UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DISEÑO Y DESARROLLO DE UN SISTEMA PARA EL REGISTRO DE SEÑALES BIOELÉCTRICAS EXTRACELULARES

AUTOR: Adrián Beneit Barajas

TUTOR: Francisco Javier Saiz Rodríguez

COTUTOR: Julio Gomis-Tena Dolz

Curso Académico: 2016-17

AGRADECIMIENTOS

A mis tutores, Francisco Javier Saiz Rodríguez y Julio Gomis-Tena Dolz, por toda la atención y ayuda prestada.

A mi familia y, en especial, a Irene por apoyarme tanto estos meses.

A mi amigo Paco, por todos sus consejos.

RESUMEN

En el siguiente proyecto se ha diseñado un sistema de adquisición de la señal biológica del electrocardiograma, para ser usado en el seguimiento continuo de pacientes que presentan episodios de fibrilación auricular. Se ha procurado que el sistema limite mínimamente al paciente en cuanto a movilidad. Por ello, el tamaño se ha reducido al máximo, obteniendo un dispositivo que un usuario podría llevar unido al cuerpo con poca molestia. El sistema se diseña para ser usado con un Smartphone como visualizador, por medio de una aplicación Android desarrollada en el proyecto. La comunicación entre el dispositivo registrador de la señal y el Smartphone se realiza por medio de conexión Bluetooth, administrada desde la misma aplicación. De esta manera, el paciente no está obligado a permanecer en un sitio concreto, sino que puede moverse libremente siempre y cuando lleve consigo el teléfono.

Palabras clave: electrocardiograma, corazón, bioelectrónica, Bluetooth, Smartphone, Android.

RESUM

En el següent projecte s'ha dissenyat un sistema d'adquisició del senyal biològic de l'electrocardiograma, per a ser usat en el seguiment continuat de pacients que presenten episodis de fibril·lació auricular. S'ha procurat que el sistema limite mínimament al pacient quant a mobilitat. Per això, la grandària s'ha reduït al màxim, obtenint un dispositiu que l'usuari podria portar unit al cos amb cap molèstia. El sistema es dissenya per a ser usat amb un Smartphone com a visualitzador, per mitjà d'una aplicació Android desenvolupada en el projecte. La comunicació entre el dispositiu registrador del senyal i el Smartphone es realitza per mitjà de connexió Bluetooth, administrada des de la mateixa aplicació. D'aquesta manera, el pacient no està obligat a permaneixer en un lloc concret, sinó que pot moure's lliurement sempre que porte amb ell el telèfon.

Paraules clau: electrocardiograma, cor, bioelectrònica, Bluetooth, Smartphone, Android.

ABSTRACT

In this project it has been designed a system for electrocardiographic signal acquisition, to be used for a continuous tracing of patients that present atrial fibrillation episodes. It has been attempted to minimize the impact on patient mobility. Hence the size has been reduced to the maximum, to obtain a dispositive easy to wear with little to no disturbance. The system is designed to be used in conjunction with a smartphone as a visualizer, through an android app. The communication between the signal recorder and the smartphone is via Bluetooth, administrated by the app. Consequently, the patient doesn't feel obliged to stay on the same place and they can move freely as long as they carry their phones with them.

Keywords: electrocardiogram, heart, bioelectronics, Bluetooth, Smartphone, Android.

ÍNDICE

DOCUMENTOS CONTENIDOS EN EL TFG

- Memoria
- Presupuesto
- Anexos
- Planos
- Pliego de condiciones

ÍNDICE DE LA MEMORIA

ÍNDICE DEL PRESUPUESTO

ÍNDICE DE ANEXOS

ÍNDICE DE PLANOS

ÍNDICE DE PLIEGO DE CONDICIONES

Memoria

CAPÍTULO 1. INTRODUCCIÓN

1.1. OBJETIVOS DEL TRABAJO

El objetivo del proyecto es el diseño e implementación de un sistema electrónico de adquisición de datos de tipo biológico; concretamente, la señal del electrocardiograma (ECG). El propósito es obtener un dispositivo de dimensiones muy reducidas, capaz de hacer un seguimiento profesional de manera autónoma a pacientes que sufren episodios de fibrilación auricular. Este seguimiento se realiza en cualquier lugar, fuera de centros de salud. Además, el sistema se diseña especialmente para poder ser utilizado por el usuario medio, que no tiene por qué tener medios ni conocimientos médicos o tecnológicos. Por ello, se utiliza un dispositivo móvil (Smartphone) para monitorizar la señal de manera continua.

1.2. MOTIVACIÓN Y JUSTIFICACIÓN

Según el informe publicado por el Instituto Nacional de Estadística sobre las causas de muerte en nuestro país, en 2014, la enfermedad cardiovascular sigue tratándose de la primera en la lista del total de fallecimientos, situándose por encima incluso de los tumores. Se da la misma situación para las estancias hospitalarias.

Defunciones según las principales causas de muerte. 2014

	Total	Mujeres %	Variación interanual % Total
Todas las causas	**395.830**	**49,1**	1,4
Enfermedades del sistema circulatorio	117.393	54,4	-0,1
Tumores	110.278	39,0	-0,7
Enfermedades del sistema respiratorio	43.841	43,2	3,0
Enfermedades del sistema nervioso y de los órganos de los sentidos	23.394	61,9	8,9
Enfermedades del sistema digestivo	19.385	48,0	0,0
Trastornos mentales y del comportamiento	18.706	66,1	10,2
Causas externas de mortalidad	14.903	37,0	1,5

Ilustración 1. Informe de causas de muerte, sacado del INE (referencia [1]).

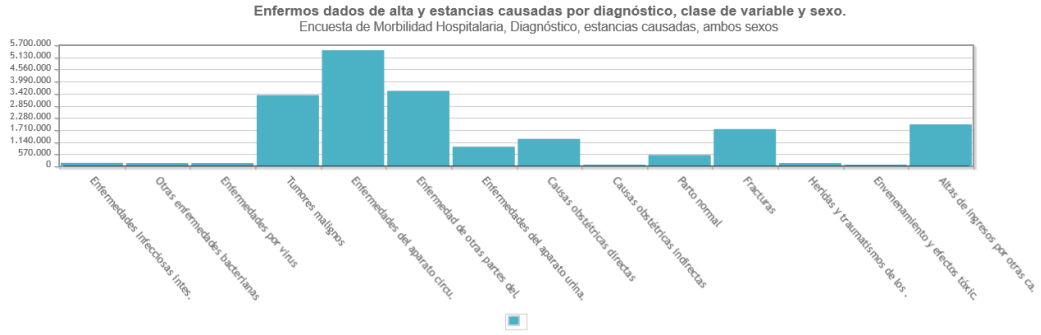


Ilustración 2. Informe de estancias hospitalarias, sacado del INE (referencia [2])

Las principales causas de muerte son las enfermedades cerebrovasculares y las enfermedades isquémicas del corazón, tanto para mujeres como para hombres, estando las mujeres un 6% por encima de ellos. A pesar de que en la última década se han ido mejorando las cifras, bajando en 2014 por primera vez del 30%, las enfermedades hipertensivas e isquémicas del corazón han crecido a lo largo de estos diez años.

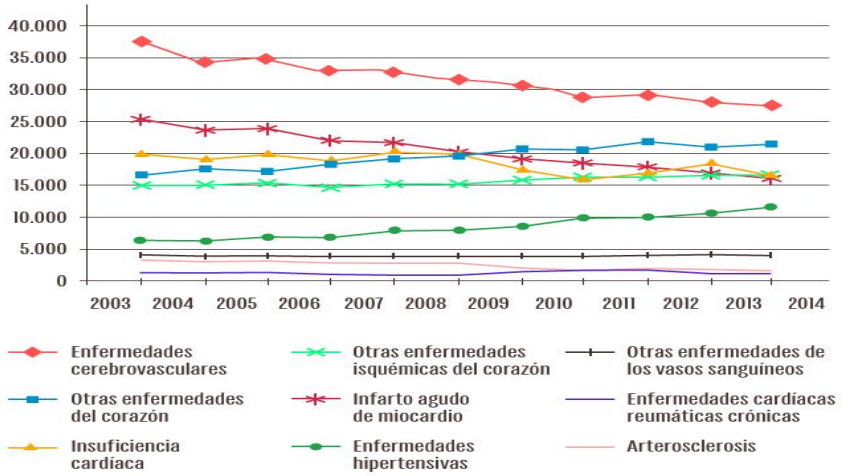


Ilustración 3. Gráfica que muestra las principales causas de mortalidad en España en la última década. Referencia [3].

Así, queda de manifiesto la importancia de análisis periódicos de la salud del corazón, especialmente en casos de riesgo o seguimiento de enfermedad cardíaca, y muy en particular de la fibrilación auricular: la arritmia más común y una de las principales cardiopatías. Con este proyecto se pretende acercar la posibilidad de realizar estos análisis para su detección, de forma económica pero fiable, sin interrupción de las actividades diarias.

Así, se ha considerado que utilizar un Smartphone para monitorizar la señal es la opción que más movilidad y autonomía permite. Estos teléfonos se han convertido en una herramienta de gran importancia para empresas y particulares. Con ello, como se ve en la siguiente figura, es posible llegar a un gran porcentaje de la población, ya que es la tecnología de mayor penetración en la sociedad capaz de realizar la monitorización.

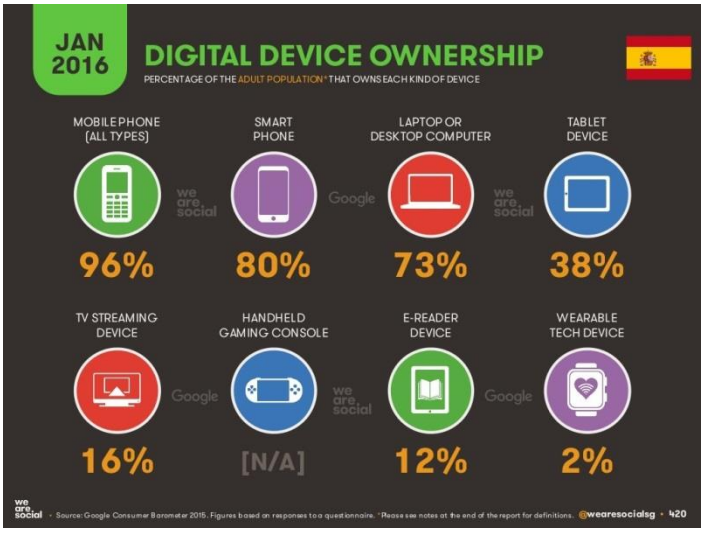


Ilustración 4. Porcentaje de usuarios de dispositivos digitales. Referencia [4].

El alcance de la aplicación es la visualización y almacenamiento del ECG. Sin embargo, la auténtica motivación del proyecto es seguir en desarrollo, implementando algoritmos de detección de una de las arritmias más comunes, la fibrilación auricular. De esta manera, se busca servir de base para otros proyectos de mayor envergadura que introduzcan un análisis médico completo de esta tipología de arritmia.

En el sentido académico, este proyecto forma parte de los créditos de formación obligatoria del Grado en Tecnologías Industriales de la Universidad Politécnica de Valencia. Consiste en 12 créditos (300 horas) necesarios para obtener el título ya nombrado. Desde el punto de vista académico, este trabajo es necesario para demostrar los conocimientos y competencias obtenidos a lo largo del grado, así como la capacidad de aplicarlos en un proyecto de ingeniería.

Concretamente, los conocimientos teóricos utilizados incluyen los aprendidos en asignaturas de tipo obligatorio: Proyectos, Sistemas Electrónicos, Tecnología Informática Industrial y Tecnología Electrónica; además de las asignaturas optativas: Desarrollo de Aplicaciones para Dispositivos Móviles y Programming Embedded Systems in C.

Por otro lado, además de aplicar lo aprendido, la realización de este trabajo ha permitido al alumno adquirir nuevos conocimientos: un método profesional para el diseño de circuitos impresos con la herramienta Eagle, manejo un sistema de comunicación inalámbrica (Bluetooth) y mejorar las habilidades preexistentes en cuanto al empleo de la electrónica y en cuanto a la programación (tanto a bajo como a alto nivel).

1.3. FUNDAMENTOS TEÓRICOS

La electrocardiografía es una técnica no invasiva mediante la cual se obtiene un registro de la actividad eléctrica del corazón, siendo uno de los métodos más usados para el estudio de este músculo. Mediante electrodos colocados estratégicamente, se recoge una señal eléctrica denominada electrocardiograma, producida por las variaciones de potencial eléctrico de las células cardíacas. La generación de corriente eléctrica en el corazón se consigue mediante el establecimiento de una

diferencia de potencial de membrana entre el interior y el exterior de las células cardiacas, cambiando la concentración iónica. Los iones sodio, potasio y cloruro son los principales en la generación del potencial de membrana, y su gradiente electroquímico es el que ayuda a determinar el voltaje de este potencial. La permeabilidad de la membrana a cada uno de los iones proporciona características eléctricas distintas en diferentes partes del corazón.

Las derivaciones son los terminales electrocardiográficos mediante los que se mide la diferencia de potencial. Nos permiten ver la misma onda desde distintas perspectivas, colocando los electrodos en distintas posiciones: entre brazos, entre brazo y pierna, etcétera. Para un electrocardiograma estándar existen hasta doce derivaciones. En el sistema de estudio, sólo se analiza una de las derivaciones precordiales, dado que nos queremos centrar en el comportamiento de la aurícula.

Según la respuesta al estímulo eléctrico, se pueden clasificar las células cardiacas en dos grupos: automáticas o de respuesta lenta, formado por tejido conductivo; y de respuesta rápida, formada por los miocitos.

El tejido conductivo recibe el nombre de automático no sin razón. Debido a su baja permeabilidad al ion K+, puede despolarizarse sin necesidad de estimulación externa del sistema nervioso. Este automatismo se da en particular en el nodo sinusal, "el marcapasos del corazón". Por otro lado, los miocitos son también llamados fibras musculares, y son conocidos por su contractilidad, capaces de reducir su longitud ante un estímulo externo. En este caso, es un impulso eléctrico. Dado que el ratio de tejido conductivo es muy pequeño en relación con los miocitos, las señales producidas por las células automáticas no son aparentes en el ECG.

A lo largo del capítulo se profundizará en el proceso electrofisiológico por el que se produce la señal ECG y su caracterización según sus tipos de ondas, segmentos y complejos, además de las derivaciones y los propios electrodos.

1.3.1. El potencial de acción

El potencial de acción es un cambio en el voltaje de las fibras cardíacas de forma transitoria. Dado un estímulo, la permeabilidad de la membrana cambia para los diferentes iones, pasando desde un potencial negativo a positivo en el interior de la membrana. Tras ello, le sigue un potencial de reposo, más lento.

Las membranas poseen unas compuertas biológicas iónicas, siendo los iones más importantes el sodio, el potasio y el calcio. La membrana cambia la permeabilidad de forma selectiva, de modo que el medio se irá polarizando o despolarizando en función de la concentración de dichos iones, y se genera un potencial eléctrico en el intercambio protónico. Los canales de apertura por los que pasan los distintos iones tienen unos tiempos de apertura y cierre distintos. La fuerza electromotriz disminuirá a medida que el potencial de membrana se acerque al potencial del ión (E_k,E_{Na}).

El proceso es el siguiente:

- Se parte de un potencial electroquímico inicial de membrana de -70mV. El estímulo externo debe superar un umbral de potencial (-55mV), y de otro modo no es posible la apertura de los canales iónicos del Na^+. Es decir, el potencial de acción tiene una respuesta todo o nada.
- Una vez superada esta despolarización inicial, se bombean iones Na^+ hacia el interior de la membrana, que genera una corriente rápida y un ascenso también acelerado del potencial de acción.
- En el instante en el que el potencial se torna positivo, ya no hay fuerza electromotriz que impulse al Na^+ al interior, pero los iones seguirán entrando. La permeabilidad bajará paulatinamente. Se denomina la repolarización inicial, y en esta etapa comienzan a abrirse algunos canales de iones K^+.
- A continuación, se da una corriente más lenta: la zona de meseta del potencial de acción, producida por la entrada de iones Ca^{2+} y una salida paulatinamente creciente de los iones K^+. La contracción del miocito se produce en esta etapa.
- Los canales de Ca^{2+} se van cerrando al bajar el potencial electroquímico y le sigue la repolarización rápida: es un proceso más lento que el de despolarización, por el cual se abren los canales de iones K^+ (lentos) y el potencial de acción cae. La fuerza electromotriz impulsará los iones al exterior de la célula, con el fin de volver a los -70mV iniciales y su permeabilidad de reposo.
- Se cierran los canales de Na^+ y se abren por completo los de K^+, dejando una alta permeabilidad a dicho ión.

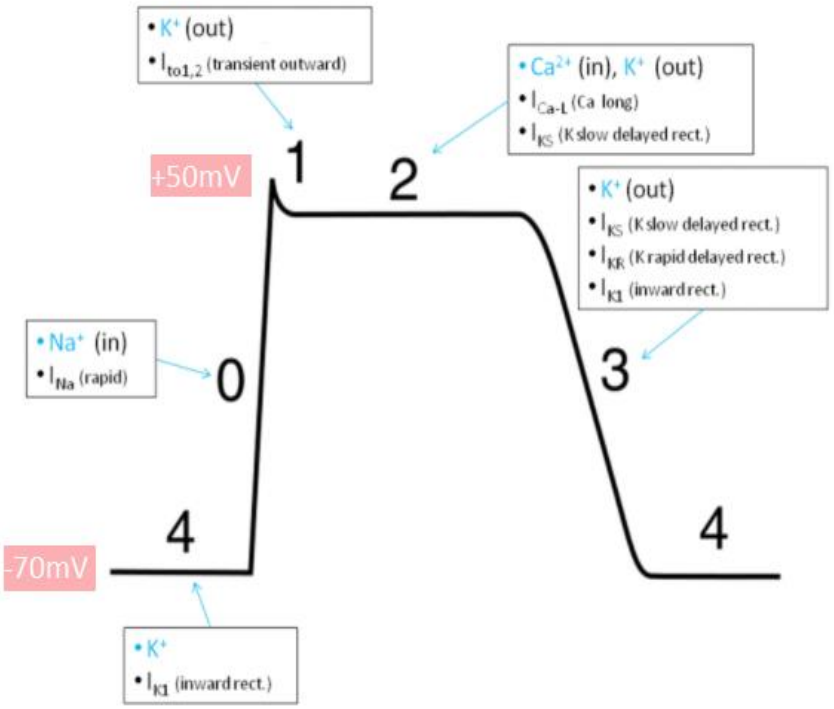


Ilustración 5. Gráfico de las distintas etapas del intercambio iónico de los miocitos

1.3.2. Estudio del electrocardiograma

El electrocardiograma comprende una serie de ondas, complejos y segmentos característicos producto del potencial de acción de los miocitos. Las ondas son las distintas curvas que presenta el trazado del ECG. Los complejos son un conjunto de ondas que representan un mismo proceso, y puede tomar distintas morfologías. Los segmentos son las líneas que unen ondas y ninguna de ellas puede estar incluida. Los intervalos son formados por segmento y onda.

Existen seis ondas por ciclo, nombradas de la P a la U, y los segmentos e intervalos se nombran en función entre qué ondas se encuentran o cuáles abarcan, respectivamente.

La onda P indica la despolarización de las aurículas. Suele ser positiva y su ausencia indica fibrilación auricular.

La onda Q puede aparecer o no en función de la derivación y la posición de los electrodos. Típicamente es estrecha y poco profunda, negativa. Le sigue la onda R, la imagen clásica de los electrocardiogramas, y la onda S, negativa. Todas ellas forman parte del complejo QRS, cambiante según la derivación. Este último representa la despolarización de los ventrículos y su duración no excede los 100 ms. Del mismo modo su voltaje debería ser inferior a los 3.5mV y suele estar del orden de 1mV. La dirección del estímulo eléctrico de este complejo se denomina eje cardíaco.

La onda T representa la repolarización de los ventrículos y es habitualmente positiva y asimétrica, mostrando un descenso más rápido. Las cardiopatías isquémicas alteran su morfología.

La onda U es típica de las derivaciones precordiales y es la última del ciclo, desconociéndose su origen. Es de escaso voltaje.

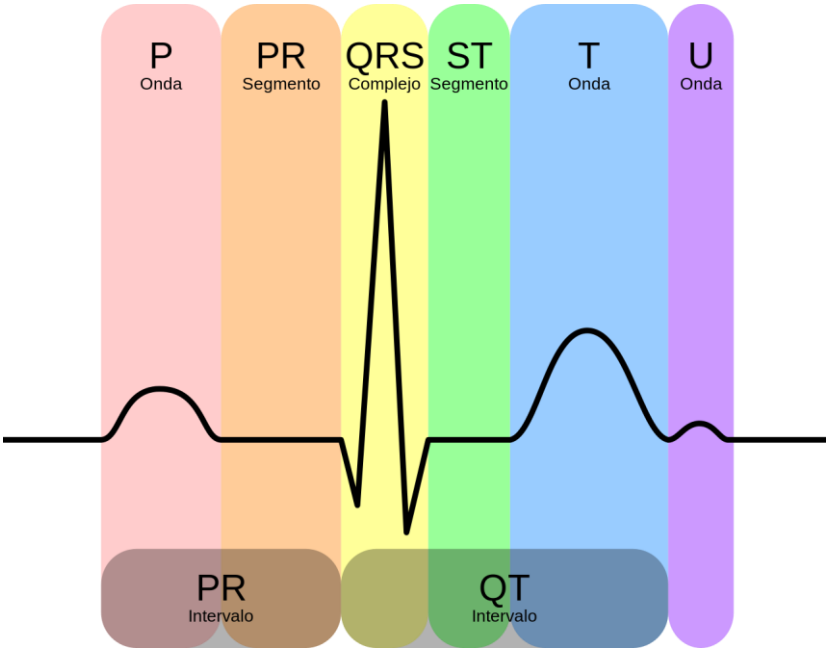


Ilustración 6. Esquema de las distintas ondas, complejos y segmentos del ECG

Desde el punto de vista matemático, la onda del electrocardiograma, igual que cualquier función periódica, se puede descomponer como sumatorio de senos de distinta frecuencia y amplitud (armónicos). La frecuencia fundamental de la onda ronda el valor de 1 Hz para un paciente en reposo. El contenido en armónicos a partir de los 60 Hz es despreciable.

1.3.3. Fibrilación auricular

La fibrilación auricular se trata de una dolencia que afecta al ritmo cardiaco. Como indica su nombre, en este tipo de arritmia la aurícula se contrae rápida e irregularmente, fuera de sincronismo como los ventrículos. La sangre no se bombea correctamente a estos últimos, aumentando el riesgo de infarto.

En un paciente sano, el impulso eléctrico del latido del corazón se origina en el nodo sinusal, viaja a través de la aurícula, donde se bombea la sangre a los ventrículos, y llega a los ventrículos para bombear la sangre al resto del cuerpo.

En un paciente con fibrilación auricular, la señal no se origina en el nodo sinusal, sino en múltiples partes de la aurícula. Las señales, consecuentemente, se transmiten de forma desordenada, rápida, provocando la fibrilación. Los ventrículos y la aurícula no trabajan de forma coordinada, y provoca que el volumen de sangre que se bombee por latido varíe, al basarse en el extraño patrón de latidos.

El diagnóstico de fibrilación auricular puede realizarse mediante el estudio del ECG, observando en él:

- Irregularidades en el complejo QRS.
- Ausencia de ondas P, sustituidas por ondas F múltiples e irregulares.

La motivación principal del proyecto es la elaboración de un equipo de detección de esta enfermedad cardiaca. Sin embargo, el estudio del ECG y, por tanto, de la detección per se de la fibrilación auricular, no se abordan en el trabajo y están fuera de los límites del mismo, relegándose a futuras ampliaciones.

1.3.4. Derivaciones

Las derivaciones son las distintas perspectivas que se captan de la onda del electrocardiograma, en función de la posición de los electrodos. Dependiendo del plano eléctrico, frontal u horizontal, se habla de derivaciones periféricas o precordiales, respectivamente. Es importante tener en cuenta que para una correcta interpretación del ECG deben analizarse las derivaciones conjuntamente.

Las derivaciones periféricas se dividen en bipolares y monopolares. Una derivación bipolar es aquella en la que se mide el potencial entre dos extremidades (tiene un polo positivo y otro negativo), mientras que en la monopolar se mide entre un punto teóricamente muy alejado y una extremidad (se anula el polo negativo).

Las clásicas se numeran del I al III y son de tipo bipolar:

- Derivación I: entre brazo derecho e izquierdo.
- Derivación II: entre brazo derecho y pierna izquierda.
- Derivación III: entre brazo izquierdo y pierna izquierda.

Estas derivaciones guardan una relación matemática, la Ley de Einthoven, por la cual las derivaciones se distribuyen siguiendo una forma triangular: DII=DI+DIII.

Existen otras derivaciones, esta vez monopolares, denominadas aumentadas. En combinación con las anteriores, se forma el sistema de referencia hexaxial de Bailey, útil para calcular el eje eléctrico del

plano frontal. Se denominan aV+Extremidad en la que se conecta el electrodo positivo, situando el negativo en el baricentro físico del triángulo de Einthoven: el punto de Wilson. Así, hablamos de avL (left, brazo izquierdo), avR (right, brazo derecho) y avF (foot, pie).

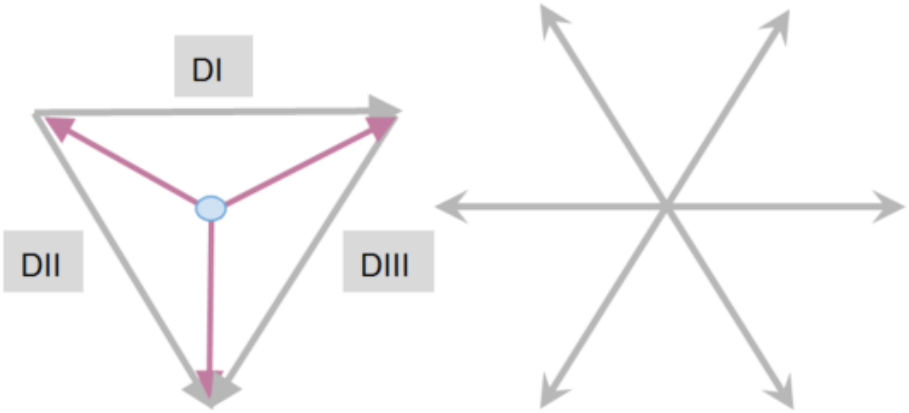


Ilustración 7. Triángulo de Einthoven (izda.) y sistema heaxial de Bailey (dcha.)

Las derivaciones precordiales son de tipo unipolar, es decir, un único electrodo se conecta en la zona torácica, mientras que el segundo se coloca en un punto muy alejado, simulando el infinito. Es el conocido punto Wilson. Al final, se consigue un registro muy similar al de las conexiones de tipo monopolar. Son nombradas del 1 al 6, precedidas por la letra V.

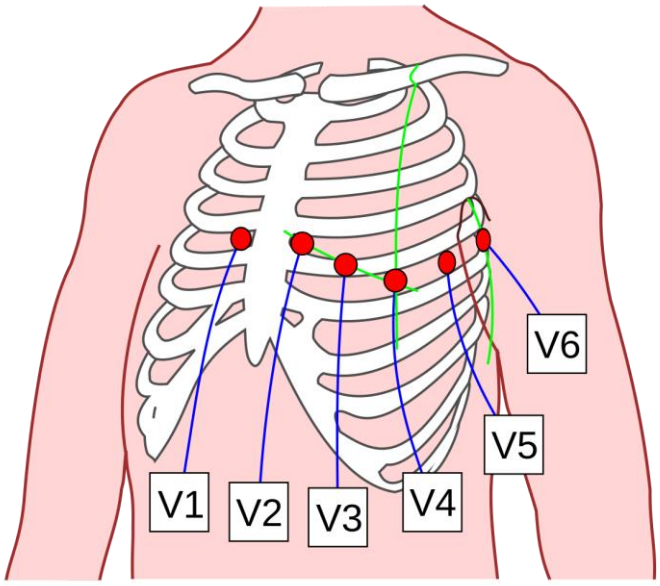


Ilustración 8. Posición de los electrodos para las distintas derivaciones precordiales.

Para un estudio completo del ECG es necesario analizar en conjunto todas las derivaciones. Sin embargo, la motivación del trabajo no es analizar por completo la onda, sino centrarse en la actividad de la aurícula. Para ello, se pretende medir únicamente entre dos puntos, ambos muy cercanos y posicionados sobre la zona de la aurícula.

1.3.5. Electrodos

Para medir las señales bioeléctricas del cuerpo, es necesario disponer de una interfaz piel-circuitos: los electrodos. Los electrodos del estudio entran en contacto directo con la piel, con o sin gel conductor (electrólito) y hacen una función equivalente a un transductor.

La transferencia de carga que se produce en la interfase electrodo-electrolito se realiza a través de reacción ox-red, o farádicas; y también mediante procesos no farádicos o sin transferencia de carga, pero con cambio de potencial. Este último comportamiento se asemeja al de un condensador, mientras que el primero al de una resistencia.

El comportamiento de la interfaz electrodo-electrolito no tiene por qué ser lineal, de modo que es necesario un modelo que se adapte a estas particularidades. Para una onda senoidal, el comportamiento puede resolverse como una resistencia y condensador posicionados en paralelo, simulando la bioimpedancia correspondiente a la interfaz electrodo-electrolito; con una resistencia adicional asociada al electrolito, en serie.

En este circuito aparece también una fuente de potencial. Es el denominado potencial de equilibrio o de media celda, y dependiendo del material utilizado se obtendrá un voltaje u otro. Este se mide respecto a un electrodo de referencia, habitualmente hidrógeno, al que se le asocia un voltaje 0 mV. Este voltaje está vinculado al contacto entre electrodo y electrólito y existirá circule o no corriente. Este potencial es producido por la distribución de cargas de la interfase que se produce al entrar en contacto los dos elementos.

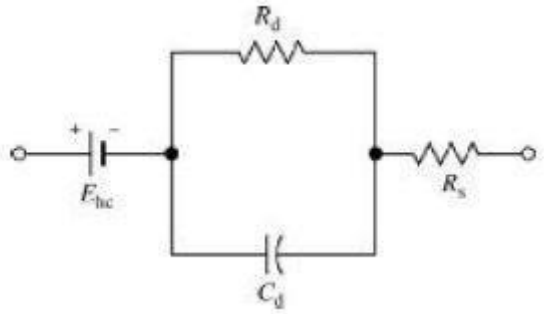


Ilustración 9. Circuito equivalente: modelización de la interfaz electrodo-electrolito.

Sin embargo, al añadir la piel en juego, debe considerarse un circuito adicional que represente esta nueva parte de la interfaz. Es un circuito análogo al de la interfaz electrolito-electrodo, y se conecta en serie a este primero.

Al final, al conectar un electrodo al paciente se obtiene un potencial o voltaje de interfase, tanto si es atravesado por corriente como si no. Este voltaje, si circula corriente, variará según las condiciones del contacto y de la piel en sí (variación de la impedancia). Por ello, es un valor difícilmente predecible con precisión y rara vez coincidirá exactamente con el potencial de interfase de otro electrodo.

Así, al muestrear la señal electrocardiográfica obtendremos el siguiente resultado:

- La onda ECG, con todos sus componentes en frecuencia, centrada en el eje eléctrico.

- Una componente de continua, que es el resultado de la diferencia de voltaje entre los distintos potenciales de interfase de los electrodos.

Los electrodos de este proyecto están dispuestos de un modo muy particular. Para un correcto estudio del ECG es necesaria la lectura de las doce derivaciones principales. Pero dada que la motivación del trabajo es la detección de una posible fibrilación auricular, se debe posicionar la instrumentación de tal modo que se centre en la actividad de la aurícula. Es por ello que sólo se utilizará una única derivación para el estudio completo, colocando los electrodos muy próximos entre sí: uno, en la derivación; el otro, en el centro del pecho.

1.3.6. Interferencias y tensión de modo común

Hay que tener en cuenta que, en todas las operaciones de medición, se pueden dar interferencias de muchos tipos. En el caso de medida de biopotenciales, se tiene que lidiar con el resto señales eléctricas del cuerpo. Sin embargo, el factor más influyente es el campo eléctrico de las líneas de potencia de alrededor y las capacidades parásitas generadas por ellos. Estos inducen una tensión de modo común de frecuencia 50 Hz que dificulta las mediciones. Además, estos componentes eléctricos, pueden generar inducción magnética en los circuitos.

1.3.7. Conclusiones de Índole técnica

La señal de electrocardiograma está compuesta por varias ondas y complejos. La frecuencia fundamental de la señal es de 1 Hz y a partir de 60, su contenido en armónicos es despreciable. Así, las posibles frecuencias de muestreo empiezan a ser viables a partir de 120 Hz.

La amplitud de los complejos ronda el valor de 1mV. Sin embargo, la onda va “montada” sobre una componente de continua de valor desconocido, generada por la diferencia entre los potenciales de interfase de los electrodos utilizados. Esta componente es, por regla general, muy superior a 1 mV. Por ello, en los aparatos clásicos de captación de electrocardiograma, se utiliza una etapa de filtrado analógico previa a la conversión a digital de los datos. Como se explica en el apartado 2.2.2.1., el chip utilizado para la conversión tiene una resolución tal que no es necesaria esta primera etapa de filtrado.

Existen interferencias en la señal a medir, esencialmente producidas por la tensión de modo común inducida por las líneas eléctricas de potencia que nos rodean. Para lidiar con estos problemas se ha implementado una funcionalidad llamada RLD (Right Leg Derivation) que, como se explica en el apartado 2.2.2.1., permite minimizar la tensión de modo común en los electrodos de medida.

CAPÍTULO 2. METODOLOGÍA DE DISEÑO

2.1. DESCRIPCIÓN GENERAL DE FUNCIONAMIENTO

Antes de entrar en detalle con el desarrollo del proyecto, es conveniente describir el sistema a diseñar en líneas generales. Con ello, se tendrá una vista global del trabajo a partir de la cual se explicarán las partes que lo componen.

Como ya se ha comentado en el objetivo, el fin del proyecto es obtener un dispositivo que recoja de manera continua la señal de ECG de una persona y la transmita en tiempo real a un teléfono móvil a través de Bluetooth. El receptor permite su visualización y almacena los datos, permitiendo, en un desarrollo posterior al presente proyecto, incorporar algoritmos de análisis de la onda.

La funcionalidad que se acaba de describir, a grandes rasgos, queda plasmada en el siguiente diagrama:

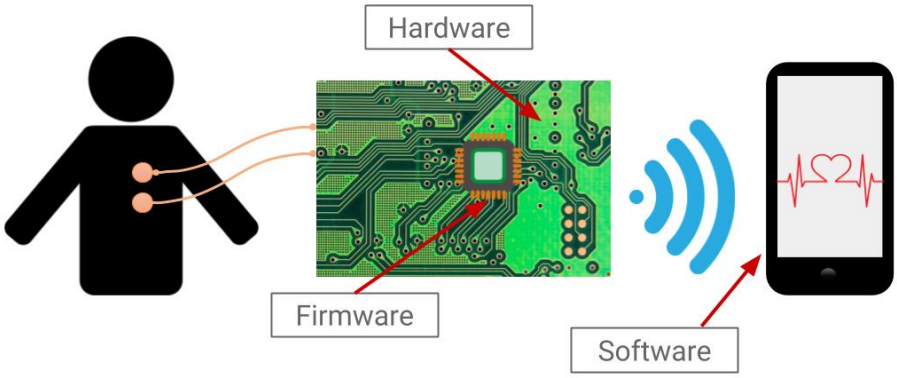


Ilustración 10. Esquema general del proyecto

De esta forma, el proyecto se descompone en 3 partes diferenciadas e independientes.

Por un lado, se encuentra el desarrollo del dispositivo electrónico que, unido al paciente mediante electrodos, capta la señal eléctrica del corazón y la envía vía Bluetooth. Este dispositivo se construye a partir de circuitos integrados comerciales (cuya funcionalidad se describe más adelante) agrupados en una única placa de circuito impreso. La explicación detallada de esta sección se encuentra en el apartado 2.2. (Hardware).

Por otro lado, se tiene el firmware o lógica de bajo nivel. El firmware es un programa informático que dirige el comportamiento del dispositivo descrito en el párrafo anterior. Se trata, pues, del código implementado en el microcontrolador, que se explica en el apartado 2.3. (Firmware Microcontrolador).

Por último, queda por estudiar el receptor de la señal del ECG, al cual se envían los datos de manera inalámbrica. Como se ha comentado previamente, este receptor será un Smartphone con sistema operativo Android, que se encarga de representar gráficamente la señal del ECG y almacenarla. Así, en el apartado 2.4. (Software dispositivo móvil) se expone la funcionalidad de la aplicación. En el futuro, esta aplicación será la encargada de analizar la señal y detectar episodios de fibrilación auricular.

La estrategia para abordar la explicación de cada apartado es partir de una visión general y profundizar cada vez más en detalle. De este modo, se conseguirá una mejor comprensión del conjunto.

2.2. HARDWARE

2.2.1. Descripción General del Hardware

Este apartado está dedicado a la parte electrónica del proyecto, esto es, al desarrollo de un circuito impreso al que se soldarán los diferentes componentes electrónicos comerciales.

Básicamente, la función del dispositivo es captar una señal eléctrica y transmitirla inalámbricamente. Centrando la atención en el flujo de información, el dispositivo debe partir de una etapa que capte la señal del cuerpo humano (electrodos) y una etapa que envíe los datos (módulo Bluetooth).

Ilustración 11. Primera aproximación al flujo de datos del Hardware

El proceso es, sin embargo, más complicado. La señal captada por los electrodos es un potencial presente en el cuerpo humano y, como ocurre con todos los procesos eléctricos de la naturaleza, se trata de una señal analógica. En cambio, los módulos Bluetooth están diseñados para transmitir información agrupada en bytes, es decir, señales digitales.

Por tanto, se requiere la presencia de una etapa que amplifique y convierta señales analógicas en digitales. Tal y como se explica en el próximo punto, esta función la llevará a cabo un circuito integrado comercial de nombre: ADS1292R.

Asimismo, es necesario un elemento que controle el funcionamiento del aparato y administre la conexión entre el dispositivo analógico-digital (de ahora en adelante, indistintamente ADS) y el módulo Bluetooth, dado que cada uno utiliza un protocolo de comunicación diferente. Este elemento, que actúa de “cerebro” y de intermediario en el flujo de datos, es un microcontrolador (de ahora en adelante también PIC) de nombre: PIC24F04KA201; este se describe, junto al resto de componentes, en el apartado siguiente.

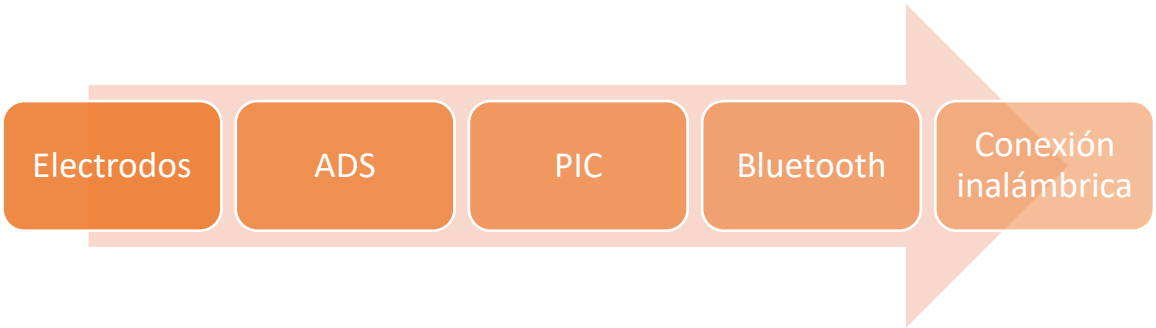


Ilustración 12.Esquema del flujo de datos en el Hardware.

Hasta aquí se ha descrito la estructura que tendría el dispositivo para cumplir su función principal, la captación y envío de una señal bioeléctrica. Así, es el momento de tratar una cuestión fundamental en cualquier producto de ingeniería: la fuente de energía.

Tal y como se ha manifestado en puntos anteriores, uno de los objetivos es que se permita la movilidad del paciente durante su uso. Así, el sistema funciona con un Smartphone como receptor, de manera que el usuario puede desplazarse libremente (siempre que lleve consigo su teléfono móvil). Como la alimentación del dispositivo tampoco debe limitar la movilidad, se incluye una batería para dotarle de autonomía. La batería escogida se describe en el punto siguiente de la memoria.

La adición de la batería implica la incorporación de otros componentes que se nombran a continuación.

En primer lugar, se necesitará un "Charge Management Controller", encargado de administrar el proceso de carga de la batería. Se trata de un circuito integrado de nombre: MCP73833. Este componente únicamente regula la tensión e intensidad en el proceso de carga de la batería. Sin embargo, este proceso no es el único que debe ser controlado.

La tensión y corriente de alimentación que proporciona la batería varía con el grado de carga que tenga. Es decir, que con la batería cargada al 100% proporcionará un voltaje diferente a cuando está cargada, por ejemplo, al 30%, lo cual podría ser perjudicial para el funcionamiento de algunos componentes electrónicos. Para evitar este problema se utilizan los reguladores de voltaje.

Por otro lado, la energía entrante para cargar la batería debe acceder al administrador de carga de algún modo. Para tal fin, se añade a la placa un conector USB hembra al que se puede conectar un cargador. El conector, además, tiene otra función de la que no se había hablado hasta el momento: la programación del microcontrolador.

Por último, se requiere el uso de un interruptor para evitar un consumo innecesario de batería cuando el usuario no utilice el aparato.

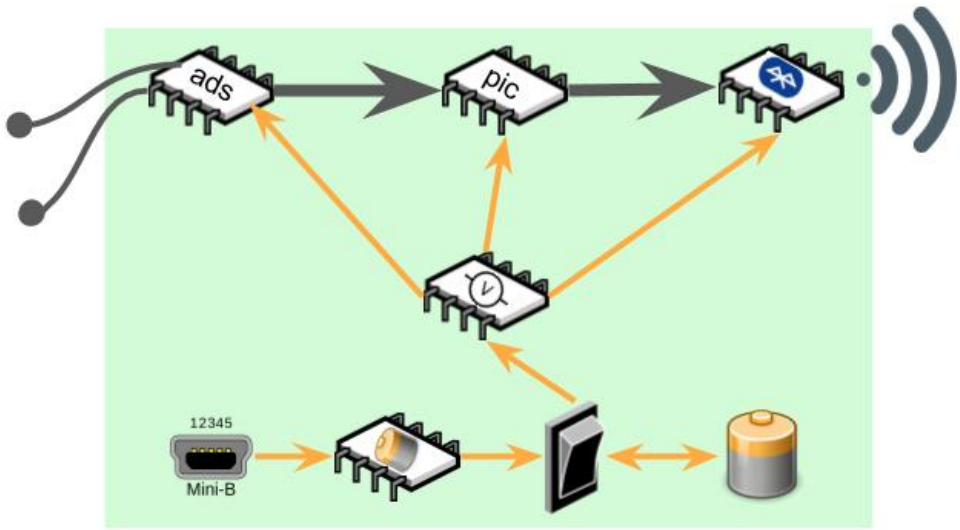


Ilustración 13. Esquema de la placa a realizar.

2.2.2. Descripción de los componentes.

En la descripción general del Hardware, se han introducido los componentes necesarios para el desarrollo de la electrónica del proyecto. A continuación, se procede a realizar una breve descripción de los mismos que permita una mejor comprensión de los posteriores apartados. Para más información de cada componente, se remite a las hojas de datos de cada uno. Referencias [10]-[22].

Chip ADS1292R (etapa adquisición y conversión Analógico-Digital)

Para adquirir la señal de electrocardiograma a formato digital se usa el componente ADS1292R de la compañía Texas Instruments, definido según los fabricantes como “Low-Power, 2-Channel, 24-Bit Analog Front-End for Biopotential Measurements”.

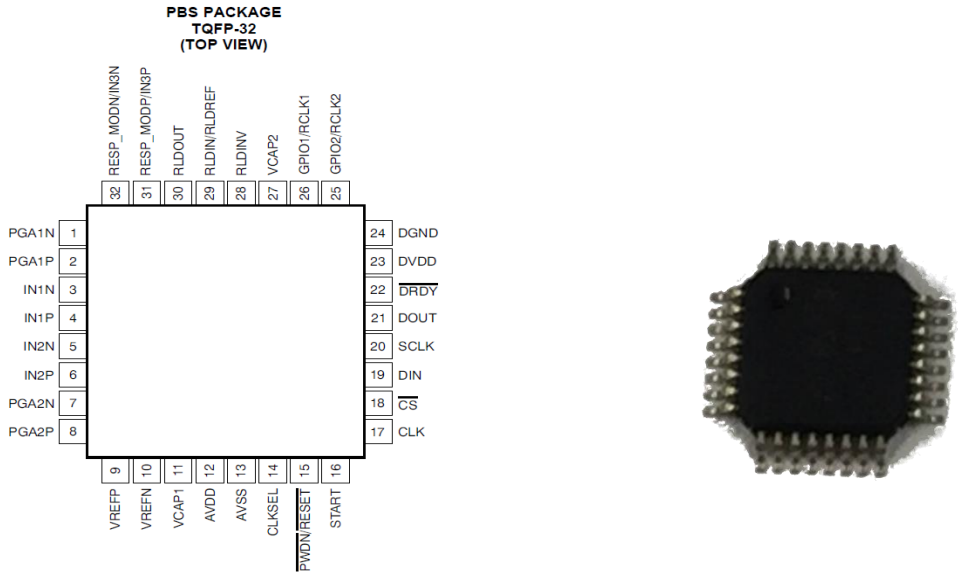


Ilustración 14. Diagrama de pines del ADS y foto real del componente.

Se trata de un dispositivo capaz de realizar muestreos simultáneos de varios canales de entrada, con amplificador de ganancia programable. Tiene posibilidad de utilizar referencia interna o externa, así como oscilador interno o externo. Dispone de un multiplexor que permite elegir el canal y otras señales internas de pruebas, lead-off, etc. Para comunicarse con otros componentes, el ADS1292 utiliza SPI (Serial Peripheral Interface).

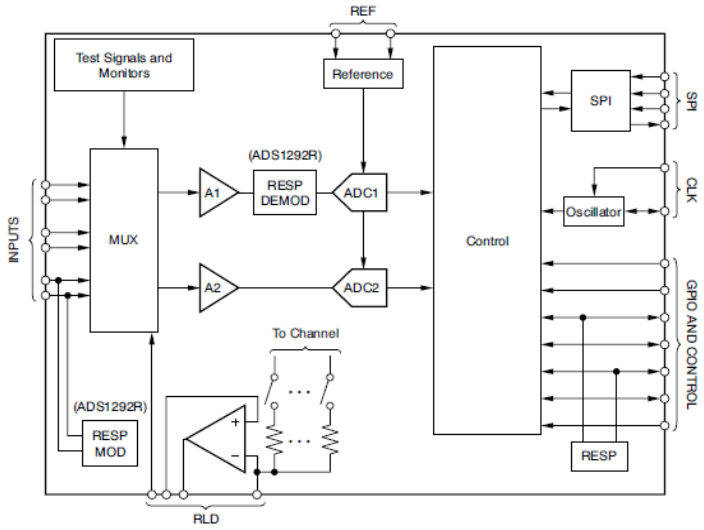


Ilustración 15. Diagrama de bloques del componente.

Este chip tiene dos posibles modos de funcionamiento*: continuous* y *single-shot*. En el primero, se convierten datos de manera ininterrumpida (a una frecuencia de muestreo determinada); mientras que, en el segundo, se debe dar una instrucción cada vez que se desee convertir un nuevo dato.

Está diseñado para adquisición de señales con alta precisión, debido a que la conversión se realiza a 24 bits codificados en complemento de a 2 (más de 16 millones de valores posibles). Con la referencia utilizada para la conversión (+- 2,4 V) se consigue una resolución del orden de 0,3 microvoltios. Con esta referencia (apartado 1.3.6), la conversión no saturará mientras la componente en continua no supere el valor de 2,4 V. Esta característica es la que lo hace idóneo para el proyecto.

Tal y como se detalló en los puntos de introducción (apartado 1.3.5), la amplitud de las ondas de ECG que se analizan son de un orden de magnitud inferior a la componente de continua generada por el potencial de la interfase piel-electrodos. Es por ello que en los aparatos clásicos de visualización de ECG es necesaria una etapa de filtrado analógico antes de la conversión a digital. Esta etapa requiere un aumento importante de espacio que haría inviable la portabilidad deseada en el proyecto. Sin embargo, con la cantidad de bits de que dispone el ADS utilizado, es posible realizar la conversión con la precisión necesaria y sin el filtrado analógico.

Además, el dispositivo incorpora funciones que lo adecúan a su uso para instrumentación médica y, en particular, lo hacen especialmente indicado para aplicaciones de baja potencia de adquisición de ECG. De las existentes, en este proyecto se hace uso de dos de ellas:

RLD

La primera y más importante es la funcionalidad RLD (Right leg Drive). Se utiliza circuitería interna para extraer la tensión de modo común de las entradas seleccionadas. Esta señal de modo común es producida por las líneas de potencia, las luminarias, etc. Esta señal extraída se puede medir, se puede utilizar como input y, como se hace en este proyecto, se puede inyectar en el cuerpo a través de un

tercer electrodo mediante una realimentación negativa. Con ello, la interferencia de modo común se ve restringida a un rango muy pequeño, lo que permite la captación adecuada de la señal de ECG.

LEAD OFF

Esta función, básicamente, permite saber si los electrodos se han soltado o no. Con esta función, entre muestreo y muestreo, inyecta una pequeña corriente por los electrodos para calcular la impedancia del contacto de los electrodos. La señal de excitación puede ser en continua o en alterna, según se configure. El estado de conexión de los electrodos se almacena en el registro LOFF_STAT. Además, la información de este registro se envía por SPI en los primeros 24 datos bits de la trama (apartado 2.3.2.).

La funcionalidad Lead Off es de especial importancia dada la motivación del proyecto. El sistema está pensado para que ser usado por el usuario de manera autónoma. Por ello, no existe ningún tipo de control por parte de personal médico sobre la colocación de los electrodos. Así, la información relativa a la conexión de los electrodos es de gran utilidad.

Para más información del componente, mirar la hoja de datos [10].

Microcontrolador

El microcontrolador seleccionado para el proyecto es el PIC24F04KA200 de la compañía Microchip. De esta familia de microcontroladores de propósito general (*General Purpose Microcontrollers*) se puede escoger entre 14 o 20 pines. Con el objetivo de optimizar el espacio, se ha utilizado el microcontrolador de 14. Como se verá en apartados posteriores, ninguno de los 14 pines se queda sin conectar.

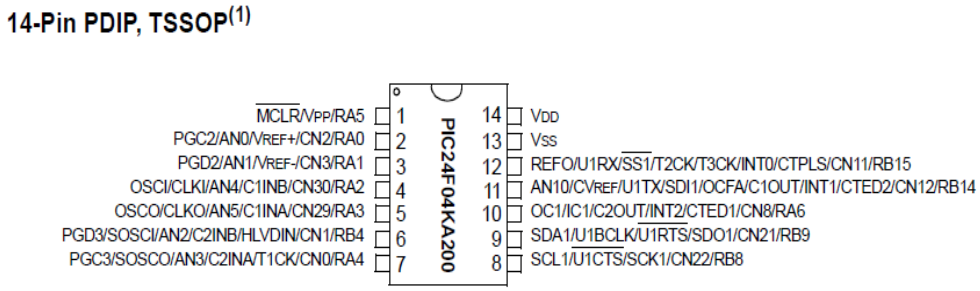


Ilustración 16. Diagrama de pines del PIC

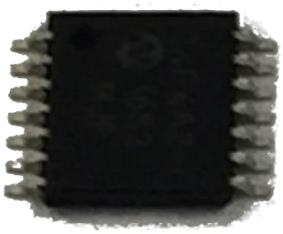


Ilustración 17. Imagen del componente real.

Como se muestra en el diagrama de pines, tener toda la funcionalidad de un microcontrolador de propósito general con tan solo 14 pines implica que muchas de las funciones coincidan en un mismo pin. Por tanto, no se podrá hacer uso de todas las ellas en la aplicación ya que, por regla general, cada pin solo puede realizar una de las funciones de las que dispone. Más adelante se verá que es preciso resolver por software esta limitación.

Para más información del componente, mirar las hojas de datos [13]-[18].

Modulo Bluetooth

El módulo Bluetooth utilizado es el Bluetooth RN42 de la compañía Microchip (hojas de datos[11]-[12]). Se trata de un módulo de pequeño tamaño y pequeña potencia. Soporta varias interfaces de comunicación, aunque está diseñado para usarse mediante el protocolo UART, reservándose el SPI para la programación, el testeo y el diagnóstico de fábrica. En el capítulo destinado al programa firmware se describen con mayor detalle los protocolos de comunicación (UART y SPI).

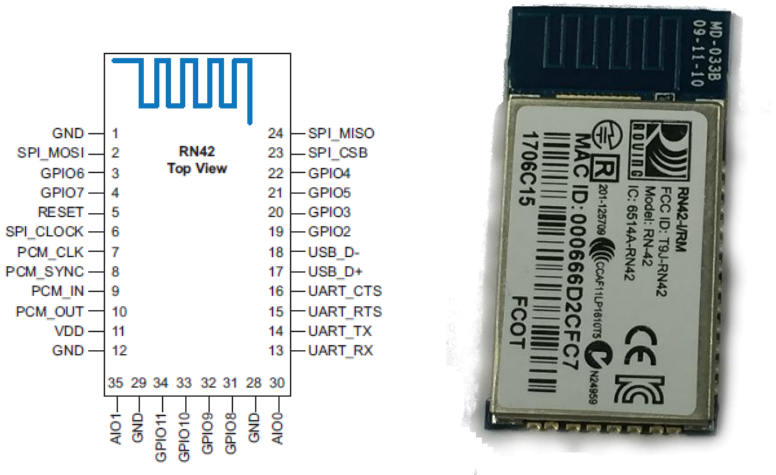


Ilustración 18. Diagrama de pines del módulo Bluetooth y fotografía del componente real.

Batería

La batería es el elemento encargado de suministrar la alimentación al resto de componentes durante el funcionamiento del sistema. Se trata de una batería de litio de designación 1ICP4/20/25. Tiene las siguientes especificaciones: 3.7 V, 155 mA*h, 0.57 W*h. Se trata de una batería pequeña, indicada para aplicaciones de baja potencia.

Además de los terminales positivo y negativo, la batería tiene un tercer pin. Este pin está unido a un termistor (sensor de temperatura) que permitirá al próximo componente controlar el proceso de carga de manera adecuada.

Ilustración 19. Batería utilizada

Charge Management Controller

Como ya se ha dicho, es necesario un componente que se encargue de controlar el proceso de carga de la batería. El dispositivo es el MCP73833 de la compañía Microchip y se define como un controlador de carga linear y avanzado, diseñado especialmente para aplicaciones de bajo coste y tamaño reducido. A continuación, se muestra una imagen de su hoja de datos (hoja de datos [19]) que muestra cómo se debe conectar para un correcto funcionamiento, así como del componente real.

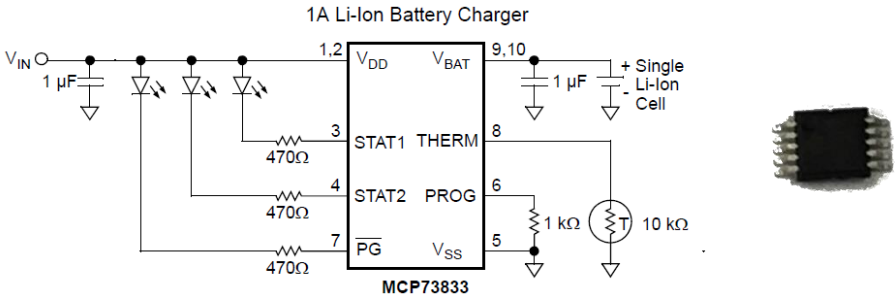


Ilustración 20. Imagen del uso de pines del Charge Management Controller y del componente real.

Se observa que tiene dos 3 pines pensados para conectarse a leds que den información visual del estado de carga. Con el objetivo de ahorrar espacio en la placa, se ha prescindido de ellos. Además del pin VDD (por donde llega la carga) y el pin Vbat (por donde se conecta a la batería para cargarla), el chip incluye dos pines de control. El pin THERM, se debe conectar al termistor de la batería, para poder controlar la temperatura de la misma durante la carga, evitando su deterioro. Las corrientes de carga se escalan a partir de la que circula por el pin PROG al conectarlo a masa mediante una resistencia.

Reguladores de Voltaje

Como se introdujo previamente, muchos componentes electrónicos son sensibles al voltaje de alimentación, el cual varía con el grado de carga de la batería. Además, la batería proporciona un voltaje nominal de 3.7 V, pero los componentes están diseñados para funcionar con una alimentación de 3.3V.

Para lidiar con estos problemas se utilizan los reguladores de voltaje de la compañía Microchip y nombre: TPS709. Se trata de circuitos integrados que proporcionan una tensión de alimentación constante de 3.3 V a partir de una entrada de voltaje variable de entre 2.7 y 30 V. Estos chips están diseñados para aplicaciones muy sensibles a la potencia e incorporan varios sistemas de seguridad: apagado térmico, limitación de corriente y protección para corrientes en sentido contrario.

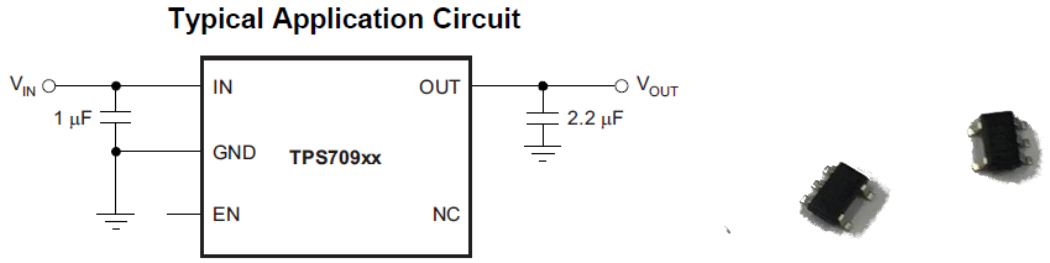


Ilustración 21. Imagen del uso de pines del regulador de voltaje y de los componentes reales.

Llama la atención que, en todo momento, se habla en plural para referirse a estos elementos. Esto se debe a que el ADS tiene una parte digital y una parte analógica que requieren alimentaciones independientes. Así, se utiliza un regulador de voltaje para el circuito de alimentación digital y otro para el analógico. Ambos proporcionan una salida de 3.3 V y ambos se alimentan desde el terminal positivo de la batería. La unión de la masa digital y la masa analógica a la batería se resuelve en el apartado 2.2.4. Para más información de estos componentes, se remite a la hoja de datos [21]

Conector USB MINI

El conector USB mini permite la conexión de la placa con otros elementos del exterior. A través de este conector se programa el microcontrolador y se carga la batería. Para la primera de las funciones se utilizan los tres pines centrales del conector. La programación se realiza en los pines adecuados del microcontrolador (ver apartado [2.2.3.B]) mediante comunicación serie. Para la carga de la batería se utilizan los pines VCC y GND.

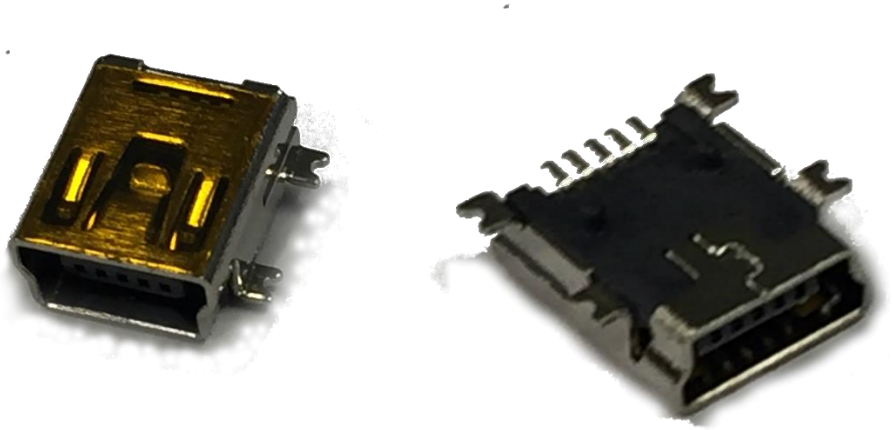


Ilustración 22. Imagen del conector USB y esquema del uso de sus pines

Interruptor

La principal función del interruptor es apagar y encender el dispositivo, de manera que este no esté gastando su batería innecesariamente. Asimismo, se ha aprovechado para separar las funciones de carga de la de alimentación de la placa. Con ello se impide que el usuario pueda usar el dispositivo mientras este se está cargando.

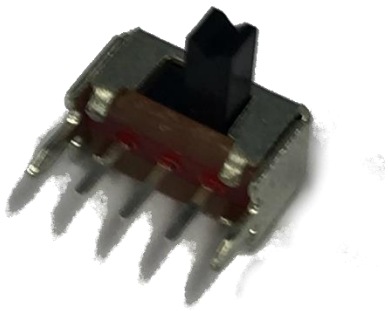
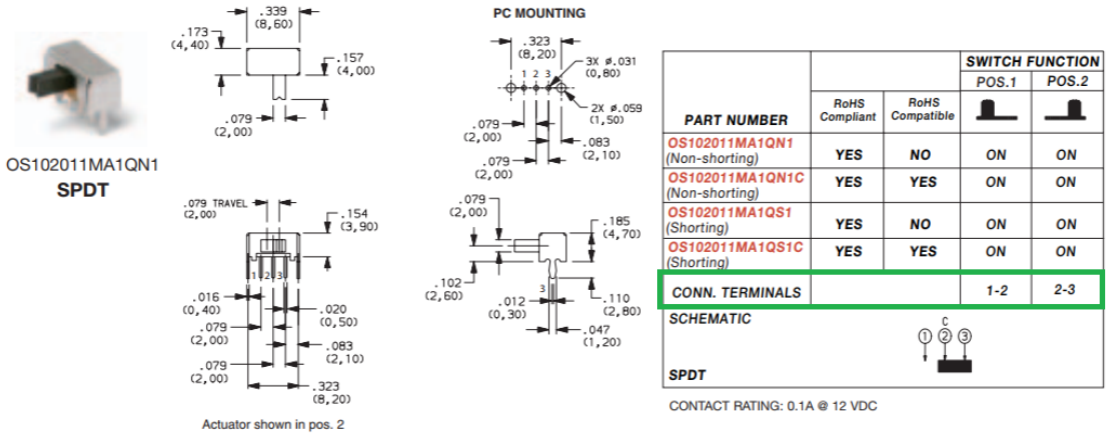


Ilustración 23. Imagen del interruptor utilizado.

En la anterior imagen se observan cinco pines del interruptor. Sin embargo, los dos laterales sirven únicamente para sujetar el interruptor a la PCB y, si se desea, dar a la carcasa metálica un potencial. Son los tres pines centrales los que se encargan de interrumpir o permitir el paso de la corriente en el circuito en cuestión.



PART NUMBER	RoHS Compliant	RoHS Compatible	SWITCH FUNCTION POS.1	SWITCH FUNCTION POS.2
OS102011MA1QN1 (Non-shorting)	YES	NO	ON	ON
OS102011MA1QN1C (Non-shorting)	YES	YES	ON	ON
OS102011MA1QS1 (Shorting)	YES	NO	ON	ON
OS102011MA1QS1C (Shorting)	YES	YES	ON	ON
CONN. TERMINALS			1-2	2-3

Ilustración 24. Recorte de la hoja de datos del interruptor. En verde las posiciones posibles.

Como se aprecia en el extracto de la hoja de datos [22], el interruptor admite dos posiciones: en una se conecta el pin central (2) con uno de los laterales; y en la otra, el pin central con el otro lateral. Se ha previsto unir el pin central (común a las dos posiciones) a la batería, uno de los laterales a la salida del Charge Management Manager (que proporciona la carga a la batería) y el otro a la entrada de los reguladores de voltaje (que proporcionan la alimentación a la placa). Así, el interruptor permitirá alternar entre dos estados: Carga y Alimentación.

2.2.3. Esquema eléctrico

Una vez analizado el sistema a desarrollar y los componentes que se utilizan, se pasará a describir la interconexión de los mismos, que da lugar al esquema eléctrico del dispositivo. El esquemático, a partir del cual se desarrolla la placa de circuito impreso, se encuentra en el anexo A3. En este punto se intentará explicar los aspectos más importantes del mismo.

Antes que nada, es importante destacar que todo el trabajo relacionado con el diseño del circuito tanto a nivel esquemático como a nivel físico se ha realizado mediante la herramienta Eagle de Autodesk.

A. CONEXIONES PRINCIPALES

De nuevo, el flujo de información de la onda de ECG es el punto de partida para comprender el esquema. El microcontrolador es intermediario entre el ADS y el Bluetooth, ya que las interfaces de comunicación de estos dispositivos no son compatibles entre sí. El ADS utiliza el estándar de comunicaciones SPI (detallado en el apartado 2.3.1) para comunicarse con el microcontrolador. El Bluetooth, por el contrario, usa el sistema UART para comunicarse (explicado en el mismo apartado).

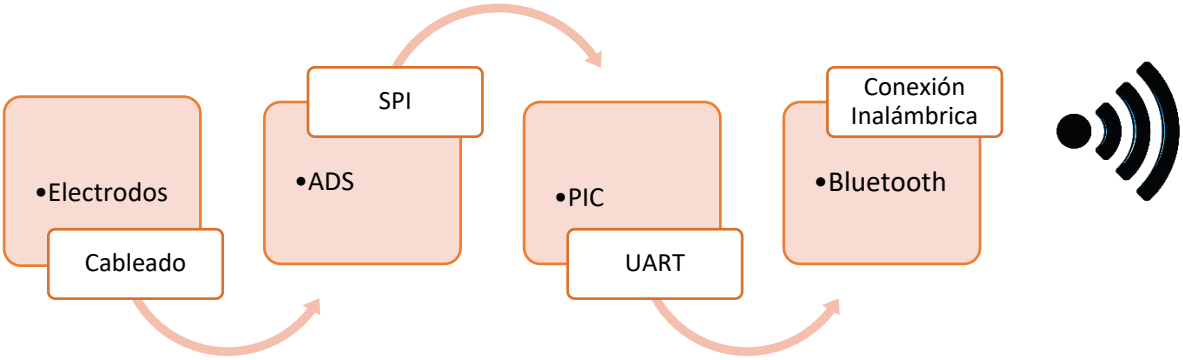


Ilustración 25. Esquema ilustrativo de las interfaces de comunicación entre los componentes del Hardware

En este punto surge el primer problema de diseño. Como se explica en el apartado 2.2.2.2., cada pin del microcontrolador tiene varias funciones, pero solo podrá realizar una de ellas. Concretamente, la funcionalidad de UART y la de SPI comparten varios pines, por lo que será imposible utilizarlas simultáneamente.

La solución pasa por implementar la funcionalidad de UART mediante software en otros pines GPIO (*General Purpose Input Output*) del microcontrolador. En el apartado 2.3.1. se explican los detalles del código utilizado para este fin. ¿La desventaja? Una leve pérdida de velocidad y tiempo de máquina respecto a lo que se obtendría utilizando el hardware, incorporado en el microcontrolador en los pines con UART. Afortunadamente, esta pérdida velocidad no es determinante para el sistema, ya que la frecuencia a la que se muestrean los datos no es demasiado alta y la carga computacional del microcontrolador tampoco es elevada.

ADS y PIC

Al liberar los pines con UART, la comunicación entre ADS y microcontrolador se puede hacer mediante el protocolo SPI. Así, se conectan los pines (“Serial Clock”, “Serial Data Input” y “Serial Data Output”) del microcontrolador con los correspondientes pines del ADS. Cabe decir que no se utiliza el pin “Slave Select” de SPI porque el microcontrolador solo se comunica con un dispositivo mediante esta interfaz.

Para completar la interconexión, queda hablar de los pines de control. De esta forma, se conectan 2 pines de propósito general (GPIO) del PIC a los pines de “Power Down/RESET” y “Chip Select” del ADS. Mediante estos pines se podrá conectar y desconectar el chip a voluntad. Por el contrario, el pin “Start” del ADS no se ha conectado. En su lugar, un comando mandado por la conexión SPI dará la instrucción al ADS de iniciar el muestreo.

Por último, el pin “Data Ready” del ADS, que indica si hay nuevos datos convertidos, se conecta a un pin del PIC que capaz de generar una interrupción. Esto último facilita el desarrollo del programa descrito en el punto 2.3.1.

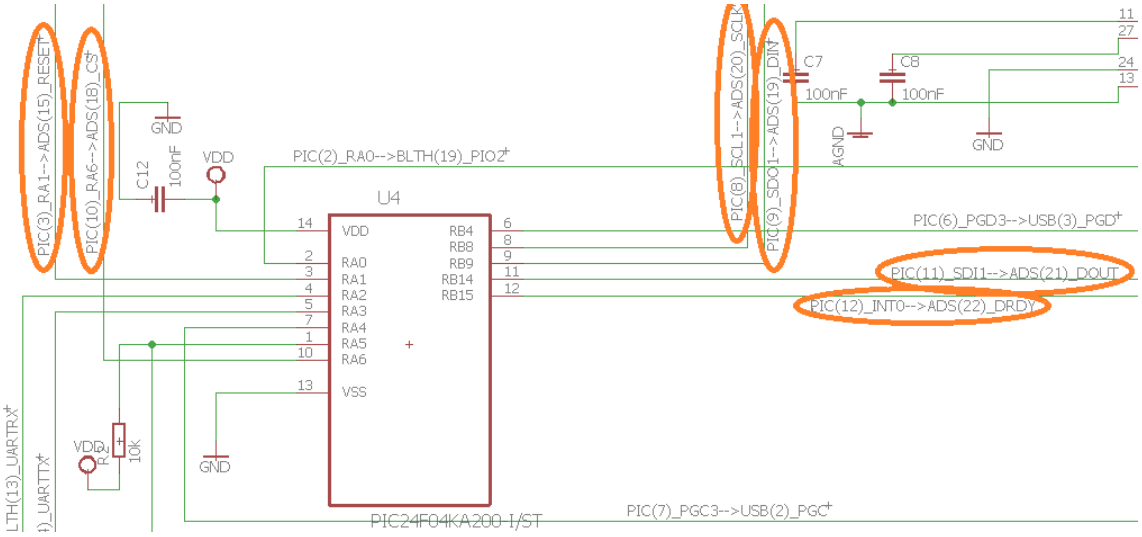


Ilustración 26. Imagen del microcontrolador, extraída del esquemático, remarcando las conexiones con el ADS.

ADS: electrodos y función RLD

Una vez concluida la cuestión relativa a la comunicación PIC-ADS, es importante destacar algunos aspectos que conciernen únicamente al chip ADS1292R, relativos a la adquisición de datos.

En primer lugar, se han pasado por alto las conexiones del ADS a los electrodos. Como es lógico, estos se deben conectar al ADC a través de los pines INPUT. En particular, se ha utilizado el canal 1 (INP1, INN1). Desde estos pines se trazan las pistas de entrada, que desembocan en puntos de soldadura a los que se soldarán los cables de los electrodos.

En el punto 2.2.2.1. ya se habló de ciertas funciones del chip desarrolladas específicamente para su uso como medidor de ECG. Concretamente se habló de la función RLD, que sirve para obtener la tensión de modo común de un conjunto seleccionado de inputs. En el caso que nos ocupa esta tensión de modo común se redirige a través de un amplificador inversor al cuerpo del paciente mediante un tercer electrodo. De esta manera, la tensión de modo común existente se vería compensada (de manera notable), por la tensión expedida a través de este tercer electrodo. Esta funcionalidad se ha implementado siguiendo las indicaciones de la hoja de datos (referencia [10]) tal y como se muestra en la siguiente imagen:

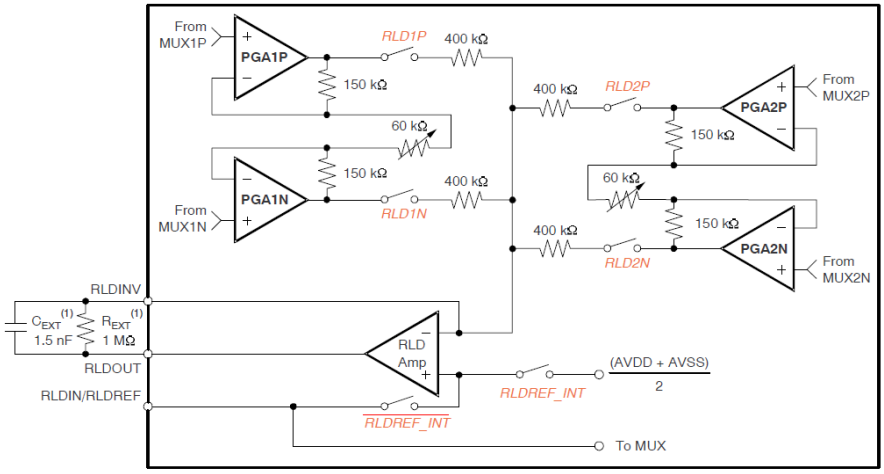


Ilustración 27. Imagen, extraída de la hoja de datos, que indica la forma de conexión de los pines RLD para la función descrita.

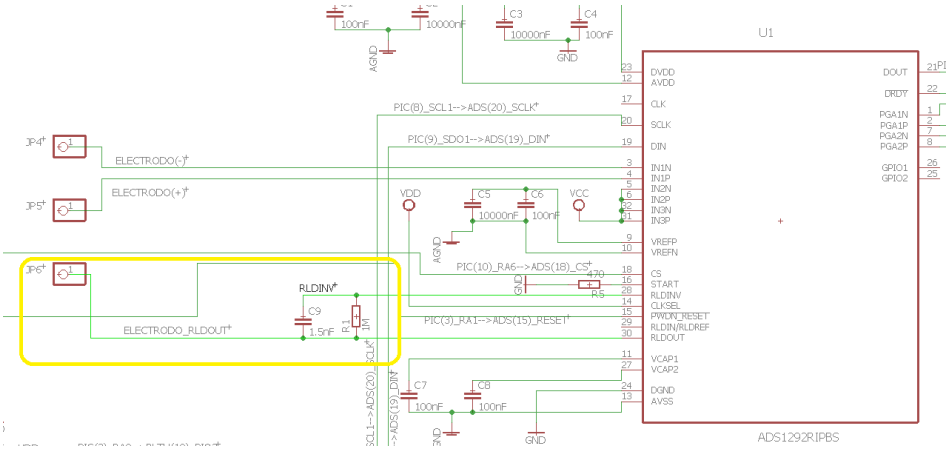


Ilustración 28. Imagen del ADS, extraída del esquemático, remarcando las conexiones descritas y los puntos de soldadura a los que se unirán los cables de los electrodos.

Cabe destacar que se ha decidido utilizar referencia interna y, dado que en este proyecto no es de utilidad la medida de la señal RLD, la entrada RLDIN no se utiliza.

PIC y módulo Bluetooth

Una vez resueltas las conexiones relativas al ADS, se retoma el tema de la comunicación entre el PIC y el módulo Bluetooth. A continuación, se muestra una imagen de la hoja de datos que expone la recomendación del fabricante para la comunicación vía UART del Bluetooth:

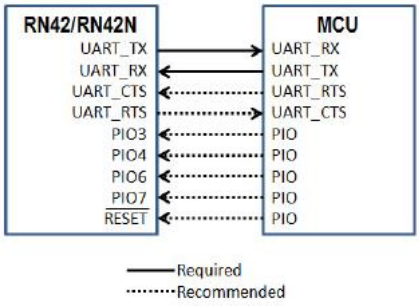


Ilustración 29. Indicaciones del fabricante del módulo Bluetooth para la comunicación vía UART.

Se puede observar que solo dos conexiones son obligatorias. Estas se corresponden con las vías por las que los datos pasan del PIC al Bluetooth y del Bluetooth al PIC. Como ya se ha explicado al inicio del apartado, no se han podido utilizar los pines del microcontrolador destinados a la función UART, utilizándose en su lugar dos pines de propósito general.

Por lo que respecta a las conexiones recomendadas, no son imprescindibles en este proyecto y se ha optado por no realizarlas. Las dos primeras sirven de señal entre emisor y receptor para indicar que el dispositivo está listo para enviar/recibir nuevos datos; lo cual, teniendo en cuenta la frecuencia de muestreo del ADS, no es necesario para el correcto funcionamiento del sistema. Por otro lado, los pines de propósito general sirven para modificar la configuración del módulo Bluetooth. Esto únicamente se hará, si es necesario, al inicio del programa mediante la escritura de sus registros a través de los pines principales de UART. Por tanto, todas las conexiones recomendadas se dejan sin enlazar.

Existen otros pines que no aparecen en las recomendaciones de la imagen anterior, pero que se han considerado de utilidad. Los pines GPIO 5 y GPIO 2 del Bluetooth son pines que muestran el estado de conexión con un dispositivo remoto. En la imagen siguiente se describe su funcionamiento. El primero se ha conectado a un led, tal y como recomienda el fabricante, para dar información visual al usuario de la conexión del dispositivo. El segundo se conecta al microcontrolador a través un pin de propósito general para permitir al programa hacer unas acciones u otras, dependiendo de si el dispositivo ha sido conectado o no.

TABLE 2-1: GPIO5 STATUS

GPIO5 Status	Description
Toggle at 1 Hz	The module is discoverable and waiting for a connection.
Toggle at 10 Hz	The module is in command mode.
High	The module is connected to another device over Bluetooth.

TABLE 2-2: GPIO2 STATUS

GPIO2 Status	Description
High	The module is connected to another device over Bluetooth.
Low	The module is not connected over Bluetooth.

Ilustración 30. Imagen de la hoja de datos del módulo Bluetooth. Funcionamiento GPIO5 y GPIO2.

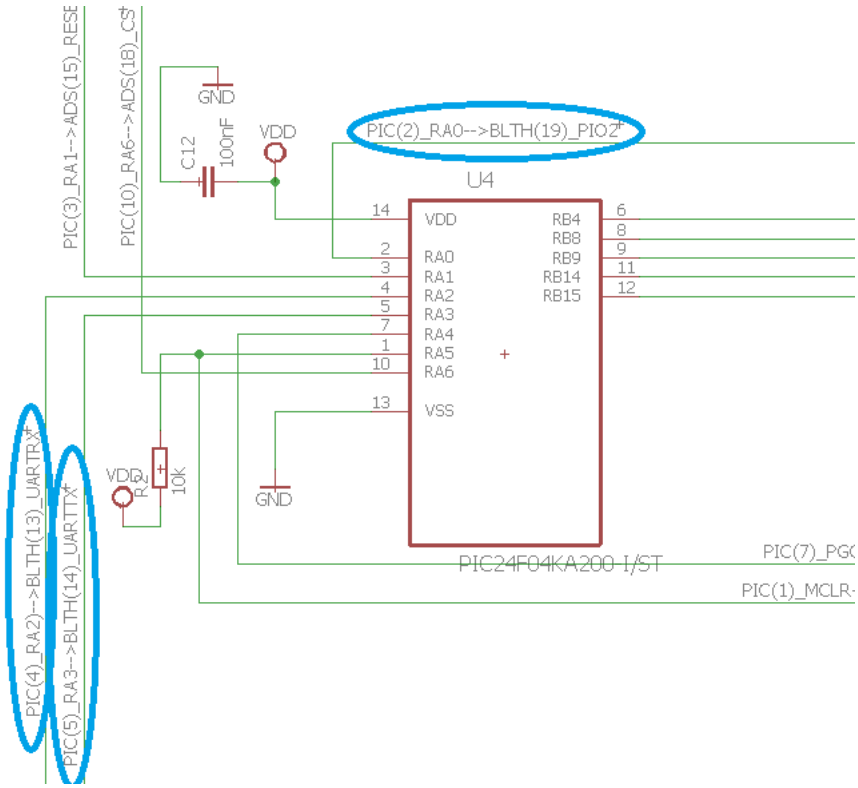


Ilustración 31. Imagen del microcontrolador, extraída del esquemático, remarcando las conexiones con el módulo Bluetooth.

B. CONEXIONES RELATIVAS A LA CARGA Y LA PROGRAMACIÓN

Programación Microcontrolador

Respecto a esto último, es necesario utilizar unos pines concretos del microcontrolador, que tengan la funcionalidad ICSP (In-Circuit Serial Programming). Estos pines (PGD, PGC) junto con el MCLR, indispensable para programar y depurar, se conectan con los tres pines centrales del conector USB de acuerdo a la información aportada en el punto 2.2.2.7.

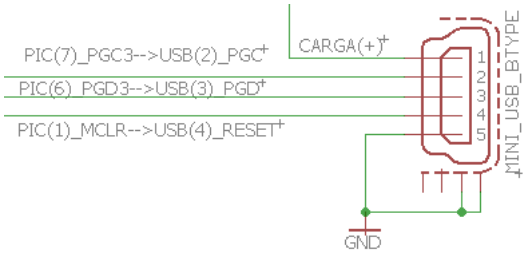


Ilustración 32. Imagen, extraída del esquemático, en la que se muestran las conexiones de los pines del USB relativas a la programación

Charge Management Controller (administrador de carga).

Los otros dos pines del conector USB, como se comentó en la descripción general del hardware, son los que proporcionan la alimentación a la batería durante el proceso de carga. De esta forma, el VCC se une a la entrada del administrador de carga y el GROUND se lleva a masa.

Continuando con el administrador de carga, se lleva su salida (que proporciona la carga) al terminal positivo de la batería a través del interruptor. Asimismo, se conecta el input termistor con el terminal de la batería que controla su temperatura, tal y como se podía deducir de la descripción de los componentes en el punto 2.2.2.5.

Interruptor

El interruptor, como ya se expuso anteriormente, tiene dos posiciones posibles: en una se cierra el circuito entre el administrador de carga y el terminal positivo de la batería (modo carga); mientras que, en la otra, este terminal está unido a la entrada de los reguladores de voltaje alimentando la placa (modo encendido).

Por tanto, se conecta el pin central al terminal de la batería; y los otros dos pines, a la salida del battery manager y a la entrada de los reguladores de voltaje, respectivamente.

Batería

Por último, los terminales de la batería, al igual que se hizo con los electrodos, se conectan a la placa mediante puntos de soldadura a partir de los cuales brotan las pistas ya descritas.

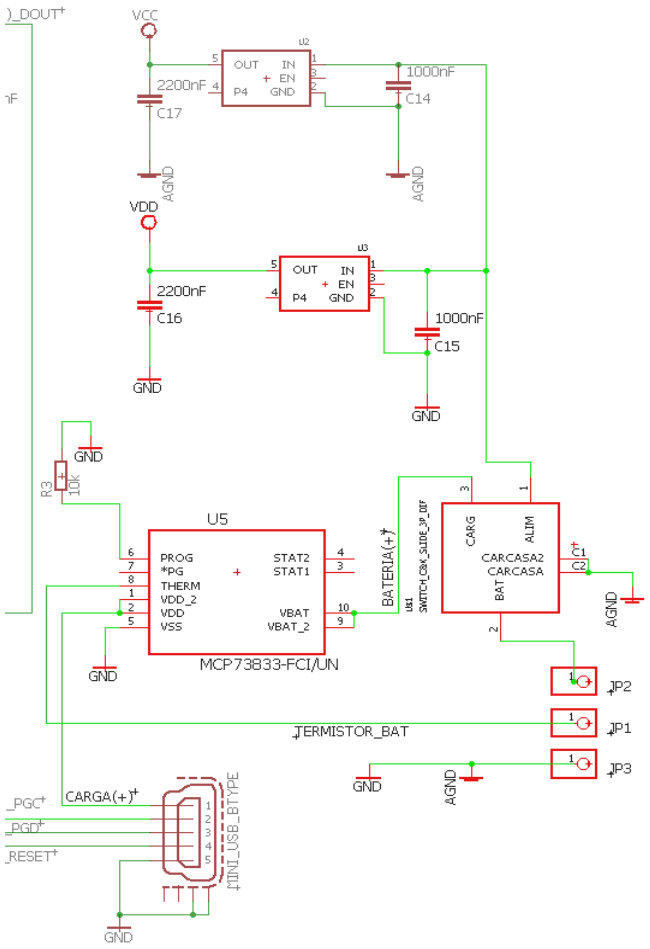


Ilustración 33. Extracto del esquemático. En él aparecen todas las conexiones relativas a la alimentación.

C. RESTO DE CONEXIONES

El resto de conexiones del esquemático, incluyendo los reguladores de montaje no se consideran fundamentales para la comprensión del trabajo. Estas se basan en las recomendaciones de los fabricantes de los circuitos integrados (referencias [10]-[22]) para el correcto funcionamiento de los mismos.

A continuación, se encuentra una imagen del esquema eléctrico desarrollado.

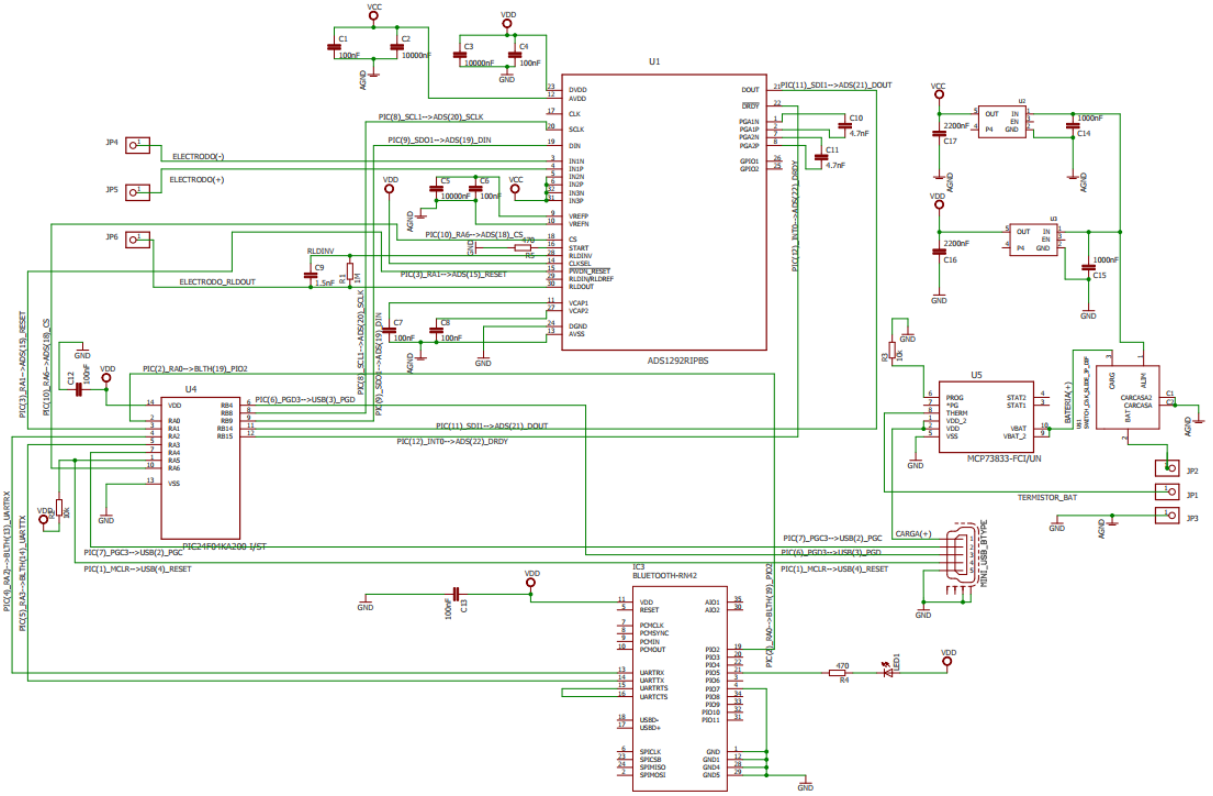


Ilustración 34. Esquemático diseñado. Para más detalle, ir al anexo A3.

2.2.4. Circuito Impreso

A. DISEÑO DEL CIRCUITO

Hasta el momento se han descrito los componentes utilizados y la forma en la que se deben conectar para conseguir que el dispositivo funcione adecuadamente. Así, por lo que respecta al hardware del proyecto, solo queda la implementación física del esquema en un circuito impreso. Esta labor se ha realizado también mediante la herramienta Eagle, y el resultado final se encuentra en el anexo A4.

A diferencia del diseño del esquemático, esta parte del trabajo es muy abierta. A la hora de plasmar un circuito en una placa existen prácticamente infinitas posibilidades en cuanto a la disposición de los componentes y al trazado de pistas. Por ello, el diseño ha seguido una serie de directrices que aseguran el funcionamiento correcto del dispositivo, así como el cumplimiento de las especificaciones. Estas reglas de diseño, junto con las soluciones que derivan de ellas, se encuentran descritas a continuación:

- La principal especificación que debía cumplir el producto es el tamaño. Ante todo, se busca que la movilidad del usuario se vea mínimamente limitada, lo cual requiere un volumen reducido. A continuación, se tiene una imagen de todos los componentes a utilizar a tamaño real:

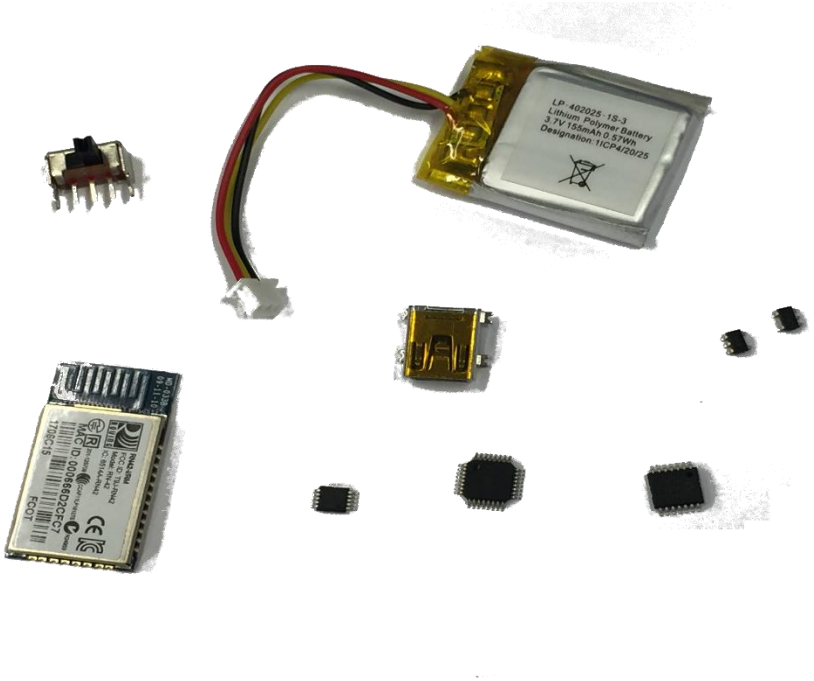


Ilustración 35. Imagen de todos los componentes que conforman el dispositivo.

Se puede ver que el reactivo limitante en este problema es la batería. Así, en la medida de lo posible, se ha intentado que las dimensiones de la placa coincidan con las de la batería. En la implementación final, la placa se monta sobre la batería aprovechando el espacio al máximo.

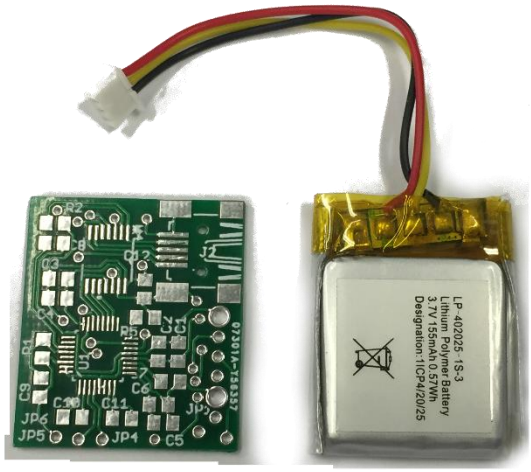


Ilustración 36. Imagen comparativa del tamaño de la batería y de la placa final.

Dado el tamaño de los componentes (en especial del Bluetooth), se necesitan dos capas, superior e inferior, en las que distribuir los elementos.

- El Bluetooth, además de las dificultades derivadas de su tamaño, tiene ciertos requerimientos propios. El fabricante indica en la hoja de datos (referencia [11]) que la antena del módulo quede al aire, es decir, que no debe apoyar en la placa.

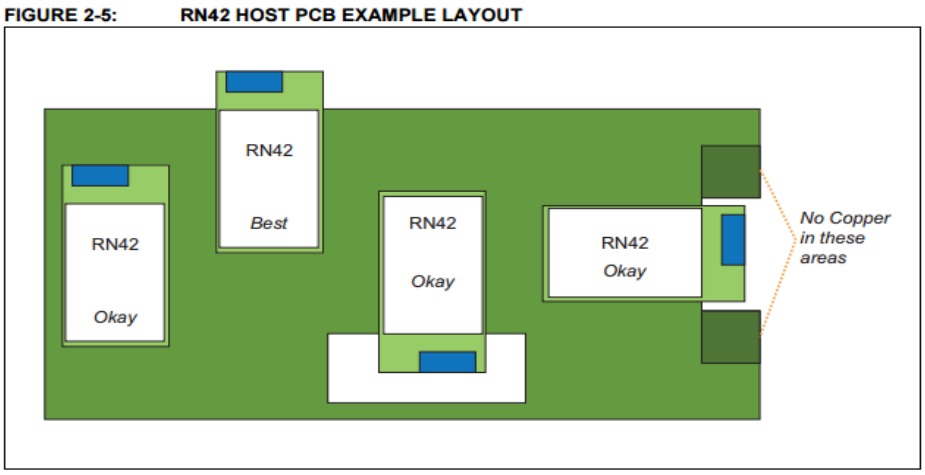


Ilustración 37. Imagen de la hoja de datos del Bluetooth. Indica las posiciones aceptadas del módulo sobre la placa.

De lo anterior se deduce que la presencia de señales eléctricas próximas a la antena podría acarrear interferencias, conduciendo a una transmisión defectuosa de la información. En consecuencia, y siguiendo el consejo de los tutores del proyecto, se ha tomado una precaución más en este sentido. Dentro de las limitaciones existentes, se ha intentado distanciar al máximo la antena del Bluetooth de la parte analógica del circuito, en especial de los puntos de soldadura de electrodos y batería. Esto queda remarcado en la imagen siguiente:

Ilustración 38. Imagen de la l Bluetooth sobre la placa en la posición que ocupará.

- Desde un punto de vista funcional, algunos componentes deben ser accesibles desde el exterior, bien porque son manipulados por el usuario, bien por necesidades técnicas de cara al montaje. Estos elementos deben situarse en la periferia de la placa y, en ciertos casos, en una posición determinada. A continuación, se exponen los componentes con esta limitación:
    - Interruptor. El accionamiento manual debe quedar en el exterior.
    - Conector USB. En este caso el componente debe sobresalir ligeramente de la placa para que, aun con el empaquetado, la conexión se pueda producir sin dificultad.
    - Puntos de soldadura. Si se conectara un cable en un punto central de la placa impediría posibles arreglos o retoques una vez la batería se hubiera colocado en su lugar (pegada a la parte inferior de la placa).

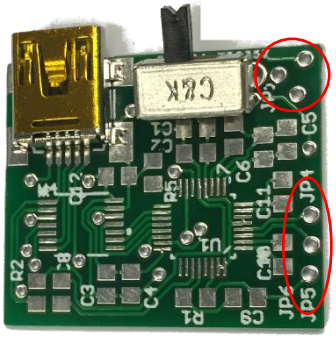


Ilustración 39. Placa con los elementos periféricos colocados.

- De la misma manera que algunos elementos se prefieren en la periferia del aparato, existen otros que conviene colocar en el interior de la placa. La razón principal es la gran cantidad de pines en uso (a menudo en todas las direcciones) que tienen algunos circuitos integrados. Así, en el proceso de diseño de la placa se ha tratado de dejar al ADS y al PIC en una posición central.

- Además, es conveniente que la distancia de la pista por la que se transmiten los datos entre ambos elementos sea lo más reducida posible. Con ello se minimizan las posibilidades de error en la transmisión. Por ello, se ha situado ambos integrados en la misma capa y uno al lado del otro (por el lado de los pines SPI). Siguiendo la misma lógica, se ha situado en la misma capa el conector USB, próximo a los pines de programación del microcontrolador.

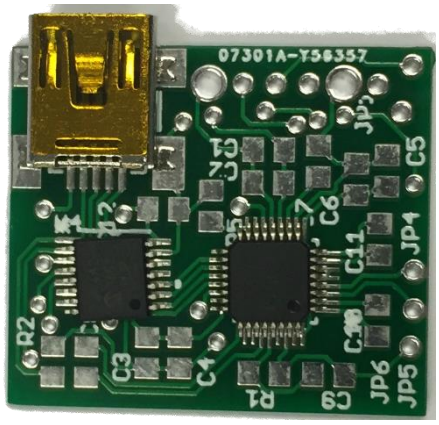


Ilustración 40. Imagen de la placa con los elementos centrales colocados.

- Como se comenta en el punto 2.2.2.6., el ADS requiere que la alimentación para la parte digital y la alimentación para la parte analógica pertenezcan a circuitos diferentes. Para ello se disponen dos reguladores de voltaje distintos, ambos con la entrada unida al terminal positivo de la batería. Sin embargo, hasta el momento no se ha dicho nada de la diferenciación entre la masa de la alimentación digital y la de la alimentación analógica.
  El circuito de masa debe ser único para conectarse a la batería, que proporciona la energía a ambas partes de la placa. Al mismo tiempo, los componentes no deben percibir la influencia digital en la parte analógica. Así, se trazarán circuitos "cuasi-independientes" para la alimentación de los circuitos integrados, que solo se unirán en el punto más lejano posible: el terminal negativo de la batería.

- Una vez los aspectos relacionados con la situación espacial de los componentes han sido considerados, solo queda por explicar el trazado de las pistas. En este sentido se recomienda el uso de pistas cortas y anchas, evitando recodos en ángulo recto o agudo. Con ello, la potencia perdida en los conductores se minimiza.
  Aunque ninguno de los componentes de los que se compone el proyecto requiere alta potencia (y por tanto alta intensidad), en el trazado se han intentado seguir las recomendaciones expuestas para evitar problemas. No obstante, una de las recomendaciones entra en conflicto con los objetivos de diseño. El tamaño de la placa se considera un factor de mayor relevancia que el grosor de las pistas, dado que todos los componentes son de baja potencia. Por ello, se ha utilizado el mínimo grosor de pista posible (ver punto 2.2.4.).

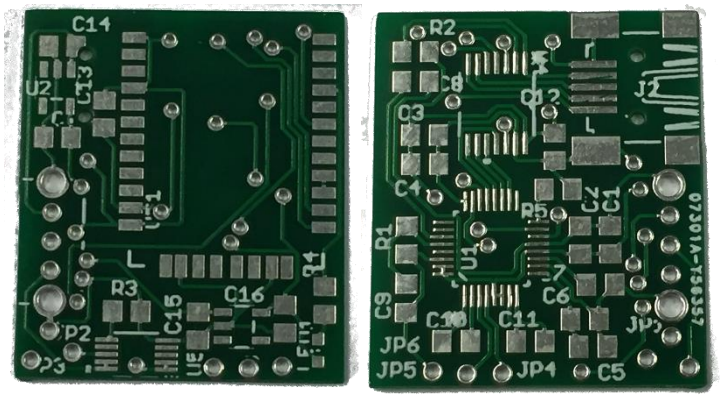


Ilustración 41. Imagen de las dos capas que componen las PCB

B. FABRICACIÓN Y MONTAJE

Hasta el punto presente, se ha explicado la metodología utilizada para el diseño del dispositivo mediante la herramienta informática Eagle. El paso siguiente es construir un prototipo físico. Debido al tamaño de la placa y de las pistas, resulta imposible hacerlo con métodos no profesionales. Se encomienda su fabricación a la empresa Itead.cc.

Para ello, la empresa expone las siguientes condiciones a cumplir por el diseñador:

Tabla 1. Requerimientos del fabricante de PCB

Layers	1 - 4
Material	FR-4
Board Dimension (max)	380mm X380mm
Board Dimension (min)	10mm X10mm
Outline Dimension Accuracy	± 0.2mm
Board Thickness	0.40mm--2.0mm
Board Thickness Tolerance	± 10%
Dielectric Separation thickness	0.075mm--5.00mm
Conductor Width (min)	0.15mm(Recommend>8mil)
Conductor Space (min)	0.15mm(Recommend>8mil)
Outer Conductor thickness	35um
Inner Conductor thickness	17um--100um
Copper to Edge	>0.3mm
Plated Component,Plated via Diameter(Mechanical)	0.3mm--6.30mm
Plated Hole Diameter Tolerance(Mechanical)	0.08mm
Unplated Hole Diameter Tolerance	0.05mm
Hole Space(min)	0.25mm
Hole to Edge	0.4mm
Annular Ring(min)	0.15mm
Aspect Ratio	8:01
Solder Resist Type	Photosensitive ink
Solder Resist Color	Black ,Green, White, Blue ,Yellow
Solder Resist Clearance	0.1mm
Solder Resist Coverage	0.1mm
Plug Hole Diameter	0.3mm--0.65mm
Selective Finish	HASL, ENIG
Silkscreen line width (mim)	6mil

El punto más conflictivo de las condiciones era el ancho de las pistas ya que, para minimizar el tamaño del dispositivo, se quería hacer las pistas lo más estrechas posible. Por ello se ha escogido, de las anchuras normalizadas que ofrece el programa, la inmediatamente superior al mínimo permitido: 0.01 pulgadas (el mínimo es 8 milésimas de pulgada). El resto de indicaciones no han sido un problema, al no ser limitantes para el diseño.

Una vez diseñada la PCB y comprobado que cumple con las condiciones necesarias, se envió a la empresa los planos Gerber necesarios para su fabricación, tal y como se solicita en la página del fabricante.

Una vez se recibió la placa de circuito impreso que había sido mandada a fabricar, hubo que soldar los componentes en el laboratorio. Esta labor se realizó utilizando el equipo de soldadura RMSE-2C.

A continuación, se muestra el resultado tras el montaje de los componentes.

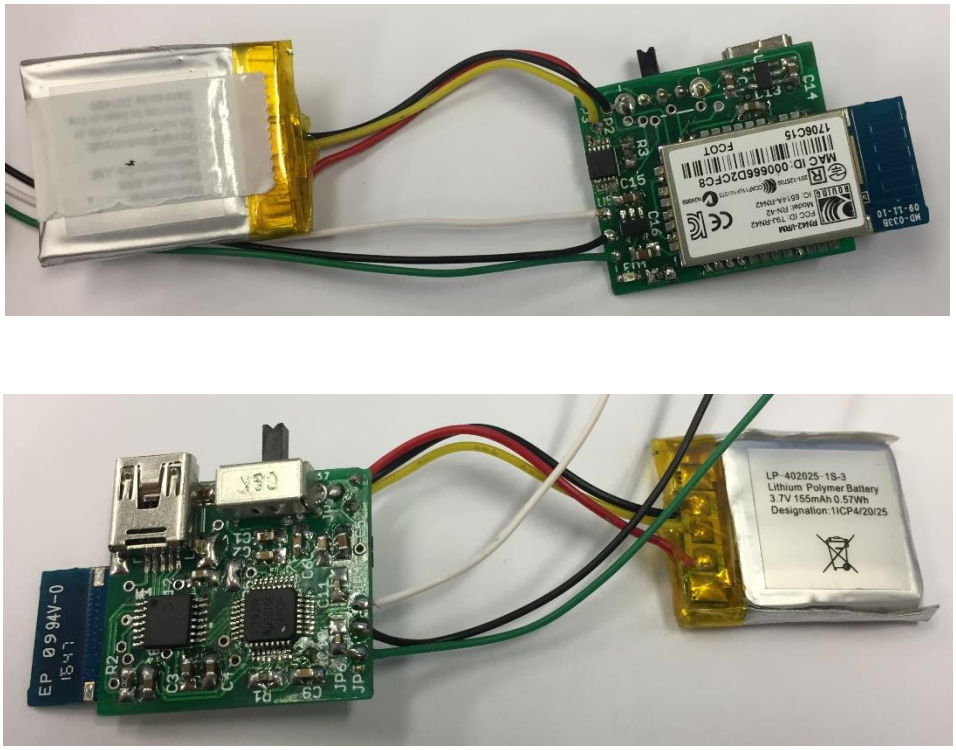


Ilustración 42. Imagen de las dos capas (TOP y BOTTOM) de la PCB con los componentes soldados.

C. UNIÓN AL CUERPO.

Para la unión del dispositivo al cuerpo del paciente se requiere el uso de tres electrodos, dos para las entradas del electrocardiograma y el tercero para la salida RLD. Se utilizan los electrodos mostrados a la siguiente imagen, con adhesivo incorporado y sin gel conductor para electro-estimulación.

Ilustración 43. Imagen de un electrodo por delante y por detrás.

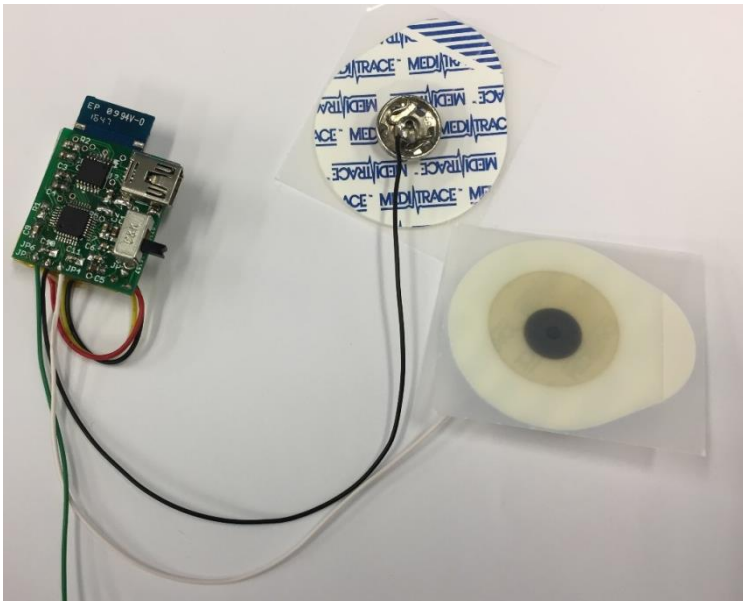


Ilustración 44. Imagen de la placa montada con dos electrodos unidos.

Con los electrodos se permite la entrada de la señal a medir, pero no se tiene una sujeción al cuerpo. Así, es necesario otro adhesivo que mantenga al dispositivo electrónico unido al pecho del paciente. Lo ideal sería, además, diseñar un recipiente para el dispositivo, de manera que la electrónica no quedara a la vista, pero que se pudiera acceder fácilmente a los elementos periféricos como el USB y el interruptor. Sin embargo, dado que se trata de un prototipo, se ha decido posponer este aspecto para futuras ampliaciones del proyecto, una vez la validación del mismo haya sido concluida.

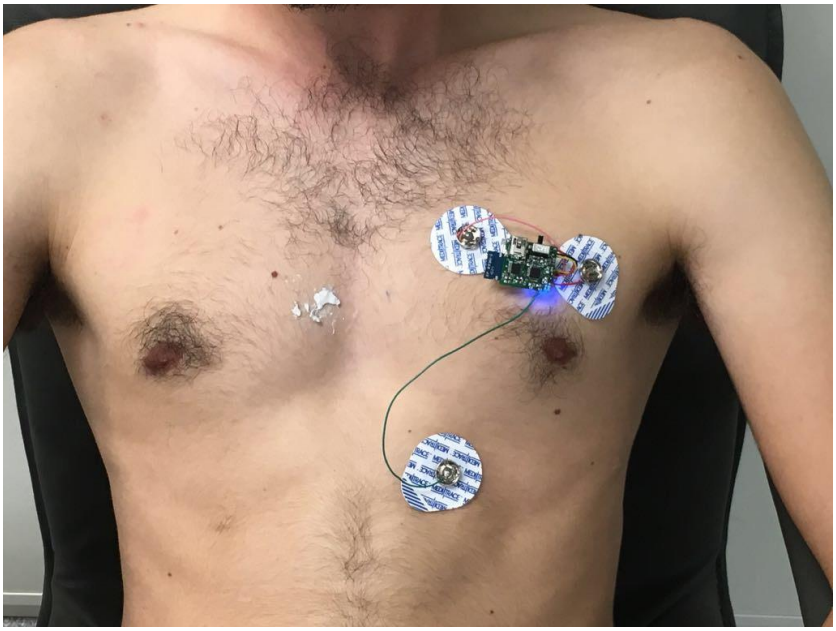


Ilustración 45. Imagen de la situación final del dispositivo sobre el paciente. Se aprecia que la posición de los electrodos de medida es muy próxima entre sí, y están situados sobre la aurícula. El electrodo de realimentación negativa es el más alejado.

2.3. FIRMWARE DEL MICROCONTROLADOR

El firmware se define como la lógica de bajo nivel que gobierna la electrónica de un dispositivo. Esta lógica se introduce en forma de programa informático en el microcontrolador y debe controlar al resto de componentes para que cumplan su función correctamente. Este programa se almacena en la memoria no volátil del microcontrolador, porque será arrancado cada vez que la placa reciba alimentación. Debe recalcarse que en la programación se hace uso de una librería perteneciente al PIC: <24F04KA200.h>.

Para comprender el funcionamiento del programa, es necesario aclarar ciertos conceptos. Se ha dividido el presente apartado en varios sub-apartados para explicar de manera diferenciada estos conceptos y, al final, el programa en sí.

2.3.1. Interfaces de comunicación.

A. SPI

El primer sistema de comunicación es la interfaz mediante la que se comunican el ADS y el microcontrolador: SPI (Serial Peripheral Interface). Se trata de un estándar de comunicaciones diseñado para la comunicación entre circuitos integrados que utiliza un bus serie.

Se compone de 4 conectores:

- SS (Slave Select). Para seleccionar el dispositivo con el que se desea comunicar
- MOSI (Master Output, Slave Input)
- MISO (Master Input, Slave Output)
- SCLCK (Serial Clock). Los pulsos de la señal de este conector, mandada por el master, introducen un nuevo bit en los conectores pertinentes. Además, indica a los receptores que hay un nuevo bit que leer.

Por lo que respecta al microcontrolador, la comunicación se realiza a través del registro SPI1BUF. Es un registro de 16 bits en el que se introducen los bits de entrada (recepción) y se almacenan, antes de ser enviados, los bits de salida (transmisión).

Lo importante en este punto es aclarar que ambos procesos están ligados. Cada vez que se produce un pulso de SCLK, el último bit del registro se envía por el Output; al mismo tiempo que el último bit del otro dispositivo se recibe a través del Input, almacenándose en la primera posición del registro. En el proceso, los bits intermedios se desplazan una posición.

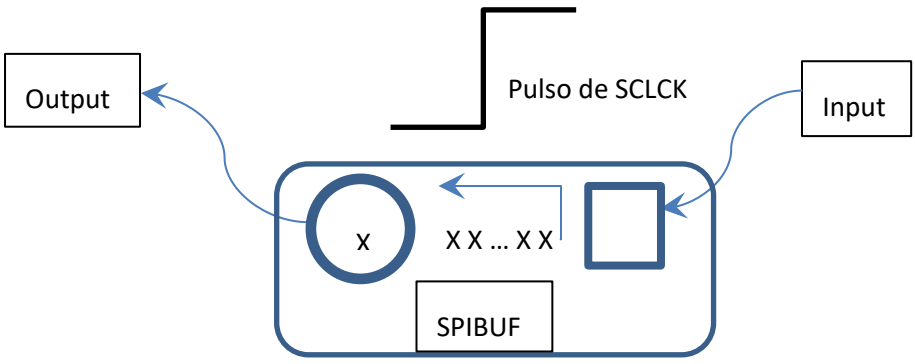


Ilustración 46. Funcionamiento del registro SPI1BUF del PIC en la comunicación SPI

Así, para enviar un dato, solo hace falta escribirlo en el registro SPI1BUF y producir la cantidad necesaria de pulsos de reloj (tantos como bits tenga el dato a enviar). Todo ello se hace con la función spi_write(dato) de la librería del microcontrolador. En el proceso, el registro se rellenará con los bits enviados por el otro dispositivo, tanto si son datos de utilidad como si son datos "basura".

Para recibir un dato, una vez seguros de que el otro dispositivo está listo para enviar, hay que producir la cantidad necesaria de pulsos de reloj y guardar el contenido del registro SPI1BUF en una variable. En el proceso, se envía el anterior contenido del registro, por lo que hay que asegurar que el otro dispositivo no va a malinterpretar esta información, confundiéndola con una instrucción. Para ello, se utiliza la función spi_write() de nuevo. La función envía un dato conocido que, por un lado, requiera los ciclos de reloj deseados; y, por otro lado, se sepa que no va a tener influencia en el receptor.

B. UART

El sistema por el cual se comunican el microcontrolador y el Bluetooth es el UART (Universal Asyncronous Receiver Transmitter). Se trata también de un sistema de transmisión serie, con la particularidad de que no hay señal de reloj compartida entre los dispositivos. Dada esta falta de sincronismo, es necesario que receptor y transmisor compartan la misma velocidad de bits por segundo (baudios). Es decir, el emisor debe compartir con el receptor el mismo período destinado a cada bit, para que no haya pérdida de información o una mala interpretación de la misma.

Lo más importante de esta interfaz de comunicación, es que los datos se envían seccionados en bytes (8 bits). De esta manera, el receptor UART sabrá que el dato ya se ha terminado de obtener al muestrear el octavo bit.

Por otro lado, el receptor debe saber cuándo empezar a muestrear el primer bit de un nuevo dato. El pin por el cual se transmite la información está a nivel alto de voltaje siempre que no se estén enviando datos. Así, para enviar un nuevo byte, la primera acción es pasar a nivel bajo, enviando el bit START. Este bit no contiene información, su única función es indicar al receptor la llegada de un nuevo dato. A continuación, se envían los ocho bits en serie, con el período por bit definido. Al finalizar, el pin vuelve a pasar a nivel alto (bit STOP) y se queda en esa posición hasta que se desee enviar un nuevo byte.

Start bit	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8	Stop Bit
0	0	1	0	0	1	0	1	0	1

Ilustración 47. Envío de u byte por UART.

La funcionalidad descrita es llevada a cabo, normalmente, por hardware específico que se encuentra en los pines con esa funcionalidad. Sin embargo, ya se comentó que en este proyecto los pines de UART están ocupados, por lo que se debe realizar la transmisión por software. Para ilustrarlo, se adjunta el fragmento del código utilizado para tal fin:

```
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#use rs232(xmit=PIN_A2, rcv=PIN_A3, baud=115200, stream=BT)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
putc(0xAA,BT);
fprintf(BT,"$$$");
//-------------------------------------------------------
```

Ilustración 48. Código necesario para implementar la funcionalidad UART por software.

La primera instrucción "#use rs232(…)" indica que se hace uso del estándar rs232 en unos pines determinados. El estándar es una norma para intercambio de datos binarios. Con esta instrucción, se está configurando la comunicación UART para realizarse por software en los pines: PIN_A2 (transmisión) y PIN_A3 (recepción), con una velocidad de 115200 bits/segundo (baudios). La etiqueta BT sirve como identificador del estándar que se acaba de definir en estos pines. De esta manera cuando se envíen datos mediante las instrucciones putc() y fprintf() con la etiqueta BT, los datos se escribirán en el pin definido.

2.3.2. Trama de datos del ADS al PIC

Un aspecto muy importante para el desarrollo del código es la información que el ADS envía al PIC vía SPI cada vez que se produce la comunicación. Además, es necesario saber en qué orden llega dicha información. De esta manera el programa podrá seleccionar la parte útil y enviarla de manera adecuada al Bluetooth.

Independientemente del modo de funcionamiento en el que se encuentre el ADS (continuo o single-shot), la trama de datos que se lee en su Output es la misma. Cada vez que se produzca la señal Data-Ready, el chip dispondrá de 72 bits a la salida. Los primeros 24 son los llamados bits de estado, que

proporcionan información de la situación de los electrodos (funcionalidad Lead-Off) y de los pines de GPIO. Los últimos 48 se dividen en 24 para un canal y 24 para el otro canal.

De los 24 bits de estado, los 4 primeros y los 13 últimos no proporcionan ninguna información. De los 7 restantes, los 5 primeros transmiten la información de Lead Off y los otros dos, la de los pines GPIO. Respecto a los bits de los canales, estos se corresponden con el dato convertido en cada canal codificado en complemento de a 2; siendo el primer bit el más significativo.

El proyecto desarrollado solo utiliza uno de los canales posibles. Sin embargo, en este caso la trama seguirá teniendo 72 bits, siendo todo 0's los bits correspondientes al canal que no se usa. Así pues, hay que leer estos ceros aunque no tengan ninguna utilidad. De la misma forma, se deben leer todos los bits de estado aunque solo 3 sean de interés (ya que los pines GPIO no se utilizan y, del Lead Off, solo importa el electrodo RLD y los dos electrodos de entrada del canal utilizado).

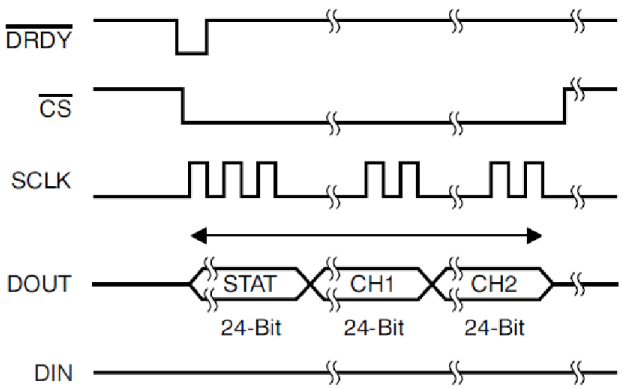


Ilustración 49. Bus de salida SPI del ADS1292R.

En la imagen anterior, se aprecia que el conector ChipSelect (CS) está a nivel bajo durante toda la transmisión y a nivel alto el resto del tiempo. Para ello, en el código del microcontrolador se da la instrucción de poner a nivel bajo el pin de CS antes de la lectura, y de volverlo a poner a nivel alto al finalizar.

También se aprecia que la señal DataReady indica que los datos están listos para ser enviados. Por ello, se habilita la interrupción de este pin en el código del microcontrolador. Como se verá, es en la interrupción donde se produce la transmisión de información, tanto con el ADS como con el módulo Bluetooth.

2.3.3. Código.

A. DIAGRAMA DE FLUJO

Para explicar el funcionamiento del código que se encuentra en el anexo A3, resulta útil el empleo del diagrama de flujo. Se trata de un croquis que ilustra la secuencia lógica del programa mediante bloques de acción (rectángulos) y decisión (rombos). Con este esquema, se facilita la comprensión del

programa en líneas generales, evitando la aspereza que implica la lectura del código. Además del diagrama en sí, se han añadido comentarios a ciertas partes para completar la información.

Para la comprensión de este diagrama de flujo y los que le suceden se recomienda la lectura de la leyenda, en el Anexo A1.

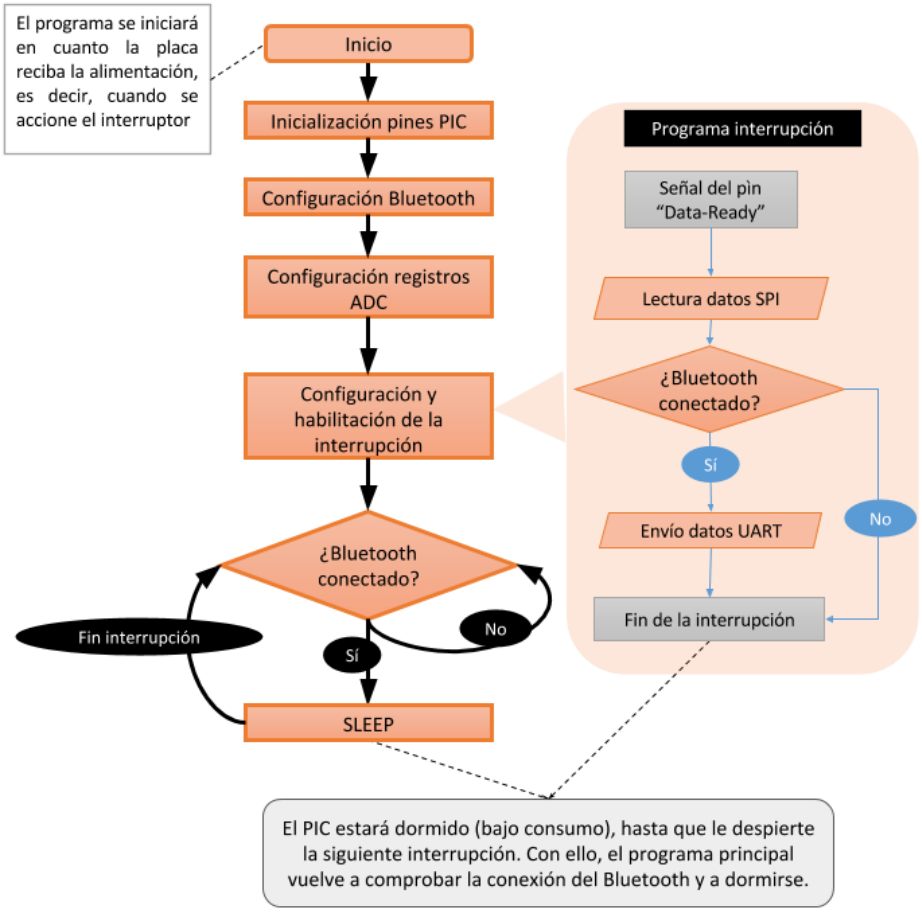


Ilustración 50. Diagrama de flujo del código del microcontrolador.

B. ASPECTOS DESTACABLES DEL CÓDIGO

Con el diagrama anterior es posible entender el funcionamiento del programa. Sin embargo, se considera necesario incidir en ciertos aspectos para entender en profundidad la actuación del sistema.

CONFIGURACIÓN DEL ADS

En primer lugar, es importante tratar la configuración del ADS. Como ya se ha dicho, este punto es de vital importancia ya que determina el modo de funcionamiento del ADS y, por tanto, del sistema global. Esta configuración se realiza mediante la escritura de registros, tal y como se especifica en la hoja de datos referencia [19]. Para una obtener una visión de detalle de esta parte, se recomienda la lectura

de la función initADS() del código en el anexo A3, junto con el punto "Register Map" del capítulo "Theory of operation" de la citada hoja de datos. A continuación, se exponen los rasgos de mayor relevancia de la configuración:

- Modo continuo de funcionamiento, con frecuencia de muestreo: 250 muestras/seg. Ver apartado 1.3.7
- Referencia interna de 2.42V. Ver apartado 1.3.6.
- Activación de los comparadores Lead Off, con los valores por defecto (de corriente y de umbrales de comparación). Apartado 2.2.2.1
- Canal 1: en modo normal de funcionamiento, con ganancia unidad del amplificador. Ver apartado 1.3.6.
- Canal 2: se dejan todos los valores por defecto (el canal 2 no se utiliza).
- Pines GPIO: tampoco se utilizan, por lo que se dejan los valores por defecto.

RECEPCIÓN Y ENVÍO DE DATOS

Otro aspecto importante del programa es cómo el microcontrolador realiza su función de intermediario en la transmisión de la señal. Del diagrama de flujo se extrae que la comunicación SPI se produce en el seno de una interrupción asociada al cambio de la señal del pin DATA READY.

En la comunicación UART, los datos se deben comunicar mediante bytes (8 bits). Es por ello que se hacen lecturas del registro SPIBUF cada 8 ciclos de reloj. De esta manera, los datos se almacenan en variables de 8 bits de longitud, que se podrían enviar directamente por UART. Esto se realiza alternando la lectura del registro, con el envío de un byte conocido (0x00) que produce 8 ciclos del Serial Clock sin afectar al ADS.

Tal y como se explicó en el apartado 2.3.1., la trama de datos consiste en 72 bits, de los cuales, solo son de interés los 5 de Lead Off (pertenecientes al primer conjunto de 24 bits) y la señal de electrocardiograma (que reside en el último conjunto).

En el código actual no se envían los bits de estado de los electrodos. Esto es así, porque se ha preferido centrar los esfuerzos en la correcta transmisión de los datos del ECG, posponiendo el uso de esta función a un desarrollo futuro. De esta manera, solo se utilizarán las variables e2, e1, e0.

```
#int_EXT0                      // Duracion de la rutina 0.9 .. 1.1 ms
void  EXT0_isr(void){          // Interrupcion cada vez que DRDY pasa a 0: 250 sps = 4 ms

    //Data Sheet del ADS: la señal en el pin CS debe estar a nivel bajo durante toda la comunicación SPI
    output_low( CS_ADS);

    //Secuencia de datos de estado: 1100 LOFF_STAT[4:0] GPIO[1:0] 13 0's ==> Total: 24 bits
    // LOFF_STAT[4:0] y GPIO[1:0] son bits de dos registros de estado del ADS
    spi_write(0);s2= SPI1BUF;   // Primeros 8 bits: 1100 LOFF_STAT[4:1] = [1 1 0 0 RLD_STAT IN2N_OFF IN2P_OFF IN1N_OFF]
    spi_write(0);s1= SPI1BUF;   // Segundos 8 bits: STAT[0] GPIO[1:0] cinco 0's= [IN1P_OFF GPIO1 GPIO0 0 0 0 0 0]
    spi_write(0);s0= SPI1BUF;   // Últimos 8 bits:  [0 0 0 0 0 0 0 0]

    spi_write(0);d2= SPI1BUF;   // Datos del canal 1 (24 bits)
    spi_write(0);d1= SPI1BUF;   // Señal de Electrocardiograma
    spi_write(0);d0= SPI1BUF;

    spi_write(0);   spi_write(0);   spi_write(0);   // Datos del canal 2: no lo almaceno

    delay_us(50);
    output_high(CS_ADS);

    if(!input(BT_Connect))
    {
        if(n==0)  putc(0xAA,BT);    // Cabecera de la trama: cada 3 envios
        putc(d2,BT);                // Dato de 24 bits  MSB
        putc(d1,BT);
        putc(d0,BT);                //                  LSB
        n++;
        if(n>2) n=0;                // Define el numero de datos por trama: 3 datos
    }

}// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Ilustración 51. Código de la interrupción. Recepción y envío de datos.

Para el envío de información al Bluetooth, se ha previsto el envío de un byte de control (0xAA). Con este byte, el receptor (Smartphone) puede identificar el inicio y el fin de nuevas tramas de datos, evitando el corrimiento de información. Este byte se manda cada 3 datos de ECG enviados. Así, cada trama de datos constará de 3 muestras de ECG (divididas en 3 bytes cada una), el byte de control en cabecera. Un total de 10 bytes.

El envío de los bytes se realiza mediante la función putc() con la etiqueta BT, de acuerdo a lo que se explicó en el apartado 2.3.1. Esto se ejecuta en la propia interrupción, una vez se comprueba que el Bluetooth sigue conectado al dispositivo remoto.

<u>SLEEP</u>

En el punto anterior se acaba de decir que los datos se envían al Bluetooth en la interrupción. Esto quiere decir que se enviarán los 3 bytes de cada dato inmediatamente después de ser leídos. De esta manera, en vez de almacenar los 10 bytes que conforman una trama y enviarlos de golpe, la trama se envía por partes, cada una en una interrupción.

Esto, que a priori no parece importante, posibilita que el microcontrolador se pueda poner en modo SLEEP entre una interrupción y la siguiente. Para explicarlo se debe tener en cuenta que el buffer de UART tiene capacidad para un máximo de 4 bytes. Por ello, si se ejecuta la orden putc() 4 o menos veces seguidas, los bytes a enviar se almacenan en el buffer y se envían rápidamente. Si, por el contrario, se quisiera enviar más de 4 bytes seguidos, estos no cabrían en el buffer y el tiempo de envío sería notablemente superior.

El ADS está configurado a 250 muestras por segundo, es decir, que llegarán nuevos datos cada 4 milisegundos. Es por ello que, una mayor velocidad de envío es fundamental para que el microcontrolador se pueda poner en modo bajo consumo (SLEEP) un tiempo significativo. De manera aproximada, el microcontrolador estará “dormido” un 75% del tiempo, aumentando la autonomía de la batería.

```
com:                              // Mientras no conecte con el Bluetooth
    while(!input(BT_Connect));     // del receptor (Smartphone) se espera aqui

loop:                             // Bucle principal: envia los datos sin parar
    sleep();                      // Para bajo consumo (Aprox. 3 ms de cada 4 ms)
    if(!input(BT_Connect)){       // Cuando termina la interrupción vuelve a testear
                            goto com;
                           };

goto loop; // Bucle principal
}
```

Ilustración 52. Recorte del programa del microcontrolador. Bucle principal

2.4. SOFTWARE DEL DISPOSITIVO MÓVIL

2.4.1. Introducción.

En los apartados precedentes, se ha descrito el diseño de la electrónica del proyecto, así como del programa que la controla. El resultado obtenido es un dispositivo capaz de captar la señal del electrocardiograma y enviarla mediante Bluetooth. Así, el único aspecto que queda para concluir el proyecto es el receptor de esta señal. Desde el principio del trabajo queda claro que este destinatario es el Smartphone del usuario; por ello, se desarrolla una aplicación para el sistema operativo Android que reciba la información y la procese de manera adecuada.

Para crear la aplicación, se utiliza el entorno de desarrollo Android Studio. En él, la programación se realiza en el lenguaje JAVA. Este lenguaje tiene una terminología que puede resultar áspera y complicada si no se está familiarizado. Así, se incorpora un glosario en el que se explican, de manera breve, los conceptos más importantes de la comunicación Bluetooth y del lenguaje en sí. De esta manera, se remite al apéndice A4 cada vez que se encuentre un término de la siguiente lista:

- Clase.
- Objeto.
- Constructor
- Hilo de ejecución
- Handler
- Hilo-Handler
- Looper
- Bluetooth Adapter
- Socket
- UUID
- InputStream, OutputStream
- BroadcastReceiver
- Dirección MAC…. Decir que esto es lo que se denomina address del device y que es lo necesario para establecer la conexión.
- Bundle
- Intent

Una vez dicho esto, se muestra un diagrama que ilustra el funcionamiento de la aplicación, de manera aproximada. Se recuerda que en el apéndice A1 se encuentra una leyenda de estos diagramas.

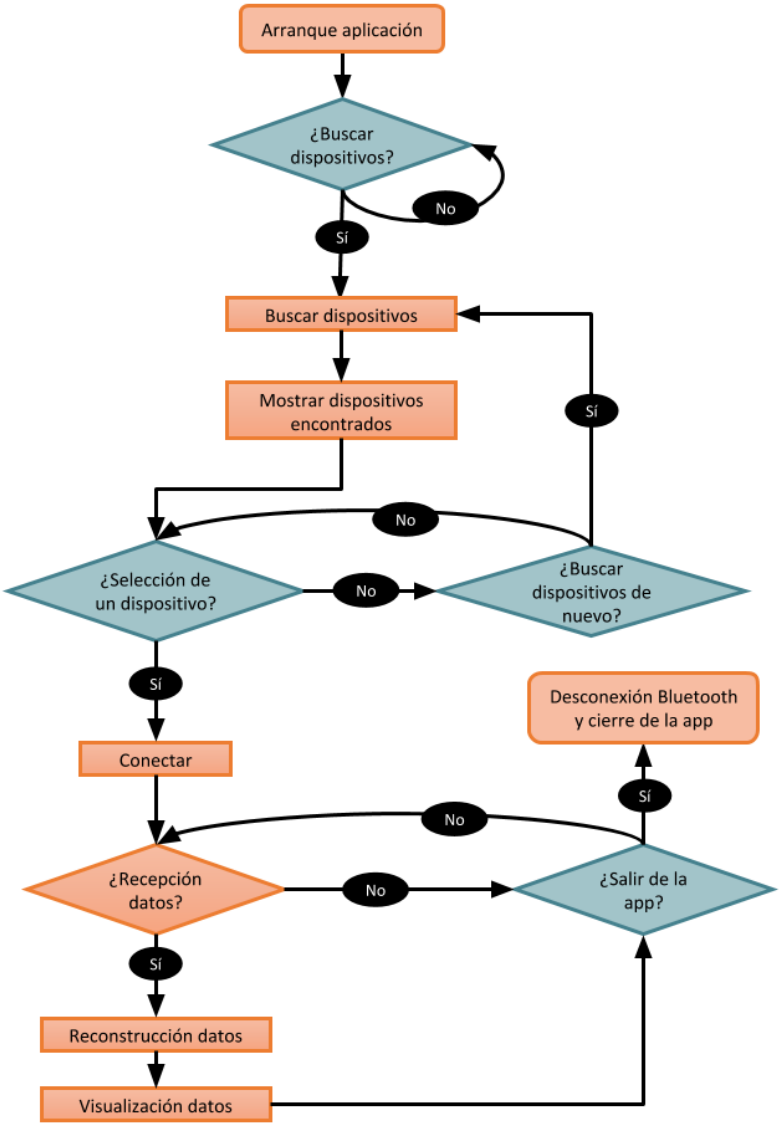


Ilustración 53. Diagrama de flujo de la aplicación. Destacar que es una aproximación al funcionamiento real: muestra los procesos que se deben llevar a cabo, como si los ejecutara un único programa de manera secuencial.

Aunque a priori pueda parecer sencillo, realizar de manera adecuada la recepción y procesado de la señal requiere estructuras de programación relativamente complejas. En la imagen precedente se muestra un diagrama de flujo que permite aproximarse a la solución. En él se sintetiza la estructura que tendría el código si se utilizase un proceso secuencial. Como se irá explicando a lo largo del capítulo, el proceso es ligeramente más complicado. Por un lado, la aplicación divide sus procesos en distintas pantallas o actividades. Por otro lado, muchas acciones se realizan de manera asíncrona, mediante hilos de ejecución diferentes.

Se puede observar en el diagrama que el teléfono es el encargado de administrar la conexión con el dispositivo electrónico. Esto es, la aplicación debe mostrar los dispositivos Bluetooth a la vista, para que el usuario indique activamente a cuál se quiere conectar. Para ello, Una vez seleccionado el

dispositivo, la aplicación se encarga automáticamente de la conexión, los cálculos y la visualización, dejando al usuario la única labor de salir de la actividad cuando lo desee.

La aplicación se estructura en actividades o pantallas, cada una de las cuales tiene asociada una función a realizar. De esta misma manera se organiza la descripción del programa en el presente trabajo. De acuerdo con el diagrama anterior, se parte de dos pantallas fundamentales. La primera se dedica a la administración de la conexión Bluetooth, mientras que la segunda realiza la monitorización y almacenamiento del electrocardiograma.

A diferencia de un sistema diseñado para centros de salud, una aplicación para teléfono móvil debe estar completamente enfocada al usuario. El destinatario de este producto es el público general, que no tiene por qué tener nociones médicas ni tecnológicas. El manejo debe ser fácil e intuitivo y, por si acaso, nunca está de más la inclusión de un dosier de instrucciones básicas. Asimismo, el diseño debe ser atractivo ya que, al final, el producto tiene que venderse.

Por todo ello, se ha considerado oportuno incluir otras dos pantallas para mejorar la comunicación entre aplicación y usuario. Una de ellas contiene un manual de instrucciones rápidas. La otra es, simplemente, una pantalla de inicio que comunica con las demás.

Con la descripción general hecha, se puede pasar a exponer de manera detallada el funcionamiento de cada actividad. Resulta muy ilustrativo comenzar con un esquema que muestre la relación entre las pantallas, así como el aspecto que tendrán.

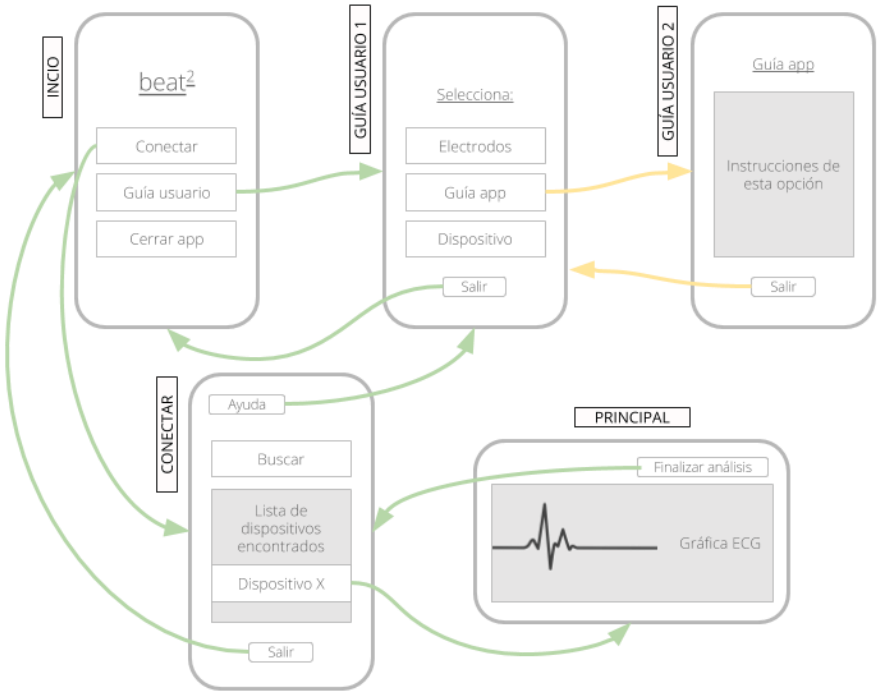


Ilustración 54. Esquema de las pantallas de la aplicación y sus relaciones.

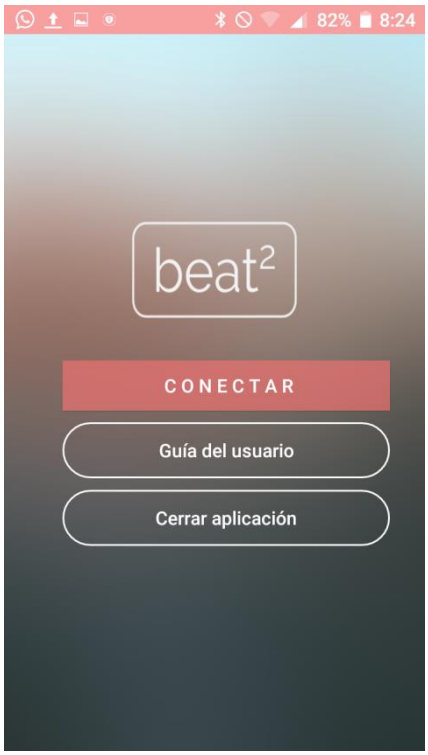
2.4.2. Pantalla de Inicio.

La pantalla de inicio es la primera que se abre cuando el usuario arranca la aplicación. Se trata de una actividad sin ninguna funcionalidad especial, cuyo único cometido es captar la atención del usuario y dirigirle hacia las otras actividades.

El objetivo es causar una buena primera impresión por parte del consumidor. Así, se ha puesto especial empeño en el diseño de esta interfaz, buscando transmitir profesionalidad al mismo tiempo que sencillez.

Se siguen los principios de diseño de Google: *Bold, graphic, intentional* (referencia [20]). El espacio, la pureza de líneas y el color son las características que destacan. Se juega con los contrastes: un fondo degradado, que aporta carácter, con botones e imágenes planos que equilibran la combinación; las curvas de los botones y las rectas de la gráfica; la dominancia de los rosas sobre otros elementos, en tonos neutros, para crear la jerarquía. La sensación final es de un layout despejado que permite al usuario centrarse en las utilidades, pero refrescante y dinámico. Diferente.

El diseño descrito se ha intentado tomar como imagen de marca, adaptándolo a las necesidades de cada pantalla. A continuación, se muestra el diseño planteado para la actividad. En la cabecera, aparece el nombre comercial pensado para el proyecto: $Beat^2$.



Captura 1. Pantalla de Inicio

En la imagen anterior se observa que hay tres botones principales. El primero de ellos comunica con la Pantalla Conexión, llevando a la aplicación por su camino natural. Es por ello que se ha destacado respecto a los demás. El segundo, lleva al usuario a la Guía del Usuario en la que se da una guía para usar el dispositivo y la aplicación. Por último, se tiene el botón que finaliza la aplicación.

2.4.3. Pantalla Guía del Usuario.

Ya se ha comentado que la aplicación está destinada al público general. Por ello, si se quiere llegar a la mayor cantidad de personas, no se puede presuponer ningún conocimiento previo por parte de los usuarios. Esta pantalla se ha diseñado para servir de ayuda tanto en aspectos relacionados con el dispositivo como en los relativos a la propia aplicación.

A esta actividad se puede acceder desde la Pantalla de Inicio y desde la Pantalla de Conexión. Cuando la actividad finaliza, se vuelve siempre a la Pantalla Inicio, sin importar desde dónde se había accedido.

Para esta actividad se han desarrollado dos diseños de pantalla diferentes que se intercambiarán mediante la interacción del usuario. El primero es al que se accede al pulsar el botón “Guía del Usuario” desde la pantalla de inicio. En él se visualizan las tres categorías de dudas que se han considerado más relevantes. Al pulsar cualquiera de los elementos se modifica la visibilidad de la actividad, cambiando el primer diseño por el segundo.

En el segundo diseño se intenta aclarar, de manera breve, el modo de empleo de la categoría seleccionada. Gráficamente, la pantalla consiste en una secuencia de vistas que se pueden desplazar de izquierda a derecha por medio de un botón. Estas vistas, realizadas mediante la herramienta FlipView, pueden contener texto, una imagen o ambas cosas.

Se han considerado tres categorías principales respecto a las que el consumidor puede tener dudas:

- Conexión de los electrodos al cuerpo
- Guía rápida APP
- Aspectos del dispositivo electrónico.

Captura 2. Pantalla Guía del Usuario

Para analizar el código utilizado, se remite al apéndice A4. Realmente, se ha creado una pantalla diferente para cada categoría. Sin embargo, en la descripción inicial del proyecto, este detalle no se ha

considerado de utilidad. Así, con el objetivo de simplificar la explicación, en el diagrama 33 se ha dibujado como una pantalla única.

Para mostrar las instrucciones que se dan en cada categoría, se exponen las siguientes capturas de cada categoría:

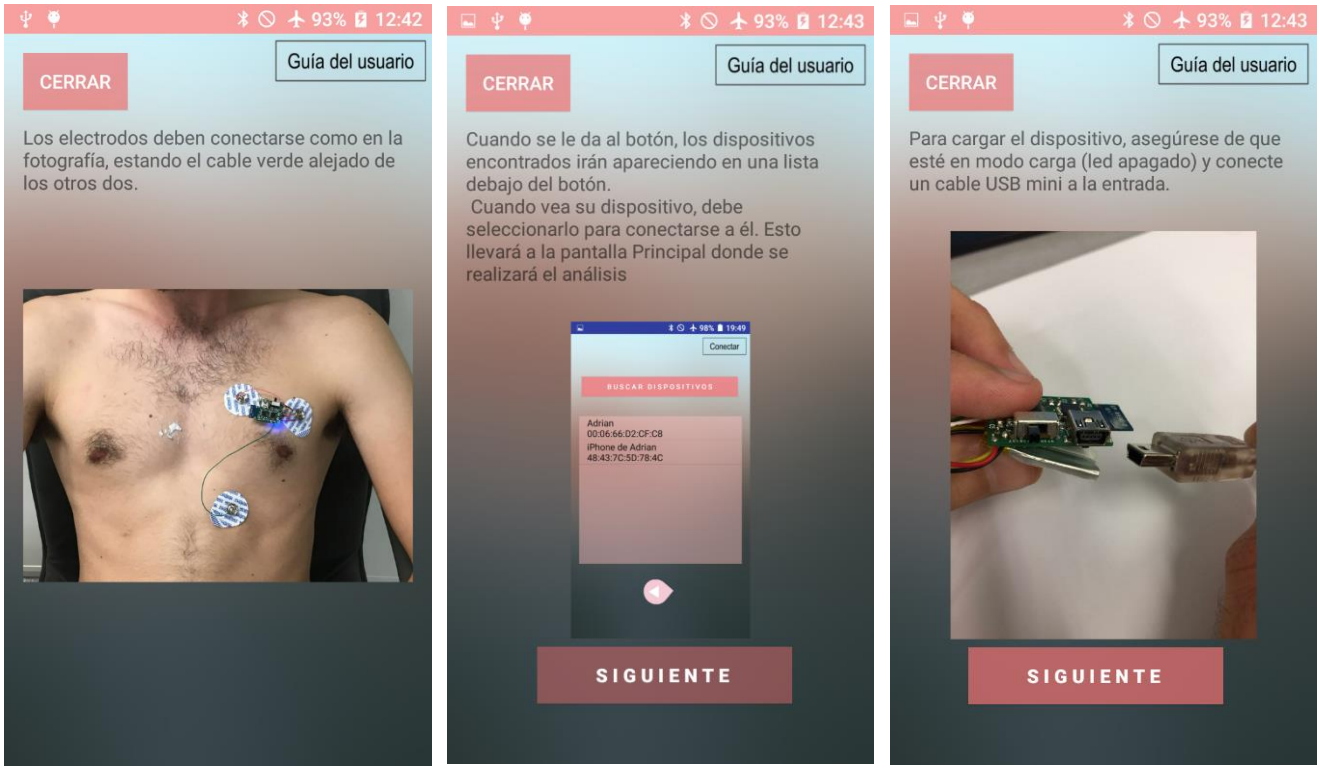


Ilustración 55. Capturas de la pantalla "Guía del Usuario". Ejemplos de instrucciones para los electrodos (izquierda), la aplicación (centro) y el dispositivo físico (derecha).

2.4.4. Pantalla Conectar.

Esta actividad realiza la función de enlace entre la Pantalla de Inicio y la Pantalla Principal, que es la que ejecuta la tarea fundamental de la aplicación. En este papel de intermediario, la aplicación debe administrar la búsqueda y selección del dispositivo electrónico $Beat^2$ con el que la aplicación se conectará.

El cometido básico es buscar los dispositivos Bluetooth que se encuentren próximos y visibles, presentarlos por pantalla al usuario y permitir que este seleccione uno de ellos para conectarse. Se trata, pues, de la primera parte del diagrama general (Ilustración 53). A continuación, se expone un diagrama de flujo que esboza el funcionamiento de la pantalla.

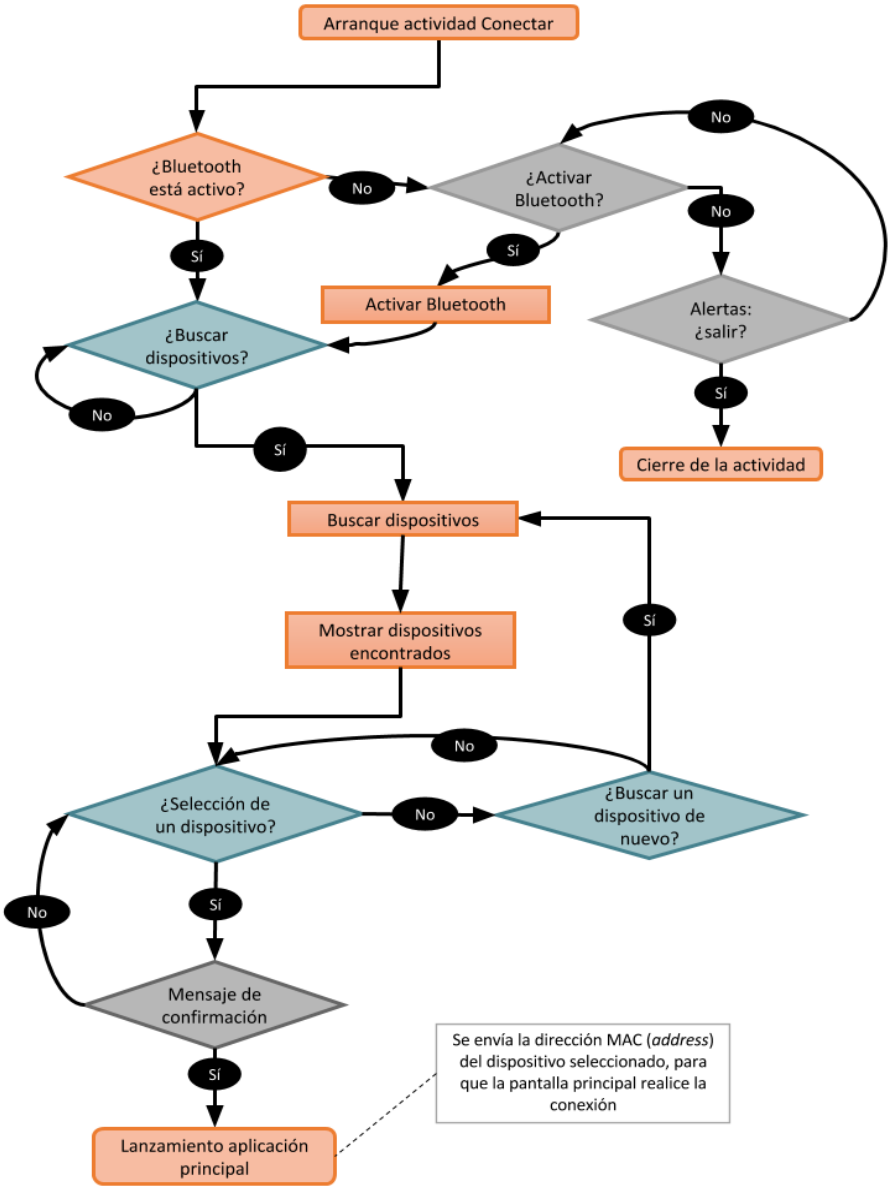
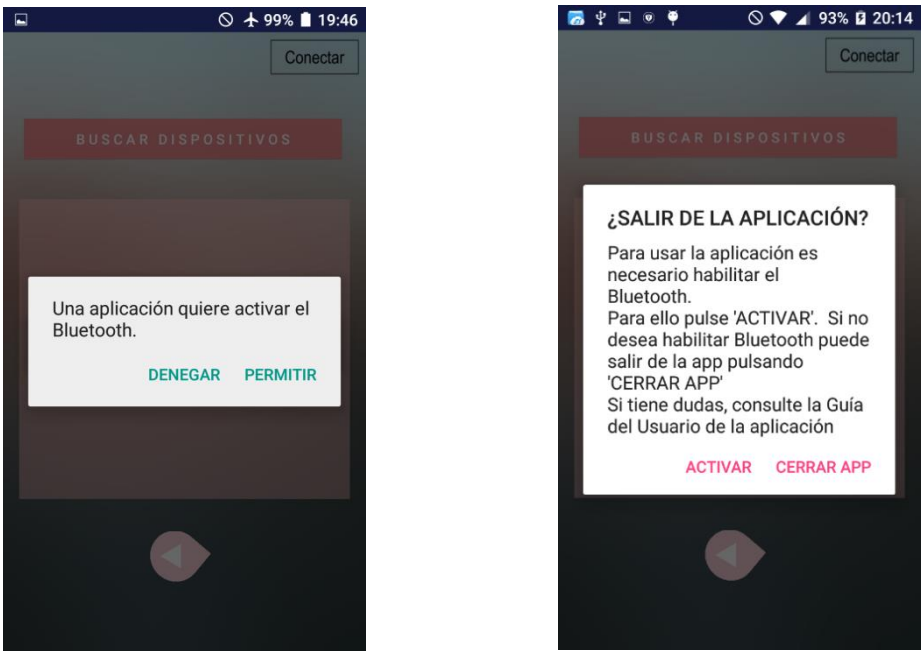


Ilustración 56. Diagrama de flujo de la Pantalla Conectar

Se trata de un extracto del precitado diagrama básico, con la diferencia de que, una vez seleccionado el dispositivo, se lanza la actividad Principal para que se encargue del resto de tareas. Este diagrama incorpora bloques que no se habían incluido en el diagrama básico al considerarlos innecesarios para la comprensión global de la app.

El primer aspecto que había sido obviado en el esquema inicial es la activación del Bluetooth. Como es lógico, la aplicación necesita hacer uso del Bluetooth del teléfono para comunicarse. En el caso de que el Bluetooth no esté activado ya, muestra un mensaje de alerta al usuario para que este lo active (o, mejor dicho, para que permita que la app lo haga). Si no acepta, se abre otro mensaje de alerta informando de que la aplicación no puede realizar su función sin Bluetooth. En este último mensaje,

se pregunta al usuario si prefiere abandonar la aplicación o activar el Bluetooth (con lo que se volvería a lanzar el primer mensaje de nuevo.

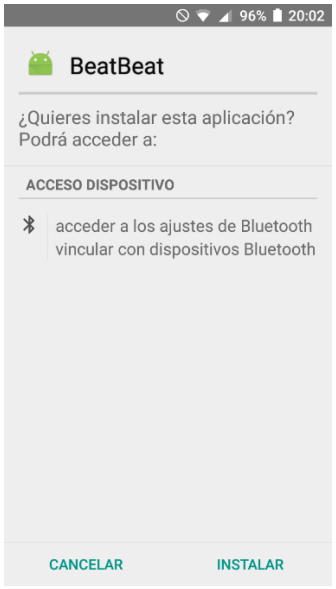


Captura 3. Mensajes de alerta de activación del Bluetooth

Por otro lado, es fundamental hacer hincapié en que la aplicación debe incluir una serie de permisos para activar o desactivar el Bluetooth, así como para conectar con otros dispositivos. Estos se incorporan en el manifiesto de la aplicación, tal y como se observa en la imagen que sigue. Cuando un usuario se descarga la app, se le informa de que esta tendrá acceso a los ajustes de Bluetooth.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.adrin.beatbeat">

    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

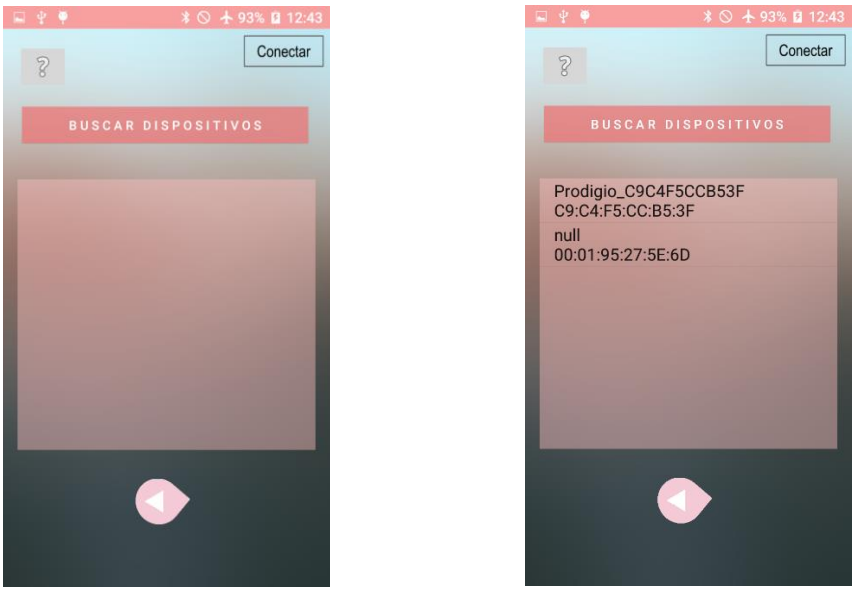
Ilustración 57. Recorte del código del manifiesto con los permisos necesarios para la aplicación.



Captura 4. Aviso de permisos al descargar la aplicación.

Como se verá en el próximo punto, el teléfono, que realiza de manera activa la conexión, es el denominado "MASTER"; mientras que el dispositivo electrónico, que simplemente acepta la conexión demandada, es el "SLAVE". Es por ello que el Bluetooth debe ser configurado, en el programa del microcontrolador, como "Slave" y con la visibilidad activada. De esta forma podrá ser detectado y conectado con la aplicación. Sin embargo, es innecesario realizar esos ajustes por código, ya que se trata de la configuración que el módulo Bluetooth trae por defecto.

Para realizar su función, la pantalla dispone de un botón central (BUSCAR), mediante el cual el usuario debe dar la orden de explorar en busca de dispositivos Bluetooth cercanos. Cuando los encuentra, estos se muestran al usuario en una lista debajo del botón. A continuación, se muestra una captura para ilustrar el diseño descrito.



Captura 5. Búsqueda de dispositivos Bluetooth.

En la imagen anterior, destaca un símbolo de interrogación en la esquina superior izquierda. Existe la posibilidad de que un usuario, inexperto en el manejo del aparato (o incluso del teléfono en sí), haya accedido a la pantalla presente sin leer las instrucciones de uso. Para esta situación se ha añadido el botón "?" que redirige al cliente desorientado a la Pantalla Guía del Usuario.

Es interesante aclarar el porqué del botón BUSCAR. Descubrir los dispositivos visibles es la función principal de la presente actividad, por lo que se podría realizar automáticamente en el arranque de la actividad. El usuario debe ser consciente de que es él quien tiene que encontrar el dispositivo $Beat^2$ desde la aplicación y conectarse, no al revés. Además, es posible que el usuario acceda a esta pantalla antes de encender el aparato o que, por alguna razón, este no se encuentre. En estos casos el usuario sí tendría que dar la orden expresa de explorar de nuevo.

El proceso de búsqueda puede tardar varios segundos. Cada vez que se pulsa el botón BUSCAR, se inicia de nuevo el proceso y se elimina el contenido de la lista. Para lidiar con usuarios impacientes, el botón BUSCAR se deshabilita hasta que la búsqueda que se estaba realizando finaliza. Esto se puede observar en el diagrama de flujo de detalle.

Una vez claro cómo funciona, a grandes rasgos, esta actividad, se puede entrar con un poco más de detalle en el proceso de detección y visualización de dispositivos Bluetooth. Para ello, se hace uso de otro diagrama de flujo que detalla este proceso, sustituyendo a las etapas de "Buscar Dispositivos" y de "Mostrar Dispositivos Encontrados" del diagrama de flujo de esta Pantalla.

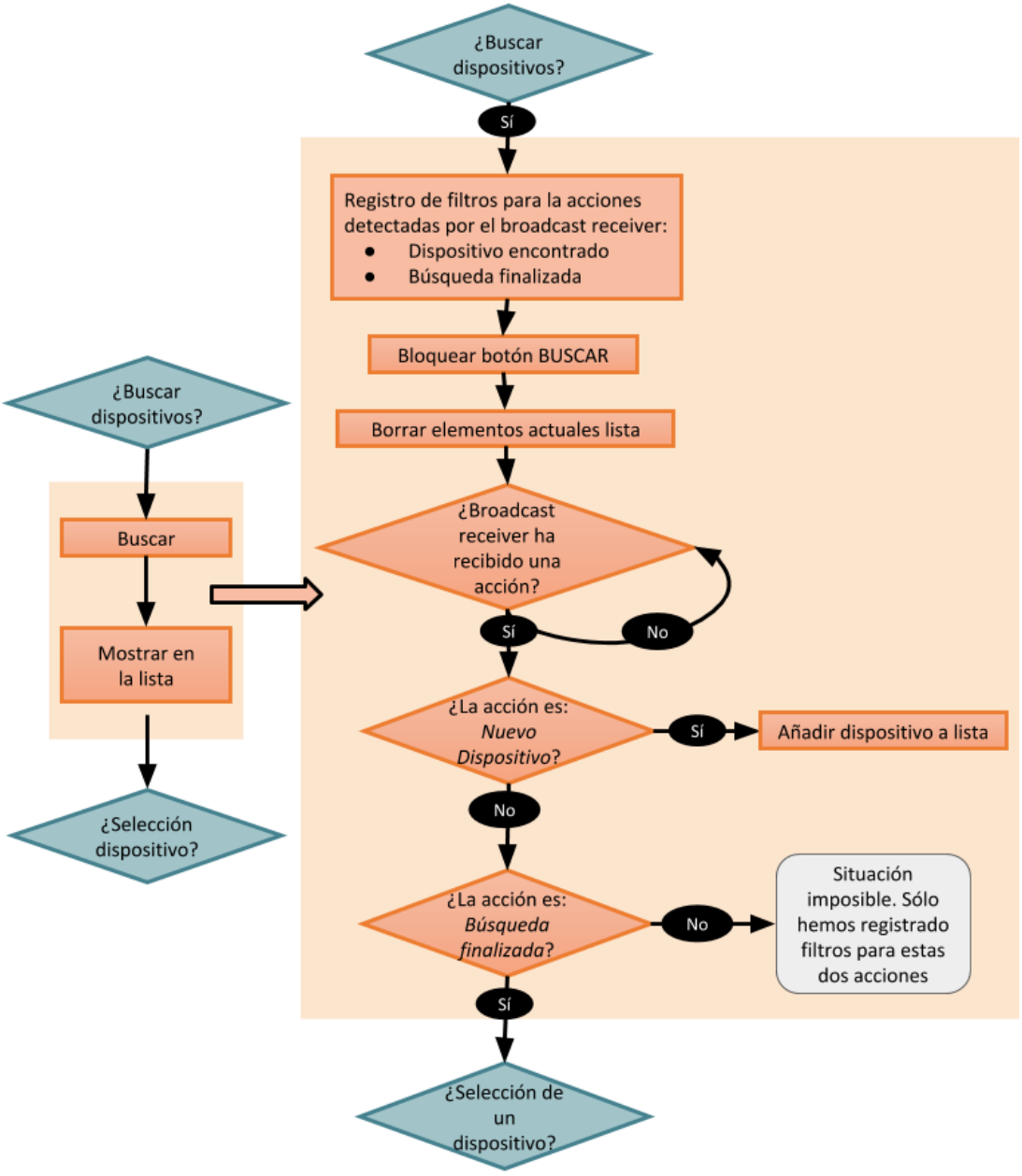
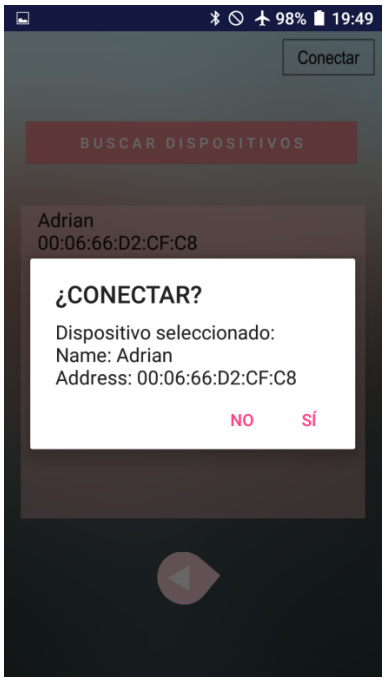


Ilustración 58. Diagrama de detalle del proceso de búsqueda de dispositivos.

En este diagrama, se introducen conceptos nuevos como el *Broadcast Receiver*, los *filtros*, las *acciones* etc. Estos elementos se encuentran descritos en el apéndice A2. Como se expone, el *Broadcast Receiver* es capaz de recibir y gestionar *Intents* relativos a eventos de la app o del sistema. Los Intentfilters se registran para que el *Broadcast Receiver* actúe solo ante los eventos de interés ("Registro de filtros para el *Broadcast Receiver*"). En el mencionado apéndice aparece un fragmento del código utilizado, para mostrar cómo se usan estas herramientas.

Finalmente, cuando la búsqueda ha concluido, todos los dispositivos encontrados se muestran en la lista a la espera de ser seleccionados. Cada elemento de la lista contiene el nombre y la dirección MAC (apéndice A2) correspondiente a un componente Bluetooth. Para conectarse, el usuario debe conocer el nombre del dispositivo electrónico y pulsar sobre él. Obviamente, en las instrucciones de la aplicación se puede localizar dicho nombre. Por si se equivoca, se abre un nuevo mensaje de alerta. En él se pide que confirme su intención de conectar con el elemento seleccionado.



Captura 6. Mensaje de confirmación.

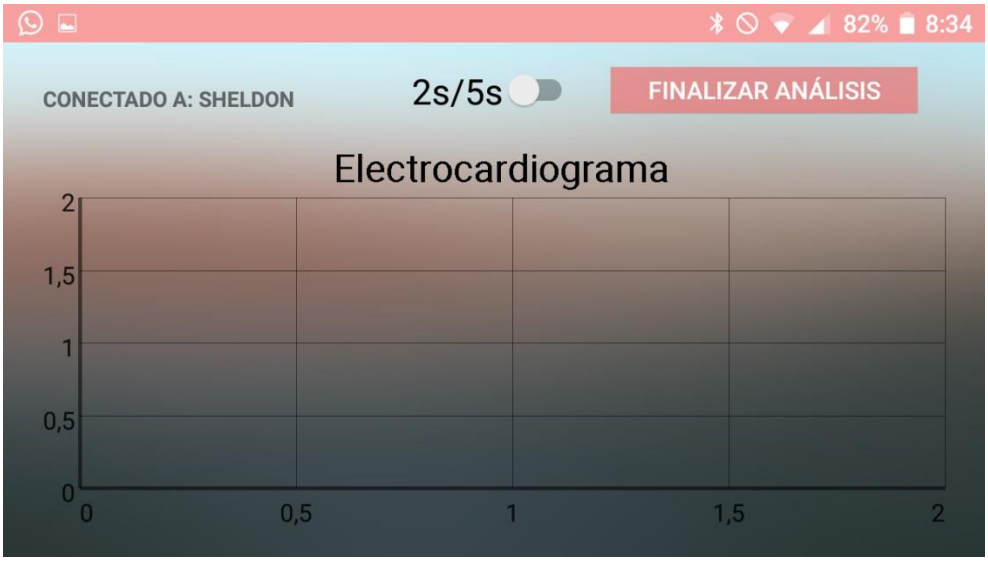
En el caso de confirmar, se lanza la actividad Principal, que es la etapa en la que verdaderamente conectan ambos aparatos. Para ello, es necesaria la *address* o *dirección MAC* del otro dispositivo, que se manda "como parámetro" al lanzar la actividad. Parámetro no es, siendo precisos, la terminología adecuada, pero es suficiente para dar una idea de lo que se pretende. La comunicación entre actividades se realiza mediante los *Bundles* (explicados en el apéndice A2).

2.4.5. Pantalla Principal.

La funcionalidad de la pantalla anterior se puede resumir en: permitir que el usuario busque y seleccione el dispositivo BeatBeat para conectarse. Una vez seleccionado, se lanza la presente actividad. En primer lugar, se realiza la conexión con el dispositivo; y, a continuación, se ejecutan las operaciones de visualización y almacenamiento. En esta pantalla es donde se prevé, en la continuación del proyecto, que se realicen los cálculos necesarios sobre la señal para la detección de episodios de fibrilación auricular.

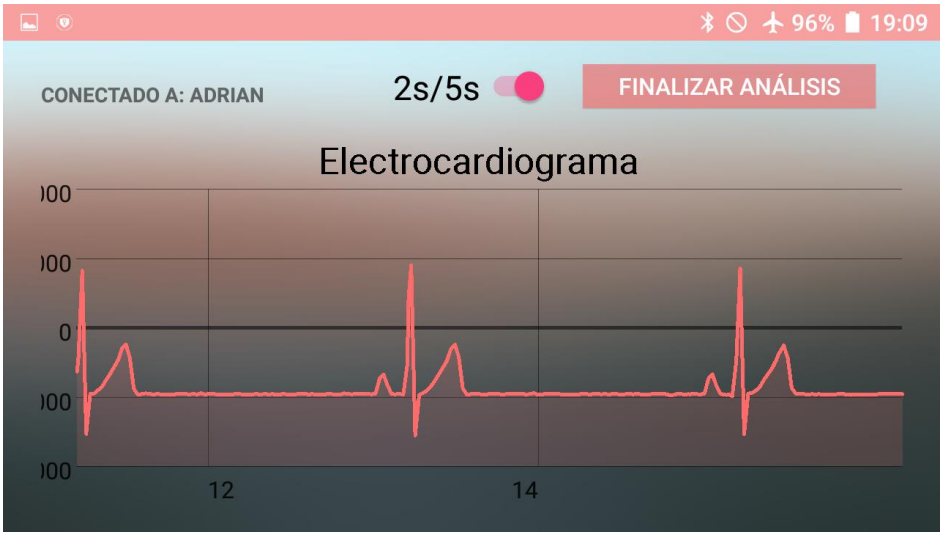
Para la visualización, se utiliza una librería especializada en gráficos de distinto tipo. Se trata de la librería GraphView, la cual no se encuentra por defecto en el compilador de Android. Así, se ha tenido que descargar desde la página oficial del desarrollador (referencia [25]).

Esta pantalla está pensada para tener una interacción mínima con el usuario, siendo su principal función la de mostrar el análisis en proceso. Por ello, a diferencia del resto de pantallas, se ha diseñado en formato apaisado. Con ello, la mayor parte de la pantalla está destinada a la gráfica del ECG. Cabe destacar que se ha bloqueado la rotación de la pantalla, de manera que siempre se encuentre en modo "landscape" independientemente de la posición del teléfono.



Captura 7. Pantalla principal.

Además de la gráfica, que ocupa la parte central de la pantalla, se observan otros elementos. En la parte superior se puede visualizar el nombre del dispositivo conectado. En la misma zona también se encuentran dos botones, uno para finalizar el análisis y volver a la Pantalla Conectar; y otro de tipo "Switch" que permite cambiar la cantidad de segundos visibles en la gráfica (hacer Zoom).



Captura 8. Imagen comparativa entre la pantalla con zoom y sin zoom.

Para realizar las tareas descritas hasta el momento, la aplicación debería seguir una secuencia como la mostrada en el siguiente diagrama.

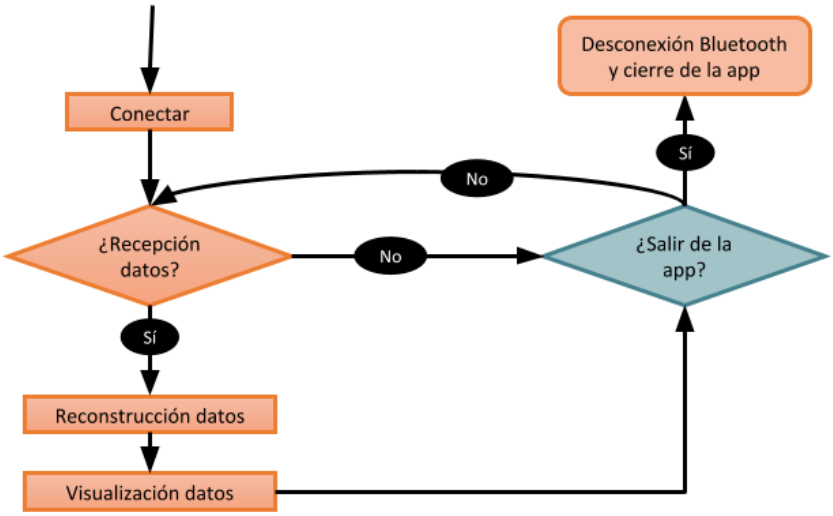


Ilustración 59. Diagrama de la Pantalla PRINCIPAL. Extracto de la Ilustración 53. Se trata de una aproximación a la solución real, que se encuentra en la Ilustración 60.

El diagrama de flujo correspondiente a esta pantalla será la mitad restante del diagrama inicial (Ilustración 53). En él se muestra una secuencia de acciones: Reconstrucción, Visualización, Cálculos y Actualización. Sin embargo, lo que en un primer momento se plantea de manera lineal, se realiza de manera simultánea en distintos hilos de ejecución paralelos. De esta forma, se evitan los posibles cuellos de botella que bloquearían (o ralentizarían) el hilo principal.

En este punto es de gran ayuda la lectura del apéndice A2, concretamente los puntos *Thread*, *Handler*, *HandlerThread* y *Looper*. En este proyecto se reciben datos a alta velocidad, por lo que atascarse en una de las tareas podría dificultar la recogida de los siguientes datos, algo inadmisible. Como se explica

en el apéndice, dividir las tareas en distintos hilos es fundamental para evitar el bloqueo del conjunto de la aplicación.

De esta manera, se utilizarán tres hilos de ejecución diferentes:

- Hilo principal. Sirve de interfaz con el usuario. Separándolo del resto se consigue que la aplicación responda a la interacción del usuario en cualquier momento. En principio, todas las instrucciones que modifiquen el aspecto de la pantalla se deben realizar a través de este hilo.
- Hilo Bluetooth. Este hilo "está pendiente" de los mensajes entrantes vía Bluetooth. Cuando se reciben datos nuevos, los reenvía a los hilos de procesado.
- Hilo Visualización. Recibe los datos del Hilo Bluetooth, reconstruye los puntos del ECG y los representa gráficamente.

Ahora sí se puede representar de manera correcta el funcionamiento de la aplicación.

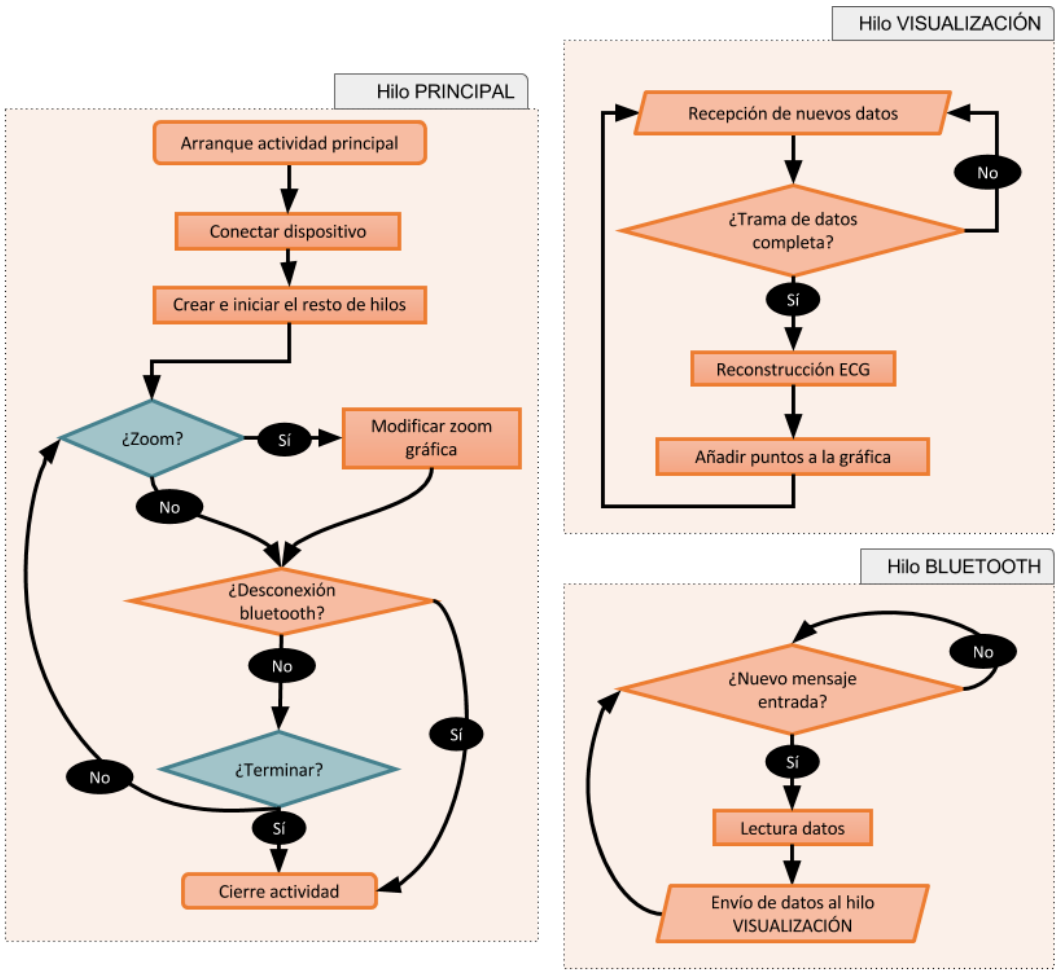


Ilustración 60. Diagrama de flujo real de la Pantalla Principal.

Para entender el diagrama es necesario aclarar la manera en la que los hilos transmiten la información. Por otro lado, en los siguientes sub-apartados se explican los hilos secundarios con un poco más de detalle.

En el apéndice se expone que los *handlers* son los elementos que permiten la comunicación entre hilos. Así, es necesario crear un *handler* para cada hilo que reciba información de otro.

El problema es ¿qué sucede con el hilo desde que termina sus operaciones con los datos recibidos hasta que llegan los siguientes? Dado que un *Thread* se destruye cuando su método *run()* finaliza, existen dos opciones: o mantener al hilo dentro del método *run()* comprobando si han llegado nuevos mensajes, o dejar que el hilo se destruya al acabar el método *run()* y que el *handler* lo cree de cero cada vez que lleguen nuevos mensajes.

Cualquiera de las dos opciones requiere un tiempo computacional para nada despreciable. En este punto entran en juego las clases *Looper* y *HandlerThread* explicadas en el mismo apéndice. En resumidas cuentas, con estas herramientas los hilos pueden estar inactivos esperando a que el *handler* reciba un nuevo mensaje.

Cabe destacar que esto solo es aplicable al Hilo Visualización. El Hilo Bluetooth no recibe los datos desde otros componentes de la aplicación, sino de la comunicación Bluetooth con el exterior. Es por ello que debe estar continuamente tratando de leer nuevos datos del *InputStream*.

Por otro lado, se ha pasado por alto un aspecto muy importante de la aplicación: la conexión con el dispositivo remoto. Ya se dijo que la Pantalla Conectar no realiza la conexión en sí, solo permite al usuario buscar y seleccionar el dispositivo. En esta actividad debe obtenerse la dirección MAC enviada desde la pantalla anterior (Conectar) por medio de la herramienta *Bundle* y establecer la conexión.

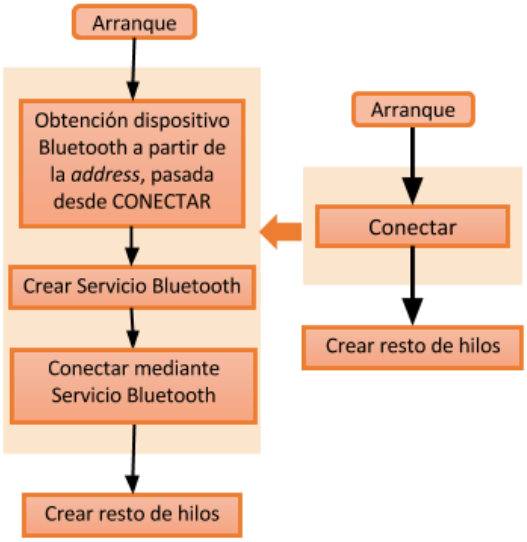


Ilustración 61. Detalle de la función Conectar del diagrama anterior.

Tanto el establecimiento como el manejo de la conexión Bluetooth son procesos relativamente complejos. Así, se ha creado una *clase* llamada Servicio Bluetooth encargada de administrar todos los aspectos relacionados con la comunicación inalámbrica. Estos aspectos incluyen la acción de conectarse, así como el Hilo Bluetooth introducido con anterioridad.

Es importante decir que la creación de esta clase no supone ninguna diferencia. La aplicación funcionaría exactamente igual si todas las funciones e hilos de la clase se hubieran creado en el

programa principal. Así, la creación de la clase responde a dos razones: la posibilidad de reutilizarla de manera independiente en futuros proyectos que requieran el uso de Bluetooth y que el código sea más limpio y organizado.

A. CLASE SERVICIO BLUETOOTH

Como ya se ha dicho, en esta clase se agrupan todos los procesos vinculados a la conexión Bluetooth. Para una mayor comprensión de este apartado y/o del código en sí (apéndice A4), se recomienda la lectura conjunta del apéndice A2. Concretamente los términos clase, objeto, constructor, socket, UUID, handler, thread, bluetoothDevice, etc.

El primer elemento destacable de la clase es el constructor. Con él se puede crear objetos de esta clase. Lo más importante de este método es el handler que se le pasa como parámetro. Este handler es al que se debe enviar los datos recibidos por Bluetooth, es decir, el handler del Hilo Visualización.

La clase contiene, a su vez, dos clases de tipo Thread. De esta manera, se pueden crear dos hilos de ejecución que realizarán las tareas relacionadas con la comunicación Bluetooth. El primer hilo se llama "Connect Thread" y se encarga de establecer la conexión remota. El segundo hilo es "Connected Thread" y administra dicha conexión.

Además de los hilos, la clase incorpora métodos (funciones) propios. Estos sirven de interfaz entre el programa principal y los hilos "Connect" y "Connected", así como entre ellos mismos. Los métodos son:

- Conectar(). Esta función sirve para crear un objeto de la clase "Connect Thread" e iniciarlo, es decir, genera el hilo para establecer la conexión. Para ello, recibe como parámetro un objeto de la clase BluetoothDevice y lo reenvía al constructor del hilo.
- EntreHilos(). Este método crea el hilo para administrar la conexión ("Connected Thread"). De manera análoga, recibe como parámetro un objeto Socket y lo reenvía al constructor del hilo.
- CerrarBluetooth(). Mediante esta función, se llama a los métodos "cancel()" de los hilos "Connect Thread" y "Connected Thread" respectivamente. Con ello, se destruyen los hilos finalizando la comunicación Bluetooth.

Con todo ello, es posible situar la ejecución de los hilos de la clase Servicio Bluetooth en el contexto global de la actividad Pantalla Principal.

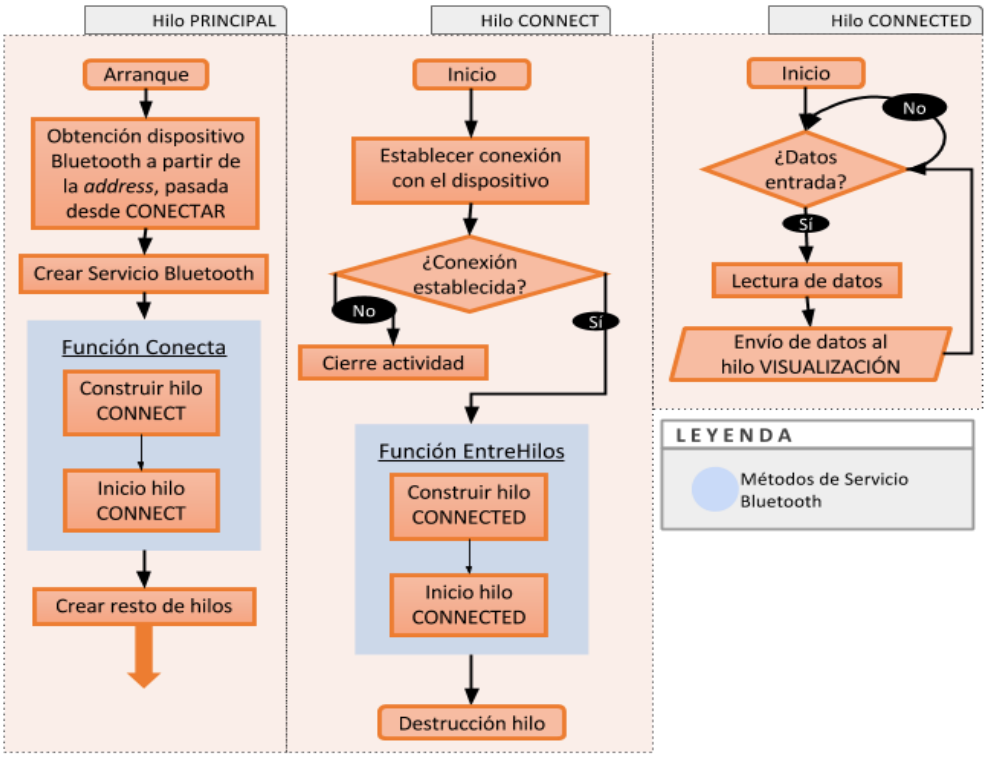


Ilustración 62. Diagrama de detalle de los hilos encargados de la conexión Bluetooth.

Toda la ejecución del hilo "Connect Thread" y la creación del "Connected Thread" se engloba en lo que, en la ilustración 60 era la acción "Conectar con el Dispositivo". Queda de manifiesto que el hilo "Connected Thread" es, precisamente, el Hilo Bluetooth incluido en el diagrama de la Pantalla Principal.

De manera un poco más técnica, el hilo "Connect Thread" se encarga de obtener el un socket a partir de la UUID del módulo Bluetooth y establecer la conexión del Socket. Cuando lo consigue, inicia el hilo "Connected Thread" pasándole el citado Socket. En la creación de este hilo se obtiene el InputStram y el OutputStream asociados al Socket y, una vez obtenidos, se mantiene a la espera de que lleguen nuevos datos al InputStream. Todos los términos de este párrafo se encuentran descritos en el anexo A2.

Una clase Servicio Bluetooth "de propósito general" debería poder conectarse en modo MASTER (el que demanda la conexión al dispositivo remoto) y en modo SLAVE (el que acepta la conexión demandada por el otro aparato). Sin embargo, esta aplicación está concebida para funcionar únicamente en modo MASTER y el hilo "Connect Thread" funciona de esa manera. Por ello, se ha obviado el posible hilo "Accept Thread" que permitiría funcionar en el otro modo.

Destacar el método write(); del hilo "Connected Thread". Este no se ha incluido en el diagrama ya que no se usa en la aplicación. No obstante, se incluye en el código por si se quisiera enviar información desde la aplicación al dispositivo en futuras versiones del proyecto.

B. HILO VISUALIZACIÓN

Este hilo se utiliza para reconstruir las muestras de la señal de ECG que envía el dispositivo BeatBeat mediante Bluetooth y añadirlas a la gráfica. Es conveniente recordar en este punto la trama de datos entrantes (apartado 2.3.2.). Los datos se envían en tramas de 10 bytes, siendo el primero un valor definido (0xAA=10101010). Este valor permite a la aplicación diferenciar una trama de datos de la siguiente.

Se recuerda que el Bluetooth envía conjuntos de bytes (8 bits), mientras que el ADS convierte los datos a 24 bits. Así, la trama de 10 bytes viene compuesta por 1 byte de inicio y 3 datos de ECG (9 bytes). La frecuencia de muestreo es de 250 SPS (4 ms por dato), por lo que una trama estará completa cada 12 ms.

En el mismo punto se indica que no se envían los 10 bytes de golpe. Cada vez que se muestrea un nuevo punto (4 ms) se envían los 3 bytes correspondientes, y una de cada 3 muestras envía además el byte 0xAA.

El encargado de recibir los datos en la aplicación es el Hilo Bluetooth. Este los almacena en un vector de bytes que envía al Hilo Visualización. Este hilo realiza lecturas del InputStream de manera asíncrona, por lo que es imposible saber cuántos bytes va a haber en la cola en el momento de la lectura. Podrían estar los 10, podría haber menos o incluso más.

De esta manera, se ha desarrollado una función relativamente compleja llamada “compruebaCódigo” dedicada a reagrupar los datos de entrada en las tramas de 10 bytes previstas. Esta función se corresponde con la etapa “¿TRAMA COMPLETA?” del diagrama de flujo de este hilo.

La reconstrucción de las tramas conlleva una dificultad, dado los bytes que contienen la información del electrocardiograma, pueden tener valor (incluyendo el byte de inicio). Con ello, se pueden producir errores muy puntuales. En el programa se ha apostado por la seguridad, de forma que, si se produjese un error como el descrito, simplemente se descartaría esa trama, en vez de añadir datos completamente erróneos.

Esta función, al igual que la etapa a la que corresponde, devuelve un TRUE o un FALSE en función de si tiene un vector de 10 bytes que comienza por 0xAA o no. En el caso de tener un vector incompleto, este se debe guardar en un vector auxiliar a la espera de ser completado en la siguiente recepción de datos.

En el caso de tener una trama completada, el programa sigue. Se reconstruyen los 3 datos de electrocardiograma a partir de los 9 bytes de información de la trama mediante la instrucción que se ve en el fragmento siguiente. A medida que se reconstruyen los datos, se van almacenando los datos en un objeto de la clase lista, preparados para ser procesados en un desarrollo posterior.

Es importante decir que la parte en la que se añaden puntos a la gráfica se realiza en el hilo Principal, que es el único que puede realizar cambios en la interfaz.

CAPÍTULO 3. CONCLUSIÓN

3.1. DISCUSIÓN DE RESULTADOS

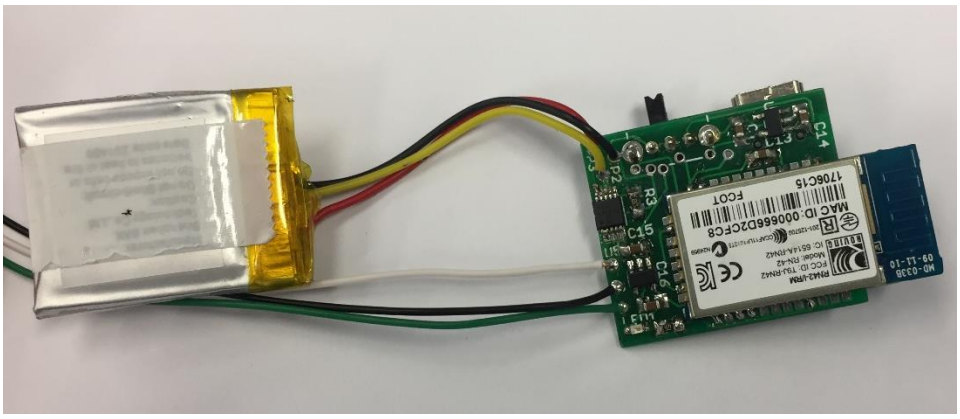
Al finalizar este trabajo se han obtenido dos productos: el dispositivo electrónico adquisidor de la señal de ECG, y la aplicación de Android para monitorizarla. En este apartado se va a hacer una valoración de los resultados obtenidos en ambos componentes.

Diseñar este dispositivo ha sido un trabajo satisfactorio, ya que el diseño ha concluido con un producto fabricado de acuerdo a los objetivos plantados y funcionando adecuadamente.

A esto hay que añadir que para llevarlo a cabo se ha tenido que aprender a manejar la herramienta de diseño Eagle. Asimismo, se ha tenido la experiencia real de los costes temporales que este tipo de proyectos conlleva: aprendizaje previo, diseño, fabricación y pruebas de validación.

Cabe destacar que el tiempo de fabricación siendo realizado por terceros no es despreciable (aproximadamente un mes) y conviene tenerlo presente para planificar la realización de otras tareas para ser más eficientes en el acotado tiempo de realización de un TFG.

Las siguientes fotografías muestran el resultado final.



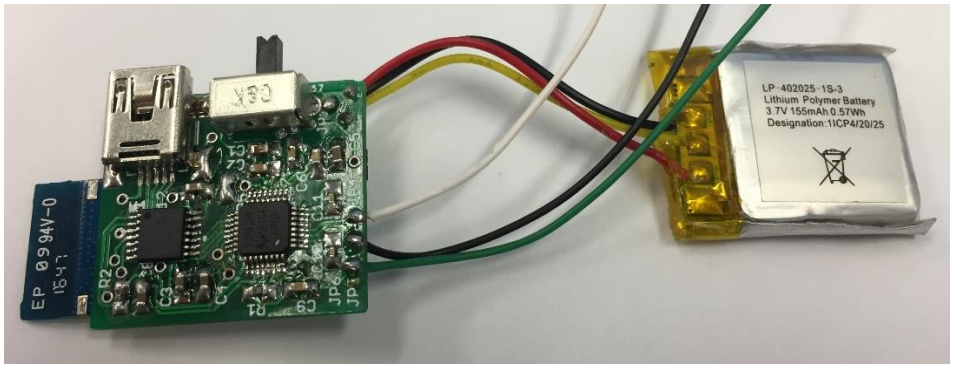


Ilustración 63. Imagen final de la placa (las dos capas) con los componentes soldados

Para la validación de la placa se ha utilizado un dispositivo proporcionado por los profesores, denominado paciente virtual. Este simula las constantes vitales de un paciente, en las mejores condiciones. El sistema tiene varias salidas, correspondientes a las derivaciones del corazón. A ellas, se conectan las entradas de la PCB. También tiene otro terminal correspondiente al electrodo RLD, por el cual se realimenta negativamente con la tensión de modo común.

Con este dispositivo, se consigue evitar los problemas de ruidos internos del cuerpo humano y los de la conexión de los electrodos a la piel. A continuación, se adjuntan unas imágenes ilustrando el funcionamiento.

Ilustración 64. Imagen del paciente virtual. Salidas para conectar los electrodos, opciones del aparato y, concretamente, la función de ECG.

Con el paciente virtual se puede probar el funcionamiento a diferentes frecuencias. A diferencia de un paciente real, la frecuencia es perfectamente constante. De esta manera, se puede evaluar de manera clara el funcionamiento del sistema.

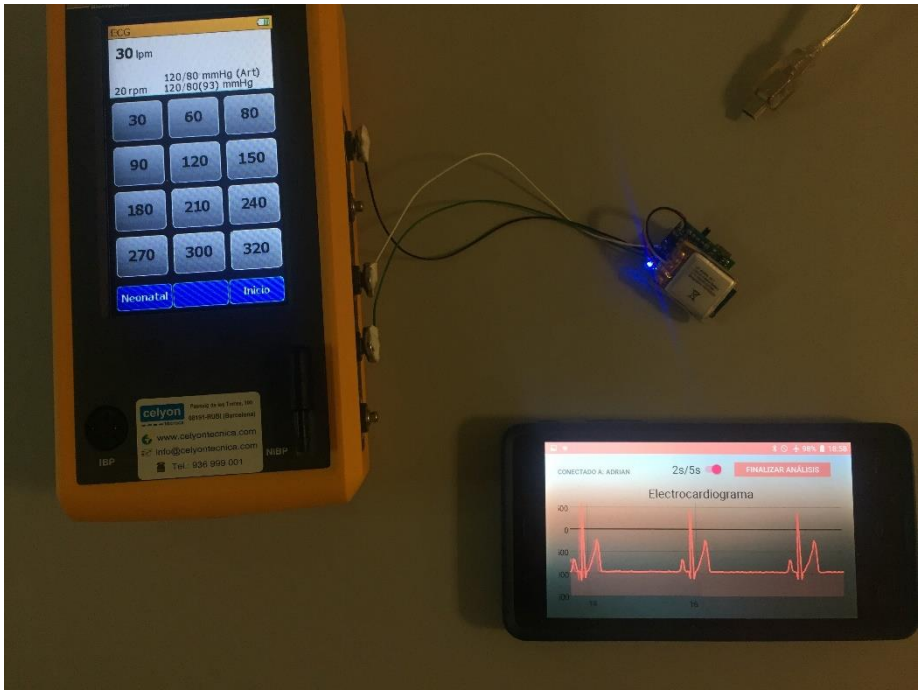


Ilustración 65. Funcionamiento del sistema a 30 latidos por minuto.

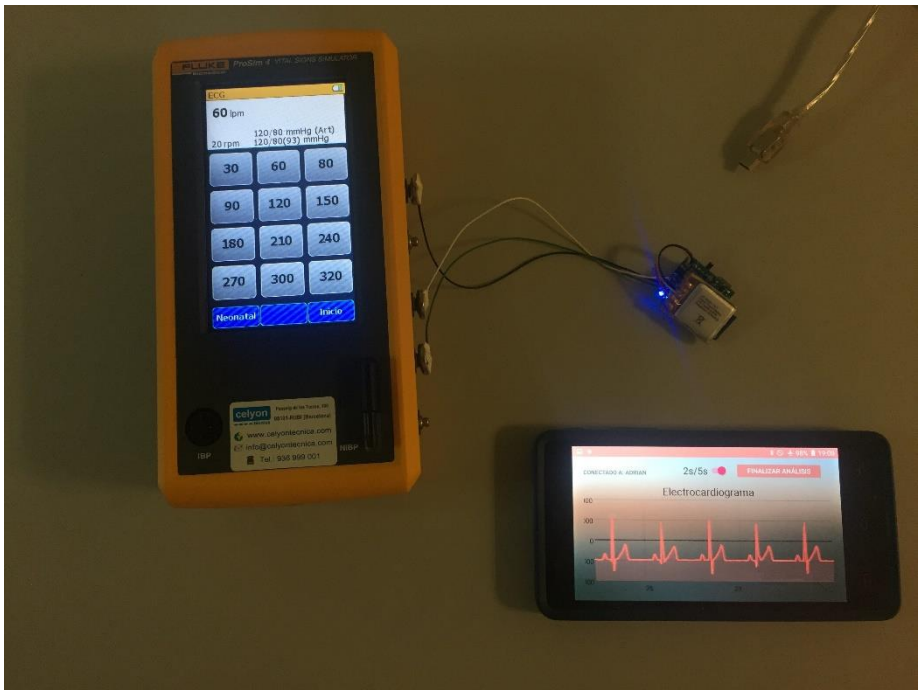


Ilustración 66. Funcionamiento del sistema a 60 latidos por minuto.

Para realizar los ensayos de validación de la placa y la aplicación, se ha conectado la entrada entre las derivaciones clásicas RA-LA ya que el paciente virtual no permite la configuración deseada en este proyecto (los dos electrodos, muy juntos sobre la zona de la aurícula). Es por ello que la forma de onda que se ve en las imágenes es la del electrocardiograma normal, introducido en los fundamentos teóricos.

La inversión en formación en aspectos de programación en Android ha sido muy importante. Se ha tenido que resolver aspectos de programación más avanzados de los que inicialmente se pensaba, y se han resuelto de forma satisfactoria, tal y como puede observarse en las diferentes fotografías que acompañan a estas líneas.

Todo ello teniendo en cuenta que no se pudieron realizar y validar los procesos de integración de la aplicación y del dispositivo electrónico hasta que éste estuvo fabricado, lo que se ha tenido que realizar en un periodo breve. De haber dispuesto de mayor tiempo, se habrían podido añadir otras ideas que han ido surgiendo durante el desarrollo y que se apuntan un poco más adelante como líneas futuras de trabajo

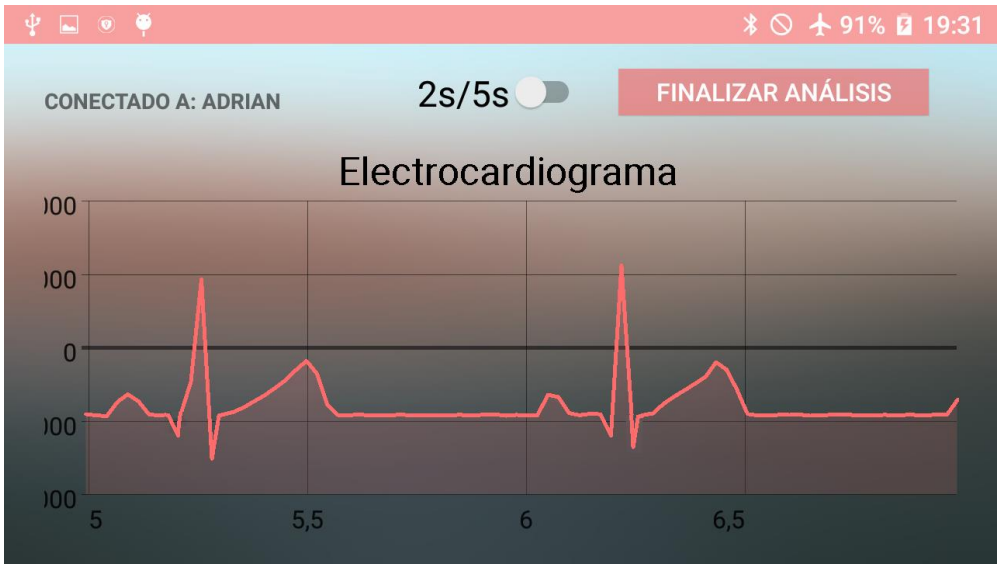


Ilustración 67. Captura de pantalla mostrando el funcionamiento de la aplicación.

En las imágenes que se han ido mostrando, se aprecia un correcto funcionamiento del sistema, tanto a nivel electrónico como a nivel de software. Sin embargo, se observa que la onda no es perfectamente limpia. Esto se debe, por una parte, a los posibles ruidos inherentes a la captación, conversión y envío de los datos. Por otra parte, ha habido algunas dificultades en cuanto al software, comentados en el apartado anterior. En una futura ampliación se debe trabajar para solucionarlos de la mejor manera posible. En este sentido, se plantean una serie de ideas en el apartado de crecimiento de la aplicación.

Una vez validado el funcionamiento del proyecto, en términos generales, se van a presentar los resultados que se obtienen al conectar los electrodos al paciente. En el inicio del documento se enfatizó que el sistema tiene un objetivo concreto y es la detección de fibrilación auricular. Para ello, los electrodos de medida se colocan justo encima de la aurícula, muy cerca el uno del otro, tal y como se muestra en las siguientes imágenes.

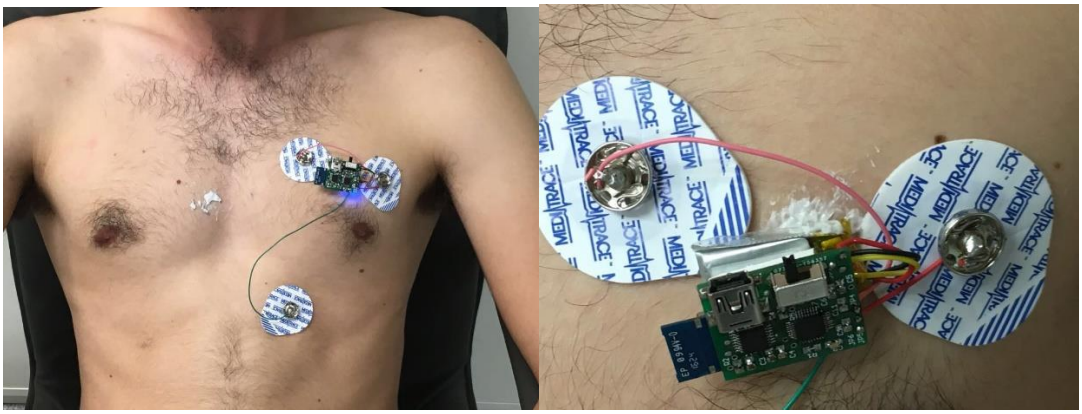


Ilustración 68. Imagen de la propuesta de colocación de los electrodos para centrarse en el movimiento de la aurícula.

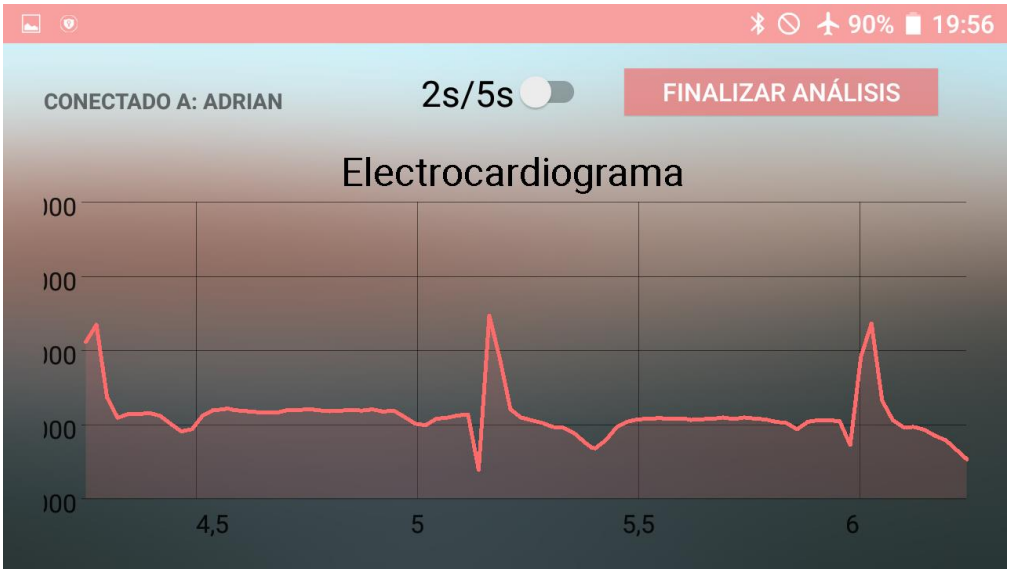


Ilustración 69. Muestra obtenida en el análisis con los electrodos sobre la aurícula

A continuación, se muestra la señal obtenida con los electrodos muy separados, para compararla con la disposición deseada en el proyecto.

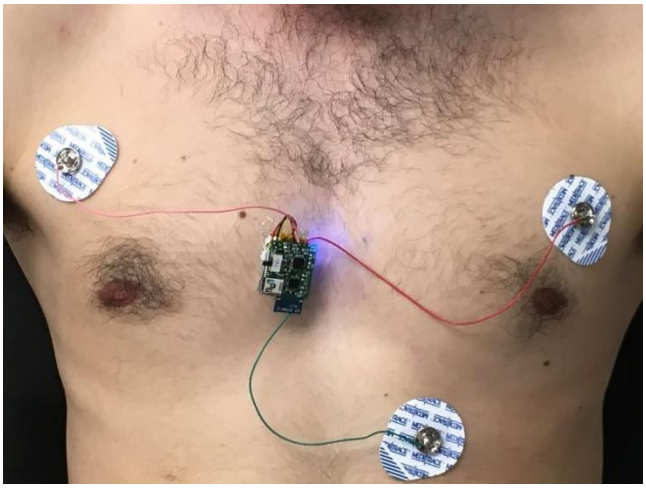


Ilustración 70. Situación de los electrodos para comparar

Ilustración 71. Muestra obtenida en el análisis con los electrodos separados

Esta disposición de los electrodos es muy diferente a las utilizadas clásicamente en el electrocardiograma, por lo que no se tiene información para comparar. Así, el análisis a nivel médico se reserva para especialistas en este campo en etapas posteriores del proyecto.

3.2. AMPLIACIONES FUTURAS DE LA APLICACIÓN

La aplicación descrita es un visualizador de electrocardiograma. Sin embargo, queda camino antes de conseguir que se pueda aplicar a la medicina de manera efectiva. Por ello, ya se dijo en el apartado de motivación que lo que se pretendía es crear un punto de partida a partir del cual seguir desarrollando. La idea principal es que el sistema incorporase algoritmos que permitieran la detección de episodios de fibrilación auricular.

Para ello, primero el sistema debe ser capaz de calcular la frecuencia cardíaca. Aunque esto se sale de los objetivos del proyecto, se han pensado varias alternativas para realizarlo. En primer lugar, se encuentra el algoritmo de Pan y Tompkins (referencia [27]), ampliamente utilizado desde hace más

de 30 años. Otra opción podría ser calcular la transformada de Fourier de la señal, para después quedarse con el armónico principal y ver a qué frecuencia corresponde.

Otra posible mejora sería la generación de un informe médico que se enviara por correo al finalizar los análisis. Además, de cara a hacer análisis de mucho tiempo, sería muy útil que tuviera la opción de funcionamiento en segundo plano, de manera que el usuario podría realizar cualquier otra actividad con su teléfono mientras tanto.

Por último, respecto a la aplicación actual, se podría realizar más testeos y pruebas, con otros Smartphones de diferentes prestaciones, ya que en este proyecto se ha trabajado únicamente con un modelo. Por otro lado, implementando herramientas de programación más avanzadas se podría obtener una mayor rapidez, de cara a teléfonos con prestaciones inferiores. Asimismo, con la incorporación de protocolos de comunicación más sofisticados, aumentaría la seguridad del sistema en cuanto a la transmisión de datos.

CAPÍTULO 4. REFERENCIAS

[1] Fuente de información: INE. Defunciones según la Causa de Muerte 2014.

[2] Fuente de información: INE. Encuesta de morbilidad hospitalaria 2000.

[3] http://secardiologia.es/multimedia/infografias/7256-la-enfermedad-cardiovascular-primera-causa-de-muerte

[4] https://www.xatakamovil.com/movil-y-sociedad/espana-territorio-smartphone

[5] *Bioelectronica. Señales Biomédicas*. Jose M. Ferrero y de Loma-Osorio, F. JavierSaiz Rodriguez, Antonio Arnau Vives. Servicio de publicaciones UPV (1994).

[6] *Neurofisiología aplicada al deporte*. Francisco Javier Calderón Montero, Julio César Legido Arce. Editorial Tébar, S. L. (2002).

[7] *Fisiología Humana. Un enfoque integrado*. Dee Unglaub Silverthorn. 4a edición.

[8] http://www.my-ekg.com/generalidades-ekg/ondas-electrocardiograma.html

[9] http://www.my-ekg.com/generalidades-ekg/derivaciones-cardiacas.html

[10] ADS1292R Data Sheet. Texas Instruments.

[11] Bluetooth RN42 Data Sheet. Microchip.

[12] Bluetooth Data Module Command Reference & Advanced Information User's Guide

[13] PIC24F04KA201 Family Data Sheet. Microchip

[14] PIC24F Family Reference Manual, Section 8. "Interrupts" (DS39707)

[15] PIC24F Family Reference Manual, Section 12. "I/O Ports withPeripheral Pin Select (PPS)" (DS39711).

[16] PIC24F Family Reference Manual, **Section 23. "Serial Peripheral Interface (SPI)"** (DS39699).

[17] PIC24F Family Reference Manual, **Section 21. "UART"** (DS39708).

[18] PIC24F Family Reference Manual, Section 4. "Program Memory" (DS39715).

[19] MCP73833/4 Data Sheet. Microchip

[20] KP-1608QBC-D Data Sheet. Kingbright

[21] TPS709 Data Sheet. Texas Instruments

[22] OS series Data Sheet. C&K

[22] https://developer.android.com/index.html?hl=es

[23] https://developer.android.com/guide/topics/connectivity/bluetooth.html?hl=es

[24] http://techtej.blogspot.com.es/2011/02/android-passing-data-between-main.html

[25] http://www.android-graphview.org/

[26] https://developer.android.com/reference/android/os/HandlerThread.html

[27] https://developer.android.com/reference/java/net/Socket.html

[28] https://developer.android.com/reference/android/content/Intent.html

[29] http://www.robots.ox.ac.uk/~gari/teaching/cdt/A3/readings/ECG/Pan+Tompkins.pdf
[30] https://material.io/guidelines/#introduction-principles
[31] http://www.plangeneralcontable.com/?tit=amortizacion-de-inmovilizado-metodo-lineal-o-de-cuotas-fijas&name=Manuales&fid=el0bcac

Presupuestos

PRESUPUESTO

1. OBJETIVO DEL DOCUMENTO

En la realidad industrial, llevar a cabo un proyecto no conlleva únicamente su desarrollo, sino que también es importante la correcta realización del mismo. El presente documento busca reflejar el coste o inversión que supondría la realización del trabajo.

En este trabajo se ha fabricado un único dispositivo. Sin embargo, la producción en serie tiene gran interés industrial dado las características del producto (dimensiones reducidas y bajo coste de los materiales). Por ello, además del presupuesto para el desarrollo de una placa individual, se ha incluido el presupuesto para una supuesta fabricación en serie de 1000 unidades.

2. PRESUPUESTO: UNA PLACA

2.1. Cuadros de precios

Cuadro de precios: Mano de obra	
Descripción del recurso	**Coste (€/hora)**
Ingeniero en Tecnologías Industriales recién titulado	12,00 €

Cuadro de precios: Materiales	
Descripción del recurso	**Coste (€/unidad)**
Placa fabricada por la empresa Itead.cc.	9,90 €
Circuito Integrado: ADS1292R (Texas Instruments)	5,58 €
Microcontrolador: PIC24F04KA200 (Microchip)	1,46 €
MóduloBluetooh: RN42-i/rm (Microchip)	12,98 €
Batería de Litio de 3,7 V. Designación: 1ICP4/20/25	10,36 €
Administrador de carga: MCP73833 (Microchip)	0,85 €
Regulador de voltaje: TPS709 (Texas Instruments)	1,03 €
Interruptor:	0,36 €
Resistencia:	0,49 €
Condensador:	0,28 €
Electrodo:	6,20 €
Cable:	0,02 €
Led:	0,11 €
Conector USB mini:	0,99 €

Para el cálculo del precio de los equipos se ha utilizado un modelo aproximado de amortización lineal. Para ello se ha calculado la diferencia entre el precio de mercado del producto y su valor residual al final de su vida útil. A continuación, se ha dividido el resultado entre la vida útil.

Para ello, el valor residual se ha supuesto 0 y la vida útil se ha obtenido del listado según el tipo de producto (referencia 31).

Cuadro de precios: Equipo	
Descripción del recurso	**Coste (€/hora)**
Ordenador VAIO Pro 13	0,02 €
Software Eagle (producto de Autodesk)	- €
Software Android Studio	- €
PCWHD: C-Aware IDE	0,01 €
Smartphone BQ Aquaris E5	0,01 €
RMSE-2C ESTACION DE REPARACION JBC BOMBA ELECTRICA	0,02 €
Paquete Office	- €

2.2. Cuadros de precios unitarios

Cuadro de Precios Unitarios		
Nº Unidad de obra	**Descripción de la unidad de obra**	**Coste unidad**
1	Diseño de la placa (esquema eléctrico y PCB) con el Software Eagle.	1.348,56 €
2	Montaje de la placa	133,17 €
3	Desarrollo y depuración del código del microcontrolador (Firmware)	306,78 €
4	Desarrollo y depuración de la aplicación con Android Studio	1.349,33 €
5	Desarrollo de los documentos del proyecto	612,98 €

2.3. Cuadros de precios descompuestos

Cuadro de Precios Descompuestos					
Nº Unidad	**Descripción**	**Unidades**	**Rendimiento**	**Precio**	**Importe**
1	**Diseño de la placa (esquema eléctrico y PCB) con el Software Eagle.**	u			
	Equipos				
	Ordenador VAIO pro 13	h	110	0,02 €	2,12 €
	Software de Eagle (producto de Autocad)	h	110	- €	- €
	Mano de Obra				
	Ingeniero en Tecnologías Industriales recién titulado	h	110	12,00 €	1.320,00 €
	Costes directos complementarios		0,02	1.322,12 €	26,44 €
			Coste Total:		1.348,56 €

2	**Montaje de la placa**	u			
	Equipos				
	RMSE-2C ESTACION DE REPARACION JBC BOMBA ELECTRICA	h	5	0,02 €	0,12 €
	Material				
	Placa fabricada por la empresa Itead.cc.	u	1	9,90 €	9,90 €
	Circuito Integrado ADS1292R (Texas Instruments)	u	1	5,58 €	5,58 €
	Microcontrolador PIC24F04KA200 (Microchip)	u	1	1,46 €	1,46 €
	MóduloBluetooh	u	1	12,98 €	12,98 €
	Batería de Litio de 3,7 V. Designación:	u	1	10,36 €	10,36 €
	Administrador de carga:	u	1	0,85 €	0,85 €
	Regulador de voltaje:	u	2	1,03 €	2,06 €
	Interruptor	u	1	0,36 €	0,36 €
	Resistencia	u	5	0,49 €	2,45 €
	Condensador	u	17	0,28 €	4,69 €
	Electrodo	u	3	6,20 €	18,60 €
	Cable	u	3	0,02 €	0,06 €
	Led	u	1	0,11 €	0,11 €
	Conector USB mini	u	1	0,99 €	0,99 €
	Mano de obra				
	Ingeniero en Tecnologías Industriales recién titulado	h	5	12,00 €	60,00 €
	Costes directos complementarios		0,02	130,56 €	2,61 €
			Coste Total:		133,17 €

3	**Desarrollo y depuración del código del microcontrolador (Firmware)**	u			
	Equipos				
	Ordenador VAIO	h	25	0,02 €	0,48 €
	PCWHD: C-Aware IDE	h	25	0,01 €	0,29 €
	Mano de obra				
	Ingeniero en Tecnologías Industriales recién titulado	h	25	12,00 €	300,00 €
	Costes directos complementarios		0,02	300,77 €	6,02 €
			Coste Total:		306,78 €

Nº Unidad	Descripción	Unidades	Rendimiento	Precio	Importe
4	**Desarrollo y depuración de la aplicación con Android Studio**	u			
	Equipos				
	Ordenador VAIO Pro 13	h	110	0,02 €	2,12 €
	Software Android Studio	h	110	- €	- €
	Smartphone BQ Aquaris	h	110	0,01 €	0,75 €
	Mano de obra				
	Ingeniero en Tecnologías Industriales recién titulado	h	110	12,00 €	1.320,00 €
	Costes directos complementarios		0,02	1.322,87 €	26,46 €
			Coste Total:		1.349,33 €

Nº Unidad	Descripción	Unidades	Rendimiento	Precio	Importe
5	**Desarrollo de los documentos del proyecto**	u			
	Equipos				
	Ordenador VAIO Pro 13	h	50	0,02 €	0,96 €
	Paquete Office	h	50	- €	- €
	Mano de obra				
	Ingeniero en Tecnologías Industriales recién titulado	h	50	12,00 €	600,00 €
	Costes directos complementarios		0,02	600,96 €	12,02 €
			Coste Total:		612,98 €

2.4. Cuadro de presupuestos parciales

Cuadro de presupuestos parciales					
Nº Capítulo	**Nombre del Capítulo**				**Importe capítulo**
1	Diseño Teórico				3.004,67 €
	Nº Unidad de Obra	Coste Unidad	Medición Unidades	Coste total unidades	
	1	1.348,56 €	1	1.348,56 €	
	3	306,78 €	1	306,78 €	
	4	1.349,33 €	1	1.349,33 €	
2	Fabricación				133,17 €
	Nº Unidad de Obra	Coste total Unidad	Medición Unidades	Coste total unidades	
	2	133,17 €	1	133,17 €	
3	Documentación				612,98 €
	Nº Unidad de Obra	Coste total Unidad	Medición Unidades	Coste total unidades	
	5	612,98 €	1	612,98 €	

2.5. Cuadro de presupuesto base de licitación

Caso A: Presupuesto de licitación para 1 placa

Nº Capítulo	Nombre Capítulo	Importe capítulo	Medición Capítulo	Importe total
1	Diseño Teórico	3.004,67 €	1,00	3.004,67 €
2	Fabricación	133,17 €	1,00	133,17 €
3	Documentación	612,98 €	1,00	612,98 €
	Presupuesto de Ejecución Material (PEM):			3.750,82 €
		Gastos generales (13%):		487,61 €
		Beneficio industrial (6%):		225,05 €
	Presupuesto de ejecución por contrata:			4.463,47 €
			IVA (21%):	937,33 €
	Presupuesto de Licitación:			**5.400,80 €**

El presupuesto de licitación asciende a un total de cinco mil cuatrocientos euros y veintiún céntimos.

3. PRESUPUESTO: MIL PLACAS

Si se presta atención a los cuadros anteriores, se observa que el precio de los componentes es relativamente pequeño. Por ello, el capítulo de fabricación tiene un precio menor respecto al de diseño y al de documentación. En el supuesto caso de realizar una fabricación serie, el diseño y la documentación solo sería necesario realizarlos una vez, repitiendo únicamente la etapa de fabricación. Con este sistema se obtendrían unos costes marginales por placa muy inferiores, maximizándose los posibles beneficios. Por esta razón se incluye el presente apartado en el documento, para dar una idea aproximada del presupuesto que supondría una producción masiva.

Hay componentes que, vendidos al por mayor, disminuyen en coste. A diferencia del resto de cuadros, el correspondiente a precios del material sufre cambios. También los sufren, en consecuencia, los cuadros de precios descompuestos del montaje de la placa, los unitarios, los parciales y el presupuesto base de licitación.

3.1. Cuadro de precios: Materiales

Cuadro de precios: Materiales	
Descripción del recurso	**Coste (€/unidad)**
Placa fabricada por la empresa Itead.cc.	9,90 €
Circuito Integrado: ADS1292R (Texas Instruments)	5,58 €
Microcontrolador: PIC24F04KA200 (Microchip)	1,06 €
MóduloBluetooh: RN42-i/rm (Microchip)	12,98 €
Batería de Litio de 3,7 V. Designación: 1ICP4/20/25	10,36 €
Administrador de carga: MCP73833 (Microchip)	0,63 €
Regulador de voltaje: TPS709 (Texas Instruments)	0,72 €
Interruptor	0,21 €
Resistencia	0,49 €
Condensador	0,28 €
Electrodo	6,20 €
Cable	0,02 €
Led	0,10 €
Conector USB mini	0,99 €

3.2. Cuadro de precios unitarios

Cuadro de Precios Unitarios		
Nº Unidad de obra	**Descripción de la unidad de obra**	**Coste unidad**
1	Diseño de la placa (esquema eléctrico y PCB) con el Software Eagle.	1.348,56 €
2	Montaje de la placa	131,74 €
3	Desarrollo y depuración del código del microcontrolador (Firmware)	306,78 €
4	Desarrollo y depuración de la aplicación con Android Studio	1.349,33 €
5	Desarrollo de los documentos del proyecto	612,98 €

3.3. Cuadro de precios descompuestos: Montaje de la placa

2	**Montaje de la placa**	u			
	Equipos				
	RMSE-2C ESTACION DE REPARACION JBC BOMBA ELECTRICA	h	5	0,02 €	0,12 €
	Material				
	Placa fabricada por la empresa Itead.cc.	u	1	9,90 €	9,90 €
	Circuito Integrado ADS1292R (Texas Instruments)	u	1	5,58 €	5,58 €
	Microcontrolador PIC24F04KA200 (Microchip)	u	1	1,06 €	1,06 €
	MóduloBluetooh	u	1	12,98 €	12,98 €
	Batería de Litio de 3,7 V. Designación:	u	1	10,36 €	10,36 €
	Administrador de carga:	u	1	0,63 €	0,63 €
	Regulador de voltaje:	u	2	0,72 €	1,44 €
	Interruptor	u	1	0,21 €	0,21 €
	Resistencia	u	5	0,49 €	2,45 €
	Condensador	u	17	0,28 €	4,69 €
	Electrodo	u	3	6,20 €	18,60 €
	Cable	u	3	0,02 €	0,06 €
	Led	u	1	0,10 €	0,10 €
	Conector USB mini	u	1	0,99 €	0,99 €
	Mano de obra				
	Ingeniero en Tecnologías Industriales recién titulado	h	5	12,00 €	60,00 €
	Costes directos complementarios		0,02	129,16 €	2,58 €
			Coste Total:		131,74 €

3.4. Cuadro de presupuestos parciales

Cuadro de presupuestos parciales					
Nº Capítulo	**Nombre del Capítulo**				**Importe capítulo**
1	Diseño Teórico				3.004,67 €
	Nº Unidad de Obra	Coste Unidad	Medición Unidades	Coste total unidades	
	1	1.348,56 €	1	1.348,56 €	
	3	306,78 €	1	306,78 €	
	4	1.349,33 €	1	1.349,33 €	
2	Fabricación				131,74 €
	Nº Unidad de Obra	Coste total Unidad	Medición Unidades	Coste total unidades	
	2	131,74 €	1	131,74 €	
3	Documentación				612,98 €
	Nº Unidad de Obra	Coste total Unidad	Medición Unidades	Coste total unidades	
	5	612,98 €	1	612,98 €	

3.5. Cuadro de presupuesto base de licitación

	Caso B: Presupuesto de licitación para 1000 placas			
Nº Capítulo	**Nombre Capítulo**	**Importe capítulo**	**Medición Capítulo**	**Importe total**
1	Diseño Teórico	3.004,67 €	1,00	3.004,67 €
2	Fabricación	131,74 €	1000,00	131.743,31 €
3	Documentación	612,98 €	1,00	612,98 €
	Presupuesto de Ejecución Material (PEM):			135.360,96 €
		Gastos generales (13%):		17.596,93 €
		Beneficio industrial (6%):		8.121,66 €
	Presupuesto de ejecución por contrata:			161.079,55 €
			IVA (21%):	33.826,70 €
	Presupuesto de Licitación:			**194.906,25 €**

El presupuesto de licitación asciende a un total de ciento noventa y cuatro mil setecientos treinta y tres euros con 10 céntimos.

Anexos

A1. LEYENDA PARA DIAGRAMAS DE FLUJO

En general, los diagramas se componen de 4 tipos de etapas diferentes. Su significado se explica en la siguiente imagen:

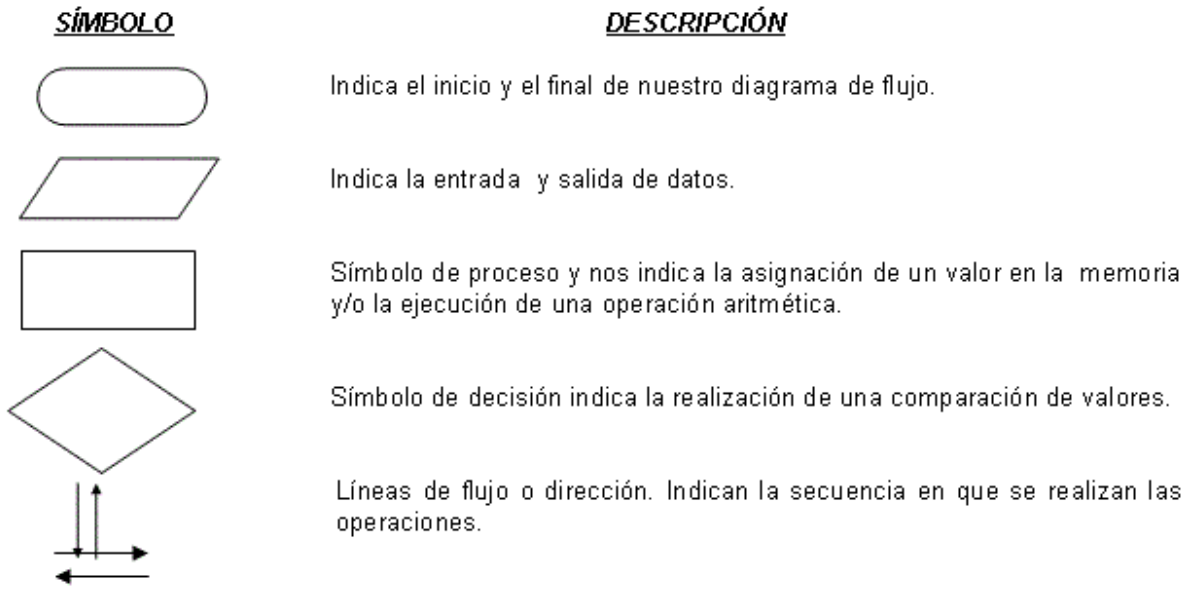


Ilustración 72. Leyenda para los diagramas de flujo del proyecto.

A diferencia del programa del microcontrolador, la ejecución de la aplicación está sujeta a las órdenes que da el usuario mediante botones o cualquier otra interfaz de la pantalla. De esta manera, habrá unos cuadros de decisión correspondientes a decisiones internas del programa y otros que se corresponden a la interfaz con el usuario. Para hacer más claros los esquemas, se ha utilizado un código de colores para diferenciar unos cuadros de decisión de otros. Los cuadros color rosado (el mismo color que el resto del diagrama), son decisiones internas del programa. Los cuadros color azul se corresponden con decisiones del usuario.

A2. GLOSARIO DE CONCEPTOS JAVA

- Clase
  Una clase es un ente abstracto que designa un tipo de objetos. Los objetos de esta clase tendrán una serie de propiedades (atributos) y una serie de operaciones o funciones (métodos).
- Objeto
  Un objeto es una instanciación de una clase. Es decir, un ente "real" almacenado en la memoria del programa sobre el cual se pueden realizar las operaciones o métodos. Por poner un ejemplo lo más sencillo posible, una clase podría ser "fruta" y, en el programa, se podrían declarar objetos de la clase fruta, de nombres: "naranja", "pera", etc.
- Constructor
  Es el método más importante de una clase. Con ella se pueden declarar (o construir) objetos de esa clase. Se trata de una función con el mismo nombre que la propia clase.
- Clase Thread (hilo de ejecución)
  Por defecto, todas las instrucciones y procesos de un programa se ejecutan en un mismo hilo (el hilo principal), una detrás de la otra. Al crear diferentes hilos de ejecución, la CPU divide el tiempo de cálculo entre ellos. De esta manera, se pueden ejecutar diferentes procesos en paralelo. Con ello, si un proceso es muy costoso, el resto de procesos no se quedan bloqueados esperando.
  Esto tiene mucha importancia en el desarrollo de aplicaciones, ya que bloquear el hilo principal, impide la interacción del usuario y da la sensación de que la aplicación "se ha colgado". Es importante decir que todas las instrucciones que modifiquen la pantalla, se deben realizar necesariamente a través del hilo principal.
- Handler
  Los objetos de la clase handler son el medio de comunicación entre hilos.
- handlerThread
  Se trata de una clase de tipo Thread que incluye un Looper (ver definición siguiente).
- Looper
  El objeto de la clase Looper de un handlerThread permite crear un handler asociado. De esta manera, se consigue un hilo de ejecución destinado al procesado de los mensajes que llegan al Handler.
- BluetoothAdapter
  Clase que permite administrar las acciones del Bluetooth (buscar dispositivos Bluetooth, instanciar objetos de la clase Bluetooth Device a partir de su dirección MAC, crear Sockets de tipo servidor, etc.).
- BluetoothDevice
  Es una clase para representar dispositivos Bluetooth a los que la aplicación se puede conectar.
- Address (Dirección MAC)
  La dirección MAC es el identificador de un dispositivo. Esta dirección es única para cada dispositivo y es el único dato que necesita la aplicación para realizar una conexión.

- Socket
  El término socket designa un concepto abstracto, mediante el cual dos sistemas pueden intercambiar datos de manera fiable. En Android, la clase con este nombre sirve para implementar un socket entre dos dispositivos.
- UUID
  La clase UUID representa un "Universal Inmutable Identifier" utilizado para la generación del socket entre la app y el dispositivo. En la hoja de datos [12] del Bluetooth aparece el UUID correspondiente al módulo Bluetooth utilizado.
- InputStream, OutputStream
  Son clases que designan de manera general cualquier flujo de entrada o salida de bytes.
- BroadcastReceiver
- Los objetos de la clase BroadcastReceiver permiten a la aplicación recibir los avisos del sistema o de la misma app, como por ejemplo "batería baja" o "dispositivo bluetooth encontrado". Para usarlo, deben registrarse filtros para tratar solo los anuncios deseados.
- Intent
  Los objetos de la clase Intent representan acciones que deben ser realizadas. Se utilizan para lanzar una pantalla desde otra o para enviar anuncios a los BroadcastReceivers.
- Bundle
  Mediante el Bundle es posible trnsmitir información desde una pantalla a otra en el momento del lanzamiento de la segunda.

A3. CÓDIGO MICROCONTROLADOR

```
#include <24F04KA200.h>
#device ICSP = 3
#use delay(internal=8000000)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#FUSES NOWDT       // No Watch Dog Timer
#FUSES CKSFSM      // Clock Switching is enabled, fail Safe clock monitor is enabled
#FUSES NOBROWNOUT  // No brownout reset
#FUSES BORV_LOW    // Brown-out Reset set to lowest voltage
#FUSES PUT         // Power-up Timer
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//                     PIN_A5  // ~RESET             ( 1) i
#define BT_Connect     PIN_A0  // BT conx establecida( 2) i
#define PWDN_ADS       PIN_A1  // Reset ADS1292      ( 3) o
//                     PIN_A2     U1TX                          ( 4) i
//                     PIN_A3     U1RX                   ( 5) o
//                     PIN_B4  // PGD3 - programador ( 6 i
//                     PIN_A4  // PGC3 - programador ( 7) i
//                     PIN_B8  // ADS PIN SCLK       ( 8) o
//                     PIN_B9  // ADS PIN DIN        ( 9) o
#define CS_ADS         PIN_A6  // ADS PIN CS         (10) o
//                     PIN_B14 // ADS PIN DOUT       (11) i
#define DRDY           PIN_B15 // ADS PIN ~DRDY Int0 (12) i
//                     VSS                           (13) -
//                     VDD                           (14) -
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#use rs232(xmit=PIN_A2, rcv=PIN_A3, baud=115200, stream=BT)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#word   PortA      =   0x02C2
#word   PortB      =   0x02C8
#byte   SPI1BUF    =   0x0248
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
void init(void){
  set_tris_a(0b0000000000110101);
// 15 / 14 / 13 / 12 / 11 / 10 /  9 /  8 /  7 /  6 /  5 /  4 /  // 3 /  2 /  1 /  0
  PortA   =  0b0000000000110101 ;
//  # -  # -  # -  # -  # -  # -  # -  # -  # -OUT -  # -Ain -//OUT - IN -PGD - CS
  set_tris_b(0b1100000000010000);
  PortB   =  0b1100000000010000 ;
// IN - IN -  # -  # -  # -  # -OUT -OUT -  # -  # -  # - IN -  // # -  # -  # -  #

  output_high(PWDN_ADS);
}
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //- - - - - - - - - - - - - -
char initBT(void){
               fprintf(BT,"$$$");           delay_ms(100);
               fprintf(BT,"SF,1\r"); delay_ms(150);
               fprintf(BT,"SA,0\r"); delay_ms(100);
               fprintf(BT,"SN,Adrian\r");    delay_ms(100);
               fprintf(BT,"R,1\r");  delay_ms(250);
               return 1;
}
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - // - - - - - - - - - - - - - -
// ADS1292R System Commands
#define START        0b00001000
// Start/restart (synchronize) conversions
#define RDATACT      0b00010000
// Enable Read Data Continuous mode (default mode at power-up)
#define SDATACT      0b00010001
// Stop Read Data Continuously mode
#define WREG0        0b01000000
// Write n nnnn registers starting at address r rrrr
// first byte 010r rrrr (2xh)(2) - second byte 000n nnnn(2)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - // - - - - - - - - - - - - - - -
```

```
char initADS(void){
                        setup_spi ( SPI_MASTER | SPI_SS_DISABLED | SPI_L_TO_H |
SPI_XMIT_H_TO_L | SPI_CLK_DIV_32);
                       // (8MHz/2)/32= 125k

                                output_high(CS_ADS); delay_ms(100);
                                output_low( 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(0b00000001);
//|SINGL_SHOT|    0      |   0      |    0      |    0     |  OSR2    //|  OSR1   |  OSR0    |
Continuous 250sps
/*CONFIG2   0x02*/spi_write(0b11100011);

//|    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(0b00010101);
//|   PD1    | GAIN1_2   | GAIN1_1  |  GAIN1_0  |  MUX1_3  | MUX1_2   //|  MUX1_1 | MUX1_0   |
CH1 Electrodes Gain=1
/*CH2SET    0x05*/spi_write(0b10010000);
//|   PD2    | GAIN2_2   | GAIN2_1  |  GAIN2_0  |  MUX2_3  | MUX2_2  |  MUX2_1 | MUX2_0   |
Channel 2 off (PD2=1)
/*RLD_SENS  0x06*/spi_write(0b00100011);//|  CHOP1   | CHOP0     | PD_RLD   |RLD_LOFFSNS|
RLDN2    | RLDP2    |  RLDN1  | RLDP1    |  RLD on (PD_RLD=1)
/*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(0b00101010);//|RSP_DM_EN | RSP_MD_EN |RESP_PH3  | RESP_PH2  |
RESP_PH1 | RESP_PH0 |    1    | REP_CTL  |  Resp off (DM_EN=0)
/*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(RDATACT);// RDATACT      0b00010000      // Enable
Read Data Continuous mode (default mode at power-up)
                            spi_write(START);  // START        0b00001000      //
Start/restart (synchronize) conversions

                            output_high(CS_ADS);
                            delay_ms(2);     // ret= 1 si todos correctos
                            return 0; // pero quiero devolver 0 si todo es correcto
}
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
unsigned int8 n, s2,s1,s0, d2,d1,d0;
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
void main(){
       delay_ms(100); // Tiempo de arranque
    init();         // Inicializa la configuracion de los pines del uC
       initBT();             // Inicializacion del modulo bluetooth
       initBT();             // Inicializacion del modulo bluetooth

    initADS();               // Inicializo el ADS1292R y lo pongo en marcha

                 ext_int_edge(0,H_TO_L);                 // Interrupcion ADS pin DRDY
```

```
                    clear_interrupt(INT_EXT0);       // Int0 = PIN 12 (RB15)
                       enable_interrupts(INT_EXT0  );
                    enable_interrupts(INTR_GLOBAL);

        initBT();             // Inicializacion del modulo bluetooth

com:                                                 // Mientras no conecte con el Bluetooth
        while(!input(BT_Connect));   // del receptor (Smartphone) se espera aqui

loop:                                                // Bucle principal: envia los datos sin
parar
        sleep();                                     // Para bajo consumo (Aprox. 3 ms de cada
4 ms)
        if(!input(BT_Connect)){              // Cuando termina la interrupción vuelve a testear
                                                goto com;
                                             };

goto loop; // Bucle principal
}
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#int_EXT0                    // Duracion de la rutina 0.9 .. 1.1 ms
void  EXT0_isr(void){
// Interrupcion cada vez que DRDY pasa a 0: 250 sps = 4 ms

      //Data Sheet del ADS: la señal en el pin CS debe estar a //nivel bajo durante toda la
      comunicación SPI
        output_low( CS_ADS);

 //Secuencia de datos de estado: 1100 LOFF_STAT[4:0] GPIO[1:0] 13 0's ==> Total: 24 bits
 // LOFF_STAT[4:0] y GPIO[1:0] son bits de dos registros de estado del ADS
// Primeros 8 bits: 1100 LOFF_STAT[4:1] = [1 1 0 0 RLD_STAT IN2N_OFF IN2P_OFF IN1N_OFF]
spi_write(0);s2= SPI1BUF;
// Segundos 8 bits: STAT[0] GPIO[1:0] cinco 0's= [IN1P_OFF GPIO1 GPIO0 0 0 0 0 0]
spi_write(0);s1= SPI1BUF;
spi_write(0);s0= SPI1BUF;   // Últimos 8 bits:  [0 0 0 0 0 0 0 0]

spi_write(0);d2= SPI1BUF;   // Datos del canal 1 (24 bits)
 spi_write(0);d1= SPI1BUF;   // Señal de Electrocardiograma
 spi_write(0);d0= SPI1BUF;

        spi_write(0);   spi_write(0);   spi_write(0);       // Datos del canal 2: no lo
almaceno

         delay_us(50);
        output_high(CS_ADS);

        if(!input(BT_Connect))
        {
                      if(n==0)  putc(0xAA,BT);      // Cabecera de la trama: cada 3 envios
                      putc(d2,BT);          // Dato de 24 bits  MSB
                      putc(d1,BT);
                      putc(d0,BT);          //LSB
                      n++;
                      if(n>2) n=0;    // Define el numero de datos por trama: 3 datos
        }

}// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - -
```

A4. CÓDIGO APLICACIÓN

a. Manifiesto de la app

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.adrin.beatbeat">

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".Inicio"
            android:configChanges="orientation|keyboardHidden|screenSize"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
            android:theme="@style/FullscreenTheme">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".GuiaDelUsuario"
            android:screenOrientation="portrait" />
        <activity
            android:name=".Conectar"
            android:screenOrientation="portrait" />
        <activity
            android:name=".Principal"
            android:screenOrientation="landscape" />
        <activity android:name=".GuiaUsuario2" />
        <activity android:name=".GuiaUsuarioDispositivo" />
        <activity android:name=".GuiaUsuarioApp"></activity>
    </application>

</manifest>
```

b. Pantalla Inicio

i. XML Inicio

```
<FrameLayout                          xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/b_beat"
    tools:context="com.example.adrin.beatbeat.Inicio">

    <!-- The primary full-screen view. This can be replaced with whatever view
         is needed to present your content, e.g. VideoView, SurfaceView,
         TextureView,                                etc.                    -->

    <!-- This FrameLayout insets its children based on system windows using
         android:fitsSystemWindows.                                          -->

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
```

```
        android:layout_marginTop="280dp">

        <Button
            android:text="       c       o       n       e       c       t       a       r"
            android:layout_height="wrap_content"
            android:id="@+id/btconectar"
            android:layout_width="280dp"
            android:layout_marginLeft="50dp"
            android:minHeight="43dp"
            android:padding="0dp"
            android:textColor="@android:color/background_light"
            android:background="@color/botonprincipal"                                    />

        <Button
            android:text="Guía                              del                              usuario"
            android:layout_width="280dp"
            android:layout_height="wrap_content"
            android:id="@+id/btInstrucciones"
            android:textAllCaps="false"
            android:layout_marginLeft="50dp"
            android:ellipsize="none"
            android:textColor="@android:color/background_light"
            android:background="@drawable/buttonprueba"
            android:layout_marginTop="10dp"                                               />

        <Button
            android:text="Cerrar                                                    aplicación"
            android:layout_width="280dp"
            android:layout_height="wrap_content"
            android:id="@+id/btCerrar"
            android:textAllCaps="false"
            android:layout_marginLeft="50dp"
            android:textColor="@android:color/background_light"
            android:background="@drawable/buttonprueba"
            android:layout_marginTop="10dp"                                               />
    </LinearLayout>

</FrameLayout>
```

ii. Código Java Inicio

```
package com.example.adrin.beatbeat;

import android.annotation.SuppressLint;
import android.content.Intent;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.os.Handler;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Button;


public class Inicio extends AppCompatActivity {

    final int NO_MENTERAO=1000;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_inicio);

        final Button btconectar= (Button)findViewById(R.id.btconectar);
        final Button btInstrucciones= (Button)findViewById(R.id.btInstrucciones);
        final Button btCerrar= (Button)findViewById(R.id.btCerrar);



        btconectar.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                Intent pasarAPrincipal= new Intent(Inicio.this, Conectar.class);
                startActivityForResult(pasarAPrincipal, NO_MENTERAO);
            }
        });

        btInstrucciones.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent pasarAGuia= new Intent(Inicio.this, GuiaDelUsuario.class);
                startActivity(pasarAGuia);
            }
        });

        btCerrar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                finish();
            }
        });
    }

    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode==NO_MENTERAO){
            //Es el resultado de la actividad conectar en el caso de negarse a habilitar el
Bluetooth
            if (resultCode == RESULT_OK){
                finish();
            }
        }
    }


}
```

c. Pantalla guía del usuario

i. XML Guía del Usuario

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_guia_del_usuario"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.adrin.beatbeat.GuiaDelUsuario"
    android:background="@drawable/b_gdu">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="400dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:weightSum="1"
        android:textAlignment="center"
        android:layout_alignParentTop="true">

        <TextView
            android:text="e s c o g e  u n a  o p c i ó n"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/textView"
            android:layout_weight="0.05"
            android:textAlignment="center"
```

```
            android:textColor="@android:color/white"
            android:textSize="14sp"
            android:textAllCaps="true"
            android:layout_marginTop="120dp" />

        <Button
            android:text="Conexión de electrodos"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/btElectrodos"
            android:layout_weight="0.05"
            android:layout_marginTop="20dp"
            android:textAllCaps="false"
            android:textSize="14sp"
            android:background="@drawable/buttonprueba"
            android:textColor="@android:color/white" />

        <Button
            android:text="Guía rápida de la app"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/btApp"
            android:layout_weight="0.05"
            android:textAllCaps="false"
            android:textSize="14sp"
            android:textColor="@android:color/white"
            android:background="@drawable/buttonprueba"
            android:layout_marginTop="10dp" />

        <Button
            android:text="Aspectos del dispositivo físico"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/btDispositivo"
            android:layout_weight="0.05"
            android:textAllCaps="false"
            android:textSize="14sp"
            android:background="@drawable/buttonprueba"
            android:textColor="@android:color/white"
            android:layout_marginTop="10dp" />

        <ImageButton
            android:id="@+id/btSalirInstrucciones"
            android:layout_weight="0.05"
            android:background="@drawable/salir1"
            android:scaleType="centerInside"
            android:layout_marginTop="80dp"
            android:layout_marginLeft="140dp"
            android:layout_height="50dp"
            android:layout_width="60dp" />

    </LinearLayout>

</RelativeLayout>
```

ii. Código Java Guía del Usuario

```
package com.example.adrin.beatbeat;

import android.bluetooth.BluetoothAdapter;
import android.content.DialogInterface;
import android.content.Intent;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageButton;

public class GuiaDelUsuario extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_guia_del_usuario);

        final ImageButton btCerrarInstruc= (ImageButton)
findViewById(R.id.btSalirInstrucciones);
        final Button btElectrodos= (Button) findViewById(R.id.btElectrodos);
        final Button btApp= (Button) findViewById(R.id.btApp);
        final Button btDisp= (Button) findViewById(R.id.btDispositivo);

        btCerrarInstruc.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                finish();
            }
        });

        btElectrodos.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intentelect= new Intent(GuiaDelUsuario.this, GuiaUsuario2.class);
                startActivity(intentelect);
            }
        });
        btApp.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intentapp= new Intent(GuiaDelUsuario.this, GuiaUsuarioApp.class);
                startActivity(intentapp);
            }
        });
        btDisp.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intentDisp= new Intent(GuiaDelUsuario.this,
GuiaUsuarioDispositivo.class);
                startActivity(intentDisp);
            }
        });
    }

}
```

d. Pantalla guía del usuario 2

i. XML Guía del usuario 2

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_guia_usuario2"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.adrin.beatbeat.GuiaUsuario2"
    android:background="@drawable/b_gdu">

        <!-- ViewSwitcher with three views first is ImageView, second is a layout in which we
have two TextView's
         and third is a layout in which we have two Button's -->
        <ViewFlipper
            android:id="@+id/simpleViewFlipper"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <ImageView
                android:layout_width="wrap_content"
```

```
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:src="@drawable/salir1" />

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:orientation="vertical">

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="First TextView" />

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_marginTop="20dp"
                    android:text="Second TextView" />


            </LinearLayout>

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:orientation="vertical">


                <Button
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="First Button" />

                <Button
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_marginTop="20dp"
                    android:text="Second Button" />


            </LinearLayout>
        </ViewFlipper>


        <Button
            android:id="@+id/buttonNext"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_marginTop="480dp"
            android:background="@color/botonprincipal"
            android:text="s i g u i e n t e "
            android:textColor="#fff"
            android:textStyle="bold"
            android:layout_marginLeft="50dp"
            android:layout_width="215dp" />

    <Button
        android:text="Cerrar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/btSalirGDU2" />


</RelativeLayout>
```

ii. Código Java Guía del usuario 2

```
package com.example.adrin.beatbeat;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.ViewFlipper;

public class GuiaUsuario2 extends AppCompatActivity {

    private ViewFlipper simpleViewFlipper;
    Button btnNext;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_guia_usuario2);


        // get The references of Button and ViewFlipper
        btnNext = (Button) findViewById(R.id.buttonNext);
        simpleViewFlipper = (ViewFlipper) findViewById(R.id.simpleViewFlipper);
          // obtener la referencia ViewFlipper
        // declarar in and out animations
        Animation in = AnimationUtils.loadAnimation(this, android.R.anim.slide_in_left);
        Animation out = AnimationUtils.loadAnimation(this, android.R.anim.slide_out_right);


        simpleViewFlipper.setInAnimation(in);
        simpleViewFlipper.setOutAnimation(out);

        // ClickListener para el boton siguiente
        // al clicar se cambian las vistas
        btnNext.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {
                 simpleViewFlipper.showNext();
            }
        });
    }
}
```

e. Pantalla conectar

i. XML Conectar

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_conectar"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.adrin.beatbeat.Conectar"
    android:background="@drawable/b_conectar">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <Button
            android:text="b u s c a r   d i s p o s i t i v o s"
```

```
            android:layout_height="40dp"
            android:id="@+id/btBuscar"
            android:textColor="@android:color/white"
            android:layout_marginTop="70dp"
            android:layout_marginLeft="5dp"
            android:layout_marginRight="20dp"
            android:textSize="12sp"
            android:background="@color/botonprincipal"
            android:layout_width="315dp" />
    </LinearLayout>

    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/btHelp"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="0dp"
        android:layout_marginLeft="0dp"
        android:background="@android:color/transparent" />

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="150dp"
        android:layout_marginBottom="135dp"
        android:id="@+id/lvAparatejos"
        android:background="#72fec1c1" />

    <ImageButton
        android:layout_height="50dp"
        android:id="@+id/btSalirConectar"
        android:scaleType="centerInside"
        android:background="@drawable/salir1"
        android:layout_marginTop="480dp"
        android:layout_marginLeft="130dp"
        android:layout_width="60dp" />

</RelativeLayout>
```

ii. Código Conectar

```
package com.example.adrin.beatbeat;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.ListView;
import android.widget.Toast;

public class Conectar extends AppCompatActivity {

    private BluetoothAdapter adaptadorBluetooth;
    private ListView listaelementos;
    private ArrayAdapter adaptador;
    private Button btBuscar;
```

```
    /*La constante REQUEST_ENABLE_BT pasada a startActivityForResult() es un valor entero
definid localmente (debe ser
     superior a 0)que el sistema te devuelve en onActivityResult() como parámetro
requestCode.*/
    private static int REQUEST_ENABLE_BT = 1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_conectar);
        //-------------------------------------------------------------------------------------
------------------------
        //DECLARACIÓN DE BOTONOES DE LA APP
        btBuscar = (Button) findViewById(R.id.btBuscar);
        final ImageButton btSalirConectar = (ImageButton) findViewById(R.id.btSalirConectar);
        final ImageButton btHelp = (ImageButton) findViewById(R.id.btHelp);

        //-------------------------------------------------------------------------------------
------------------------

        //-------------------------------------------------------------------------------------
------------------------
        //INICIALIZACIÓN BOTONES
        btHelp.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent pasarAInstrucciones = new Intent(Conectar.this, GuiaDelUsuario.class);
                startActivity(pasarAInstrucciones);
            }
        });
        btSalirConectar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                finish();
            }
        });
        //-------------------------------------------------------------------------------------
------------------------

        //-------------------------------------------------------------------------------------
------------------------
        // SE INTENTA OBTENER EL BLUETOOTH ADAPTER. Se debería conseguir 114%
        adaptadorBluetooth = BluetoothAdapter.getDefaultAdapter();
        if (adaptadorBluetooth == null) mensaje("Parece que el dispositivo no soporta
Bluetooth\n");
            //-------------------------------------------------------------------------------------
----------------------------

            //-------------------------------------------------------------------------------------
----------------------------
            //SE HACE Y DECLARA LO DEMÁS DE LA ACTIVIDAD (Si hemos obtenido el adaptador
        else {

            //tratamos de habilitar el Bluetooth (se manda un INTENT con la acción).
            if (!adaptadorBluetooth.isEnabled()) {
                Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
            }

            //-------------------------------------------------------------------------------------
----------------------------
            //Inicializamos el ListView y el Adaptador
            listaelementos = (ListView) findViewById(R.id.lvAparatejos);
            adaptador = new ArrayAdapter(this, android.R.layout.simple_list_item_1);
            listaelementos.setAdapter(adaptador);
            //-------------------------------------------------------------------------------------
----------------------


//*************************************************************************************
****************
            //A PARTIR DE AQUÍ, SOLO SE HARÁ SI EL RESULTADO DEL INTENT ANTERIOR HA SIDO
POSITIVO
```

```
//*************************************************************************************************
*****************

            //Tenemos que registrar el BroadcastReceiver para las dos acciones posibles:
            //ACTION FOUND y ACTION DISCOVERY FINISHED
            IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
            registerReceiver(receptorBC, filter);
            filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
            registerReceiver(receptorBC, filter);

            //-----------------------------------------------------------------------------------
----------------------------
            //Inicializamos el botón buscar
            btBuscar.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    //Se cancela la búsqueda que se podría estar realizando
                    if (adaptadorBluetooth.isDiscovering())
adaptadorBluetooth.cancelDiscovery();

                    // se deshabilita el botón buscar
                    btBuscar.setClickable(Boolean.FALSE);
                    // se eliminan los elementos que hubiera en la lista
                    adaptador.clear();

                    // Por último se inicia la búsqueda
                    adaptadorBluetooth.startDiscovery();
                    mensaje("Buscando ... El proceso durará varios segundos");
                }
            });
            //-----------------------------------------------------------------------------------
----------------------------

            listaelementos.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                @Override
                public void onItemClick(AdapterView<?> parent, View view, int position, long
id) {

                    //Si hemos encontrado el dispositivo, lo primero es hacer que le bluetooth
deje de buscar
                    adaptadorBluetooth.cancelDiscovery();

                    //tenemos que recuperar la address que se encuentra escrita en la posicion
seleccionada
                    String texto = (listaelementos.getItemAtPosition(position)).toString();

                    //-----------------------------------------------------------------------
---------------------------
                    //Del String total, separo el name y la address
                    int separacion = texto.indexOf("\n");
                    final String address = texto.substring(separacion + 1);
                    final String name = texto.substring(0, separacion);
                    //En cada posicion del listview esta el name y la address de cada device
                    //lo que tengo que pasar es la address, aunque igual necesito el name tb
                    //-----------------------------------------------------------------------
---------------------------

                    //-----------------------------------------------------------------------
-------------------------
                    //Diálogo de confirmación de que te quieres conectar a este dispositivo
                    new AlertDialog.Builder(Conectar.this)
                            .setTitle("¿CONECTAR?")
                            .setMessage("Dispositivo seleccionado: \nName: " + name +
"\nAddress: " + address)
                            .setPositiveButton("SÍ", new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog, int id) {
                                    Intent accesoprinci = new Intent(Conectar.this,
Principal.class);

                                    accesoprinci.putExtra("Direccion", address);
                                    startActivity(accesoprinci);
                                    Log.d("RECEPTOR", "Actividad lanzada");
                                    dialog.cancel();
                                }
                            }).setNegativeButton("NO", new DialogInterface.OnClickListener() {
```

```
                        public void onClick(DialogInterface dialog, int id) {
                            dialog.cancel();
                        }
                    }).show();
                    //-------------------------------------------------------------------------
-------------------------
                }
            });
        }
    }
    // Creamos el Broadcast Receiver para recibir las acciones de tipo:
    // Nuevo dispositivo --> ACTION_FOUND; y Búsqueda finalizada --> ACTION_DISCOVERY_FINISHED
    private final BroadcastReceiver receptorBC = new BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();

            if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                //Cuando llega un aviso de "Nuevo dispositivo":
                //obtenemos el device y añadimos su nombre y direccion al Adaptador para
mostrarlo en el ListView
                BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                adaptador.add(device.getName() + "\n" + device.getAddress());
            } else {
                if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
                    //TODO volvemos a habilitar el boton buscar
                    btBuscar.setClickable(Boolean.TRUE);
                    if (adaptador.getCount() == 0)
                        mensaje("No se han encontrado dispositivos. Reinicia el BeatBeat y
pulsa BUSCAR");
                    else mensaje("Búsqueda finalizada. Si no ves tu dispositivo, reinicia el
BeatBeat y pulsa BUSCAR");
                }
            }
        }
    };

    @Override
    protected void onActivityResult(int requestCode, final int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode==REQUEST_ENABLE_BT){
            //Es el resultado del intent de habilitar el Bluetooth
            if (resultCode == RESULT_OK) mensaje("Bluetooth Habilitado.");
            else {
                // El usuario no ha habilitado el Bluetooth
                // No debe ser muy avispado, pero le damos otra oportunidad
                mensaje("Esta aplicación solo puede funcionar con Bluetooth.");

                //-------------------------------------------------------------------------
-------------------------
                //Se le da otra oportunidad al usuario, que esta empanao. Se muestra otro
mensaje de alerta
                new AlertDialog.Builder(Conectar.this)
                        .setTitle("¿SALIR DE LA APLICACIÓN?")
                        .setMessage("Para usar la aplicación es necesario habilitar el
Bluetooth. " +
                                "\nPara ello pulse 'ACTIVAR'. " +
                                " Si no desea habilitar Bluetooth puede salir de la app
pulsando 'CERRAR APP'"+
                                "\nSi tiene dudas, consulte la Guía del Usuario de la
aplicación")
                        .setPositiveButton("CERRAR APP", new DialogInterface.OnClickListener()
{
                            public void onClick(DialogInterface dialog, int id) {
                                dialog.cancel();
                                finish();
                            }
                        }).setNegativeButton("ACTIVAR", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        dialog.cancel();
                        Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                        startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
```

```
                    }
                }).show();

            }
        }
        else mensaje("No se que coño ha podido pasar. No he hecho mas
startForActivityResult");
    }

    //FUNCION BASE PARA ESCRIBIR MENSAJES POR PANTALLA
    protected void mensaje(String comentario) {
        Toast toast1 = Toast.makeText(getApplicationContext(), comentario, Toast.LENGTH_LONG);
        toast1.show();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        //TODO ver porque mierda falla esto
        unregisterReceiver(receptorBC);
    }

    @Override
    public void onBackPressed() {
        super.onBackPressed();
        adaptadorBluetooth.cancelDiscovery();
        finish();
    }
}
```

f. Pantalla Principal

i. XML Principal

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_principal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.adrin.beatbeat.Principal"
    android:background="@drawable/bgdegradado">

    <com.jjoe64.graphview.GraphView
        android:layout_width="match_parent"
        android:id="@+id/funcion"
        android:layout_marginLeft="10dp"
        android:background="@android:color/transparent"
        android:layout_marginTop="50dp"
        android:layout_height="305dp" />

    <Button
        android:text="Finalizar análisis"
        android:layout_width="200dp"
        android:layout_height="30dp"
        android:id="@+id/btSal"
        android:background="@color/botonprincipal"
        android:textColor="@android:color/white"
        android:layout_marginLeft="380dp" />

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="300dp"
        android:background="@color/colorAccent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
```

```
        android:textAlignment="center"
        android:layout_marginBottom="25dp">

        <TextView
            android:text="Frecuencia"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/tvPalabra"
            android:layout_weight="0.52"
            android:textSize="16sp"
            android:background="@color/botonprincipal"
            android:textAlignment="textEnd" />

        <TextView
            android:text="00 bpm"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/tvFrecuencia"
            android:layout_weight="1"
            android:textSize="16sp"
            android:textAlignment="center"
            android:background="@color/botonprincipal" />
    </LinearLayout>

    <TextView
        android:text="c o n e c t a d o  a . . ."
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/tvConectadoA"
        android:textAllCaps="true"
        android:shadowRadius="3"
        android:layout_alignBottom="@+id/btSal"
        android:layout_alignLeft="@+id/funcion"
        android:layout_alignStart="@+id/funcion"
        android:textStyle="normal|bold" />

    <Switch
        android:text="2s/5s"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/switch1"
        android:textSize="20sp"
        android:layout_alignBottom="@+id/btSal"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

ii. Código Principal

```
 private List valores;
 private Button salir;

 private BluetoothAdapter adaptadorBlth;
 private BluetoothDevice devicerecib;
 private SerivicioBluetooth servicioBlth;

 private final int LECTURA_DATOS =1111;
 private final int LECTURA_HILO=2222;

 private final double T=0.008; //Aqui tenemos basicamente el periodo de muestreo de nuestro
aparato (separacion entre los puntos del ECG)
 private double lastX=0; //la ultima coordenada X utilizada en el grafico (la primera coord es
0)
 private int lastY=0;
 private GraphView grafica;
 private LineGraphSeries<DataPoint> puntos;

 private byte[]  vectorAux, vectorUtil;
 private DataPoint punticos, punto1,punto2, punto3;
 private int numero;
 private int nuevacant=0;
```

```
 private int posicionAux, posicionUtil;

 private MyHandler handlerHilo;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_principal);

     vectorAux=new byte[10];
     vectorUtil=new byte[10];

     HandlerThread hilo = new HandlerThread("Worker Thread");
     hilo.start();
     Looper mLooper = hilo.getLooper();
     handlerHilo = new MyHandler(mLooper);

     //Recibimos la ADDRESS que hemos pasado en el intent y obtenemos el DEVICE a partir de
ella.
     Bundle extras=getIntent().getExtras();
     String address=extras.getString("Direccion");

     adaptadorBlth=BluetoothAdapter.getDefaultAdapter();

     deviceрecib= adaptadorBlth.getRemoteDevice(address);

     mensaje("Dispositivo conectado:");
     mensaje("Nombre: "+deviceрecib.getName()+ ". Address: "+deviceрecib.getAddress());

     Log.d("PRINCIPAL", "Actividad arrancada, Extra obtenid, BluetoothAdapter Obtenido,
RemoteDevice obtenido");

     //Creamos un objeto ServicioBluetooth y llamamos al metodo CONECTAR con el DEVICE que
hemos recibido
     servicioBlth=new SerivicioBluetooth(handlerHilo);
     Log.d("PRINCIPAL", "Servicio Bluetooth creado");

     servicioBlth.conectar(deviceрecib);
     Log.d("PRINCIPAL", "ServicioBluetooth.Connect lanzada correctamente");
     //El metodo conectar, en el caso de conseguir la conexion, llama por si solo al metodo
entreHilos que administra la conexion

     //Declaramos el filtro del Broadcast Receiver para laposible desconexión
     IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_ACL_DISCONNECTED);
     registerReceiver(desconexión, filter);


     //-----------------------------------------------------------------------------------------
--------------------
     //AJUSTES RELATIVOS A LA GRÁFICA
     final GraphView grafica = (GraphView)findViewById(R.id.funcion);
     Viewport ajustes= grafica.getViewport();
     puntos= new LineGraphSeries<DataPoint>();
     grafica.addSeries(puntos);
     grafica.setTitle("Electrocardiograma");
     grafica.setTitleColor(Color.BLACK);
     grafica.setTitleTextSize(50);
     puntos.setDrawBackground(true);
     puntos.setBackgroundColor(Color.argb(40,255,120,120));
     puntos.setThickness(5);
     puntos.setColor(Color.rgb(255, 107, 107));
     ajustes.setBorderColor(Color.WHITE);
     ajustes.setYAxisBoundsManual(false); //voy a dejar que la grafica se autoajuste a ver que
tal
     //ajustes.setMinY(7960000);
     //ajustes.setMaxX(7980000);
     //viewport.setScalableY(true); //Igual si el pulso se ve muy pequeño en la grafica,
podria estar bien hacer zoom (en el eje y)
     ajustes.setXAxisBoundsManual(true); //Necesitamos hacer esto TRUE, para que el metodo
scrollToEnd() funcione:
     ajustes.setMinX(0);
     ajustes.setMaxX(3);
     ajustes.setScrollable(true);
     //-----------------------------------------------------------------------------------------
```

```
--------------------

    //-------------------------------------------------------------------------------------------
--------------------
    //declaracion resto elementos pantalla

    final Button btFinalizar= (Button)findViewById(R.id.btSal);

    btFinalizar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            servicioBlth.cerrarBluetooth();
        }
    });
    frecuencia= (TextView)findViewById(R.id.tvFrecuencia);
    //-------------------------------------------------------------------------------------------
--------------------

 }


/*****************************************************************************************
*******************
 *   HANDLER
 *   EL handler se usa para comunicar el hilo de connected thread con el hilo principal
 *   Cuando nos llega información al bluetooth, este se la pasa al handler con el codigo
LECTURA DATOS
 *   Esta informacion son puntos del ECG.
 *   La info viene descompuesta en diferentes bytes.
 *   El handler llama a una funcion que procesa los datos. Esto lo hace en un hilo diferente
al principal (para no bloquearlo)
 *   */
 class MyHandler extends Handler {
    public MyHandler(Looper myLooper) {
        super(myLooper);
    }
    public void handleMessage(Message msg) {
        switch (msg.what){
            case LECTURA_DATOS:
                generaPuntos((byte[])msg.obj, msg.arg1);
                break;
            default:
                break;
        }
    }
 }
 //Fin de HANDLER
 //-------------------------------------------------------------------------------------------
-------------------------


/*****************************************************************************************
************************
 *   FUNCION GENERA PUNTOS
 *   A partir de los datos enviados por el hilo creado en el handler,
 *   la funcion debe regenerar los valores reales (que venian descompuestos en diferentes bits
 *   y crear puntos DataPoint para el grafico*/
 private void generaPuntos(byte[] vector, int cantidad){

    Log.d("FUNCION", "Hemos entrado en la funcion genera_puntos. Hemos recibido: "+cantidad+"
bytes.");

    if (compruebaCodigo(vector, cantidad)){
        //el for siguiente exluye a los bytes de seguridad [0] y [cantidad]
        //estos bytes son los que se han utilizado en la funcion compruebaCodigo

        addEntry(vectorUtil);
        if (posicionAux==10){
            //tenemos el vector auxiliar completo
            Log.d("FUNCION", "Tenemos el vector auxiliar a tope");
            addEntry(vectorAux);
        }else{
            if (posicionAux!=0){
                vectorUtil=vectorAux;
```

```
                posicionUtil=posicionAux;
            }

        }
    }
 }
 //Fin de funcion GENERA_PUNTOS
 //-------------------------------------------------------------------------------------------------
--------------------



/*************************************************************************************************
*************************
* FUNCION COMPRUEBA CODIGO
* Se supone que mando un byte al principio y otro al final desde el aparato,
* que me sirven para comprobar que los datos que he enviado son correctos
*/
 private Boolean compruebaCodigo(byte [] vectorRecib, int cantRecib){

     final byte start= (byte) 0xAA;
     //valor conocido que el micro pone al principio del conjunto de bytes a enviar mediante
el bluetooth
     if (vectorRecib[0]==start){
         //el vector se corresponde con el inicio de una transmision
         if (cantRecib==10){
             //la transmision ha sido perfecta. Juraria que simplemente no tengo que hacer
nada
             Log.d("COMPROBACION", "Transmision perfecta");
             vectorUtil=vectorRecib;
             posicionUtil=0;
             posicionAux=0;
             return Boolean.TRUE;
         }
         else{
             vectorUtil=new byte[10];
             vectorAux=new byte[10];


             if (cantRecib<10){
                 //hemos recibido la primera parte de un vector
                 Log.d("COMPROBACION", "Transmision incompleta. Tenemos la primera parte "+
cantRecib+ "bytes");
                 for (int i=0; i<cantRecib; i++) vectorUtil[i]=vectorRecib[i];
                 posicionUtil=cantRecib;
                 return Boolean.FALSE;
             }
             else{
                 //vector de mas de 10 bytes
                 Log.d("COMPROBACION", "Vector muy largo");
                 int i=0;
                 vectorUtil=new byte[10];
                 do {
                     vectorUtil[i]=vectorRecib[i];
                     i++;
                 }while (vectorRecib[i]!=start && i<10);
                 //hasta aqui rellenamos el vector bueno
                 //Ahora tenemos que guardar lo que sobra en otro auxiliar

                 posicionAux=0;
                 int min= Math.min(20,cantRecib);
                 for (int j=10; j<min; j++){
                     //vamos a asumir que no llegan mas de 20 de golpe
                     //Si no, se complica mucho el asunto
                     vectorAux[posicionAux]= vectorRecib[j];
                     posicionAux++;
                 }
                 return Boolean.TRUE;
             }
         }
     }
     else {
         if (lastX==0) return Boolean.FALSE;
         // el vector NO es el inicio de una transmision.
```

```
        if (vectorUtil==null){
            vectorUtil=new byte[10];
            posicionUtil=0;
        }

        int i=0;
        while (vectorRecib[i]!=start && posicionUtil<10 && i<cantRecib){
            vectorUtil[posicionUtil]=vectorRecib[i];
            posicionUtil++;
            i++;
        } //en este while completamos el vector (hasta llegar al siguiente byte de inicio)


        if (posicionUtil ==10){
            //eso es que hemos llegado al bit de inicio y por tanto el vector auxiliar esta
completo
            Log.d("COMPROBACION", "Transmision completada. Tenemos la segunda parte "+
cantRecib+ " bytes");

            posicionAux=0;
            if (i<cantRecib){
                Log.d("COMP1", "Se ha completado el vector util pero siguen quedando bytes:
"+ (cantRecib-i));
                //basicamente, tendriamos parte del siguiente vector (o el vector entero)
                do {
                    vectorAux[posicionAux]=vectorRecib[i];
                    i++;
                    posicionAux++;
                }while (vectorRecib[i]!=start && posicionAux<10 && i<cantRecib);
            }

            return Boolean.TRUE;
        }
        else{
            Log.d("COMPROBACION", "Transmision incompleta. Tenemos ¿otra? parte "+ cantRecib+
"bytes");

            if (vectorUtil==null){
                vectorUtil=new byte[10];
                posicionUtil=0;
            }

            while (vectorRecib[i]!=start && posicionUtil<10 && i<cantRecib){
                vectorUtil[posicionUtil]=vectorRecib[i];
                posicionUtil++;
                i++;
            }

            //bueno, aun asi no hemos llegado a completar el vector de 10. Creo que esto es
imposible
            return Boolean.FALSE;
        }
        //Log.d("COMPROBACION", "El primer byte no era 0xAA
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++");
    }
 }
 //Fin de funcion COMPRUEBA CÓDIGO
 //-------------------------------------------------------------------------------------------
-----------------------


 protected BroadcastReceiver desconexión =new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (BluetoothDevice.ACTION_ACL_DISCONNECTED.equals(action)){
            //Oh no se nos ha desconectado el dispositivo
            new AlertDialog.Builder(Principal.this)
                    .setTitle("EL DISPOSITIVO SE HA DESCONECTADO")
                    .setMessage("Regrese a la pantalla de búsqueda. Apague y encienda el
dispositivo. " +
                            "Vuelva a buscarlo y seleccionarlo.\nSi tiene dudas consulte la
Guía del Usuario.")
                    .setPositiveButton("VOLVER", new DialogInterface.OnClickListener() {
```

```
                        public void onClick(DialogInterface dialog, int id) {
                            finish();
                            dialog.cancel();
                        }
                    }).setNegativeButton("NO", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.cancel();
                }
            }).show();
        }
    }
};

@Override
protected void onDestroy() {
    super.onDestroy();
    unregisterReceiver(desconexión);
    servicioBlth.cerrarBluetooth();
}

//FUNCION BASE PARA ESCRIBIR MENSAJES POR PANTALLA
protected void mensaje(String comentario) {
    Toast toast1 = Toast.makeText(getApplicationContext(), comentario, Toast.LENGTH_SHORT);
    toast1.show();
}


/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++
 * FUNCION ADD ENTRY
 * Funcion simple que añade nuevos puntos a la grafica (los generados por la funcion
genera Puntos)
 * Para ello utilizamos el metodo appendData
 * */
private void addEntry(byte[] nuevo) {
    //parametros del metodo appendData:
    //  DataPoint (pasado como parametro por la funcion generaPuntos)
    //  True (indica que despoues de añadir el dato hace scrollToEnd)
    //      Yo he puesto una funcion scrollToEnd justo detras pq no me lo hacia, pero creo
que no hace falta
    //  Int--> es el numero de puntos visibles de la funcion. Yo quiero ver todos los puntos
desde el incio del programa al final.

    //Log.d("FUNCION", "Hemos entrado en la funcion addEntry");

    int valor=0;
    double antiguoX=lastX;


    for (int  i=0; i<3;i++){


        //Cada valor generado en el device tiene 24 bits y se divide en 3 bytes que en esta
funcion tenemos que reconstruir
        //Asi cada 3 bytes tendremos un punto
        //De cada 3 bytes, uno seran los 8 primeros bits, otro los 8 siguientes y el que
queda los 8 últimos

        // Viendo los logs anteriores, cuando transformo un numero mayor q 128 (2^7) a byte,
y luego lo vuelvo a hacer entero,
        //  lo entiende como si fuera negativo (complementoa2). La representacion binaria de
este numero negativo son 24 bits a 1
        //  y luego los 8 bits correspondientes al byte (el bueno, el que he generado)
        // Asi de la cadena de 32 bytes, quiero recuperar los 8 últimos.
        //   Para eso, hago una operacion logica de tipo & con el numero 0xFF=11111111;
        //   Así la nueva variable contendra exactamente los 8 ultimos bits del byte que nos
ha llegado
        //   Pero sin tomarlo como un byte (con complemento a 2), sino como un entero
positivo de 8 bits


        valor=(int) (((nuevo[3*i+1]&0xFF)^0x80)<<16) | ((nuevo[3*i+2]&0xFF)<<8) |
(nuevo[3*i+3]&0xFF);
```

```
            valores.add(valor);

            if (Math.abs(valor-lastY)>10000 && lastY!=0) {
                Log.d("FILTRO", "Punto filtrado: "+ valor);
                valor=lastY;
            }

            //en la variable valor ya tenemos todos los bits correspondientes a un dato dentro de
un mismo int
            //tengo que conseguir el valor real de esto
            //Si es un ADC normal, lo unico que necesito es entre que 2 valores mide el aparatito
            //valor_real=valor; //suponiendo que no hacemos ninguna conversion:--> todo hacer la
conversion que toque

            antiguoX=antiguoX+T; // añadimos un periodo mas a lastX que contiene la ultima
coordenada X de la grafica

            punticos=new DataPoint(antiguoX, valor);
            Log.d("PUNTO", "Nuevo punto: "+ punticos);
            lastY=valor;

            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    //Log.d("HILO", "Hemos creado lo de runear en el hilo principal");
                    puntos.appendData(punticos,true,500);
                    //addEntry(punticos);
                }
            });
        }
        lastX=antiguoX;
    }
    //Fin de funcion ADD ENTRY
    //-------------------------------------------------------------------------------------------
------------------

}
```

1. Clase Servicio Bluetooth

```
package com.example.adrin.beatbeat;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Handler;
import android.util.Log;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;

/**
 * Created by Adrián on 02/06/2017.
 */

public class SerivicioBluetooth {
    private BluetoothAdapter adaptadorBlth;
    private ConnectThread hiloconexion;
    private ConnectedThread hilo_adm_conexion;
    private Handler handlerservicio;
    private Context contextoapp;

    private final String frase_uuid="00001101-0000-1000-8000-00805F9B34FB";
    private final UUID uuid_aparatito= UUID.fromString(frase_uuid);
```

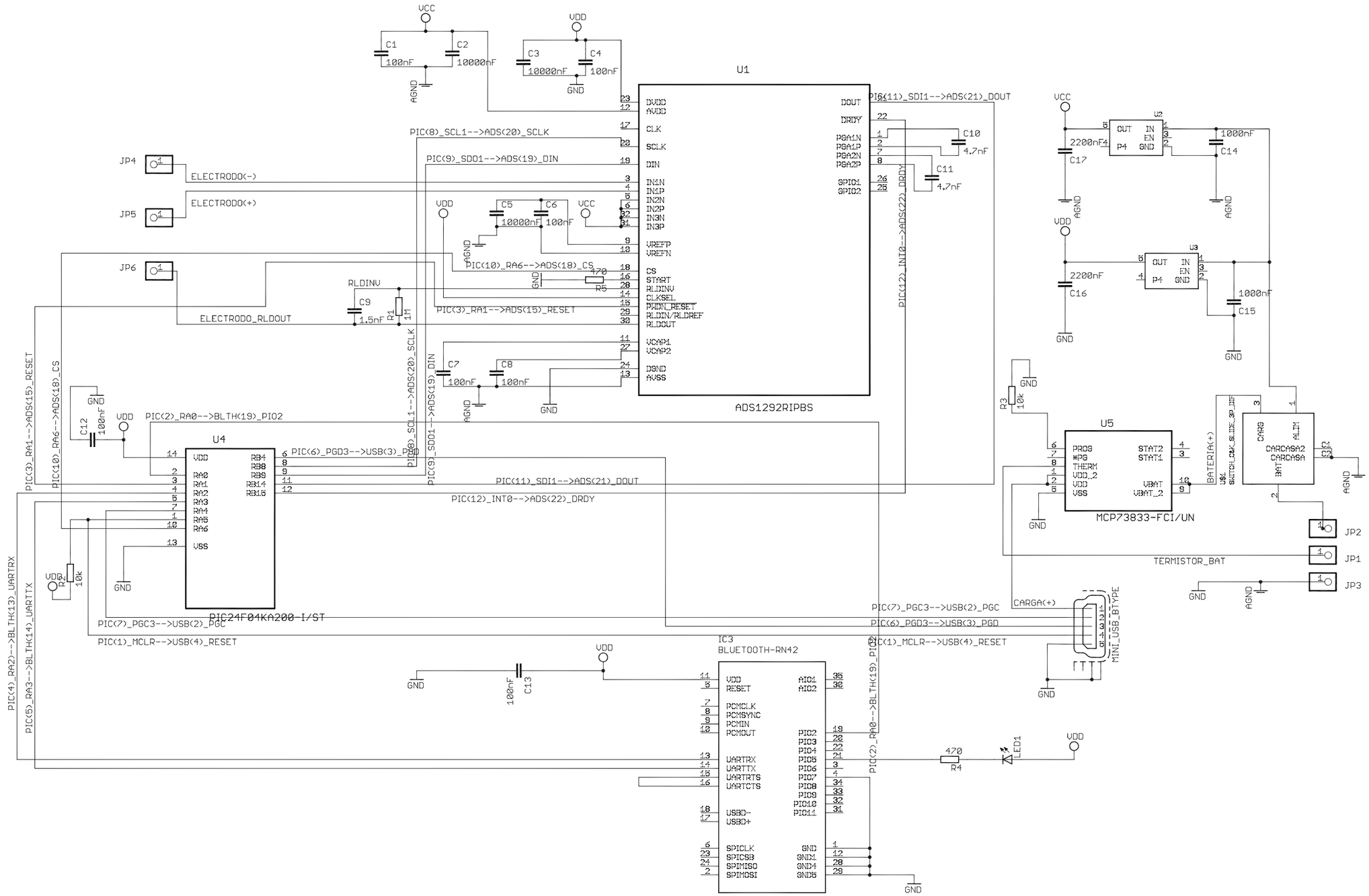
```
    private final int LECTURA_DATOS= 1111;
    //private final int DEBUGGEADOR=2222;

    private SharedPreferences logs;


/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++
    * CONSTRUCTOR DE LA CLASE SERVICIO BLUETOOTH
    * Le pasamos como parametro el handler que se ha creado en la actividad principal para
recibir los mensajes
    * */
    public SerivicioBluetooth(Handler hndl){
        handlerservicio=hndl;
        adaptadorBlth=BluetoothAdapter.getDefaultAdapter();
        Log.d("SERVICIO BLUETOOTH", "Servicio creado");
    }
    //Fin del CONSTRUCTOR
    //-----------------------------------------------------------------------------------------------
----------------------


/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++
    * FUNCION CERRAR BLUETOOTH
    * Funcion que nos sirve para parar las conexiones que hayamos creado
    * */
    public void cerrarBluetooth(){
        if (hiloconexion!=null){
            hiloconexion.cancel();
            hiloconexion=null;
        }
        if (hilo_adm_conexion!=null){
            hilo_adm_conexion.cancel();
            hilo_adm_conexion=null;
        }
        Log.d("SERVICIO BLUETOOTH", "Cerrando Servicio");

        // mensaje("CERRANDO EL SERVICIO BLUETOOTH");
    }//Fin de la funcion CERRAR_BLUETOOTH
    //-----------------------------------------------------------------------------------------------
------------------------



/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++
    * FUNCION CONECTAR --> PARA INICIAR EL HILO DE CONEXION
    * Lo primero es comprobar que se hace es comprobar si hubiera algo conectado o
conectandose ya
     * En ese caso, cancelamos la conexion que pudiera haber
     * Despues creamos un nuevo hilo de conexion y ejecutamos su funcion run() (con el metodo
start())
     * */
    public synchronized void conectar (BluetoothDevice device){
        if (hiloconexion!=null){
            hiloconexion.cancel();
            hiloconexion=null;
        }
        if (hilo_adm_conexion!=null){
            hilo_adm_conexion.cancel();
            hilo_adm_conexion=null;
        }
        Log.d("SERVICIO BLUETOOTH", "Dentro de funcion conectar");

        hiloconexion=new ConnectThread(device);
        hiloconexion.start();
        //mensaje("Intentando conectar con el dispositivo encontrado");
    }
    //Fin de la funcion CONECTAR
    //-----------------------------------------------------------------------------------------------
----------------------
```

```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++
    * FUNCION ENTRE HILOS
    * Cuando el hilo CONNECT ha establecido la conexion, se ha de llamar al hilo CONNECTED
para q la administre
    * En ejemplos que he enecontrado en internet, hace CONNECT.CANCEL()
    * A mi esto me parece absurdo. si cancelas el hilo de conexion, haces SOCKET.CLOSE();
    * Y justamene necesitas este socket para el siguiente hilo
    * Así solo cancelo otros posibles CONNECTED que puedan existir
    * */
    private void entreHilos(BluetoothSocket socket_creado){
        if (hilo_adm_conexion!=null){
            hilo_adm_conexion.cancel();
            hilo_adm_conexion=null;
        }
        Log.d("SERVICIO BLUETOOTH", "Dentro de funcion entrehilos");

        hilo_adm_conexion=new ConnectedThread(socket_creado);
        hilo_adm_conexion.start();
        //mensaje("Conexión establecida. Intentando acceder al hilo que administra la
conexion");
    }//Fin de funcon ENTRE HILOS
    //-------------------------------------------------------------------------------------------
-------------------------


/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++
    * CLASE CONNECT_THREAD
    * Es un hilo cuya única función es establecer la conexión con el dispositivo seleccionado
    * Establecer la conexión consiste en crear el BLUETOOTH SOCKET a partir de la UUID
    * Una vez establecida la conexión, se llama a una función para pasar al otro hilo
    * */
    private class ConnectThread extends Thread {
        private final BluetoothSocket Socketcom;
        private final BluetoothDevice miDevice;

        public ConnectThread(BluetoothDevice device) {
            BluetoothSocket temporal = null;
            miDevice = device;

            try {
                //Supongo que tendré que poner la UUID que viene en el data sheet del
bluetooth.
                temporal = miDevice.createRfcommSocketToServiceRecord(uuid_aparatito);
            } catch (IOException e) { }
            Socketcom = temporal;
            Log.d("SERV BLTH CONNECT", "Constructor de CONNECT");

        }

        public void run() {

            Log.d("SERV BLTH CONNECT", "CONNECT funcion run");

            try {
                Socketcom.connect();
                Log.d("SERV BLTH CONNECT", "Conexion establecida");

            } catch (IOException connectException) {
                try {
                    Socketcom.close();
                    Log.d("SERV BLTH CONNECT", "Conexion fallida. Cerrando socket");
                } catch (IOException closeException) { }
                return;
            }

            entreHilos(Socketcom);
        }

        public void cancel() {
            try {
                Socketcom.close();
                Log.d("SERV BLTH CONNECT", "CONNECT funcion cancel");
```

```
            } catch (IOException e) { }
        }
    }//Fin de clase CONNECT THREAD
    //-------------------------------------------------------------------------------------------------
----------------------


    /*
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++
    *   CLASE CONNECTED THREAD
     *  Hilo encargado de administrar la conexión una vez establecida
     *  Hace falta pasarle el socket que se creó en el hilo de CONNECT
     *  */
    private class ConnectedThread extends Thread {
        private final BluetoothSocket Socketcom;
        private final InputStream entrada;
        private final OutputStream salida;

        public ConnectedThread(BluetoothSocket socket) {
            Socketcom = socket;
            InputStream InTemporal = null;
            OutputStream OutTemporal = null;

            try {
                InTemporal = socket.getInputStream();
                OutTemporal = socket.getOutputStream();
            } catch (IOException e) { }

            entrada = InTemporal;
            salida = OutTemporal;
            Log.d("SERV BLTH CONNECTED", "Constructor de CONNECTED");

        }

        public void run() {
            byte[] buffer = new byte[1024];
            int bytes;
            Log.d("SERV BLTH CONNECTED", "CONNECTED funcion run. Teoricamente listo para
recibir info");


            while (true) {
                try {

                    bytes = entrada.read(buffer);

                    handlerservicio.obtainMessage(LECTURA_DATOS, bytes, -1,
buffer).sendToTarget();
                } catch (IOException e) {
                    break;
                }
            }
        }

        public void write(byte[] bytes) {
            try {
                salida.write(bytes);
            } catch (IOException e) { }
        }


        public void cancel() {
            try {
                Socketcom.close();
                Log.d("SERV BLTH CONNECT", "CONNECT funcion cancel");
            } catch (IOException e) { }
        }
    }//Fin de clase CONNECTED_THREAD
    //-------------------------------------------------------------------------------------------------
----------------------
}
```

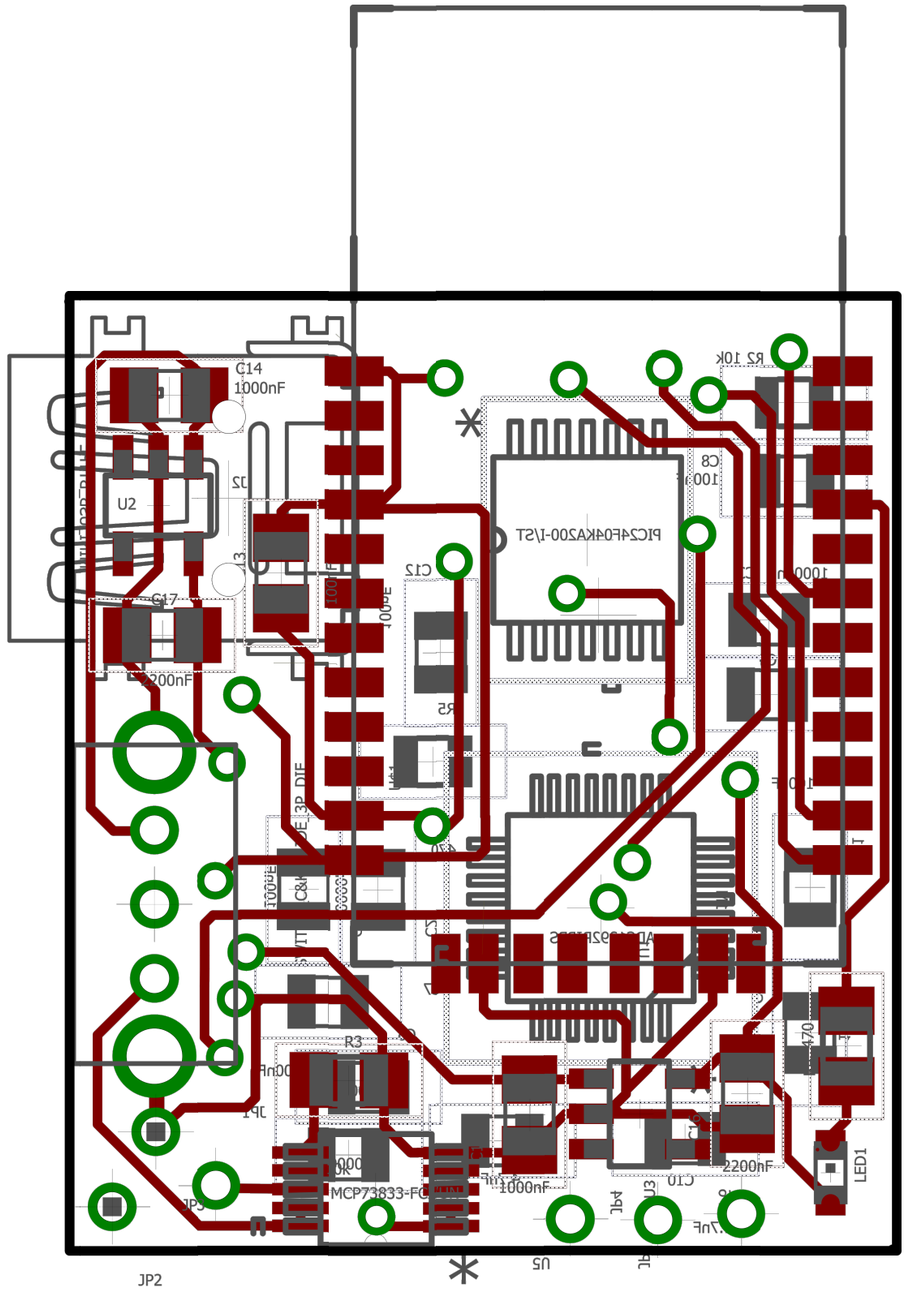
Planos

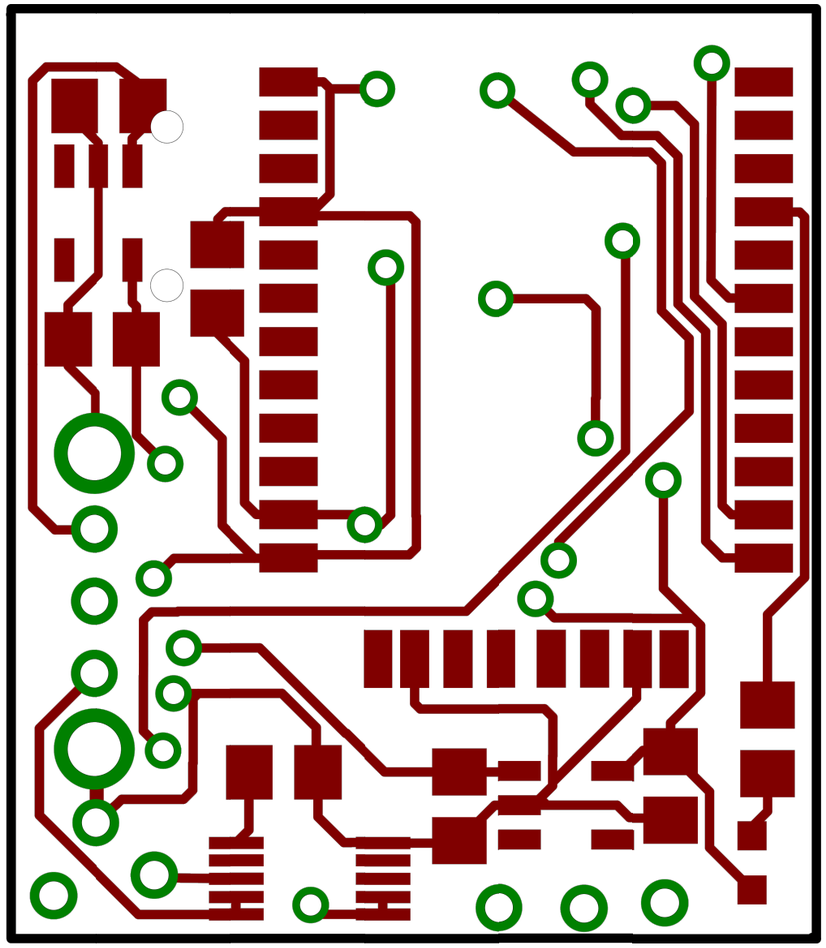
P1. Esquema eléctrico



TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES UNIVERSITAT POLITÈCNICA DE VALÈNCIA — ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: DISEÑO Y DESAROLLO DE UN SISTEMA PARA EL REGISTRO DE SEÑALES BIOELÉCTRICAS EXTRACELULARES	Plano: Esquema eléctrico del dispositivo Autor: Adrián Beneit Barajas	Fecha: Julio 2017 Escala:	Nº Plano: 1

P2. Circuito impreso TOP





TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Proyecto: DISEÑO Y DESAROLLO DE UN SISTEMA PARA EL REGISTRO DE SEÑALES BIOELÉCTRICAS EXTRACELULARES

Plano: Plano de serigrafía superior, con y sin componentes

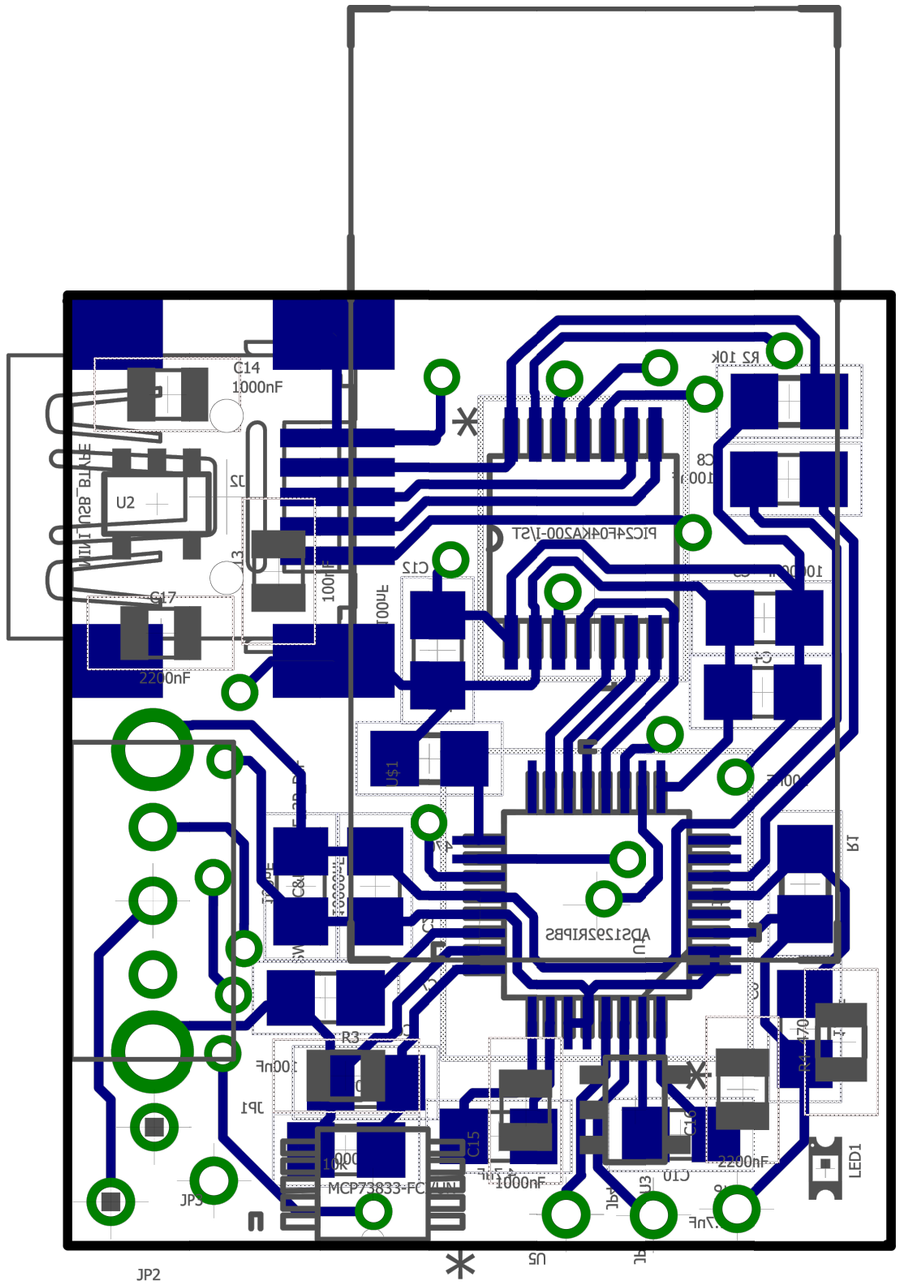
Autor: Adrián Beneit Barajas

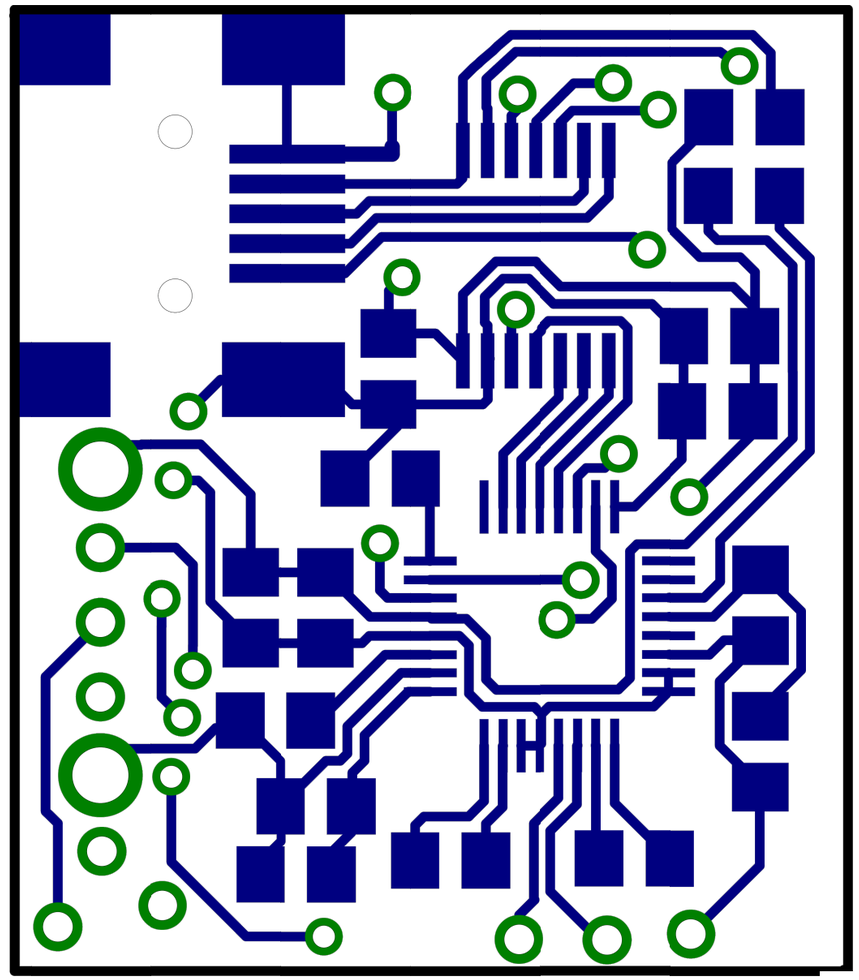
Fecha: Julio 2017

Escala: 10:1

Nº Plano: 3

P3. Circuito impreso BOTTOM





TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES UNIVERSITAT POLITÈCNICA DE VALÈNCIA ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: DISEÑO Y DESAROLLO DE UN SISTEMA PARA EL REGISTRO DE SEÑALES BIOELÉCTRICAS EXTRACELULARES	Plano: Plano de serigrafía inferior, con y sin componentes Autor: Adrián Beneit Barajas Fecha: Julio 2017 Escala: 10:1 Nº Plano: 2

Pliego de condiciones

1. LISTA COMPONENTES

COMPONENTE	NÚMERO
ADS1292R	1
PIC24F04KA200	1
RBBT42-i/rm	1
Batería 3,7V= 1ICP4/20/25	1
MCP73833	1
TPS709	2
Interruptor 2 posiciones	1
Electrodo de un solo uso	3
Cables conexión electrodos	3
Led SMD	1
Conector USB mini	1
Condensador 100 nF	7
Condensador 1000nF	5
Condensador 2000nF	2
Condensador 470 nF	2
Condensador 1.5nF	1
Resistencia 10K Ohm	2
Resistencia 470 Ohm	2
Resistencia 1M Ohm	1

2. ESPECIFICACIONES Y LEGISLACIÓN

La legislación referente al uso de los componentes se encuentra contenida en las correspondientes hojas de datos, referencias [10]-[21]. Con el objetivo de no hacer el documento más largo, estas no se han incluido en el pliego. Es de especial interés la correspondiente al dispositivo ADS1292R, por ser el único componente que se encuentra eléctricamente unido al cuerpo del paciente.