



Universidad  
Politécnica  
de Cartagena



**industriales**

etsii UPCT

# Diseño y Desarrollo de un Gadget de e-Health Basado en Bluetooth Low Energy y Compatible con Dispositivos Móviles Inteligentes Android

**Titulación:** Ingeniería Industrial

**Alumno:** José Francisco Beltrán Sánchez

**Director:** Juan Antonio López Riquelme

Cartagena, 28 de septiembre de 2016

# ÍNDICE

<b>CAPÍTULO 1: INTRODUCCIÓN .....</b>	<b>1</b>
1.1 Contexto.....	1
1.2 Objetivos .....	2
1.3 Desarrollo de la Memoria.....	3
1.3.1 Capítulo 1 - Introducción .....	3
1.3.2 Capítulo 2. Estado del Arte .....	3
1.3.3 Capítulo 3. Descripción del Sistema .....	3
1.3.4 Capítulo 4. Descripción del Hardware y Software Desarrollado.....	3
1.3.5 Capítulo 5. Conclusiones y Trabajos futuros .....	3
<b>CAPÍTULO 2: ESTADO DEL ARTE.....</b>	<b>5</b>
2.1 Introducción .....	5
2.2 Monitorización cardíaca .....	6
2.2.1 Electrocardiograma, ECG .....	6
2.2.2 Fotoplestimografía, PPG .....	7
2.3 Dispositivos móviles inteligentes y Sistemas Operativos. ....	9
2.3.1 Características de los diferentes SOs .....	9
2.4 Protocolos inalámbricos.....	14
2.4.1 Introducción a las Redes WPAN .....	14
2.4.2 Bluetooth Classic.....	15
2.4.3 Bluetooth Low Energy .....	15
2.4.4 Bluetooth 5 .....	18
2.4.5 ZigBee .....	18
2.4.6 WiFi .....	19
2.4.7 Conclusión.....	21
2.5 Módulos y Kits de desarrollo.....	22
2.5.1 Texas Instruments CC254X.....	22
2.5.2 Texas Instruments CC26xx .....	24

2.5.3	Nordic Semiconductor Serie nRF52.....	25
2.5.4	Dialog Semiconductor DA14580.....	26
2.5.5	Bluegiga, EFR32™ Blue Gecko Bluetooth® Low Energy SoCs.....	27
2.5.6	Comparativa de características de diferentes SoCs .....	28
<b>2.6</b>	<b>Conclusiones.....</b>	<b>29</b>
 <b>CAPÍTULO 3: DESCRIPCIÓN DEL SISTEMA.....</b>		<b>31</b>
<b>3.1</b>	<b>Introducción .....</b>	<b>31</b>
<b>3.2</b>	<b>Descripción del sistema de medida de ritmo cardíaco .....</b>	<b>31</b>
<b>3.3</b>	<b>Descripción del protocolo Bluetooth Low energy .....</b>	<b>33</b>
3.3.1	Introducción.....	33
3.3.2	Protocolos y perfiles .....	34
3.3.3	Perfil Genérico de Acceso (GAP).....	34
3.3.4	Perfil Genérico de Atributos (GATT).....	38
3.3.5	Perfiles específicos .....	41
<b>3.4</b>	<b>Descripción del hardware y entorno de desarrollo para el SoC CC2541 de TI .....</b>	<b>41</b>
3.4.1	Kit de desarrollo y descripción del hardware .....	41
3.4.2	Entorno de Desarrollo IAR Embedded Workbench .....	44
3.4.3	Proyecto de BLE en CC254x.....	49
<b>3.5</b>	<b>Desarrollo de APP para Android.....</b>	<b>53</b>
3.5.1	Instalación del Java SDK.....	53
3.5.2	Instalación de Android Studio y Android SDK .....	54
3.5.3	Configuración del dispositivo móvil para depurar aplicaciones .....	55
3.5.4	Creación de un proyecto en Android Studio.....	56
3.5.5	Estructura de un proyecto Android.....	58
3.5.6	Instalar la aplicación en un dispositivo con Android.....	60

<b>CAPÍTULO 4: DESCRIPCIÓN DEL HARDWARE Y SOFTWARE DESARROLLADO .....</b>	<b>63</b>
<b>4.1    Introducción .....</b>	<b>63</b>
<b>4.2    Descripción del hardware de adquisición y acondicionamiento de la señal de ritmo cardíaco .....</b>	<b>63</b>
4.2.1    Sensor emisor y receptor de luz.....	64
4.2.2    Etapa de filtrado y amplificación.....	65
4.2.3    Circuito completo y salida digital .....	66
<b>4.3    Descripción del firmware del módulo CC2541 Sensor Tag .....</b>	<b>67</b>
4.3.1    Creación de un perfil personalizado .....	67
4.3.2    Desarrollo del código de la aplicación.....	74
<b>4.4    Descripción de la App desarrollada en Android .....</b>	<b>77</b>
4.4.1    Creación del proyecto en Android Studio .....	77
4.4.2    Estructura de la aplicación de Android .....	78
4.4.3    Programación del archivo MainActivity.java.....	78
4.4.4    Programación del archivo BluetoothLeService.java.....	82
4.4.5    Programación del archivo activity_main.xml .....	85
4.4.6    Programación del archivo AndroidManifest.xml .....	88
<b>CAPÍTULO 5: CONCLUSIONES Y TRABAJOS FUTUROS .....</b>	<b>91</b>
<b>5.1    Análisis de Resultados .....</b>	<b>91</b>
<b>5.2    Conclusiones.....</b>	<b>92</b>
<b>5.3    Trabajos Futuros.....</b>	<b>92</b>
<b>BIBLIOGRAFÍA.....</b>	<b>95</b>



# Capítulo 1

---

## Introducción

---

### *1.1 Contexto*

Los dispositivos móviles inteligentes de tipo Smartphone no son meros dispositivos para llamar por teléfono y están caracterizados por tener, generalmente, una elevada capacidad de almacenamiento, cómputo y conectividad. Además, estos dispositivos se están proliferando mucho en los últimos años.

En este contexto se pueden desarrollar nuevos sistemas aprovechando la tecnología mencionada anteriormente. Concretamente, en el ámbito del *e-Health* y del deporte resulta muy interesante que los usuarios de los dispositivos móviles inteligentes puedan monitorizar ciertos parámetros importantes durante su actividad física y diaria, como el pulso cardiaco, los pasos realizados, entre otros.

Este trabajo consiste en diseñar los elementos hardware y software necesarios para obtener un sistema que permita monitorizar el pulso cardiaco. Para ello, será necesario diseñar y desarrollar toda la arquitectura hardware necesaria, es decir, la selección del sensor y del *SoC* que realizará las tareas de adquisición de datos, almacenamiento y transmisión de datos, así como el diseño de la electrónica de acondicionamiento entre ambos elementos.

También formará parte de este trabajo el diseño y desarrollo de la arquitectura software necesaria para el dispositivo empotrado con objeto de poder adquirir datos del sensor, así como almacenarlos temporalmente en la memoria y enviarlos usando el protocolo *Bluetooth Low Energy*.

Otro elemento del sistema será la aplicación Android que se desarrollará para poder intercambiar información de forma bidireccional con el sensor. En este caso se seguirán las fases típicas en diseño de software de análisis de requisitos, diseño e implementación para realizar la aplicación mencionada.

## 1.2 Objetivos

Este trabajo tiene como objetivo diseñar un sistema que permita monitorizar el ritmo cardíaco, utilizando un circuito diseñado especialmente para acondicionar dicha señal, y apoyándose en un *SoC* (System on Chip) para hacer la lectura de la señal, la recopilación de datos y el envío de los datos al Smartphone, y una aplicación de Android para monitorizarlos. Por tanto, se plantean los siguientes objetivos para realizar el sistema propuesto:

- Estudiar y elegir entre los diferentes métodos utilizados para la medición del pulso cardíaco, seleccionar el sensor y componentes necesarios para la utilización del método.
- Seleccionar una placa de desarrollo para implementar el protocolo *Bluetooth Low Energy* a través de un sistema de tipo *SoC*.
- Diseñar la electrónica de acondicionamiento necesaria para incorporar el sensor al sistema.
- Estudiar el diseño de aplicaciones con protocolo *BLE* en sistemas *SoC*.
- Estudiar la arquitectura de Android para implementar una aplicación sobre un dispositivo móvil inteligente.
- Realizar las pruebas de funcionamiento necesarias para validar la arquitectura propuesta.

## ***1.3 Desarrollo de la Memoria***

### ***1.3.1 Capítulo 1 - Introducción***

Este capítulo presenta la motivación de este trabajo, además de enumerar los objetivos del mismo y describir la memoria presentada.

### ***1.3.2 Capítulo 2. Estado del Arte***

En el capítulo 2 se expone el estado del arte de las diferentes tecnologías que se ven involucradas en este proyecto.

Se presentan las técnicas de monitorización cardíaca, los diferentes protocolos de para comunicación de dispositivos *wearables*, los diferentes alternativas de hardware que existen en el mercado y sus plataformas de desarrollo. También se analizan los diferentes sistemas operativos móviles, su cuota de mercado, arquitectura y desarrollo de aplicaciones.

Por último se concluye con la elección de las tecnologías más convenientes.

### ***1.3.3 Capítulo 3. Descripción del Sistema***

En este capítulo se desarrollaran las tecnologías del capítulo anterior que han sido escogidas. Se explica con más detalle el funcionamiento, su marco de trabajo, los recursos y las herramientas de las que disponen, así como del software necesario para su uso.

### ***1.3.4 Capítulo 4. Descripción del Hardware y Software Desarrollado***

En este capítulo se explica en profundidad el trabajo realizado durante el proyecto y las soluciones adoptadas, tanto a nivel de hardware como de software.

### ***1.3.5 Capítulo 5. Conclusiones y Trabajos futuros***

Este capítulo incluye los resultados obtenidos finalmente, las conclusiones a las que se ha llegado durante el trabajo realizado, así como las conclusiones sobre la consecución de los objetivos planteados.

Por último se aportan ideas de trabajos futuros, que no se han realizado por quedar fuera del alcance de este proyecto.





# Capítulo 2

---

## Estado del Arte

---

### *2.1 Introducción*

Es el fin de este capítulo el presentar el estado actual de la tecnología *wearable* enfocada a la monitorización de la salud para aplicaciones no médicas, en especial la dedicada a la monitorización del ritmo cardíaco.

Estos sistemas están basados en un gadget con capacidad de computación y a la vez permite vestirlos. Se refiere a productos que pueden almacenar, procesar y transferir datos como las computadoras, pero que se pueden utilizar como prendas de vestir y accesorios, los cuales forman parte del atuendo diario con los que convivimos y llevamos a todas partes: relojes, gafas, ropa, zapatos, gorras, anillos, pendientes, etc., muy distintos a las computadoras que conocemos actualmente.

A lo largo del capítulo, se presentarán y analizarán las diferentes alternativas que se han tenido en cuenta en la realización del proyecto en los diferentes ámbitos:

- Tecnología de monitorización cardíaca.
- Protocolo inalámbrico para transmisión de datos.
- El sistema hardware encargado de adquirir los datos, tratarlos y transmitirlos.

- Sistema operativo móvil sobre el que diseñar la aplicación donde se muestran los datos.

## ***2.2 Monitorización cardíaca***

La monitorización cardíaca consiste en medir la actividad del corazón. La principal actividad a medir son los impulsos que genera el corazón al bombear sangre al cuerpo, y su evolución con respecto al tiempo. De este modo, se pueden adquirir variables como frecuencia de ritmo cardíaco, variabilidad de la frecuencia cardíaca, tensión arterial y venosa, etc., permitiendo observar el comportamiento del corazón y la evaluación de las condiciones física de las personas a la hora de hacer deporte, así como la detección de posibles enfermedades cardíacas.

En este proyecto se toma como base la medición del ritmo cardíaco. Si bien con la misma tecnología se podrían implementar otras medidas, esto se deja para posteriores trabajos.

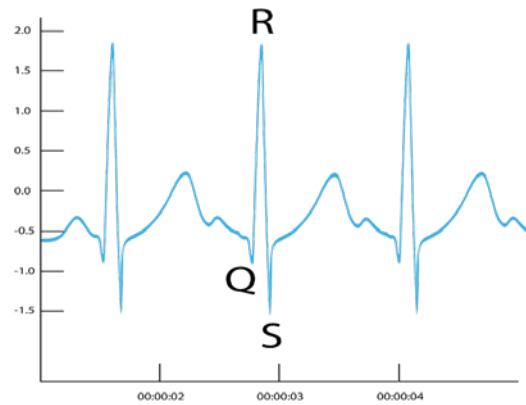
Existen diversas tecnologías disponibles para la monitorización cardíaca. En este análisis sólo se incorporan las dos más estudiadas a la hora de incorporar en un dispositivo *wearable*.

### ***2.2.1 Electrocardiograma, ECG***

El electrocardiograma (Wikipedia) mide la actividad eléctrica del corazón mediante el uso de electrodos colocados sobre la piel. Para que el corazón contraiga y bombee sangre, una serie de señales eléctricas coordinadas son enviadas al corazón por el sistema nervioso autónomo.

A pesar de que ocurren profundamente dentro del cuerpo, estas señales se pueden medir a nivel de la piel eficazmente usando dos o más electrodos colocados en varias posiciones en el pecho y las extremidades. Lo que resulta es un patrón con mucha información de la actividad eléctrica llamada onda ECG.

Aunque hay muchos aspectos de esta señal que pueden cuantificarse, la característica más destacada se conoce como el complejo QRS, que indica las principales contracciones de bombeo del corazón. Esta característica se muestra en la siguiente imagen:



**Ilustración 2-1. El complejo QRS**

Dentro del complejo QRS, cada letra muestra una localización diferente de la señal, y por lo tanto una acción diferente del corazón. El pico R es la mayor cantidad de actividad eléctrica producida por el corazón, y se usa comúnmente en las mediciones de la frecuencia cardíaca. Esto es lo que se utiliza en los algoritmos de frecuencia cardíaca para medir la cantidad de tiempo que se produce entre cada latido del corazón pulsante.

Debido a que las señales eléctricas viajan casi al instante, las mediciones basadas en ECG son generalmente precisas en un par de milisegundos, por lo que es una medida altamente confiable de la frecuencia cardíaca.

El ECG requiere que el potencial se mida a través del corazón. Esto significa que es necesario al menos dos puntos de contacto, un electrodo positivo y otro negativo. Normalmente, esta conexión sólo se realiza durante un período de tiempo finito. Dependiendo del tipo de ECG necesario, es posible que necesite varios canales de tipo ADC para realizar la medición.

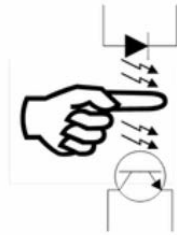
### ***2.2.2 Fotoplestimografía, PPG***

La fotoplestismografía (Texas Instruments) mide el cambio volumétrico del corazón midiendo la transmisión de luz o la reflexión. A medida que el corazón se contrae, la presión sanguínea dentro del ventrículo izquierdo - la cámara de bombeo principal - aumenta.

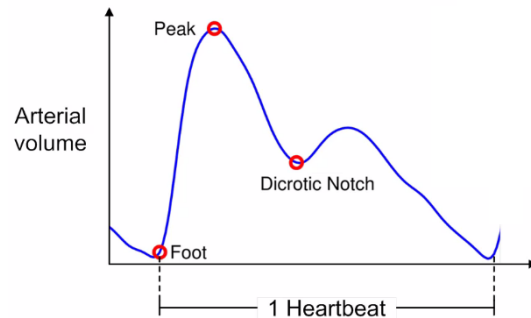
Este aumento obliga a un "pulso" presurizado de sangre en las arterias del cuerpo, lo que hace que se hinchen ligeramente antes de volver una vez más a su estado anterior.

La PPG se mide iluminando la piel y el tejido subcutáneo con luz de una longitud de onda específica. Esta luz proviene de un LED, o un diodo emisor de luz. El aumento de la presión debido al pulso causará una diferencia medible en la cantidad de luz absorbida,

transmitida o reflejada hacia atrás. Un sensor de fotodiodo mide la luz que se transmite o refleja, dependiendo de dónde se coloca en relación con el LED. El fotodiodo convierte entonces la luz medida en una señal eléctrica. La luz LED debe por tanto colocarse en un área donde las arterias estén cerca de la piel, como la punta de un dedo o un lóbulo de la oreja.



**Ilustración 2-2. Funcionamiento de fotopleletismografía**



**Ilustración 2-3. Forma de onda de presión arterial**

La amplitud de esta señal es directamente proporcional a la presión del pulso, cuanto más alto es el pico, más fuerte es el pulso. Aunque la señal puede ser menos evidente si se mide en la piel en un punto lejos del corazón (tal como un dedo), el cambio en la presión es todavía suficiente para expandir estas arterias hasta un grado que sea medible.

Cada pico en la señal resultante puede ser identificado por un algoritmo de frecuencia cardíaca especial que puede determinar en última instancia la cantidad de tiempo que se produce entre cada pico sucesivo dando otra medida de la frecuencia cardíaca.

Debido a que PPG es una técnica indirecta (que no registra la actividad verdadera del corazón que ocurre en su fuente), no es siempre exacta al nivel del milisegundo, aun así es una técnica fácilmente aplicable con la cual medir la frecuencia cardíaca de un individuo.

Por otro lado, proporciona más información sobre el flujo sanguíneo y la presión arterial. Esta medición puede realizarse en varios lugares del cuerpo para examinar el flujo sanguíneo a diferentes regiones. Cuando se mide más cerca de la aorta del corazón en el brazo izquierdo, se puede obtener información adicional sobre el gasto cardíaco y la función de la válvula cardíaca.

Una ventaja que tiene la PPG es en el número de contactos de la piel que se requieren para medirla. Al poder determinar la PPG bien con luz reflejada o transmitida, solamente es necesario un punto de contacto para poder medir. Esto permite medidas

fáciles continuadas en el tiempo, lo cual es la ventaja más atractiva para la electrónica *wearable*.

## ***2.3 Dispositivos móviles inteligentes y Sistemas Operativos.***

Los sistemas operativos móviles, son sistemas operativos especialmente diseñados para ejecutarse en dispositivos móviles, como puede ser *smartphones*, *tablets*, *smartwatches* y cualquier otro dispositivo que pueda ser transportado en la mano. Su diseño está orientado a la conectividad inalámbrica, los formatos multimedia para móviles y a la forma de introducir información. Estos dispositivos han llegado a ser un parte esencial de nuestro día a día, encontrando incluso personas que utilizan varios de estos dispositivos simultáneamente.

Observando el impacto que estos dispositivos han creado en nuestras vidas, se ha desarrollado una amplia variedad de sistemas operativos en el mercado, que cuentan con características tales como pantalla táctil, telefonía móvil, conexión Bluetooth, Wi-Fi, sistema de navegación GPS, cámara, reconocimiento de voz, grabadora, reproductor de música, etc. Las ventas mundiales de dispositivos móviles, especialmente de *smartphones*, crecen día a día. Según el último estudio de mercado de la compañía de análisis Gartner, se estima que 1.495 millones de *smartphones* han sido vendidos a usuarios en 2016 (Gartner), de los cuales el 81,7% utiliza un sistema operativo Android.

Este capítulo ofrece un resumen sobre los sistemas operativos móviles más comúnmente usados en el mercado y sus principales características. La elección del sistema operativo en el que se desarrolla la aplicación de lectura del gadget se decide usando este resumen como base. La elección del terminal no resulta de gran importancia, a excepción de que soporte el protocolo de comunicación elegido para el proyecto. Sin embargo la elección del sistema operativo es trascendental, pues de él dependen las características del programa y el proceso de desarrollo, además de los protocolos inalámbricos que soporta.

### ***2.3.1 Características de los diferentes SOs***

Existe una amplia variedad de sistemas operativos disponibles en el mercado para los diferentes dispositivos móviles, pero sólo se analizan los más usados populares. La Tabla 2-1 muestra una comparación de las ventas y cuota de mercado de los cuatro sistemas operativos más vendidos entre los años 2016 y 2017 (IDC Research, 2017).

Periodo	Android	iOS	Windows Phone	Otros
2016Q1	83.4%	15.4%	0.8%	0.4%
2016Q2	87.6%	11.7%	0.4%	0.3%
2016Q3	86.8%	12.5%	0.3%	0.4%
2016Q4	81.4%	18.2%	0.2%	0.2%
2017Q1	85.0%	14.7%	0.1%	0.1%

Tabla 2-1. Cuota de mercados de los diferentes SO móviles

### 2.3.1.1 Sistema Operativo Android

Android es un completo set de software para dispositivos móviles, el cual incluye un sistema operativo, middleware (software de intercambio de información entre aplicaciones) y aplicaciones móviles clave. El sistema operativo Android está basado en el Kernel de Linux y está diseñado principalmente para dispositivos móviles con pantalla táctil, tanto *smartphones* como *tablets*. Android fue mostrado por primera vez en 2007, a la vez que la fundación de la *Open Handset Alliance*: un consorcio de compañías de hardware, software y telecomunicaciones dedicadas al avance de los estándares abiertos de los dispositivos móviles. El terminal HTC Dream, lanzado en octubre de 2008, fue el primer dispositivo móvil con Android como sistema operativo. El código fuente de Android fue liberado por Google bajo licencia *Apache*, la cual permite que el software se pueda modificar libremente y distribuirlo por fabricantes, operadoras de telecomunicaciones y desarrolladores entusiastas.

Android posee el mayor número de aplicaciones o *apps* disponibles para descargar en la tienda *Google Play*: cerca de 1 millón de aplicaciones y sobre 50.000 millones de descargas. Android es la plataforma de desarrollo más utilizada con un 71% de los desarrolladores de aplicaciones móviles trabajando en ella (Ltd). En mayo de 2012, Android se convirtió en el sistema operativo móvil más popular, y líder de mercado en la mayoría de países, incluyendo EEUU (Lisa Mahapatra). Desde el primer lanzamiento en septiembre de 2008, Android ha sufrido numerosas actualizaciones que ha mejorado el sistema operativo a través de añadir nuevas características y arreglar errores. La última versión lanzada es la versión 8.0, conocida como Oreo, lanzada en agosto de 2017 (Wikipedia). La Tabla 2-2 muestra las diferentes versiones de Android más utilizados con su nombre, fecha de lanzamiento y nivel de la API (Wikipedia: Android (operating system)).

Versión	Nombre	Fecha de lanzamiento	Nivel de API	Distribución
8.0	Oreo	Agosto 2017	26	0%
7.0 – 7.1.2	Nougat	Junio 2016	24 – 25	13.5%
6.0 – 6.0.1	Marshmallow	Octubre 2015	23	32.3%
5.0 – 5.1.1	Lollipop	Noviembre 2014	21 -22	29.2%
4.4 – 4.4.4	KitKat	Octubre 2013	19	16.0%
4.1 – 4.3.1	Jelly Bean	Julio 2012	16 – 17 - 18	6.5%
4.0 – 4.0.4	Ice Cream Sandwich	Diciembre 2011	15	0.7%

Tabla 2-2. Distribución de las versiones del SO Android

La arquitectura de Android está distribuida en diferentes capas, las cuales se describen a continuación:

- *Applications* (Aplicaciones): Las aplicaciones bases incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.
- *Application Framework* (Marco de trabajo de aplicaciones): Los desarrolladores tienen acceso completo a los mismos APIs del *framework* usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del *framework*). Este mecanismo permite que los componentes sean reemplazados por el usuario.
- *Libraries* (Bibliotecas): Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del *framework* de aplicaciones de Android. Las bibliotecas escritas en lenguaje C/C++ incluyen un administrador de pantalla táctil (*surface manager*), un *framework Open Core* (para el aprovechamiento de las capacidades multimedia), una base de datos relacional *SQLite*, una API gráfica *OpenGL ES 2.0 3D*, un motor de renderizado *WebKit*, un motor gráfico SGL, el protocolo de comunicación segura SSL y una biblioteca estándar de C, llamada “*Bionic*” y desarrollada por Google específicamente para Android a partir de bibliotecas estándar “*lib*” de BSD.
- *Android Runtime* (Funcionalidad en tiempo de ejecución): Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en



las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual *Dalvik*. *Dalvik* ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. *Dalvik* ejecuta archivos en el formato *Dalvik Executable* (.dex), el cual está optimizado para memoria mínima.

- *Linux Kernel* (Núcleo Linux): Android dispone de un núcleo basado en *Linux* para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores, y también actúa como una capa de abstracción entre el *hardware* y el resto de la pila de *software*.

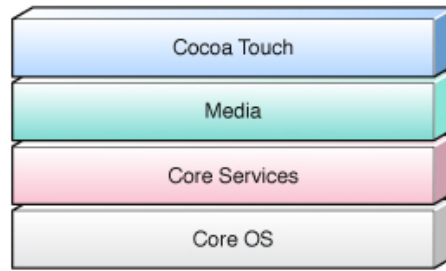
La gestión de memoria en *Android* ha sido diseñada de manera que la gestione eficientemente, gestionando automáticamente las aplicaciones almacenadas en memoria, lo cual asegura un consumo de energía mínimo. El sistema suspende automáticamente las aplicaciones de la memoria cuando no han sido utilizadas durante un periodo de tiempo para así ahorrar energía de la batería y poder de procesamiento.

Seguridad y privacidad: En Android, las aplicaciones son ejecutadas en un sandbox (caja de arena, simulación de un sistema operativo) para restringir el acceso de estas a recursos del sistema sin el permiso del usuario. El usuario puede aceptar o rechazar los permisos en el momento de la instalación de la aplicación. La versión de Android 4.2 Jelly Bean lanzada en 2012 incluye un scanner de malware integrado a Google Play, el cual puede escanear aplicaciones instaladas de terceras partes así como alertar al sistema. La naturaleza Open source de Android permite los usuarios y desarrolladores modificar las características de seguridad de acuerdo a sus requerimientos. Los problemas con los sistemas operativos Android provienen de algunos fabricantes, que deciden agregar una interfaz de usuario alternativa la cual reduce la consistencia del OS, y también de las aplicaciones que se instalan sin ser validadas.

### ***2.3.1.2 iOS***

iOs es un sistema operativo móvil desarrollado y distribuido por Apple Inc. Conocido previamente como iPhone OS, fue desvelado en 2007 para el iPhone. Tiene un 13,9% de la cuota de los sistemas operativos de *smartphones* vendidos en 2015, por detrás de Android de Google (IDC Research, 2017). iOs es la versión móvil del sistema operativo de Apple OS X, con el que comparte su base en Darwin y varios *frameworks* de aplicaciones. La última versión lanzada en marzo de 2017, corresponde al iOs 10.3 (Wikipedia: iOs).

El sistema operativo de Apple está distribuido en cuatro capas diferenciadas por su funcionalidad:



**Ilustración 2-4. Arquitectura de capas iOS**

- *Cocoa Touch*: Es la capa superior, la que los usuarios utilizan para interactuar con las aplicaciones, es decir, la capa visible. Es la zona donde nos encontramos los componentes visuales, se trata de una capa de abstracción.
- *Media*: Se trata de una capa basada en la mezcla de lenguaje C y Objective C que contiene las tecnologías que dan acceso a ficheros multimedia relacionados con audio, gráficos, vídeos, etc.
- *Core Services*: Se trata de la capa de servicios principales disponibles en el dispositivo y que pueden ser utilizados por todas las aplicaciones, como pueden ser: base de datos SQLite, acceso a la red, soporte para XML.
- *Core OS*: El núcleo del sistema. Recordar que el sistema operativo iOS está basado en el OS X de Apple, que fue desarrollado a partir de una base Unix. Elementos de seguridad, memoria, procesos o manejo de ficheros son los que podemos encontrar en esta capa.

*Gestión de memoria*: iOS utiliza el método *Reference counting* (conteo por referencias) para la gestión de memoria. En este método cada objeto lleva una cuenta de cuantos otros objetos se están utilizando, y cuando la cuenta llega a cero, el objeto es desasignado y la memoria se libera. Cuando en la memoria del dispositivo queda poco almacenamiento, todas las aplicaciones en iOS reciben una notificación de memoria baja para liberar cualquier tipo de memoria que no estén utilizando en ese instante. Si la condición de memoria insuficiente persiste, el sistema puede finalmente cerrar algunas aplicaciones.

*Seguridad y Privacidad*: Las características del iOS ayudan a proteger la información personal encriptando automáticamente los mensajes de e-mail y las aplicaciones de terceras partes usando un código. iOS aporta privacidad bloqueando cookies en la navegación web y previniendo el rastreo en sitios web. La iOS 7.0.6 aborda vulnerabilidades en el tiempo de manipulación de comunicaciones encriptadas, lo que permitía interceptar, leer o modificar emails encriptados, búsquedas web, tweets, y otros datos transmitidos- Las aplicaciones que

son desarrollados para usarse en iOS han de ser aprobadas por Apple previamente antes de estar disponibles en el mercado. iOS tampoco tiene soporte para Adobe Flash.

### ***2.3.1.3 Windows***

Windows es el sistema operativo de ordenador de escritorio más popular. Los últimos cinco años ha comenzado a prestar más atención a los sistemas operativos de los dispositivos móviles. Microsoft diseñó Windows CE (Compact Edition) específicamente para dispositivos de mano, basado en el API de Windows. El último sistema operativo móvil lanzado es Windows 8.1, en abril de 2014, soporta una gran cantidad de características, como soporte a procesadores multi núcleo, resolución de pantalla hifi, gran capacidad de almacenamiento y comunicaciones de corto alcance. Este sistema operativo simula en gran medida la versión de los ordenadores de escritorio de Windows 8. La próxima versión Windows 10 Mobile estará disponible en el mercado a finales de 2016. El punto fuerte de esta nueva versión reside en que mientras se esté ejecutando en tu dispositivo móvil con una pantalla pequeña optimizada, aún se ejecutará como Windows 10. Por lo que aplicaciones y las características serán iguales tanto en el dispositivo móvil como en el ordenador. Se podrán utilizar las versiones completas de Office, Word y PowerPoint se ejecutarán igual que en la versión de escritorio- La configuración de pantalla se verá y operara de la misma manera entre dispositivos y las aplicaciones serán universales, por lo que se podrá cambiar entre dispositivos instantáneamente.

## ***2.4 Protocolos inalámbricos.***

### ***2.4.1 Introducción a las Redes WPAN***

Las redes inalámbricas de área personal (WPAN, Wireless Personal Area Network) son redes que comúnmente cubren distancias del orden de los 10 metros como máximo, normalmente utilizadas para conectar varios dispositivos portátiles personales sin la necesidad de utilizar cables. Esta comunicación de dispositivos peer-to-peer normalmente no requiere de altos índices de transmisión de datos. La tecnología inalámbrica Bluetooth, por ejemplo, tiene un índice nominal de 10 metros con índices de datos de hasta 1 Mbps. El tipo de ámbito y los relativos bajos índices de datos tienen como resultado un bajo consumo de energía haciendo a la tecnología WPAN adecuada para el uso con dispositivos móviles pequeños, que funcionan con baterías, tales como teléfonos inteligentes, dispositivos cuantificadores de salud o cámaras digitales (Camargo Olivares).

### ***2.4.2 Bluetooth Classic***

Bluetooth (Bluetooth SIG) es una tecnología utilizada para la conectividad inalámbrica de corto alcance entre dispositivos como *smartphones*, teclados, impresoras, portátiles, módems, proyectores, etc. El principal mercado es la transferencia de datos y voz entre dispositivos y computadoras personales; es una tecnología de radiofrecuencia (RF) que opera en la banda de 2,4 GHz y utiliza salto de frecuencia para expansión del espectro. Opera con una tasa binaria máxima de 720 Kbps.

La distancia de conexión en Bluetooth puede ser de hasta 10 metros o más, dependiendo del incremento de la potencia del transmisor, pero los dispositivos no necesitan estar en línea de vista, ya que las señales de RF pueden atravesar paredes y otros objetos no metálicos sin problema.

Bluetooth puede ser usado para aplicaciones en redes residenciales o en pequeñas oficinas, es decir ambientes de tipo *WPAN*.

Este estándar define una conexión inalámbrica que permite la transmisión de voz y datos entre diferentes equipos mediante un enlace por radiofrecuencia. Los principales objetivos que se pretende conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

La tecnología Bluetooth comprende hardware, software y requerimientos de interoperabilidad, por lo que para su desarrollo ha sido necesaria la participación de los principales fabricantes de los sectores de las telecomunicaciones y la informática. Dichas empresas crearon a inicios de 1998 un grupo de interés especial denominado SIG, cuyo propósito fue establecer un estándar para interfaz aérea junto con un software de control, con el fin de asegurar la interoperabilidad de los equipos entre diversos fabricantes.

### ***2.4.3 Bluetooth Low Energy***

*Bluetooth Low Energy* (BLE, también comercializado como Bluetooth Smart) comenzó como parte de la especificación Bluetooth 4.0 (Bluetooth SIG).

Originalmente diseñado por Nokia como Wibree antes de ser adoptado por el Bluetooth Special Interest Group (GSIP, en español Grupo de Especial Interés en Bluetooth), los desarrolladores no estaban intentando sugerir otra solución inalámbrica capaz de resolver todos los posibles problemas de las conexiones inalámbricas. Desde el principio el foco se centró en diseñar un estándar de radio con el menor consumo de energía posible, específicamente optimizado para el bajo coste, bajo ancho de banda, baja potencia, y baja complejidad.

Estas metas de diseño son evidentes en las especificaciones fundamentales, las cuales contemplan hacer del BLE un estándar de baja potencia, diseñado para que se pueda implementar actualmente por los fabricantes de componentes y que se integre fácilmente en las aplicaciones actuales. Posiblemente sea el primer estándar ampliamente adoptado que realmente proclama que puede ser usado durante un largo periodo de tiempo desde una batería tipo botón.

Para un estándar relativamente joven (fue introducido en 2010), BLE ha tenido un tasa de adopción muy rápida, y el número de productos diseñados que ya incluyen BLE lo sitúa en la cabeza junto a otras tecnologías inalámbricas con mayor tiempo de lanzamiento. Comparado con otras tecnologías inalámbricas el crecimiento del BLE es relativamente sencillo de explicar: BLE está ligado a la computación móvil, tablets y smartphones, y por tanto a su alto crecimiento. Una temprana y activa adopción por parte de algunas compañías importantes dentro de la industria móvil como Apple o Samsung abrió las puertas para una amplia implementación de BLE. Apple, en particular, ha hecho un esfuerzo significativo produciendo una pila BLE fiable y publicando líneas generales de diseño en torno al BLE. Esto empujó a los fabricantes de componentes a asignar sus limitados recursos a la tecnología que mayor tasa de éxito y rentabilidad supusieran a la larga, y el sello de aprobación de Apple a esta tecnología fue un gran argumento a su favor.

Mientras los mercados de tablets y smartphones van incrementando su madurez y los costes y los márgenes van decreciendo, la necesidad de conectividad con el mundo exterior en estos dispositivos ha crecido exponencialmente, y esto ofrece a los vendedores de periféricos una oportunidad única de proveer innovadoras soluciones a problemas de los que las personas no se habían dado cuenta hasta ahora.

Todos esos beneficios han convergido en torno al BLE, y se ha abierto una gran oportunidad para los pequeños desarrolladores de productos al ganar acceso un potencial mercado masivo con productos creativos, innovadores, destinados a tareas específicas y a un modesto presupuesto de diseño. Actualmente se pueden comprar sistemas SoC (System On a Chip, un microcontrolador con todo el sistema incorporado) a precios y volúmenes por debajo de los que permiten otras tecnologías como WiFi, GSM, ZigBee, y BLE permite

diseñar productos viables que a día de hoy pueden comunicarse con cualquier plataforma móvil moderna usando herramientas y estándares fácilmente accesibles.

Una diferencia fundamental con el Bluetooth clásico, el cual está enfocado en un conjunto de situaciones, es que BLE está concebido para permitir que cualquiera pueda hacer un intercambio de datos desde un accesorio, sin tener gran conocimiento acerca de las capas inferiores de la tecnología. Los fabricantes de smartphones también proveen una flexible y relativamente de bajo nivel APIs para dar a los desarrolladores de aplicaciones móviles la libertad de usar el framework BLE.

Con un modelo de datos relativamente sencillo de entender, costes de licencia no intrusivos, sin tasas para el acceso a las core specs, y un una pila de protocolo simple en general, queda expuesto porqué BLE es uno de los estándares más elegido en la actualidad.

### ***2.4.3.1 La especificación (Specification of the Bluetooth System)***

En Junio de 2010, Bluetooth SIG introdujo el protocolo BLE con la versión 4.0 de las Bluetooth Core Specification (Especificaciones fundamentales Bluetooth).

La primera actualización, Bluetooth 4.1, fue lanzada en Diciembre de 2013. Aunque los bloques básicos de construcción, procedimientos, y conceptos permanecen intactos, esta actualización también introdujo múltiples cambios y mejoras para suavizar la experiencia del usuario.

En Diciembre de 2014 se lanza la actualización 4.2 del protocolo, conocido como *Bluetooth Smart*. Los dispositivos que utilizan esta versión del protocolo son retrocompatibles con las versiones 4.0 y 4.1, y las características de bajo consumo de energía, por lo que Bluetooth SIG recomienda implementar este protocolo en todos los nuevos diseños. Algunas mejoras que introduce esta versión del protocolo son

- Capacidades de IOT: Low-power IP (IPv6/6LoWPAN) y Bluetooth Smart Internet Gateways (GATT), permitiendo a los sensores con BLE 4.2 transmitir datos a través de Internet.
- Seguridad: LE Privacy y LE Secure Connections son características adicionales que permiten rastrear la localización del dispositivo y emparejamiento sólo a usuarios de confianza.
- Velocidad: Hay cambios en la longitud de los paquetes de datos que permiten una transmisión de datos 250% más rápida y 10 veces más capacidad de paquetes.

### 2.4.4 Bluetooth 5

La última actualización del protocolo Bluetooth es la versión 5.0, anunciada en junio de 2016, aunque aún tiene una adopción muy minoritaria. Las características clave de esta nueva versión de baja energía son (Bluetooth SIG, 2016):

- Incremento del doble de ancho de banda con respecto a BLE 4.2, manteniendo el bajo consumo.
- Aumento en cuatro veces el rango de alcance.
- Mayor capacidad en los mensajes de *broadcasting*, permitiendo paquetes ocho veces mayores.
- Prevención de interferencias en la misma banda y en bandas vecinas.
- Esta versión es retrocompatible con las otras versiones *Low Energy*.

### 2.4.5 ZigBee

El término ZigBee describe un protocolo inalámbrico normalizado para la conexión de una Red de Área Personal Inalámbrico o WPAN (Zigbee alliance).

ZigBee es diferente de los otros estándares inalámbricos, ha sido diseñado para soportar un diverso mercado de aplicaciones con una conectividad más sofisticada que los anteriores sistemas inalámbricos. El estándar enfoca un segmento del mercado no atendido por los estándares existentes, con baja tasa de transmisión de datos, bajo ciclo de servicio de conectividad y bajo costo.

La razón de promover un nuevo estándar, es para permitir la interoperabilidad entre dispositivos fabricados por compañías diferentes. ZigBee es un estándar donde el estándar IEEE 802.15.4 sólo contempla las capas PHY (Physical Layer) y MAC (Medium Access Control); las capa NWK (Network Layer) y APS (Application Layer) han sido establecidas por la Alianza ZigBee.

La arquitectura ZigBee debe permitir el diseño fácil y el desarrollo de los dispositivos baratos y de baja potencia. La interoperabilidad debe ser considerada como una de las razones principales para la estandarización por lo que la arquitectura debe definir el stack de tal manera de que la terminología esté normalizada. La arquitectura también debe permitir versiones actualizadas y extensiones en el futuro.

ZigBee se ha implementado en la banda mundial de 2,4 GHz, sin necesidad de licencias, o en una de las bandas regionales de 868/915 MHz. El espectro de radio sin

licencia, está designado por un acuerdo internacional y pone la carga de adhesión de la especificación sobre el fabricante del equipo.

La banda de 2,4 GHz es la preferida porque es una banda libre de licencias, y porque su uso es a nivel internacional. Hay muchas bandas sin licencia en las frecuencias más altas y más bajas. Las bandas de 2,4 GHz y 868/915 MHz fueron escogidas por el estándar IEEE 802.15.4 debido a sus características de propagación. Las frecuencias 868/915 MHz y 2,4 GHz tienen buena penetración tanto a través de paredes como de techos, pero tienen un rango limitado. La limitación de rango es realmente deseable para reducir las interferencias.

El estándar ZigBee permite una transferencia de datos de 250 Kbps y de 20 Kbps con bajo consumo energético. Esto representa la cantidad de datos que puede ser transferida cuando la cabecera de la trama de datos se ha retirado.

El hardware ZigBee puede comunicarse sobre un rango entre 10 a 75 metros. Un hardware típico a 2,4 GHz presenta una distancia de trabajo hasta 30 metros dentro de un edificio y más de 100 metros en campo abierto.

ZigBee permite que las redes manejen hasta  $2^{16}$  dispositivos. Este atributo es fundamental para la creación de series masivas de sensores y redes de mando.

## ***2.4.6 WiFi***

WiFi es una tecnología de comunicación inalámbrica ampliamente extendida en el mercado, destinada a conectar todo tipo de dispositivos electrónicos en una red local sin la necesidad de cables físicos.

### ***2.4.6.1 Historia de WiFi***

Sus orígenes se remontan a 1971, cuando un grupo de investigadores diseñaron la primera red de área local inalámbrica (WLAN), a la cual se la denominó ALOHAnet. Esta red utilizaba ondas de radio UHF para realizar las conexiones entre los ordenadores situados en las distintas islas hawaianas. En 1985, la comisión de las comunicaciones de Estados Unidos introdujo las bandas ISM, las cuales designaban las bandas de frecuencia de uso no comercial en las que se debía trabajar para poder realizar comunicaciones inalámbricas de tipo Industrial, Científico y Médico.

Ya en 1991, NCR y AT&T crearon las bases de lo que posteriormente sería el estándar 802.11, el cual define la normativa referente al funcionamiento de la capa física y la capa de enlace de datos en las redes WLAN. En aquella época, las velocidades de



transmisión de datos eran muy inferiores a las utilizadas actualmente, pero gracias a una tecnología desarrollada por John O'Sullivan para una aplicación relacionada con la astrofísica, se consiguió aumentar la velocidad de las transmisiones.

Fue en 1997 cuando el IEEE por fin anunció oficialmente el estándar 802.11, en torno al cual se creó en 1999 una asociación sin ánimo de lucro llamada WECA, la cual fomentaba el desarrollo de dispositivos electrónicos compatibles con el estándar 802.11 del IEEE.

En el 2002, WECA pasó a denominarse *WiFi Alliance*, y desde entonces cualquier producto que quiera llevar el logo que indique que el producto ha pasado el proceso de certificado de la WiFi Alliance, debe cumplir el estándar 802.11 marcado por el IEEE, los estándares de seguridad WPA y WPA2, y el estándar de identificación EAP.

#### ***2.4.6.2 Características de WiFi***

El estándar 802.11, en el que se basa la tecnología WiFi (Wikipedia) ha ido experimentando modificaciones con el paso de los años, ofreciendo nuevas características que se adecuan a las nuevas necesidades de los dispositivos que incorporan la tecnología WiFi, las principales versiones de este protocolo son:

- IEEE 802.11b: trabaja en la banda de los 2,4 GHz y alcanza una velocidad de 11 Mbps. El espectro electromagnético de este protocolo se divide en 14 canales que se superponen, dando opción a que convivan distintas redes WiFi en el mismo espacio.
- IEEE 802.11g: trabaja en la banda de los 2,4 GHz, alcanza una velocidad de 54 Mbps. El espectro electromagnético de este protocolo se divide en 14 canales que se superponen.
- IEEE 802.11n: trabaja en la banda de los 2,4 GHz y opcionalmente en la de 5 GHz, alcanza una velocidad de entre 54 Mbps y 600 Mbps. El espectro electromagnético de este protocolo se divide en 14 canales que se superponen.
- IEEE 802.11ac: trabaja en la banda de los 5 GHz, alcanza una velocidad de 1300 Mbps, aunque al trabajar en una frecuencia mayor, la señal pierde un poco de alcance. El espectro electromagnético de este protocolo se divide en 23 canales que no se superponen.

Aunque el espectro electromagnético del WiFi se divide en varios canales, las distintas redes WiFi presentes en la misma zona pueden compartirlos sin interferirse entre ellas, ya que aunque por los canales haya más paquetes de datos, como cada red tiene su propio SSID (*Service Set Identifier*), el cual consiste en 32 bytes que identifican una determinada red, los dispositivos sólo tienen que ignorar los paquetes que no vayan dirigidos al SSID de la red a la que pertenecen.

En cuanto al alcance de la señal, un punto de acceso que cumpla con los estándares 802.11b y 802.11g puede alcanzar unos 100 metros de alcance, por lo que el WiFi se convierte en un buen sustituto de las comunicaciones cableadas en áreas locales.

La seguridad de los datos transmitidos inalámbricamente corre a cargo de los estándares de encriptación de datos WEP, WPA y WPA2. WPA utiliza el protocolo TKIP, el cual refuerza la seguridad proporcionada por WEP, aunque se ha comprobado que este protocolo también presenta vulnerabilidades, por lo que finalmente se recomienda utilizar WPA2, el cual utiliza AES (*Advanced Encryption Standard*) para proteger la red.

Por sus características, WiFi está presente en casi cualquier producto del mercado que necesite comunicarse inalámbricamente con otro dispositivo electrónico. Es el caso de los ordenadores de sobremesa, los portátiles, los smartphones, las tabletas, las Smart TV, las neveras, las bombillas inteligentes y todo tipo de productos que necesitan acceder a una red local para transmitir datos inalámbricamente. Es compatible con prácticamente todos los sistemas operativos actuales, tanto a nivel de escritorio, como son Windows, Mac OS X y Linux, como a nivel de dispositivos móviles como pueden ser Android, iOS y Windows Phone.

### ***2.4.7 Conclusión***

La decisión del protocolo, definirá el hardware a utilizar.

*Bluetooth Low Energy* cumple todas las necesidades de un gadget personal enfocado a la monitorización cardíaca. Su implantación está suficientemente extendida entre los smartphones, razón por la que se impone a otros protocolos de baja energía como ZigBee.

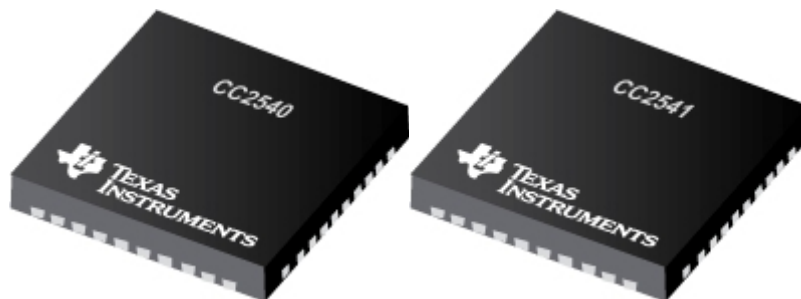
Bluetooth clásico y WiFi también son protocolos válidos, pero su consumo energético es mucho mayor, y aunque las tasas de transmisión de datos son mucho mayores, la tasa del BLE es suficiente.

## ***2.5 Módulos y Kits de desarrollo***

En la mayoría de los casos que implican un uso del protocolo BLE, el dispositivo con el que interactuar actúa como periférico en lugar de como central (este rol lo asumen los smartphones, tablets u ordenadores portátiles). En la actualidad existen diversos módulos SoC (System-On –Chip) especialmente diseñados para aplicaciones basadas en BLE (económico y bajo consumo de energía).

### ***2.5.1 Texas Instruments CC254X***

El CC2540 y su versión mejorada, el CC2541, han sido muy populares durante varios años. Ambos dispositivos incluyen un procesador 8051 que permite ejecutar aplicaciones sin un microcontrolador externo, con lo cual el coste del sistema disminuye. Aunque esto también tiene un inconveniente común en muchas soluciones integradas y es que la aplicación está atada a este procesador, y en algunos casos los productos requieren más desempeño.



**Ilustración 2-5. SoCs de la familia CC254x de Texas Instruments**

Algunas de las especificaciones del componente son:

- Sistema completo que integra BLE, microcontrolador y gestión de periféricos.
- Microcontrolador 8051 con 256 kB o 128 kB de memoria flash y 8 kB de RAM.
- Empaquetamiento 6x6 mm QFN.
- Máxima potencia de salida de 0 dBm.
- Consumo de corriente RX de 17,9 mA a 14,7 mA; y TX 18,2 mA a 14,3 mA.
- Sensibilidad del receptor -94 dBm a 1 Mbps.

Además, el dispositivo se completa con diversos periféricos para dar cobertura a todo tipo de productos.

Un aspecto importante de los dispositivos CC2540 y 2541 es el kit de herramientas de desarrollo. La mayoría de los chips actuales de BLE son pequeños dispositivos de radio con software especializado, por lo tanto es este firmware el que gestiona algunas de las funcionalidades de bajo nivel del BLE. Texas Instruments provee con frecuencia nuevas y mejoradas versiones de firmware. El desarrollo las aplicaciones está ligado al entorno de desarrollo IAR Workbench 8051.

El dispositivo CC2541, además de radio BLE, puede ejecutar otros protocolos propietarios que el CC2540 no es capaz de ejecutar. Otras diferencias entre ambos dispositivos son:

- La corriente Rx en el modo de menor consumo se reduce de 19,6mA a 17,9mA.
- CC2541 no tiene interfaz USB.
- CC2541 tiene interfaz I<sup>2</sup>C.
- CC2541 puede transmitir a una potencia de 0 dBm, frente a los 4 dBm del CC2540.

### 2.5.1.1 Kits de desarrollo

Texas Instruments ha desarrollado un grupo de kits de desarrollo y ejemplos con los que implementar y desarrollar soluciones con chips de la familia CC254x:

CC2541 Mini Development Kit	CC2541 Sensor Tag Development Kit	CC2540 Development Kit	CC2541 Advanced Remote Control Kit
 <p>Incluye un <i>dongle</i>, <i>keyfob</i> y el <i>debugger</i>, para facilitar el desarrollo y análisis de aplicaciones.</p>	 <p>Incluye una gran variedad de sensores y un entorno de trabajo para crear <i>apps</i> para <i>smartphones</i>.</p>	 <p>Una completa plataforma de pruebas y desarrollo de software para <i>BLE</i> para prototipado.</p>	 <p>Diseño de referencia para aplicaciones de control mediante <i>BLE</i>.</p>

Tabla 2-3. Kits de desarrollo de la familia CC254x (Texas Instruments)

## ***2.5.2 Texas Instruments CC26xx***

Normalmente, el microprocesador del dispositivo comparte tiempo y recursos entre la aplicación y el protocolo de comunicación BLE. En el caso de la familia CC26XX, existe un microprocesador cortex-M0 ejecutando las capas más bajas del protocolo BLE, incluyendo la capa física y de enlace. A este dispositivo se ha añadido un CORTEX-M3 que ejecuta la aplicación, las capas superiores del protocolo y algo de la capa de enlace.

Un microprocesador Cortex-M3 es más potente y puede ejecutar mejor algoritmos complejos, usados por ejemplo en monitores de fitness, controles industriales y automatización del hogar, por lo que se puede evitar en algunos casos incorporar un procesador externo.

Respecto a su predecesor, el CC2540/2541, el cuál su consumo de energía era demasiado alto, ha disminuido considerablemente este consumo. El CC26xx también tiene un controlador específico de periféricos diseñado para descargar de procesamiento el microprocesador y por lo tanto reduce el consumo.

Algunas de las especificaciones del dispositivo son:

- Microprocesador ARM® Cortex™-M3 48-MHz con 128 KB de memoria flash y 20 KB de SRAM.
- Voltaje de alimentación de 1,8V-3,8V.
- Versión Bluetooth 5, la versión más actual del protocolo.
- Sensibilidad -97dBm Rx.
- Controlador de periféricos independiente.
- Encapsulados disponibles: QFN 7x7 mm y WLCSP 3,9x3,5 mm.
- Tiene disponible un IDE de desarrollo gratuito (Code Composer Studio).

### 2.5.2.1 Kits de desarrollo





CC2650 Launchpad Evaluation Board	CC2560 Sensor Tag Iot Kit	CC2650 Smart RF06 Development Kit	CC2650 remote control
 <p>Kit sencillo de desarrollo de dispositivos BLE</p>	 <p>Enfocado a productos Iot, conectados a la nube.</p>	 <p>Este kit incluye todo lo necesario para desarrollar y evaluar dispositivos BLE.</p>	 <p>Diseño de referencia para aplicaciones de control mediante BLE.</p>

Tabla 2-4. Kits de desarrollo de la familia CC26xx (Texas Instruments)

### 2.5.3 Nordic Semiconductor Serie nRF52



Ilustración 2-6. SoC de la serie nRF52 de Nordic Semiconductor

Esta familia es la versión mejorada de chips anteriores de Nordic Semiconductor, como la serie nRF8001 que requería microcontrolador externo o la Serie nRF51. Incluye soporte por hardware para Bluetooth 5, lo que le permite mayor ancho de banda y rango de alcance.

Los SoCs de la serie nRF52 incorporan un procesador Cortex-M4F para cumplir requerimientos de aplicaciones más demandantes en un solo chip. Soporta los protocolos Bluetooth 5 y Bluetooth Low Energy, así como ANT, un protocolo propio a 2,4GHz y emparejamiento mediante NFC.

Incorpora una gran variedad de periféricos para cubrir la mayor parte de requerimientos de diseño.

Trae capacidad NFC, que permite sacar ventaja de la funcionalidad ‘*touch-to-pair*’ o tocar para emparejar.

Pueden funcionar en una gama de voltajes entre los 1,7 y 3,6 V, y activar periféricos y funcionalidades según su necesidad, creando un dispositivo con consumo eficiente.

### 2.5.3.1 Kits de desarrollo




nRF52 DK	nRFready Smart Remote 3 for nRF52 Series	Power Profiler Kit
 <p>Kit de desarrollo versátil para BLE, ANT y 2,4 GHz propietario. Compatible con Arduino.</p>	 <p>Diseño de referencia de mando inteligente con entrada para voz y bajo consumo.</p>	 <p>Herramienta de medida de consumo y corriente para la optimización de sistemas embebidos.</p>

Tabla 2-5. Kits de desarrollo del nRF52 (Nordic Semiconductor)

### 2.5.4 Dialog Semiconductor DA14580

Este chip de la familia *SmartBand* no tiene memoria flash integrada, lo que disminuye el consumo de energía y abarata su coste, ya que la memoria flash requiere circuitos especializados y voltajes más altos. Algunas de sus características son:

- Versión de Bluetooth soportado BLE v4.2 Single Mode.
- Incorpora un microprocesador ARM® Cortex™-M0 que se ejecuta a 16 MHz, con 32 kB de memoria PROM (memoria de sólo lectura programable una sola vez).
- Voltaje de alimentación de 0,9V-3,6 V
- Impedancia de salida para RF de 50 Ohm
- Consumo de corriente en TX y RX de 4,9 mA.

Para el desarrollo, la aplicación y el protocolo se cargan en la memoria vía JTAG. El dispositivo puede conectarse a una memoria externa vía SPI o I<sup>2</sup>C para cargar el firmware. Aunque no es tan flexible como tener una memoria flash en el chip, puede ser usada para actualizaciones de firmware.

Puesto que la salida para RF ya está ajustada a 50 Ohms, no son necesarios componentes externos.

### 2.5.5 Bluegiga, EFR32™ Blue Gecko Bluetooth® Low Energy SoCs

Algunas de las especificaciones del dispositivo son:

- Microprocesador ARM® Cortex™-M4 40-MHz con 1 MB de memoria flash y 256 KB de SRAM.
- 2 modos de energía, consumo mínimo de 1,5 uA.
- Voltaje de alimentación de 1,8V-3,8V.
- Bluetooth Low Energy, protocolo propietario 2,4 GHz y radio Sub-GHz.
- Gran cantidad de periféricos digitales y analógicos.
- Software de desarrollo, Simplicity Studio.

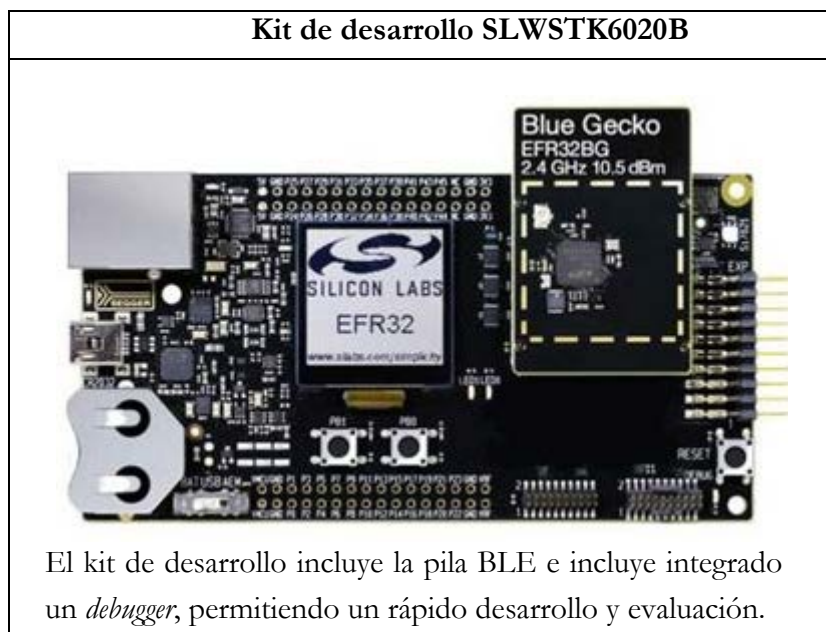


Tabla 2-6. Kit de desarrollo del ERF32 Blue Gecko (Silicon Labs)



## 2.5.6 Comparativa de características de diferentes SoCs

	Nordic Semi nRF51822	Texas Instruments CC2640	Texas Instruments CC2541	Dialog DA14580
Mode	Single Mode BLE v4.1 / ANT	Single Mode BLE v4.1	Single Mode BLE v4.0	Single Mode BLE v4.1
Maximum Tx Power	+4 dBm	+5 dBm	0 dBm	0 dBm
Rx Sensitivity	- 92 dBm	- 97 dBm	- 94 dBm	- 93 dBm
Integrated Processor	Cortex-M0	Cortex-M3 + Cortex-M0	8051	Cortex-M0
Flash	128kB / 256kB	128kB + ROM	128kB/256kB	32kB OTP + ROM
RAM	16kB / 32kB	20kB + 4kB	8kB	42kB + 8kB
Cache	No	8kB	No	No
Current Consumption - BLE Rx	9.5mA	5.9mA	14.3mA / 18.2mA	4.9mA
Current Consumption - BLE Tx @ 0dBm	8.1mA	6.5mA	14.7mA / 17.9mA	4.9mA
Current Consumption - Deep Sleep	400nA	100nA	500nA	600nA
Timers	3	4	3	2
GPIO	Up to 31	10, 15, 31	21	12, 22, 32
Interfaces	1x I2C UART	2x SPI I2C UART	2x USART/SPI I2C	2x UARTS SPI I2C
Analog	Temperature Sensor	Comparator, Current Source	Op-Amp	ADC
ADC	10-bit	12-bit 200ksps	12-bit	10-bit
Sensor Controller	No	Yes	No	No
Other	RTC, Quadrature Decoder	I2S, RTC, CapSense	PWM	Quadrature Decoder, Keyboard Controller

	CSR CSR101x	Cypress PSoC 4 BLE	ST BlueNRG- 1
Mode	Single Mode BLE v4.1	Single Mode BLE v4.1	Single Mode BLE v4.2
Maximum Tx Power	+7 dBm	+3 dBm	+8 dBm
Rx Sensitivity	- 92 dBm	- 92 dBm	- 88 dBm
Integrated Processor	16-bit RISC	Cortex-M0	Cortex-M0
Flash	64kB	128kB / 256kB	160kB
RAM	64kB	16kB / 32kB	24kB
Cache	No	No	No
Current Consumption - BLE Rx	16mA	15.6mA	7.3mA
Current Consumption - BLE Tx @ 0dBm	16mA	16.4mA	8.2mA
Current Consumption - Deep Sleep	600nA	150nA	5nA
Timers	1	4	??
GPIO	12	Up to 36	15
Interfaces	UART I2C/SPI	2x Comm Blocks	SPI I2C UART
Analog	ADC	4x OpAmps 2x Comparators	4x OpAmps 2x Comparators
ADC	10-bit	12-bit 1Msps	10-bit
Sensor Controller	No	Yes	No
Other	PWM	CapSense	PWM

Tabla 2-7. Comparativa de características técnicas de diferente SoCs, (GT-Tronics)

## *2.6 Conclusiones*

Tras la presentación de las diferentes alternativas de tecnologías y modelos que se puede encontrar en el mercado en la actualidad, se ha realizado una selección para la realización de este proyecto:

- Fotopletismografía, es la técnica seleccionada para monitorizar el ritmo cardíaco. Su integración implica bajo coste y la fácil adaptación a un gadget. Además, permite en un futuro analizar otras medidas.
- Android, como sistema operativo móvil donde desarrollar la aplicación al ser el más extendido, su gran compatibilidad, y al tener disponible una amplia documentación. Además es el SO con mayor disponibilidad a la hora de realizar el proceso para desarrollar y comprobar el funcionamiento de la aplicación.
- Bluetooth Low Energy, como ya se ha comentado anteriormente, debido a su bajo consumo, tasa de transferencia de datos suficiente y disponibilidad en gran cantidad de dispositivos. Si se desarrollara actualmente el proyecto Bluetooth 5 ofrece una gran alternativa.
- CC254X, como SoC encargado de adquirir la señal, transformarla y transmitirla. Entre los diferentes SoCs se ha elegido este por la disponibilidad de la documentación y de Kits de desarrollo dentro del departamento.



# Capítulo 3

---

## Descripción del Sistema

---

### *3.1 Introducción*

Durante el capítulo anterior se han presentado las principales alternativas que se han tenido en cuenta a la hora de realizar este proyecto.

En este capítulo se desarrollaran las tecnologías del capítulo anterior que han sido escogidas. Se explica en más detalle el funcionamiento de las diferentes tecnologías, la descripción en profundidad del funcionamiento del protocolo BLE, la configuración del kit de desarrollo y los pasos para desarrollar el firmware para la plataforma. Por último, también se expone el proceso de instalación del software necesario y de creación de una aplicación en el sistema operativo Android.

### *3.2 Descripción del sistema de medida de ritmo cardíaco*

Cuando el corazón late, una onda de presión se mueve a lo largo de las arterias a unos pocos metros por segundo (sensiblemente más rápido que la sangre realmente fluye). Esta onda de presión se puede sentir en la muñeca, pero también causa un aumento en el volumen de sangre en los tejidos, que puede ser detectado por un pletismógrafo (PI Training). A lo largo de los años, se han utilizado todo tipo de dispositivos, pero aquí se

describe un pletismógrafo de pulso fotoeléctrico, que es robusto y fácil de fabricar y que permitirá grabar el latido del corazón sin necesidad de realizar conexiones eléctricas directas al cuerpo.

El sensor consiste en una fuente de luz y un fotodetector; la luz se refleja a través de los tejidos y la variación en el volumen sanguíneo altera la cantidad de luz que cae sobre el detector. La fuente y el detector se pueden montar uno al lado del otro para observar cambios en la luz reflejada o en cualquier lado de un dedo o lóbulo de la oreja para detectar cambios en la luz transmitida. La disposición particular aquí utiliza una clavija de ropa de madera para sostener un diodo emisor de luz infrarrojo y un fototransistor emparejado. El filtro de infrarrojos del fototransistor reduce la interferencia de las luces fluorescentes, que tienen un gran componente de CA en su salida.

La señal medida mediante el receptor será una señal analógica, por lo que habrá que filtrarla adecuadamente para evitar los ruidos generados por los propios componentes electrónicos utilizados. Este filtrado va a consistir en una serie de filtros paso banda compuestos por resistencias, condensadores y amplificadores operacionales.

Una vez filtrada la señal, habrá que amplificarla a unos niveles adecuados para poder ser tratados por el *SoC*, por lo que se utilizarán una serie de etapas de amplificación compuestas por amplificadores operacionales.

Tras haber obtenido una señal analógica con unos niveles adecuados y sin demasiado ruido, hay que digitalizar dicha señal para poder detectar los latidos del corazón. Por lo que mediante un amplificador operacional en modo comparador, se comparará la señal analógica, ya filtrada, con una tensión de referencia. Esta tensión de referencia se fijará en unos milivoltios por encima del cero. De este modo, pequeñas variaciones por encima de 0 V serán filtradas, a su vez que cada vez que se produzca una pulsación, el amplificador operacional generará una señal a nivel alto, y el resto del tiempo una señal a nivel bajo, identificando con un pulso electrónico los pulsos cardíacos.

Esta configuración permite el cálculo de la frecuencia cardíaca “latido a latido” (instantánea). Una vez obtenidos los tiempos de dos picos adyacentes, restar el primero del segundo para dar el tiempo entre los dos latidos (intervalo entre latidos) y expresarlo en segundos. Dividiendo esto en 60 se obtiene la frecuencia cardíaca instantánea en latidos por minuto.

El problema más común cuando se utiliza un pletismógrafo de pulso es la influencia del movimiento, que la señal varíe bruscamente. Por lo que es necesario tomar la medida en un estado de reposo del sujeto.

### 3.3 Descripción del protocolo Bluetooth Low energy

#### 3.3.1 Introducción

Si bien los usuarios suelen interactuar directamente sólo con las capas superiores de la pila del protocolo BLE, es necesaria una descripción básica de la pila completa para comprender el funcionamiento de esta.

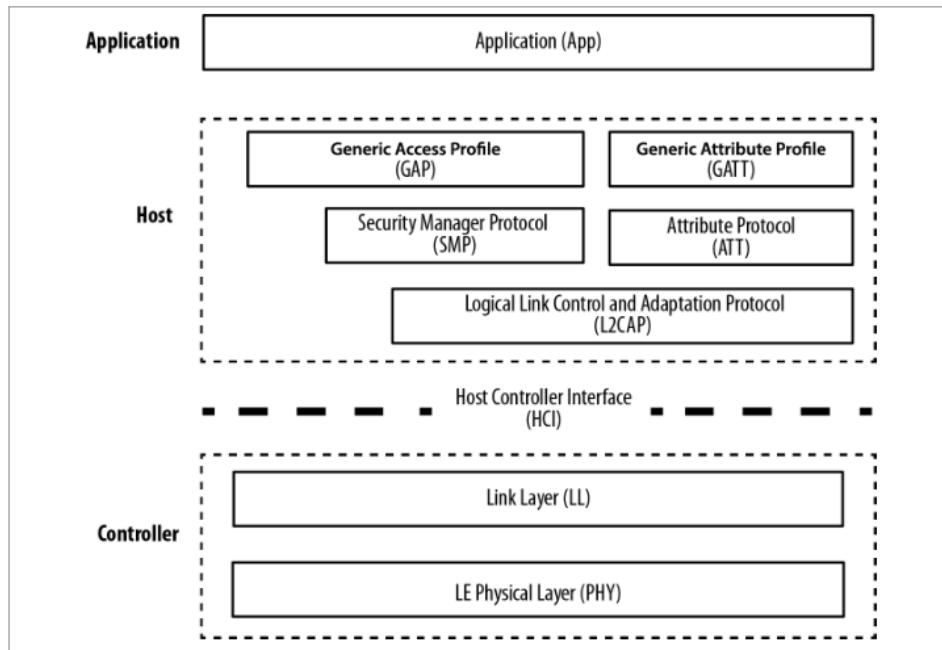


Ilustración 3-1. Pila del protocolo BLE (Townsend, Cufi, Akiba, & Davidson, 2014)

Como se muestra en la figura, un dispositivo BLE está dividido en tres partes: *controller*, *host* y *application*. Cada uno de estos bloques básicos de la pila del protocolo se separa en diferentes capas que proporcionan la funcionalidad requerida para operar:

**Application**, la aplicación, como en otros tipos de sistemas, es la capa de mayor nivel y la responsable de contener la lógica, la interfaz de usuario, y el tratamiento de datos de todo lo relacionado con la implementación de la aplicación. La arquitectura de la aplicación depende de cada implementación particular.

**Host**, incluye las siguientes capas:

- Generic Access Profile (GAP)
- Generic Attribute Profile (GATT)
- Logical Link Control and Adaptation Protocol (L2CAP)

- Attribute Protocol (ATT)
- Security Manager (SM)
- Host Controller Interface (HCI), Host side

**Controller**, incluye las siguientes capas:

- Host Controller Interface (HCI), Controller side
- Link Layer (LL)
- Physical Layer (PHY)

### ***3.3.2 Protocolos y perfiles***

Desde su creación, la especificación Bluetooth introdujo una clara separación entre los distintos conceptos de protocolos y perfiles (Townsend, Cufi, Akiba, & Davidson, 2014):

**Protocolos**, Los bloques de construcción utilizados por todos los dispositivos que conforman la especificación Bluetooth. Los protocolos son las capas que implementan los diferentes formatos de paquetes, enrutamiento, multiplexación, codificación y decodificación que permiten que los datos se envíen eficazmente entre pares.

**Perfiles**, "porciones verticales " de funcionalidad que abarcan los modos básicos de operación requeridos por todos los dispositivos (Perfil de Acceso Genérico o GAP, Perfil Genérico de Atributos o GATT) o casos de uso específicos (perfil de proximidad, perfil de glucosa, etc.). Los perfiles definen esencialmente cómo deben usarse los protocolos para lograr un objetivo particular, ya sea genérico o específico.

Los perfiles genéricos están definidos por la especificación y es importante entender cómo dos de ellos son fundamentales para asegurar la interoperabilidad entre dispositivos BLE de diferentes proveedores.

### ***3.3.3 Perfil Genérico de Acceso (GAP)***

La capa GAP del protocolo BLE es la responsable del manejo de los modos de acceso y procedimientos del dispositivo, incluyendo el descubrimiento de dispositivos, establecimiento de enlaces, finalización de enlaces, iniciación de características de seguridad y configuración del dispositivo. GAP es en esencia, la capa de control más alta de BLE. Este perfil es obligatorio para todos los dispositivos BLE, y todos deben cumplirlo.

La capa GAP trabaja en uno de estos 4 roles:

- **Broadcaster**, un *advertiser* al que no se puede conectar. (*Advertiser* podría traducirse como “anunciador”).
- **Observer**, escanea en busca de *advertisements*, pero no puede iniciar conexiones.
- **Peripheral**, un *advertiser* que es conectable, y opera como *slave* en una conexión simple en la LL (link layer).
- **Central**, escanea en busca de *advertisements* e inicia conexiones; opera como *master* en una o múltiples conexiones LL. Normalmente, un *central* puede soportar 3 conexiones simultáneas.

La especificación BLE permite ciertas combinaciones de múltiples roles. En un típico sistema BLE, el *peripheral* envía *advertisements* con datos específicos, mostrando a cualquier *central* que es conectable. Estos *advertisements* contienen la dirección del dispositivo, y pueden contener algunos datos adicionales como el nombre del dispositivo.

El *central*, una vez que recibe el *advertisement*, envía una “*scan request*” al *peripheral*. El *peripheral* responde con una “*scan response*”.

Este es el proceso de descubrimiento de dispositivos, en el que el *central* reconoce al *peripheral*, y sabe que puede establecer una conexión con él. El *central* puede enviar una petición para establecer un enlace con el *peripheral*.

### 3.3.3.1 *Parámetros de conexión*

La petición (*request*) de conexión contiene diferentes parámetros:

- **Connection Interval:** En la conexión BLE entre 2 dispositivos, se utiliza una *frequency-hopping* (frecuencia de salto), en la que dos dispositivos, cada uno envía y recibe datos de uno al otro por un canal específico, entonces cada cierto tiempo, ambos se “encuentran” en un nuevo canal (la LL del BLE maneja el cambio de canal). Este encuentro donde los dos dispositivos envían y reciben datos es conocido como “connection event”. Incluso si no hay datos de la *application* para ser enviados o recibidos, los dispositivos seguirán intercambiando datos de la LL para mantener la conexión. El *connection interval* puede variar desde un mínimo valor de 6 (7,5 ms) hasta un máximo de 3200 (4 s). Diferentes aplicaciones requieren un diferente *connection interval*. La ventaja de tener intervalos de conexión muy largos es que se ahorra bastante más energía.



La desventaja es que para enviar un dato hay que esperar hasta la siguiente conexión.

- **Slave Latency:** Este parámetro le da al *slave (peripheral)* la opción de saltarse un número de *connection events*. Esto le aporta cierta flexibilidad, por lo que si no tienen datos que mandar puede elegir saltarse *connection events* y permanecer “dormido” y por lo tanto gastando menos. La decisión es tomada por el *peripheral*. El valor del *slave latency* representa el máximo número de conexiones que pueden ser saltados. El rango va desde un mínimo de 0 (no se saltan) a un máximo de 500; sin embargo el máximo valor no puede hacer que el *effective connection interval* explicado más abajo) sea mayor a 16 s.
- **Supervision Timeout:** (Pausa de supervisión) Es la máxima cantidad de tiempo entre dos *connection events* correctos. Si esa cantidad de tiempo pasa sin un *connection event* exitoso, el dispositivo considera la conexión como perdida, y vuelve a un estado de desconectado. El valor del parámetro es representado en unidades de 10 ms. El *supervision timeout* puede ir desde un valor mínimo de 10 (100 ms) hasta 3200 (32 s). Además, el *supervision timeout* debe ser mayor que el *effective connection interval* (explicado abajo).

El “*effective connection interval*” es igual a la cantidad de tiempo entre dos *connections events*, asumiendo que el *slave* se salta el mayor número posible de eventos si la *slave latency* está permitida. Se puede calcular utilizando la fórmula:

$$\text{Effective Connection Interval} = (\text{Connection Interval}) \cdot (1 + (\text{Slave Latency}))$$

Ejemplo:

Connection Interval: 80 (100 ms)

Slave Latency: 4

Effective Connection Interval:  $(100\text{ms}) * (1 + 4) = 500\text{ms}$

Seleccionar le correcto grupo de parámetros de conexión juega un importante papel en la optimización del consumo. La siguiente tabla nos da un resumen de algunas compensaciones en la configuración de los parámetros de conexión:

Reducir el <i>connection interval</i>	<ul style="list-style-type: none"> <li>- Incrementa el consumo de ambos dispositivos.</li> <li>- Incrementa el rendimiento en ambas direcciones.</li> <li>- Reduce el tiempo que tarda un dato en ser enviado.</li> </ul>
Incrementar el <i>connection interval</i>	<ul style="list-style-type: none"> <li>- Reduce el consumo de ambos disp.</li> <li>- Reduce el rendimiento en ambas direcciones.</li> <li>- Incrementa el tiempo que toma enviar un dato.</li> </ul>
Reducir la <i>slave latency</i> (o configurarla a cero)	<ul style="list-style-type: none"> <li>- Incrementa el consumo del <i>peripheral</i>.</li> <li>- Reduce el tiempo que toma a los datos ser enviados desde el <i>central</i> al <i>peripheral</i>.</li> </ul>
Incrementar la <i>slave latency</i>	<ul style="list-style-type: none"> <li>- Reduce el consumo del <i>peripheral</i> cuando este no está enviando datos.</li> <li>- Incrementa el tiempo enviando los datos desde el <i>central</i> al <i>peripheral</i>.</li> </ul>

Tabla 3-1. Configuraciones de los parámetros de conexión

En algunos casos, el *central* solicitará un conexión con un *peripheral* con unos parámetros de conexión que serán desfavorables para el *peripheral*, en otros casos el *peripheral* quizás quiera cambiar los parámetros en medio de una conexión, basado en la aplicación del *peripheral*. El *peripheral* puede solicitar al *central* cambiar la configuración enviando un *Connection Parameter Update Request*. Esta petición es manejada por la capa L2CAP del *protocol stack*.

Esta petición contiene cuatro parámetros: *minimum connection interval*, *maximum connection interval*, *slave latency*, y *timeout*. Estos valores representan los parámetros que el *peripheral* desea para la conexión (el *connection interval* es dado como un rango). Cuando el *central* recibe la petición, este tiene la opción de aceptar o rechazar los nuevos parámetros.

Una conexión puede ser finalizada voluntariamente por cualquiera, *master* o *slave*, por cualquier razón. Un lado inicia la finalización, y el otro lado deber responder en consecuencia antes de que ambos dispositivos salgan del estado conectado.

### 3.3.3.2 Características de seguridad

El GAP también maneja la iniciación de características de seguridad durante la conexión BLE.

Ciertos datos pueden ser leídos o escritos sólo con un *authenticated connection*. Una vez que una conexión se crea, los dos dispositivos comienzan un proceso llamado *pairing*. Cuando el *pairing* actúa, se establecen *keys* las cuales encriptan y autentifican el enlace. En un caso típico, el *peripheral* solicitará que el *central* le provea un *passkey* para completar el proceso de *pairing*. Este *passkey* podría ser un valor fijo, como “000000”, o podría ser un valor generado aleatoriamente que se provee al usuario (tal como un display). Después de

que el *central* envíe el *passkey* correcto, los dos dispositivos intercambian las *keys* de seguridad para encriptar y autenticar el enlace.

En muchos casos, los mismos *central* y *peripheral* se conectarán regularmente entre ellos.

BLE tiene una característica de seguridad que permite a dos dispositivos, cuando realizan el *pairing*, darse entre ellos una configuración a largo plazo de *claves* de seguridad. Esta característica llamada *bonding*, permite a los dos dispositivos restablecer rápidamente una encriptación y autenticación después de reconectarse sin volver a realizar el proceso de *pairing* cada vez que se conectan.

### 3.3.4 Perfil Genérico de Atributos (GATT)

Al tratar con el intercambio de datos en BLE, el GATT define un modelo de datos básicos y procedimientos para permitir a los dispositivos descubrir, leer, escribir y empujar elementos de datos entre ellos. Es, en esencia, la capa de datos más alta de BLE.

La capa GATT de la pila de protocolo BLE está diseñada para ser usada por la *application* para la comunicación de datos entre dos dispositivos conectados. Desde el punto de vista del GATT, cuando dos dispositivos se conectan, cada uno asume un de estos dos roles:

- *GATT client*: Este es el dispositivos que está leyendo/escribiendo datos desde/hacia el *GATT server*.
- *GATT server*: Este es el dispositivo que contiene los datos que están siendo leídos/escritos por el *GATT client*.

Es importante darse cuenta que los roles del *GATT client* y *server* son completamente independientes de los roles de la LL (link-layer) *slave* y *master*, o de los roles del GAP *peripheral* y *central*. Tanto un *slave* como un *master* pueden ser ambos *GATT client* o *GATT server*.

#### 3.3.4.1 Jerarquía de datos GATT

Un *GATT server* consta de uno o más **services** GATT (servicios), los cuáles son colecciones de datos que cumplen un función o característica particular.

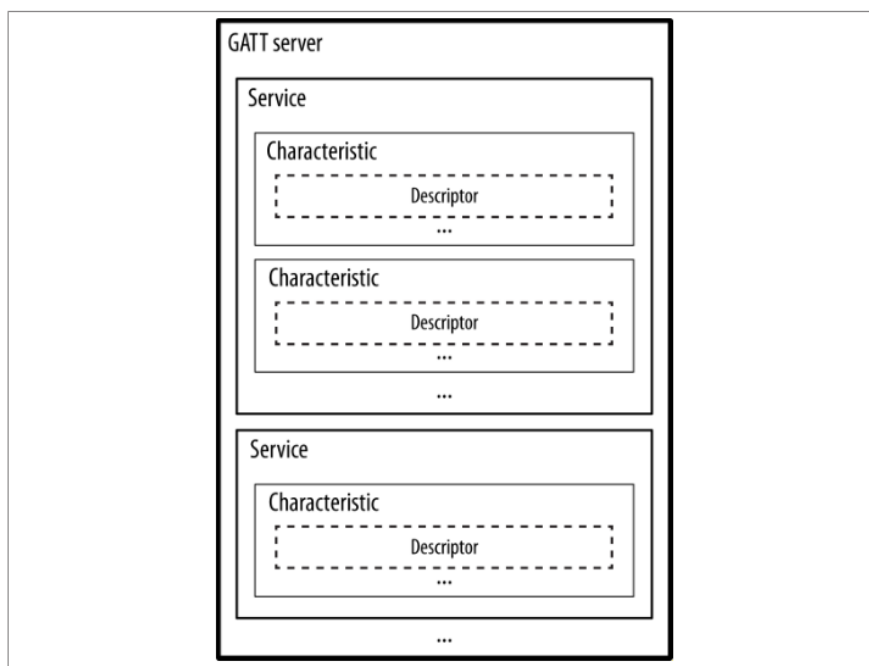


Ilustración 3-2. Jerarquía de datos *GATT* (Townsend, Cufi, Akiba, & Davidson, 2014)

Algunos de los servicios GATT más comunes e importantes son:

- *Mandatory GAP Service* – Este servicio contiene información del dispositivo y de acceso, tal como el nombre del dispositivo e identificación de producto y fabricante, es una parte de la pila de protocolo BLE. Se requiere para todos los dispositivos BLE y es parte de la especificación.
- *Mandatory GATT Service* – Este servicio contienen información acerca del *GATT server* y es una parte de la pila de protocolo BLE. Se requiere para todos los dispositivos *GATT server* y es parte de la especificación.

Las Características (*Characteristics*) son valores que son usados por un servicio, junto con propiedades e información de configuración. GATT define subprocedimientos para descubrir, leer o escribir atributos sobre una conexión BLE. Las características, junto con sus propiedades y sus datos de configuración (conocidos como *descriptors*) son almacenados en la tabla de atributos del *GATT server*. La tabla de atributos es simplemente una base de datos que contiene pequeñas partes de datos llamados Atributos (*attributes*). A parte del valor en sí mismo, cada *attribute* tiene las siguientes propiedades asociadas a él:

- **Manejador (handler):** Esto es la dirección del atributo en la tabla. Cada atributo tiene un manejador único.

- **Tipo (type):** Indica que tipo de datos contienen. Normalmente se nombra como UUID (Universal Unique Identifier) y es asignado por el Bluetooth SIG, o de forma personalizada.
- **Permisos:** Esta propiedad establece si el servidor *GATT* puede acceder al valor del atributo y de qué manera.

*GATT* define varios subprocedimientos de comunicación entre el servidor *GATT* y el cliente. Algunos de ellos son:

- Leer valor de característica - El cliente solicita leer el valor de la característica de un manejador específico, y el servidor responde al cliente con el valor (asumiendo que el atributo tiene permisos de lectura).
- Leer usando el UUID de la Característica - El cliente solicita leer todos los valores de características de un cierto tipo (que tenga un cierto UUID), el servidor responde al cliente con los manejadores y valores (asumiendo permisos) de todas las características que coincidan en ese tipo. El cliente no necesita conocer los detalles de estas características.
- Leer valores de múltiples características - El cliente solicita leer los valores de varios manejadores en una sola petición, y el servidor responde con los valores. El cliente debe saber cómo analizar los datos entre los diferentes valores de las características.
- Leer el descriptor de característica - El cliente solicita leer un descriptor de una característica de un manejador específico, y el servidor responde con el valor del descriptor.
- Descubrir característica por UUID - El cliente solicita descubrir el manejador de una característica por su tipo. El servidor responde con la declaración de la característica, que contiene el manejar el valor de la característica y los permisos.
- Escribir el valor de la característica: el cliente solicita escribir un valor de la característica en un identificador específico al servidor y el servidor responde al cliente para indicar si la escritura ha sido correcta o no.
- Escribir el descriptor de la característica - El cliente solicita escribir en un descriptor característico en un identificador específico al servidor, y el servidor responde al cliente para indicar si la escritura fue correcta o no.
- Notificación de valor característico: el servidor notifica al cliente un valor de característica. El cliente no necesita solicitar al servidor los datos, tampoco necesita

enviar ninguna respuesta cuando se recibe una notificación, pero primero debe configurar la característica para permitir notificaciones. Un perfil define cuando se supone que el servidor envía los datos.

Cada perfil inicia su servicio correspondiente e internamente registra el servicio con el servidor *GATT* del dispositivo. El servidor *GATT* añade el servicio entero a la tabla de atributos, y asigna manejadores únicos a cada atributo.

### ***3.3.5 Perfiles específicos***

Los perfiles específicos se limitan actualmente a perfiles basados en *GATT*. Esto significa que todos estos perfiles utilizan los procedimientos y modelos operativos del perfil del *GATT* como base para todas las aplicaciones.

Existen perfiles específicos definidos por SIG. El Bluetooth SIG va más allá de proporcionar un sólido marco de referencia para las capas de control y capas de datos de los dispositivos involucrados en una red BLE. Al igual que la especificación USB, también proporciona un conjunto predefinido de perfiles de casos de uso, basados en el *GATT*, que cubren completamente todos los procedimientos y formatos de datos necesarios para implementar una amplia gama de casos de uso específicos, entre los que se incluyen los siguientes ejemplos: Perfil “Encuéntrame”, perfil de proximidad, perfil para protocolo HID, perfil de glucosa, de temperatura, entre otros.

La especificación Bluetooth también permite a los proveedores definir sus propios perfiles para casos de uso no cubiertos por los perfiles definidos por SIG. Esos perfiles pueden mantenerse privados para los dos pares involucrados en un caso de uso particular (por ejemplo, un accesorio de salud y una aplicación de smartphone), o también pueden ser publicados por el proveedor para que otras partes puedan proporcionar implementaciones del perfil basadas en la especificación suministrada por el proveedor.

Este tipo de perfil específico será implementado durante el desarrollo del proyecto.

## ***3.4 Descripción del hardware y entorno de desarrollo para el SoC CC2541 de Texas Instruments***

### ***3.4.1 Kit de desarrollo y descripción del hardware***

Para facilitar el trabajo basado en el SoC CC2541 (Texas Instruments), se va a utilizar como plataforma de desarrollo el kit “CC2541 SensorTag Development Kit” (Wiki Texas Instruments). El *SensorTag* es un kit especialmente diseñado para sensores no

cableados y su comunicación a través de BLE con un Smartphone. El kit incluye un SensorTag con carcasa y pila. La carcasa de plástico se puede retirar fácilmente para acceder a la electrónica.

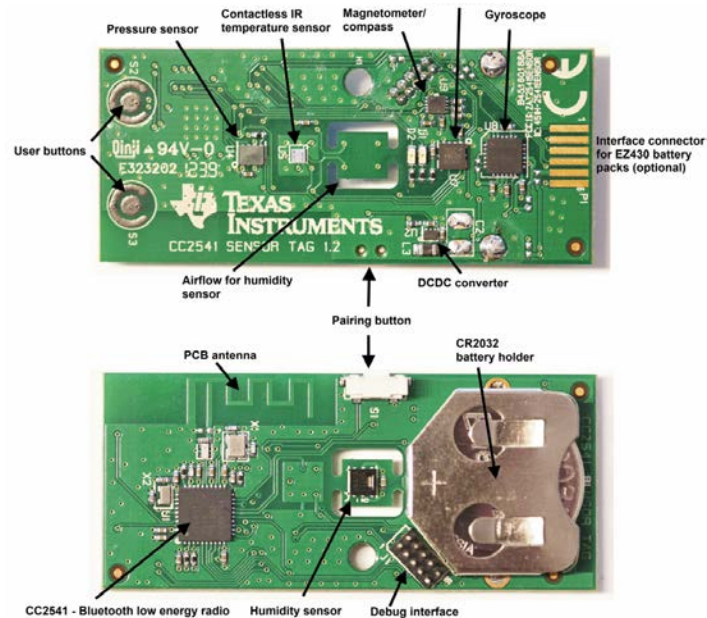


Ilustración 3-3. Componentes del *sensortag* (Wiki TI)

### 3.4.1.1 Sensores

El *SensorTag* incluye un amplio surtido de sensores para realizar aplicaciones con ellos (Wiki TI). Todos los sensores han sido escogidos por ser pequeños, de bajo coste, con montaje superficial, para que cumplan los requisitos de los dispositivos *wearables*. Los sensores usan una interfaz I<sup>2</sup>C para conectar a través de un mismo Bus con el microprocesador. Para minimizar el consumo de corriente todos los sensores vienen en modo “dormir” por defecto, al igual que entran en este modo entre medidas. Los diferentes sensores que incluye el *SensorTag* son:

- Sensor de temperatura infrarrojo sin contacto (Texas Instruments TMP006)
- Sensor de humedad (Sensirion SHT21)
- Giróscopo (Invensense IMU-3000)
- Acelerómetro (Kionix KXTJ9)
- Magnetómetro (Freescale MAG3110)
- Sensor de presión barométrica (Epcos T5400)

- Sensor de temperatura del chip (Implementado dentro del CC2541)
- Sensor de batería/voltaje (Implementado dentro del CC2541)

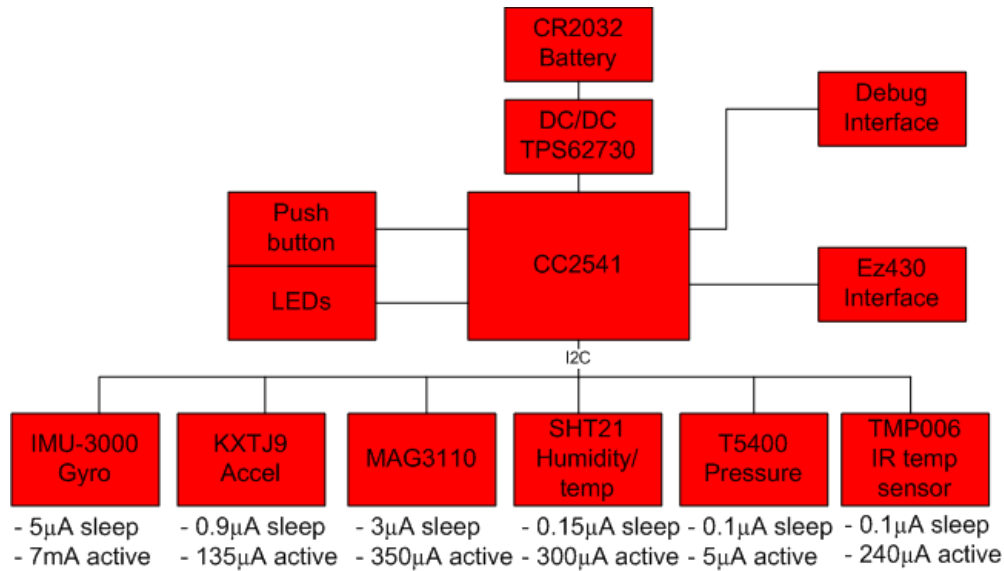


Ilustración 3-4. Diagrama de hardware y sensores del *SensorTag*

### 3.4.1.2 Interfaz de programación y depuración

La programación del *SensorTag* se hace a través de un conector de 2x5 pines con 1,27 mm de distancia entre pines (J1) que también es usado como interfaz de depuración. A continuación se puede ver la distribución de los pines:

Signal name	Pin #	Pin #	Signal name
GND	1	2	VDD
DC	3	4	DD
CSN	5	6	SCLK
EM_RESET	7	8	MOSI
NC	9	10	MISO

Tabla 3-2. Distribución de pines para programación/depuración

El CC2541 puede ser programado a través del hardware específico *CC debugger*. Este dispositivo permite utilizar una interfaz USB y a través de esta programar o depurar el *SensorTag*. Texas Instruments provee una herramienta de software para realizar el volcado del programa al CC2541 llamado *Flash Programmer*, aunque esto también se puede realizar a través del IDE de desarrollo.



### ***3.4.2 Entorno de Desarrollo IAR Embedded Workbench***

Todo el firmware desarrollado para el CC2540/41 tiene que ser desarrollado usando el entorno de desarrollo *Embedded Workbench for 8051* de IAR Software (IAR Systems). En esta sección se tratan las características básicas de dicho *software*, aspectos básicos de uso, como crear proyectos nuevos, y configurar proyectos para utilizar *BLE*.

IAR Embedded Workbench es un software privado que se debe adquirir a través de su compra. También tiene una edición de evaluación durante 30 días que puede ser descargada desde su página web.

#### ***3.4.2.1 Crear un proyecto nuevo***

A la hora de crear un proyecto nuevo de BLE, Texas Instruments recomienda partir de uno de los proyectos de ejemplo de los que nos provee (Texas Instruments, 2015). Por ejemplo el proyecto *SimpleBLEPeripheral*, en el que hay desarrollado un firmware de un dispositivo que actúa como *peripheral*. Una vez que tenemos el proyecto abierto se puede editar a conveniencia, con la mayoría de funcionalidades ya implementadas.

Para abrir un proyecto ya existente, primero se inicia el IDE, usando el método normal en Windows, encontrando el programa en **Inicio > Programas > IAR Systems > IAR Embedded Workbench for MCS-51 8.10.4 > IAR Embedded Workbench**.

Una vez abierto IAR, hacer click en **File > Open > Workspace**. Seleccionar entonces un proyecto de la carpeta proporcionada por TI. En el caso del firmware *SimpleBLEPeripheral* la ruta es **C:\TexasInstruments\BLECC254X1.4\Projects\ble\SimpleBLEPeripheral\CC2540DB\SimpleBLEPeripheral.eww**

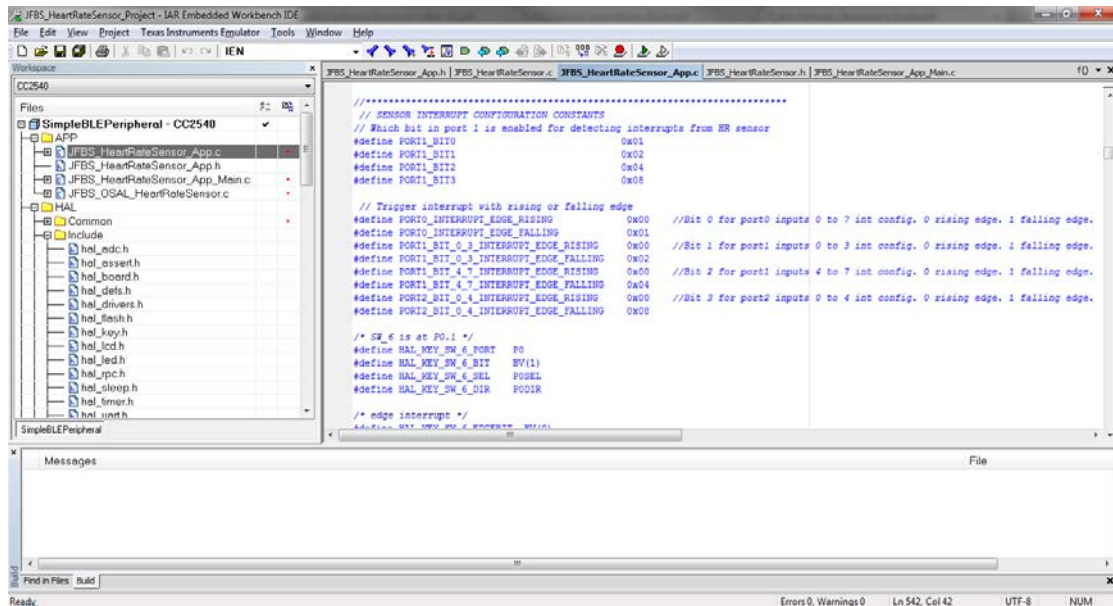


Ilustración 3-5. IAR Embedded Workbench

### 3.4.2.2 Opciones del proyecto, configuraciones y definición de símbolos

Los proyectos tienen un conjunto de opciones, que incluyen la configuración del compilador, *linker*, depurador y más. Para ver estas opciones, se debe hacer clic derecho en el nombre del proyecto, arriba de la lista de archivos, y seleccionar “Options...”

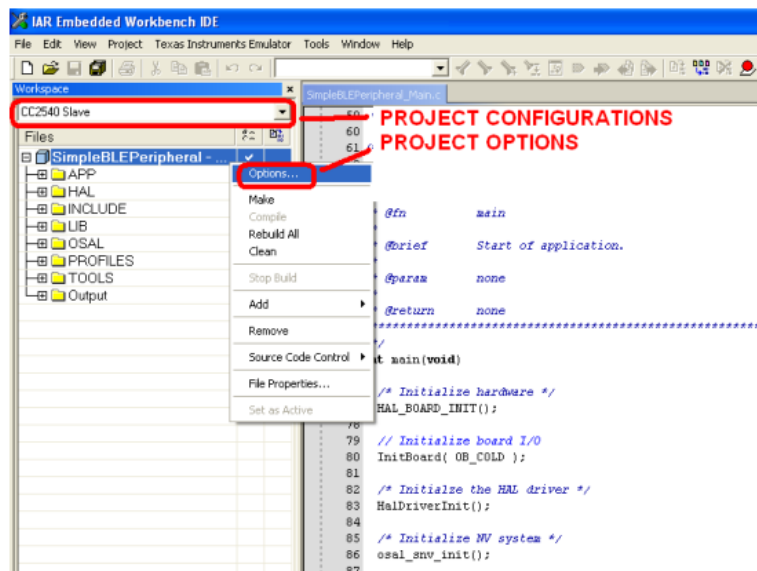


Ilustración 3-6. Configuración y opciones del proyecto

Después de hacer clic en “Options...” aparece una nueva ventana, mostrando las opciones del proyecto. A veces es útil tener algunas configuraciones diferentes de opciones para diferentes configuraciones de hardware, como cuando se utilizan varias plataformas de

hardware. La configuración predeterminada en el proyecto *SimpleBLEPeripheral* es la configuración "CC2540/41 Slave", que está dirigida a la plataforma de hardware *SmartRF05 + CC2540 / 41EM* que se incluye con el kit de desarrollo *CC2540 / 41DK*. La otra opción disponible, "CC2540 / 41DK-MINI Keyfob Slave", está optimizada para la tarjeta "keyfob" del kit de desarrollo *CC2540 / 41DK-MINI*.

Una de las configuraciones importantes a la hora de construir un proyecto son los símbolos definidos por el preprocesador del compilador. Estos valores se pueden encontrar (y establecer) haciendo clic en la categoría "C/C++ Compiler" a la izquierda y, a continuación, haciendo clic en la pestaña "Preprocesador" a la derecha:

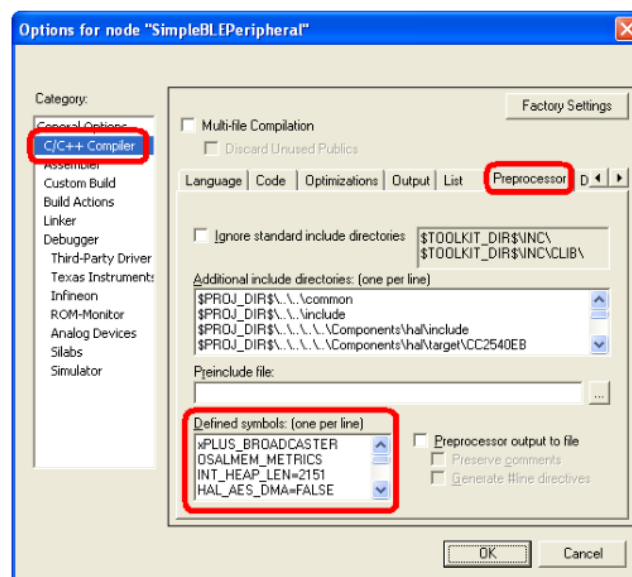


Ilustración 3-7. Ajustes de símbolos definidos en el preprocesador

Los símbolos que se definen aquí se aplicarán a todos los archivos del proyecto. Los símbolos se pueden definir con o sin valores. Por conveniencia, algunos símbolos pueden ser definidos con el carácter 'x' delante de ellos. Esto generalmente significa que una función está siendo desactivada, y se puede habilitar eliminando la 'x' y dejando que el nombre propio del símbolo sea definido.

Además de la lista de símbolos definidos en la configuración del compilador, los símbolos se pueden definir en los archivos de configuración, que se incluyen al compilar. La pestaña "Extra options" bajo la configuración del compilador permite configurar los archivos de configuración que se incluirán. El archivo *config.cfg* debe ser incluido con cada compilación, ya que define algunas constantes universales requeridas. Los archivos *config\_master.cfg* y *config\_slave.cfg* incluidos con el kit de desarrollo de software definirán los símbolos apropiados para las compilaciones *master* y *slave*, respectivamente.

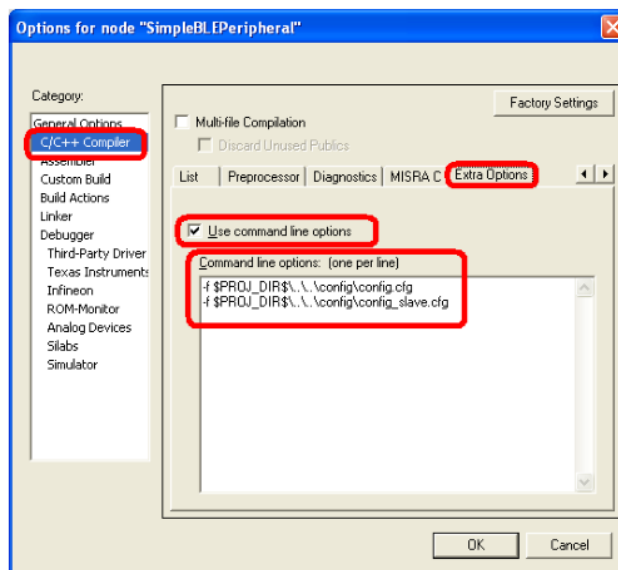


Ilustración 3-8. Configuración del archivo de configuración

Los siguientes símbolos son utilizados por la pila de protocolos BLE y el software, y se pueden encontrar en el proyecto de ejemplo:

#### Símbolos obligatorios para la pila *BLE*:

- INT\_HEAP\_LEN - Este símbolo define el tamaño de la pila utilizado por el Administrador de memoria OSAL (detallado más adelante) en bytes. El valor predeterminado en el proyecto de ejemplo es 3072. Este valor se puede aumentar si se requiere memoria de pila adicional por la aplicación; sin embargo si se aumenta demasiado se puede exceder el límite de RAM. Si la aplicación necesita más memoria para las variables locales, puede ser necesario disminuir este valor.
- HALNODEBUG - Este símbolo debe ser definido para todos los proyectos para desactivar las aserciones HAL.
- OSAL\_CBTIMER\_NUM\_TASKS - Este símbolo define el número de temporizadores de devolución de llamada OSAL que se pueden utilizar. La pila de protocolo BLE utiliza el temporizador de devolución de llamada OSAL y, por lo tanto, este valor debe definirse como 1 o 2 (se permite un máximo de dos temporizadores de devolución de llamada). Para aplicaciones que no utilizan temporizadores de devolución de llamada, como la aplicación de ejemplo, este valor debe definirse como 1.
- HAL\_AES\_DMA - Este símbolo debe definirse como 'TRUE 1, ya que la pila BLE utiliza DMA para el cifrado AES.

- HAL\_DMA - Este valor debe definirse como VERDADERO para todos los proyectos BLE, ya que el controlador DMA es utilizado por la pila al leer y escribir en flash.

Existen otros muchos símbolos opcionales pero de gran interés, que deben consultarse durante la configuración. Se pueden consultar en el documento “TI CC254x Bluetooth Low Energy Software Developer’s Guide”.

### 3.4.2.3 Compilación y depuración del proyecto

Para compilar un proyecto, se hace clic con el botón derecho en el nombre del espacio de trabajo y a continuación hacer clic en "Make". Esto compilará el código fuente, vinculará los archivos y creará el proyecto. Cualquier error o advertencia del compilador aparecerá en la ventana de mensajes en la parte inferior de la pantalla.

Para descargar el código compilado en un dispositivo CC2540/41 y depurar, se conecta la plataforma utilizando un depurador de hardware (como el CC debugger), conectado a la PC a través de USB. Pulsar el botón "Debug" en la parte superior derecha de la ventana de IAR.

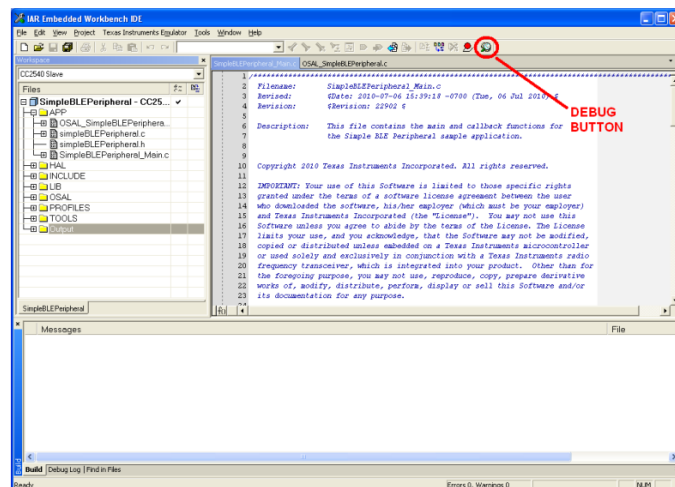


Ilustración 3-9. Botón de depuración

Una vez que se descargue el código, aparecerá una barra de herramientas con los comandos debug en la esquina superior izquierda de la pantalla. Se puede iniciar la ejecución del programa pulsando el botón "Go" de la barra de herramientas. Una vez que el programa se está ejecutando, se puede salir del modo de depuración presionando el botón "Stop Debug". Ambos botones se muestran en la siguiente imagen:

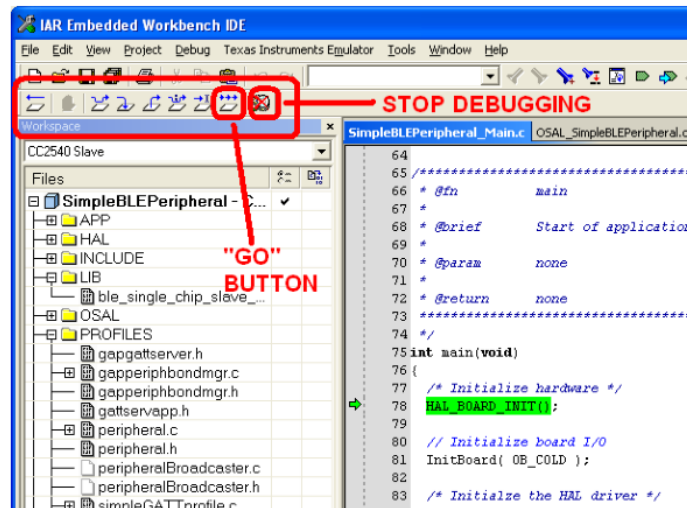


Ilustración 3-10. Barra de depuración

En este punto, el programa debería estar ejecutándose por sí solo. El depurador de hardware puede estar conectado desde el CC2540/41 y continuará funcionando mientras el dispositivo permanezca encendido.

### 3.4.3 Proyecto de BLE en CC254x

#### 3.4.3.1 Estructura del proyecto

Los archivos del proyecto se dividen en los siguientes grupos:

- APP: son el código fuente de la aplicación y los archivos de cabecera.
- HAL - Este grupo contiene el código fuente HAL y los archivos de cabecera. Más adelante se puede ver más información sobre el HAL.
- INCLUDE: este grupo incluye todos los archivos de encabezado necesarios para la API de la pila de protocolos BLE.
- LIB: este grupo contiene el archivo de biblioteca de pila de protocolos *CC2540\_ble\_single\_chip\_peri.lib*.
- NPI - Interfaz de procesador de red, una capa de transporte que permite enrutar datos HCI a una interfaz serie.
- OSAL: este grupo contiene el código fuente OSAL y los archivos de cabecera. Más información sobre el OSAL se describirá más adelante.

- **PROFILES:** este grupo contiene el código fuente y los archivos de cabecera para el perfil de rol GAP, el perfil de seguridad GAP y el perfil GATT. También incluye los perfiles creados manualmente.
- **TOOLS** - Este grupo contiene los archivos de configuración *buildComponents.cfg* y *buildConfig.cfg*. También contiene los archivos *OnBoard.c* y *OnBoard.h*, que manejan las funciones de la interfaz de usuario.
- **OUTPUT:** este grupo contiene archivos generados por IAR durante el proceso de compilación, incluidos los binarios y el archivo de mapeado de memoria.

### 3.4.3.2 *Inicialización*

La ejecución del programa comienza en la función *main*. En ella se configura inicialmente el hardware, pero la verdadera inicialización tiene lugar en la capa OSAL.

OSAL (*Operating System Abstraction Layer*, o capa de abstracción del sistema operativo) es una capa situada por encima de las capas que interactúan con el hardware y por debajo de la aplicación. Su función es simular un sistema operativo, que se encarga de asignar recursos a tareas y mantener la ejecución continua del programa.

Al final de la función *main*, se invoca a la rutina *osal\_start\_system* que inicia el sistema e invoca a la función *osalInitTasks*, configurada por la aplicación.

#### **osalInitTasks:**

- Se define por la *application*, es decir, la define el programador.
- Todas las capas del software que usan el OSAL (LL, HCI, L2CAP, GAP...) deben ser inicializadas por esta función.
- Dentro de la función, la rutina de inicialización de cada capa es convocada y un “*task ID*” de 8-bits le es asignado.
- La prioridad de las tareas viene determinada por el “*task ID*” cuanto más bajo sea este, más prioridad tendrá la tarea. Es por eso, que al crear una aplicación, esta debe ser añadida al final de la lista, para que las tareas de las pilas de protocolos tengan la máxima prioridad.

### 3.4.3.3 *Eventos de tareas y procesamiento de eventos*

Cuando el OSAL completa la inicialización, ejecuta un bucle comprobando *task events*. Este bucle se encuentra en la función *osal\_start\_system* en el archivo *OSAL.c*. Los *Tasks events*, o eventos de tareas, son implementados como una variable de 16-bit (uno por cada tarea), donde cada bit corresponde a un único evento.

- *Task*, se entiende cada tarea como cada una de las capas iniciadas en el OSAL.
- *Event*, los eventos son hitos en el bucle de aplicación, según los cuales se ejecuta el programa. Sus indicadores son los *event flags*. Los eventos incluyen timers, messages y otros eventos definidos por el programador.

La definición y uso de estos *event flags* son exclusivamente de la aplicación. El único flag que no se puede definir por la aplicación, puesto que está reservado es el 0x8000 que corresponde al *event* SYS\_EVENT\_MSG.

Cuando el OSAL detecta el *event* de una *task*, invocará la *event processing routine* de la *task*. Cada capa debe añadir su *event procesing routine* a la tabla formada por el array de funciones punteros *taskArr* (localizada en la *application*). El orden de los *event procesing routine* en el *taskArr* debe ser idéntico al orden de los “ID task” en la función *osalInitTasks* para que sean procesados por la capa correcta.

Tener en cuenta que una vez que el *event* ha sido “apuntado” y si no se quita el *event flag*, el OSAL continuará llamando al *task’s process event handler*. Es por eso que normalmente, cuando un evento se ejecuta, se devuelve como variable un *event* de 16-bit con la *flag* desmarcada.

Es posible para cualquier capa del software, el marcar un *OSAL event* para otras capas y para sí misma:

- La forma más sencilla para marcar un *OSAL event* es usar la función *osal\_set\_event* (prototipo en *OSAL.h*), la cual inmediatamente programa un nuevo *event*. Con esta función, especificas el “*task ID*” y el *event flags* como parámetros.
- Otra forma es usar la función *osal\_start\_timerEx* (prototipada en *OSAL\_Timers.h*). Esta función opera como la función *osal\_set\_event*, seleccionas el *task ID* de la tarea que procesará el *event* y el *event flag* como parámetros; sin embargo en esta función existe un tercer parámetro que introducir, un *timeout* (pausa) en milisegundos. El OSAL configurará un *timer*, y el *event* no se ejecutará hasta que el *timeout* especificado expire.



#### 3.4.3.4 *Mensajes OSAL*

La OSAL también provee un sistema para diferentes subsistemas del software para comunicarse entre ellos enviando o recibiendo mensajes. Los *messages* pueden contener cualquier tipo de datos y pueden ser de cualquier tamaño. Para enviar un *OSAL message*, primero la memoria debe ser designada para llamar a la función *osal\_msg\_allocate*, pasándole la longitud del mensaje como único parámetro. Se devuelve un puntero a un *buffer* que contiene el espacio designado. Si no hay memoria disponible, se devuelve un puntero NULL.

Entonces se copian los datos en el buffer. Para enviar el mensaje, se llama a la función *osal\_msg\_send*, con la *task* de destino del mensaje indicada como parámetro.

El OSAL notificará a la *task* destinataria que el mensaje está llegando marcando la *flag* SYS\_EVENT\_MSG de la *task*. Esto causa que se llame a la función “*receiving task’s event handler*”. La *task* destinataria puede entonces recuperar los datos llamando a la función *osal\_msg\_receive*, y puede procesarlo según los datos recibidos.

Es recomendable que toda *task* del OSAL tenga un función de procesamiento de mensajes local que decida qué acción tomar en base al tipo de mensaje recibido.

Una vez que la *task* ha completa el procesado del mensaje, debe desasignar la memoria haciendo uso de la función *osal\_msg\_deallocate* (no es necesario utilizar *osal\_mem\_free* cuando se utiliza *osal\_msg\_deallocate*).

#### 3.4.3.5 *HAL, Capa de abstracción del Hardware*

La HAL del software del CC5240/41 provee una interfaz de abstracción entre el hardware físico, y la *application* y la pila de protocolos. Esto permite el desarrollo de nuevo hardware (como una nueva PCB) sin hacer cambios en código fuente de la pila de protocolos o de la *application*. La HAL incluye el software para las interfaces de comunicación SPI y UART, ADC, LEDs, etc. Los drivers de la HAL proveen soporte a las siguientes plataformas de hardware:

SmartRF05EB + CC2540EM

SmartRF05EB + CC2541EM

CC2540 Keyfob

CC2541 Keyfob

CC2541 SensorTag

CC2540 USB Dongle

Cuando el desarrollo es con una plataforma diferente de hardware, es necesario modificar el código fuente de la sección HAL para que el proyecto siga siendo compatible.

### ***3.5 Desarrollo de APP para Android***

Dado que la aplicación para monitorizar el ritmo cardíaco va a ser utilizada en dispositivos con el sistema operativo Android, es necesario instalar una serie de utilidades de software necesarias para poder desarrollar aplicaciones para esta plataforma:

- Java SDK
- Android SDK
- Android Studio

#### ***3.5.1 Instalación del Java SDK***

En primer lugar, para poder desarrollar aplicaciones en Android, es necesario tener instalado el kit de desarrollo de Java (*Java Development Kit*) (Oracle). Éste es necesario, ya que el desarrollo de aplicaciones en Android implica el uso de archivos .java, y es necesario contar con un compilador que transforme los archivos .java en archivos .class de bytecode.

Por tanto, lo primero es descargar el JDK .Posteriormente, será necesario aceptar la licencia pulsando en “*Accept License Agreement*”, y seleccionar nuestro sistema operativo y su arquitectura. En este caso, se ha seleccionado Windows x64, cuyo paquete tiene el siguiente nombre jdk-8u60-windows-x64.exe, debido a que la versión más reciente disponible para Windows es la 8u60. Hay que tener en cuenta que para poder desarrollar aplicaciones para Android es necesario que la versión del JDK sea superior a la 6, y en el caso de querer desarrollar para Android 5.0 en adelante, la versión del JDK debe ser como mínimo la 7.

Una vez descargado, se puede instalar haciendo doble click sobre el archivo descargado, para posteriormente continuar con el asistente por defecto. Al finalizar la instalación, habrá que configurar las variables de entorno para que el siguiente paquete a instalar pueda encontrar el lugar donde se ha instalado Java. Para ello, hay que hacer click derecho sobre el icono de Equipo situado en el menú Inicio, seleccionar “Propiedades” > “Configuración avanzada del sistema” > “Variables de entorno...” y en el apartado de

VARIABLES del sistema pulsar sobre “Nueva...” y como nombre de la variable se puede utilizar “JAVA\_HOME”, y como valor “C:\Program Files\Java\jdk1.8.0\_31”.

Con esto se habrá quedado configurado el JDK para poder desarrollar aplicaciones que requieran el compilador de java.

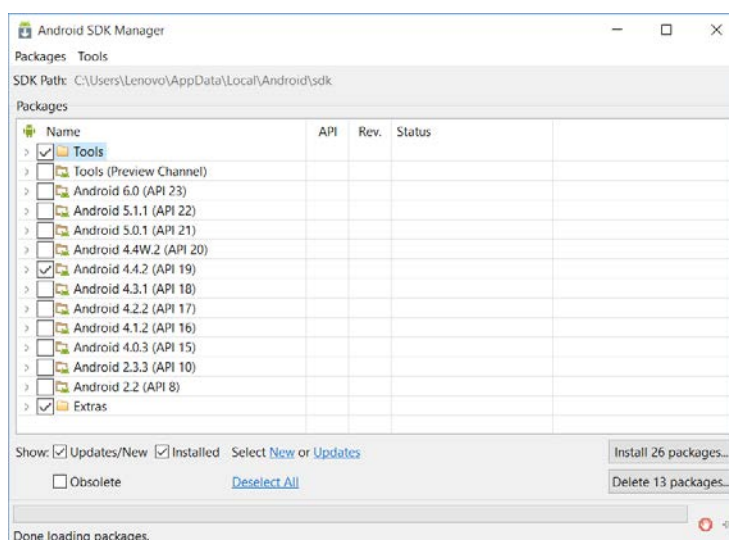
### ***3.5.2 Instalación de Android Studio y Android SDK***

Los pasos para instalar el JDK han sido descritos en la sección anterior, pero aún son necesarios el entorno de desarrollo en el que poder programar, y el conjunto de APIs y herramientas que permiten desarrollar una aplicación.

Por ello, lo siguiente es instalar el entorno de desarrollo Android Studio (Android Studio IDE), junto con las herramientas de desarrollo de Android (Android SDK). Google se encarga de proporcionar las dos herramientas empaquetadas, por lo que al ser la vía oficial para desarrollar aplicaciones para Android, se ha obviado la instalación de Android SDK y un entorno de desarrollo como Eclipse por separado, y se ha optado por la opción oficial de utilizar Android SDK junto con el IDE Android Studio.

Una vez instalado, es necesario instalar una serie de paquetes que permiten desarrollar aplicaciones de Android. Para ello, una vez abierto Android Studio, hay que pulsar sobre en el icono “*SDK Manager*” situado en la barra superior. Se abrirá el SDK Manager, con el cual se puede administrar las herramientas de desarrollo disponibles. Como mínimo hay que seleccionar las siguientes herramientas:

- Android SDK Tools
- Android SDK Platform-tools
- Android SDK Build-tools
- Android Support Repository
- Android Support Library
- Android 4.4 (API19) SDK Platform



**Ilustración 3-11. Android SDK Manager**

Esto proporciona acceso a las librerías proporcionadas por la API seleccionadas (en este caso la API19), así como a las herramientas de compilación necesarias para generar el archivo .apk (que instalaremos en el dispositivo móvil una vez compilemos la aplicación). Tras haber seleccionado estos elementos, se pulsa en “Instalar paquetes”, se aceptan la licencia y se pulsa nuevamente en “Instalar”. Tras esto, ya estarán las herramientas de desarrollo listas para crear aplicaciones para Android.

Aunque ya se puedan desarrollar aplicaciones, es necesario probarlas para encontrar posibles fallos, por lo que será necesario un dispositivo que ejecute Android. Concretamente, Android Studio permite configurar un dispositivo virtual (AVD), pero dado que se dispone de un dispositivo móvil para realizar el proyecto, se realizará la configuración del mismo para poder depurar las aplicaciones directamente sobre él.

### ***3.5.3 Configuración del dispositivo móvil para depurar aplicaciones***

Para poder depurar aplicaciones en el dispositivo móvil, es necesario realizar una pequeña configuración en el dispositivo. Cualquier dispositivo con Android 4.2 en adelante tiene las opciones de desarrollo ocultas, por lo que lo primero es habilitar las opciones de desarrollo. Para ello, hay que abrir los ajustes del dispositivo, y pulsar en “Información del teléfono”. Ahora se pulsan 7 veces seguidas en “Número de compilación” hasta que aparezca un mensaje indicando que se han activado las opciones de desarrollo.

Tras activarlas y volviendo a los Ajustes del dispositivo, ya se puede pulsar en la nueva opción “Opciones de desarrollo”. Una vez dentro, se pulsa en “Depuración USB”, y con eso ya estaría configurado el dispositivo. Ahora sólo queda asegurar de que están

instalados y actualizados los drivers de nuestro dispositivo móvil en el ordenador, para que el ordenador pueda reconocer correctamente el dispositivo cuando se conecte mediante un cable USB.

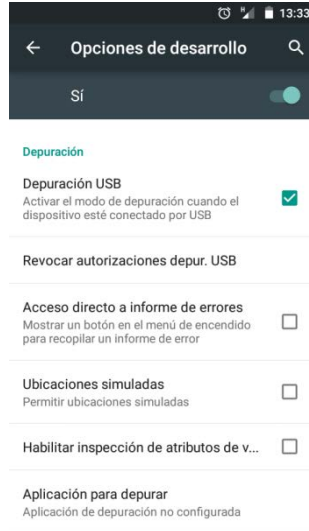


Ilustración 3-12. Dispositivo con Android mostrando las opciones de desarrollo

### 3.5.4 Creación de un proyecto en Android Studio

Al abrir Android Studio por primera vez, si no se ha creado ningún proyecto se abrirá una ventana de bienvenida (Sgoliver.net). Aquí se debe seleccionar “*Start a new Android Studio Project*”. A continuación, hay que escribir un nombre a la aplicación, así como indicar el dominio de la compañía. Por último, se elige el lugar del disco duro en el que guardar el proyecto.

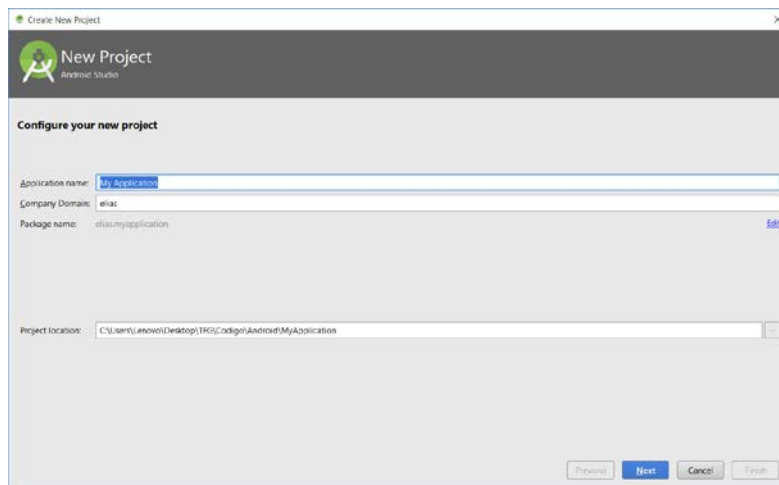


Ilustración 3-13. Ventana de creación de un nuevo proyecto en Android Studio

Lo siguiente es establecer los dispositivos para los que funcionará la aplicación, ya que Android puede ejecutarse en una gran variedad de dispositivos, ya sean *smartphones*, *tablets*, TV, *wearables* o automóviles. Una vez elegido el dispositivo, lo siguiente es establecer cuál será la versión mínima del SDK. La versión del SDK determinará la versión mínima de Android que debe tener instalado el dispositivo para poder ejecutar la aplicación.

Hay que tener en cuenta que aunque utilizar la versión más reciente del SDK (y por tanto de la API) proporciona acceso a las últimas características y mejoras introducidas en Android, debido a la fragmentación de Android se corre el riesgo de que una aplicación sea incompatible con la mayoría de los dispositivos existentes en el mercado, ya que estos suelen utilizar versiones de Android más antiguas, que apenas reciben un año de soporte por parte del fabricante, por lo que enseguida se quedan sin actualizaciones.

Por ello, lo ideal es elegir la primera versión del SDK en la que se introdujeron las características de la API que se quieren utilizar en una aplicación, ya que así se conseguirá que una aplicación sea capaz de acceder a los recursos de la API que necesita para llevar a cabo el proyecto que se tiene en mente, y a la vez se conseguirá que la aplicación sea compatible con el mayor número de dispositivos posible, aunque haya que renunciar a las últimas funciones y mejoras de Android.

Tras haber elegido la versión mínima del SDK, hay que añadir una actividad al proyecto. En este caso, lo normal es comenzar con una “*Blank Activity*” (Android Developers), la cual se explicará más adelante. Por defecto, la actividad creada se llamará “*MyActivity*”, por lo que se puede aceptar la configuración por defecto y pulsar en el botón “Finish”, con lo que se habrá creado el proyecto.

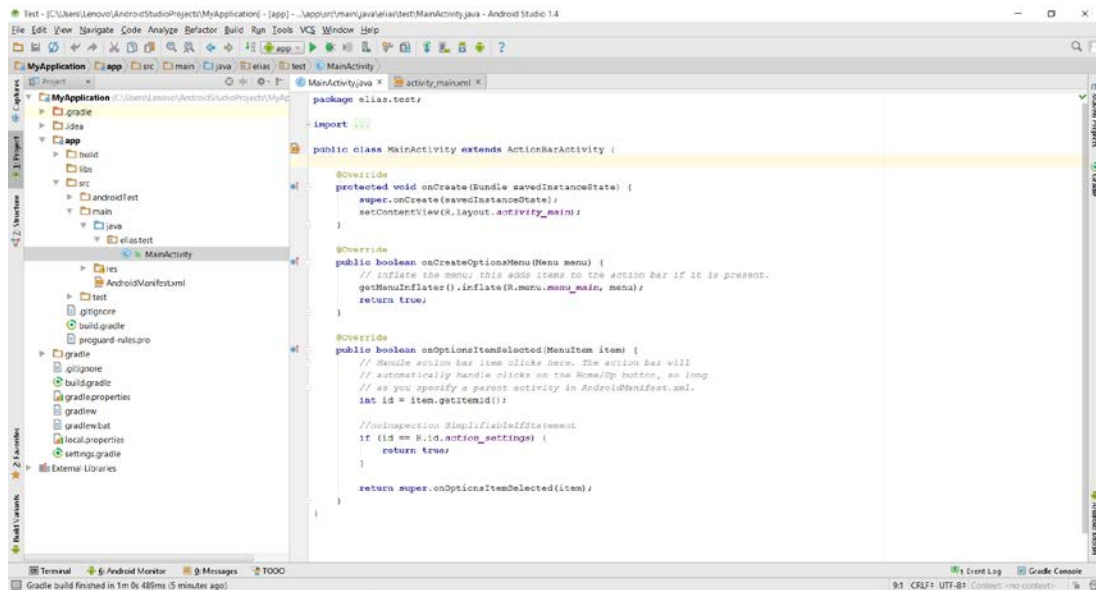


Ilustración 3-14. Android Studio mostrando el fichero MainActivity.java

Una vez preparado el entorno de desarrollo y creado un nuevo proyecto, lo siguiente es tener claros algunos conceptos sobre la estructura de un proyecto Android, es decir, los elementos que lo componen.

### ***3.5.5 Estructura de un proyecto Android***

Para poder hacerse una mejor idea de la estructura que tiene el proyecto es necesario realizar una modificación en Android Studio. Dicho cambio implica pulsar en la lista desplegable situada en la parte superior izquierda, y cambiar de “Android” a “Project”. En la nueva jerarquía lo primero que aparece es una carpeta con el nombre del proyecto, y otras carpetas que representan los distintos módulos de la aplicación.

Normalmente, sólo se utilizará el módulo “app”, el cual incluye dos carpetas fundamentales:

- /app/src/main/java
- /app/src/main/res

La carpeta “java” contiene archivos con el código fuente de la aplicación, sin incluir los archivos que definen la interfaz de usuario. Por lo tanto, cualquier archivo .java estará aquí.

Cuando se inicie una aplicación de Android, el primer fichero que se ejecutará será el MainActivity.java, por lo que se debe configurar adecuadamente. Dado que la clase *MainActivity* se extiende de una clase *Activity*, su ciclo de vida es el mismo que el de cualquier Activity en Android (ver Figura 3-12).

El primer método que se ejecuta cuando se crea una *Activity* es el método “onCreate()”, por lo que en este método se debe establecer el archivo .xml del que se obtendrá la apariencia de nuestra Activity, así como obtener la referencia a los distintos elementos que se hayan situado en la pantalla.

A su vez, los métodos “onStart()”, “onResume()”, “onPause()”, “onStop()” y “onDestroy” permiten realizar distintas acciones en función de los procesos por los que vaya pasando una Activity, ya que puede ser que se cambie de una Activity a otra, que se cierre la aplicación, o que posteriormente se vuelva a la misma Activity. Por lo tanto, con los métodos listados anteriormente, se pueden procesar adecuadamente todo este tipo de situaciones.

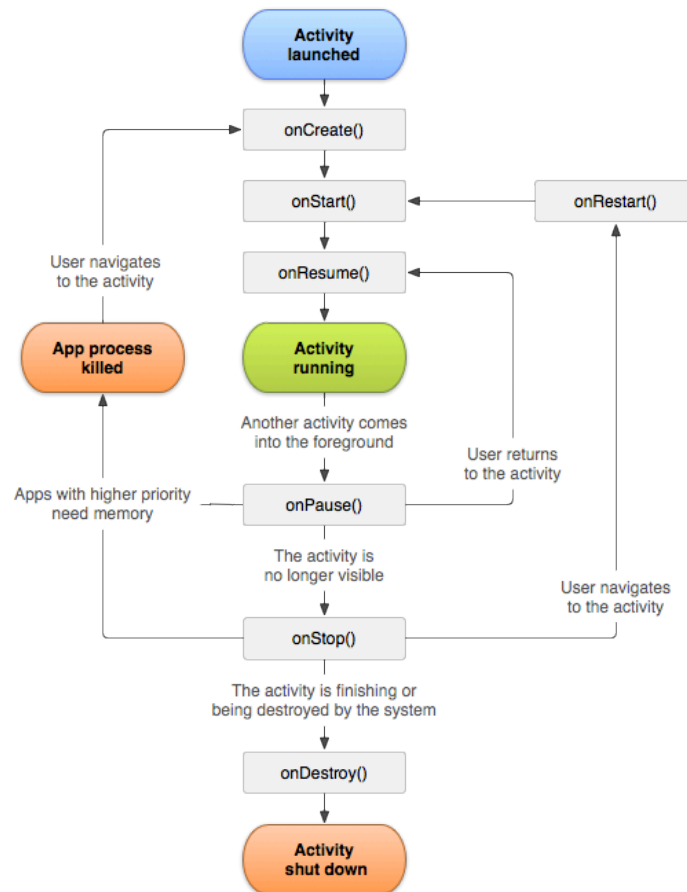


Ilustración 3-15. Ciclo de vida de una Activity en Android (Sgoliver.net)

La carpeta “res” contiene archivos que definen la interfaz de usuario. Por lo tanto, cualquier archivo como imágenes, iconos, sonidos, strings, layouts se incluyen en esta carpeta.

En un layout se pueden definir mediante el lenguaje de marcado XML los distintos elementos que compondrán la interfaz, ya sean cuadros de texto, botones, listas desplegables, menús, e incluso se pueden crear controles propios para la interfaz.

Dentro de un layout, hay que definir la disposición de los elementos dentro de la interfaz. Lo más habitual para diseñar interfaces sencillas es utilizar un *LinearLayout* o un *RelativeLayout*. Mediante *LinearLayout* se sitúan los elementos uno a continuación del otro, ya sea en horizontal o en vertical, y mediante *RelativeLayout* se distribuyen los elementos unos respecto de otros. Por lo que elegir entre uno u otro será elección del desarrollador.

Una vez definida la distribución de los elementos, los nuevos elementos que se incluyan tendrán una serie de propiedades que se podrán modificar mediante etiquetas, las cuales modificarán su tamaño, forma, distribución, márgenes respecto a otros elementos, colores, entre otros.



Volviendo a la estructura del proyecto, hay dos archivos muy importantes situados fuera de la carpeta java y res:

- /app/src/main/AndroidManifest.xml
- /app/build.gradle

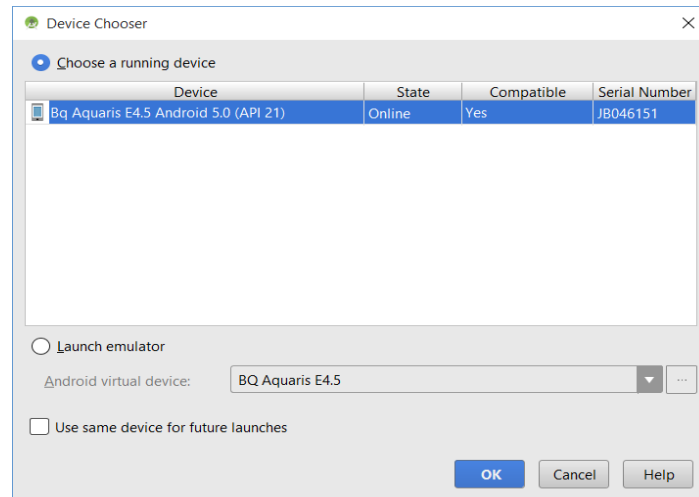
El archivo *AndroidManifest.xml* define los aspectos fundamentales de la aplicación, como son el nombre de la aplicación, su icono, las distintas pantallas y servicios que tiene la aplicación, y los permisos que requiere la aplicación para funcionar.

El archivo *build.gradle* contiene la información necesaria para poder compilar la aplicación de Android, como la versión del SDK utilizada para realizar la compilación, el ID de la aplicación y la versión mínima del SDK. Hay un *build.gradle* para cada módulo del proyecto, así como un *build.gradle* para todo el proyecto.

### ***3.5.6 Instalar la aplicación en un dispositivo con Android***

Una vez creada la aplicación, se puede realizar la compilación e instalación en un dispositivo con Android para probarla y detectar posibles fallos.

Para ello, lo primero es haber configurado correctamente el dispositivo móvil con Android para poder depurar aplicaciones mediante USB, y conectarlo mediante el cable USB al ordenador. A continuación, se pulsa sobre el botón de la barra superior “Run ‘app’”, tras lo cual Android Studio comenzará a compilar el proyecto, hasta que muestre una ventana como la de la Ilustración 3-13. En esta ventana se puede elegir entre ejecutar la aplicación en un dispositivo virtual, o instalarla en un dispositivo real. Por lo que, en este trabajo se elegirá el dispositivo que aparece en la lista.



**Ilustración 3-16.** Ventana para elegir el dispositivo con Android en el que instalar la aplicación

Cuando la aplicación se haya instalado y se esté en ejecución, en la parte inferior de Android Studio se podrá hacer uso de la pestaña llamada “*Android Monitor*”, dentro de la cual se seleccionará la pestaña “*logcat*” y visualizar en tiempo real la información relativa al dispositivo Android, por lo que si la aplicación experimenta algún problema, se verá reflejado en este lugar y será posible detectar el fallo.



# Capítulo 4

---

## Descripción del Hardware y Software desarrollado

---

### *4.1 Introducción*

En el capítulo anterior se describieron los elementos hardware y software necesarios para realizar este Proyecto Fin de Carrera.

En este capítulo se describe el dispositivo diseñado y desarrollado en este trabajo. En concreto, se describe todo el hardware y el software desarrollado.

En primer lugar, se describe todo el hardware desarrollado, es decir, el sensor de pulso cardiaco a partir de un emisor y receptor de luz. A continuación, se describe el software desarrollado en los elementos de la arquitectura que componen el sistema.

### *4.2 Descripción del hardware de adquisición y acondicionamiento de la señal de ritmo cardíaco*

El circuito de adquisición y acondicionamiento de la señal están dividido en varios subsistemas. Estudiando cada uno de ellos por separado se comprende mejor el comportamiento global.

### 4.2.1 Sensor emisor y receptor de luz

Para realizar la fotoplestimografía, el sensor utilizado es un foto reflector modelo RPR-220, de ROHM semiconductor. Este componente está formado por un diodo que emite luz infrarroja, y un fototransistor que detecta señales de una longitud de onda de hasta 800 nm.

El sensor, en un principio no destinado para usarlo como fotoplestimografía sino como detector óptico, puede ser utilizado como tal si se realiza un adecuado condicionamiento de la señal detectada. El componente emite luz infrarroja a través de un diodo, parte de esta luz es reflejada sobre la superficie donde se proyecte. El fototransistor entonces emite una señal en función de la cantidad de luz infrarroja que ha sido reflejada. La cantidad de luz reflejada sobre tejidos con arterias cercanas, variará influenciado por el cambio de presión en estas a través de los impulsos de sangre del corazón

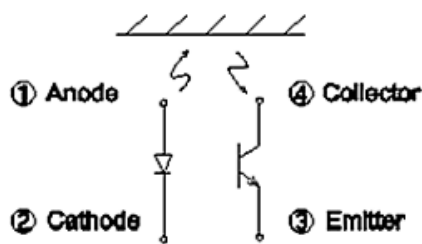


Ilustración 4-1. Diagrama sensor óptico RPR-220

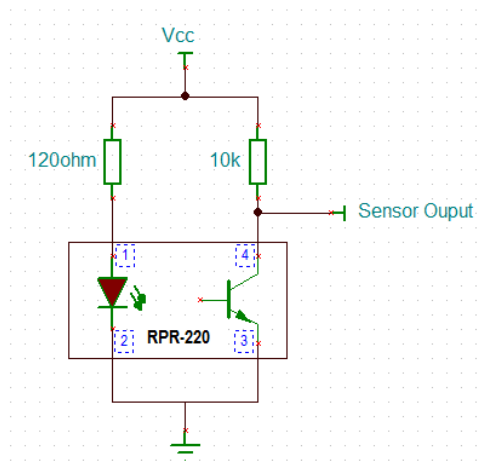


Ilustración 4-2. Circuito emisor y receptor de luz

Las conexiones del sensor se realizan de la siguiente forma:

- Pin1 – Ánodo del diodo emisor de luz, se conecta a través de una resistencia a la tensión de alimentación  $V_{CC} = 3,3 V$ .
- Pin 2 – Cátodo del diodo emisor de luz, se conecta a masa (  $0 V$  ).
- Pin 3 – Emisor del fototransistor, se conecta a masa (  $0 V$  ).

- Pin 4 – Colector del fototransistor, se conecta a través de una resistencia de alto valor (10 k $\Omega$ ) a la tensión de alimentación  $V_{CC} = 3,3 V$ , para que actúe de resistencia *pull-up*.

El diodo emisor tiene una caída de tensión en polarización directa de  $V_F = 1,34 V$ . Al colocar una resistencia de 120  $\Omega$  entre alimentación y ánodo, se obtiene una corriente de funcionamiento en el diodo  $I_F = (3,3 - 1,34)/120 = 16 mA.$ , que se ha determinado como la intensidad óptima después de realizar varias pruebas en este trabajo.

### 4.2.2 Etapa de filtrado y amplificación

Al diseñar el circuito acondicionador de la señal plestimográfica se tiene que tener en cuenta la variable que se quiere medir. En este caso, el interés recae sobre la frecuencia cardíaca.

El rango de frecuencias entre las que puede variar el corazón humano va desde 40 hasta 220 pulsaciones por minuto, por lo que se incluye un filtrado pasabanda para eliminar frecuencias fuera de este rango.

El circuito cuenta con dos etapas similares, incluyendo cada una la misma configuración: Un filtro pasa alto pasivo con una frecuencia de corte de 0,7 Hz, y un filtro pasa bajo activo, con frecuencia de corte 2,34 Hz, y una ganancia de 101.

El filtro paso bajo está formado por una resistencia  $R = 47 k\Omega$  y  $C = 4,7 \mu F$ , con un frecuencia de corte de  $f_c = \frac{1}{2\pi RC} = 0,7 Hz.$

El filtro paso alto está formado por una resistencia de realimentación  $R_F = 680 k\Omega$ ,  $C = 100 nF$  y resistencia de inversor  $R_i = 6,8 k\Omega$ , con lo que se obtiene:

$$f_c = \frac{1}{2\pi R_F C} = 2,34 Hz; \text{Ganancia del sistema } G = 1 + \frac{R_F}{R_i} = 101$$

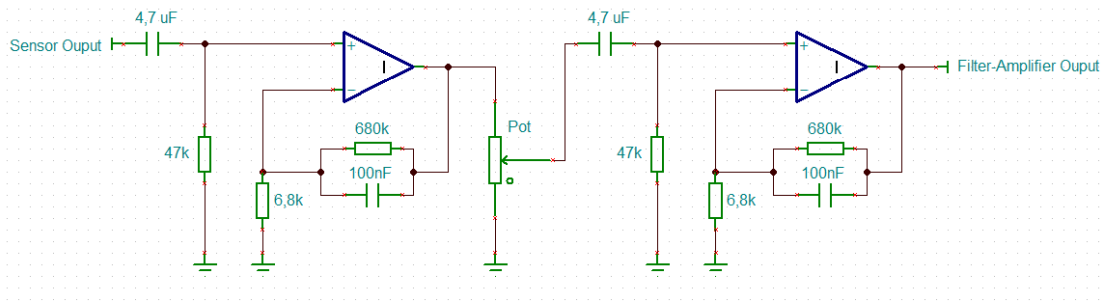


Ilustración 4-3. Circuito de filtrado y amplificación

En una primera etapa se filtra la señal y se amplifica con un factor de 101. En la segunda etapa se vuelve a realizar el filtrado, por si al amplificar alguna frecuencia no deseada se inserta en la señal, entonces se vuelve a amplificar. De este modo se consigue un filtro de un mayor orden y más restrictivo.

La implementación de las dos etapas de amplificación busca una señal que sature ante pulsos dentro de la frecuencia del rango deseado. Así se obtienen pulsos digitales cada vez que aparece un pulso cardíaco. La diferencia de tiempo entre estos pulsos cardíacos permitirá más adelante el cálculo de la frecuencia cardíaca.

### 4.2.3 Circuito completo y salida digital

Uniéndolos ambos bloques da lugar al circuito completo. Este circuito puede ser alimentado a una tensión de 3,3 V, que coincide con el voltaje de alimentación del módulo BLE. Esto permite alimentar todo el sistema desde una única batería o pila, haciéndolo más portable, característica necesaria en los sistemas *wearables*.

La salida del circuito corresponderá a una señal analógica, pero que actúa como una digital, asociando un pulso de nivel lógico alto cada vez que se produce un latido. Una característica deseable en los amplificadores operacionales es que tengan la capacidad *rail to rail*, para que permitan tener en la salida un voltaje muy próximo al voltaje con el que son alimentados, permitiendo así que el pulso tenga también un voltaje de 3,3V que corresponde a un nivel lógico alto.

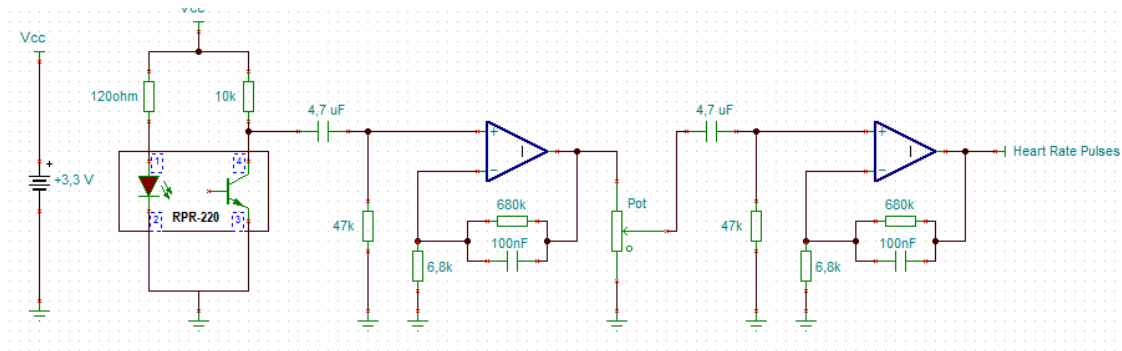


Ilustración 4-4. Circuito completo del sensor y adquisición de señal

La salida estará conectada a un pin del módulo Sensor Tag, que ejecutará una rutina de interrupción cada vez que llegue una nueva señal, es decir, se conecta a una entrada digital en la que se configura la interrupción asociada.

### 4.3 Descripción del firmware del módulo CC2541 Sensor Tag

En esta sección se describen los componentes software desarrollados para el dispositivo empotrada. Para facilitar la comprensión del desarrollo realizado para este dispositivo, la descripción se dividirá en varias secciones.

#### 4.3.1 Creación de un perfil personalizado

Como se especifica en el capítulo anterior, Bluetooth SIG define perfiles para dar una amplia gama de funcionalidad, tanto en las capas de control como en las de datos. Existen perfiles especialmente diseñados para aplicaciones de e-health, como pueden ser el perfil de glucosa, de temperatura, de ritmo cardíaco, etc. Aunque el perfil para intercambio de datos sobre el ritmo cardíaco está definido, y es conveniente usar este para mantener compatibilidad con otros dispositivos, en el desarrollo de este proyecto se va a diseñar un perfil personalizado con fines académicos.

Los perfiles específicos se limitan actualmente a perfiles basados en GATT. Esto significa que todos estos perfiles utilizan los procedimientos y modelos operativos del perfil del GATT como base para todas las aplicaciones.

Cada perfil define la solución total al describir el comportamiento de ambos extremos del enlace, el servidor GATT y el cliente GATT, así como los servicios que lo



contienen en el servidor del GATT. Un servicio define una colección de características y cómo se usan. Un perfil define una colección de uno o más servicios y define cómo se pueden usar los servicios para habilitar una aplicación o caso de uso.

El firmware que se ejecuta en el *Sensor Tag* actúa como cliente GATT. Habitualmente, el cliente del GATT toma la iniciativa y el servidor del GATT responde, pero con indicaciones y notificaciones el servidor tomará la iniciativa de notificar a un cliente del GATT que ha cambiado un valor característico, evitando que el cliente tenga que sondear el atributo de valor.

En la Figura 4-5 se puede ver la jerarquía genérica para perfiles basados en GATT. El perfil es la estructura principal, mientras que los servicios definen el contenido dentro del perfil.

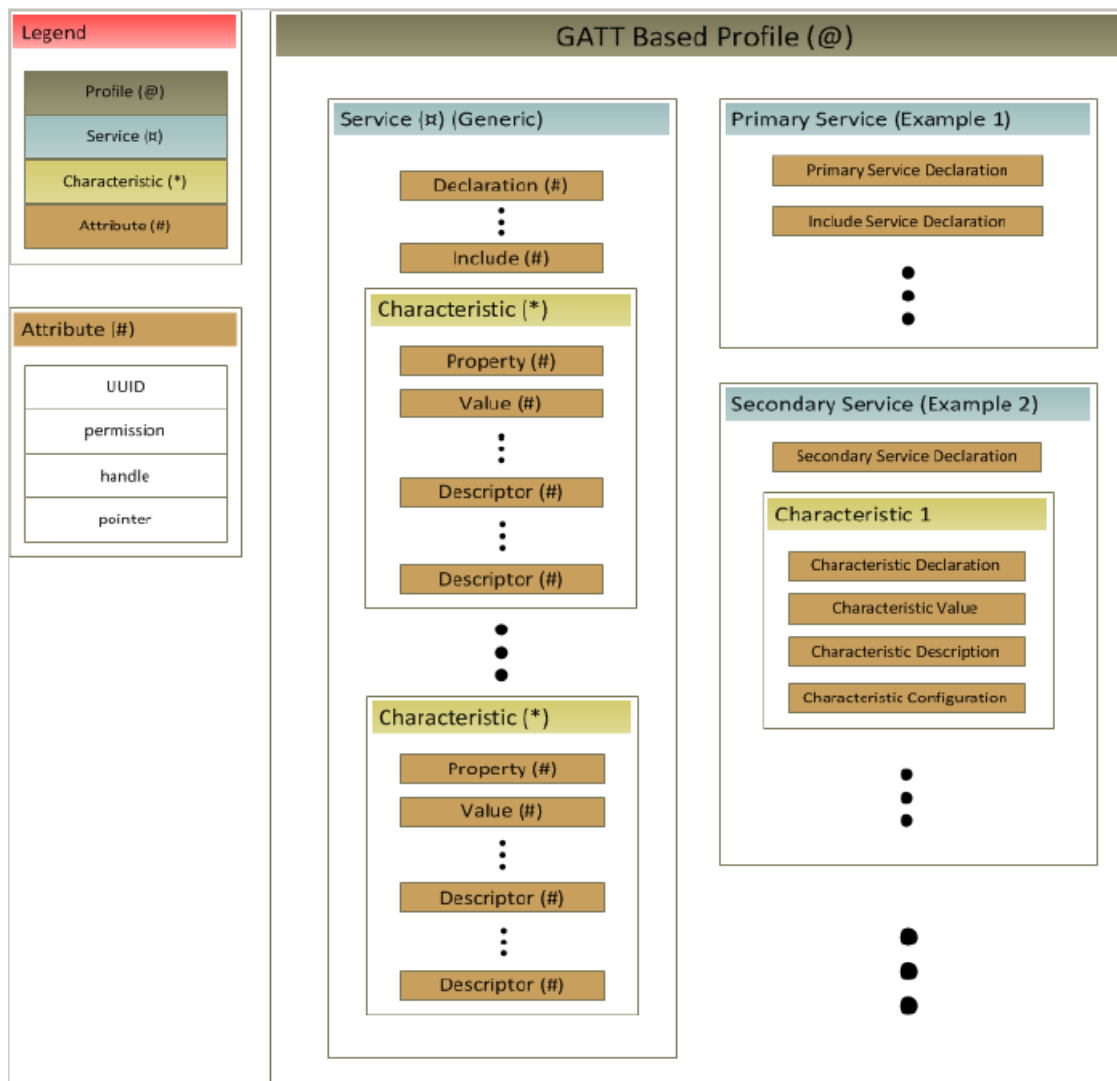


Ilustración 4-5. Jerarquía de un perfil basado en GATT

Para crear un perfil personalizado, básicamente hay que definir una relación de función y servicio. Para crear un servicio personalizado, se debe definir una tabla de atributos del GATT específica con las características requeridas y cómo se usan. Además, para las nuevas características que aún no están definidas por Bluetooth SIG, se deben crear nuevos identificadores exclusivos universalmente únicos (UUID) de 128 bits para los atributos.

El perfil no se ha construido desde cero, sino que se ha utilizado un perfil genérico ya creado (SimpleBLEPeripheral) y se ha editado para que se adapte a las necesidades, reaprovechando la estructura y las funciones del perfil.

El perfil se conforma por dos archivos: JFBS\_HeartRateSensor.c y JFBS\_HeartRateSensor.h.

#### 4.3.1.1 Definición del servicio

Las siguientes características se van a definir en el servicio:

Característica	Propiedades	Descripción
Characteristic 1	Lectura	Valor del “ritmo cardíaco”
Characteristic 2	Notificación	Notificación del valor del “ritmo cardíaco”
Characteristic 2 Client Characteristic Configuration descriptor	Lectura, Escritura	Configuración de notificaciones del valor del “ritmo cardíaco”

Tabla 4-1. Características del Servicio

La Característica 1 lee y almacena el valor del ritmo cardíaco.

La Característica 2 copia el valor del ritmo cardíaco desde la característica 1 y lo notifica al servidor cada vez que sufre un cambio.

La característica *Characteristic 2 Client Characteristic Configuration descriptor* no es una característica separada, sino que forma parte de la característica 2. Se muestra resaltada para mostrar la funcionalidad de habilitar las notificaciones.

En el archivo JFBS\_HeartRateSensor.h se crean las definiciones de las características y UUID. Como se utilizan perfiles personalizados, es necesario utilizar UUID de 128 bits en lugar de los comunes de 16 bit. Para ello, se utiliza una base aleatoria definida por Texas Instruments a la que se le agregan los últimos 16 bits.

```
// Profile Parameters
#define JFBS_HEARTRATE_CHAR1      0 //Profile Characteristic 1 value
#define JFBS_HEARTRATE_2_MEASURE  1 //Profile Characteristic 2 value
#define JFBS_HEARTRATE_2_MEASURE_CHAR_CFG  3

// UUID for JFBS_heartrate service
#define JFBS_HEARTRATE_SERV_UUID      0xC0C0
/* The 16 bit UUID listen above is only a part of the
 * full TI random 128 bit UUID:
 * F000C0C0-0451-4000-B000-000000000000. */

// JFBS_Heartrate Service characteristic UUID
#define JFBS_HEARTRATE_CHAR1_UUID      0xC0C1
#define JFBS_HEARTRATE_2_MEASURE_UUID  0xC0C2
```

Listado 4-1. Código fuente asociado a las características y UUIDs.

En JFBS\_HeartRateSensor.c, se crean las variables de los atributos del servicio:

```
/* *****
 * Service Attributes - variables
 */
// JFBS_Heartrate Service attribute
static CONST gattAttrType_t JFBS_heartrateService = { ATT_UUID_SIZE,
JFBS_heartrateServUUID };

// +++ JFBS_Heartrate Service Characteristic 1 Properties
// + Characteristic 1 Properties
static uint8 JFBS_heartrateChar1Props = GATT_PROP_READ | GATT_PROP_WRITE;
// + Characteristic 1 Value
static uint8 JFBS_heartrateChar1 = 0;
// + Service Characteristic 1 User Description
static uint8 JFBS_heartrateChar1UserDesp[22] = "JFBS_Heartrate Value\0";

// +++ JFBS_Heartrate Characteristic 2 MEASURE VALUE
// + Service Characteristic 2 Properties
static uint8 JFBS_heartrateChar2Props = GATT_PROP_NOTIFY;
// + Characteristic 2 Value
static uint8 JFBS_heartrateChar2 = 0;
// + Service Characteristic 2 Configuration.
static gattCharCfg_t JFBS_heartrateChar2Config[GATT_MAX_NUM_CONN];
// + Service Characteristic 2 User Description
static uint8 JFBS_heartrateChar2UserDesp[] = "JFBS_Heartrate Value
Notification\0";

// JFBS_Heartrate Service characteristic UUID
#define JFBS_HEARTRATE_CHAR1_UUID      0xC0C1
#define JFBS_HEARTRATE_2_MEASURE_UUID  0xC0C2
```

**Listado 4-2. Creación tabla de atributos del servicio**

En el Listado 4-3 se recoge la versión simplificada de la tabla de atributos del servicio. En ella se declara de que tipo es cada atributo dependiendo de su UUID (característica, descriptor, etc.), los permisos que tiene, una dirección de manejador, y al puntero al valor que tienen.

```

/*****
 * Profile Attributes - Table
 */

static gattAttribute_t JFBS_heartRateAttrTbl[SERVAPP_NUM_ATTR_SUPPORTED] =
{
    // JFBS_HeartRate Service
    {
        { ATT_BT_UUID_SIZE, primaryServiceUUID }, /* type */
        GATT_PERMIT_READ, /* permissions */
        0, /* handle */
        (uint8 *)&JFBS_heartRateService /* pValue */
    },
    // Characteristic 1 Declaration
    {
        { ATT_BT_UUID_SIZE, characterUUID }, GATT_PERMIT_READ ,
        0,
        &JFBS_heartRateChar1Props
    },
    // Characteristic Value 1
    {
        { ATT_UUID_SIZE, JFBS_heartRateChar1UUID },
        GATT_PERMIT_READ | GATT_PERMIT_WRITE,
        0,
        &JFBS_heartRateChar1
    },
    . . .
}

```

**Listado 4-3. Código con los parámetros de la tabla de atributos**

Con la configuración utilizada, queda definida la tabla de atributos del servicio como:

Tipo (hex)	Tipo (#DEFINE)	Valor (por defecto)	Permisos
0x2800	GATT_SERVICE_UUID	0xC0C0 (JFBS_HEARTRATE_SERV_UUID)	GATT_PERMIT_READ
0x2803	GATT_CHARACTER_UUID	(Write and read permission) (UUID 0xC0C1)	GATT_PERMIT_READ
0xC0C1	JFBS_heartratechar1UUID	0	GATT_PERMIT_READ   GATT_PERMIT_WRITE
0x2901	GATT_CHAR_USER_DESC_UUID	"JFBS_Heartrate Value"	GATT_PERMIT_READ
0x2803	GATT_CHARACTER_UUID	(Notify Permission) (UUID 0xC0C1)	GATT_PERMIT_READ
0xC0C2	JFBS_heartratechar2UUID	0	0
0x2902	GATT_CLIENT_CHAR_CFG_UUID	0x0000	GATT_PERMIT_READ   GATT_PERMIT_WRITE
0x2901	GATT_CHAR_USER_DESC_UUID	"JFBS_Heartrate Value Notification"	GATT_PERMIT_READ

Tabla 4-2. Tabla de atributos del servicio

#### 4.3.1.2 Funciones del Perfil

El perfil también incluye un conjunto de funciones para el intercambio de datos y configuración del perfil, entre ellas se han definido:

- Añadir el servicio: *JFBS\_Heartrate\_AddService(services);*
- Registrar los callbacks de la app: *JFBS\_Heartrate\_RegisterAppCBs(\*appCallbacks);*
- Establece el valor de un atributo: *JFBS\_Heartrate\_SetParameter(param, len, \*value);*
- Lee el parámetro de una atributo: *JFBS\_Heartrate\_GetParameter(param, \*value);*
- Cambios en el estado de la conexión:  
*JFBS\_heartrate\_HandleConnStatusCB(connHandle, changeType);*
- Envía una notificación que contiene el ritmo cardíaco:  
*JFBS\_HeartRate\_MeasureNotify(connHandle, \*pNoti);*

Estas funciones se editan a partir del modelo utilizado *SimpleBLEPeripheral.c*, que incluye funciones similares con el mismo fin. Las modificaciones más importantes pasan por adaptar los nombres de las funciones y variables, y adaptar al comportamiento al número y función de cada uno de los atributos.

A continuación se muestra la función de establecer el valor de un atributo, que establece un valor en una característica, y envía una notificación en caso de que esta esté configurada. Si la función se utiliza con la característica 1, guarda en ella un valor, si lo hace en la característica 2, además de guardarlo también lo notifica si tienen esta opción habilitada.

```

bStatus_t JFBS_Heartrate_SetParameter( uint8 param, uint8 len, void *value )
{
switch ( param )
{
case JFBS_HEARTRATE_CHAR1:
if ( len == sizeof ( uint8 ) )
{ JFBS_hearttrateChar1 = *((uint8*)value); }
break;

case JFBS_HEARTRATE_2_MEASURE:
if ( len == sizeof ( uint8 ) )
{
JFBS_hearttrateChar2 = *((uint8*)value);
// See if Notification has been enabled
GATTServApp_ProcessCharCfg( JFBS_hearttrateChar2Config, &JFBS_hearttrateChar2,
FALSE, JFBS_hearttrateAttrTbl, GATT_NUM_ATTRS( JFBS_hearttrateAttrTbl
), INVALID_TASK_ID);
}
break;
}
}
}

```

**Listado 4-4. Función para establecer un parámetro en una característica**

También se hace necesario incorporar una función para el tratamiento de las UUIDs de 128 bits y su adaptación a versiones de 16 bits. Para el correcto uso de esta función y de las UUIDs extendidas será necesario implementar la directiva `GATT_TI_UUID_128_BIT` en el preprocesador del compilador. La siguiente función permite extraer la versión de 16 bits a partir de la de 128 bits:

```

bStatus_t utilExtractUuid16 ( gattAttribute_t *pAttr, uint16_t *pUuid )
{
bStatus_t status = SUCCESS;
if ( pAttr->type.len == ATT_BT_UUID_SIZE )
{
// 16-bit UUID direct
*pUuid = BUILD_UINT16( pAttr->type.uuid[0], pAttr->type.uuid[1]);
#ifdef GATT_TI_UUID_128_BIT
}
else if ( pAttr->type.len == ATT_UUID_SIZE )
{
// 16-bit UUID extracted bytes 12 and 13
*pUuid = BUILD_UINT16( pAttr->type.uuid[12], pAttr->type.uuid[13]);
#endif
} else {
*pUuid = 0xFFFF;
status = FAILURE;
}
return status;
}

```

**Listado 4-5. Función que extrae UUID de 16 bits a partir de 128 bits**

### 4.3.2 *Desarrollo del código de la aplicación*

La aplicación tiene dos rutinas específicas para el funcionamiento de este proyecto. El resto de funciones son heredadas de la aplicación *SimpleBLEPeripheral*.

Existe una rutina de atención de interrupciones GPIO. Esta detecta cada vez que llega un flanco a un pin proveniente del circuito de adecuación y amplificación de la señal de ritmo cardíaco. Recién comenzada a ejecutar la rutina se guarda el valor de tiempo, y hace la diferencia con el valor de tiempo anterior para obtener el intervalo de tiempo entre pulsos. Este valor se almacena en un array de 16 elementos, desechando el valor más antiguo.

Otra función, se ejecuta periódicamente cada vez que un *timer* alcanza el valor indicado. Esta función (*PeriodicTask*) promedia los valores guardados en ese momento en el array que guarda los intervalos de pulsos cardíacos, con el valor promediado calcula el valor de ritmo cardíaco y lo guarda en la Característica 2. Si la opción de notificación está activada, se notifica al servidor que hay un nuevo valor y transmite el dato.

El código del programa se desarrolla en cuatro archivos:

- *JFBS\_HeartRateSensor\_APP\_Main.c*: Contiene la función *main* donde comienza la ejecución. Sólo se ejecuta una vez al inicio, y su misión es inicializar el hardware, las diferentes capas del protocolo, el OSAL, habilitar interrupciones, etc. Se utiliza el archivo diseñado anteriormente correspondiente a *SimpleBLEPeripheral\_Main.c* y no es necesaria la edición. Al finalizar invoca la ejecución del OSAL.
- *JFBS\_OSAL\_HeartRateSensor.c*: El OSAL actúa como sistema operativo, ejecutando tareas y adjudicándoles prioridades. Es necesario por tanto incorporar tanto en la inicialización como en la función de gestión de eventos de tareas, la tarea correspondiente al programa.

Se inserta al final de la función de gestión de eventos, la tarea correspondiente al procesamiento de eventos de la aplicación:

```

const pTaskEventHandlerFn tasksArr[] =
{
    LL_ProcessEvent,                // task 0
    Hal_ProcessEvent,              // task 1
    HCI_ProcessEvent,              // task 2
#if defined ( OSAL_CBTIMER_NUM_TASKS )
    OSAL_CBTIMER_PROCESS_EVENT( osal_CbTimerProcessEvent ), // task 3
#endif
    L2CAP_ProcessEvent,            // task 4
    GAP_ProcessEvent,              // task 5
    GATT_ProcessEvent,            // task 6
    SM_ProcessEvent,              // task 7
    GAPRole_ProcessEvent,         // task 8
    GAPBondMgr_ProcessEvent,      // task 9
    GATTServApp_ProcessEvent,     // task 10
    JFBS_HeartRateSensor_ProcessEvent // task 11
};

```

También al final de la función de inicialización del OSAL es necesario insertar la llamada a la función de inicialización de la aplicación:

```

void osalInitTasks( void )
{
    uint8 taskID = 0;

    . . .

    /* Profiles */
    GAPRole_Init( taskID++ );
    GAPBondMgr_Init( taskID++ );
    GATTServApp_Init( taskID++ );
    /* Application */
    JFBS_HeartRateSensor_Init( taskID );
}

```

- *JFBS\_HeartRateSensor\_App.h*: Es el fichero de cabecera de la aplicación, incluye las definiciones de las funciones y la declaración de los eventos:

```

// JFBS_HeartRateSensor_Task_Events
#define JFBS_HR_START_DEVICE_EVT          0x0001
#define JFBS_HR_PERIODIC_EVT             0x0002
#define JFBS_HR_KEY_CHANGE_EVT           0x0010

```

- *JFBS\_HeartRateSensor\_App.c*: Contiene el código principal de la aplicación explicado al principio del apartado, la inicialización del programa, se configuran los paquetes de anuncio del dispositivo, se configuran los periféricos y se añaden todas las funciones y rutinas que sean necesarias incorporar a la aplicación.



Las rutinas incluidas en este archivo que describen el funcionamiento principal son la rutina de ejecución de tareas periódicas (*JFBS\_heartRatePeriodicTask*) y la rutina de atención de interrupciones:

```
/******  
* @fn      JFBS_heartRatePeriodicTask  
*  
* @brief   Perform a periodic application task. This function gets  
*          called every five seconds as a result of the JFBS_HR_PERIODIC_EVT  
*          OSAL event. In this program: Perform a periodic heart rate  
*          application task.  
*/  
static void JFBS_heartRatePeriodicTask( void )  
{  
    JFBS_AVG_MSECS=0;  
  
    for ( int i = 0; i < 16; i++ )  
        {  
            JFBS_AVG_MSECS =JFBS_AVG_MSECS + JFBS_VECT_DIFF_MSECS[i];  
        }  
    JFBS_AVG_MSECS=JFBS_AVG_MSECS/16;  
    JFBS_BPM=(1/(JFBS_AVG_MSECS/1000))*60;  
  
    JFBS_Heartrate_SetParameter(JFBS_HEARTRATE_2_MEASURE , sizeof(uint8_t),  
                                &JFBS_BPM);  
  
    // Restart timer  
    osal_start_timerEx( JFBS_heartRateSensor_TaskID, JFBS_HR_PERIODIC_EVT,  
JFBS_DEFAULT_HEARTRATE_PERIOD );  
}
```

Lístando 4-5. Rutina de ejecución periódica

```
HAL_ISR_FUNCTION( halKeyPort0Isr, P0INT_VECTOR )  
{  
    HAL_ENTER_ISR();  
  
    if (HAL_KEY_SW_6_PXIFG & HAL_KEY_SW_6_BIT)  
        {  
            JFBS_NEW_MSECS=osal_GetSystemClock();  
        }  
  
    JFBS_DIFF_MSECS=JFBS_NEW_MSECS-JFBS_OLD_MSECS;  
  
    if(JFBS_DIFF_MSECS>285 & JFBS_DIFF_MSECS<1714)  
        {  
            for ( int i = 16; i > 0; i-- )  
                {  
                    JFBS_VECT_DIFF_MSECS[i]=JFBS_VECT_DIFF_MSECS[i-1];  
                    JFBS_VECT_DIFF_MSECS[0]=JFBS_DIFF_MSECS;  
                }  
        }  
    JFBS_OLD_MSECS=JFBS_NEW_MSECS;  
  
    HAL_KEY_SW_6_PXIFG = 0;  
    HAL_KEY_CPU_PORT_0_IF = 0;  
  
    CLEAR_SLEEP_MODE();  
    HAL_EXIT_ISR();  
    return;  
}
```

**Listado 4-6. Rutina de tratamiento de interrupciones**

## ***4.4 Descripción de la App desarrollada en Android***

Tras haber montado el sistema de medida y haber sido capaces de programar el *Sensor Tag* para que obtenga las pulsaciones por minuto y las envíe por Bluetooth, hay que crear la aplicación para Android que permita recibir las pulsaciones del *Sensor Tag* por Bluetooth y mostrarlas en tiempo real.

### ***4.4.1 Creación del proyecto en Android Studio***

Lo primero será crear un nuevo proyecto siguiendo los pasos explicados en el capítulo anterior. Para ello, tras abrir Android Studio, se pulsa en “Start a new Android Studio project”. En la primera ventana se establece el nombre de la aplicación. En este proyecto se ha utilizado como nombre “BLE”. Se elige la carpeta donde guardar el proyecto y se pulsa en “Next”.

Ahora se tiene que elegir la plataforma o plataformas en las que se ejecutará la aplicación. En este proyecto se ha elegido “Phone and Tablet”, y la versión mínima del SDK se ha establecido en la API 19, correspondiente a Android 4.4 KitKat, dado que esta versión soporta el uso del Bluetooth Low Energy. Una vez elegida se hace click sobre “Next”.

La siguiente pantalla permite elegir el tipo de Activity con el que comenzará la aplicación. En este proyecto se ha elegido la “Blank Activity”. Por lo que se selecciona y se hace click en “Next”.

Se llegará a la última pantalla, en la que se debe dar un nombre a la actividad. En este proyecto se ha dejado el nombre que viene por defecto, “MainActivity”. Se hace click en “Finish”, y comenzará la creación del proyecto y su posterior compilación para verificar que todos los archivos se han creado correctamente.

Cuando la creación del proyecto haya terminado, en la ventana principal de Android Studio se habrá abierto el fichero “MainActivity.java”.

## ***4.4.2 Estructura de la aplicación de Android***

Con el proyecto ya creado, se van a comentar los distintos ficheros de los que va a estar formado este proyecto, ya que van a ser los que definan la interfaz de la aplicación y las tareas que va a llevar a cabo.

El proyecto se ha dividido en dos archivos “.java”:

- **MainActivity.java**: este archivo contiene la actividad principal, y es el primero que se ejecuta cuando se abre la aplicación. Se va a encargar de mostrar la interfaz de usuario, lanzar el servicio del Bluetooth, lanzar la orden de conectarse al *Sensor Tag* y actualizar las pulsaciones por minuto en la interfaz gráfica cuando las reciba del servicio de Bluetooth.
- **BluetoothLeService.java**: este archivo contiene la definición del servicio que se va a ejecutar de forma paralela a la actividad principal. Este servicio es el que realmente va a realizar todas las operaciones con Bluetooth. Se va a encargar de iniciar la conexión con el dispositivo, de descubrir sus servicios y características, de habilitar las notificaciones y de recibir el valor de las pulsaciones que se envíen. Este servicio será el intermediario a la hora de transferir datos y gestionar la conexión entre la actividad principal y el dispositivo Bluetooth.

Junto con estos archivos, se contará con un “activity\_main.xml” en el que se podrá diseñar la interfaz de la aplicación, y con un “AndroidManifest.xml”, en el que se podrá asegurar que las activities, servicios y content providers se tendrán en cuenta de forma correcta a la hora de compilar la aplicación.

## ***4.4.3 Programación del archivo MainActivity.java***

En este archivo comienza creando una clase llamada “MainActivity”, que extiende de la clase “ActionBarActivity”. Dentro de esta clase, se definen las variables que se van a utilizar en la actividad. Las dos primeras se llaman “Pulsaciones” y “Posicion”, y son de tipo “TextView”. Permiten modificar posteriormente los valores de las pulsaciones por minuto en la interfaz de la aplicación.

La tercera variable es el objeto “mBluetoothAdapter”. Este objeto es de tipo “BluetoothAdapter”, y va a permitir que la aplicación pueda utilizar el Bluetooth del dispositivo. A su vez, se ha creado de tipo “public static” para que pueda ser utilizada desde “BluetoothLeService.java” sin necesidad de crear un objeto.

La cuarta variable es el objeto “mBluetoothLeService”. Este objeto es de tipo “BluetoothLeService” (el servicio que se va a ejecutar en segundo plano), por lo que se utilizará para ejecutar métodos definidos en “BluetoothLeService.java”. Es la usada para realizar una comunicación con el servicio de Bluetooth (que va a estar ejecutándose en segundo plano) y poder realizar una conexión con el dispositivo Bluetooth.

La última variable es el objeto “mServiceConnection”. Este objeto es de tipo “ServiceConnection”, y se utiliza para manejar los eventos producidos cuando el servicio se crea o se destruye.

El método “onCreate” se ejecuta cuando se abra la aplicación y se lanza la actividad principal. Dentro de este método se establece en primer lugar la apariencia de la aplicación, usando el archivo activity\_main.xml para cargar los datos de la interfaz mediante el siguiente código:

```
public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mPulsaciones= (TextView) findViewById(R.id.pulsaciones);
    mPosicion= (TextView) findViewById(R.id.posicion)
```

En cuanto se crea la interfaz, lo siguiente es obtener la referencia a los dos “TextView” que se han creado en la interfaz, esto permite modificarlos posteriormente. “Pulsaciones” y “posición” es el ID que se le ha dado a estos “TextView” en el archivo “activity\_main.xml”.

Una vez que se ha tratado lo referente a la interfaz gráfica de la actividad, se inicia el adaptador Bluetooth mediante el *BluetoothManager*, y una vez obtenido al adaptador, se verifica si el Bluetooth está encendido. Si no es así, se lanza un *intent* para que el sistema Android solicite habilitar el Bluetooth.

```
// tomar la referencia a BluetoothAdapter a través del BluetoothManager
final BluetoothManager mBluetoothManager=
    (BluetoothManager) getSystemService(BLUETOOTH_SERVICE);
mBluetoothAdapter=mBluetoothManager.getAdapter();

//Esta Bluetooth encendido?
if (!mBluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

Lo último que hay que hacer dentro del método “onCreate” es crear un “intent” con el que se podrá lanzar el servicio de Bluetooth. A este intent habrá que indicarle la

actividad en la que está y la clase en la que hay que basarse para crear el servicio. Por lo que se utilizará este código:

```
Intent gattServiceIntent= new Intent(this, BluetoothLeService.class);
bindService(gattServiceIntent, mServiceConnection, BIND_AUTO_CREATE);
```

Para lanzar el servicio, se utiliza “bindService”, ya que se quiere enlazar el servicio con esta actividad para poder intercambiar datos entre ellas. Además, esto permite que cuando desaparezca la actividad principal, el servicio también desaparezca, y no se quede en segundo plano. Como parámetro se ha pasado “mServiceConnection”, ya que se va a utilizar para poder conocer que el servicio se ha creado o se ha cerrado. Una vez que el servicio se haya creado, “mServiceConnection” llama al método “onServiceConnected”, mediante el cual se obtienen un objeto del servicio que se acaba de lanzar. Tras haber obtenido el objeto del servicio, se llama a su método “connect” para conectarse al *Sensor Tag*, para lo cual se le pasa como parámetro la dirección MAC del dispositivo.

```
bindService(gattServiceIntent, mServiceConnection, BIND_AUTO_CREATE);
mBluetoothLeService=((BluetoothLeService.LocalBinder) service).getService();
mBluetoothLeService.connect(mDeviceAddress);
```

Con esto el resto de la ejecución de la aplicación correría a cargo del servicio de Bluetooth que se ha lanzado, por lo que en la actividad principal sólo hay que quedarse a la espera de que el servicio de Bluetooth informe con alguna novedad sobre el estado de la conexión o mande algún dato.

El servicio de Bluetooth informará de estos eventos utilizando un “Broadcast Update”, por lo que la actividad principal deberá quedarse a la espera de recibir uno de estos mensajes mediante la creación de un “Intent Filter”.

Se define por tanto un *intent filter* llamado “intentFilter”, al que se le añadirán las “acciones” a las que debe prestarles atención. Esto hará que el “intent filter” le preste atención a un “Broadcast Update” enviado por el servicio Bluetooth en el que se indique que el dispositivo móvil se acaba de conectar dispositivo.

```
private static IntentFilter makeGattUpdateIntentFilter() {
    final IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction(BluetoothLeService.ACTION_GATT_CONNECTED);
    intentFilter.addAction(BluetoothLeService.ACTION_GATT_DISCONNECTED);
    intentFilter.addAction(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED);
    intentFilter.addAction(BluetoothLeService.ACTION_DATA_AVAILABLE);
    return intentFilter;
}
```

Tras haber creado el filtro para prestarle atención solamente a los “Broadcast Update” que se hayan definido, se deberá crear un “BroadcastReceiver” llamado “mGattUpdateReceiver” en el cual se realizará una acción en función del “Broadcast Update” que se haya recibido.

Antes de definir las acciones que realizará el “BroadcastReceiver”, se deberá asociar el “Intent Filter” al “BroadcastReceiver. Esto se realiza con el método “onResume()”, el cual se ejecuta después del “onCreate()”, y justo antes de que la actividad comience a ejecutarse.

```
public void onResume(){  
    registerReceiver(mGattUpdateReceiver, makeGattUpdateIntentFilter() );  
}
```

Dentro del “mGattUpdateReceiver” hay un método llamado “onReceive”, el cual se ejecuta cuando el “Intent Filter” detecta una acción de las que se le han asociado.

Mediante “intent.getAction()” se comprueba el motivo por el que se ha recibido el “BroadcastReceiver”. La acción que interesa es “ACTION\_DATA\_AVAILABLE”, la cual indica que se acaba de recibir un dato desde el dispositivo conectado. Este dato recibido es necesario guardarlo.

```
final int dato_a_actualizar =  
intent.getStringExtra(BluetoothLeService.EXTRA_DATA);
```

Es necesario extraer el dato que hay adjunto al *intent*. Una vez que se tiene el dato con las pulsaciones, hay que proceder a actualizar el “TextView” de la interfaz que se encarga de mostrar las pulsaciones por minuto.

```
Pulsaciones.setText(String.valueOf(dato_a_actualizar));
```

Hay que tener en cuenta que el dato recibido es tipo “int”, y se quiere establecer el texto de un “Text View”. Mediante el método “String.valueOf()” se puede extraer una cadena de texto de la variable de tipo “int”.

Una vez que se ha actualizado la interfaz gráfica mostrando el valor más reciente de las pulsaciones, se muestra un histórico de todos los valores recibidos desde que se estableció la conexión. Puesto que esta función es un añadido, no se desarrolla en este proyecto su funcionamiento.

#### 4.4.4 Programación del archivo *BluetoothLeService.java*

En este archivo lo primero será crear una clase llamada “BluetoothLeService”, la cual se extiende de “Service”, por lo tanto será una clase destinada a definir el servicio que se ha lanzado para controlar el Bluetooth. Lo siguiente será definir las variables que se van a utilizar en este fichero.

El primer objeto se llamará “mBinder” y será de tipo “IBinder”, el cual se utilizará cuando se cree o se destruya el servicio.

En segundo lugar se creará un objeto de tipo “BluetoothGatt” que se ha llamado “mBluetoothGatt”. Este objeto se usará para realizar acciones concretas sobre el *Sensor Tag*, como puede ser mandarle una orden para conectarse a él, o mandarle una petición de escritura o de lectura en una característica.

El siguiente objeto se va a utilizar solamente una vez, para poder activar las notificaciones en el *Sensor Tag*. Se llama “característica\_pulsaciones” y es de tipo “BluetoothGattCharacteristic”. Junto a este objeto se definirán otros tres, llamados “PULSACIONES\_SERVICE”, “PULSACIONES\_CHARACTERISTIC” y “DESCRIPTOR”, los cuales contendrán el UUID del servicio, característica y descriptor (respectivamente) que se han creado en el dispositivo para almacenar el valor de las pulsaciones.

Por último, se definirán cinco cadenas de texto que contendrán el mensaje que el “BroadcastUpdate” de este servicio enviará, para que el “BroadcastReceiver” de la actividad principal pueda ir enterándose de los eventos que se van produciendo en lo que a la conexión Bluetooth se refiere. Las cadenas de texto que se han definido han sido “ACTION\_GATT\_CONNECTED” para cuando la aplicación se conecte por Bluetooth con el dispositivo, “ACTION\_GATT\_DISCONNECTED” para cuando la aplicación se desconecte del dispositivo, “ACTION\_GATT\_SERVICES\_DISCOVERED” para cuando la aplicación haya terminado de escanear todos los servicios, “ACTION\_DATA\_AVAILABLE” para cuando el *Sensor Tag* envíe el valor de alguna característica, “EXTRA\_DATA” para incluir la característica que se haya recibido del dispositivo en el “intent” que se enviará a la actividad principal.

Tras definir las variables, se va a comentar los métodos creados en este archivo. Los dos primeros se van a utilizar cuando se cree o se destruya el servicio, son “onBind” y “onUnbind”. “onBind” devolverá el objeto “mBinder”, el cual se usará en la actividad principal para poder comunicarse con el servicio. En cuanto a “onUnbind”, llamará a un método llamado “close()” que cerrará correctamente la conexión Bluetooth con el dispositivo.

El siguiente método de interés es el método “connect(final String address)”, el cual es llamado desde la actividad principal cuando esta recibe el aviso de que el servicio se ha creado. En el método “connect” se va a crear un “device” de tipo “BluetoothDevice”, al que se le pasará la dirección MAC enviada desde la actividad principal, la cual se usará con el objeto “mBluetoothAdapter” que se ha creado en la actividad principal. Esto generará un objeto que contendrá toda la información necesaria para iniciar la conexión con el dispositivo. Ahora sólo falta utilizar el objeto “mBluetoothGatt” para administrar las acciones que se quieren realizar sobre el *Sensor Tag*, para lo que se utiliza el siguiente código:

```
mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
```

Se observa como el método “connectGatt” lanza un objeto llamado “mGattCallback” de la clase “BluetoothGattCallback”, la cual es una “callback function” que lanzará uno de sus métodos cuando se produzca algún evento en la conexión Bluetooth con el *Sensor Tag*.

La última parte del archivo que queda por tratar es lo que ocurre en el objeto “mGattCallBack” cuando recibe algún evento del Bluetooth del dispositivo.

El primer método se llama “onConnectionStateChange”, y la aplicación entra en él cuando el estado de la conexión Bluetooth ha cambiado, por ello se deberá comprobar si el parámetro “newState” indica que el dispositivo móvil se acaba de conectar por Bluetooth al *Sensor Tag* (BluetoothProfile.STATE\_CONNECTED) o que el dispositivo móvil se acaba de desconectar (BluetoothProfile.STATE\_DISCONNECTED). En caso de que el dispositivo móvil se acabe de conectar, se iniciará el proceso de descubrir los servicios del *Sensor Tag* llamando al método “discoverServices()” del objeto “mBluetoothGatt” que se ha dicho que se utilizaría para ejecutar comandos de Bluetooth.

Cuando haya terminado el proceso de descubrir los servicios del *Sensor Tag*, la callback function ejecutará automáticamente el método “onServicesDiscovered”. Indica que ya se ha terminado el proceso de descubrir los servicios, por lo que ya se pueden activar las notificaciones en el *Sensor Tag* para que éste pueda mandar el valor actualizado de las pulsaciones cuando se produzca alguna.

Hay que tener en cuenta que el motivo por el que se ha realizado un escaneo antes de activar las notificaciones, es porque aunque se conozcan el UUID del servicio y de la característica a la que se quiere activar las notificaciones, el stack Bluetooth de Android no puede crear directamente un objeto de tipo descriptor con los UUID que se conocen y escribir en el *Sensor Tag*. Antes de realizar cualquier operación sobre un atributo del servidor (en este caso el *Sensor Tag*), el dispositivo con Android necesita haber realizado un escaneo para haber indexado los manejadores o *handles* de todos los servicios del dispositivo.



Por lo tanto, ahora ya se puede construir el objeto “característica\_pulsaciones” con el UUID del servicio y de la característica que contiene el dato con las pulsaciones mediante:

```
caracteristica_pulsaciones =  
mBluetoothGatt.getService(PULSACIONES_SERVICE).getCharacteristic(PULSA  
CIONES_CHARACTERISTIC);
```

Ahora se completa el objeto “descriptor” creado anteriormente con los datos de la característica a la que se le quieren activar las notificaciones, ya que cada característica tiene su propio descriptor. Y se lanza el comando para escribir en el descriptor, utilizando para ello el objeto “mBluetoothGatt”, el cual se encarga de realizar directamente operaciones con el dispositivo Bluetooth al que el dispositivo móvil se haya conectado.

```
mBluetoothGatt.setCharacteristicNotification(caracteristica_pulsaciones,  
true);  
  
BluetoothGattDescriptor descriptor =  
caracteristica_pulsaciones.getDescriptor(Descriptor);  
  
descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);  
  
mBluetoothGatt.writeDescriptor(descriptor);
```

El último método de la “callback function” al que hay que prestarle atención es a “onCharacteristicChanged”. Este método se lanzará cuando una característica del *Sensor Tag* haya cambiado (en este caso la característica con el valor de las Pulsaciones), por lo que se deberá lanzar un “BroadcastUpdate” indicando que hay nueva información disponible, así como adjuntar el dato recibido.

Esto se hace en primer lugar creando un “intent” que contendrá la acción “ACTION\_DATA\_AVAILABLE”, para que el “intent filter” de la actividad principal pueda prestarle atención, dado que es uno de los filtros que se ha definido en el “intent filter” de la actividad principal.

A continuación se creará una variable llamada “data” de tipo “byte[]” para poder almacenar el dato que ha enviado el *Sensor Tag* por Bluetooth. Lo siguiente será convertir ese byte de datos a una variable de tipo “int”, para que las pulsaciones puedan ser leídas en formato decimal fácilmente. El inconveniente aquí, es que el *Sensor Tag* ha mandado el valor de las pulsaciones en un byte sin signo, pero en java los byte tienen signo, así que es necesario realizar la conversión del tipo a la hora de almacenar el valor del byte dentro del “int”.

```
int value = (data[0] & 0xFF);
```

Por último se adjunta el “value” en el “intent” mediante el método “putExtra”, el cual sigue un esquema de “Clave – Valor”, por el que a la información almacenada en el *intent* se le asigna como clave “EXTRA\_DATA”. Finalmente se lanza el intent con:

```
sendBroadcast(intent);
```

#### ***4.4.5 Programación del archivo activity\_main.xml***

En este archivo se ha definido la interfaz gráfica de la aplicación. Dado que el lenguaje utilizado es XML, mediante una serie de etiquetas se van definiendo los distintos elementos que componen la interfaz.

En primer lugar hay que elegir como se van a disponer los elementos en la pantalla. En este proyecto se ha optado por un layout de tipo relativo, un “RelativeLayout”. Este tipo de layout distribuye los elementos mediante una serie de relaciones entre ellos, por lo que en vez de definir posiciones absolutas en la pantalla, se colocan una serie de elementos que se podrían considerar como los pilares de la interfaz, y el resto de elementos se van colocando en una posición relativa a estos, ya sea debajo, a la izquierda, o a una distancia determinada de ellos.

Unos segundos después de haber abierto la aplicación, el dispositivo con Android ya se habrá conectado por Bluetooth al *SensorTag* y habrá activado las notificaciones, por lo que automáticamente se debería empezar a recibir en tiempo real las pulsaciones por minuto recibidas.

En la interfaz tan sólo se utilizan cuatro “TextView”, los cuales son capaces de mostrar un texto en la interfaz. Se utilizan cuatro porque dos de ellos se van a utilizar a modo de texto que se quedará permanentemente indicando la unidad del dato que se está recibiendo, y los otros dos “TextView” se irán actualizando con el dato recibido, uno mostrará las pulsaciones por minuto, y el otro la posición del cuerpo en la que se encuentra el sensor, también recibido desde el dispositivo.

Por lo tanto, tras haberlos creado, hay que asignarles un “id”. Este “id” va a permitir que se puedan situar unos “TextView” en una cierta posición respecto a los otros elementos del *layout*, además de utilizarse para que puedan ser identificados en los archivos .java para poder ser actualizados cuando se reciba un nuevo valor.

El resto de propiedades de los “TextView” son meramente estéticas, ya sea para definir el tamaño del texto, o para definir su color. Ejemplo de configuración de un “TextView”:

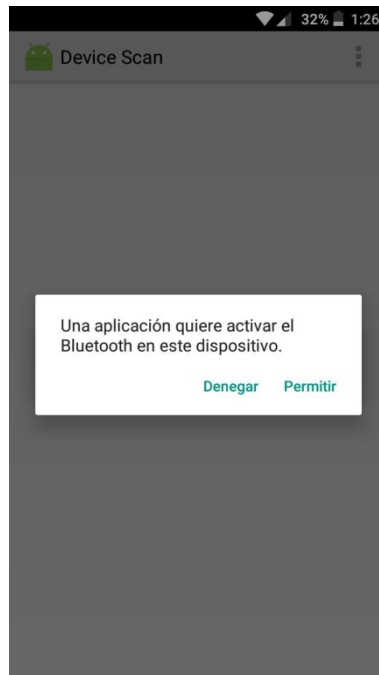
```
<LinearLayout android:orientation="horizontal"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_margin="10dp">
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Pulaciones por minuto:  "
    android:textSize="18sp"/>
  <TextView android:id="@+id/notification_display"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/no_data"
    android:textSize="18sp"/>
</LinearLayout>
```

Mediante “wrap\_content” se indica que el “TextView” debe “envolver” al texto que contenga. Esto hará que las dimensiones dependan del contenido del “TextView”.

En cuanto a “textSize” y “layout\_marginTop”, lo destacable son las unidades que se han utilizado, ya que en vez de utilizar píxeles, se ha utilizado “sp” para el texto y “dp” para los márgenes. Estas unidades hacen que el tamaño no venga fijado por una cantidad determinada de píxeles, sino que vayan en función de la resolución y la densidad de píxeles que tenga cada pantalla.

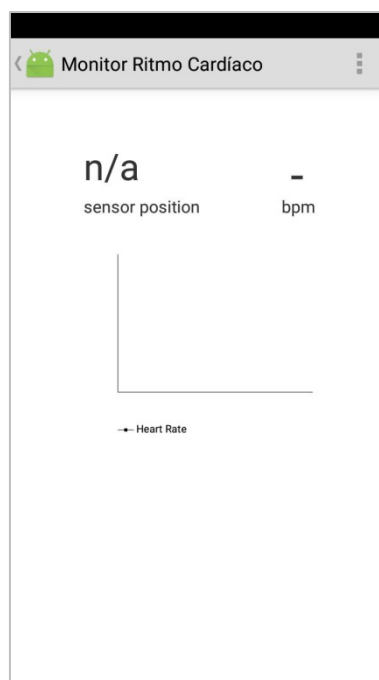
Son por tanto una unidad relativa, que permite que los elementos se adapten correctamente a diferentes pantallas, ya que si se utilizaran directamente los píxeles, una pantalla con mucha resolución tendría una distancia en píxeles distinta de una pantalla que tenga menos resolución y para la que por tanto los mismos píxeles ocuparían un mayor espacio de la pantalla.

Tras definir la interfaz, al abrir la aplicación por defecto se solicitará que se active el Bluetooth si no estaba activado:



**Ilustración 4-6. Aplicación solicitando que se active el Bluetooth**

Tras darle a “PERMITIR”, la interfaz principal mostrará su apariencia por defecto, mostrando un valor nulo de pulsaciones por defecto y posición:



**Ilustración 4-7. Aplicación mostrando su apariencia por defecto**

Si se ha conseguido conectar el dispositivo móvil al *Sensor Tag* mediante Bluetooth, la aplicación comenzará a recibir los valores de las pulsaciones, mostrándolos así:

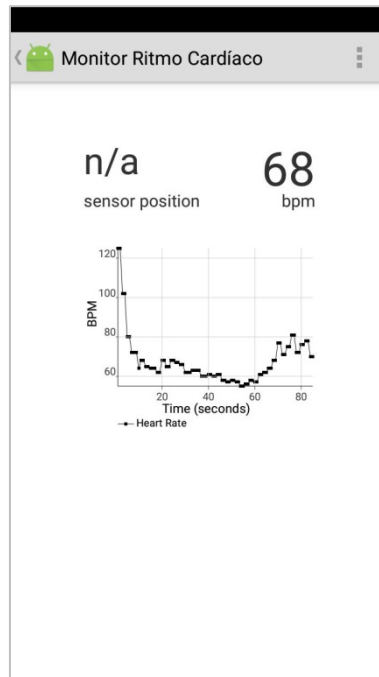


Ilustración 4-8. Aplicación conectada al *Sensor Tag* mostrando las pulsaciones en tiempo real

#### 4.4.6 Programación del archivo *AndroidManifest.xml*

Se trata del último archivo que hay que verificar antes de compilar y ejecutar la aplicación, el cual también se edita mediante etiquetas XML, al igual que el fichero *activity\_main.xml*, descrito anteriormente.

Lo primero que hay que hacer es añadir los permisos que va a necesitar la aplicación para poder llevar a cabo los objetivos que se han propuesto. Los permisos son los siguientes:

```
<!-- Habilitar permisos para conectar, request, aceptar conexiones y
transferir datos Bluetooth -->
<uses-permission android:name="android.permission.BLUETOOTH"/>
<!-- Habilitar permisos para descubrir y manipular configuraciones de
conexiones Bluetooth, requiere los permisos BLUETOOTH -->
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<!-- Declarar aplicación sólo disponible para BLE -->
<uses-feature android:name="android.hardware.bluetooth_le"
android:required="true"/>
```

El permiso “*android.permission.BLUETOOTH*” se va a utilizar para que la aplicación pueda realizar comunicaciones utilizando el Bluetooth, ya sea iniciar una conexión, aceptar una conexión o transferir datos.

El permiso “android.permission.BLUETOOTH\_ADMIN” se va a utilizar para poder realizar acciones más avanzadas con el Bluetooth, como escanear dispositivos o cambiar los ajustes del Bluetooth.

El resto del archivo está compuesto por etiquetas que expresan la estructura que se ha creado para la aplicación. Se encuentra una “<activity>”, la cual es la actividad principal. Dentro de esta actividad hay un “<intent-filter>” para poder recibir los eventos del “Broadcast Update”. Después se encuentra un “<service>”, el cual hace referencia al servicio que se ha creado para manejar los eventos del Bluetooth. Por último, se encuentra una referencia al “<provider>” que se ha creado, el cual incorpora una serie de permisos de lectura y escritura, así como una etiqueta en la que se define la “authoritie” del content provider, para que pueda ser accedido por otras aplicaciones.



# Capítulo 5

---

## Conclusiones y Trabajos Futuros

---

### *5.1 Análisis de Resultados*

Una vez concluido el desarrollo y montaje del prototipo se procede a evaluar si su funcionamiento es correcto.

Ante un test global de funcionamiento, en el que se comparaba el valor obtenido en el Smartphone con el valor de frecuencia cardiaca que ofrece una banda deportiva para la medición del ritmo cardíaco, el resultado fue satisfactorio.

En un alto porcentaje de ocasiones, se detectaba el pulso cardíaco en el mismo instante, y la señal medida ofrecía valores similares. Pero estos datos son muy dependientes de la correcta configuración del fotodetector.

El componente más sensible a fallos es el sensor. Puesto que es un componente que no ha sido diseñado para este fin, su comportamiento puede ser errático a veces, y hay que ser muy meticuloso con su uso. Debe trabajar sobre zonas donde la piel no es muy gruesa (dedo, lóbulo y muñeca), pero un pequeño movimiento que levante el sensor de la piel dejará entrar luz que se leerá como un pulso cardíaco. Esto obliga a que sólo sea efectivo su uso en reposo.



Por otro lado, la etapa de amplificación tiene un ancho de banda algo estrecho, sobre todo a altas frecuencias, aunque esto no supone un problema ya que para que funcione el conjunto se debe estar en reposo.

El comportamiento de comunicación entre el gadget y el *Smartphone*, es correcto, pues ambos se emparejan correctamente y se transfieren los datos necesarios para su funcionamiento. Sería conveniente realizar con diferentes *smartphones* para comprobar el funcionamiento correcto en general.

## 5.2 Conclusiones

Una vez finalizado el proyecto, se puede evaluar si la consecución de los objetivos ha sido satisfactoria para cada uno de ellos:

- Se han estudiado los diferentes métodos utilizados para la medición del pulso cardíaco más específicos para tecnología portable. Se ha realizado la elección del método, seleccionado el sensor y componentes necesarios para realizar la medida.
- Se ha seleccionado una placa de desarrollo para implementar el protocolo *Bluetooth Low Energy* a través de un sistema SoC.
- Se ha diseñado la electrónica de acondicionamiento necesaria para incorporar el sensor al sistema.
- Se ha estudiado el diseño de aplicaciones con protocolo *BLE* en sistemas SoC.
- Se ha estudiado la arquitectura de Android para implementar una aplicación sobre un dispositivo móvil inteligente.
- Se han realizado pruebas de funcionamiento necesarias para validar la arquitectura propuesta.

Por todo esto, se puede considerar que la consecución de los objetivos planteados inicialmente ha tenido una correcta consecución.

## 5.3 Trabajos Futuros

Durante el desarrollo de este proyecto, han surgido nuevas necesidades e inquietudes sobre el alcance de este, debido al conocimiento que se ha ido adquiriendo. Existen también ciertos estudios, implementaciones y mejoras que superan el alcance inicial propuesto en este proyecto.

Es por eso, que aunque se han satisfecho los objetivos consecuentemente, se lista aquí algunas ideas de trabajos futuros tomando como base este proyecto:

- Cambio del sensor de detección de pulso cardíaco. En la actualidad existen sensores integrados en un chip, con comunicación digital directa con el controlador, que permiten una configuración más avanzada, detección de frecuencia cardíaca, saturación de oxígeno, menos interferencias, entre otros. Ejemplo de esto es el sensor MAX30102 de Maxim Integrated.
- Implementar nuevos sensores y algoritmos al *wearable*, pues gran parte del trabajo sería reutilizable para diferentes sensores.
- Estudiar el uso de energía y como minimizar el consumo.
- Diseñar y construir un diseño sobre PCB adaptado a las partes del cuerpo donde se vestiría el wearable, como la muñeca.
- Mejorar el algoritmo de cálculo de la frecuencia cardíaca por uno que ofrezca mayor precisión.
- Otorgar más funcionalidades a la aplicación de Android (históricos, alarmas, visualización), e interactuar con servicios en la nube.
- Desarrollar la interfaz de usuario para otros sistemas operativos, tales como iOS y Windows.



---

---

# Bibliografía

---

---

- Getting started with Bluetooth Low Energy. Kevin Townsend, Carles Cufí, Akiba, and Robert Davidson. Ed. O'Reilly Media
- Philips, Stewart, Hardy, Marsicano. Programación con Android - Edición 2016. Ed. Anaya
- Curso de Programación Android. Salvador Gómez Oliver. [www.sgoliver.net](http://www.sgoliver.net)
- Instrumentación Electrónica. Miguel A. Pérez García. Ed. Thomson
- Circuitos Electrónicos. Análisis Simulación y Diseño. Norbert R. Malik. Ed. Prentice Hall.
- TI CC254x Bluetooth Low Energy Software Developer's Guide
- Specification of the Bluetooth System, Version 4