stand-Alone Linear Li-Ion / Li-Polymer Charge Management Controller" – Microchip (2009)

[12] "ULTRALOW-POWER 50-mA LOW-DROPOUT LINEAR REGULATORS" – Texas Instruments (2001)

[13] https://www.subtel.es – Subtel (2020)

[14] "Diseño e implementación de un sistema integral de extracción de parámetros para cuantificar la variabilidad del ritmo cardiaco. Aplicación experimental: impacto del tabaquismo en las arritmias cardiacas en ratones" – Javier Villar Valero (2020)

II. PRESUPUESTO

1. PRESUPUESTO

El presupuesto del presente proyecto se ha dividido en tres grandes grupos. En el primero, todo componente del dispositivo ya sea comercial o de encargo especial para nuestro diseño. El segundo donde se tienen en cuenta todos los gastos de los recursos utilizados (hardware y software), y el último donde figura el coste del trabajo realizado por parte del ingeniero industrial encargado del trabajo.

- Los precios de los componentes han sido sacados de internet y de otros proyectos de ingeniería.
- El coste del trabajo de un ingeniero se ha estipulado de 35€/h (valor medio actual).
- Los precios por hora de los recursos, se ha obtenido dividiendo su coste total entre su vida útil.
- En cuanto a las horas dedicadas a cada fase del proyecto, es imposible tener un control absoluto, por lo que muchas de ellas se han establecido por aproximación.

1.1 COMPONENTES

COMPONENTE	CANTIDAD (unidad)	PRECIO/unidad (€)	TOTAL (€)
PCB	1	8,670	8,670
PIC24FJ128GC006	1	4,960	4,960
ADS1292R	1	11,030	11,030
BATERIA LI-ION	1	6,500	6,500
USD	1	4,130	4,130
ZÓCALO USD	1	0,050	0,050
NCP551SN30	3	0,647	1,941
MINI USB	1	0,587	0,587
MCP73833	1	0,799	0,799
SG-3030CM	1	1,500	1,500
SWITCH 3 POSICIONES	1	0,100	0,100
PULSADOR	2	0,150	0,300
LED ROJO	2	0,930	1,860
LED VERDE	1	0,255	0,255
LED BLANCO	1	0,270	0,270
DIODO	1	0,331	0,331
CRISTAL	1	1,338	1,338
ELECTRODO	4	0,123	0,492
CABLE (0,5m)	-	-	0,234
BOBINA 4.7 uH	1	0,124	0,124
CONDENSADOR 1 µF	10	0,128	1,280
CONDENSADOR 10 µF	5	0,210	1,050
CONDENSADOR 2.2 nF	4	0,036	0,144
CONDENSADOR 4.7 nF	2	0,037	0,074
CONDENSADOR 100 nF	10	0,015	0,150
CONDENSADOR 20 pF	2	0,076	0,152
CONDENSADOR 47 pF	2	0,058	0,116
RESISTENCIA 47	3	0,005	0,015
RESISTENCIA 220	2	0,011	0,022
RESISTENCIA 560	10	0,016	0,160
RESISTENCIA 1K33	1	0,016	0,016
RESISTENCIA 2.2 K	1	0,003	0,003
RESISTENCIA 8.2 K	1	0,057	0,057
RESISTENCIA 10 K	2	0,003	0,006
TOTAL	-	-	48,716 €

Tabla 4 Coste de los componentes

1.2 RECURSOS

NOMBRE	CANTIDAD (horas)	PRECIO/hora (€)	TOTAL (€)
MACBOOK AIR	295	0,024	7,080
PICKit 3	5	0,002	0,010
EAGLE 9.5.1	110	0,114	12,540
CCS	50	0,039	1,950
MATLAB	50	0,038	1,900
TOTAL	-	-	23,480 €

Tabla 5 Coste de los recursos utilizados

1.3 DISEÑO Y FUNCIONAMIENTO

ACTIVIDAD	CANTIDAD (horas)	PRECIO/hora (€)	TOTAL (€)
ESTUDIO PREVIO	40	35	1400,000
DISEÑO ESQUEMÁTICO	60	35	2100,000
DISEÑO PCB	50	35	1750,000
MONTAJE	10	35	350,000
PROGRAMACIÓN	50	35	1750,000
MONITORIZACIÓN	50	35	1750,000
FUNCIONAMIENTO	20	35	700,000
RESULTADOS	5	35	175,000
PRUEBAS	15	35	525,000
REDACCIÓN MEMORIA	80	35	2800,000
TOTAL	-	-	13300,000 €

Tabla 6 Coste del diseño y de funcionamiento

1.4 COSTE TOTAL

NOMBRE	TOTAL (€)
COMPONENTES	48,716
RECURSOS	23,480
DISEÑO Y FUNCIONAMIENTO	13300,000
TOTAL	13372,196 €

Tabla 7 Coste total

III. PLANOS

1. PLANOS

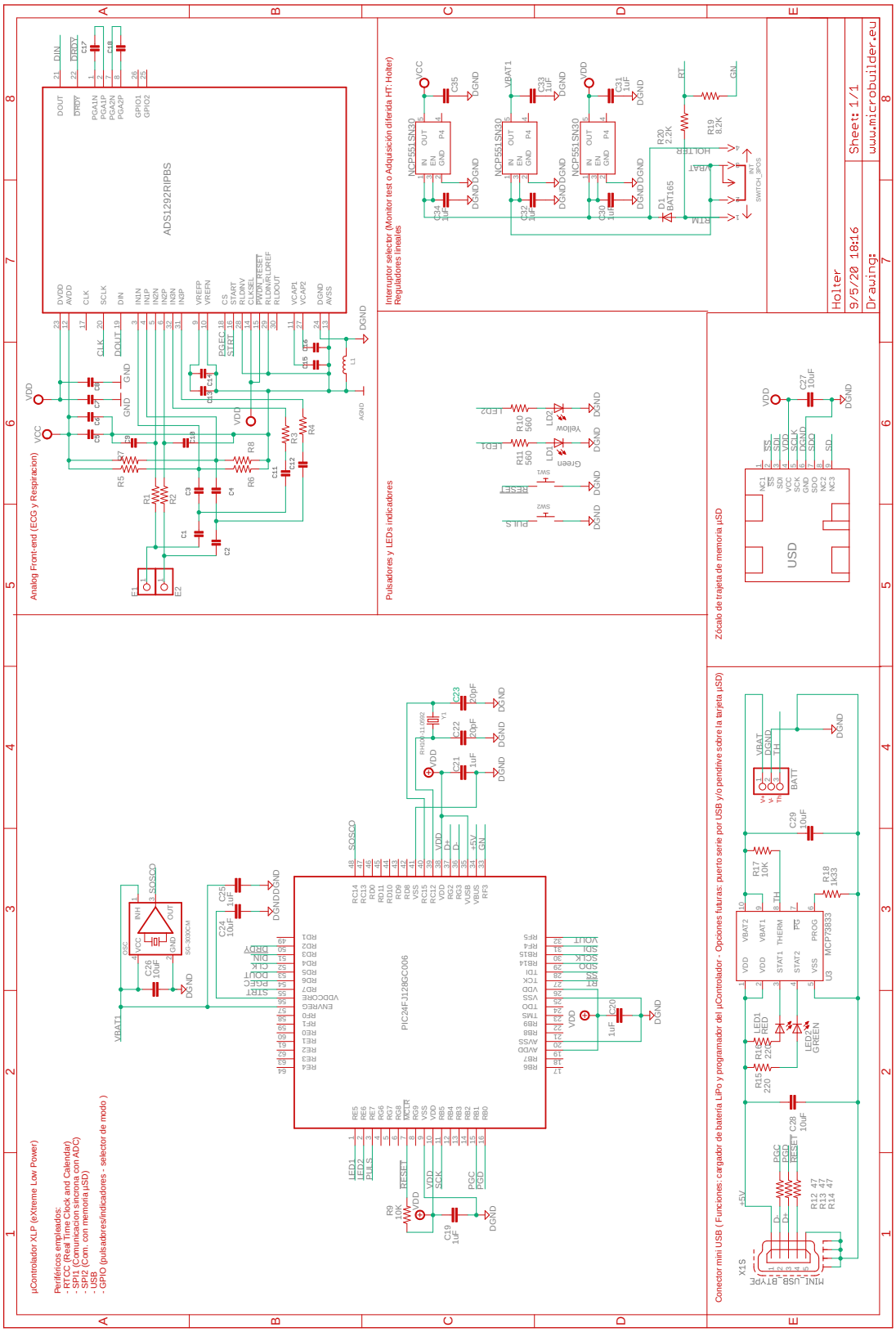

Figura 51 Plano de esquemático completo del Holter

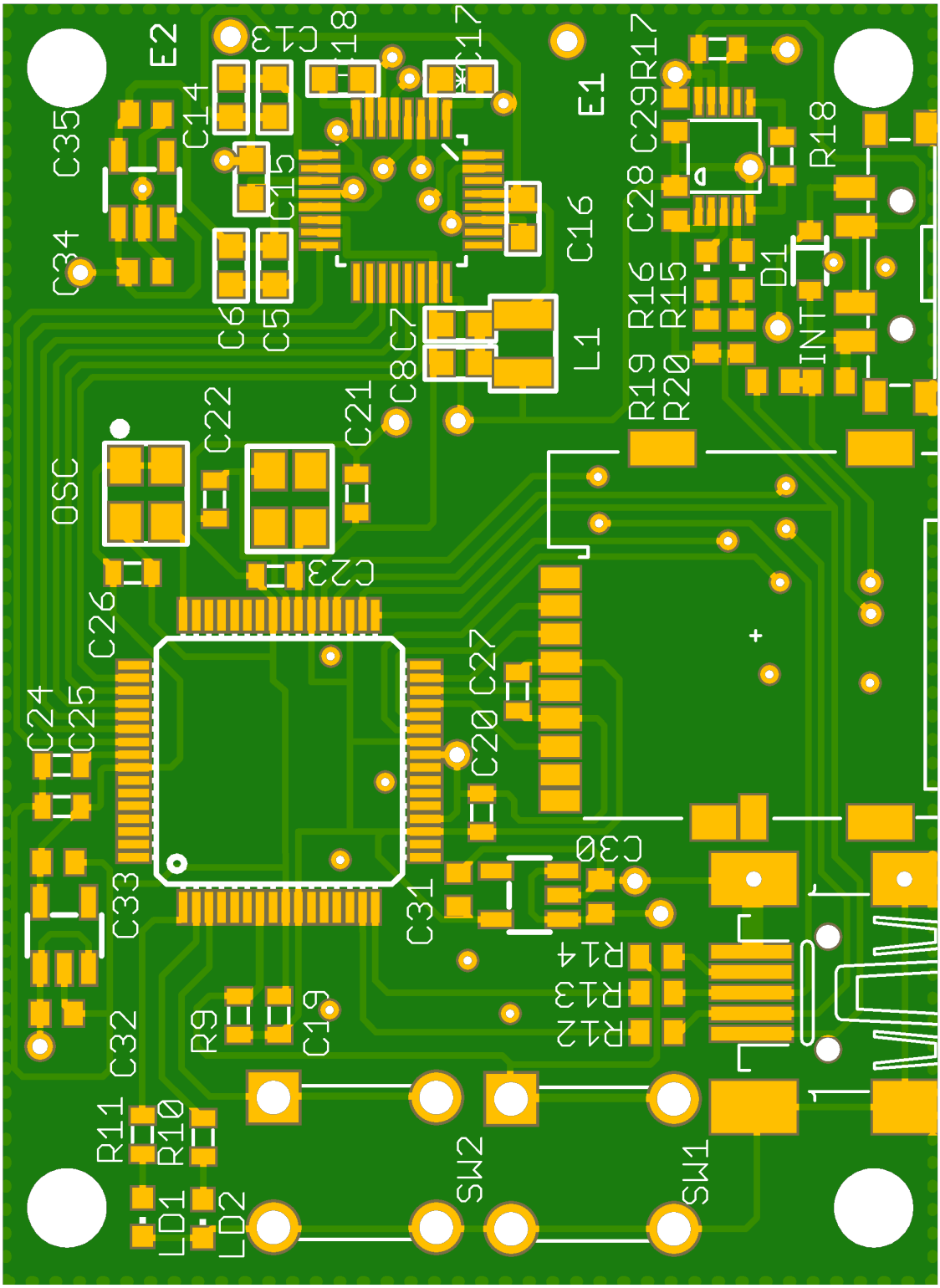

Figura 52 PCB Top Side

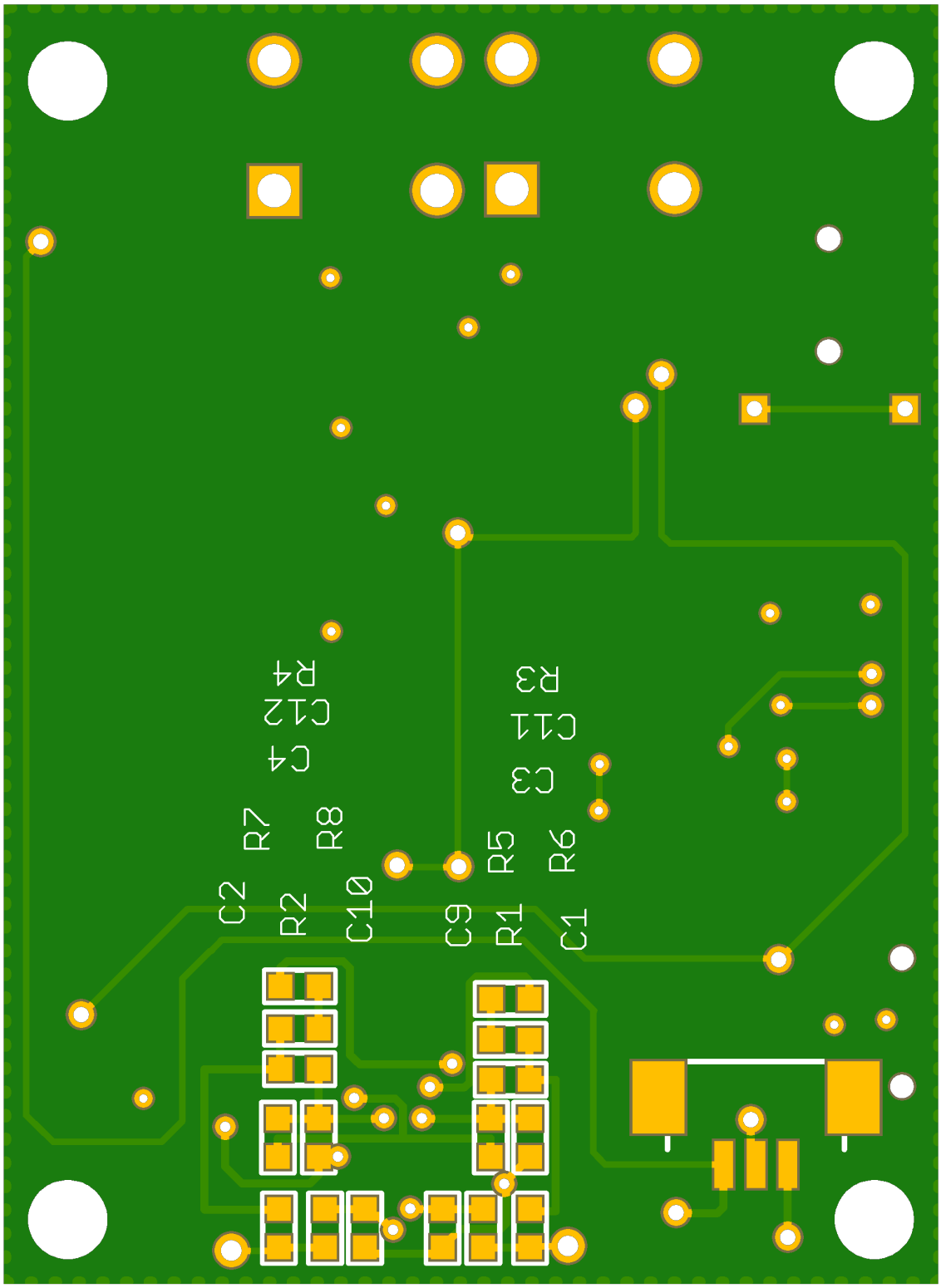

Figura 53 PCB Bot Side

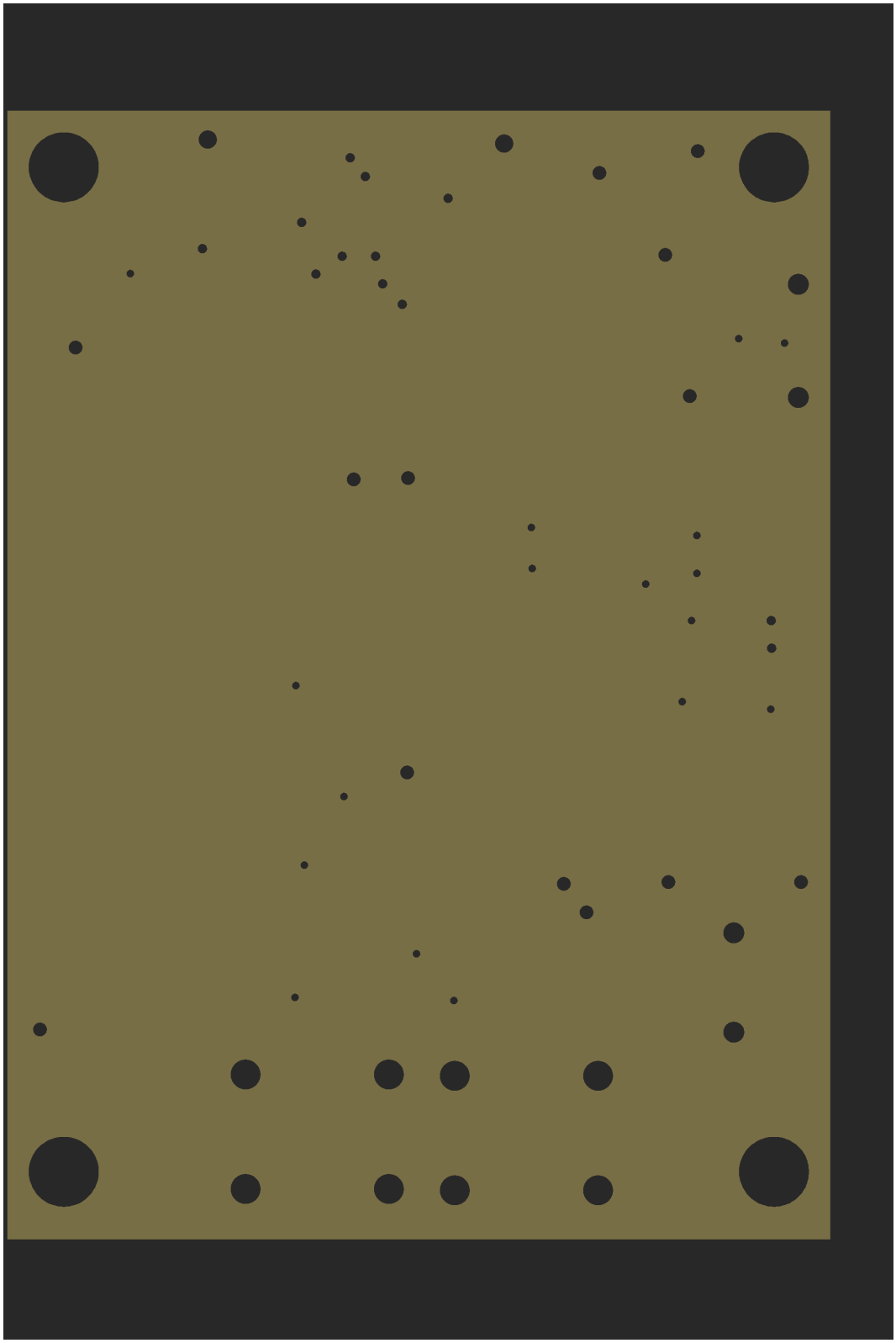

Figura 54 PCB Drills

IV. ANEXOS

1. PROGRAMA

```
#ifndef main_h
#define main_h 0
#include <24FJ128GC006.h>
#device ADC=12
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//| CW1: FLASH CONFIGURATION WORD 1
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//|     15     |     14      |     13      |     12      |     11      |     10      |      9      |      8      |
//|     --     |   JTAGEN    |     GCP     |    GWRP     |   ~DEBUG    |   ~LPCFG    |    ICS1     |    ICS0     |
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//|      7     |      6      |      5      |      4      |      3      |      2      |      1      |      0      |
//|  FWDTEN1   |   FWDTEN0   |   WINDIS    |    FWPSA    |   WDTPS3    |   WDTPS2    |   WDTPS1    |   WDTPS0    |
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#fuses NOJTAG           // JTAG port is disabled
#fuses NOPROTECT        // Code protection is disabled
#fuses WRT              // Writes to program memory are allowed
#fuses LVR              // Low-power, low-voltage/retention regulator is enabled and controlled in firmware - RETEN bit
#fuses ICSP1            // Emulator functions are shared with PGEC1/PGED1
#fuses NOWDT            // WDT is disabled; SWDTEN bit is disabled
//#fuses WINDIS         // Standard Watchdog Timer is enabled
//#fuses WDT128         // WDT Prescaler Ratio Select bit: Prescaler ratio of 1:128
//#fuses WPOSTS16       // Watchdog Timer Postscaler Select bits-> 1:32,768 (NOT USED)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//| CW2: FLASH CONFIGURATION WORD 2
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//|     15     |     14      |     13      |     12      |     11      |     10      |      9      |      8      |
//|    IESO    |   VBTBOR    |   WDTCMX    |  ALTCVREF   |  ALTCVREF   |   FNOSC2    |   FNOSC1    |   FNOSC0    |
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//|      7     |      6      |      5      |      4      |      3      |      2      |      1      |      0      |
//|   FCKSM1   |   FCKSM0    |  OSCIOFCN   |   WDTCLK1   |   WDTCLK0   |     --      |   POSCMD1   |   POSCMD0   |
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//#fuses NOIESO         // Internal External Switchover bit: IESO mode (Two-Speed Start-up) is disabled
                                        // VBAT Brownout Reset is enabled (NO ESTA IMPLEMENTADO EN EL COMPILADOR)
//#fuses WDTCMX         // WDT Clock Multiplex Control bit: Enables WDT clock multiplexing
//#fuses CVREFNORM      // External CVREF+/CVREF- Location Select bit (NOT USED)
//#fuses VREFNORM       // External AVREF+/AVREF- Location Select bit (NOT USED)
//#fuses FRC_PS         // Initial Oscillator Select bits: Fast RC Oscillator with Postscaler (FRCDIV)
//#fuses CKSNOFSM       // Clock switching is enabled, Fail-Safe Clock Monitor is disabled
#fuses OSCIO            // OSCO Pin Configuration bit: OSCO/CLKO/RC15 functions as port I/O (RC15)
//#fuses WDTCLK_SOSC    // WDT Clock Source Select bits: SOSC input
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//| CW3: FLASH CONFIGURATION WORD 3
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//|     15     |     14      |     13      |     12      |     11      |     10      |      9      |      8      |
//|   WPEND    |   WPCFG     |   WPDIS     |  BOREN      |     --      |  WDTWIN1    |  WDTWIN0    |  SOSCSEL    |
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//|      7     |      6      |      5      |      4      |      3      |      2      |      1      |      0      |
//|     --     |  WPFP6      |   WPFP5     |  WPFP4      |  WPFP3      |  WPFP2      |  WPFP1      |  WPFP0      |
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//#fuses WPEND          // Segment Write Protection End Page
//#fuses NOWPCFG    // Configuration Word Code Page Write Protection
//#fuses WPDIS          // Segment Write Protection Disabled
//#fuses NOBROWNOUT // Brown-out Reset Disabled
//#fuses WDTWIN_50% // Watchdog Timer Window Width Select bits
//#fuses SOSC_SEL   // SOSC Selection bit: SOSC circuit is selected
//#fuses NOWPFP         // Write-Protected Code Segment Boundary Page bits(
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//| CW4: FLASH CONFIGURATION WORD 4
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//|     15     |     14      |     13      |     12      |     11      |     10      |      9      |      8      |
//|  IOL1WAY   |  I2C2SEL    |  PLLDIV3    |  PLLDIV2    |  PLLDIV1    |  PLLDIV0    |  RTCBAT     |  DSSWEN     |
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//|      7     |      6      |      5      |      4      |      3      |      2      |      1      |      0      |
//|  DSWDTEN   |  DSBOREN    |  DSWDTOSC   |  DSWDPS4    |  DSWDPS3    |  DSWDPS2    |  DSWDPS1    |  DSWDPS0    |
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//#fuses PLL1                   // USB 96 MHz PLL Prescaler Select bits: Oscillator input used directly (4 MHz input)
#fuses RTCBAT                   // RTCC operation continues when the device is in VBAT mode
//#fuses DS_SW                  // Deep Sleep operation is enabled and controlled by the DSEN bit
```

```
#fuses NODSWDT                  // Deep Sleep Watchdog Timer Enable bit: Deep Sleep WDT is disabled
#fuses NODSBOR                  // Deep Sleep Brown-out Reset
//#fuses DSWDTCK_SOSC   // Deep Sleep Watchdog Timer Clock Select bit: Clock source is SOSC
//#fuses DSWDT_25DAYS   // Deep Sleep Watchdog Timer Postscaler
#fuses DEBUG
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//Reset -> RCDIV 2..0= 001 = 4 MHz (divide-by-2) (default)
// MACROS  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#define ON( pin)   output_high(pin)
#define OFF(pin)   output_low( pin)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#define LED1          PIN_E5   //                   ( 1)
#define LED2          PIN_E6   //                   ( 2)
#define PULS          PIN_E7   //                   ( 3)
//                    PIN_G6   // uSD PIN SDO       ( 4)
//                    PIN_G7   // uSD PIN SCK       ( 5)
//                    PIN_G8   // uSD PIN SDI       ( 6)
//                    ~RESET   //                   ( 7)
//                    PIN_G9   // DAC1              ( 8)|
//                    VSS      //                   ( 9)
//                    VDD      //                   (10)
//                    PIN_B5   //                   (11)
//                    PIN_B4   //                   (12)
//                    PIN_B3   //                   (13)
//                    PIN_B2   //                   (14)
//                    PIN_B1   // PGC1              (15)
//                    PIN_B0   // PGD1              (16)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//                    PIN_B6   //                   (17)
//                    PIN_B7   //                   (18)
//                    AVDD     //                   (19)
//                    AVSS     //                   (20)
//                    PIN_B8   //      @ GND        (21)
//                    PIN_B9   // CH0+              (22)
//                    PIN_B10  // CH0- @ GND        (23)
//                    PIN_B11  // CH1+              (24)
//                    VSS      // CH1- @ GND        (25)
//                    SVDD     //                   (26)
#define RT            PIN_B12  //                   (27)
#define CS            PIN_B13  //                   (28)
#pin_select SDI2 =     PIN_B14  //                   (29)
#pin_select SCK2OUT =  PIN_B15  //                   (30)
#pin_select SDO2 =     PIN_F4   //                   (31)
#pin_select U1TX =     PIN_F5   //                   (32)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#define GN            PIN_F3   //                   (33)
//                    PIN_F7   //                   (34)
//               VUSB VDD      //                   (35)
//                    PIN_G3   //                   (36)
//                    PIN_G2   //                   (37)
//                    VDD      //                   (38)
//                    PIN_C12  //                   (39)
//                    PIN_C15  //                   (40)
//                    VSS      //                   (41)
#pin_select U1RX =     PIN_D8   //                   (42)
//                    PIN_D9   //                   (43)
//                    PIN_D10  //                   (44)
//                    PIN_D11  //                   (45)
//                    PIN_D0   //                   (46)
//                    PIN_C13  //                   (47)
//                    PIN_C14  // Secund Oscillator (48)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//                    PIN_D1   //                   (49)
#pin_select INT1 =     PIN_D2   // DRDY              (50)
#pin_select SDI1 =     PIN_D3   // MISO              (51)
#pin_select SCK1OUT =  PIN_D4   // SCK               (52)
#pin_select SDO1 =     PIN_D5   // MOSI              (53)
#define CS_ADS        PIN_D6   // CS                (54)
#define STRT          PIN_D7   //                   (55)
//                    VCAP     //                   (56)
//                    VDD      //                   (57)
//                    PIN_F0   //                   (58)
//                    PIN_F1   //                   (59)
//                    PIN_E0   //                   (60)
//                    PIN_E1   //                   (61)
//                    PIN_E2   //                   (62)
//                    PIN_E3   //                   (63)
//                    PIN_E4   //                   (64)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#byte   SPI1BUF    =   0x0248
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#fuses HS
#fuses PR       // Primary Oscillator Configuration bits: Primary Oscillator mode is enabled
#use delay (crystal = 8000000, clock = 8000000)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#endif
```

```
#ifndef INC_ADS1292R
#define INC_ADS1292R 1
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
// System Commands
#define WAKEUP       0b00000010      // Wake-up from standby mode
#define STANDBY      0b00000100      // Enter standby mode
#define RESET        0b00000110      // Reset the device
#define START        0b00001000      // Start/restart (synchronize) conversions
#define STOP         0b00001010      // Stop conversion
#define OFFSETCAL    0b00011010      // Data Read Commands
#define RDATACT      0b00010000      // Enable Read Data Continuous mode (default mode at power-up)
#define SDATACT      0b00010001      // Stop Read Data Continuously mode
#define RDATA        0b00010010      // Read data by command; supports multiple read back
#define RREG0        0b00100000      // Read n nnnn registers starting at address r rrrr
                                     // first byte 001r rrrr (2xh)(2) - second byte 000n nnnn(2)
#define WREG0        0b01000000      // Write n nnnn registers starting at address r rrrr
                                     // first byte 010r rrrr (2xh)(2) - second byte 000n nnnn(2)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
/* ID: ID Control Register (Factory-Programmed, Read-Only)
 * Address = 00h
 * ------------------------------------------------------------------------
 * |BIT 7  |BIT 6    |BIT 5   |BIT 4 |BIT 3   |BIT 2   |BIT 1   |BIT 0   |
 * |-------|---------|--------|------|--------|--------|--------|------- |
 * |REV_ID4| REV_ID3 |REV_ID2 |  N/A |   N/A  |   N/A  |REV_ID1 |REV_ID0 |
 * ------------------------------------------------------------------------
 *
 * The ID Control Register is programmed during device manufacture to indicate device characteristics.
 *
 * Bits[7:5] : Revision identification These bits identify the device revision.
 * 000 = TBD
 * 001 = TBD
 * 010 = TBD
 * 100 = TBD
 * 101 = TBD
 * 110 = TBD
 * 111 = TBD
 *
 * Bits[4:2] N/A
 *
 * BIT[1:0] : Revision identification These bits identify the device revision.
 * 00 = ADS1191
 * 01 = ADS1192
 * 10 = ADS1291
 * 11 = ADS1292/2R
 *
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 * CONFIG1: Configuration Register 1
 * Address = 01h
 *-----------------------------------------------------------------
 * |BIT 7       |BIT 6 |BIT 5  |BIT 4 |BIT 3 |BIT 2 |BIT 1  |BIT 0 |
 * |------------|------|-------|------|------|------|-------|------|
 * |SINGLE_SHOT |   0  |   0   | 0    |  0   | OSR2 | OSR1  | OSR0 |
 *-----------------------------------------------------------------
 *
 * Bit 7 SINGLE_SHOT: Single-shot conversion
 *    This bit sets the conversion mode
 *       0 = Continuous conversion mode (default)
 *       1 = Single-shot mode
 *    NOTE: Pulse mode is disabled when individual data rate control is selected.
 *
 * Bits[6:3] Must be set to '0'
 *
 *  Bits[2:0] OSR[2:0]: Channel oversampling ratio
 *  These bits determine the oversampling ratio of both channel 1 and channel 2.
 * ------------------------------------------------------------
 *          BIT    | OVERSAMPLING RATIO |   SAMPLE RATE      |
 *          000    | fMOD/1024          |   125SPS           |
 *          001    | fMOD/512           |   250SPS           |
```

```
*          010    | fMOD/256           |   500SPS           |
*          011    | fMOD/128           |   1024SPS          |
*          100    | fMOD/128           |   2048SPS          |
*          101    | Do Not Use         |   Do Not Use       |
*          110    | Do Not Use         |   Do Not Use       |
*          111    | Do Not Use         |   Do Not Use       |
* ------------------------------------------------------------
*
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* CONFIG2: Configuration Register 2
* Address = 02h
*-------------------------------------------------------------------------------
* |BIT 7  |BIT 6         |BIT 5      | BIT 4   |  BIT 3 |BIT 2  |BIT 1    |BIT 0    |
* |-------|--------------|-----------|---------|--------|-------|---------|---------|
* |   1   |PDB_LOFF_COMP |PDB_REFBUF| VREF_4V | CLK_EN |   0   |INT_TEST |TEST_FREQ|
*-------------------------------------------------------------------------------
*
* Configuration Register 2 configures the test signal generation. See the Input Multiplexer section for more details.
*    Bit 7 Must always be set to '1'
*
*  Bit 6 PDB_LOFF_COMP : Lead-off comparator power-down
*     This bit powers down the lead-off comparators.
*     0 = Lead-off comparators disabled (default)
*     1 = Lead-off comparators enabled
*
*  Bit 5 PDB_REFBUF : Reference buffer power-down
*       This bit powers down the internal reference buffer so that the external reference can be used.
*     0 = Reference buffer is powered down (default)
*       1 = Reference buffer is enabled
*
*   Bit 4 VREF_4V: Enables 4-V reference
*       This bit chooses between 2.4V and 4V reference.
*      0 = 2.4-V reference (default)
*      1 = 4-V reference
*
* Bit 3 CLKOUT_EN: CLK connection
*      This bit determines if the internal oscillator signal is connected to the CLK pin when an internal oscillator is used.
*      0 = Oscillator clock output disabled (default)
*      1 = Oscillator clock output enabled
*
*    Bit 2 Must be set to '0'
*
*  Bit 1 TEST_AMP: Test signal amplitude
*      This bit determines the test signal amplitude.
*       0 = No test signal (default)
*      1 = ±(VREFP ñ VREFN)/2400
*
*  Bit 0 TEST_FREQ: Test signal frequency.
*      This bit determines the test signal frequency.
*      0 = At dc (default)
*      1 = Square wave at 1 Hz
*
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*
* LOFF: Lead-Off Control Register
* Address = 03h
*-------------------------------------------------------------------------------------------
* | BIT 7    |BIT 6    |BIT 5    |BIT 4     | BIT 3      |BIT 2       |BIT 1         | BIT 0     |
* |----------|-------  |-------  |----------|----------- |----------   |--------------|-----------|
* |COMP_TH2  |COMP_TH1 |COMP_TH0 | 0        | ILEAD_OFF1 | ILEAD_OFF0 |0             | FLEAD_OFF|
*-------------------------------------------------------------------------------------------
*
* The Lead-Off Control Register configures the Lead-Off detection operation.
*
* Bits[7:5] Lead-off comparator threshold
*      These bits determine the lead-off comparator threshold.
*      Comparator positive side in %
*      000 = 95 (default)
```

```
*      001 = 92.5
*      010 = 90
*      011 = 87.5
*      100 = 85
*      101 = 80
*      110 = 75
*      111 = 70
*      Comparator negative side in %
*      000 = 5 (default)
*      001 = 7.5
*      010 = 10
*      011 = 12.5
*      100 = 15
*      101 = 20
*      110 = 25
*      111 = 30
*
* Bit 4 Must be set to 1
*
* Bits[3:2] ILEAD_OFF[1:0]: Lead-off current magnitude
*      These bits determine the magnitude of current for the current lead-off mode.
*      00 = 6 nA (default)
*      01 = 24 nA
*      10 = 6 microA
*      11 = 24 microA
*
* Bit 1 Must be set to '0'
*
* Bit 0 FLEAD_OFF: Lead-off frequency
*      This bit generates the LEAD_OFF_CLK signal. It also generates AC_LEAD_OFF, which is '1' when FLEAD_OFF is '1'.
*      0 = At dc lead-off detect (default)
*      1 = At ac lead-off detect at DECICLK/4 (500 Hz for an 2-kHz output rate)
*
*
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*   CHnSET: Individual Channel Settings
*    Address = 04h
*-------------------------------------------------------------------------------
* | BIT 7    |BIT 6    |BIT 5   |  BIT 4  | BIT 3  | BIT 2  |  BIT 1  | BIT 0     |
* |----------|---------|--------|---------|--------|--------|---------|---------|
* |   PD1    |   GAIN1_2 |GAIN1_1 | GAIN1_0 |  MUX1_3| MUX1_2 |  MUX1_1 | MUX1_0  |
*-------------------------------------------------------------------------------
*
* The CH1SET Control Register configures the power mode, PGA gain, and multiplexer settings channels.
*
* Bit 7 PD1: Channel 1 power-down
*      0 = Normal operation (default)
*      1 = Channel 1 power-down
*
* Bits[6:4 ] GAIN1[2:0]: Channel 1 PGA gain setting
*      These bits determine the PGA gain setting for channel 1.
*      000 = 6 (default)
*      001 = 1
*      010 = 2
*      011 = 3
*      100 = 4
*      101 = 8
*      110 = 12
*
* Bits[3:0] MUX1[3:0]: Channel 1 input selection
*      These bits determine the channel 1 input selection.
*      0000 = Normal electrode input (default)
*      0001 = Input shorted (for offset measurements)
*      0010 = RLD_MEASURE
*      0011 = VDD/2 for supply measurement
*      0100 = Temperature sensor
*      0101 = Cal signal
*      0110 = RLD_DRP (positive electrode is the driver)
```

```
*      0111 = RLD_DRM (negative electrode is the driver)
*      1000 = Reserved
*      1001 = MUX RESPP/RESPN to INP/INM
*      1010 = Reserved
*
*
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*   CHnSET: Individual Channel Settings
*    Address = 05h
*-------------------------------------------------------------------------
* | BIT 7    |BIT 6    |BIT 5   |  BIT 4  | BIT 3  | BIT 2  |  BIT 1  | BIT 0     |
* |----------|---------|--------|---------|--------|--------|---------|---------|
* |   PD2    |   GAIN2_2 |GAIN2_1 | GAIN2_0 |  MUX2_3| MUX2_2 |  MUX2_1 | MUX2_0  |
*-------------------------------------------------------------------------
*
* The CH1SET Control Register configures the power mode, PGA gain, and multiplexer settings channels.
*
* Bit 7 PD2: Channel 1 power-down
*      0 = Normal operation (default)
*      1 = Channel 1 power-down
*
* Bits[6:4 ] GAIN2[2:0]: Channel 1 PGA gain setting
*      These bits determine the PGA gain setting for channel 1.
*      000 = 6 (default)
*      001 = 1
*      010 = 2
*      011 = 3
*      100 = 4
*      101 = 8
*      110 = 12
*
* Bits[3:0] MUX2[3:0]: Channel 1 input selection
*      These bits determine the channel 1 input selection.
*      0000 = Normal electrode input (default)
*      0001 = Input shorted (for offset measurements)
*      0010 = RLD_MEASURE
*      0011 = VDD/2 for supply measurement
*      0100 = Temperature sensor
*      0101 = Cal signal
*      0110 = RLD_DRP (positive electrode is the driver)
*      0111 = RLD_DRM (negative electrode is the driver)
*      1000 = Reserved
*      1001 = MUX RESPP/RESPN to INP/INM
*      1010 = Reserved
*
*
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*
* RLD_SENSP
* Address = 06h
*-----------------------------------------------------------------------------------
* | BIT 7    |BIT 6    |BIT 5   |  BIT 4          | BIT 3  | BIT 2  |  BIT 1  | BIT 0     |
* |----------|---------|--------|-----------------|--------|--------|---------|---------|
* |  CHOP1   |   CHOP0   |PD_RLD  |  RLD_LOFF_SENS  | RLDN2  | RLDP2  |  RLDN1  | RLDP1   |
*-----------------------------------------------------------------------------------
* This register controls the selection of the positive and negative signals from each channel for right leg drive
* derivation. See the Right Leg Drive (RLD DC Bias Circuit) subsection of the ECG-Specific Functions section for
* details.
*
* Bits[7:6] CHOP[1:0]: Chop frequency
*      These bits determine PGA chop frequency
*      00 = fMOD/16
*      01 = fMOD/32
*      10 = fMOD/2
*      11 = fMOD/4
*
* Bit 5 PDB_RLD: RLD buffer power
*      This bit determines the RLD buffer power state.
```

```
*      0 = RLD buffer is powered down (default)
*      1 = RLD buffer is enabled
*
* Bit 4 RLD_LOFF_SENSE: RLD lead-off sense function
*      This bit enables the RLD lead-off sense function.
*      0 = RLD lead-off sense is disabled (default)
*      1 = RLD lead-off sense is enabled
*
* Bit 3 RLDN2: Channel 2 RLD negative inputs
*      This bit controls the selection of negative inputs from channel 2 for right leg drive derivation.
*      0 = Not connected (default)
*      1 = RLD connected to IN2N
*
* Bit 2 RLDP2: Channel 2 RLD positive inputs
*      This bit controls the selection of positive inputs from channel 2 for right leg drive derivation.
*      0 = Not connected (default)
*      1 = RLD connected to IN2P
*
* Bit 1 RLDN1: Channel 1 RLD negative inputs
*      This bit controls the selection of negative inputs from channel 1 for right leg drive derivation.
*      0 = Not connected (default)
*      1 = RLD connected to IN1N
*
* Bit 0 RLDP1: Channel 1 RLD positive inputs
*      This bit controls the selection of positive inputs from channel 1 for right leg drive derivation.
*      0 = Not connected (default)
*      1 = RLD connected to IN1P
*
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*
* LOFF_SENS
* Address = 07h
*--------------------------------------------------------------------------------
* | BIT 7    |BIT 6    |BIT 5   |  BIT 4  | BIT 3  | BIT 2  |  BIT 1  | BIT 0     |
* |----------|---------|--------|---------|--------|--------|---------|---------|
* | 0        |   0     |FLIP2   |  FLIP1  | LOFFN2 | LOFFP2 |  LOFFN1 | LOFFP1  |
*--------------------------------------------------------------------------------
*   This register selects the positive and negative side from each channel for lead-off detection. See the Lead-Off
*   Detection subsection of the ECG-Specific Functions section for details. Note that the LOFF_STAT register bits
*   should be ignored if the corresponding LOFF_SENS bits are set to '1'.
*
*   Bits[7:6] Must be set to '0'
*
*   Bit 5 FLIP2: Current direction selection
*      This bit controls the direction of the current used for lead off derivation.
*      0 = Disabled (default)
*      1 = Enabled
*
*   Bit 4 FLIP1: Current direction selection
*      This bit controls the direction of the current used for lead off derivation.
*      0 = Disabled (default)
*      1 = Enabled
*
*   Bit 3 LOFFN2: Channel 2 lead-off detection negative inputs
*      This bit controls the selection of negative input from channel 2 for lead off detection.
*      0 = Disabled (default)
*      1 = Enabled
*
*   Bit 2 LOFFP2: Channel 2 lead-off detection positive inputs
*      This bit controls the selection of positive input from channel 2 for lead off detection.
*      0 = Disabled (default)
*      1 = Enabled
*
*   Bit 1 LOFFN1: Channel 1 lead-off detection negative inputs
*      This bit controls the selection of negative input from channel 1 for lead off detection.
*      0 = Disabled (default)
*      1 = Enabled
```

```
*   Bit 0 LOFFP1: Channel 1 lead-off detection positive inputs
*      This bit controls the selection of positive input from channel 1 for lead off detection.
*      0 = Disabled (default)
*      1 = Enabled
*
*
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*
* LOFF_STAT
* Address = 08h
*-------------------------------------------------------------------------------------------------
* | BIT 7    |BIT 6    |BIT 5   |  BIT 4     | BIT 3    | BIT 2    |  BIT 1    | BIT 0     |
* |----------|---------|--------|------------|----------|----------|-----------|-----------|
* |  0         |MOD_FREQ |0       |  RLD_STAT  | IN2N_OFF | IN2P_OFF | IN1N_OFF  | IN1P_OFF   |
*-------------------------------------------------------------------------------------------------
*   This register stores the status of whether the positive or negative electrode on each channel is on or off. See the
*   Lead-Off Detection subsection of the ECG-Specific Functions section for details. Ignore the LOFF_STAT values
*   if the corresponding LOFF_SENS bits are not set to '1'.
*   '0' is lead-on (default) and '1' is lead-off. When the LOFF_SENS bits are '0', the LOFF_STAT bits should be
*   ignored.
*
*    Bit 7 Must be set to '0'
*
*    Bit 6 MOD_FREQ: System frequency selection
*      This bit sets the modular frequency. Two external clock values are supported: 512 kHz and 2.048 MHz. This bit must be set
*      so that MOD_FREQ = 128 kHz.
*      0 = External_CLK/4 (default)
*      1 = External_CLK/16
*
*    Bit 5 Must be set to '0'
*
*    Bit 4 RLD_STAT: RLD lead-off status
*      This bit determines the status of RLD.
*      0 = RLD is connected (default)
*      1 = RLD is not connected
*
*    Bit 3 IN2N_OFF: Channel 2 negative electrode status
*      This bit determines if the channel 2 negative electrode is connected or not.
*      0 = Connected (default)
*      1 = Not connected
*
*    Bit 2 IN2P_OFF: Channel 2 positive electrode status
*      This bit determines if the channel 2 positive electrode is connected or not.
*      0 = Connected (default)
*      1 = Not connected
*
*    Bit 1 IN1N_OFF: Channel 1 negative electrode status
*      This bit determines if the channel 1 negative electrode is connected or not.
*      0 = Connected (default)
*      1 = Not connected
*
*    Bit 0 IN1P_OFF: Channel 1 positive electrode status
*      This bit determines if the channel 1 positive electrode is connected or not.
*      0 = Connected (default)
*      1 = Not connected
*
*
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*
* RESP1: Respiration Control Register 1
* Address = 09h
*  -------------------------------------------------------------------------------------------
* | BIT 7       |  BIT 6       |BIT 5    |  BIT 4  | BIT 3    | BIT 2    |  BIT 1  | BIT 0   |
* |-------------|--------------|---------|---------|----------|----------|---------|---------|
* | RESP_MOD_EN |   RESP_MOD_EN |RESP_PH3 |RESP_PH2 | RESP_PH1 | RESP_PH0 |  1      | REP_CTL |
*  -------------------------------------------------------------------------------------------
*    This register controls the respiration functionality.
*    Bit 7 RESP_DEMOD_EN1: Enables respiration demodulation circuitry
```

```
*       This bit enables/disables the demodulation circuitry on channel 1.
*       0 = RESP demodulation circuitry turned off (default)
*       1 = RESP demodulation circuitry turned on
*
*    Bit 6 RESP_MOD_EN: Enables respiration modulation circuitry
*       This bit enables/disables the modulation circuitry on channel 1.
*       0 = RESP modulation circuitry turned off (default)
*       1 = RESP modulation circuitry turned on
*
*    Bits[5:2] RESP_PH[3:0]: Respiration phase(1)
*       These bits control the phase of the respiration demodulation control signal.
*       RESP_PH[3:0] RESP_CLK = 32kHz RESP_CLK = 64kHz
*       0000 0∞ (default) 0∞ (default)
*       0001 11.25∞ 22.5∞
*       0010 22.5∞ 45∞
*       0011 33.75∞ 67.5∞
*       0100 45∞ 90∞
*       0101 56.25∞ 112.5∞
*       0110 67.5∞ 135∞
*       0111 78.75∞ 157.5∞
*       1000 90∞ Not available
*       1001 101.25∞ Not available
*       1010 112.5∞ Not available
*       1011 123.75∞ Not available
*       1100 135∞ Not available
*       1101 146.25∞ Not available
*       1110 157.5∞ Not available
*       1111 168.75∞ Not available
*
*    (1) The RESP_PH3 bit is ignored when RESP_CLK = 64kHz.
*    Bit 1 Must be set to '1'
*
*    Bit 0 RESP_CTRL: Respiration control
*       This bit sets the mode of the respiration circuitry.
*       0 = Internal respiration with internal clock
*       1 = Internal respiration with external clock
*
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*
* RESP2: Respiration Control Register 2
* Address = 0Ah
*  ---------------------------------------------------------------------------------------
* | BIT 7    |BIT 6     | BIT 5    |   BIT 4  |  BIT 3   |  BIT 2   |   BIT 1    | BIT 0    |
* |----------|----------|----------|----------|----------|----------|------------|----------|
* |CALIB_ON  |0         |   0      |     0    |     0    | RESP_FREQ| RLDREF_INT |  1       |
*  ---------------------------------------------------------------------------------------
*    This register controls the respiration functionality.
*    Bit 7 CALIB_ON: Calibration on
*       This bit is used to enable offset calibration.
*       0 = Off (default)
*       1 = On
*
*    Bits[6:3] Must be '0'
*
*    Bit 2 RESP_FREQ: Respiration control frequency
*       This bit controls the respiration control frequency when RESP_CTRL[1:0] = 10.
*       0 = 32 kHz (default)
*       1 = 64 kHz
*
*    Bit 1 RLDREF_INT: RLDREF signal
*       This bit determines the RLDREF signal source.
*       0 = RLDREF signal fed externally
*       1 = RLDREF signal (AVDD ñ AVSS)/2 generated internally (default)
*
*    Bit 0 Must be set to '1;
```

```
 * GPIO: General-Purpose I/O Register
 * Address = 0Ch
 *  ------------------------------------------------------------------------------
 * | BIT 7    |BIT 6     | BIT 5    |   BIT 4  |  BIT 3   |  BIT 2   |   BIT 1  | BIT 0    |
 * |----------|----------|----------|----------|----------|----------|----------|----------|
 * | 0        | 0        |      0   |   0      | GPIOC2   | GPIOC1   | GPIOD2   | GPIOD1   |
 *  ------------------------------------------------------------------------------
 *    This register controls the GPIO pins.
 *    Bits[7:4] Must be '0'
 *
 *    Bits[3:2] GPIOC[2:1]: GPIO 1 and 2 control
 *       These bits determine if the corresponding GPIOD pin is an input or output.
 *       0 = Output
 *       1 = Input (default)
 *
 *    Bits[1:0] GPIOD[2:1]: GPIO 1 and 2 data
 *    These bits are used to read and write data to the GPIO ports.
 *    When reading the register, the data returned correspond to the state of the GPIO external pins, whether they are
 *    programmed as inputs or as outputs. As outputs, a write to the GPIOD sets the output value. As inputs, a write to the
 *    GPIOD has no effect. GPIO is not available in certain respiration modes.
 *  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 */
enum   option{
                              Normal   = 0b0000 ,// Normal electrode input (default)
                              Shorted  = 0b0001 ,// Input shorted (for offset measurements)
                              RLD_Meas = 0b0010 ,// RLD_MEASURE
                              MVDD     = 0b0011 ,// MVDD = [0.5(AVDD + AVSS)] for supply measurement (for channel2 0.25)
                              Temp     = 0b0100 ,// Temperature sensor
                              Test     = 0b0101 ,// Test signal
                              RLD_DRP  = 0b0110 ,// RLD_DRP (plus: positive input is connected to RLDIN)
                              RLD_DRM  = 0b0111 ,// RLD_DRM (minus:negative input is connected to RLDIN)
                              RLD_DRPM = 0b1000 ,// RLD_DRPM (both positive and negative inputs are connected to RLDIN)
                              IN3_to_1 = 0b1001  // Route IN3P and IN3N to channel 1 inputs
                              //0b1010 = Reserved
                              };
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int16 ADS_init(option);
void on_ADS(void);
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#endif
```

```
#include "_main.h"
#include "ADS1292.h"
#include "SD_Card.h"
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int16 ADS_init(option opcion)
{
 unsigned int8  configbyte= opcion;
 unsigned int16 ret=0;

                ON(  CS_ADS  );
                OFF( STRT    );
                delay_ms(100);
                        setup_spi ( SPI_MASTER | SPI_SS_DISABLED | SPI_L_TO_H | SPI_XMIT_H_TO_L | SPI_CLK_DIV_5); // (8MHz/2)/8= 500k
                ON(  CS_ADS  );
                delay_ms(100);
                OFF( CS_ADS  );

                  spi_write(SDATACT);  // SDATACT      0b00010001  // Stop Read Data Continuously mode

                  spi_write(WREG0);    // WREG0        0b01000000  // Write n nnnn registers starting at address r rrrr
                  spi_write(11);       //   11 =       0b00001011  // first byte 010r rrrr (2xh)(2) - second byte 000n nnnn(2)
/*ID        0x00*/spi_write(0b01110011);//|REV_ID4    | REV_ID3    |REV_ID2    |  N/A       |   N/A      |   N/A      | REV_ID1  | REV_ID0   |
/*CONFIG1   0x01*/spi_write(0b00000000);//|SINGL_SHOT|   0        |   0       |   0        |   0        |  OSR2      |  OSR1    |  OSR0     |  Continuous 250sps
/*CONFIG2   0x02*/spi_write(0b11100000);//|    1      |PDB_LOFFCMP|PDB_REFBUF| VREF_4V    | CLK_EN     |   0        |INT_TEST  | TEST_FREQ|  LeadOff - 2.4Vref
/*LOFF      0x03*/spi_write(0b00010000);//| COMP_TH2  | COMP_TH1   | COMP_TH0  |   1        |ILEAD_OFF1|ILEAD_OFF0|   0      | FLEAD_OFF|  LeadOff DC
/*CH1SET    0x04*/spi_write(configbyte);//|   PD1     | GAIN1_2    | GAIN1_1   |  GAIN1_0   |  MUX1_3    | MUX1_2     |  MUX1_1  | MUX1_0    |  Electrodes Gain=6
/*CH2SET    0x05*/spi_write(configbyte);//|   PD2     | GAIN2_2    | GAIN2_1   |  GAIN2_0   |  MUX2_3    | MUX2_2     |  MUX2_1  | MUX2_0    |  Electrodes Gain=1
/*RLD_SENS  0x06*/spi_write(0b00101100);//|  CHOP1    | CHOP0      | PD_RLD    |RLD_LOFFSNS| RLDN2      | RLDP2      |  RLDN1   | RLDP1     |  RLD on (1 = RLD buffer is enabled)
/*LOFF_SENS 0x07*/spi_write(0b00001111);//|    0      |     0      | FLIP2     |  FLIP1     | LOFFN2     | LOFFP2     |  LOFFN1  | LOFFP1    |
/*LOFF_STAT 0x08*/spi_write(0b00000000);//|    0      | MOD_FREQ   |   0       | RLD_STAT   | IN2N_OFF   | IN2P_OFF   | IN1N_OFF| IN1P_OFF  |
/*RESP1     0x09*/spi_write(0b11010010);//|RSP_DM_EN | RSP_MD_EN  |RESP_PH3   | RESP_PH2   | RESP_PH1   | RESP_PH0   |    1     | REP_CTL   |  Resp enabled
/*RESP2     0x0A*/spi_write(0b00000011);//|CALIB_ON  |    0       |   0       |    0       |    0       | RESP_FREQ|RLDREFINT|    1      |
/*GPIO      0x0B*/spi_write(0b00001100);//|    0      |    0       |   0       |    0       | GPIOC2     | GPIOC1     | GPIOD2   | GPIOD1    |
                                  delay_us(100);

                  spi_write(RREG0);    // RREG0        0b00100000  // Read n nnnn registers starting at address r rrrr
                  spi_write(11);       //   11 =       0b00001011  // first byte 010r rrrr (2xh)(2) - second byte 000n nnnn(2)
/*ID        0x00*/spi_write(0b01110011);//|REV_ID4    | REV_ID3    |REV_ID2    |  N/A       |   N/A      |   N/A      | REV_ID1 | REV_ID0   |
                         ret =(SPI1BUF==0b01110011)?(ret|0x0001):ret;
/*CONFIG1   0x01*/spi_write(0b00000000);//|SINGL_SHOT|   0        |   0       |   0        |   0        |  OSR2      |  OSR1    |  OSR0     |  Continuous 250sps
                         ret =(SPI1BUF==0b00000000)?(ret|0x0002):ret;
/*CONFIG2   0x02*/spi_write(0b11100000);//|    1      |PDB_LOFFCMP|PDB_REFBUF| VREF_4V    | CLK_EN     |   0        |INT_TEST  | TEST_FREQ|  LeadOff - 2.4Vref
                         ret =(SPI1BUF==0b11100000)?(ret|0x0004):ret;
/*LOFF      0x03*/spi_write(0b00010000);//| COMP_TH2  | COMP_TH1   | COMP_TH0  |   1        |ILEAD_OFF1|ILEAD_OFF0|   0      | FLEAD_OFF|  LeadOff DC
                         ret =(SPI1BUF==0b00010000)?(ret|0x0008):ret;
/*CH1SET    0x04*/spi_write(configbyte);//|   PD1     | GAIN1_2    | GAIN1_1   |  GAIN1_0   |  MUX1_3    | MUX1_2     |  MUX1_1  | MUX1_0    |  Electrodes Gain=6
                         ret =(SPI1BUF==configbyte)?(ret|0x0010):ret;
/*CH2SET    0x05*/spi_write(configbyte);//|   PD2     | GAIN2_2    | GAIN2_1   |  GAIN2_0   |  MUX2_3    | MUX2_2     |  MUX2_1  | MUX2_0    |  Electrodes Gain=1
                         ret =(SPI1BUF==configbyte)?(ret|0x0020):ret;
/*RLD_SENS  0x06*/spi_write(0b00101100);//|  CHOP1    | CHOP0      | PD_RLD    |RLD_LOFFSNS| RLDN2      | RLDP2      |  RLDN1   | RLDP1     |  RLD on (1 = RLD buffer is enabled)
                         ret =(SPI1BUF==0b00101100)?(ret|0x0040):ret;
/*LOFF_SENS 0x07*/spi_write(0b00001111);//|    0      |     0      | FLIP2     |  FLIP1     | LOFFN2     | LOFFP2     |  LOFFN1  | LOFFP1    |
                         ret =(SPI1BUF==0b00001111)?(ret|0x0080):ret;
/*LOFF_STAT 0x08*/spi_write(0b00000000);//|    0      | MOD_FREQ   |   0       | RLD_STAT   | IN2N_OFF   | IN2P_OFF   | IN1N_OFF| IN1P_OFF  |
                     ret =(SPI1BUF==0b00000000)?(ret|0x0100):ret;
/*RESP1     0x09*/spi_write(0b11010010);//|RSP_DM_EN | RSP_MD_EN  |RESP_PH3   | RESP_PH2   | RESP_PH1   | RESP_PH0   |    1     | REP_CTL   |  Resp enabled
                         ret =(SPI1BUF==0b11010010)?(ret|0x0200):ret;
/*RESP2     0x0A*/spi_write(0b00000011);//|CALIB_ON  |    0       |   0       |    0       |    0       | RESP_FREQ|RLDREFINT|    1      |
                         ret =(SPI1BUF==0b00000011)?(ret|0x0400):ret;
/*GPIO      0x0B*/spi_write(0b00001100);//|    0      |    0       |   0       |    0       | GPIOC2     | GPIOC1     | GPIOD2   | GPIOD1    |
                         ret =(SPI1BUF==0b00001100)?(ret|0x0800):ret;
                                  delay_us(100);

                  spi_write(OFFSETCAL); //OFFSETCAL    0b00011010

                                 output_high(CS_ADS);
                                 delay_ms(2);
                                 return ret;
```

```
}
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
void on_ADS(void)
{
    OFF( CS_ADS );                      delay_us(250);
    spi_write( RDATACT );       delay_us(250);
    ON(  CS_ADS );                      delay_us(250);
    OFF(  STRT  );              delay_us(250);
    ON(   STRT  );
    ext_int_edge(1,H_TO_L);
    clear_interrupt(INT_EXT1);
        enable_interrupts(INT_EXT1 ); // Interrupcion ADS1
}
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
extern int8 recibido;
extern unsigned int8 rsp3[16],rsp2[16],rsp1[16];
extern unsigned int8 ecg3[16],ecg2[16],ecg1[16];
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#int_EXT1
void  EXT1_isr(void)          // Interrupcion cada vez que DRDY pasa a 0: 125 SPS = 8 ms
{
        char b3,b2,b1;


                OFF( CS_ADS);             // LOFF_STAT: RLD_STAT IN2N_OFF IN2P_OFF IN1N_OFF IN1P_OFF

                         b3= spi_read(0); // 1100 LOFF_STAT[4:0] GPIO[1:0] 13 0's
                         b2= spi_read(0); //[1100 abcd eXX0 0000] 0000 0000
                         b1= spi_read(0); // Status

                         rsp3[recibido]= spi_read(0);   // Respiracion
                         rsp2[recibido]= spi_read(0);
                         rsp1[recibido]= spi_read(0);

                         ecg3[recibido]= spi_read(0);   // ECG
                         ecg2[recibido]= spi_read(0);
                         ecg1[recibido]= spi_read(0);

            delay_us(10);
                ON(  CS_ADS );
                recibido=(recibido<15)?(recibido+1):15;
}
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
#include "_main.h"
#include "ADS1292.h"
#include "SD_Card.h"
#include <stdlib.h>
#zero_ram
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int8 rsp3[16],rsp2[16],rsp1[16];
unsigned int8 ecg3[16],ecg2[16],ecg1[16];
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int8 recibido=0;
unsigned int8 procesado=0;
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int8 dt[512];            /// Buffer de escritura  en uSD

//variables SD CARD: inicio de la FAT1 y la FAT2 y de las entradas de ficheros
unsigned  int8 sect_x_cluster=64;
unsigned int32 sect_fat_1;
unsigned int32 sect_fat_2;


unsigned int32 sect_entry;
unsigned int16 cluster =     0x0005;
unsigned int32 size;
unsigned int32 sector_ini;
unsigned int32 sector;

unsigned int16 sect_fat;
unsigned int16 pos_sector=0;
unsigned int16 pos_fat=0;

unsigned int16 year = 2020;     //Fecha a poner en los ficheros creados
unsigned int8 month = 1;
unsigned int8 day   = 1;
unsigned int8 hour  = 0;
unsigned int8 minute= 0;
unsigned int8 second= 0;

  signed int16 linea_ini=4; //Linea (de 32 bytes) en el raiz -> ficheros
  signed int16 linea=4;
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
int ha_pasado_1_hora = 0;
int hora_adquisicion = 0;
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
void main(void)
{
    output_float(PULS); set_pullup(TRUE, PULS);

    output_float(RT);
        output_low(  GN);       delay_ms(100);

        if(input(RT)){  ON( LED1);      OFF(LED2);      while((ADS_init(Normal))==0);   }
        else             {      OFF(LED1);      ON( LED2);      while((ADS_init(Test  ))==0);   }

        output_float(GN); delay_ms(500);        OFF(LED1);      OFF(LED2); delay_ms(500);

        sd_virgen();

        delay_ms(1000);

        on_ADS();

        setup_rtc(RTC_ENABLE, 0x00);
        setup_etc_alarm(RTC_ALARM_ENABLE | RTC_CHIME_ENABLE,
                                        RTC_ALARM_HOUR
                                        , 0);
        enable_interrupts(INT_RTC);

        for(;;) // Bucle principal
```

```
#include "_main.h"
#include "ADS1292.h"
#include "SD_Card.h"
#include <stdlib.h>
#zero_ram
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int8 rsp3[16],rsp2[16],rsp1[16];
unsigned int8 ecg3[16],ecg2[16],ecg1[16];
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int8 recibido=0;
unsigned int8 procesado=0;
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int8 dt[512];           /// Buffer de escritura  en uSD

//variables SD CARD: inicio de la FAT1 y la FAT2 y de las entradas de ficheros
unsigned  int8 sect_x_cluster=64;
unsigned int32 sect_fat_1;
unsigned int32 sect_fat_2;


unsigned int32 sect_entry;
unsigned int16 cluster =     0x0005;
unsigned int32 size;
unsigned int32 sector_ini;
unsigned int32 sector;

unsigned int16 sect_fat;
unsigned int16 pos_sector=0;
unsigned int16 pos_fat=0;

unsigned int16 year = 2020;     //Fecha a poner en los ficheros creados
unsigned int8 month = 1;
unsigned int8 day   = 1;
unsigned int8 hour  = 0;
unsigned int8 minute= 0;
unsigned int8 second= 0;

  signed int16 linea_ini=4; //Linea (de 32 bytes) en el raiz -> ficheros
  signed int16 linea=4;
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
int ha_pasado_1_hora = 0;
int hora_adquisicion = 0;
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
void main(void)
{
    output_float(PULS); set_pullup(TRUE, PULS);

    output_float(RT);
        output_low(  GN);       delay_ms(100);

        if(input(RT)){  ON( LED1);      OFF(LED2);      while((ADS_init(Normal))==0);   }
        else            {      OFF(LED1);      ON( LED2);      while((ADS_init(Test  ))==0);   }

        output_float(GN); delay_ms(500);        OFF(LED1);      OFF(LED2); delay_ms(500);

        sd_virgen();

        delay_ms(1000);

        on_ADS();

        setup_rtc(RTC_ENABLE, 0x00);
        setup_etc_alarm(RTC_ALARM_ENABLE | RTC_CHIME_ENABLE,
                                        RTC_ALARM_HOUR
                                        , 0);
        enable_interrupts(INT_RTC);

        for(;;) // Bucle principal
```

```
{
    if(recibido==0)
    {
        OFF( LED2 );

    }
    else
    {
    ON(  LED2 );

            while(procesado<recibido)  //Podria ser que durante el proceso se reciban varios datos
            {
                if(pos_sector<511)
                {
                    dt[pos_sector++]=rsp3[procesado];
                    dt[pos_sector++]=rsp2[procesado];
                    dt[pos_sector++]=rsp1[procesado];
                    dt[pos_sector++]=0x01;
                    dt[pos_sector++]=ecg3[procesado];
                    dt[pos_sector++]=ecg2[procesado];
                    dt[pos_sector++]=ecg1[procesado];
                    dt[pos_sector++]=0x02;

                    if(pos_sector>510)  // He llenado un sector -> Guardar 64 x 8 = 512 (64 muestras por sector eq. a 512 ms)
                    {
                        pos_sector=0;
                            size+=512;

                            if(sector>=(sector_ini+sect_x_cluster))              // He llenado un cluster -> Actualizar el tamaÒo y la FAT y su copia
                            if(((sector-sector_ini)%sect_x_cluster)==0)          // Equivale a (sector_x_cluster x 512 ms) P.ej. cada 16.384 s
                            {
                                //Size -> Root
                            sd_read_block(sect_entry+(unsigned int32)((linea_ini + hora_adquisicion)/16), dt);
                                    linea = ((linea_ini + hora_adquisicion)%16);
                                    dt[32*linea +28]=make8(size,0);
                                    dt[32*linea +29]=make8(size,1);
                                    dt[32*linea +30]=make8(size,2);
                                    dt[32*linea +31]=make8(size,3);                 sd_write_block(sect_entry+(unsigned int32)((linea_ini + hora_adquisicion)/16), dt);

                                    //FAT

                                    sect_fat= sect_fat_1 + make8(cluster,1);    sd_read_block(sect_fat, dt);
                                    pos_FAT = make8(cluster,0)*2;
                                            dt[pos_FAT  ]=make8(cluster+1,0);
                                            dt[pos_FAT+1]=make8(cluster+1,1);

                                            if(pos_FAT<510)
                                            {
                                              dt[pos_FAT+2]=0xFF;
                                              dt[pos_FAT+3]=0xFF;
                                            }
                                    sect_fat= sect_fat_1 + make8(cluster,1);    sd_write_block(sect_fat, dt);       // FAT 1
                                    sect_fat= sect_fat_2 + make8(cluster,1);    sd_write_block(sect_fat, dt);       // FAT 2

                                    cluster++;

                                    if(pos_FAT==510)
                                    {
                                    memset(dt,0,512);
                                    dt[0]=0xFF;
                                    dt[1]=0xFF;
                                    sect_fat= sect_fat_1 + make8(cluster,1);    sd_write_block(sect_fat, dt);       // FAT 1
                                    sect_fat= sect_fat_2 + make8(cluster,1);    sd_write_block(sect_fat, dt);       // FAT 2
                                    }
```

```
cluster++;

if(pos_FAT==510)
{
 memset(dt,0,512);
 dt[0]=0xFF;
 dt[1]=0xFF;
 sect_fat= sect_fat_1 + make8(cluster,1);       sd_write_block(sect_fat, dt);        // FAT 1
 sect_fat= sect_fat_2 + make8(cluster,1);       sd_write_block(sect_fat, dt);        // FAT 2
}


if( ha_pasado_1_hora )  //Ha pasado 1 hora -> nuevo fichero (Root y FAT)
{
      ha_pasado_1_hora=0;
      cluster--;
           sect_fat=sect_fat_1+make8(cluster,1);   sd_read_block(sect_fat, dt);
              dt[make8(cluster,0)*2 ]=0xFF;          // Se termina el fichero y la cadena de clusters
              dt[make8(cluster,0)*2+1]=0xFF;         // Ya no apunta al siguiente cluster: FF FF

           sect_fat=sect_fat_1+make8(cluster,1);   sd_write_block(sect_fat, dt);
           sect_fat=sect_fat_2+make8(cluster,1);   sd_write_block(sect_fat, dt);
      cluster++;

      sd_read_block(sect_entry+(unsigned int32)((linea_ini + hora_adquisicion)/16), dt);
           linea = (linea_ini + hora_adquisicion)%16;

                dt[32*linea + 0]='D';   //Name
                dt[32*linea + 1]='0';
                dt[32*linea + 2]='';
                dt[32*linea + 3]='0' + hora_adquisicion/10; // Nuevo nombre fichero
                dt[32*linea + 4]='0' + hora_adquisicion%10; // (incremento hora)
                dt[32*linea + 5]='_';
                dt[32*linea + 6]='0';
                dt[32*linea + 7]='0';
                dt[32*linea + 8]='D';   //Ext
                dt[32*linea + 9]='A';
                dt[32*linea +10]='T';
                dt[32*linea +11]=0x20;  //File Attrib
           dt[32*linea +12]=0x00;       //Rsved
             dt[32*linea +13]=0x40;     //createMs
         year = 2020; month = 1; day    = 1;
     hour = 1 + hora_adquisicion;
        minute= 0; second= 0;
                dt[32*linea +14]=make8(getHour(),0); dt[32*linea +15]=make8(getHour(),1); //createHour
                dt[32*linea +16]=make8(getDate(),0); dt[32*linea +17]=make8(getDate(),1); //createDate
                dt[32*linea +18]=make8(getDate(),0); dt[32*linea +19]=make8(getDate(),1); //lastAccess
                dt[32*linea +22]=make8(getHour(),0); dt[32*linea +23]=make8(getHour(),1); //modifHour
                dt[32*linea +24]=make8(getDate(),0); dt[32*linea +25]=make8(getDate(),1); //modifDate

                dt[32*linea +26]=make8(cluster,0);   dt[32*linea +27]=make8(cluster,1); //Cluster

           size=0;   dt[32*linea +28]=make8(size,0);
                     dt[32*linea +29]=make8(size,1);
                     dt[32*linea +30]=make8(size,2);
                     dt[32*linea +31]=make8(size,3); //Size
((linea_ini + hora_adquisicion)/16), dt);                                  sd_write_block(sect_entry+(unsigned int32)
      } // ha_pasado_1_hora
    } // Se lleno un cluster
  memset(dt,0,512);
 } // Se lleno un sector
```

```
                        procesado=(procesado<15)?(procesado+1):15;

                  }//while procesado<recibido
                    recibido=0;
                    procesado=0;
          }//recibido>0

        }
}
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#int_RTC
void RTC_isr(void)   // cada 1 hora
{
        ha_pasado_1_hora=1;
        hora_adquisicion++;
}
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
#ifndef INC_SD_CARD

#define INC_SD_CARD 2
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//Commands
#define   CMD0                0x40    //go to idle
#define   CMD1                0x41    //initialization process
#define   CMD8                0x48    //verify interface
#define   CMD17               0x51    //read single block
#define   CMD24               0x58    //write single block
#define   CMD55               0x77    //escape for app specific command
#define   CMD58               0x7a    //read OCR
#define   ACMD41              0x69    //poll operation range
//Responses
#define   R1_READY_STATE      0x00    //sd ready
#define   R1_IDLE_STATE       0x01    //card in idle state
#define   R1_ILLEGAL_COMMAND  0x04    //illegal command
#define   DATA_START_BLOCK    0xFE    //start token for read or write
#define   DATA_RES_MASK       0x1F    //mask for data response
#define   DATA_RES_ACCEPTED   0x05    //write accepted
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#define MAX_ENTRIES_SECTOR_FAT16     256
#define ENDFILE16                  0xFFFF
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
///variables globales
extern unsigned int16 year  ;
extern unsigned int8 month  ;
extern unsigned int8 day    ;
extern unsigned int8 hour   ;
extern unsigned int8 minute ;
extern unsigned int8 second ;

///funciones
#define getHour() (int16)((int16)hour << 11) + (int16)((int16)minute << 5) + (int16) (second >> 1)                  //5 bits: horas, 6 bits: minutos, 5 bits: segundos/2
#define getDate() (((int16)year - 1980) << 9 ) + ((int16)month << 5) + (int16)day     //7 bits: aõo (desde 1980), 4 bits: mes, 5 bits: dia
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
inline unsigned int8 xfer_spi(char);
unsigned int8 Commnd(char, int32, char);
unsigned int8 sd_init();
unsigned int8 sd_write_block(unsigned int32, unsigned char*);
unsigned int8  sd_read_block(unsigned int32, unsigned char*);

unsigned int8 sd_estructura();
unsigned int8 sd_virgen();
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#endifº
```

```
#include "_main.h"
#include "SD_Card.h"
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
           unsigned int8 sdhc;
extern unsigned int8 dt[512];            /// Buffer de escritura  en uSD
           unsigned int32 LBA0;

//variables clave: inicio de la FAT1 y la FAT2 y de las entradas de ficheros
extern unsigned int16 sect_x_cluster;
extern unsigned int32 sect_fat_1;
extern unsigned int32 sect_fat_2;
           unsigned int16 RootEntCnt;
           unsigned int32 sect_ini_datos;
           unsigned int32 sect_x_fat;
extern unsigned int32 sect_entry;
           unsigned int16 cluster_ini;
extern unsigned int16 cluster;
extern unsigned int32 size;
extern unsigned int32 sector_ini;
extern unsigned int32 sector;

extern   signed int16 linea_ini;
extern   signed int16 linea;
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
inline unsigned int8 xfer_spi(char envio){
 unsigned int8 ret;
  ON( LED1);
  ret = spi_read2(envio);
  OFF(LED1);
  return ret;
}//fin xfer_spi
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int8 Commnd(char CMD, int32 SD_Adress, char CRC){
   unsigned int8 iC1;

   xfer_spi(0xFF);
   xfer_spi(CMD);
   xfer_spi(make8(SD_Adress, 3));
   xfer_spi(make8(SD_Adress, 2));
   xfer_spi(make8(SD_Adress, 1));
   xfer_spi(make8(SD_Adress, 0));
   xfer_spi(CRC);

   do{iC1 = xfer_spi(0xFF);
   }while(iC1 == 0xFF);
   return iC1;
}//fin Commnd
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int8 sd_init(){
 unsigned  int8 R[17]={0};
 unsigned  int8 versionSD= 1;
 unsigned  int8 crc;
 unsigned int16 iI;
 unsigned int32 arg=0;

 setup_spi2(SPI_MASTER|SPI_SS_DISABLED|SPI_L_TO_H|SPI_XMIT_L_TO_H|SPI_CLK_DIV_16);

 memset(dt,0,512);

 // CMD0 - GO_IDLE_STATE  (R1)  Card Reset
        do{     output_high(CS);        // tarjeta deshabilitada
                for (iI = 0; iI < 10; iI++)     xfer_spi(0xFF);
                output_low(CS);         // tarjeta habilitada
                R[0] = Commnd( CMD0 , 0x00000000 , 0x95); //go to idle
        }while( R[0] != R1_IDLE_STATE);

  // CMD8 - SEND_IF_COND (R7)   Send Interface Condition Command
```

```
  // Argument 00 - 00 - 01 (Voltage supplied = 2.7-3.6V) - AA (Check pattern)
  // Response: illegal command -> Version 1
  // Response: echo-back           -> Version 2
        R[0] = Commnd(CMD8, 0x000001AA, 0x87);
    if(R[0]& R1_ILLEGAL_COMMAND){                              versionSD = 1;
                      }else{  R[1] = xfer_spi(0xFF);
                                  R[2] = xfer_spi(0xFF);
                                  R[3] = xfer_spi(0xFF);
                                  R[4] = xfer_spi(0xFF);
                              if(R[4]!=0xAA){     output_high(CS); return 1; }            //error
                                                                              versionSD = 2;
                      }

  // ACMD41 - SD_SEND_OP_ COND (R1)     Initialization Command
  // If host supports SDHC or SDXC: HCS (Host Capacity Support) is set to 1 (arg= 0x40000000)
  // If host only support SDSC:     HCS (Host Capacity Support) is set to 0 (arg= 0)
   arg = (versionSD == 2) ? 0x40000000 : 0;
   crc = (versionSD == 2) ? 0x77 : 0xFF;

   do{            Commnd(CMD55,    0, 0x65);    // El proceso puede llegar a durar hasta 1 segundo
          R[0] = Commnd(ACMD41, arg, crc);             // Application-Specific Command ñ APP_CMD (CMD55)
                  delay_ms(4);
   }while(R[0] != R1_READY_STATE);

  if(versionSD == 2){

  // CMD58 - READ_OCR (R3)
  // Bit 30 - Card Capacity Status bit:         0 indicates that the card is SDSC.
  //                                                                  1 indicates that the card is SDHC or SDXC
        R[0] = Commnd(CMD58, 0x00000000, 0xFF);
    if(R[0] == R1_READY_STATE){ R[1] = xfer_spi(0xFF);
                                                              R[2] = xfer_spi(0xFF);
                                                              R[3] = xfer_spi(0xFF);
                                                              R[4] = xfer_spi(0xFF);
                                                              sdhc = ((R[1]& 0xC0) == 0xC0) ? 1 : 0;
                      }else{                          output_high(CS); return 1; }            //error
  // ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑ ∑

  }// versionSD == 2 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

  setup_spi2(SPI_MASTER|SPI_SS_DISABLED|SPI_L_TO_H|SPI_XMIT_L_TO_H|SPI_CLK_DIV_1);

  output_high(CS);                                                            // tarjeta deshabilitada
  return 0;
}//fin sd_init
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int8 sd_write_block(unsigned int32 address, unsigned char* ptr){
   unsigned int16 iW;

   if(sdhc==0){ address <<= 9; }    //memoria SC -> bytes              memoria HC -> sectores
   output_low(CS);

              iW = Commnd(CMD24, address, 0xFF);
    if( iW == R1_READY_STATE){
                                                                      xfer_spi(0xFF);
                                                                      xfer_spi(DATA_START_BLOCK);   // Data start token
                                              for(iW = 0; iW < 512; iW++) xfer_spi(ptr[iW]);
                                                                              xfer_spi(0xFF);   // Dummy bytes
                                                                              xfer_spi(0xFF);   // en lugar del CRC16

                      do{       iW = xfer_spi(0xFF);
                      }while( (iW & DATA_RES_MASK) != DATA_RES_ACCEPTED);

                      do{       iW = xfer_spi(0xFF); delay_us(100);
                      }while(   iW == 0 );

                                              xfer_spi(0xFF);
                                              xfer_spi(0xFF);
```

```
                                        output_high(CS);   return 0; //correcto
                                        }
                                        output_high(CS);   return 1; //error
}//fin sd_write_block
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int8 sd_read_block(unsigned int32 address, unsigned char* ptr){
   unsigned int16 jR;
   unsigned int8  iR;

    if(sdhc==0){ address <<= 9; }    //memoria SC -> bytes             memoria HC -> sectores
    output_low(CS);

       iR  = Commnd(CMD17, address, 0xFF);
    if(iR == R1_READY_STATE){
                                                        do{     iR  = xfer_spi(0xFF);
                                                        }while( iR != DATA_START_BLOCK);

                                                        if(iR == DATA_START_BLOCK){
                                                        for(jR = 0; jR < 512; jR++)
                                                                dt[jR] = xfer_spi(0xFF);
                                                        xfer_spi(0xFF);
                                                        xfer_spi(0xFF);
                                                        xfer_spi(0xFF);
                                                        }
                                                output_high(CS);      return 0; //correcto
                                                }
                                                output_high(CS);      return 1; //error
}//fin sd_read_block
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int8 sd_estructura()
{
        unsigned int32 sctrs;
        int16 RsrvdCnt;

        sd_read_block(0, dt);   //Lee el sector fisico 0 para obtener el LBA0
        if(dt[510]==0x55)
        if(dt[511]==0xAA){              // Efectivamente parece que estoy en sct 0
                                                // Tabla de particiones
                                                        LBA0 = make32(dt[457],dt[456],dt[455],dt[454]);
                                                        if(dt[0]==0xEB) // MBR
                                                        LBA0 = 0;       //Sector Logico 0
                                          }//55AA
        else return 1; // No ha encontrado el LBA0 -> Hay que formatear la tarjeta

        sd_read_block(LBA0, dt);
                sect_x_cluster = (unsigned int32)dt[13] & 0x00FF;
                RsrvdCnt   = make16(dt[15],dt[14]);
                RootEntCnt = make16(dt[18],dt[17]);
                sect_x_fat = make16(dt[23],dt[22]);

        /* LBA0 */                sctrs  = LBA0;                // Inicio de cuenta: Direccion del Logic Block cero
                                        sctrs += (unsigned int32) RsrvdCnt & 0xFFFF;

    sect_fat_1          = sctrs;                // A partir de ahÍ: FAT 1
                                        sctrs += (unsigned int32) sect_x_fat & 0xFFFF;

        sect_fat_2      = sctrs;                                // A partir de ahÍ: FAT 2 (copia de seguridad)
                                        sctrs += (unsigned int32) sect_x_fat & 0xFFFF;

        sect_entry      = sctrs ;                               // A partir de ahÍ empieza el raÍz (la lista de ficheros)
                                        sctrs += (unsigned int32)(RootEntCnt>>4)&0xFFFF;

        sect_ini_datos  = sctrs ;                               // A partir de ahÍ empieza el contenido de los ficheros

        cluster=cluster_ini=0x0005;

        sector=sector_ini=sect_ini_datos+(unsigned int32)((cluster_ini-2)*sect_x_cluster);
```

```
}//fin sd_estructura
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int8 sd_virgen()
{


        sd_init();
        sd_estructura();
        memset(dt,0,512);

        dt[0]=0xF8;dt[1]=0xFF;
        dt[2]=0xFF;dt[3]=0xFF;
        dt[4]=0xFF;dt[5]=0xFF; // Esto es nuevo
        dt[6]=0xFF;dt[7]=0xFF; // de Windows 10
        dt[8]=0xFF;dt[9]=0xFF;

        dt[10]=0xFF;dt[11]=0xFF; // Esto es de mi fichero

                                                                                        sd_write_block(sect_fat_1, dt); // FAT 1
                                                                                        sd_write_block(sect_fat_2, dt); // FAT 2
                                                                                        //- - - - -
                                                                                        sd_read_block(sect_entry, dt);  // Root
        memset(dt+128,0,512-128);
        //- - - - -
        if(dt[32*3]==0) linea_ini=3;
        else                    linea_ini=4;
        linea=linea_ini;
        dt[32*linea + 0]='D';   //Name
        dt[32*linea + 1]='0';
        dt[32*linea + 2]='_';
        dt[32*linea + 3]='0';
        dt[32*linea + 4]='0';
        dt[32*linea + 5]='_';
        dt[32*linea + 6]='0';
        dt[32*linea + 7]='0';
        dt[32*linea + 8]='D';   //Ext
        dt[32*linea + 9]='A';
        dt[32*linea +10]='T';
        dt[32*linea +11]=0x20;  //File Attrib
    dt[32*linea +12]=0x00;      //Rsved
    dt[32*linea +13]=0x40;      //createMs
        year = 2020; month = 1; day   = 1;
    hour  = 1; minute= 0; second= 0;
        dt[32*linea +14]=make8(getHour(),0);
        dt[32*linea +15]=make8(getHour(),1); //createHour
        dt[32*linea +16]=make8(getDate(),0);
        dt[32*linea +17]=make8(getDate(),1); //createDate
        dt[32*linea +18]=make8(getDate(),0);
        dt[32*linea +19]=make8(getDate(),1); //lastAccess
        dt[32*linea +22]=make8(getHour(),0);
        dt[32*linea +23]=make8(getHour(),1); //modifHour
        dt[32*linea +24]=make8(getDate(),0);
        dt[32*linea +25]=make8(getDate(),1); //modifDate
        cluster=cluster_ini=0x0005;
        sector=sector_ini=sect_ini_datos+(unsigned int32)((cluster_ini-2)*sect_x_cluster);
        dt[32*linea +26]=make8(cluster,0);
        dt[32*linea +27]=make8(cluster,1); //Cluster
        size=0;
        dt[32*linea +28]=make8(size,0);
        dt[32*linea +29]=make8(size,1);
        dt[32*linea +30]=make8(size,2);
        dt[32*linea +31]=make8(size,3); //Size

                                                                                        sd_write_block(sect_entry, dt);
        memset(dt,0,512);
        return 0;
}//fin sd_virgen
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```