UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA BIOMÉDICA

TRABAJO FIN DE MÁSTER

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA ELECTRÓNICO PARA LA OBTENCIÓN DE VARIABLES FISIOLÓGICAS EN TIEMPO REAL BASADO EN MICROCONTROLADORES ARM

JOSÉ CARLOS MONTESINOS PÉREZ

2017

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

Reunido el tribunal examinador en el día de la fecha, constituido por

Presidente:

Vocal:

Secretario:

Suplente:

para juzgar el Trabajo Fin de Máster titulado:

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA ELECTRÓNICO PARA LA OBTENCIÓN DE VARIABLES FISIOLÓGICAS EN TIEMPO REAL BASADO EN MICROCONTROLADORES ARM

del alumno D. José Carlos Montesinos Pérez
dirigido por Álvaro Gutiérrez Martín
del Departamento de Tecnología Fotónica y Bioingeniería

Acuerdan otorgar la calificación de: _____

Y, para que conste, se extiende firmada por los componentes del tribunal, la presente diligencia

Madrid, de de 2017

El Presidente	El Vocal	El Secretario
Fdo: _____	Fdo: _____	Fdo: _____

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA BIOMÉDICA

TRABAJO FIN DE MÁSTER

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA ELECTRÓNICO PARA LA OBTENCIÓN DE VARIABLES FISIOLÓGICAS EN TIEMPO REAL BASADO EN MICROCONTROLADORES ARM

JOSÉ CARLOS MONTESINOS PÉREZ

2017

Resumen

En este trabajo se va a desarrollar un prototipo basado en microcontroladores ARM para la obtención de parámetros fisiológicos de los usuarios con el fin de realizar una monitorización continua de su salud y actividad. Para ello se trabajará con distintos sensores para la adquisición de datos, como frecuencia cardíaca, temperatura, etc, y plataformas de desarrollo, en concreto una tarjeta de evaluación STM32L053R8 con un microcontrolador ARM Cortex-M0+, para la implementación del sistema de control. Paralelamente se desarrollará un sistema de comunicación a través de Bluetooth Low Energy (BLE) para la transmisión de estos parámetros a una aplicación Android que se encargará de emparejarse con el sistema y de mostrar los datos en tiempo real. Se pondrá el foco en un buen rendimiento con un bajo consumo, lo que se refleja tanto en el hardware utilizado (STM32L053R8) como en la tecnología utilizada para la transmisión de los parámetros medidos (BLE).

Summary

The objective of this project is to develop a prototype based on ARM microcontrollers to obtain physiological parameters of users in order to continuously monitor their health and activity. For this, several sensors will be used to acquire heart rate, temperature data, etc, as well as developmnet platforms, in this case a STM32L053R8 evaluation board with and ARM Cortex-M0+ microcontroller will be used for the implementation of the control system of this project. In parallel, a communication system through Bluetooth Low Energy (BLE) will be developed for the transmission of these parameters to an Android application that will be in charge of pairing with the control system and of displaying the data in real time. The focus will be on good performance with low power consumption, which is reflected in both the hardware used (STM32L053R8) and the technology used for the transmission of measured parameters (BLE).

Índice general

Índice de figuras

Lista de Acrónimos

TFM: Trabajo Fin de Máster.

BLE: Bluetooth Low Energy.

IoT: Internet of things.

WPAN: Wireless Personal Area Network.

Bluetooth SIG: Bluetooth Special Interest Group.

HCI: Host Controller Interface.

L2CAP: Logical Link Control and Adaptation Protocol.

SM: Security Manager.

GAP: Generic Access Profile.

ATT: Attribute Protocol.

GATT: Generic Attribute Profile.

UUID: Universally Unique Identifier.

HAL: Hardware Abstraction Layer.

CMSIS: Cortex Microcontroller System Interface Standard.

BSP: Board Support Package.

IDE: Integrated Development Environment.

MSP: Multi-channel Serial Port.

PCB: Printed Circuit Board.

Capítulo 1

Introducción

1.1. Presentación del trabajo

Desde hace años la tecnología está ayudando a una mejora del sistema sanitario y de la salud de los individuos. Antes, la mayoría de los diagnósticos se basaban en los conocimientos, la pericia y la intuición de los médicos, ya que no contaban con las herramientas necesarias para poder saber con precisión las causas de los posibles problemas de los pacientes. No obstante, hoy no se concibe la actividad médica sin ayuda de los dispositivos médicos que, por un lado ayudan al diagnóstico de los problemas de los pacientes, y por otro ayudan a mejorar los tratamientos y las intervenciones necesarias para solucionar distintos problemas médicos.[1][2]

De esta manera, se hace uso actualmente de la tecnología para, una vez aparecidos los síntomas, descubrir el problema y atajarlo de forma rápida, eficiente y segura para el paciente. Sin embargo, actualmente estamos dando un paso más allá, que es el de la anticipación a estos problemas con el fin de minimizar síntomas, reduciendo significativamente los problemas de los pacientes, facilitando el trabajo de los profesionales sanitarios y, además, reduciendo en gran medida los costes de un sistema sanitario que, debido al drástico incremento de la esperanza de vida, está viendo aumentado su volumen de costes año tras año. Para ello, está comenzando a primar la monitorización continua de pacientes para la obtención de sus parámetros fisiológicos de forma continua, de manera que podamos analizar toda esta gran cantidad de información para poder determinar cuál es el estado del paciente en cada momento, utilizando técnicas de Machine Learning o Big Data. Con esto, se está consiguiendo poco a poco anticiparse a los problemas con tiempo suficiente para actuar de la mejor manera posible.[3][4][5]

En esta nueva revolución de la tecnología en la salud, además de las nuevas técnicas algorítmicas previamente comentadas que ayudan a la detección precoz de problemas, son necesarios dispositivos que sean capaces de realizar esta obtención de datos de forma continua. Dado que se requiere llevar estos dispositivos durante todo el día existen varias necesidades:

- Que sea cómodo de llevar.

- Que no sea de gran tamaño. Debería ser lo suficientemente discreto para poder usarse en cualquier ocasión.

- Que tenga una duración suficiente para monitorizar la actividad durante uno o varios días.

- Que su coste no sea excesivo, para poder acceder a la mayor parte de la población.

- Que sea lo más sencillo de utilizar, independientemente del rango de edad del usuario.

Gracias a los avances en la electrónica, se ha conseguido obtener una gran variedad de sensores de menor tamaño cada vez, al igual que microcontroladores, sin sacrificar capacidad de conmutación y aumentándola gradualmente de forma realmente significativa. Además, la electrónica actual también pone intensamente el foco en la reducción de consumo, resultando en dispositivos más eficientes y de mayor duración. Estos avances han permitido realizar este tipo de dispositivos que comúnmente se conocen como *wearables*, ya que son dispositivos "llevables", es decir, que se ajustan a los criterios definidos a la lista anterior. Aunque se han probado distintas formas para modelar estos dispositivos, como por ejemplo gafas o forma de collar, la forma más comunmente utilizada es la de reloj o pulsera. No en vano se conoce también a muchos de estos dispositivos como pulseras inteligentes por su diseño.[6]

De esta manera, introducimos el objetivo de este TFM como el de centrarse en el diseño y la implementación de un prototipo de uno de estos dispositivos que sea capaz de obtener parámetros fisiológicos a través de distintos sensores, para almacenarlos y poder realizar una monitorización continua de la actividad física y la salud del usuario. Además de la obtención de datos, el sistema se encargará de enviar los datos inalámbricamente haciendo uso de la tecnología Bluetooth Low Energy (BLE), ya que otro de los requisitos de estos sistemas es el de eliminar en la medida de lo posible la comunicación a través de cables, haciendo más transparente el proceso de monitorización de los usuarios.

Este objetivo se divide en varias tareas. Tendremos que inicialmente analizar y seleccionar los recursos que vamos a utilizar, donde cobra fuerza la tarjeta de evaluación que vamos a utilizar para el desarrollo del proyecto, en este caso una STM32L053R8. Posteriormente, se estudiará en detalle el funcionamiento del Bluetooth Low Energy, para implementar en la tarjeta de evaluación un código capaz de enviar los parámetros medidos a una aplicación Android haciendo uso de esta tecnología inalámbrica. Para hacer las pruebas, se enviarán datos ficticios creados desde la propia tarjeta de evaluación, pero posteriormente se implementará el código para comunicar los sensores, de manera que se obtengan los parámetros fisiológicos reales y éstos sean enviados a través de BLE a la aplicación Android, completando de esta manera el prototipo.

1.2. Justificación y objetivos

1.2.1. Justificación

Ya se han logrado muchas mejoras para la medicina, la salud y la calidad de vida de las personas gracias a la incorporación gradual de la tecnología a este ámbito. Esto ha conseguido que muchos problemas que antes no tenían solución puedan solventarse, gracias a diagnósticos certeros de los problemas y a precisas intervenciones para resolverlos. No obstante, el incremento progresivo de la esperanza de vida está haciendo que, aunque estos problemas se vayan erradicando poco a poco, vayan apareciendo nuevos problemas asociados a la edad que en muchos casos llegan a hacerse crónicos, reduciendo así la calidad de vida de las personas y aumentando los costes significativamente para los sistemas sanitarios.[7][8]

Como se mencionaba en la Sección 1.1, ya existen las herramientas para diagnósticar este tipo de problemas en el momento en el que aparecen los síntomas, así como tratarlas, en caso de este tipo de problemas crónicos como puede ser la hipertensión o la diabetes. Además de los tratamientos existentes, está probado por numerosos estudios que la mayoría de estas situaciones podrían evitarse manteniendo un control continuo de la actividad y la salud del individuo, tratando de mantener sus constantes fisiológicas dentro de los valores que se consideran normales para un individuo sano, tanto antes de padecer este tipo de problemas como durante ellos. Es por esto que los sistemas sanitarios están tendiendo a potenciar una vida sana fomentando la actividad y los hábitos saludables para no solamente vivir más, sino vivir bien durante esos años de más.[9]

Para ello, es muy importante el seguimiento y la monitorización continua de las personas para prevenir situaciones de riesgo y mantener los parámetros fisiológicos dentro de rangos de control, que es el verdadero objetivo. Para esto, y gracias a los avances en la tecnología, y más concretamente a la electrónica, hemos conseguido dispositivos capaces de realizar esta monitorización continua de manera lo suficientemente precisa y siendo estos dispositivos de un tamaño aceptable, cómodos, de coste en general asequible y de bajo consumo para poder llevarse durante varios días. He aquí donde radica la motivación de este TFM, realizar un dispositivo de este tipo, capaz de realizar una monitorización continua de la actividad física y de la salud de las personas que necesiten de este tipo de seguimiento.

Este proyecto continúa el trabajo realizado previamente en el TFM titulado "ANÁLISIS E IMPLEMENTACIÓN DE UN SISTEMA EN LA PLATAFORMA ARDUINO Y ANDROID PARA OBTENER CONSTANTES VITALES ORIENTADO A PERSONAS CON DIABETES TIPO I"[10], realizado por Miriam Pavón Buenache, donde se realizó un primer prototipo de wearable en la plataforma Arduino, con varios sensores implementados y capaz de conectarse a través de Bluetooth con una aplicación Android que también se incluyó en el trabajo. Siguiendo su estela, y una vez comprobado que la prueba de concepto era válida, en este trabajo se ha continuado con la misma idea, implementándola ahora en un microcontrolador de bajo consumo, como es el ARM Cortex-M0+ y utilizando una tecnología de comunicación

inalámbrica de bajo consumo, pasando en este caso a utilizar Bluetooth Low Energy para la comunicación inalámbrica entre el microcontrolador y una nueva aplicación Android desarrollada para este trabajo.

Una vez finalizado este trabajo, el objetivo es que sea continuado para realizar la implementación final del sistema en forma de pulsera o brazalete, diseñando el sistema a la medida que se considere que sea la mejor de acuerdo al microcontrolador, los sensores utilizados y la topología que pueda derivarse de toda su integración en conjunto. Este sistema final tendrá por supuesto integración con la aplicación que ha sido desarrollada durante este trabajo y será mejorada de cara al sistema final, con nuevos parámetros y funcionalidades que sean consideradas como requerimientos.

1.2.2. Objetivos

El objetivo principal es el de realizar un prototipo de sistema, basado en el microcontrolador ARM Cortex-M0+, capaz de obtener distintos parámetros fisiológicos de varios sensores de forma continua y enviarlos, en tiempo real, a través de Bluetooth Low Energy a una aplicación Android que muestra estos datos recibidos. Por tanto puede dividirse este objetivo principal en varios subobjetivos con objeto de concretar las distintas tareas en las que se divide este trabajo para su consecución:

1. **Envío de datos a través de BLE:** El objetivo en este caso es el de implementar un sistema, utilizando la tarjeta de evaluación STM32L053R8, capaz de enviar datos a través de Bluetooth Low Energy de forma continua. Para ello se seguirán los siguientes pasos:

   - Familiarización con el hardware y el entorno de desarrollo de la tarjeta de evaluación.
   - Estudio sobre Bluetooth Low Energy. Funcionamiento general, capas de protocolos, etc.
   - Aprendizaje sobre el uso del módulo de extensión X-NUCLEO-IDB05A1 para la transmisión de datos a través de BLE.
   - Implementación del sistema.

2. **Creación de aplicación Android para recepción y muestra de datos:** Una vez se haya creado el sistema para el envío de datos a través de la tarjeta de evaluación, se deberá crear el sistema que sea capaz de detectarla, recibir sus datos de forma continua a través de BLE y mostrarlos en una interfaz de usuario. Para ello se va a crear una aplicación Android que sea capaz de reconocer dispositivos Bluetooth, emparejarse con ellos y recibir datos de manera continua a través de BLE. Una vez completada, se probará que ambos dispositivos se comunican y reciben datos correctamente. Para ello, se enviarán datos ficticios que van cambiando de manera continua desde la tarjeta de evaluación y observaremos si se produce la recepción de éstos en nuestro dispositivo móvil y se muestran en la interfaz de usuario.

3. **Implementación de sensores:** El objetivo final, una vez se ha comprobado que las comunicaciones entre sistemas funcionan, es el de obtener datos reales de los sensores que queremos que tenga el sistema. Se añadirán los siguientes:

   - Heart Rate 3 Click: Con él se puede obtener la frecuencia cardíaca y la temperatura.
   - LMT70: Sensor para medir de forma continua la temperatura corporal.

   Se implementarán en la tarjeta de evaluación que será la encargada posteriormente de enviarlos, a través de BLE, a nuestra aplicación Android para, de esta manera, completar el prototipo propuesto.

1.3. Estructura del TFM

El trabajo queda organizado en diferentes secciones que se mencionan a continuación junto con un breve resumen de su contenido:

- **Capítulo 1: Introducción.** En este Capítulo se describen brevemente el contexto de este TFM así como los pasos a seguir para su implementación. A continuación, se explica el porqué de la elección de esta temática para este trabajo, definiendo posteriormente los objetivos para su consecución.
- **Capítulo 2: Estado del arte.** En este Capítulo se expone el estado de tecnología actual de los *wearables*. Qué son y cómo aparecieron, cómo se usan actualmente y hacia dónde tiende el uso este tipo de dispositivos.
- **Capítulo 3: Desarrollo: Recursos utilizados e implementación.** Se explica detalladamente cómo se ha realizado la implementación del prototipo, explicando los distintos elementos y tecnologías que lo componen, y dividiéndose por las distintas secciones de su desarrollo.
- **Capítulo 4: Conclusiones y líneas futuras.** Una vez descritos el entorno del trabajo, sus objetivos y todo el proceso de desarrollo, se analizarán las distintas conclusiones que se desprenden de todo el trabajo realizado. Posteriormente, se mencionan futuras líneas de trabajo para la evolución y mejora de este proyecto.
- **Apéndice I: Manual de instalación y del desarrollador.** En esta guía se detallan los pasos a seguir de los futuros desarrolladores implicados en la mejora de este proyecto para poder comenzar a trabajar en él. Incluye explicaciones para la instalación de los entornos de desarrollo necesarios y explicaciones sobre cómo cargar el código en ellos. También se incluyen explicaciones detalladas de todo el código y los archivos utilizados para una fácil adaptación al punto actual de desarrollo del proyecto.
- **Apéndice II: Manual de uso del sistema.** En este apéndice se detalla una guía de uso del sistema. El objetivo es poder explicar los distintos elementos

que lo componen, cómo interactúan entre ellos y cómo el usuario hace uso de este sistema.

Capítulo 2

Estado del arte

2.1. Wearables

La llegada de los Smartphones a nuestras vidas supuso una ruptura para muchas personas de la barrera que para ellos suponía la incorporación de la tecnología a sus vidas. Gracias a ello, muchos de los electrodomésticos y dispositivos que han salido al mercado han ido añadiendo componentes tecnológicos para añadir más funcionalidades a un elemento que antes solo tenía una función. Un ejemplo de ello podrían ser las *Smart TV*. Esta revolución se suele denominar la revolución del internet de las cosas o *Internet of Things*. Pero esta revolución no ha llegado tan solo a los hogares, sino que su uso está extendiéndose cada vez más para ofrecer de soporte y ayuda a multitud de trabajos, independientemente del sector, e incluso y como no suele ser de otra manera, en el área militar.

Uno de estos nuevos dispositivos que están cada vez entrando con más fuerza en el mercado son los *wearables*. Cuando hablamos de *wearables* hablamos de ropa o complementos que interactúan continuamente con el usuario con el objetivo de realizar alguna función específica. Muchos dispositivos pueden considerarse *wearables* como por ejemplo los relojes inteligentes, pulseras de actividad o las gafas inteligentes. La palabra *wearable* se trata de un anglicismo cuyo significado podría clasificarse como "llevable" o "vestible" en referencia a lo que podríamos definir como pequeños ordenadores con una funcionalidad claramente definida que el usuario lleva siempre puesto consigo. Por tanto, estos dispositivos van a aportar algo más de lo que la propia prenda tiene como función inherente y va a añadirle mucho más valor. En función del uso que se haga de ellos se pueden dividir en varios grupos:

- Salud: por ejemplo monitores continuos de glucosa o electrocardiógrafos.
- Deporte y bienestar: como las pulseras de actividad o las bandas de monitorización cardíaca.
- Entretenimiento: como las gafas de realidad virtual o incluso los *Smartphones*.
- Industrial: dispositivos para detectar altos niveles de temperatura o niveles elevados de gases potencialmente tóxicos.

- Militar: por ejemplo cascos inteligentes o sistemas de control remoto de vehículos no tripulados.

El primer dispositivo *wearable* se remonta a los años 60 y 70, que fue cuando Edward O. Thorp y Claude Shannon, investigadores del MIT, crearon un dispositivo contenido en un zapato que era capaz de calcular las probabilidades del lugar donde iba a caer la bola de la ruleta del casino[11]. Tenía un índice de acierto de un 44 % y era enviado a través de radio a la persona que estaba haciendo la apuesta. Puede verse en la Figura 2.1.

Figura 2.1: Dispositivo transmisor (izda)[12] y wearable receptor (drcha)[11]

Fue en 1972, cuando comenzó a verse algo más parecido a los wearables actuales. Hamilton Watch Company, una marca del grupo Swatch, lanzó el Pulsar P1, que fue el primer reloj digital en lanzarse al mercado. Tan solo tres años después la misma compañía lanzó el Pulsar Calculator Watch, que fue el primer reloj en incorporar una calculadora[11].

Avanzando un poco más en el tiempo hasta los años 80, se encuentra otro *wearable*, que fue creado por Steve Mann, consistente en una cámara que se coloca en el ojo y tiene también pantalla incorporada[13][14]. A pesar de que, como es lógico, este dispositivo era muy aparatoso, durante todos estos años ha ido trabajando en la tecnología, miniaturizándola y acercándose más a lo que es el concepto de *wearable* en la actualidad. Puede comprobarse de hecho esta evolución en la siguiente imagen:

Cabe destacar, que a pesar de que los orígenes de estos dispositivos se datan en los 60, como hemos comentado anteriormente, la palabra *wearable* fue utilizada por primera vez en un informe de DARPA a mediados de los 90 [11] y no es hasta principios del siglo XXI que hemos comenzado a ver el verdadero potencial de estos dispositivos. De hecho, fue en 2006 cuando salió Nike+, una colaboración entre Apple y Nike para mantener un control total sobre nuestros entrenamientos gracias a unos medidores que se colocaban en las zapatillas para registrar la cantidad de pasos y velocidad del usuario, transmitiendo estos datos a través de Bluetooth a un ipod. Más adelante, en

Figura 2.2: Evolución del wearable creado por Steve Mann a lo largo de varias décadas [14].

2008, llegó Fitbit a mercado, marcando una referencia sobre lo que ahora realmente se conoce como pulseras inteligentes o de monitorización de actividad.

Sin embargo, no es hasta la feria de consumo electrónico del año 2014, CES 2014, donde gran cantidad de empresas, entre las que se incluyen grandes como Adidas, Google, Intel o Sony, presentaron multitud de dispositivos que incorporaban microcontroladores y diversos periféricos para añadir nuevas funcionalidades a cosas que antes no se pensaba que tuvieran alguna función diferente que para la que fueron pensadas[15].

De hecho, existen actualmente multitud de aplicaciones para la tecnología *wearable* con el objetivo de satisfacer multitud de necesidades y mejorar la calidad de vida de las personas. Por ejemplo, una de las revoluciones en el sector de los *wearables* fueron las Google Glass[15]. Este dispositivo consistía en unas lentes inteligentes que podían ser controladas por la propia voz y que, a través de una pequeña pantalla, mostraban la información solicitada. También se podían hacer fotografías y grabar vídeos de alta definición de todo lo que se estuviese viendo para poder reproducirlo en cualquier otro momento. Por supuesto, se podían realizar otro tipo de gestiones como consultar el correo, ver las noticias, traducir carteles en tiempo real que estuviésemos viendo, ver la ruta más corta a tu destino o incluso realizar videoconferencias en tiempo real.

También existen muchos accesorios, prendas de ropa o tejidos inteligentes y complementos que son capaces de monitorizar continuamente a los usuarios que hacen uso de ellos. Un ejemplo de ello es monitorizar variables como la distancia recorrida, tu frecuencia cardíaca durante todo el día, los ciclos de sueño... Además pueden ofrecer otro tipo de funcionalidades como puede ser el interactuar con otros dispositivos externos para realizar pagos de forma inalámbrica, encender el motor de tu vehículo o controlar tu hogar. Este tipo de dispositivos pueden ser utilizados simplemente para monitorizar el estilo de vida de las personas, como puede ser el ejemplo de las pulseras Fitbit[16], o incluso para mantener un estricto control de la salud de un usuario para poder recomendarle rutinas o actuar rápidamente ante problemas que puedan ser potencialmente graves, como por ejemplo las bombas de insulina que están controladas por las medidas de glucosa obtenidas de un sensor colocado en el usuario.

Otra de las posibles aplicaciones de estos dispositivos es la de mejorar la seguridad de los trabajadores, pudiendo informar de los posibles peligros que pueden ocasionar

perjuicio al trabajador, sirviendo como ejemplo informar de posibles bajos niveles de oxígeno y altos niveles de temperatura que puede sufrir un bombero o un GPS con un modo de emergencia para localizar a personas que tengan que trabajar en zonas alejadas de la población y hayan sufrido un percance. En el rango militar, como ya se mencionaba anteriormente, existen también múltiples aplicaciones que puedan alertar de los posibles peligros que estas personas afrontan durante su actividad.

Lo que está claro es que el futuro conlleva llevar la tecnología con nosotros a todas partes, para facilitarnos el día a día. Es prácticamente un hecho que la tecnología wearable ha llegado para quedarse.

2.2. Wearables para la monitorización de la actividad y salud

Este trabajo, como ya se ha introducido en la Sección 1, se centra en la realización de un prototipo que sea capaz de monitorizar diversas variables fisiológicas de forma continua para poder realizar un estudio de su estilo de vida, controlar su salud, realizar recomendaciones en estos ámbitos o bien poder avisarle de posibles situaciones que requieran atención inmediata.

En la Sección 2.1, ya se ha introducido que uno de los grandes usos que se le puede dar a este tipo de dispositivos es el de la monitorización continua de actividad y salud, de manera que ahora en esta Sección se va a extender un poco más la explicación sobre los *wearables* de este sector en concreto y sobre qué funcionalidades aportan.

Hay que decir que el uso de los *wearables* de monitorización del estilo de vida y la salud ha crecido drásticamente debido en gran medida a que la gente se da cuenta de que la salud puede ser cuantificable a bajo coste y de forma cómoda y se pueden observar mejoras comparativamente en la salud si cambiamos o seguimos ciertas rutinas. Realmente se ha podido comprobar que la salud es un tema que preocupa a gran parte de la población. Este tipo de monitorización también ha ayudado mucho a los deportistas de élite y también amateurs para la mejora de su actividad deportiva, debido a que si se tiene una monitorización continua de ciertas variables, puede conseguirse una mayor individualización del ejercicio propuesto, consiguiendo así mejores resultados para cada persona. Todo ello además queda registrado para poder realizar incluso análisis posteriores y realizar mejoras ya sea para la salud del paciente o para obtener mejores rendimientos deportivos.

Se comprueba también que este tipo de dispositivos aportan información muy amplia, ya que pueden medirse gran parte de las variables fisiológicas del ser humano como la frecuencia cardíaca, la tensión arterial, la temperatura corporal, la frecuencia respiratoria... Y realmente está comprobado que analizando todas estas variables en conjunto puede extraerse información válida y relevante para los usuarios. Además, este tipo de dispositivos muchas veces incluyen GPS que permiten una localización geográfica de la persona, lo que puede resultar de gran interés por ejemplo para personas que sufran demencias, que hagan deporte sufriendo algún tipo de cardiopatía problemática o para casos en los que los usuarios realicen deportes extremos o de

riesgo.

Existen multitud de dispositivos para realizar toda clase de funcionalidades como las que se han definido previamente, que pueden tener distintas formas como relojes, pulseras, cintas o ropa, e incluso puede llegar a considerarse a los Smartphones como *wearables* en sí debido a la variedad de sensores que suelen aportar. A continuación, se mencionan algunos dispositivos y se describen sus características y funcionalidades para conocer qué es lo que ofrece actualmente el mercado para este sector:

- **Fitbit Alta HR:** Se trata de una de las muchas pulseras de actividad de las que dispone Fitbit. Fitbit siempre se caracteriza por ofrecer wearables de calidad, con un buen diseño y cómodos de usar, además de ofrecer multitud de datos y análisis de utilidad para sus usuarios a través de su aplicación para Smartphone. Su precio es de 149,95€. Algunas de las características de esta pulsera son las siguientes[17]:

  - Medición de la frecuencia cardíaca.
  - Monitorización de los ciclos del sueño.
  - Monitorización de la actividad realizada. Pasos, calorías quemadas y distancia recorrida.
  - Reconocimiento automático de ejercicio.
  - Duración de la batería de hasta 7 días.

Figura 2.3: Fitbit Alta HR[17].

- **Garmin Forerunner 735XT:** En este caso, aunque en el fondo son wearables para monitorización de la actividad y la salud, Garmin está más centrado en Smartwatches y además siempre ofrecen funcionalidades muy profesionales y especializadas. Este en particular es uno de los muchos que ofrece Garmin, aunque es bastante especializado. Su precio es de 449,99€y ofrece las siguientes características[18]:

  - Medición de la frecuencia cardíaca.

- Proporciona dinámicas avanzadas de ciclismo, natación y carrera como el equilibrio de tiempo de contacto con el suelo, la longitud de zancada, el ratio vertical...

- Ofrece funciones de estimación del VO2 máximo, umbral de lactato en combinación con otro dispositivo Garmin, pronóstico de carrera y control de recuperación.

- Monitor de actividad y ciclos del sueño.

Figura 2.4: Garmin Forerunner 735XT[18].

Como se ha podido comprobar, los dispositivos Garmin aportan funcionalidades muy específicas aparte de las usuales. Además de este tipo de dispositivos, Garmin ofrece otros muchos que pueden estar especializados en sectores tan concretos como la automoción, la aeronáutica o la navegación.

- **Runtastic Heart Rate Combo Monitor:** Este dispositivo de la marca Runtastic se trata de una banda que se coloca alrededor del pecho para monitorizar la actividad del usuario mientras realiza sus carreras. Es el dispositivo más económico de los comentados en esta sección con un precio de 69,99€, aunque también el que menos funcionalidades tiene. Algunas de ellas se mencionan a continuación[19]:

  - Medición de la frecuencia cardíaca

  - Registro de velocidad, ritmo, elevación, distancia y tiempo.

  - Tecnología Bluetooth Smart y transmisión a frecuencia de 5,3kHz para ser compatible con la mayoría de máquinas cardio (como por ejemplo las cintas de correr).

Figura 2.5: Runtastic Heart Rate Combo Monitor[19].

- **Samsung Gear S3 Frontier:** Samsung también tiene un hueco en el mercado de los wearables y este dispositivo es una prueba de que también tienen mucho que aportar a él. Tiene un coste aproximado de 400€. Tras un análisis, pueden observarse algunas de sus características[20]:
  - Tiene altímetro, barómetro y velocímetro incorporado.
  - Dispone de localización a través de GPS o Glonass.
  - Medición de la frecuencia cardíaca.
  - Batería con duración de hasta cuatro días.
  - Permite guardar un registro de múltiples actividades como running, ciclismo...

Figura 2.6: Samsung Gear S3 Frontier[20].

Como se ha podido ver, existen multitud de dispositivos en el mercado. Aunque estos son solo un pocos ejemplos, puede comprobarse que la mayoría tienen o bien forma de pulsera o de reloj. Nuevos dispositivos se encuentra en estado de creación y poco a poco comenzarán a verse algunos con formas diferentes como prendas de ropa, gafas o incluso algunos implantes.

2.3. El futuro de los wearables: Salud 2.0

Una vez analizados en profundidad los dispositivos wearables en la actualidad, se ha podido comprender su gran potencial de cara a mejorar la vida de las personas. Sin embargo, esta revolución tan solo acaba de empezar y se espera que estos dispositivos cada vez nos acompañen en más actividades y tareas cotidianas.

Una de las áreas en las que se está empezando a poner el foco y muchos recursos para su evolución e integración es en el área de la mejora de la salud de las personas. Como ya se comentó en las secciones 1 y 1.2, el incremento gradual de la esperanza de vida ha ocasionado un aumento de numerosas enfermedades y la cronicidad de otras. El aumento de ciertas enfermedades también debe asociarse a un mal estilo de vida y un mal cuidado de su enfermedad. Por tanto, se está haciendo cada vez más necesario para la sostenibilidad del sistema y para una mejora en la calidad de vida durante la madurez, mantener un control continuo de la salud de las personas y actuar cuanto antes mejor frente a los síntomas para que las consecuencias se vean reducidas tanto en términos de calidad y esperanza de vida como en términos económicos para los sistemas de salud. Es por esto, que poco a poco se está comenzando a hablar de un nuevo concepto en nuestras vidas. Hablamos de la salud 2.0.

La salud 2.0 es una actualización del concepto de salud tradicional, de manera que se añaden nuevos métodos, medios, herramientas y formas de comunicación para mejorar la gestión del proceso de control de la salud de las personas. La salud 2.0 es la nueva manera de hacer medicina a través de las nuevas tecnologías y en la que intervienen todos los agentes del sistema sanitario, tanto los profesionales de la salud, como los pacientes o instituciones sanitarias.

Este nuevo concepto de salud tiene una serie de características y modificaciones con respecto al modelo de salud tradicional que hacen de él algo útil, necesario e innovador. Por un lado, esto permite que los profesionales sanitarios puedan prestar sus servicios allí donde se les necesite sin preocuparse de las barreras que puedan suponer el tiempo y la distancia. Los nuevos servicios telemáticos de mejora para la movilidad permiten un mayor grado de acceso a la información, a los profesionales y los pacientes, lo que facilita la comunicación de esta información y el conocimiento, así como la atención a pacientes allí donde se precise. Por tanto la accesibilidad es un factor clave en la salud 2.0 independientemente del lugar y de la persona que lo necesite.

Otra de las nuevas mejoras añadidas al sistema de salud es la personalización. Las características individuales de cada paciente, edad, sexo, estado actual de salud, datos fisiológicos diarios almacenados... permiten determinar cuáles son los servicios que va a requerir una persona y cómo deben aplicarse exactamente para su situación particular. Tanto los tratamientos como las terapias son personalizados, todo ello gracias a sistemas inteligentes que, gracias a una adquisición continua de datos del usuario y al uso de algoritmos de análisis, son capaces de saber con gran precisión el estado actual y futuro de una persona y actuar de manera precisa cuando se estime necesario.

Uno de los factores clave en la salud 2.0, en el que se están poniendo muchos recursos y un gran enfásis a todos los agentes sanitarios, es la interoperabilidad. Es la interoperabilidad la que permite la accesibilidad y personalización del sistema sanitario, dado que permite a distintos sistemas comunicarse entre sí. Para conseguirlo es necesario hacer uso de estándares ampliamente extendidos y aceptados y que tengan del renombre y la protección de los organismos de estandarización internacionales. De esta manera, podemos conseguir sistemas sin fisuras que faciliten la colaboración y compartición de información entre distintos sistemas, independientemente del hospital, comunidad, país o incluso continente. Este es uno de los grandes retos actuales de la sanidad y uno de los más importante para todos los agentes sanitarios.

Finalmente, la monitorización es también de gran importancia para este nuevo concepto. La monitorización es útil en muchos sentidos. Por un lado, la telemonitorización permite una mayor independencia de los pacientes, al no tener que estar acudiendo a la consulta muchas veces o al no tener que quedarse más tiempo en el hospital. Ésta también permite que los profesionales sanitarios puedan recibir información continua de los pacientes, de forma sencilla y económica para el sistema sanitario, permitiendo ver los parámetros importantes rápidamente, lo que hace que puedan gestionar un mayor numero de pacientes en menos tiempo y además, el sistema pueda alertar de forma rápida de cambios que puedan ser problemáticos para ellos y de esta manera tanto los pacientes como los profesionales sanitarios salen ganando. Además, la recogida continua de datos ayuda a la personalización de la terapia o tratamientos de los pacientes, como ya comentábamos anteriormente y extenderemos más adelante. En esencia, el uso de las herramientas que actualmente se están usando y empezando a usar para la monitorización continua facilitan un seguimiento más preciso y personalizado de la salud del paciente así como una valoración sencilla e inmediata.

Es en el área de la monitorización donde los *wearables* cobran gran importancia, a pesar de que también puedan ser usados en otras áreas, como por ejemplo intervenciones quirúrgicas. Antes, la monitorización de pacientes se veía limitada por las rondas que hicieran los profesionales sanitarios por las diversas consultas o habitaciones de los pacientes. Sin embargo, gracias a los nuevos dispositivos wearables es posible realizar una gestión remota y continua de los pacientes, ya sea que los pacientes estén en el mismo hospital o que sean monitorizados desde sus propias casas, pudiendo realizar recomendaciones telemáticamente e incluso modificar sus tratamientos acorde a los cambios detectados en su estado de salud[21][22]. Esto por ejemplo está sucediendo actualmente con el control de las personas que tienen un marcapasos, reduciendo su paso por consulta ya que se toman las mediciones desde casa y se envían a su médico en el hospital, pudiendo gestionar más pacientes en menos tiempo y reduciendo de igual manera el esfuerzo requerido por los pacientes. También, gracias a los *wearables* que están monitorizando de forma continua a los pacientes, en caso de necesidad se envían alertas en tiempo real que informan sobre cualquier deterioro de la condición de algún paciente, reduciendo los tiempos de respuesta ante problemas que pudieran llegar a ser potencialmente mortales, conservando así los

recursos del sistema sanitario a largo plazo. También esto permite pasar por alto algún ligero cambio en el estado de los pacientes antes de darles el alta.

Otra de las grandes mejoras de este tipo de dispositivos es que el conocer las condiciones exactas de un paciente y su evolución a lo largo de un tiempo determinado permite poder personalizar el tratamiento para cada uno. Esto puede hacerse de forma analítica por los profesionales sanitarios, pero cada vez más se están empezando a usar técnicas más novedosas para llegar a tomar decisiones clínicas. Hoy en día se está trabajando con gran intensidad en el desarrollo de software de apoyo a la toma de decisiones clínicas que puede funcionar tanto para personalizar terapias, por ejemplo cuándo administrar un fármaco y en qué cantidad o bien para la toma de decisiones, como cuán profunda debe ser la incisión en una intervención quirúrgica, por dónde se debe acceder o qué tejido es considerado patológico o con probabilidad de serlo y debe ser extirpado[23][24][25]. Para ello, se está trabajando en la creación de nuevos algoritmos utilizando técnicas de Machine Learning o Big Data. Estos algoritmos se nutren de ingentes cantidades de datos, que pueden haber sido obtenidos en el pasado y se combinan también con datos obtenidos en tiempo real, y es aquí donde entran en juego los wearables, ya que, gracias a ellos tenemos un sistema que combina adquisición de datos, análisis y toma de decisiones y muestra de resultados a los usuarios. Este está siendo uno de los grandes avances en la salud, además de en muchas otras áreas, y que en el futuro estará presente en la mayoría de sistemas y muchas decisiones, diagnósticos e intervenciones serán totalmente dependientes de este tipo de sistemas. Además, este tipo de sistemas inteligentes serán capaces de anticipar problemas mucho antes de que pasen, por ejemplo ya hay empresas que dicen ser capaces de predecir un infarto con horas de antelación para dar tiempo a ir al hospital, o por ejemplo puede detectarse un principio de epilepsia para avisar tanto al portador como a los especialistas médicos que puedan prestarle ayuda de forma inmediata. También, la recogida masiva de datos con wearables y analizarlos con algoritmos de Machine learning, va a permitir obtener gran cantidad de nuevos conocimientos sobre enfermedades que aún guardan muchos misterios para nosotros.

Ya se ha entendido lo que se quiere hacer y por qué va a hacerse durante este TFM. A continuación, en el siguiente apartado, va a describirse cómo va a hacerse el prototipo de wearable con todo lujo de detalles, describiéndose cada sección del desarrollo, para que al final el lector logre comprender la manera en que este tipo de dispositivos son diseñados y los pasos a seguir para lograr que cumplan los requisitos deseados para este tipo de sistemas.

Capítulo 3

Desarrollo: Recursos utilizados e implementación

3.1. Introducción

En esta Sección se describen los distintos recursos que hemos utilizado para realizar este sistema así como va a detallarse el proceso de implementación.

Inicialmente, se describe la tarjeta de evaluación a utilizar, la STM32L053R8. Se describirán de forma general sus características y funcionalidades, para posteriormente mencionar los distintos entornos de desarrollo que pueden utilizarse y se utilizarán para el desarrollo de este trabajo.

La segunda sección del desarrollo se centra en la implementación de un sistema de transmisión de datos utilizando la tecnología inalámbrica Bluetooth Low Energy. Primero, se introducen y explican los principios y el funcionamiento de la tecnología BLE con el suficiente detalle para lograr la comprensión de la tecnología necesaria para el desarrollo de esta funcionalidad del sistema. Seguidamente, se pone el foco en el desarrollo del sistema de transmisión. Esto se logra utilizando un módulo de expansión para la tarjeta de evaluación que dotará de la capacidad de transmitir datos vía BLE. Se estudiará el funcionamiento de éste y se describirá el proceso de la implementación de este sistema de transmisión. Finalmente, quedaría el diseño y la implementación del sistema de recepción de datos a través de BLE. En este caso, se trata de una aplicación desarrollada en Android. Esta aplicación se encargará de detectar los distintos dispositivos Bluetooth disponibles y emparejarse con la tarjeta de evaluación. Posteriormente leerá los paquetes recibidos desde la tarjeta para obtener los datos enviados por ella. Inicialmente se probará el sistema enviando datos ficticios, para poder realizar pruebas del conjunto desarrollado.

En la última sección del desarrollo se describen los sensores a utilizar para la adquisición de datos del sistema y, más adelante, se explica cómo ponerlos en funcionamiento y obtener datos de manera continua, para recibir los datos en la tarjeta de evaluación. Una vez se hayan implementado todos los sensores por separado, se juntará e integrará el código realizado para la adquisición de datos a través de los sensores y se unificará con el código desarrollado para el envío de datos a través de

BLE. De esta manera, el sistema estará completo, adquiriendo los datos en tiempo real de los sensores y enviándolos a través de BLE de manera continua a la aplicación Android, donde se mostrarán en la interfaz de usuario.

3.2. Tarjeta de evaluación STM32L053R8

Como ya se ha mencionado varias veces en secciones anteriores, va a utilizarse la tarjeta de evaluación STM32L053R8 para el desarrollo de este proyecto[26]. Esta tarjeta será la encargada de recibir los datos de los sensores y mandarlos a través de BLE. Uno de los requisitos de este trabajo de fin de máster es el diseño de un prototipo utilizando microcontroladores ARM. La elección de esta tarjeta no viene solo justificada por la inclusión de este tipo de microcontrolador, sino además por el modelo que utiliza, un ARM Cortex-M0+, que hace que la tarjeta de evaluación elegida destaque por su bajo consumo que, unido al BLE, hará que los recursos necesarios para mover este conjunto sean muy bajos, lo que es uno de los puntos fuertes del sistema. Además incluye los puertos y funcionalidades necesarias para el proyecto como puertos SPI e I2C, varias entradas y salidas digitales y analógicas, conversor ADC... Además, es posible añadirle un módulo de expansión BLE para poder realizar diseños e implementaciones con esta funcionalidad de forma mucho más sencilla.

En la Figura 3.1 se muestra una imagen de la tarjeta de evaluación.

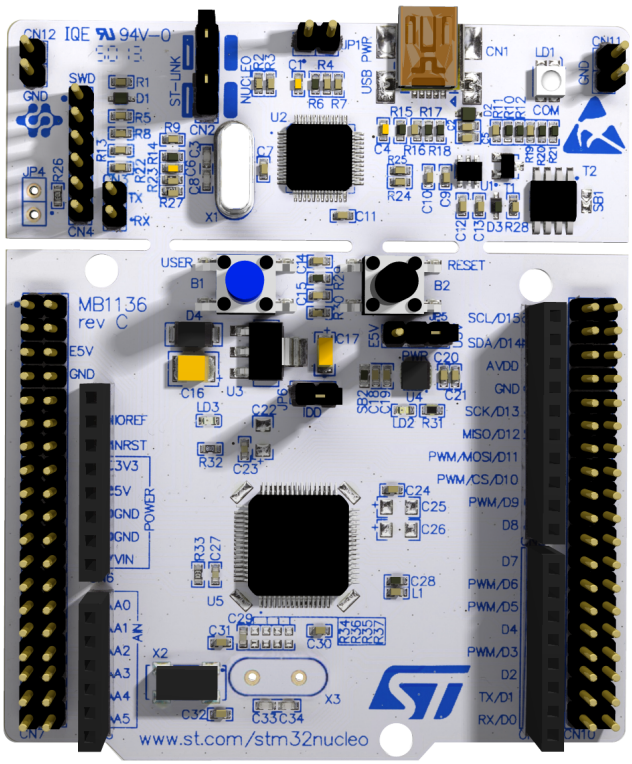

Figura 3.1: Tarjeta de evaluación STM32L053R8.

Pueden apreciarse distintos elementos en la tarjeta, como botones, leds, conectores, etc.

3.2.1. Hardware

La tarjeta de evaluación de ultra-baja potencia STM32L053R8 incorpora un microprocesador ARM de bajo consumo y alto rendimiento y eficiencia para una larga serie de rangos de funcionamiento, además de un gran número de puertos de entrada/salida y periféricos. Este dispositivo es capaz de conseguir una alta eficiencia gracias a un extenso número de posibilidades de elección de relojes internos y externos que pueden elegirse, así como varias etapas internas de cambios de voltaje y varios modos de funcionamiento de bajo consumo disponibles.

Las características generales del hardware que incorpora la tarjeta de evaluación STM32L053R8 son las siguientes:

- Microprocesador ARM Cortex-M0+ RISC de 32 bits, funcionando hasta 32MHz.
- Memoria Flash de hasta 64 kB.
- EEPROM de 2 kB.
- RAM de 8 kB.
- Puerto USB 2.0.
- ADC de 12 bits con sobremuestreo hardware de hasta 16 canales.
- DAC de 12 bits de un solo canal.
- Tres timers de 16 bits y uno de bajo consumo.
- Dos puertos I2C y uno I2S.
- Dos puertos SPI (16 Mbits/s).
- Dos puertos USART y un puerto USART de bajo consumo (LPUART).
- Controlador DMA de 7 canales que soporta el ADC, SPI, I2C, USART, DAC y los timers.
- Y más puertos, periféricos y funcionalidades de menor importancia.

La tarjeta de evaluación opera en el rango de 1,8V a 3,6V de potencia, hasta poder llegar a los 1,65V al bajar la potencia de consumo. Su rango de temperaturas en las que es posible operar va desde los -40ºC hasta los 125 ºC. Como se mencionaba antes, hay varios modos de ahorro de energía que permiten el desarrollo de aplicaciones de bajo consumo. De hecho, además de los rangos de tensión de operación mencionados en el anterior párrafo, es interesante observar las características que aporta la tarjeta en cuanto a bajo consumo en base a sus distintos modos de funcionamiento que mencionamos a continuación:

- Consumo de 0.27µA en modo Standby.
- Consumo de 0.4µA en modo Stop.
- Consumo de 88µA/MHz en modo Run.
- Tarda 3.5µs en despertarse de RAM.
- Tarda 5µs en despertarse de memoria Flash.

Un resumen de todas las características mencionadas anteriormente podemos verlo en la Figura 3.2:

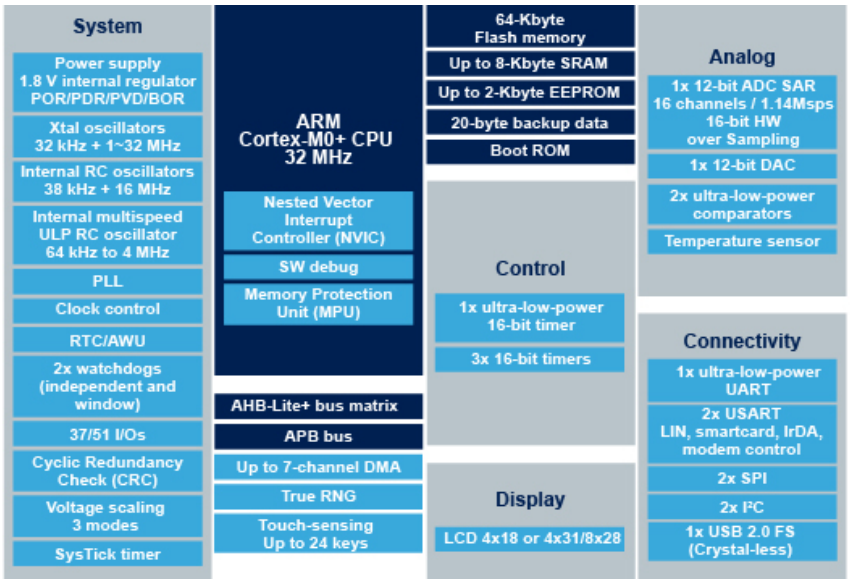

Figura 3.2: Resumen de las características de la tarjeta de evaluación[27]

Anteriormente, hemos visto en la Figura 3.1 las distintas conexiones de las que dispone la tarjeta de evaluación STM32l053R8. No obstante, desconocemos exactamente qué funcionalidad tiene cada puerto. En la Figura 3.3 y la Figura 3.4 puede observarse que tiene dos tipos de pines: los tipo Arduino y los tipo Morpho. Además, en estas figuras se detalla la funcionalidad que tiene cada conexión, ya sea tipo Arduino o Morpho.

3.2.2. Entorno de desarrollo

Una vez vistas las posibilidades que ofrece la tarjeta de evaluación a utilizar en el desarrollo del TFM, ahora hay que aprender cómo poder controlar las funcionalidades que se ofrecen. Existen dos posibilidades a nivel general, escribir el código directamente creando y editando los archivos y compilando con un Makefile o bien hacer uso de los entornos de desarrollo.

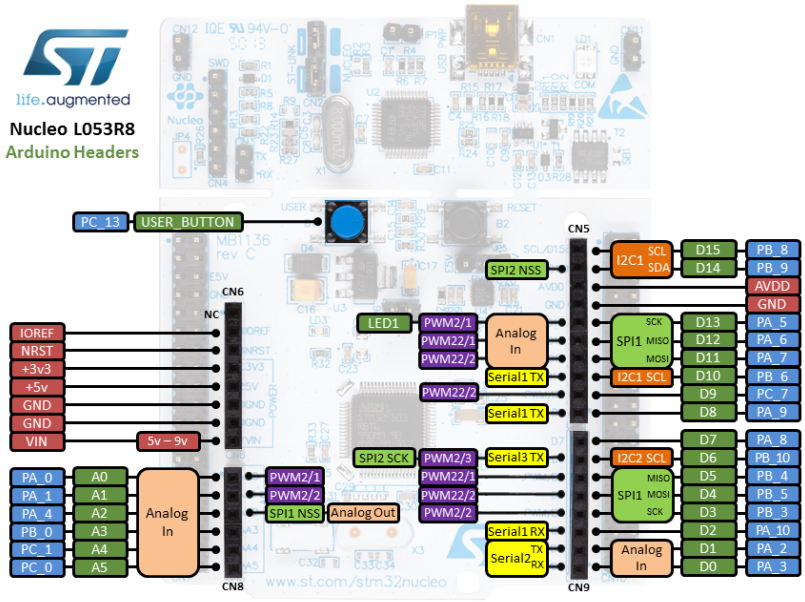

Figura 3.3: Descripción de funcionalidad de las conexiones tipo Arduino[28].

El primer caso puede ser más tedioso, pero también es donde se tiene realmente el control de todo lo que sucede, y por tanto es más sencillo realizar la depuración de posibles errores que puedan surgir, así como evitar errores nuevos a la hora de añadir nuevo código, ya que cada vez que se añade algo se va comprobando que no hay errores. En cualquier caso, existen algunas herramientas que pueden ayudar aun no queriendo hacer uso de los entornos de desarrollo para facilitar la tarea de la implementación de nuevas funcionalidades. La primera forma viene dada por ST Microelectronics, la empresa fabricante de la tarjeta de evaluación que va a utilizarse en este proyecto, ya que aporta numerosos ejemplos que sirven, por un lado para aprender sobre el funcionamiento de la tarjeta y por otro para tomar esos ejemplos como base de programas mucho mayores y además dan un modelo de plantilla preparado para comenzar a desarrollar nuevo código. Otra ayuda, que de nuevo da ST Microelectronics, es el uso de un programa, llamado STM32CubeMX, cuya funcionalidad es la de generar un código que tenga ya implementada la configuración que se quiera a través de una interfaz visual, es decir, haciendo uso de este programa se puede seleccionar por ejemplo que va a configurarse el SPI1 activo o la configuración del reloj de determinada manera y el programa se encargará de generar todo el código en los archivos necesarios con las funcionalidades que hayamos configurado activadas, para que solo haya que centrarse en el desarrollo del programa que se necesite directamente.

La segunda opción es la de hacer uso de entornos de desarrollo para realizar nuestras creaciones. Los entornos de desarrollo van a tratar de facilitar la tarea de creación de nuevo código. Normalmente aportan distintas funcionalidades como

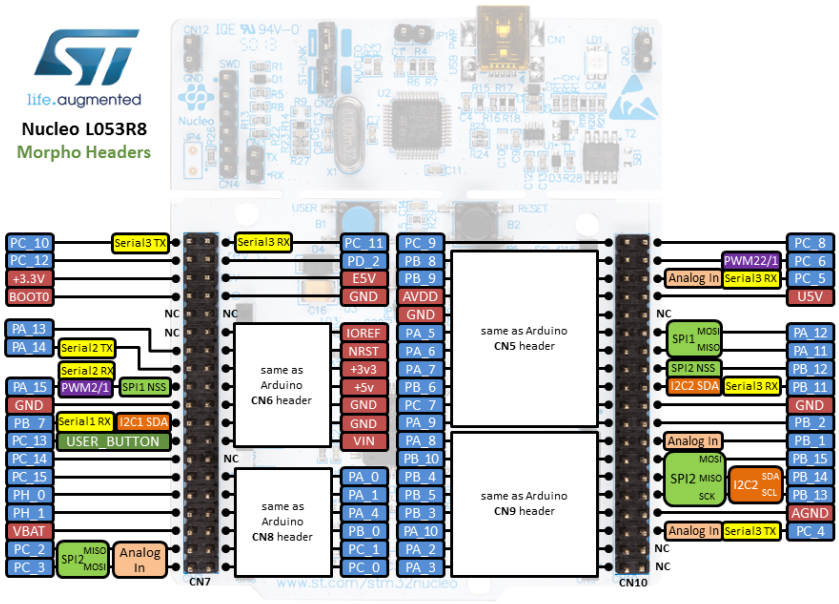

Figura 3.4: Descripción de funcionalidad de las conexiones tipo Morpho[28].

creación de código conforme a una configuración inicial que establezcamos (como STM32CubeMX), disponibilidad de distintos ejemplos para tener de donde partir a la hora de implementar ciertas funcionalidades hardware, como por ejemplo leer del puerto I2C, o depurar código, permitiendo localizar los errores de manera sencilla e incluso aportando posibles soluciones o alternativas. Desde ST Microelectronics recomiendan directamente los siguientes entornos:

- ARM Keil: MDK-ARM: herramienta para creación, desarrollo y prueba de código compatible con tarjetas STM32. Es gratis para programas de tamaño menor a 32 kB de código. Programas de mayor tamaño no serán compilables con la versión gratuita.

- IAR EWARM: herramienta de pago con características similares a Keil. Más enfocado a la optimización.

- IDEs basados en GCC como SW4STM32, Atollic TrueSTUDIO entre otros.

- ARM mbed online: herramientra online que permite desarrollar y compilar código de manera rápida y sencilla, sin necesidad de instalaciones.

Este trabajo se comenzó utilizando los propios ejemplos que aporta ST Microelectronics y creando desde ellos el código necesario para cada funcionalidad. No obstante, debido a problemas con la carga de distintos drivers al compilar con el archivo Makefile se decidió, por no hacer de este problema un retraso en el trabajo, utilizar el entorno

de desarrollo ARM Keil para la creación, desarrollo y prueba de código para la tarjeta STM32L053R8.

En el Apéndice I podremos encontrar una guía más detallada del uso de este entorno de desarrollo y las diversas funcionalidades que se han utilizado para el desarrollo de este trabajo.

3.3. Bluetooth

Una de las partes más importantes de este trabajo fin de máster es la de implementar la transmisión de datos desde nuestra tarjeta de evaluación a una aplicación Android a través de Bluetooth Low Energy. En esta Sección se va a detallar cómo se ha implementado este tipo de comunicación inalámbrica, tanto en la parte de transmisión como en la de recepción. Pero antes debe conocerse más en detalle como funciona esta tecnología para poder implementar una solución funcional y eficiente.

Bluetooth se trata de un protocolo de comunicaciones inalámbrico para establecer conexiones simples, para dispositivos de bajo consumo y seguras. Este estándar permite crear redes inalámbricas de área personal (WPAN) para, en esencia, vincular dispositivos que se encuentren en un área cercana. La transmisión se realiza mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz. Este estándar se creó para facilitar las comunicaciones entre dispositivos a corta distancia y eliminar la necesidad de cables y conectores. También este tipo de tecnología es muy utilizada para facilitar la sincronización de datos, que puede ser cada cierto tiempo o continua, entre equipos personales[29][30].

Algunos ejemplos de aplicaciones en los que se utiliza activamente la tecnología Bluetooth son los siguientes:

- Control remoto de dispositivos: mando de la televisión, mandos de videoconsolas, etc.
- Conexión con dispositivos de manos libres.
- Enlace entre sistemas de audio y sus altavoces.
- Conexión con periféricos para el ordenador: ratones, teclado...

Cada vez más y más dispositivos hacen uso de esta tecnología para sus comunicaciones inalámbricas por ser sencilla de implementar y de bajo coste, tanto económico como de consumo a la hora del uso del dispositivo. El objetivo que se persigue para este TFM es el de establecer una conexión Bluetooth para el envío de datos continuos desde el wearable hasta la aplicación Android.

3.3.1. Introducción al Bluetooth Low Energy (BLE)

3.3.1.1. Introducción y orígenes de Bluetooth Low Energy

Bluetooth, desde sus inicios con la version 1.0, ha ido evolucionando a lo largo del tiempo incorporando nuevos arreglos y funcionalidades para mejorar esta tecnología.

Actualmente ya ha comenzado a llegar la versión 5.0, aunque está todavía en fase temprana de expansión[30][31]. Realmente, la versión que se encuentra más en uso es la 4.0 y es en esta versión donde se encuentra una de las funcionalidades de gran utilidad que se utilizarán e implementarán durante el trabajo, el Bluetooth Low Energy.

Bluetooth Low Energy se denominó en un inicio también Bluetooth Smart. De hecho, se podían distinguir dos tipos de dispositivos en función de su nomenclatura Smart:

- **Bluetooth SMART Ready:** indica que el dispositivo es modo dual, es decir, que opera tanto con periféricos con Bluetooth clásico como Bluetooth Low Energy.
- **Bluetooth SMART:** indica que el dispositivo opera tan solo en modo Low Energy y requiere de otro dispositivo SMART Ready o SMART para funcionar.

Esta clasificación la determinó el Bluetooth SIG (Special Interest Group) con la intención de clarificar la compatibilidad entre dispositivos Bluetooth Low Energy. No obstante, desde mayo de 2016, el Bluetooth SIG decidió eliminar los logotipos de SMART y SMART READY, volviendo a utilizar de nuevo el logotipo de Bluetooth al que estábamos acostumbrados.

El Bluetooth tradicional ofrece unos rangos de distancia que oscilan entre 1 y 100 metros aproximadamente, dependiendo de la potencia del transmisor utilizado. Cuanto mayor es la distancia a cubrir mayor es la potencia transmitida y la energía que se consume. Muchas veces no tiene sentido irradiar mucha potencia si va a usarse un dispositivo que se encuentra a centímetros de distancia y, de la misma manera, hay veces que la transmisión de datos no es continua y en consecuencia no es necesaria una conexión persistente. Este tipo de situaciones motivaron la necesidad de establecer un nuevo tipo de especificaciones que satisfagan esta necesidad de reducción de consumo en los casos necesarios.

Los orígenes de BLE los podemos encontrar en Nokia[32][33]. En 2001, investigadores de Nokia encontraron varias situaciones que las tecnologías inalámbricas de aquel entonces no conseguían satisfacer de forma adecuada. A raíz de esto, la compañía comenzó a desarrollar un nuevo tipo de tecnología inalámbrica, adaptada del estándar Bluetooth, que podría aportar los mismos resultados a un menor coste y con menor consumo. Los resultados se publicaron en 2004 bajo el nombre de Bluetooth Low End Extension. Se continuó posteriormente con el desarrollo, de la mano de socios como Logitech o ST Microelectronics, en el proyecto de nombre MIMOSA[32]. Esta tecnología se dió a conocer al público general en 2006 bajo el nombre de Wibree. Después de negociaciones con los miembros del Bluetooth SIG, se llegó a un acuerdo en 2007 para incluir esta tecnología en futuras especificaciones de Bluetooth, y fue en 2010 con Bluetooth 4.0 cuando se completó totalmente la integración[32].

En comparación con el Bluetooth clásico, Bluetooth Low Energy se diseñó para reducir significativamente el consumo durante su operación. Esto permite la comunicación y transmisión de datos con dispositivos que tengan requerimientos de consumo estrictos, como por ejemplo sensores de proximidad, monitores de la

frecuencia cardíaca u otros dispositivos para la monitorización continua de la actividad o la salud. Para que tengamos una referencia de lo que BLE puede conseguir en términos de reducción de consumo, un sensor alimentado con una pila de botón y con soporte BLE podría estar encendido durante meses o quizás llegando incluso al año sin que hubiera una necesidad de cambiar las baterías. Gracias a la tecnología BLE, todo ello puede conseguirse a un bajo coste y con dispositivos de tamaño pequeño. Además, hoy en día numerosos dispositivos ya son compatibles con Bluetooth Low Energy.

3.3.1.2. Arquitectura y funcionamiento de Bluetooth Low Energy

Ahora se va a entrar más en detalle sobre lo que hay detrás de Bluetooth Low Energy. Es de vital importancia la comprensión de esta sección ya que muchas cosas con las que se va a trabajar en el código que se implemente están directamente basadas en los conceptos explicados aquí.

En BLE se definen dos roles que van a caracterizar cómo se establecen y se llevan las comunicaciones. Habrá un dispositivo con el rol de dispositivo central (Master) que será el que escanee en búsqueda de dipositivos con los que emparejarse, buscando por eventos de Advertising, que se explicará más adelante, y habrá otro dispositivo con el rol de periférico (Slave), que es el que envía estos eventos de Advertising. Los dispositivos centrales son capaces de, mientras tienen varias conexiones en la capa de enlace con periféricos, realizar simultáneamente búsquedas de otros dispositivos. Además, un dispositivo puede enviar eventos de Advertising sin esperar ningún tipo de conexión, de esta manera se pueden enviar datos a los dispositivos centrales que se encuentran escaneando sin tener que establecer una conexión directa.

Para administrar la conexión y el intercambio de datos entre dispositivos, se define una pila de protocolos. Esta pila de protocolos se divide entre la parte de aplicación, el núcleo de la pila de protocolos BLE (Host) y la parte física, como podemos ver en la Figura 3.5.

Definimos el sistema típico BLE como un controlador Low Energy y un host. El controlador LE dispone de capa física (PHY Layer) que incluye la radio y la capa de enlace, y del Host Controller Interface (HCI). El host incorpora también HCI y las capas superiores de la pila de protocolos, como son L2CAP, SM, ATT, GATT y GAP. Para manejar el controlador LE el host le envía comandos HCI ya que es la capa HCI la que provee de una interfaz estándar que permite esta comunicación entre dispositivos[34]. Se describen más en detalle a continuación las distintas capas de la pila de protocolos de BLE para una mejor comprensión del sistema.

- **Capa física BLE:** Esta capa es la que asegura la comunicación entre la pila de protocolos y los datos transmitidos a través del aire. BLE opera también a 2,4 GHz en la banda ISM, al igual que el Bluetooth tradicional, y define 40 canales con un espaciado de 2 MHz. La numeración es de 0 a 39 y tienen una anchura de 1 MHz cada uno. Hay 3 canales para Advertising (37, 38, 39) y 37 para la transmisión de datos. La distribución de los canales puede verse en la Figura 3.7.

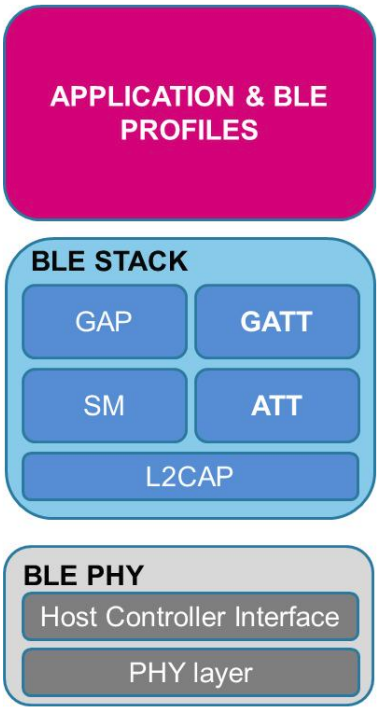

Figura 3.5: Pila de protocolos de BLE[34].

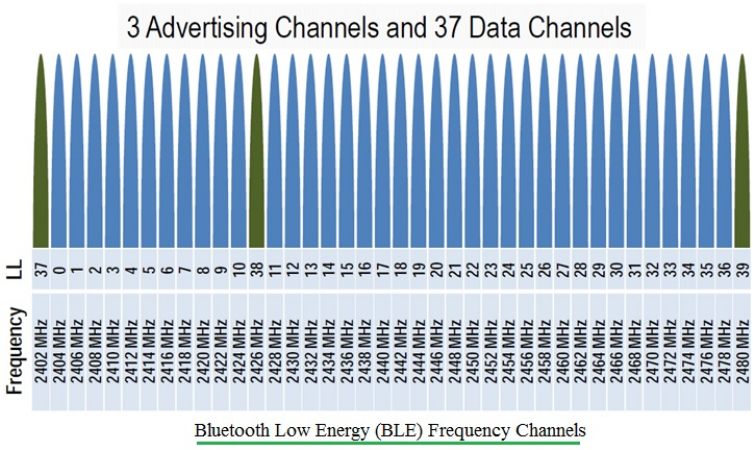

Figura 3.6: Distribución de los canales BLE[35].

Como puede comprobarse, los tres canales de Advertising están situados estratégicamente para evitar interferencias que puedan causar otras tecnologías inalámbricas que conviven en el mismo espectro, como por ejemplo WiFi o Zigbee.

En la capa física se encuentra, como se ha mencionado antes, la capa de enlace.

El funcionamiento de la capa de enlace puede describirse como una máquina de estados, como puede verse en la Figura 3.7.

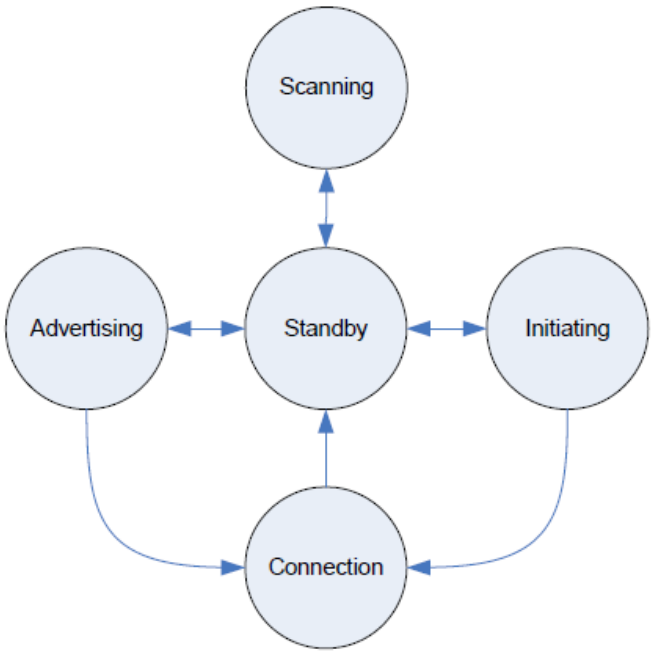

Figura 3.7: Máquina de estados que define la capa de enlace[36].

Por tanto, se definen los siguientes cinco estados de funcionamiento:

- **Standby:** en este estado no se transmiten ni reciben paquetes. Se asocia a un estado de ahorro de energía.
- **Advertising:** se transmiten paquetes de Advertising, y se escucha y responde a respuestas asociadas a la recepción de esos paquetes de Advertising. Este estado es de los que más consumo asociado pueden tener, ya que es directamente dependiente del tiempo que se encuentre mandando paquetes.
- **Scanning:** se está continuamente escuchando, buscando paquetes de Advertising de otros dispositivos.
- **Initiating:** en este estado se buscan paquetes de Advertising de dispositivos específicos y se responde a estos paquetes para iniciar una conexión con otro dispositivo.
- **Connection:** se ha establecido la conexión con un dispositivo. Se definen los roles de dispositivo central (si se entra desde el estado Initiating) y dispositivo periférico (si se entra desde el estado Advertising). En este estado se intercambian periódicamente datos entre dispositivo central y periférico a través de eventos de conexión.

Tan solo se permite tener un estado activo al mismo tiempo y la capa de enlace debe tener, por lo menos, una máquina de estados que tenga o bien un estado de Advertising o bien de Scanning.

Finalmente, como ya se resumió anteriormente, la capa HCI define la conexión física entre el host y el controlador LE a través del uso de comandos HCI.

- **Núcleo de protocolos BLE:** Como hemos visto en la Figura 3.5, el núcleo de protocolos BLE incluye cinco capas de protocolo, que son las que verdaderamente van a regir el comportamiento interno de Bluetooth Low Energy. Se definen a continuación:

  - **L2CAP:** es responsable de la multiplexación de los protocolos superiores y de la segmentación de datos en paquetes de menor tamaño para la capa de enlace y de demultiplexar y reunificar para la otra dirección.
  - **SM:** esta capa es responsable del emparejamiento y de la distribución de claves. Además posibilita una conexión segura y el intercambio de datos con otro dispositivo.
  - **GAP:** esta capa especifica los roles de los dispositivos, modos de funcionamiento (Scanning, Advertising...), procedimientos para el descubrimiento de dispositivos y servicios, administración del establecimiento de la conexión y su seguridad. Además, la capa GAP controla el inicio de las características de seguridad. Esta capa define cuatro roles distintos: Broadcaster, Observer, Peripheral (dispositivo periférico) y Central (dispositivo central).
  - **ATT:** este protocolo permite a un dispositivo mostrar ciertas cantidades de datos, conocidas como atributos, a otro dispositivo. Los atributos pueden ser "descubiertos", leídos y escritos por los dispositivos sincronizados. ATT define la comunicación entre dos dispositivos con los roles de servidor y cliente, respectivamente, sobre un canal L2CAP.
  - **GATT:** este último protocolo describe un "framework" que utiliza ATT para el descubrimiento de servicios y la lectura y escritura de los valores de las características de un dispositivo. Hace de enlace con las aplicaciones a traves de los perfiles de aplicación.

Durante la descripción de los protocolos, se ha hablado de perfiles, servicios características y atributos sin entrar en detalle sobre qué son o para qué se utilizan en Bluetooth Low Energy. Las aplicaciones utilizan la pila de protocolos BLE a través de los perfiles GAP y GATT. El perfil GAP sirve para inicializar la pila y establecer la conexión con otros dispositivos. El perfil GATT especifica una manera de enviar y recibir pequeños trozos de datos, que se conocen como atributos, a través de BLE. El perfil GATT permite la creación de perfiles y servicios dentro de estos perfiles de aplicación. Un resumen de ello puede apreciarse en la Figura 3.8.

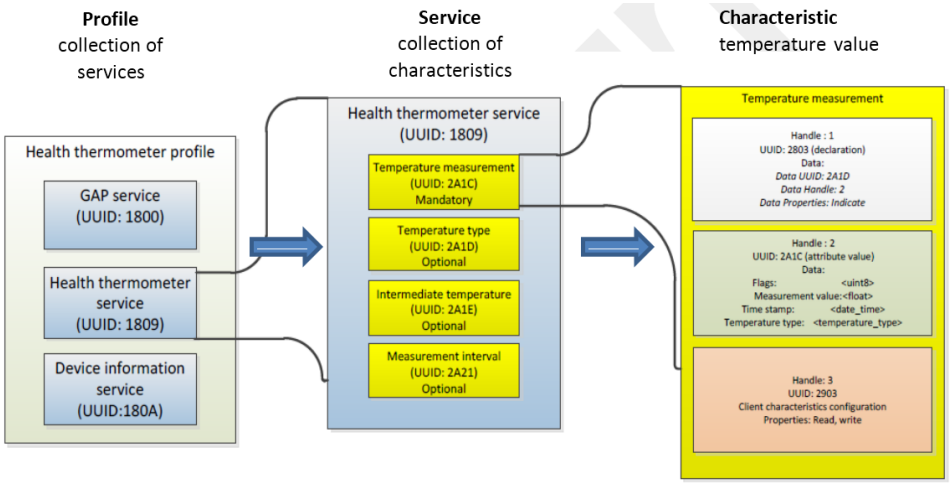

Figura 3.8: Estructura de los perfiles, servicios y características[34].

Se llama perfiles a la agrupación de servicios, como puede verse en la Figura 3.8, en la que se muestran como ejemplo tres servicios. Cada servicio, contiene una serie de características que definen el servicio y el tipo de datos que provee como parte del servicio. Cada una de estas características define el tipo de datos que recoge y su valor. Por tanto, se puede concretar que las características están compuestas por atributos, que definen el valor de estas características, y pueden tener también un descriptor. Los descriptores describen el valor de una característica. Por ejemplo, pueden especificar un rango aceptable para los valores de esa característica o bien la unidad de medida del dato que almacena. Cada característica tiene al menos dos atributos: el atributo principal y el atributo que contiene el dato. El atributo principal es el que define el valor del "atribute handler" y el UUID, que permiten a cualquier cliente saber cómo acceder y poder leer el valor del atributo. Los UUIDs son números de 128 bits que se usan como identificadores. En el caso de BLE sirven para identificar servicios, para acceder a sus características y también identifican características para acceder a sus atributos. Pueden implementarse UUIDs aleatorios, pero el Bluetooth SIG aporta una base para modificar solo sus primeros 32 bits, que es la siguiente: xxxxxxxx-0000-1000-8000-00805F9B34FB. Aunque debido a la longitud del UUID es muy improbable, hay que evitar escoger algunos de los UUIDs ya reservados por el Bluetooth SIG.

Para finalizar, se puede concretar que se llama cliente a un dispositivo que envía comandos y peticiones GATT y acepta respuestas, como por ejemplo un Smartphone. Por otra parte, se denomina un servidor cuando hablamos de un dispositivo que recibe comandos y peticiones GATT y devuelve respuestas, como por ejemplo un sensor de temperatura.

Gracias a toda la descripción anterior, ya se tiene una comprensión general de la composición y la funcionalidad de los distintos elementos que componen la pila de protocolos de BLE, que como se comentó anteriormente, es de vital importancia para

la comprensión del desarrollo del prototipo que se realizará más adelante.

3.3.2. Análisis de la problemática y breve descripción del proceso a seguir

En esta Sección debe analizarse el objetivo que se persigue con este trabajo de fin de máster para planificar los pasos a seguir y lograr su consecución. Como ya se ha mencionado múltiples veces, el objetivo es el diseño de un prototipo de wearable. Este prototipo será capaz de obtener datos de varios sensores y mandarlos a través de BLE a una aplicación Android donde se mostrarán al usuario. Por tanto, habrá dos secciones principales: la implementación de la comunicación por Bluetooth Low Energy y la implementación de la recogida de datos de los sensores.

Comenzando por el análisis de la implementación de la comunicación BLE hay varias cuestiones que resolver. La transmisión de datos se hará desde la tarjeta de evaluación STM32L053R8, aunque la propia tarjeta no dispone de la capacidad de transmitir datos a través de BLE. Para resolver esta situación, necesitaremos ampliar las funcionalidades de la tarjeta de evaluación y esto lo se hará utilizando un módulo de expansión BLE, fabricado también por ST Microelectronics. Hay que comprender cómo hacerlo funcionar para añadirlo al sistema y completar la sección del transmisor de datos BLE. Posteriormente, queda implementar el receptor de datos BLE, que en este caso se trata de una aplicación Android. Esta sección en particular no tiene mayor complicación que comprender cómo usar la API de BLE de Android y, en conjunto con los conocimientos adquiridos previamente sobre BLE, crear la aplicación. La parte final de la sección BLE es la de comprobar que todo el sistema funciona correctamente. Para ello, se enviarán datos ficticios desde el transmisor para ver si se reciben correctamente en la app Android y se muestran al usuario de manera continua.

Queda, por tanto, analizar la segunda sección, la de la implementación de los sensores. En este caso hay que estudiar el funcionamiento de cada sensor para saber de qué manera comunicar con ellos para obtener los datos que se necesitan. Una vez se conoce su funcionamiento, se implementará la conexión con la tarjeta de evaluación a través de las interfaces que utilicen (I2C, SPI...). Cuando se compruebe que la recogida de datos a través de los sensores funciona por separado, se realizará la integración con el sistema completo de manera que se recojan datos de sensores en tiempo real, se envíen a través de BLE y se reciban los datos en la aplicación Android mostrándose de manera continua. De esta manera, se tiene un prototipo capaz de realizar una monitorización continua de la actividad y de la salud del usuario, que es el objetivo de este TFM.

Por tanto, el resumen del proceso que se va a seguir en secciones posteriores se detalla a continuación:

- Estudiar funcionamiento del módulo de expansión Bluetooth Low Energy de ST Microelectronics.
- Implementar código para enviar datos ficticios a través de BLE con la tarjeta de evaluación usando el módulo de expansión BLE.

- Estudio de API Android para el uso de BLE.
- Crear app Android para recepción y muestra de datos recibidos a través de BLE.
- Aprendizaje y estudio del funcionamiento de los sensores.
- Implementación de la recogida de datos a través de los sensores utilizando la tarjeta de evaluación.
- Integración de sistema de recogida de datos a través de los sensores con sistema de transmisión de datos por BLE.
- Prueba y análisis de los resultados del sistema completo: recogida de datos de sensores en tiempo real, envío de esos datos a través de BLE y recepción y muestra de resultados en la aplicación Android.

Se pasará entonces a detallar, en los apartados posteriores, cada una de las secciones del proceso de desarrollo e implementación del prototipo mencionadas anteriormente.

3.3.3. Comunicación Bluetooth: Tarjeta de evaluación

En esta sección va a analizarse cómo hemos realizado la implementación del sistema de transmisión de datos a través de BLE utilizando la tarjeta de evaluación STM32L053R8 y un módulo de expansión Bluetooth Low Energy que permite añadir esta funcionalidad al sistema.

3.3.3.1. Módulos de expansión Bluetooth

Ya se ha hablado de la necesidad de añadir una funcionalidad BLE a la tarjeta de evaluación para hacer un sistema de transmisión de nuestros datos a través de BLE. Lo primero que debe hacerse es analizar las opciones que existen. ST Microelectronics aporta dos módulos diferentes de expansión BLE:

- **Módulo de expansión X-NUCLEO-IDB04A1:** este módulo permite añadir la funcionalidad de enviar y recibir datos a través de BLE a las tarjetas STM32 Nucleo. Este módulo está equipado con conectores Arduino UNO R3. Integra el circuito de sintonización de antena BAL-NRG-013, de muy pequeño tamaño y compatible con Bluetooth 4.0 y el BlueNRG, que es un coprocesador de bajo consumo de red Bluetooth Low Energy. La interfaz de conexión con la tarjeta de evaluación es a través de los puertos SPI y GPIO[37]. Puede verse el módulo en la Figura 3.9.
- **Módulo de expansión X-NUCLEO-IDB05A1:** este módulo añade, de igual manera, la capacidad de transmitir y recibir datos usando BLE a las tarjetas STM32 Nucleo. De nuevo, este módulo incorpora conexiones Arduino UNO R3, pero además añade conectores tipo ST Morpho para ser utilizados por el

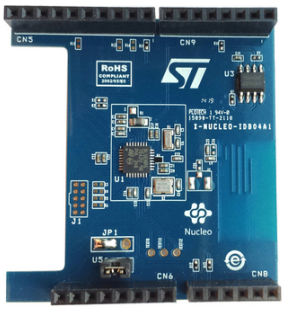

Figura 3.9: Módulo de expansión X-NUCLEO-IDB04A1[37].

usuario en caso de que sean requeridos. Integra el módulo SPBTLE-RF basado en el procesador de red Bluetooth BlueNRG-MS, compatible con Bluetooth 4.1 además del circuito de sintonización de antena BALF-NRG-01D3. La interfaz de conexión con la tarjeta de evaluación también es a través de los puertos GPIO y SPI[38]. Puede verse el módulo en la Figura 3.10.

Puede verse una imagen del módulo a continuación:

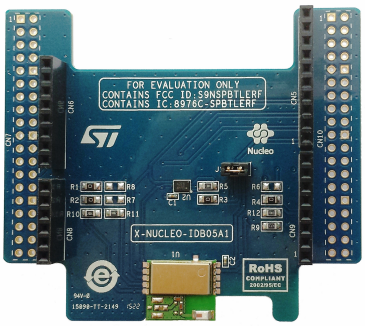

Figura 3.10: Módulo de expansión X-NUCLEO-IDB05A1[38].

Ahora debe hacerse la elección del módulo de expansión que va a utilizarse durante todo el trabajo. Comparativamente son dispositivos muy similares. Sin embargo, son las diferencias las que van a marcar la decisión frente a uno o a otro. En este caso, el módulo X-NUCLEO-IDB05A1 dispone de conectores adicionales ST Morpho en comparación con el módulo X-NUCLEO-IDB04A1 y además es compatible con Bluetooth 4.1, a diferencia del otro que solo es hasta 4.0. Algunas de las ventajas que incorpora Bluetooth 4.1 son[39]:

- Mayor corrección de interferencias. Esta nueva funcionalidad se ocupa de que las señales de Bluetooth y LTE se comuniquen entre sí para coordinar las transmisiones y disminuir la probabilidad de interferencias.
- Reconexiones automáticas en caso de pérdida momentánea de conexión.
- Transferencia más eficaz de datos entre dispositivos.

- Conexión simultánea. Un mismo dispositivo puede usarse como transmisor y receptor de datos al mismo tiempo.

La elección por tanto del módulo va a venir dada por la clara ventaja de Bluetooth 4.1 frente al 4.0. Por esto, el módulo elegido va a ser el X-NUCLEO-IDB05A1 y es el que se utilizará para la implementación del prototipo.

3.3.3.2. Recursos utilizados para el desarrollo

Como se ha comentado antes, uno de los recursos principales y necesarios es el módulo de expansión, ya que añade la capacidad a la tarjeta de evaluación de enviar y recibir datos a través de BLE. Para la unión con la tarjeta de evaluación se utilizan las conexiones Arduino. Con esta unión nos referimos a la unión física del hardware, pero ahora queda hacer que ambos sistemas se comuniquen entre ellos por software para poder hacer uso de sus características.

Para poder utilizar los recursos del módulo de evaluación va a utilizarse un paquete software de expansión diseñado por ST: el X-CUBE-BLE1. Este software se ejecuta sobre tarjetas STM32 y tiene los drivers necesarios para controlar dispositivos Bluetooth Low Energy BlueNRG o BlueNRG-MS, como es el caso de los dos módulos de expansión mencionados anteriormente. En esencia, este paquete aporta el middleware necesario para construir aplicaciones BLE utilizando estos módulos. Además, el paquete incluye numerosos ejemplos para una mejor comprensión y poder empezar a trabajar con estos módulos de manera más rápida, sencilla e intuitiva[40].

Finalmente, un resumen de lo que incluye este paquete software lo se encuentra en la Figura 3.11.

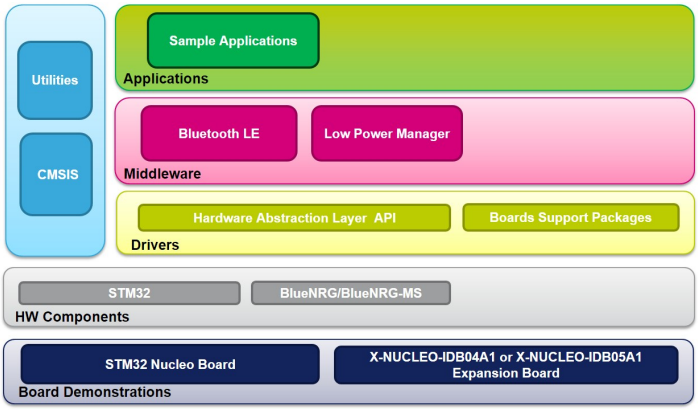

Figura 3.11: Resumen de los contenidos del paquete software X-CUBE-BLE1[40].

Además, ST Microelectronics aporta una aplicación con la que se podrá comprobar la funcionalidad BLE del módulo de expansión. Esta aplicación se llama BlueNRG y permite establecer conexión con el módulo de expansión BLE a través de nuestro Smartphone, mover un cubo una vez establecida la conexión pulsando el *User Button* de la tarjeta de evaluación y mostrar datos de sensores ficticios enviados por el módulo

de expansión a nuestro Smartphone a través de BLE[41]. En la Figura 3.12 podemos ver dos capturas de esta aplicación.

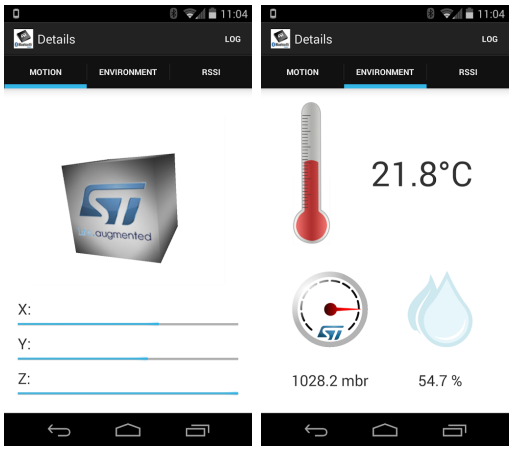

Figura 3.12: Capturas de la aplicación para pruebas BlueNRG[41].

3.3.3.3. Implementación

Ahora que ya se comprende todo el entorno y los recursos que van a utilizarse, va a comenzar la implementación de la parte de transmisión de datos a través de BLE. Para la explicación de la solución, se irán comentando los pasos necesarios para la implementación, presentando algunas secciones de código a nivel ilustrativo. Sin embargo, el código completo se detallará en el Apéndice II, que es el manual de instalación y del desarrollador.

El código que se implemente, será ejecutado en la tarjeta de evaluación STM32L053R8 unida con el módulo de expansión X-NUCLEO-IDB05A1. Hay que tener en cuenta que el módulo de expansión va a necesitar las siguientes conexiones de la tarjeta de evaluación para funcionar y por tanto no se podrá hacer uso de ellas en el código:

- PA0: SPI IRQ
- PA1: SPI CSN (Chip Select)
- PA9: nS (Chip Select EEPROM)
- PB3: CLK (Clock)
- PA8: RST (RESET)
- PA6: SPI MISO
- PA7: SPI MOSI

Hay otras conexiones opcionales que pueden habilitarse, pero en caso de no encontrarse activadas no se hará uso de más conexiones que las anteriormente mencionadas.

La tarjeta de evaluación y el módulo de expansión acoplados a través de las conexiones tipo Arduino quedan de la siguiente manera:

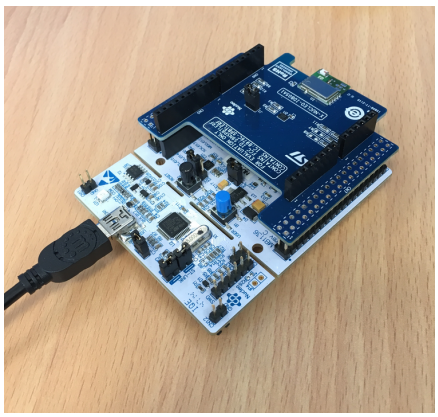

Figura 3.13: Tarjeta de evaluación STM32L053R8 y módulo de expansión X-NUCLEO-IDB05A1 acoplados.

Una vez está el sistema montado y se conocen las limitaciones, comienza la implementación del código del sistema. Para la programación se utilizará C como lenguaje.

Lo primero que debe hacerse es iniciar el sistema para dejarlo preparado para recibir órdenes que ejecutar. Por un lado, se inicializa la tarjeta de evaluación con la librería HAL, que controla las funcionalidades de nuestra tarjeta de evaluación y ajustamos los parámetros del reloj del sistema. También va a inicializarse el módulo de expansión BLE. Todo ello se realiza con las siguientes sentencias de código:

```
/* STM32Cube HAL library initialization*/
HAL_Init();

/* Configure the system clock */
SystemClock_Config();

/* Initialize the BlueNRG SPI driver */
BNRG_SPI_Init();

/* Initialize the BlueNRG HCI */
HCI_Init();

/* Reset BlueNRG hardware */
BlueNRG_RST();
```

A continuación, se inicializa el protocolo GATT y se asigna a la tarjeta de evaluación el rol de dispositivo periférico utilizando el protocolo GAP, que es el encargado de asignar los roles de los dispositivos en la comunicación. En este caso, la aplicación Android hará de dispositivo central. Para esto se utiliza el siguiente código:

```
/* Initialize GATT and GAP Profile to asign peripheral role to STM32
    Board */
aci_gatt_init();
aci_gap_init_IDB05A1(GAP_PERIPHERAL_ROLE_IDB05A1, 0, 0x07, &
    service_handle, &dev_name_char_handle, &appearance_char_handle);
```

Posteriormente, con los perfiles GATT y GAP funcionando, van a añadirse los servicios con sus correspondientes características al perfil GATT. En este caso, va a añadirse un servicio, que es el que va a agrupar los datos de los sensores que recojamos, que se denomina *Environmental Sensor Service*. A este servicio queda añadirle las características que van a representar y alojar los datos que se recojan de los sensores. Por tanto, va a añadirse una nueva característica de temperatura y otra nueva característica para la frecuencia cardíaca. Además se añaden descriptores a ambas características que van a definir el formato numérico en el que se almacenan, el tipo de unidad a la que hace referencia el valor almacenado (por ejemplo grados celsius) y otros parámetros que puedan ser de utilidad a la hora de leer el valor de esa característica.

Estas órdenes vienen recogidas en el siguiente fragmento de código:

```
/* Add Environmental Sensor Service */
aci_gatt_add_serv(UUID_TYPE_128,  uuid, PRIMARY_SERVICE, 10, &
    envSensServHandle);

/* Add Temperature Characteristic and descriptor*/
aci_gatt_add_char(envSensServHandle, UUID_TYPE_128, uuid, 2,
    CHAR_PROP_READ, ATTR_PERMISSION_NONE,
    GATT_NOTIFY_READ_REQ_AND_WAIT_FOR_APPL_RESP, 16, 0, &tempCharHandle
    );
aci_gatt_add_char_desc(envSensServHandle, tempCharHandle, UUID_TYPE_16
    , (uint8_t *)&uuid16, 7, 7, (void *)&charFormat,
    ATTR_PERMISSION_NONE, ATTR_ACCESS_READ_ONLY, 0, 16, FALSE, &
    descHandle);
```

No se ha mostrado el código para añadir la característica y el descriptor de la frecuencia cardíaca ya que el proceso es análogo al del parámetro de temperatura.

Ahora que ya está configurada la estructura del perfil GATT, hay que configurar el dispositivo para que se muestre como dispositivo disponible para conexión, es decir, se anunciará el dispositivo para que nuestra aplicación Android lo encuentre al escanearlo y pueda enviar una solicitud de conexión. En este caso se ha configurado el "anuncio" para que se muestre de forma pública a cualquier dispositivo y no requiera de clave para establecer la conexión, pero todo ello puede configurarse en caso de que sea requerido. De esta manera para mostrar la tarjeta de evaluación a otros dispositivos que estén escaneando se ejecuta la siguiente sentencia:

```
/* Put STM32 Board on Connectable Mode */
aci_gap_set_discoverable(ADV_IND, 0, 0, PUBLIC_ADDR, NO_WHITE_LIST_USE
    , sizeof(local_name), local_name, 0, NULL, 0, 0);
```

El dispositivo se quedará en este modo hasta que la aplicación Android se sincronice con él. A partir de ahí, pasará a estar en modo conectado y dejará de publicitarse como dispositivo disponible. A partir de este momento, se llamará en un bucle infinito a la función *HCI_Process()*. Esta función a su vez llamará a otra, *HCI_Event_CB()* que se encargará de obtener los paquetes BLE y descodificarlos para comprobar si se está solicitando una conexión con el dispositivo, una desconexión o bien si se están recibiendo solicitudes de datos. En cada caso se actuará en consecuencia, pero cabe detallar que en el caso de una solicitud de lectura, se llamará a una tercera función, *Read_Request_CB()*, que actualizará los valores de las características del servicio con el último valor obtenido para que se envíen los datos solicitados actualizados. Para actualizar el valor de una característica se utiliza el siguiente comando:

```
/* Update temperature Characteristic */
aci_gatt_update_char_value(envSensServHandle, tempCharHandle, 0, 2, (
    uint8_t*)&temp);
```

De nuevo, hacemos referencia a la característica de temperatura, pero este comando es totalmente análogo para la característica de la frecuencia cardíaca. La actualización de los valores de los datos en este punto se hará con valores ficticios de ambos parámetros que irán aumentándose de uno en uno en cada solicitud para comprobar que los datos se modifican y envían en tiempo real.

Dado que el proceso de lectura de paquetes BLE ya se encuentra ejecutándose de forma continua en este punto, puede afirmarse que la transmisión de datos a través de BLE ya se encuentra implementada con el código previamente explicado utilizando la tarjeta de evaluación STM32L053R8 junto con el módulo de expansión X-NUCLEO-IDB05A1. No obstante, hasta que no se desarrolle la aplicación Android no habra manera de saber si los datos se están enviando de forma correcta ya que no se dispone en este trabajo de ninguna herramienta que sea capaz de recibir datos BLE ni mostrarlos por el momento. Va a explicarse a continuación en la Sección 3.3.4 cómo va a desarrollarse la aplicación Android y será en la Sección 3.3.5 donde van a mostrarse los resultados del sistema de transmisión y recepción funcionando en conjunto.

3.3.4. Comunicación Bluetooth: Aplicación Android

Como se ha introducido previamente, en esta Sección va a explicarse cómo va a realizarse el receptor de datos BLE, que en este caso se va a tratar de una aplicación Android.

El objetivo es el de hacer un *wearable* para la monitorización de la actividad y la salud, pero además de este objetivo principal, hay otra serie de requisitos entre los que se encuentra que el coste no sea muy alto y sea accesible para la mayoría de la población. Es por esto que los Smartphones se han convertido en una herramienta esencial para la compartición y visualización de los datos adquiridos por los wearables, ya que la inmensa mayoría de la población tienen uno o varios, es una herramienta de

comunicación que siempre se lleva encima y además dispone de los recursos necesarios para actuar como enlace y visualizador de los datos de los wearables gracias a disponer de pantalla, microcontrolador con suficiente potencia, Bluetooth, WiFi...

Además de los beneficios anteriormente mencionados, caben destacar las facilidades que ponen los sistemas operativos de los Smartphones (Android, iOS...) para hacer uso de sus recursos gracias a las APIs desarrolladas para ello. En el caso particular que se ocupa en este trabajo, va a desarrollarse una aplicación para Smartphones que utilicen el sistema operativo Android. La plataforma Android incluye compatibilidad con la pila de red Bluetooth, que permite que un Smartphone intercambie datos de manera inalámbrica con otros dispositivos Bluetooth. El "framework" de la aplicación proporciona acceso a la funcionalidad Bluetooth haciendo uso de la API que existe en Android para ello. En este TFM se hará uso de la API BLE que permite a dispositivos con una versión de Android igual o superior a la 4.3 (API level 18) actuar de dispositivos centrales. Algunas de las funcionalidades que aporta esta API son:

- Descubrir dispositivos y establecer conexión con ellos.
- Solicitar lectura de servicios y características.
- Transmisión de información entre dispositivos.

Existen guías muy detalladas explicando el funcionamiento de la pila de protocolos BLE, así como explicando cómo hacer uso de la API BLE para crear aplicaciones que puedan transmitir y recibir datos haciendo uso de esta tecnología inalámbrica. Además, existen ejemplos que pueden utilizarse en cualquier entorno de desarrollo Android, para tener una mejor comprensión del funcionamiento real de la API BLE en Android y para tener algo como base sobre lo que crear aplicaciones de mayor complejidad.

3.3.4.1. Descripción del entorno

De igual manera que para el caso del desarrollo de la aplicación para la transmisión de datos BLE usando nuestra tarjeta de evaluación, va a utilizarse un entorno de desarrollo para crear la aplicación Android dadas las facilidades a la creación de nuevos proyectos y ventajas y comodidades para el desarrollo, mencionadas previamente en la sección 3.2.2, y que en el caso de creación de aplicaciones Android van a ser mucho mayores.

Existen varios entornos de desarrollo que pueden ser de utilidad para la creación de aplicaciones Android como Eclipse o Netbeans. Sin embargo, para el desarrollo de la aplicación, se va a utilizar Android Studio. Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android y se basa en IntelliJ IDEA. Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio aporta multitud de funciones adicionales para aumentar la productividad de los desarrolladores durante la compilación de las aplicaciones para Android. Algunas de estas funciones de utilidad son:

- Un emulador rápido y con varias funciones.

- Un entorno unificado en el que puedes realizar desarrollos para todos los dispositivos Android.

- Instant Run para aplicar cambios mientras la app se ejecuta sin la necesidad de compilar un nuevo APK.

- Integración de plantillas de código y GitHub para ayudarte a compilar funciones comunes de las apps e importar ejemplos de código.

- Gran cantidad de herramientas y frameworks de prueba.

- Y mucho más...

3.3.4.2. Recursos utilizados para el desarrollo

Para el desarrollo de la aplicación se ha utilizado una serie de recursos para facilitar la tarea de la creación de la aplicación Android para la recepción y muestra de los datos obtenidos por BLE de la tarjeta de evaluación STM32L053R8. Por supuesto se utiliza Android para programar la aplicación y se hace utilizando el entorno de desarrollo de Android Studio, como se ha comentado anteriormente. Concretamente dentro de Android, se utilizará la API BLE para acceder a esta funcionalidad del Smartphone y poder hacer uso de ella para poder implementar el sistema de recepción de datos BLE[42].

Además de todo lo anteriormente mencionado, se ha hecho uso de un ejemplo, creado por el equipo de Google para explicar el uso de BLE en dispositivos Android. El título de este proyecto de ejemplo es *Android BluetoothLeGatt Sample*. Esta aplicación muestra una lista con los dispositivos BLE detectados y disponibles y dota al usuario de una interfaz para establecer conexión y mostrar los servicios y características del perfil GATT de los dispositivos con los que se ha establecido una conexión. A continuación, pueden verse algunas de las capturas de esta aplicación que muestran su interfaz y algunas de sus funcionalidades:

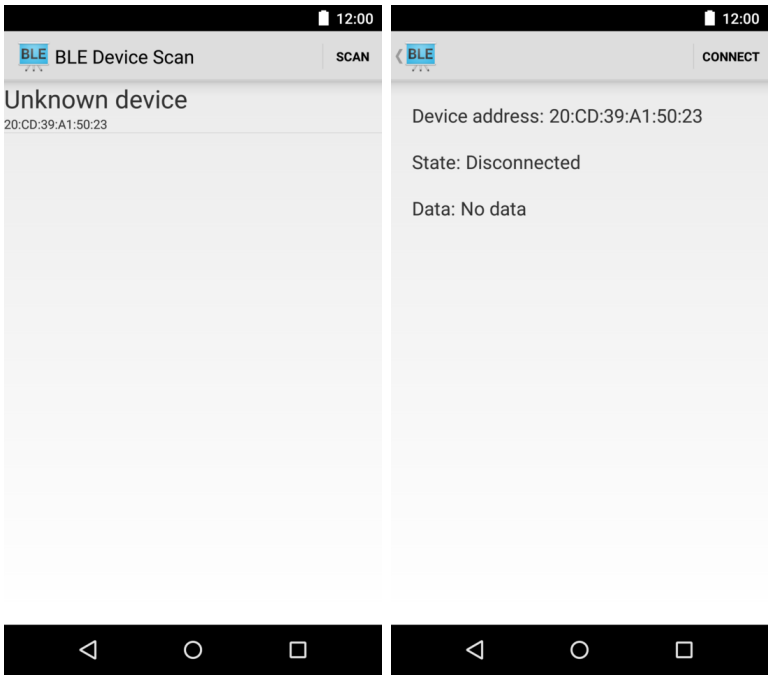

Figura 3.14: Ejemplo Android - Bluetooth LE GATT

Esta aplicación permite comprender más en profundidad el uso de la API BLE de Android para hacer uso de varias de las funcionalidades anteriomente mencionadas. Gracias a esto se podrá crear la app que se necesita con mayor seguridad, rapidez y eficiencia.

Todo ello combinado con las guías explicativas tanto de Android como del uso de la API BLE creadas por el equipo de desarrolladores de Google, dan el marco de conocimientos y recursos necesarios para hacer de la aplicación Android un sistema eficiente, funcional y útil para hacer de receptor de datos a través de BLE y para mostrarlos en su interfaz al usuario del sistema.

3.3.4.3. Implementación

Ya se comprenden las herramientas y el entorno que va a utilizarse para crear el receptor BLE en forma de aplicación Android. De nuevo, se irá comentando cómo se ha ido realizando la implementación por pasos y se irán mostrando las partes de código asociadas a esos avances. Cabe mencionar que el código de la aplicación completo se encuentra detallado en el Apéndice II de este documento, de manera que solo se mostrarán las partes del código relevantes o esenciales para la comprensión de la implementación de la aplicación.

Antes de comenzar a describir los distintos pasos de la implementación de la aplicación, para tener un esquema general del proceso que va a seguirse, conviene resumir la aplicación en los tres elementos principales que la componen: dos actividades y un servicio. Se resumen a continuación:

- **ScanActivity:** Esta actividad es la encargada de escanear en búsqueda de dispositivos BLE disponibles y mostrarlos en un listado. El layout de esta actividad se muestra a continuación:

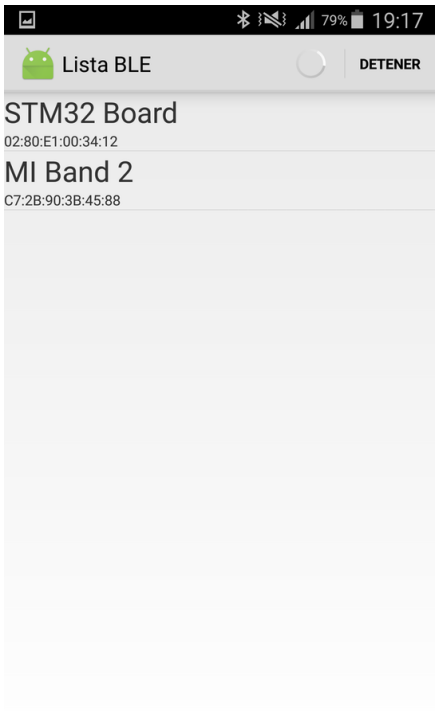

Figura 3.15: Captura ScanActivity

- **DataShowActivity:** Al iniciar esta actividad, se establece la conexión con el dispositivo seleccionado en la actividad *ScanActivity* previamente. Esta actividad es la encargada de solicitar la lectura de datos obtenidos a través de BLE y mostrarlos en la interfaz. Los datos que se muestran son los siguientes:

  - Nombre del dispositivo.
  - Dirección MAC del dispositivo con el que se ha establecido la conexión.
  - Estado: conectado o desconectado.
  - Valor obtenido del sensor de temperatura.
  - Valor obtenido del sensor de frecuencia cardíaca.

  Puede verse la interfaz de esta actividad en la Figura 3.16.

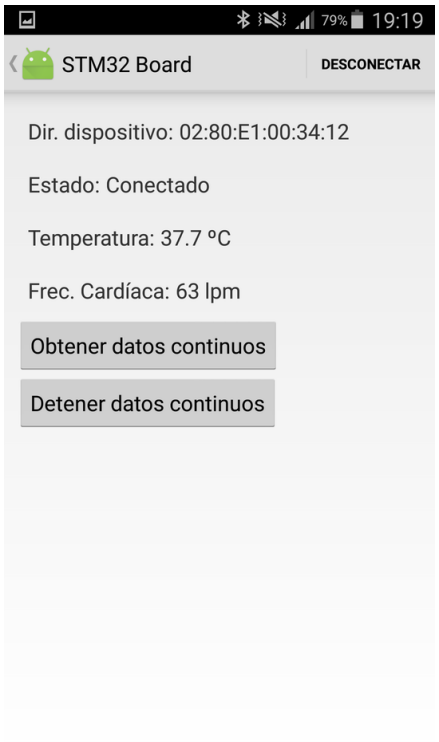

Figura 3.16: Capturas de DataShowActivity

- **BLEService:** Este último elemento de los principales, se trata de un clase que hace las veces y hereda de *Android Service*, que va a recibir y administrar todas las operaciones Bluetooth que tengan que llevarse a cabo. Estas operaciones Bluetooth serán solicitadas desde la actividad *DataShowActivity*. Algunos ejemplos de operaciones que podrán ser solicitadas son la conexión o desconexión con dispositivos o la lectura de los valores alojados en las características obtenidas. De hecho será en esta clase donde se descodifiquen los datos recibidos por BLE de la tarjeta de evaluación.

Con este esquema en mente, comienza entonces un nuevo proyecto utilizando el entorno de desarrollo de Android Studio. Lo primero que debe hacerse es añadir los permisos Bluetooth en el "Manifest" del proyecto para poder hacer uso de los recursos Bluetooth del Smartphone en el que se esté ejecutando la aplicación. Las sentencias a añadir son las siguientes:

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

Ahora se continúa con la actividad *ScanActivity*. Lo primero que debe comprobarse al inicio de la aplicación es si el dispositivo Android es compatible con Bluetooth Low Energy, para en caso de no serlo, cerrar directamente la aplicación ya que no podrá hacer uso de sus funcionalidades. Esto se comprueba con el siguiente código:

```
// It is determined if BLE is supported on the device. If not the app
    ends.
if (!getPackageManager().hasSystemFeature(PackageManager.
    FEATURE_BLUETOOTH_LE)) {
    Toast.makeText(this, R.string.ble_not_supported, Toast.LENGTH_SHORT)
        .show();
    finish();
}
```

Una vez se comprueba que el dispositivo es compatible con Bluetooth Low Energy se obtiene el *Bluetooth Adapter* y posteriormente se compruea si se ha devuelto un elemento nulo, ya que eso significaría que el dispositivo no es compatible con Bluetooth o no funciona correctamente y habría que cerrar la aplicación directamente ya que de nuevo no podría hacer uso de sus funcionalidades. Para ello se utiliza el siguiente código:

```
// Initializes a Bluetooth adapter.
final BluetoothManager bluetoothManager =
                (BluetoothManager) getSystemService(Context.
                    BLUETOOTH_SERVICE);
mBluetoothAdapter = bluetoothManager.getAdapter();

// Checks if Bluetooth is supported on the device. If not the app ends.
if (mBluetoothAdapter == null) {
    Toast.makeText(this, R.string.error_bluetooth_not_supported, Toast.
        LENGTH_SHORT).show();
    finish();
    return;
}
```

Lo último que debe comprobarse al inicio de la aplicación, una vez que ya se sabe que el dispositivo dispone de Bluetooth y es compatible con BLE, es si éste está activado. En caso de no estarlo se muestra un error informando de esta situación y se muestra al usuario un menú para activar o no el Bluetooth. Esto se hace con las siguientes sentencias:

```
// Ensures Bluetooth is enabled on the device.  If Bluetooth is not
    currently enabled,
// fire an intent to display a dialog asking the user to grant
    permission to enable it.
if (!mBluetoothAdapter.isEnabled()) {
    if (!mBluetoothAdapter.isEnabled()) {
        Intent enableBtIntent = new Intent(BluetoothAdapter.
            ACTION_REQUEST_ENABLE);
        startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
    }
}
```

Como es lógico si el usuario decide no activar el Bluetooth la aplicación de nuevo se cerrará. En cambio, si todo ha ido correctamente, la actividad comenzará a escanear en busca de dispositivos disponibles para conexión. Esta solicitud de escaneado se hace llamando a la función *scanLeDevice()* con un booleano de valor *true* como parámetro

en caso de querer iniciar un escaneo o uno de valor *false* en caso de querer detenerlo. Este escaneado está programado para que no realice búsquedas de más de 10 segundos pensando en reducir el consumo de la batería.

A partir de este punto ya se mostrarán en la lista que se encuentra en el layout de la actividad los dispositivos que se hayan encontrado disponibles para conexión. Ahora lo que nos queda por hacer es que al pulsar sobre el dispositivo de la lista al que queramos conectarnos, se realice una llamada a la actividad *DataShowActivity* para que se establezca una conexión con ese dispositivo y obtener sus datos. Para ello es necesario enviar el nombre del dispositivo y su dirección MAC a la nueva actividad que va a ejecutarse. Esto lo se hará con el siguiente bloque de código:

```
final Intent intent = new Intent(this, DataShowActivity.class);
intent.putExtra(DataShowActivity.EXTRAS_DEVICE_NAME, device.getName());
intent.putExtra(DataShowActivity.EXTRAS_DEVICE_ADDRESS, device.
    getAddress());
```

A continuación se inicia la actividad *DataShowActivity*. El código necesario para el funcionamiento de esta actividad se divide entre *DataShowActivity*, que hace las solicitudes de conexión y lectura/escritura de datos a través de BLE y *BLEService*, que es el servicio que interactúa con la API BLE de Android para gestionar esas solicitudes. Se irá comentando específicamente en cada momento.

Como se ha comentado, lo primero que se hace al iniciar esta actividad es conectar con el dispositivo en cuestión, aunque más concretamente se establece conexión con el servidor GATT del dispositivo con el que se quiere conectar. Para ello se ha obtenido la dirección MAC del dispositivo, que había mandado previamente la actividad *ScanActivity*, y que se usará como parámetro para solicitar a *BLEService* que establezca conexión con ese dispositivo. Se hace de la siguiente manera:

```
mBluetoothLeService.connect(mDeviceAddress);
```

Que llama al método *connect()* del servicio *BLEService* que es el que establece directamente la conexión con el dispositivo encontrado cuya dirección coincida con la obtenida y almacenada en *mDeviceAddress*. Primero se comprueba si se trata de un dispositivo con el que estábamos conectados anteriormente para tratar de reconectar. Si no, iniciamos una nueva conexión. Esto se hace con las siguientes sentencias:

```
final BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(
    mDeviceaddress);
mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
```

Dentro de *BLEService* se encuentran implementados los métodos de "callback" de las solicitudes que se hagan a la API BLE de Android a través de un objeto de tipo *BluetoothGattCallback* que se denomina *mGattCallback* que recoge las respuestas de las solicitudes del objeto *mBluetoothGatt*, mencionado anteriormente y con el que se encuentra asociado de acuerdo a la última sentencia del código anterior. Se implementan cuatro métodos de "callback" que se describen brevemente a continuación:

- **onConnectionStateChange:** Este método de "callback" es llamado cuando la conexión con el dispositivo cambia de estado o bien a conectado o bien a desconectado. En el caso de que se pase a estado conectado, se solicita "descubrir" los servicios del dispositivo con el que se ha establecido la conexión con la sentencia *mBluetoothGatt.discoverServices()*. Para el caso de desconexión simplemente se notifica el cambio de estado.

- **onServicesDiscovered:** Este método es llamado como respuesta a la solicitud de descubrimiento de servicios realizada al establecer conexión con un dispositivo. Se notifica esta situación como respuesta.

- **onCharacteristicRead:** Se hará una llamada a este método cuando se solicite la lectura del valor de una característica. Esto ocurre al ejecutar la sentencia *mBluetoothGatt.readCharacteristic(characteristic)*. Se notifica la situación mandando como parámetro adicional la característica leída.

- **onCharacteristicChanged:** Este último método es muy similar al anterior, con la diferencia de que se va a ejecutar cuando el dispositivo notifique a nuestra aplicación que ha cambiado el valor de una característica, mandando como respuesta lo mismo que para el caso de una lectura normal.

En todos los casos, al final de la ejecución, se ha hablado de que se notifica la situación. Esto lo se hace llamando a otro método de *BLEService* llamado *broadcastUpdate()* que se va a encargar de enviar a la actividad *DataShowActivity* esta notificación.

En los dos primeros métodos de "callback" se va a llamar al método ejecutando las siguientes sentencias:

```
/* En respuesta a onConnectionStateChange */
broadcastUpdate(ACTION_GATT_CONNECTED);

/* En respuesta a onServicesDiscovered */
broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED);
```

Que llaman al método *broadcastUpdate()* que tiene la siguiente forma:

```
private void broadcastUpdate(final String action) {
    final Intent intent = new Intent(action);
    sendBroadcast(intent);
}
```

Enviando la notificación de la situación a la actividad *DataShowActivity*.

En los dos últimos métodos de "callback", *onCharacteristicRead()* y *onCharacteristicChanged()* se va a llamar al método ejecutando la misma sentencia para ambos:

```
/* En respuesta a ambos metodos */
broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
```

Como puede verse en este caso se añade un parámetro nuevo para la función *broadcastUpdate()*, que va a ser la característica que hemos leído. De esta manera, el método se ejecuta de manera análoga al anterior pero descodificando el valor de la característica leída entre medias para enviarlo a *DataShowActivity*.

Ya se ha descrito la funcionalidad de *BLEService*, de manera que se vuelve a la descripción de la implementación de *DataShowActivity* para terminar con esta sección de la aplicación Android. Se ha hablado antes de que se envían las notificaciones de las respuestas de los métodos de "callback" a *DataShowActivity*. Para ello hay un objeto en esta actividad del tipo *BroadcastReceiver* y denominado *mGattUpdateReceiver* que se va a encargar de recibir estas notificaciones y actuar en consecuencia. Como es lógico, para manejar las cuatro notificaciones actuará de cuatro maneras diferentes que se describen a continuación:

- **Notificación de conexión con dispositivo:** Se actualiza el estado de conexión a conectado. Se modifica el valor del menú de la barra superior.
- **Notificación de desconexión con dispositivo:** Se actualiza el estado de conexión a desconectado. Se modifica el valor del menú de la barra superior.
- **Notificación de descubrimiento de servicios:** Se llama por primera vez al método *obtainData()* que se encuentra en la misma actividad. Este método se va a encargar de leer los servicios disponibles del dispositivo para localizar el que queremos, que en este caso es el de los sensores. Posteriormente, solicita la lectura de los valores de sus características a *BLEService*, desencadenando las sucesivas llamadas a métodos que desembocan en la obtención final de los datos requeridos.
- **Notificación de datos disponibles:** En este caso, se notifica que existen datos disponibles enviados por *BLEService* que contienen los valores de características leídas. Se actúa ejecutando la siguiente sentencia:

```
        // If there is available data, it is displayed on the UI
    displayData(intent.getStringExtra(BLEService.EXTRA_ID), intent.
        getStringExtra(BLEService.EXTRA_DATA));
```

  De esta manera, puede verse que se llama al método *displayData()*, pasándole como parámetros los valores obtenidos de *BLEService*. Estos parámetros son, por un lado el valor de la característica leída en formato String y por otro lado un parámetro que denominamos como "EXTRA_ID" que contiene un valor que permite discriminar a *displayData()* de qué característica se trata. Finalmente se actualiza la interfaz de usuario con los nuevos valores de las características.

Ahora, solo queda implementar la lectura continua de datos a través de BLE. Para ello va a añadirse esta funcionalidad a los botones que pueden verse en la Figura 3.16. Cuando se pulse sobre "Obtener datos continuos" se hará una llamada a la función *obtainContinuousData()* que se encuentra en esta misma actividad. Esta función está

configurada para llamar cada segundo a la función *obtainData()*, de manera que cada ese lapso de tiempo se actualizarán los valores en el layout de esta actividad mientras se mantenga establecida la conexión con el dispositivo. Finalmente, al pulsar sobre "Detener datos continuos", se detiene la solicitud continua de datos.

3.3.5. Resultados

Ya ha finalizado la implementación del sistema de transmisión y recepción de datos BLE, que incluye la tarjeta de evaluación STM32L053R8 con el módulo de expansión X-NUCLEO-IDB05A1 (transmisión) y la aplicación Android cuya implementación acaba de describirse en la sección anterior (recepción). Por tanto, como ya está el sistema de comunicación completo, puede comenzarse a realizar las pruebas para comprobar que la implementación se ha realizado correctamente.

Lo primero que se realiza es la comprobación de si por un lado la tarjeta de evaluación se "publicita" correctamente y por otro lado la aplicación es capaz de escanear y mostrar los dispositivos disponibles para conexión, y entre ellos la tarjeta de evaluación como es lógico. Los resultados de esta búsqueda pueden verse verlos en la Figura 3.15 mostrada anteriormente, donde vemos que se muestra el listado de los dispositivos según se van encontrando, entre ellos nuestra tarjeta de evaluación con el nombre de *STM32 Board*, y que puede activarse o desactivarse el escaneado con el menú de la parte superior derecha. Se comprueba entonces que esta funcionalidad funciona correctamente.

A continuación, se comprueba si es posible establecer conexión con la tarjeta de evaluación, leer sus servicios y características mostrando sus valores en la interfaz de usuario de manera estática o continua según lo que se elija. Lo primero que se hace es pulsar sobre el nombre de la tarjeta de evaluación que está en la lista, *STM32 Board*, y seguidamente se realizará una llamada a *DataShowActivity* para establecer una conexión con ella y leer sus servicios y los valores de sus características. Como aún no se ha implementado la recogida de datos de sensores reales, los datos que se mandan desde la tarjeta de evaluación son ficticios. En este caso son numeros enteros aleatorios a los que se irán sumando una unidad cada vez que se solicite un dato para comprobar que funciona el envío de datos continuos. Es importante mencionar que en esta versión pone temperatura y humedad porque en la versión de prueba se pusieron nombres que no tenían que ver del todo con la versión final al tratarse de datos aleatorios. Puede verse en la Figura 3.17 la actividad *DataShowActivity* mostrando los datos iniciales recibidos al iniciar por primera vez la actividad (izquierda) y cómo se han modificado los datos al pulsar sobre el botón "Obtener datos continuos" después de cierto tiempo (derecha):

Se ha podido comprobar que los datos se obtienen de manera continua correctamente, sumando una unidad a los datos en cada llamada, también que podemos parar/reestablecer el flujo de datos BLE con los botones de la interfaz, y que podemos desconectarnos del dispositivo y volver a conectarnos de manera satisfactoria con el menú de la parte superior derecha de la interfaz. Puede verse en la Figura 3.18 como quedaría ésta si pulsamos sobre el botón de "Desconectar".

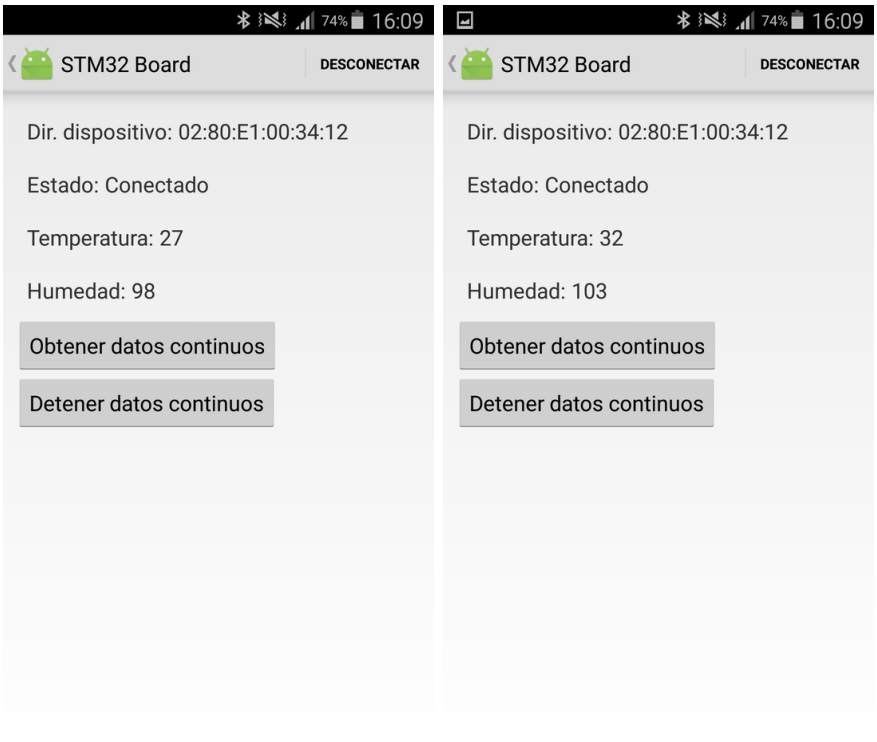

Figura 3.17: DataShowActivity: Resultados obtenidos en un inicio (izda) y resultados al tiempo de habilitar el flujo continuo de datos (drcha)

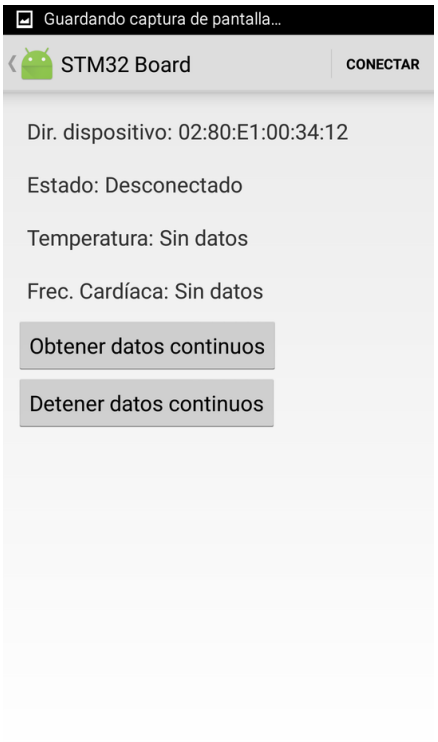

Figura 3.18: Resultado de desconectarnos del dispositivo

En este caso se trata de una captura de la versión final y de ahí las discrepancias con el texto de la interfaz con respecto a las imágenes anteriores.

Dados estos resultados, puede garantizarse que se ha conseguido implementar el sistema de comunicación Bluetooth Low Energy de manera satisfactoria al funcionar de la manera planificada tanto el sistema de transmisión como el de recepción. Se ha conseguido detectar y establecer conexión con el dispositivo requerido, se han podido enviar los datos correctamente a través de BLE y se han recibido y descifrado estos datos en el formato correcto en la aplicación Android, pudiendo activar un flujo continuo de datos y detenerlo cuando se estime oportuno. Se puede entonces pasar a continuación a describir la última sección del desarrollo de este TFM, la implementación de la recogida de datos de sensores reales utilizando la tarjeta de evaluación STM32L053R8.

3.4. Descripción e implementación de los sensores

3.4.1. Introducción

En esta sección final de la parte de desarrollo, va a explicarse todo lo relativo a los sensores con los que se va a trabajar y a cómo van a implementarse para completar el prototipo de *wearable* para este TFM.

Como se ha comentado en varias secciones anteriores, la planificación que va a seguirse para este apartado es la siguiente:

- Introducción y breve descripción de las tareas a realizar y los recursos involucrados, que se hace en esta sección.
- Descripción en detalle de los sensores e implementación en la tarjeta de evaluación individualmente.
- Integración de todo el sistema en conjunto, uniendo el sistema de comunicación BLE, implementado previamente, y el sistema de adquisición de datos de los sensores en tiempo real, que se implementa en esta sección.
- Análisis y muestra de resultados.

Los sensores que van a utilizarse son el Heart Rate Click de Mikroelektronika, que va a usarse para medir el valor de la frecuencia cardíaca, y el módulo de evaluación LMT70EVM para obtener valores de la temperatura del usuario. Se realizará la implementación de ambos sensores por separado, para simplificar el proceso, pero se hará teniendo en cuenta los puertos que están ocupados por el módulo de expansión X-NUCLEO-IDB05A1 y que se han mencionado previamente en la Sección 3.3.3.3. Para ello, habrá que analizar las hojas de características de ambos sensores en detalle para entender cómo establecer comunicación con ellos, para establecer una configuración inicial e ir solicitando datos continuos de forma periódica. Una vez se entienda esto, debe comprenderse cómo recibir estos datos en la tarjeta de evaluación STM32L053R8. Este proceso incluye, ver qué tipo de puertos utilizan los sensores con

los que se va a trabajar (I2C, SPI...), ver qué puertos están libres en la tarjeta de evaluación y aprender cómo utilizar estos puertos, es decir, en esencia cómo inicializar, leer y escribir a través de esos puertos desde la tarjeta de evaluación. Finalmente, habrá que comprobar que los datos se reciben adecuadamente, en el formato correcto y de manera continua y sin errores de transmisión.

Cuando estén ambos sensores funcionando, se unirá el sistema de comunicación BLE con el sistema de recogida de datos de sensores para tener el prototipo de wearable que era el objetivo de este TFM. Se mostrarán los resultados finales en la Sección 3.4.4.

3.4.2. Sensor de frecuencia cardíaca

3.4.2.1. Descripción

Como se ha introducido previamente, el sensor que va a utilizarse para el cálculo de la frecuencia cardíaca del usuario es el Heart Rate Click de Mikroelektronika. Este sensor lleva el pulsioxímetro y sensor de frecuencia cardíaca MAX30100 integrado en el mismo dispositivo con la circuitería necesaria para hacerlo funcionar. El MAX30100 se trata de un sensor óptico que obtiene sus mediciones emitiendo dos haces de luz de dos longitudes de onda utilizando dos tipos de leds, uno rojo y otro infrarrojo, y posteriormente midiendo la absorbancia de la sangre pulsátil a través de un fotodetector. Esta combinación de leds está optimizada para la lectura de datos a través de la punta del dedo. La señal recibida se procesa por una unidad de procesamiento de señal analógica de bajo ruido y se comunica con el microcontrolador objetivo, en este caso la tarjeta de evaluación, a través de su interfaz mikroBUS I2C[43].

Hay que tener en cuenta que el funcionamiento del sensor puede verse afectado por un exceso de movimiento durante la medida y por cambios de temperatura. Además, si se aplica demasiada presión sobre el sensor, puede ocurrir que haya una constricción del flujo sanguíneo en los capilares del dedo, de manera que la fiabilidad de las mediciones podría verse comprometida. Puede verse en la Figura 3.19 el sensor Heart Rate Click.

Como puede verse en la Figura 3.19, el sensor dispone de las siguientes conexiones:

- 3V3: Para alimentar el dispositivo con una entrada de 3.3 voltios.
- GND: La tierra del dipositivo. Dispone de dos conexiones de este tipo.
- SDA: Puerto de datos I2C. Es bidireccional. A través de este puerto se mandan datos al sensor, para cambiar su configuración o solicitar datos, y también se mandan datos a la tarjeta de evaluación, que son los datos de los que se ha solicitado una lectura.
- SCL: Entrada del reloj I2C. Tiene como fin la sincronización de la comunicación entre la tarjeta de evaluación y el sensor.

Figura 3.19: Sensor Heart Rate Click por delante (izda) y por detrás (drcha)[43]

- INT: Pin de interrupción. Sirve para avisar al dispositivo al que esté conectado de que ha ocurrido un evento en el sensor. Las situaciones por las que esta interrupción es activada pueden ser configuradas.

El MAX30100 es completamente configurable a través de registros software y la salida de datos se va almacenando en un buffer FIFO que es capaz de almacenar 16 muestras en el sensor. De esta manera, esta cola FIFO permite al sensor estar conectado a un microcontrolador en un bus compartido, donde no es necesario recibir los datos de manera continua. El sensor es capaz de obtener medidas del la saturación de oxígeno en sangre (SpO2), de la frecuencia cardíaca y de la temperatura ambiente. Esta última funcionalidad puede usarse opcionalmente para calibrar el sensor para la medición de SpO2.

Para poder comenzar con la implementación de este sensor en la tarjeta de evaluación será necesario conocer más en profundidad el sensor MAX30100 para saber de qué manera establecer comunicación con el sensor y cómo realizar solicitudes de datos. Para ello, deben conocerse los registros que componen el sensor y para qué sirven y posteriormente debe entenderse cómo realizar escrituras o solicitar lecturas al sensor, pudiendo comenzar a continuación a explicar la implementación[44].

A continuación se describen los registros que componen el sensor MAX30100:

- **Interrupt Status:** la dirección de este registro es la 0x00. Tiene 5 interrupciones distintas que se ponen a nivel bajo cuando alguna de ellas es activada. Una vez activadas las interrupciones no se desactivan hasta que se lee este registro o bien el registro que activó la interrupción. Este registro puede resumirse con la Figura 3.20.

  Se resume a continuación la utilidad de cada bit:

  - Bit 7: FIFO Almost Full Flag (A_FULL). Se activa cuando se llena el buffer de la FIFO.

Interrupt Status (0x00)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
Interrupt Status	A_FULL	TEMP_ RDY	HR_RDY	SPO2_ RDY				PWR_ RDY	0x00	0X00	R

Figura 3.20: Registro Interrupt Status[44].

- Bit 6: Temperature Ready Flag (TEMP_RDY). Se activa cuando hay un dato de temperatura listo.
- Bit 5: Heart Rate Data Ready (HR_RDY). Se activa cuando se ha terminado una lectura de frecuencia cardíaca (led infrarrojo).
- Bit 4: SpO 2 Data Ready (SPO2_RDY). Se activa cuando se ha terminado una lectura de SpO2 (led rojo + led infrarrojo).
- Bits 3, 2 y 1: Reservados.
- Bit 0: Power Ready Flag (PWR_RDY). Se activa al encenderse el dispositivo.

- **Interrupt Enable:** Cada una de las interrupciones, a excepción de la de *Power-Ready* que siempre está activada, puede ser habilitada o deshabilitada mediante la configuración de este registro. Se resume el registro con la Figura 3.21.

Interrupt Enable (0x01)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
Interrupt Enable	ENB_A_ FULL	ENB_TE P_RDY	ENB_HR_ RDY	ENB_S O2_RDY					0x01	0X00	R/W

Figura 3.21: Registro Interrupt Enable[44].

Para desactivar una interrupción concreta debe escribirse en el bit de su enable correspondiente con un cero. Cuando se desactiva, la interrupción correspondiente aparece con valor 1 en el registro *Interrupt Status*, pero el pin *INT* no se pondrá a nivel bajo, es decir, cualquier sistema externo ignorará que ha ocurrido esta interrupción. La dirección de este registro es la 0x01.

- **FIFO Write Pointer:** Este puntero apunta hacia la dirección donde el MAX30100 va a escribir la próxima muestra a almacenar. Se va autoincrementando según van entrando muestras en la FIFO. La dirección de este registro es la 0x02.

- **FIFO Overflow Counter:** Cuando la FIFO está llena, no se meten mas muestras dentro y se pierden. Este registro va contando el número de muestras que se han perdido. Se satura cuando alcanza el valor 0xF. En el momento en

que comienzan de nuevo a salir muestras de la FIFO este contador se resetea a cero. La dirección de este registro es la 0x03.

- **FIFO Read Pointer:** Este puntero apunta a la dirección de donde el procesador coge la siguiente muestra de la FIFO. Se va autoincrementando según van saliendo muestras de la FIFO. Se puede escribir en este registro para poder releer muestras de nuevo. La dirección de este registro es la 0x04.

- **FIFO Data:** La FIFO circular es capaz de almacenar 16 muestras de datos SpO2, lo que implica datos tanto del led rojo como del infrarrojo. Cada muestra tiene 4 bytes de datos (2 bytes para el dato del led rojo y 2 bytes para el del led infrarrojo), de manera que para obtener un dato completo hay que leer cuatro veces este registro. De esta manera, puede concluirse que se pueden almacenar un total de 64 bytes de datos en la FIFO. Hay que tener en cuenta que en el modo de frecuencia cardíaca los bits 3 y 4 de cada muestra son ceros, ya que no se utiliza el led rojo, pero habrá que leer los 4 bytes de todas maneras para pasar a la siguiente muestra y no hacer lecturas erróneas. Puede verse una imagen que ilustra esta distribución de los datos de cada muestra en la Figura 3.22.

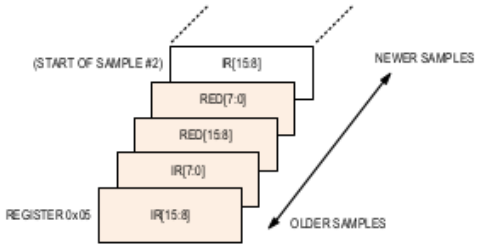

Figura 3.22: Representación gráfica del registro FIFO Data[44].

El puntero de dirección de este registro no se autoincrementa con cada lectura, pero el *FIFO Read Pointer* si lo hace, de manera que el siguiente dato enviado será el siguiente dato disponible en la FIFO. La dirección de este registro es la 0x05.

Puede verse un resumen de todos los registros FIFO en la Figura 3.23.

FIFO (0x02–0x05)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
FIFO Write Pointer					FIFO_WR_PTR[3:0]				0x02	0x00	R/W
Over Flow Counter					OVF_COUNTER[3:0]				0x03	0x00	R/W
FIFO Read Pointer					FIFO_RD_PTR[3:0]				0x04	0x00	R/W
FIFO Data Register	FIFO_DATA[7:0]								0x05	0x00	R/W

Figura 3.23: Resumen de los registros FIFO[44].

- **Mode Configuration:** Registro que sirve para habilitar diversas configuraciones y modos de trabajo del MAX30100. Puede verse un resumen de este registro en la Figura 3.24.

Mode Configuration (0x06)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
Mode Configuration	SHDN	RESET			TEMP_EN	MODE[2:0]			0x06	0x00	R/W

Figura 3.24: Registro Mode Configuration[44].

La utilidad o funcionalidad de cada bit se resume a continuación:

- Bit 7: Shutdown Control (SHDN): Escribiendo uno en este bit, el dispositivo se pone en modo ahorro de energía. En este modo todos los registros mantienen intactos sus valores y las operaciones de lectura/escritura siguen funcionando de manera normal. No obstante, todas las interrupciones se fijan a cero en este modo.
- Bit 6: Reset Control (RESET): Cuando se escribe uno en este bit toda la configuración y los registros se reestablecen al estado en el que estaban al arrancar el dispositivo. Este bit se reestablece a valor cero en cuanto la secuencia de reseteo se completa.
- Bits 4 y 5 reservados.
- Bit 3: Temperature Enable (TEMP_EN): Cuando se establece el valor de este bit a 1, se inicia una lectura de la temperatura medida por el sensor de temperatura que incorpora este dispositivo. Este bit vuelve automáticamente a su valor inicial 0 cuando termina la lectura del dato de temperatura.
- Bits 2 a 0: Mode Control: Estos bits son los que controlan el modo de operación del MAX30100. Cambiar de modo no cambia ningún otro ajuste ni elimina los datos almacenados dentro de los registros de datos.

Los distintos modos posibles se resumen en la Figura 3.25.

MODE[2:0]	MODE
000	Unused
001	Reserved (Do not use)
010	HR only enabled
011	SPO_2 enabled
100–111	Unused

Figura 3.25: Configuraciones de los bits de control del modo de funcionamiento del MAX30100[44].

Donde HR (Heart Rate) significa que solo se usa el led infrarrojo que aporta el valor necesario para poder discriminar la frecuencia cardíaca del usuario. El modo SpO2 utiliza ambos leds para calcular el nivel de saturación de oxígeno en sangre.

La dirección de este registro es la 0x06.

- **SpO2 Configuration:** Este registro establece la configuración del modo SpO2 pero también sirve para la configuración del modo HR, ya que no existe un registro de configuración HR. Se resume el contenido del registro con la Figura 3.26.

SpO2 Configuration (0x07)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
SPO_2 Configuration		SPO2_HI_RES_EN	Reserved	SPO2_SR[2:0]			LED_PW[1:0]		0x07	0x00	R/W

Figura 3.26: Registro SpO2 Configuration[44].

Se describe a continuación el fin de cada bit de este registro:

- Bit 6: SpO2 High Resolution Enable (SPO2_HI_RES_EN).
- Bit 5: Reservado. Poner a nivel bajo por defecto.
- Bits 4 a 2: SpO2 Sample Rate Control. There are several rates to choose.
- Bits 1 a 0: LED Pulse Width Control. Both leds have the same width. Indirectamente ajustan el tiempo de integración del ADC en cada muestra.

La dirección de este registro es la 0x07.

- **LED Configuration:** Este registro controla la corriente que pasa por el led rojo y por el led infrarrojo. De esta manera puede aplicarse una mayor o menor intensidad de luz así como desactivar algún led en concreto ajustando la

intensidad de corriente a 0 mA. Puede verse un resumen de este registro en la Figura 3.27.

LED Configuration (0x09)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
LED Configuration	RED_PA[3:0]				IR_PA[3:0]				0x09	0x00	R/W

Figura 3.27: Registro Led Config[44].

Donde *RED_PA* controla el nivel de corriente del led rojo e *IR_PA* controla el nivel de corriente del led infrarrojo. La dirección de este registro es la 0x09.

- **Temperature data:** Este registro almacena el valor de la temperatura leída por el sensor. Puede verse un resumen del registro a continuación:

Temperature Data (0x16–0x17)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
Temp_Integer	TINT[7:0]								0x16	0x00	R/W
Temp_Fraction					TFRAC[3:0]				0x17	0x00	R/W

Figura 3.28: Registro Temperature Data[44].

Para obtener la temperatura medida, habrá que sumar los dos registros entre ellos de la siguiente manera:

$$T_{MEDIDA} = T_{INTEGER} + T_{FRACTION}$$

Como puede verse se almacenará en el primer registro, *Temp_Integer*, el valor de la parte entera de la temperatura medida y almacenada en complemento a dos y en el segundo registro, *Temp_Fraction*, la parte decimal de la misma, con una precisión de $\frac{1}{16}$ ºC. Este registro involucra dos registros de 1 byte cada uno en las direcciones 0x16 y 0x17 respectivamente.

Ahora que ya se conoce en detalle la estructura del sensor MAX30100 y cómo utilizar y configurar sus distintas funcionalidades, tan solo queda por comprender cómo establecer comunicación con el sensor, es decir, habrá que conocer cómo pueden realizarse las operaciones de lectura y escritura.

Para escribir en el MAX30100 se envía el *slave ID* como primer byte seguido de la dirección del registro en el que queremos escribir para finalmente enviar uno o más bytes que contienen los datos que queremos escribir. El puntero de la dirección del registro se incrementa automáticamente después de cada byte de datos recibido para poder escribir varios bits de manera consecutiva. El *slave ID* consta de 8 bits. Los 7 primeros bits son *1010111*, seguidos de 1 bit que indica si la operación va a ser de

lectura (1) o de escritura (0), de manera que el *slave ID* para escritura es *0xAE* y para lectura es *0xAF*. Puede ver un resumen gráfico del proceso en la Figura 3.29.

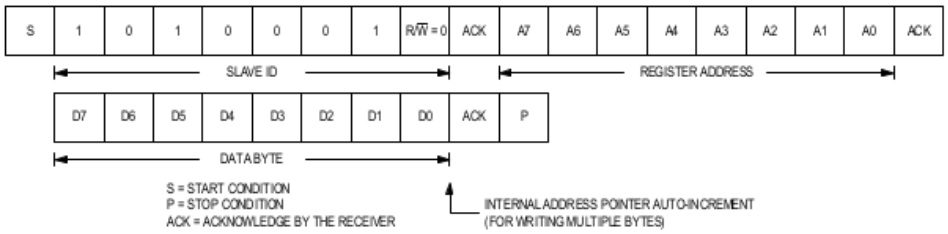

Figura 3.29: Operación de escritura de un byte en el MAX30100[44].

Para la lectura en el MAX30100 primero, al igual que en el caso de escritura, se envía el *slave ID*, pero para solicitar una lectura (0xAF) seguido de la dirección del registro del que queremos leer. Posteriormente se reenvía de nuevo el *slave ID* de lectura y acto seguido el MAX30100 comienza a enviar los bytes de datos del registro que hayamos solicitado. Un resumen del proceso de lectura de un byte puede verse a continuación:

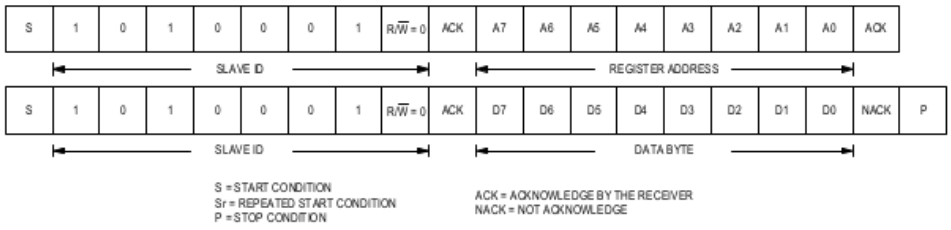

Figura 3.30: Operación de lectura de un byte en el MAX30100[44].

Para el caso de lectura de más de un byte el proceso puede verse resumido también en la Figura 3.31.

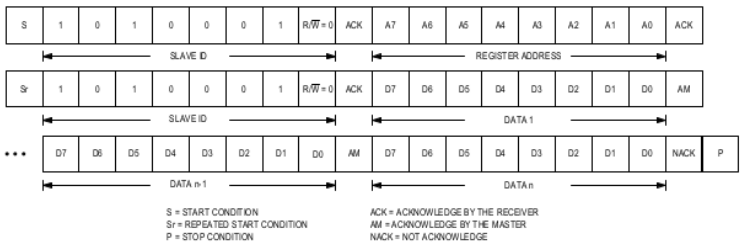

Figura 3.31: Operación de lectura de varios bytes en el MAX30100[44].

Con toda esta información ya se tienen los conocimientos necesarios sobre el uso y funcionalidades que ofrece este sensor de frecuencia cardíaca para comenzar a realizar la implementación.

3.4.2.2. Recursos utilizados para el desarrollo

Antes de realizar la implementación y durante ésta, se han utilizado varios recursos que han permitido aprender más sobre este sensor, así como realizar la implementación.

El sensor Heart Rate Click está compuesto por el sensor MAX30100 y la circuitería necesaria para hacerlo funcionar, como ya se comentó en la sección anterior. Es por esto, que realmente nuestro objetivo es la comprensión del funcionamiento del sensor MAX30100. Una de las primeras cosas que es necesario revisar es el *datasheet*[44]. de este sensor, ya que en él viene la amplia mayoría de la información sobre los registros de los que se compone y sus funcionalidades y configuraciones, así como la manera de comunicarnos con el sensor, tanto para lectura como escritura.

Además de la documentación anterior, para comprender en profundidad el funcionamiento del sensor de frecuencia cardíaca, ha sido conveniente la revisión de un ejemplo de la implementación de la lectura de frecuencia cardíaca y SpO2 utilizando el sensor MAX30100 para clarificar algunas dudas de funcionamiento que pudieran haber quedado después de la lectura del *datasheet*. El ejemplo puede verse en [45].

Conocido ya en profundidad el funcionamiento del sensor y vista su implementación en casos reales, solo queda comprender cómo hacer uso del puerto I2C de la tarjeta de evaluación STM32L053R8. Para ello, se han revisado ejemplos de uso del puerto I2C en tarjetas de tipo STM32 de los provistos por ST Microelectronics. Además, de nuevo para una mejor comprensión, se ha revisado una implementación real del uso del puerto I2C para operaciones de escritura y lectura. El ejemplo puede verse en [46].

Para el desarrollo de la implementación y las pruebas se ha utilizado el entorno de desarrollo *ARM Keil* y la tarjeta de evaluación STM32L053R8, al igual que para el caso de BLE. Para la implementación final se ha unido el código con el del sistema de comunicación BLE y se ha probado la recepción de datos con la aplicación Android.

3.4.2.3. Implementación

En esta sección se da por supuesto el conocimiento sobre el protocolo de comunicación I2C para la implementación, para poner el foco solamente en lo relativo al sensor de frecuencia cardíaca. Para comenzar conviene explicar la estructura de archivos que va a seguirse para la implementación del sensor de frecuencia cardíaca en la tarjeta STM32L053R8. Se describen a continuación los archivos que van a utilizarse para tener una visión general del proyecto:

- **main.c:** Este es el archivo principal del proyecto. En él se configura el puerto I2C de la tarjeta de evaluación y también se configura el sensor de frecuencia cardíaca. Además es aquí donde se realizan las lecturas de los datos del sensor,

para obtener los datos obtenidos por el led infrarrojo, y con ellos se calcula la frecuencia cardíaca del usuario. Utiliza, además de las implementadas en este archivo, las funciones descritas en *mpl.c*.

- **main.h:** Se trata del fichero de cabecera de *main.c*. En este archivo se definen constantes relativas al funcionamiento de la comunicación a través del puerto I2C, que se utilizarán durante la ejecución de nuestro proyecto.

- **mpl.c:** Este archivo actúa de driver para realizar las lecturas y escrituras a través del puerto I2C de la tarjeta de evaluación. En esencia, se definen la función de lectura y de escritura a través del puerto I2C para poder realizar estas operaciones de manera simplificada desde el archivo principal *main.c*.

- **mpl.h:** Se trata del fichero de cabecera de *mpl.c*. En él se definen las direcciones de los registros del sensor MAX30100 a los que se accederá para realizar operaciones de lectura/escritura durante la ejecución desde el archivo *main.c*.

Para la descripción de la implementación, no será necesario describir el contenido de los archivos de cabecera *main.h* y *mpl.h*, ya que su contenido ya ha sido explicado en el resumen anterior. Por tanto, se comenzará explicando inicialmente el contenido del archivo *mpl.c*, que es el que define las funciones simplificadas de lectura y escritura a través del puerto I2C. Para la escritura se define en este archivo la función *MAX30100_writeRegister()* que va a hacer una llamada a una función definida en los drivers HAL de la tarjeta de evaluación para la escritura a través del puerto I2C. La función queda como sigue en el siguiente fragmento de código:

```
// Write a byte of data to a specific address
void MAX30100_writeRegister(char regAddr, uint8_t data){
        while(HAL_I2C_Mem_Write(&I2cHandle, (uint16_t)
            MAX30100_WRITE_ADDRESS, (uint16_t) regAddr, (uint16_t)
            I2C_MEMADD_SIZE_8BIT, &data, 1, 100)!= HAL_OK){}
}
```

Como puede verse, a esta función se le pasan como parámetros la dirección del registro en el que se quiere escribir (regAddr) y el byte de datos que se quiere escribir (data). Con ello, la función llama a la función predefinida en la capa HAL, *HAL_I2C_Mem_Write()* que va a realizar la escritura utilizando el protocolo de comunicación I2C y mandando los bits a través del puerto de comunicación I2C que se haya definido en el archivo principal. La configuración del puerto I2C vendrá dada por el parámetro *I2cHandle* y se define el *slave ID* de escritura en la constante *MAX30100_WRITE_ADDRESS*, que también es un parámetro de la función. En este caso, solo se solicita la escritura de 1 byte. Se utiliza un bucle *while* ya que hasta que no se comprueba que la comunicación ha tenido éxito (!= HAL_OK) esta función se mantiene a la espera. En caso de ocurrir algún error en la comunicación se saltará a la función *Error_Handler()*.

Para la lectura se define la función *MAX30100_readRegister()*, que hará también una llamada a una función definida en la capa HAL, *HAL_I2C_Mem_Read()*, para

la lectura a través del puerto I2C de la tarjeta de evaluación. Se puede observar la función a continuación:

```
// Read a byte of data from a specific address
uint8_t MAX30100_readRegister(uint8_t regAddr){
  uint8_t data=0xff;

        while(HAL_I2C_Mem_Read(&I2cHandle, (uint16_t)
            MAX30100_READ_ADDRESS, (uint16_t) regAddr, (uint16_t)
            I2C_MEMADD_SIZE_8BIT, &data, 1, 100)!= HAL_OK){}

    return data;
}
```

En este caso solo se pasa como parámetro la dirección del registro del que se quiere leer (regAddr), ya que los datos los mandará el sensor. En este caso se manda como parámetro el *slave ID* de lectura, que está contenido en la constante *MAX30100_READ_ADDRESS*. La lectura también se hará de un solo byte que se almacena en la variable *data* y se devuelve como resultado de la llamada a la función.

Ahora que ya se sabe cómo realizar operaciones de lectura/escritura a través de los puertos I2C de la tarjeta de evaluación, se va a pasar a definir la implementación de la funcionalidad principal del sensor de frecuencia cardíaca que se encuentra en el archivo *main.c*. Lo primero es crear un *Handle* que va a almacenar toda la configuración del puerto I2C que se utilice:

```
/* I2C handler declaration */
I2C_HandleTypeDef I2cHandle;
```

A continuación, usando este *Handle*, se configura el puerto I2C de la manera requerida. En este caso lo más relevante es que se activa el puerto I2C 1, con un modo de direccionamiento de 7 bits. Una vez definidos los parámetros de configuración se inicializa el puerto I2C configurado según lo determinado en el *Handle*. Las sentencias necesarias para esto son las siguientes:

```
/* Configure the I2C peripheral */
  I2cHandle.Instance             = I2Cx;
  I2cHandle.Init.Timing          = I2C_TIMING;
  I2cHandle.Init.OwnAddress1     = I2C_ADDRESS;
  I2cHandle.Init.AddressingMode  = I2C_ADDRESSINGMODE_7BIT;
  I2cHandle.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  I2cHandle.Init.OwnAddress2     = 0xFF;
  I2cHandle.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  I2cHandle.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  I2cHandle.Init.NoStretchMode   = I2C_NOSTRETCH_DISABLE;

  if(HAL_I2C_Init(&I2cHandle) != HAL_OK)
  {
    /* Initialization Error */
    Error_Handler();
  }
```

De manera similar al caso de lectura/escritura, se comprueba si la inicialización se

ha realizado correctamente (HAL_OK) y, si no, se deriva a la función *Error_Handler()*. Esto será extensible a todos los comandos I2C definidos en la HAL.

Una vez inicializado el puerto I2C correctamente, ya se podrán realizar operaciones de lectura/escritura en el sensor MAX30100. Por tanto, se va a comenzar a configurar el sensor para que funcione de la manera requerida. Lo primero que se hace es habilitar la interrupción *heart rate ready* que avisará de cuando se ha terminado la adquisición de una muestra (lectura del led infrarrojo). Posteriormente, se configura el sensor en modo frecuencia cardíaca, se configura el registro de SpO2, ya que contiene parámetros de configuración también para el modo de frecuencia cardíaca y finalmente se desactiva el led rojo, ya que en el modo frecuencia cardíaca solo se necesita el infrarrojo. Esto se hace escribiendo en los registros requeridos utilizando las siguientes sentencias:

```
// Enable of heart rate ready interruption
MAX30100_writeRegister(MAX30100_REG_INTERRUPCION_ENABLE,
    MAX30100_INT_PULSO);
// The sensor is configured in heart rate mode
MAX30100_writeRegister(MAX30100_REG_CONF_MOD, MAX30100_MODO_PULSO);
// Parameters of data acquisition spO2
MAX30100_writeRegister(MAX30100_CONFIGURACION_SPO2,
    MAX30100_ALTA_RESOLUCION|MAX30100_PULSO_100_MPS|
    MAX30100_TIEMPO_LED_1600);
// Only infrarred led
MAX30100_writeRegister(MAX30100_CONFIGURACION_LED, MAX30100_ROJO_0|
    MAX30100_INFRARROJO_50000);
```

En este punto, el sensor ya se encuentra configurado y funcionando. A partir de este momento lo que habrá que hacer es solicitar lecturas para ir almacenando los datos obtenidos y calcular así el valor de la frecuencia cardíaca del usuario. Antes, conviene recordar que, como se mencionaba previamente en la sección 3.4.2.1, en el modo de frecuencia cardíaca las muestras solo ocupan 2 bytes, ya que el led rojo no se usa, pero habrá que leer los 4 bytes de todas maneras aunque solo se almacenen los dos primeros por muestra. Dado que se quieren leer los datos de manera continua, se implementará lo que queda de código dentro del bucle *while* del archivo *main.c*. Se hará una comprobación en cada momento de si la interrupción *heart rate ready* está activada para, en cuanto lo esté, solicitar la lectura del dato que se encuentre en la FIFO, teniendo en cuenta que el primer byte es el más significativo, y se guardará el valor de la medida obtenida por el led infrarrojo en la variable denominada *intensidad*. Esta primera parte se hace con las siguientes sentencias:

```
if(MAX30100_readRegister(MAX30100_REG_INTERRUPCION_STATUS)&
    MAX30100_PULSO_LISTO){

        // We request infrarred data
        max30100_buffer[0] = MAX30100_readRegister(MAX30100_FIFO_DATOS);
        max30100_buffer[1] = MAX30100_readRegister(MAX30100_FIFO_DATOS);
        /* In HR mode, bytes 3 and 4 of FIFO are returning zeros, due to
             previous deactivation of red led. Then we must read 3 and 4
            byte in any case to increment read pointer of FIFO and avoid
            to read wrong data */
        max30100_buffer[2] = MAX30100_readRegister(MAX30100_FIFO_DATOS);
```

```
        max30100_buffer[3] = MAX30100_readRegister(MAX30100_FIFO_DATOS);

        // Relevant data is on the first two bytes
        intensidad = max30100_buffer[0];
        intensidad = max30100_buffer[1]|intensidad<<8;

        //If statement continues...
```

Cuando se ponga el dedo sobre el sensor, el valor de la intensidad medida estará en torno a 40.000 y al apartar el dedo este valor disminuirá drásticamente, de manera que se debe colocar una condición para comprobar cuándo se mide un dato real y cuándo no. Hay que saber, que las medidas que devuelve el led infrarrojo irán oscilando en forma de diente de sierra con cada pulsación, de manera que para hacer el cálculo de la frecuencia cardíaca se mide el tiempo entre mínimos de la variable *intensidad*, que es el tiempo de un latido, y de ahí se estiman los latidos por minuto. Se almacenan varios valores y se hace una media de ellos para minimizar el error de medida y esta media es la que se toma como valor de frecuencia cardíaca el latidos por minuto. Para ello se utiliza el siguiente bloque de código:

```
/* If there is something to read */
if(intensidad > 20000){

        /* We calculate heart rate obtaining the number of samples
            between minimums and then, supposing a certain value of
            miliseconds between samples, the value of beats per minute is
             obtained */
        if(intensidad > intensidad_anterior && cambio == 0){
                //Calculamos hr en latidos por minuto
                hr_temp += 60000.0/(num_subida*260);
                cambio = 1;
                num_subida = 1;
                ind_media++;
        }else if(intensidad < intensidad_anterior && cambio == 1){
                cambio = 0;
                num_subida++;
        }else{
                num_subida++;
        }

        // We perform an average of measures to estimate the BPM
        if(ind_media == 100){
                hr = hr_temp/ind_media;
                hr_temp = 0;
                ind_media = 0;
        }

/* If not, there is no heart rate to show */
}else{
        hr = 0;
}
```

De esta manera, se obtiene de forma continua el valor de la frecuencia cardíaca

del usuario cuando pone la yema del dedo sobre el sensor MAX30100, finalizándose la implementación del sensor Heart Rate Click.

3.4.3. Sensor de temperatura

3.4.3.1. Descripción

El sensor que se va a utilizar para la adquisición de datos continuos de temperatura corporal de los usuarios es el LMT70. Para este trabajo se dispone de este sensor incorporado en el módulo de evaluación LMT70EVM de Texas Instruments. Este módulo permite analizar el rendimiento del sensor LMT70 de forma muy sencilla. Puede verse una imagen del módulo de evaluación LMT70EVM en la Figura 3.32.

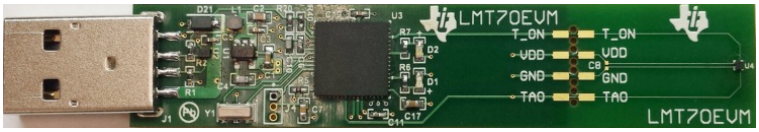

Figura 3.32: Módulo de evaluación LMT70EVM[47].

Este módulo viene en forma de stick USB y monta un microcontrolador MSP430F5528 que interactúa con el USB y con el sensor LMT70. Además, el módulo de evaluación viene con unos huecos perforados que el usuario puede romper para separar el microcontrolador y el sensor LMT70 para realizar medidas de temperaturas a distancia. Finalmente, el módulo dispone de un software que provee Texas Instruments para poder realizar una monitorización de las medidas que recibe el sensor LMT70 en nuestro ordenador. Sin embargo, va a ignorarse esta utilidad y se tomarán las medidas directamente del LMT70. De esta manera, vamos a centrarnos en conocer a fondo este sensor de temperatura por separado para poder realizar la implementación en nuestro sistema[47].

El LMT70 se trata de un sensor de muy pequeño tamaño. Está diseñado para funcionar en aplicaciones que requieran gran precisión, baja potencia y sean de bajo coste como por ejemplo aplicaciones IoT, termómetros médicos o instrumental de gran precisión. Su *output enable* permite a múltiples LMT70 compartir el mismo canal ADC, simplificando la calibración del ADC y reduciendo significativamente el coste total para un sistema de medida de temperatura de precisión. Además el LMT70 tiene también una impedancia de salida baja y lineal que permite una integración perfecta con un ADC o microcontrolador general. Finalmente, mencionar que el LMT70 disipa menos de 36W y tiene un calentamiento muy bajo durante el funcionamiento permitiendo medir un amplio rango de temperaturas con gran precisión[48]. Puede verse a continuación un esquema de sus conexiones:

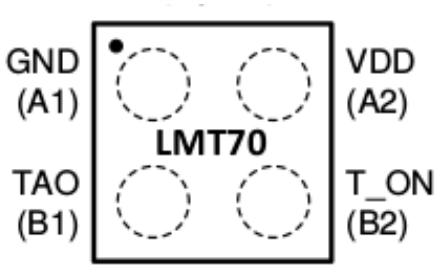

Figura 3.33: Esquema de conexiones del LMT70[48].

Se describe la funcionalidad de cada una de estas conexiones a continuación:

- **GND:** Referencia a tierra del dispositivo.
- **VDD:** Tensión de alimentación del dispositivo. Máximo de 6 Voltios.
- **TAO:** *Temperature Analog Output*. Pin de salida del valor medido de temperatura. Habrá que hacer una transformación del dato para obtener el valor real, pero lo más óptimo es conectar esta salida a un ADC de 12 bits, al igual que el que tiene el microcontrolador MSP430. Se verá más adelante. Los rangos de temperaturas que puede medir van de los -55ºC a los 150ºC.
- **T_ON:** Pin de control de la salida TAO. Si T_ON = 0 entonces la salida TAO está en circuito abierto. Si T_ON = 1 entonces la salida TAO está conectada a la salida del voltaje que contiene el valor de temperatura. Si no se quiere utilizar este pin como enable, se puede cortocircuitar con VDD para habilitar la salida TAO siempre, que es lo que se hará en esta implementación.

Por tanto, en esencia una vez conectado el dispositivo a GND y +VDD, y cortocircuitado T_ON con +VDD, ya se estarán obteniendo medidas de la temperatura obtenidas por el LMT70 de forma continua a través de la salida TAO. Esta salida, da un voltaje de salida, que se define como V_{TAO}, y que luego se transformará al valor de temperatura real medido por el LMT70. Por tanto, se establece una función de transferencia que relaciona el voltaje de salida del sensor y la temperatura medida. En un primer vistazo puede parecer que se trata de una función de transferencia lineal, pero si la inspeccionamos con más detalle puede verse que no es realmente lineal y que puede ser descrita mejor utilizando una función de transferencia con una ecuación de segundo o tercer orden. Finalmente, nos inclinamos por usar la ecuación de tercer orden para el cálculo de la temperatura medida ya que se trata de la ecuación más precisa en un mayor rango de temperaturas. La ecuación es la siguiente[48]:

$$T_M = a(V_{TAO})^3 + b(V_{TAO})^2 + c(V_{TAO}) + d \tag{3.1}$$

Donde V_{TAO} está en mV y T_M es la temperatura real medida en ºC. Además, pueden elegirse los parámetros a, b, c y d para que encajen mejor en el rango entre -55ºC y 150ºC o en el rango entre -10ºC y 110ºC. Dado que las temperaturas que se van a medir se encuentran en valores intermedios y no extremos se decide usar los valores que mejor encajan para el rango de -10ºC a 110ºC que son los siguientes[48]:

a	-1.809628E-09
b	-3.325395E-06
c	-1.814103E-01
d	2.055894E+02

Tabla 3.1: Valores de los parámetros de la ecuación de tercer orden para el cálculo de la temperatura medida por el LMT70 [48].

Lo último que se debe tener en cuenta con el sensor LMT70 es que es posible que en entornos con ruido la medición puede verse alterada. Si existe ruido en el pin TAO se puede poner un condensador entre el pin TAO y tierra. Para entornos ruidosos también se recomienda, desde Texas Instruments, colocar un condensador entre VDD y GND. Finalmente, aunque el LMT70 tiene una capa de protección para reducir la exposición a la luz, es posible que la luz ambiental afecte al funcionamiento del sensor. Dependiendo de la cantidad de luz a la que el sensor esté expuesto, puede esperarse un incremento en el error de medida de la temperatura. El LMT70 es más sensible a la radiación infrarroja. Lo más recomendable es, a la hora de integrar el producto final, incluir protección contra posibles fuentes de luz durante su funcionamiento.

3.4.3.2. Recursos utilizados para el desarrollo

Para hacer funcionar este sensor utilizando la tarjeta de evaluación STM32L053R8 como sistema central, lo primero que se ha hecho,y de cuyo resultado se desprende el contenido de la sección anterior, es estudiar el funcionamiento del sensor LMT70 utilizando el *datasheet*[48]. Una vez ya se conocía su funcionamiento, había que utilizar un ADC como entrada de datos de la tarjeta de evaluación, de manera que se han revisado los ejemplos provistos por ST Microelectronics sobre el uso de los puertos ADC en la tarjeta de evaluación, para implementar la lectura continua de datos del LMT70.

Finalmente, para el desarrollo de la implementación y las pruebas se ha utilizado de nuevo el entorno de desarrollo *ARM Keil* y la tarjeta de evaluación STM32L053R8. Para las pruebas finales se ha unido el código con el del sistema de comunicación BLE, con el del sensor Heart Rate Click y se ha probado la recepción de datos con la aplicación Android.

3.4.3.3. Implementación

Una vez ya se ha comprendido el funcionamiento del sensor LMT70, se pasa a realizar la implementación de su funcionalidad utilizando la tarjeta de evaluación STM32L053R8. En este caso, en comparación con el sensor Heart Rate Click, su implementación va a ser mucho más sencilla.

Lo primero que se debe hacer es definir el *Handle* que va a almacenar la configuración del puerto ADC de la tarjeta de evaluación y otra variable que va a almacenar la información del canal que va a utilizarse en el mismo puerto ADC. Esto lo se hace con las siguientes sentencias:

```
/* ADC handle declaration */
ADC_HandleTypeDef AdcHandle;

/* ADC channel configuration structure declaration */
ADC_ChannelConfTypeDef sConfig;
```

A continuación se va a almacenar la configuración que se quiere establecer en el ADC en estos archivos de configuración. La configuración principal que se quiere para el ADC es que utilice 12 bits y use el canal 4. Esta configuración y otras adicionales se consiguen mediante el siguiente bloque de código:

```
/* AdcHandle Configuration */
AdcHandle.Instance = ADC1;
AdcHandle.Init.OversamplingMode      = DISABLE;
AdcHandle.Init.ClockPrescaler        = ADC_CLOCK_SYNC_PCLK_DIV1;
AdcHandle.Init.LowPowerAutoPowerOff  = DISABLE;
AdcHandle.Init.LowPowerFrequencyMode = ENABLE;
AdcHandle.Init.LowPowerAutoWait      = DISABLE;
AdcHandle.Init.Resolution            = ADC_RESOLUTION_12B;
AdcHandle.Init.SamplingTime          = ADC_SAMPLETIME_7CYCLES_5;
AdcHandle.Init.ScanConvMode          = ADC_SCAN_DIRECTION_FORWARD;
AdcHandle.Init.DataAlign             = ADC_DATAALIGN_RIGHT;
AdcHandle.Init.ContinuousConvMode    = ENABLE;
AdcHandle.Init.DiscontinuousConvMode = DISABLE;
AdcHandle.Init.ExternalTrigConvEdge  = ADC_EXTERNALTRIGCONVEDGE_NONE;
AdcHandle.Init.EOCSelection          = ADC_EOC_SINGLE_CONV;
AdcHandle.Init.DMAContinuousRequests = DISABLE;

/* sConfig Configuration */
sConfig.Channel = ADC_CHANNEL_4; //PA4
```

Una vez se han seleccionado los parámetros del ADC que se quieren en los archivos de configuración, se inicializa el sistema. Lo que se hace es inicializar el ADC, calibrarlo, configurar el canal de uso como el canal 4 y finalmente se comienza la actividad del ADC, empezando a convertir muestras. Esto se hace ejecutando las siguientes funciones:

```
/* Initialize ADC peripheral according to the passed parameters */
  if (HAL_ADC_Init(&AdcHandle) != HAL_OK)
  {
    Error_Handler();
```

```
  }

  /* Start calibration */
  if (HAL_ADCEx_Calibration_Start(&AdcHandle, ADC_SINGLE_ENDED) !=
      HAL_OK){}

  /* Channel configuration */
  /* Select Channel 4 to be converted (PA4 PIN)*/
  sConfig.Channel = ADC_CHANNEL_4;
  if (HAL_ADC_ConfigChannel(&AdcHandle, &sConfig) != HAL_OK){}

 /* Start the conversion process */
  if(HAL_ADC_Start(&AdcHandle) != HAL_OK){}
```

Al igual que en el caso del sensor de frecuencia cardíaca, al realizar cualquiera de estas operaciones de la HAL se espera a que la operación se haya realizado correctamente (HAL_OK) y en caso contrario se llama a la función *Error_Handler()* para gestionar el error.

A partir de este momento, el ADC ya se encuentra recibiendo y convirtiendo las muestras que le llegan. Ahora, habrá que traducir el valor que lee el ADC, que se define como V_{ADC}, al valor que había a la salida del sensor de temperatura y que ya se definió previamente como V_{TAO}. Se sabe que se está utilizando una conversión de 12 bits, de manera que el rango de datos leídos (V_{ADC}) va de 0 a $2^N - 1$, donde N es el número de bits, es decir, de 0 a 4095. También se sabe que el valor de voltaje de referencia utilizado por el ADC es de $V_{ref} = 3,3V$[26],ya que no se utiliza una referencia externa a la tarjeta de evaluación, de manera que podemos obtener el valor de entrada real al puerto ADC mediante la siguiente relación:

$$V_{TAO} = \frac{V_{ADC}V_{Ref}(mV)}{4095}mV \tag{3.2}$$

Una vez se ha calculado este valor, puede sustituirse en la ecuación (3.1) para obtener el valor de la temperatura real que está dando el sensor LMT70. Este valor se almacena en la variable que se ha denominado como *temperature*. Todo esto se hace con las siguientes sentencias de código que se estarán ejecutando de forma continua dentro del bucle while del archivo *main.c*:

```
/* Wait for the end of conversion */
/*  Before starting a new conversion, the current state of the
    peripheral must be checked; if it is busy you need to wait for the
    end of current conversion before starting a new one.*/
HAL_ADC_PollForConversion(&AdcHandle, 10);

/* Check if the continuous conversion of regular channel is finished */
if ((HAL_ADC_GetState(&AdcHandle) & HAL_ADC_STATE_REG_EOC) ==
    HAL_ADC_STATE_REG_EOC)
{
    /* Get the converted value of regular channel */
```

```
    uwADCxConvertedValue = HAL_ADC_GetValue(&AdcHandle);

        /*We obtain the value of output voltage in mV (VREF = 3.3V)*/
        VTAO = uwADCxConvertedValue*3300/4095;

        /* Temperature in celsius degrees is obtained using a third
            order function to be more accurate in the range between -10
            and 110 celsius degrees */
        temperature = a*(VTAO^3) + b*(VTAO^2) + c*VTAO + d;
}
```

Finalizando de esta manera la implementación del sensor LMT70 para obtener valores continuos de temperatura utilizando la tarjeta de evaluación STM32L053R8 como sistema central.

3.4.4. Resultados

Resumiendo los logros que se han conseguido llegando a este punto, se ha logrado tener un sistema de comunicación funcional, tanto de transmisión como recepción, de datos continuos a través de BLE, y en esta sección se ha terminado la implementación del sensor de frecuencia cardíaca Heart Rate Click y del sensor de temperatura LMT70. De manera que, lo último que queda para completar el diseño del prototipo, que era el objetivo de este trabajo, es implementar la adquisición de datos de los sensores en tiempo real, para que se envíen de forma continua a la aplicación Android a través de BLE.

Lo que se hace entonces es unir el código del sensor de frecuencia cardíaca y del sensor de temperatura, con su misma distribución de archivos y posiciones en el código, con el código desarrollado para la transmisión BLE, para tener el sistema de adquisición de datos y de tranmisión de éstos ejecutándose en la misma tarjeta de evaluación que se ha utilizado a lo largo de todo el trabajo. No obstante, aunque se una el código de esta manera, se podrá ver que el sistema no funciona. Esto es debido a que, como cada proyecto se ha creado de manera independiente, sus distintos archivos de configuración e inicialización contienen el código relativo a los periféricos que van a usarse en cada caso, como el módulo de extensión BLE, el puerto I2C, el puerto ADC... De manera que existen dos opciones para resolver esta situación: generar una plantilla de código nuevo utilizando el programa *STM32CubeMX* detallando todos los puertos y configuraciones que van a utilizarse en el sistema final para que solo quede juntar el código, o bien revisar los archivos de configuración de cada proyecto por separado y modificar uno que haga de código central para que tenga habilitadas, configuradas e inicializadas todas las secciones que sean necesarias para el funcionamiento de cada una de las partes del proyecto en uno solo. Finalmente, se ha decidido abogar por realizar esta última opción, de manera que los archivos que habrá que revisar y fusionar son los siguientes:

- **stm32l0xx_hal_msp.c:** Este archivo incluye la inicialización y "de-inicialización" MSP de los periféricos usados en la aplicación.

- **stm32l0xx_it.c y stm32l0xx_it.h:** En este archivo y su cabecera se incluye el código para manejar las interrupciones de los periféricos y excepciones del código.

- **stm32l0xx_hal_conf.h:** Archivo de configuración de la capa HAL donde se habilitan los módulos que van a utilizarse en el proyecto como por ejemplo el módulo I2C, SPI...

Es importante resaltar de nuevo que es estrictamente necesario que los archivos anteriores se encuentren configurados adecuadamente para el correcto funcionamiento del proyecto.

Una vez que se ha aunado el código correctamente, se comienzan a realizar las pruebas del sistema, para comprobar si se recogen los datos de ambos sensores a la vez en tiempo real, si se envían ambos a través de BLE y si se reciben correctamente de forma continua en la aplicación Android, mostrándose los valores en tiempo real de forma continua en la interfaz de usuario. A continuación puede verse cómo queda la tarjeta de evaluación con el módulo de expansión BLE y ambos sensores acoplados para la recogida continua de datos y su transmisión a través de BLE:

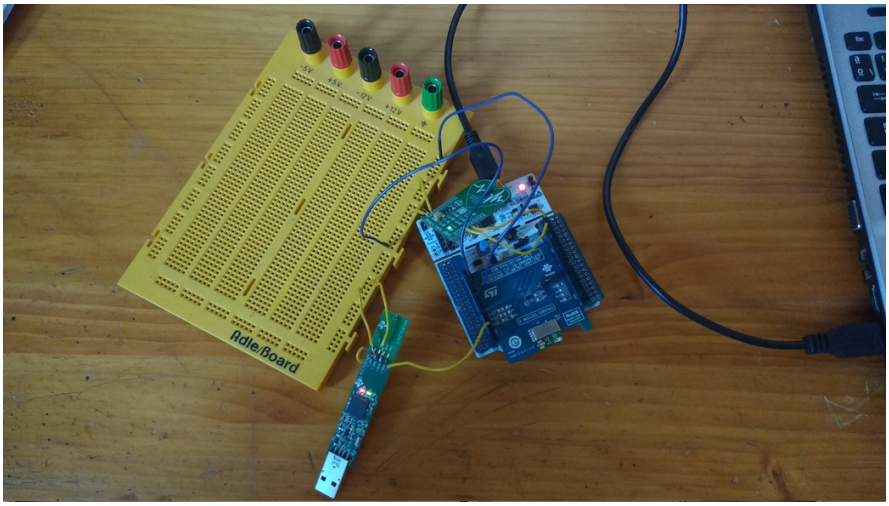

Figura 3.34: Tarjeta de evaluación STM32L053R8 con el módulo de expansión y los sensores acoplados.

Comenzamos entoces a probar el sistema completo. Se inicia el programa en la tarjeta de evaluación para que comiencen a recogerse datos de los sensores y vayan enviándose a través de BLE. A continuación, se inicia la aplicación Android para escanear en busca de dispositivos y encontramos los resultados que pueden apreciarse en la Figura 3.15 como en pruebas anteriores. Seleccionamos el dispositivo STM32 Board para establecer la conexión con él y tratar de comenzar a recibir sus datos, abriéndose la actividad *DataShowActivity*. Al inicio, no se mantiene contacto con ningún sensor y la actividad muestra los siguientes valores en su interfaz:

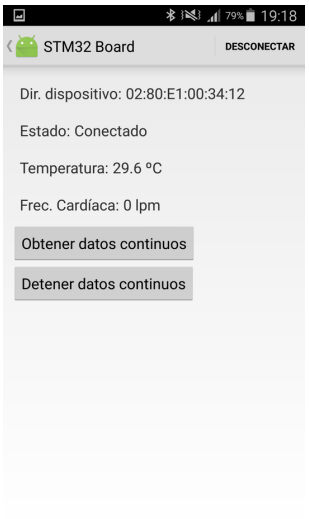

Figura 3.35: DataShowActivity mostrando los datos sin estar tomándose medidas del usuario.

Puede verse que la temperatura es la que capta el sensor del ambiente y que, al no estar poniendo el usuario el dedo en el sensor de frecuencia cardíaca, se muestran cero pulsaciones por minuto. Ha podido comprobarse en este momento que se está enviando por lo menos un dato de temperatura de manera correcta. A continuación, se activa el modo de envío de datos continuos y se establece contacto, primero con el sensor de frecuencia cardíaca y posteriormente con el sensor de temperatura a la vez. Los resultados que se obtienen son los que se pueden apreciar en la Figura 3.36.

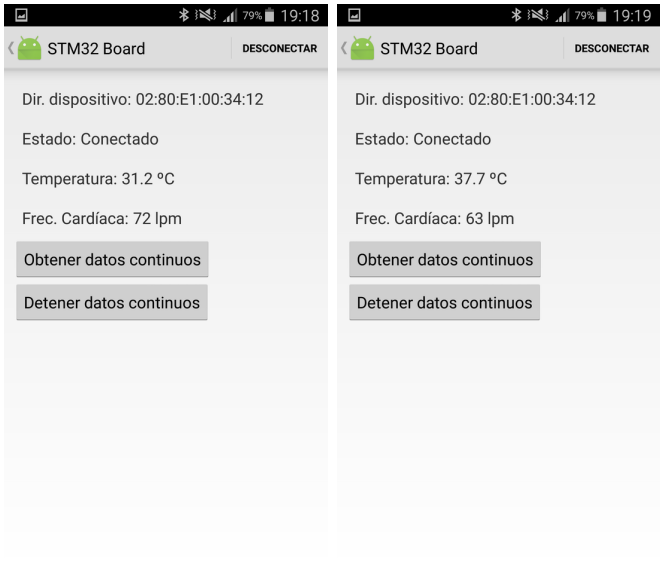

Figura 3.36: Capturas de DataShowActivity cuando se toman medidas del sensor de frecuencia cardíaca (izda) y del sensor de temperatura (drcha).

Puede apreciarse en la imagen de la izquierda cómo desde el momento en que el usuario pone su dedo sobre el sensor MAX30100 comienzan a recibirse lecturas de los valores de la frecuencia cardíaca y en la imagen de la derecha puede comprobarse también que el valor de la temperatura aumenta conforme a la medición tomada del usuario cuando coloca también su dedo sobre el sensor LMT70, adquiriendo también a la vez mientras tanto valores de la frecuencia cardíaca, ya que el usuario no ha dejado de poner su dedo sobre el sensor de frecuencia cardíaca en ningún momento. Todo ello puede verse en tiempo real y que, cuando está activado el modo de adquisición continua de datos, estos valores van cambiando continuamente conforme los valores que se van obteniendo de ambos sensores en cada momento. Puede comprobarse también que el sistema tiene la opción de deshabilitar la recepción continua de datos y habilitarla en cualquier momento, pulsando los botones que se encuentran en la interfaz de la actividad, como ya se comprobó previamente en la sección 3.3.5. Además, y como se comprobó en esta misma sección, el sistema puede conectarse y desconectarse de la tarjeta de evaluación en cualquier momento.

De esta manera, se ha comprobado que se ha finalizado la implementación del prototipo de manera satisfactoria, ya que el sistema completo se ha comprobado capaz de adquirir datos de los sensores en tiempo real y de manera continua, enviar estos datos a través de BLE, recibirlos utilizando una aplicación Android y mostrarlos, de nuevo de manera continua y en tiempo real, en la interfaz de usuario.

Capítulo 4

Conclusiones y líneas futuras

4.1. Conclusiones

Durante la realización de este TFM he trabajado para lograr construir un dispositivo que sirva para una mejora en la calidad de vida de las personas aportando un seguimiento continuo de la actividad y la salud, que con los datos adquiridos y realizando el análisis adecuado, puede resultar en una herramienta de gran valor.

Ateniéndonos a los objetivos presentados en la Sección 1.2.2, puede confirmarse que se ha logrado implementar con éxito el sistema deseado, ya que el sistema de transmisión de datos utilizando Bluetooth Low Energy funciona de la manera deseada, incluyendo el funcionamiento para el envío de datos de forma continua. También se ha diseñado y completado la aplicación Android siendo capaz ésta de recibir y mostrar los datos deseados en tiempo real y de forma continua, permitiendo ciertas interacciones por parte del usuario para su control y finalmente, se han conseguido implementar los sensores para la adquisición de datos fisiológicos reales del usuario completando con éxito el sistema.

A nivel personal, en este trabajo he aprendido y trabajado utilizando distintas ramas de la ingeniería aplicadas a resolver un problema sanitario. He trabajado con dispositivos electrónicos, como los sensores, la tarjeta de evaluación y el módulo de expansión BLE, usando los recursos que me ofrecían para crear el sistema que yo tenía como objetivo. También he aprendido sobre el funcionamiento del protocolo Bluetooth Low Energy para la transmisión de datos de forma inalámbrica con bajo consumo y he trabajado con el sistema operativo Android, para crear una aplicación que fuese capaz de comunicarse a través de BLE para recibir datos que mostrar al usuario en una interfaz. He partido desde la planificación global y el diseño del sistema, pasando por la implementación de cada sección así como finalizando por la realización de pruebas, para comprobar que el sistema funcionaba acorde a lo esperado y la resolución de errores durante todo el proceso.

Además de los conocimientos adquiridos, este TFM ha supuesto una experiencia muy importante, ya que es cuando nos enfrentamos a problemas reales cuando se aprende más sobre la aplicación de la ingeniería en el mundo real, y permite obtener una experiencia práctica que de cara al futuro laboral, ya más próximo que nunca,

que es muy valiosa, tanto desde mi punto de vista de trabajador, como desde el punto de vista de las empresas de cara a valorar las aptitudes de un futuro candidato. Es importante también añadir, que este trabajo me ha permitido afianzar más mi motivación hacia la profesión de Ingeniero Biomédico y más concretamente hacia el diseño de dispositivos electrónicos que ayuden a resolver problemas de la sanidad actual.

De esta manera, este TFM me ha dado la oportunidad de aplicar y afianzar los conocimientos que ya tenía previamente gracias a mi Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación y los adquiridos durante la realización de este Máster en Ingeniería Biomédica a un problema que relaciona la salud y la ingeniería, como es el diseño de dispositivos, en este caso wearables, para la monitorización continua de los parámetros fisiológicos de un usuario.

4.2. Líneas futuras

Para finalizar este trabajo fin de máster, se mencionan a continuación varias líneas futuras de trabajo para ampliar y mejorar el sistema:

- **Adición de nuevos parámetros fisiológicos:** Se propone añadir nuevos sensores que permitan monitorizar una cantidad mayor de parámetros fisiológicos que permitan dar una mayor información al usuario y poder realizar un mejor análisis de su actividad y su salud. Algunos ejemplos de parámetros fisiológicos que pueden añadirse son: datos provenientes de un acelerómetro o de un giroscopio o la respuesta galvánica de la piel entre otros.

- **Implementación del sistema en una placa PCB:** Esta mejora se propone para, una vez diseñado un prototipo con todos los sensores que se requieren para el *wearable* planificado, y que se haya probado lo suficiente como para garantizar un correcto funcionamiento en cualquier situación, realizar la implementación de todos los dispositivos electronicos (sensores, microcontrolador...) en una misma tarjeta PCB. También está incluido añadir la posibilidad de usar este sistema con una batería que permita la carga a través de USB.

- **Diseño final del wearable:** Esta mejora propone realizar el diseño final que ya estaría preparado para su puesta al público. En esta mejora se incluye el diseño exterior del dispositivo, por ejemplo en forma de pulsera o brazalete, el diseño de la PCB acorde a la disposición de los sensores conforme a este diseño previo y las protecciones necesarias para algunos de los sensores, para que no haya problemas durante su funcionamiento, o para el sistema completo, como podría ser plantear el diseño para que el dispositivo sea sumergible.

- **Creación de un sistema de análisis:** Esta última mejora se trata también de la más ambiciosa. Para realizarla se propone crear un sistema de análisis de los datos fisiológicos obtenidos de los sensores, utilizando un algoritmo de aprendizaje que sea capaz de dotar al usuario de datos de utilidad como puede

ser la predicción de su estado futuro de salud en función de datos pasados y actuales, o bien la personalización de rutinas y ejercicios para la mejora de la actividad deportiva o como tratamiento de ciertas patologías que tienen una relación directa con la actividad del usuario.

Bibliografía

[1] Andras Gedeon. *Science and Technology in Medicine*. Springer, 2006.

[2] La tecnología en la medicina. `http://bit.ly/2u0CzgB`. Last accessed 27 April 2017.

[3] Igor Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *ScienceDirect*, 23:89–109, August 2001.

[4] Zhanpeng Jin, Joseph Oresko, Shimeng Huang, and Allen C. Cheng. Hearttogo: A personalized medicine technology for cardiovascular disease prevention and detection. In *Life Science Systems and Applications Workshop*, pages 80–83. IEEE, April 2009.

[5] Joseph A. Cruz and David S. Wishart. Applications of machine learning in cancer prediction and prognosis. *Cancer Informatics Journal*, pages 59–77, 2006.

[6] Kiana Tehrani and Michael Andrew. Wearable technology and wearable devices: Everything you need to know. `http://bit.ly/1ycsN2J`, March 2014. Last accessed 30 April 2017.

[7] Pablo Hernández de Cos and Enrique Moral-Benito. Eficiencia y regulación en el gasto sanitario en los países de la OCDE. Technical report, Banco de España, 2011.

[8] Namkee Ahn, Javier Alonso Meseguer, and José A. Herce San Miguel. Gasto sanitario y envejecimiento de la población en españa. Technical report, Fundación BBVA, 2003.

[9] ESCronicos. La cronicidad en cifras. `http://bit.ly/2m8zbK6`. Last accessed 1 May 2017.

[10] Miriam Pavón Buenache. Análisis e implementación de un sistema en la plataforma arduino y android para obtener constantes vitales orientado a personas con diabetes tipo I. Master's thesis, ETSIT UPM, 2016.

[11] El androide libre. La historia de los wearables: cinco siglos intentando vestir tecnología. `http://bit.ly/2tE1rbH`, 2016. Last accessed 10 May 2017.

[12] Roulette30. Can roulette computers predict numbers and win? `http://bit.ly/2u1dW3y`, 2015. Last accessed 10 May 2017.

[13] Cyborg Anthropology. Steve mann. http://bit.ly/2uG1Af9. Last accessed 10 May 2017.

[14] Wikipedia. Steve mann. http://bit.ly/2uZAezU. Last accessed 10 May 2017.

[15] Dispositivos Wearables. ¿qué es wearable? – los dispositivos vestibles. http://bit.ly/1QZ4JKh. Last accessed 15 May 2017.

[16] Inc. Fitbit. Dispositivos fitbit. http://fitbit.link/2rrmqM8. Last accessed 20 May 2017.

[17] Inc. Fitbit. Fitbit alta hr. pulsera de fitness y ritmo cardíaco. http://fitbit.link/2tEelqt. Last accessed 20 May 2017.

[18] Garmin. Garmin forerunner 735xt. fitness y outdoor. running. http://fitbit.link/2tEelqt. Last accessed 20 May 2017.

[19] Runtastic. Runtastic heart rate combo monitor. http://bit.ly/2t36Nvg. Last accessed 20 May 2017.

[20] Samsung. Samsung gear s3 frontier. http://bit.ly/2t37xAs. Last accessed 20 May 2017.

[21] Tom H Van De Belt, Lucien Engelen, Sivera AA Berben, and Lisette Schoonhoven. Definition of health 2.0 and medicine 2.0: A systematic review. *Journal of Medical Internet Research*, 2010.

[22] Wikipedia. Salud 2.0. http://bit.ly/2uGs3ct. Last accessed 23 May 2017.

[23] Julio Bonis, Juan J Sancho, and Ferran Sanz. Sistemas informáticos de soporte a la decisión clínica. *Medicina Clínica*, pages 39–44, 2004.

[24] Silvia Layes, M. Falappa, and G. Simari. Sistemas de soporte a las decisiones clínicas. In *4to Congreso Argentino de Informatica y Salud, CAIS 2013*, pages 291–300. JAIIO, April 2013.

[25] Brainlab. Software de planificación quirúrgica. http://bit.ly/2tEoQdb. Last accessed 23 May 2017.

[26] ST Microelectronics. *STM32L053R8 Datasheet. Ultra-low-power 32-bit MCU ARM-based Cortex-M0+, up to 64KB Flash, 8KB SRAM, 2KB EEPROM, LCD, USB, ADC, DAC.*, textscRev 7 edition, October 2016.

[27] ST Microelectronics. Stm32l053r8. Ultra-low-power ARM Cortex-M0+ MCU with 64 Kbytes Flash, 32 MHz CPU, USB, LCD. http://bit.ly/2uHwFig. Last accessed 26 May 2017.

[28] ARMmbed. Nucleo-l053r8. affordable and flexible platform to ease prototyping using a stm32l053r8t6 microcontroller. http://bit.ly/2sFHbFG. Last accessed 26 May 2017.

[29] Inc. Bluetooth SIG. Bluetooth. `http://bit.ly/1XR4Rlf`. Last accessed 02 June 2017.

[30] Wikipedia. Bluetooth. `http://bit.ly/1xElCQX`. Last accessed 02 June 2017.

[31] Inc. Bluetooth SIG. Bluetooth 5: What it is all about. `http://bit.ly/2h3Y9bm`. Last accessed 02 June 2017.

[32] Wikipedia. Bluetooth low energy. `http://bit.ly/2rE3HBu`. Last accessed 02 June 2017.

[33] Ian Poole. Bluetooth low energy. an overview of the bluetooth low energy system formerly known as wibree standard. `http://bit.ly/2u3L2Qh`. Last accessed 02 June 2017.

[34] ST Microelectronics. *Getting started with the X-CUBE-BLE1 Bluetooth Low Energy software expansion for STM32Cube.*, textscRev 2 edition, January 2017.

[35] RF Wireless World. Bluetooth smart, bluetooth low energy tutorial, ble tutorial. `http://bit.ly/2v2wb5S`. Last accessed 04 June 2017.

[36] Inc. Bluetooth SIG. Technical considerations for bluetooth low energy application developers. `http://bit.ly/2u3GnxQ`. Last accessed 04 June 2017.

[37] ST Microelectronics. X-nucleo-idb04a1. bluetooth low energy expansion board based on bluenrg for stm32 nucleo. `http://bit.ly/2u8zcoN`. Last accessed 05 June 2017.

[38] ST Microelectronics. X-nucleo-idb05a1. bluetooth low energy expansion board based on spbtle-rf module for stm32 nucleo. `http://bit.ly/2u92Dam`. Last accessed 05 June 2017.

[39] Juan Antonio Pascual. Se presenta bluetooth 4.1. ¿qué aporta? `http://bit.ly/2u92Dam`, Diciembre 2013. Last accessed 05 June 2017.

[40] ST Microelectronics. X-cube-ble1. bluetooth low energy software expansion for stm32cube. `http://bit.ly/2sGlLIi`. Last accessed 05 June 2017.

[41] ST Microelectronics NV. Bluenrg app. `http://bit.ly/2sVGps6`. Last accessed 06 June 2017.

[42] Google Developers. Bluetooth low energy. `http://bit.ly/19kqB3c`. Last accessed 07 June 2017.

[43] Mikroelektronika. Heart rate click. description and specifications. `http://bit.ly/2u9lLVw`. Last accessed 12 June 2017.

[44] Maxim Integrated Products, Inc. *MAX30100. Pulse Oximeter and Heart-Rate Sensor IC for Wearable Health.*, Rev 0 edition, September 2014.

[45] Víctor Ventura. MAX30100 sensor de latido de corazón y oxímetro de pulso con comunicaciones I2C para wearables de salud. `http://bit.ly/2uRncEu`, 2016.

[46] Jean-Christophe Toussaint. Interfacing a STM32L053-discovery with an I2C sensor. `http://bit.ly/2uxUrgQ`, March 2017.

[47] Texas Instruments Inc. Lmt70 evaluation module precise analog output temperature sensor with output enable. `http://bit.ly/2tG6ngn`. Last accessed 12 June 2017.

[48] Texas Instruments. *LMT70, LMT70A ±0.05ºC Precision Analog Temperature Sensor, RTD and Precision NTC. Thermistor IC.*, textscRev 07/15 edition, March 2015.

Apéndices

Apéndice I. Manual de instalación y del desarrollador

Este manual pretende ser una guía para futuros desarrolladores que quieran hacer crecer este proyecto. De esta manera, se va a detallar el software utilizado, el proceso seguido para su instalación y un breve tutorial de su uso. Posteriormente, se va a mostrar como se interconecta todo el hardware para que quede preparado para ejecutar el código del sistema. Finalmente, se va a adjuntar todo el código utilizado para el desarrollo de este TFM, describiendo brevemente la función de cada una de estas secciones de código como parte de un sistema final. Conviene aclarar antes de comenzar, que el uso del software que se menciona en esta guía, así como la disposición y conexionado del hardware es opcional de cara a que el sistema funcione, es decir, pueden utilizarse otros entornos de desarrollo para crear y probar código y se pueden utilizar otros puertos diferentes de la tarjeta de evaluación para recibir los datos de los sensores.

Lo primero que debe hacerse es preparar el entorno para poder visualizar, desarrollar, ejecutar y probar el código de las aplicaciones que han sido desarrolladas en este TFM. Los entornos que se han utilizado para el desarrollo de este trabajo, que ya han sido mencionados previamente, son el *ARM Keil* para la creación de programas de control de la tarjeta de evaluación STM32L053R8, y el *Android Studio* para el desarrollo de la aplicación Android. Para instalar el *ARM Keil* hay que acceder a la siguiente página para descargar el programa: http://www2.keil.com/mdk5/install. A continuación, hay que pulsar sobre el botón *Download MDK-Core* que llevará a otra página donde hay que rellenar un formulario para acceder a la descarga y se obtendrá finalmente el software. Posteriormente, se procede a su instalación. Conviene aclarar que este programa tan solo es válido para su uso en sistemas operativos Windows.

Nada más terminar la instalación de *ARM Keil* se va a iniciar una utilidad denominada el *Pack Installer*. Si no se inicializa automáticamente se deberá acceder usando el menú *Project - Manage - Pack Installer*. Esta utilidad permite instalar los paquetes software que incluyen lo necesario para crear programas para un objetivo específico. En este caso, tendremos que instalar los paquetes de la tarjeta de evaluación que estamos usando en este proyecto, es decir, la tarjeta STM32L053R8Tx. Puede verse el menú del *Pack Installer* en la Figura 4.1.

Además, estos paquetes ofrecen opcionalmente la posibilidad de instalar numerosos ejemplos que son de gran utilidad para la comprensión del funcionamiento de los periféricos de la tarjeta de evaluación.

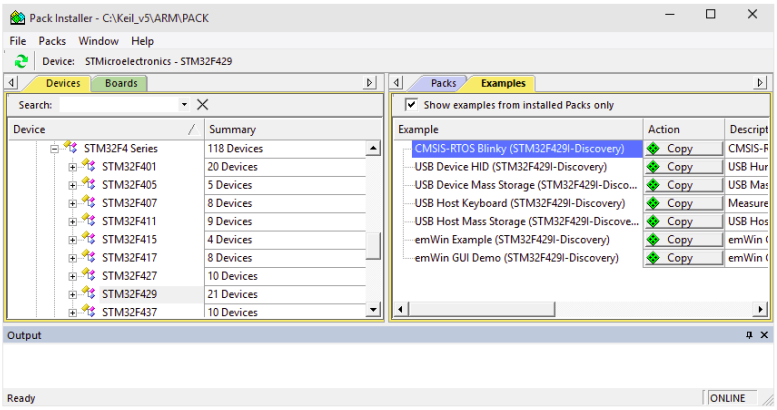

Figura 4.1: Menú del Pack Installer.

Una vez ya está instalado todo el software para hacer uso de nuestra tarjeta de evaluación podemos comenzar a usar el *ARM Keil* para crear un nuevo proyecto o para cargar uno ya creado previamente, como es este caso particular. La estructura de carpetas que tiene el proyecto es la siguiente:

- **Raíz:** Aquí se encuentran los archivos de configuración de inicio (startup) y los archivos para iniciar el proyecto en *ARM Keil* (extensión .uvprojx para abrir el programa). No habrá que modificarlos.
- **Src:** Dentro de esta carpeta se encuentran los archivos que contienen el código principal del proyecto. De estos archivos solo habrá que usar *main.c y sensor_service.c* para modificar la funcionalidad del programa. En algunos casos es posible que haya que modificar los archivos de configuración como se menciona en la Sección 3.4.4.
- **Inc:** Aquí se encuentran los archivos de cabecera de los programas principales de la carpeta Src. De aquí solo habrá que modificar los archivos *main.h y sensor_service.h*. En algunos casos habrá que modificar las cabeceras de los archivos de configuración como se menciona en la Sección 3.4.4.
- **Drivers:** En esta carpeta se encuentran los drivers que controlan la tarjeta de evaluación STM32L053R8. Caben destacar los archivos *mpl.c y mpl.h*, que se encuentran en la carpeta MPL, que contienen las funciones de lectura y escritura I2C simplificadas como ya se mencionó en la Sección 3.4.2.3. No habrá que modificar nada de aquí, a excepción quizás de los archivos *mpl.c y mpl.h*.
- **Middlewares:** En esta carpeta se encuentran los archivos necesarios para poder hacer uso del módulo de expansión BLE de ST Microelectronics. No habrá que modificar nada de aquí.
- **STM32L053R8_NUCLEO:** Finalmente, en esta carpeta aparecen diversos archivos resultado de compilar el proyecto, que en principio carecen de interés de cara al desarrollo.

Por tanto, para abrir un proyecto en *ARM Keil*, hay que acceder a la carpeta donde se encuentre alojado y abrir el archivo que tenga la extensión *.uvprojx* y se cargará en el programa. Se verá entonces el programa como en la Figura 4.2 que muestra el *ARM Keil* cuando ha abierto uno de los ejemplos, que es el Blinky.

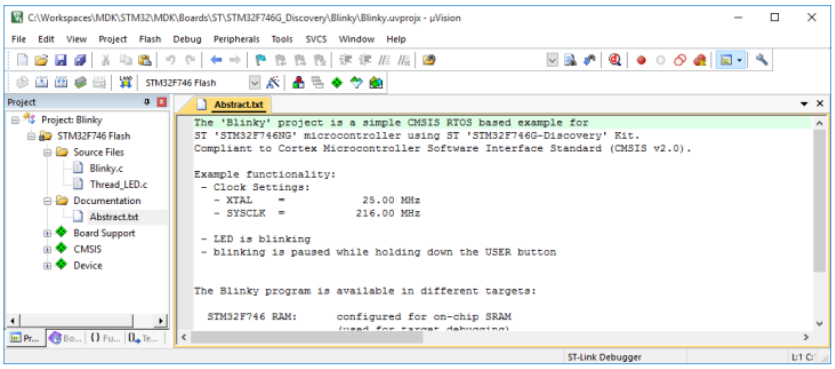

Figura 4.2: Interfaz del ARM Keil con el proyecto Blinky abierto.

Se puede ver que la estructura de archivos se carga en el menú de la izquierda y el código de los archivos se carga en la parte central. En el menú superior hay varios botones que merece la pena explicar, ya que se han utilizado durante el desarrollo del proyecto. En la Figura 4.3 se ven 3 botones: el botón *Build Application*, el botón *Load application* y el botón *Debug Application* de izquierda a derecha respectivamente.

Figura 4.3: Botones de utilidad para el desarrollo del menú superior de ARM Keil.

El botón *Build Application* va a permitir compilar la aplicación. En caso de haber concluido con éxito o de existir algún error, podrá verse en el menú inferior denominado *Build Output*. Puede verse un ejemplo de ello en la Figura 4.4.

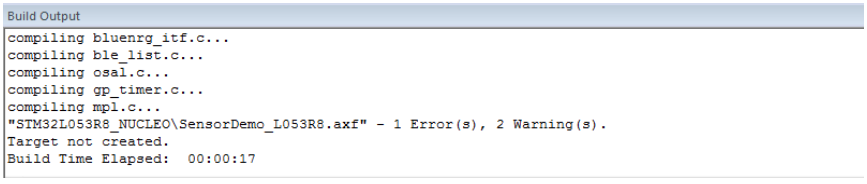

Figura 4.4: Menú Building Output de ARM Keil después de haber compilado con éxito una aplicación.

El botón *Load Application* va a permitir descargar el programa en la tarjeta de evaluación que se haya asignado como objetivo, en este caso la tarjeta STM32L053R8. Una vez descargado el programa en la tarjeta, comenzará a ejecutarse. Finalmente, el botón *Debug Application* permite entrar en el modo debug del *ARM Keil* para poder probar todos los aspectos del programa. Para entrar es necesario que la tarjeta objetivo se encuentre conectada al ordenador. Cuando se entra al modo debug se abre una nueva interfaz como la que se puede observar en la Figura 4.5.

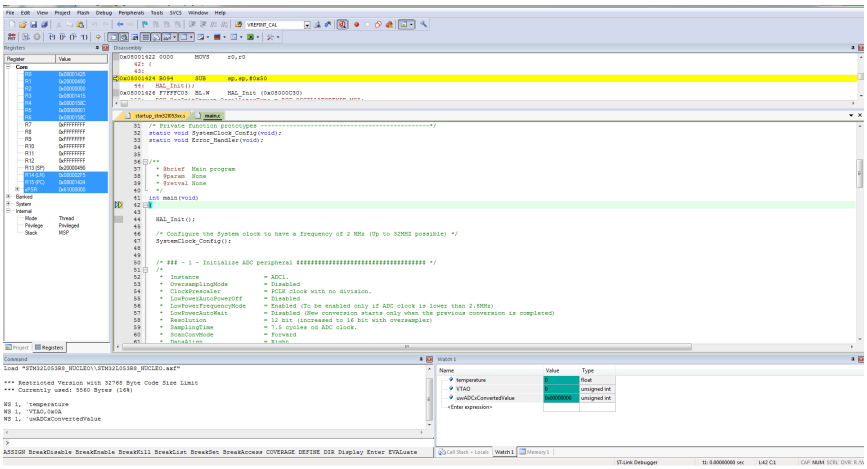

Figura 4.5: Interfaz del modo debug del ARM Keil.

En esta nueva interfaz puede verse que en el centro sigue estando el código, pero ahora en la parte superior aparece una nueva ventana, denominada *Disassembly*, con el código en ensamblador del proyecto, otra ventana a la izquierda con el contenido de los registros, un nuevo menú superior con distintas opciones y una ventana en la parte inferior derecha, denominada *Watch 1*, que puede habilitarse para ver cómo va cambiando el valor de las variables de las que nosotros queramos realizar un seguimiento en tiempo real. En el caso del ejemplo de la Figura 4.5 se ven variables asociadas al sensor de temperatura LMT70. Para comenzar la ejecución del código se pulsará el botón *Run* y para pararla se pulsará el botón *Stop*, que pueden verse de izquierda a derecha respectivamente en la Figura 4.6.

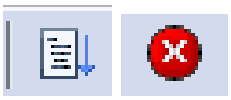

Figura 4.6: Interfaz del modo debug del ARM Keil.

El modo debug incluye multitud de opciones adicionales que pueden ser de utilidad como poner puntos de parada o ir ejecutando el código sentencia a sentencia o bloque a bloque, pero se han mencionado tan solo las opciones que son más relevantes de

cara a la comprensión del funcionamiento de este modo. Finalmente, hay que tener en cuenta que la licencia gratuita de *ARM Keil* tan solo permite compilar, montar, usar el modo debug y ejecutar programas de tamaño menor a 32 KB, que para esta aplicación son más que necesarios.

Ahora es el turno de la instalación del entorno de desarrollo de la aplicación Android. Como ya se explicó en la Sección 3.3.4.1, el entorno de desarrollo que se va a utilizar para crear la aplicación Android es Android Studio. En este caso existe versión tanto para Windows como Mac o Linux, y se puede descargar desde este enlace: https://developer.android.com/studio/install.html?hl=es-419. Una vez se descargue el archivo se instala y se abre el programa. Al abrirlo se pedirá que o bien creemos un nuevo proyecto o bien abramos uno ya creado, que es lo que se hará. Una vez abierto Android Studio, podemos ver cómo es su interfaz en la Figura 4.7.

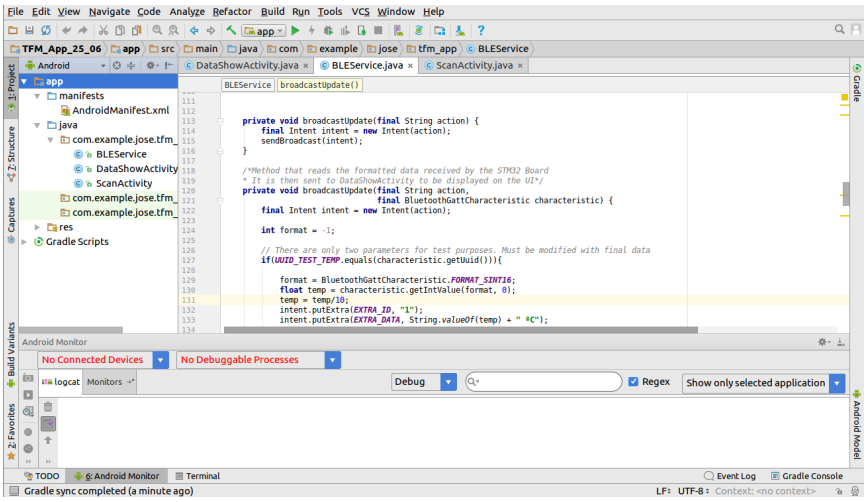

Figura 4.7: Interfaz de Android Studio.

Puede verse que sigue la misma estructura que muchos entornos de desarrollo, incluido el *ARM Keil*, con la estructura de los archivos del proyecto a la izquierda, el código en el centro y un menú superior con distintas funcionalidades para el desarrollador. Aunque Android Studio ofrece un sinfín de funciones útiles lo que realmente se hará es crear los archivos que contienen la funcionalidad de las actividades (archivos java), los layouts que contienen el código que define la interfaz gráfica de cada actividad (archivos XML) y posteriormente probar la aplicación pulsando el botón de la Figura 4.8.

Figura 4.8: Botón de Debug de Android Studio.

Esté botón inicializa el modo debug. Primero se abre una ventana como la que se ve en la Figura 4.9 donde se elige el dispositivo sobre el que va a "correr" nuestra aplicación, que puede ser o bien un emulador o bien un Smartphone con sistema operativo Android.

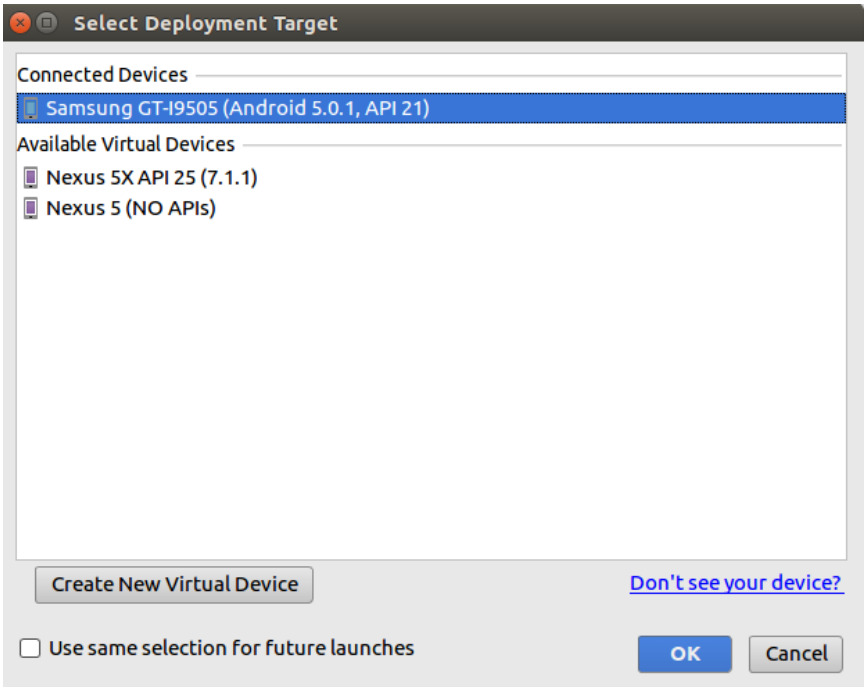

Figura 4.9: Interfaz de selección de dispositivos para el modo debug.

Al elegir una opción, se instala la app y comienza a ejecutarse. Se podrán probar las distintas funcionalidades y se podrán ver los resultados de la ejecución en tiempo real en el menú inferior de Android Studio, denominado Android Monitor, y que puede verse en la parte inferior de la Figura 4.7. En función de cómo se haya programado el código, se podrá ver información de utilidad y, en caso de error, podrá revisarse este "Log" para ver cuál ha sido el error y arreglarlo.

En este punto ya se ha explicado cómo instalar los dos entornos de desarrollo utilizados a lo largo de este TFM. En la Figura 4.10 se puede ver un esquema de cómo se ha realizado el conexionado de los sensores con la tarjeta de evaluación STM32L053R8.

No aparece el módulo de expansión BLE en el esquema de la Figura 4.10 ya que simplemente se conecta en los pines tipo Arduino de la tarjeta de evaluación. Dado que están todos utilizados por el módulo de expansión BLE, este módulo dispone de conectores superiores de tipo Arduino que será donde se conecten las conexiones de los sensores, teniendo en cuenta los puertos que deja libres el módulo de expansión BLE, tal como vimos en la Sección 3.3.3.3 y que la disposición de los puertos (número y tipo de puerto) es la misma que para la tarjeta de evaluación STM32L053R8.

Como parte final de esta guía se adjunta el código desarrollado durante este

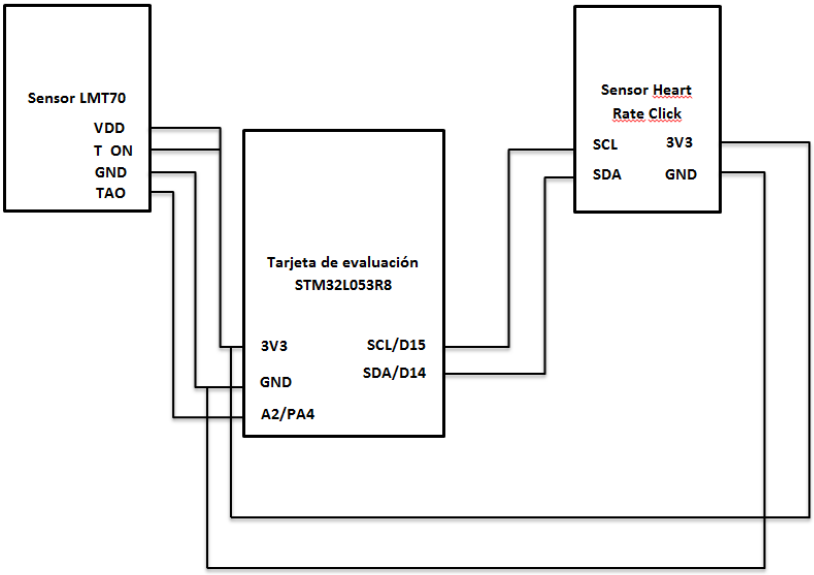

Figura 4.10: Esquema de conexiones de la tarjeta de evaluación con los sensores.

TFM con una breve descripción de la funcionalidad de cada archivo. Se adjunta el código relativo a la funcionalidad principal del proyecto, pero no se adjunta el código de los achivos de configuración del código de la tarjeta de evaluación u otros archivos menores en el caso de la aplicación Android por no hacer muy extenso el documento y que además puede ser consultado por el futuro desarrollador revisando los archivos directamente del directorio del proyecto. Su localización se ha mencionado previamente en esta misma sección. Todo el código está comentado para una mejor comprensión por parte del futuro desarrollador.

Código de la Tarjeta de Evaluación STM32L053R8

- **main.c:** Este archivo contiene el código principal de la aplicación que va a ejecutarse en la tarjeta de evaluación. Se ha juntado el código del sistema de transmisión BLE y el de la adquisición de datos del sensor de frecuencia cardíaca y de temperatura en este mismo archivo. Desde este archivo se utilizan funcionalidades que se encuentran en los archivos *sensor_service.c* y *mpl.c*. El código completo se encuentra a continuación:

```
/**
  * @file    main.c
  * @author  Jose Carlos Montesinos
  * @version V1.0.0
  * @date    22-May-2017
  * @brief   This application contains a code to connect and send
     data to an
        *                                                Android app
            through Bluetooth Low Energy. The communication is done
  *               using a Nucleo board and a Smartphone with BLE.
  */

#include "cube_hal.h"
#include "main.h"
#include "mpl.h"

#include "osal.h"
#include "sensor_service.h"
#include "debug.h"
#include "stm32_bluenrg_ble.h"
#include "bluenrg_utils.h"

#define BDADDR_SIZE 6

extern volatile uint8_t set_connectable;
extern volatile int connected;
uint8_t bnrg_expansion_board = IDB04A1; /* It is possible to use
    both BT expansion boards */


/* ADC handle declaration */
ADC_HandleTypeDef               AdcHandle;

/* ADC channel configuration structure declaration */
ADC_ChannelConfTypeDef          sConfig;

/* Variable used to get converted value */
__IO uint32_t uwADCxConvertedValue = 0;
/* Variable used to store calculated value of TAO output voltage */
uint32_t VTAO = 0;
/* Variable to store obtained body temperature in celsius degrees*/
float temperature = 0;
float a = -1.809628E-09;
float b = -3.325395E-06;
```

```
float c = -1.814103E-01;
float d = 2.055894E+02;

/* HR3 Click */
#define I2C_ADDRESS          0x30F
#define I2C_TIMING        0x00B1112E
uint8_t max30100_buffer[4];
unsigned int intensidad;
unsigned int intensidad_anterior=0;
unsigned int hr = 0;
unsigned int hr_temp = 0;
unsigned int cambio = 0;
unsigned int num_subida = 1;
unsigned int ind_media = 0;

/* I2C handler declaration */
I2C_HandleTypeDef I2cHandle;


/* Private function prototypes */
void User_Process();
static void Error_Handler(void);

/*Main function to start sending data through BLE to third devices
    */
int main(void)
{
  const char *name = "BlueNRG";
  uint8_t SERVER_BDADDR[] = {0x12, 0x34, 0x00, 0xE1, 0x80, 0x03};
  uint8_t bdaddr[BDADDR_SIZE];
  uint16_t service_handle, dev_name_char_handle,
      appearance_char_handle;

  uint8_t  hwVersion;
  uint16_t fwVersion;

  int ret;

  /* STM32Cube HAL library initialization*/
  HAL_Init();

  /* Configure the User Button in GPIO Mode */
  BSP_PB_Init(BUTTON_KEY, BUTTON_MODE_GPIO);

  /* Configure the system clock */
  SystemClock_Config();

          /* Configure LED2 */
  BSP_LED_Init(LED2);

        /* Initialize ADC peripheral*/
  /*
   *  Instance                  = ADC1.
   *  OversamplingMode          = Disabled
```

```
  *   ClockPrescaler          = PCLK clock with no division.
  *   LowPowerAutoPowerOff    = Disabled
  *   LowPowerFrequencyMode   = Enabled (To be enabled only if
      ADC clock is lower than 2.8MHz)
  *   LowPowerAutoWait        = Disabled (New conversion starts
      only when the previous conversion is completed)
  *   Resolution              = 12 bit (increased to 16 bit with
      oversampler)
  *   SamplingTime            = 7.5 cycles od ADC clock.
  *   ScanConvMode            = Forward
  *   DataAlign               = Right
  *   ContinuousConvMode      = Enabled
  *   DiscontinuousConvMode   = Disabled
  *   ExternalTrigConvEdge    = None (Software start)
  *   EOCSelection            = End Of Conversion event
  *   DMAContinuousRequests   = DISABLE
  */

AdcHandle.Instance = ADC1;

AdcHandle.Init.OversamplingMode      = DISABLE;

AdcHandle.Init.ClockPrescaler        = ADC_CLOCK_SYNC_PCLK_DIV1;
AdcHandle.Init.LowPowerAutoPowerOff  = DISABLE;
AdcHandle.Init.LowPowerFrequencyMode = ENABLE;
AdcHandle.Init.LowPowerAutoWait      = DISABLE;

AdcHandle.Init.Resolution            = ADC_RESOLUTION_12B;
AdcHandle.Init.SamplingTime          = ADC_SAMPLETIME_7CYCLES_5;
AdcHandle.Init.ScanConvMode          = ADC_SCAN_DIRECTION_FORWARD
    ;
AdcHandle.Init.DataAlign             = ADC_DATAALIGN_RIGHT;
AdcHandle.Init.ContinuousConvMode    = ENABLE;
AdcHandle.Init.DiscontinuousConvMode = DISABLE;
AdcHandle.Init.ExternalTrigConvEdge  =
    ADC_EXTERNALTRIGCONVEDGE_NONE;
AdcHandle.Init.EOCSelection          = ADC_EOC_SINGLE_CONV;
AdcHandle.Init.DMAContinuousRequests = DISABLE;

/* Initialize ADC peripheral according to the passed parameters
    */
if (HAL_ADC_Init(&AdcHandle) != HAL_OK)
{
  Error_Handler();
}


/*Start calibration*/
if (HAL_ADCEx_Calibration_Start(&AdcHandle, ADC_SINGLE_ENDED) !=
    HAL_OK)
{
  Error_Handler();
}
```

```
  /* Channel configuration */
  /* Select Channel 4 to be converted (PA4 PIN)*/
  sConfig.Channel = ADC_CHANNEL_4;
  if (HAL_ADC_ConfigChannel(&AdcHandle, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

 /* Start the conversion process */
  if(HAL_ADC_Start(&AdcHandle) != HAL_OK)
  {
    /* Start Conversion Error */
    Error_Handler();
  }

        /* Configure the I2C peripheral */
  I2cHandle.Instance             = I2Cx;
  I2cHandle.Init.Timing          = I2C_TIMING;
  I2cHandle.Init.OwnAddress1     = I2C_ADDRESS;

//  I2cHandle.Init.AddressingMode  = I2C_ADDRESSINGMODE_10BIT;

  /* Very important. Device address is 7 bits long */
  I2cHandle.Init.AddressingMode  = I2C_ADDRESSINGMODE_7BIT;
  I2cHandle.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  I2cHandle.Init.OwnAddress2     = 0xFF;
  I2cHandle.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  I2cHandle.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  I2cHandle.Init.NoStretchMode   = I2C_NOSTRETCH_DISABLE;

  if(HAL_I2C_Init(&I2cHandle) != HAL_OK)
  {
    /* Initialization Error */
    Error_Handler();
  }


        // Enable of heart rate ready interruption
        MAX30100_writeRegister(MAX30100_REG_INTERRUPCION_ENABLE,
            MAX30100_INT_PULSO);
        // The sensor is configured in heart rate mode
        MAX30100_writeRegister(MAX30100_REG_CONF_MOD,
            MAX30100_MODO_PULSO);
        // Parameters of data acquisition spO2
        MAX30100_writeRegister(MAX30100_CONFIGURACION_SPO2,
            MAX30100_ALTA_RESOLUCION|MAX30100_PULSO_100_MPS|
            MAX30100_TIEMPO_LED_1600);
        // Only infrarred led
        MAX30100_writeRegister(MAX30100_CONFIGURACION_LED,
            MAX30100_ROJO_0|MAX30100_INFRARROJO_50000);
        // Reset of FIFO registers
        MAX30100_writeRegister(MAX30100_FIFO_PUNTERO_ESCRITURA, 0
            x00);
        MAX30100_writeRegister(MAX30100_FIFO_DESBORDAMIENTO, 0x00);
```

```
        MAX30100_writeRegister(MAX30100_FIFO_PUNTERO_LECTURA, 0x00)
            ;


/* Initialize the BlueNRG SPI driver */
BNRG_SPI_Init();

/* Initialize the BlueNRG HCI */
HCI_Init();

/* Reset BlueNRG hardware */
BlueNRG_RST();

/* get the BlueNRG HW and FW versions */
getBlueNRGVersion(&hwVersion, &fwVersion);

/*
 * Reset BlueNRG again otherwise we won't
 * be able to change its MAC address.
 * aci_hal_write_config_data() must be the first
 * command after reset otherwise it will fail.
 */
BlueNRG_RST();

//PRINTF("HWver %d, FWver %d", hwVersion, fwVersion);

if (hwVersion > 0x30) { /* X-NUCLEO-IDB05A1 expansion board is
    used */
  bnrg_expansion_board = IDB05A1;
  /*
   * Change the MAC address to avoid issues with Android cache:
   * if different boards have the same MAC address, Android
   * applications unless you restart Bluetooth on tablet/phone
   */
  SERVER_BDADDR[5] = 0x02;
}

/* The Nucleo board must be configured as SERVER */
Osal_MemCpy(bdaddr, SERVER_BDADDR, sizeof(SERVER_BDADDR));

ret = aci_hal_write_config_data(CONFIG_DATA_PUBADDR_OFFSET,
                                CONFIG_DATA_PUBADDR_LEN,
                                bdaddr);
if(ret){
  PRINTF("Setting BD_ADDR failed.\n");
}

ret = aci_gatt_init();
if(ret){
  PRINTF("GATT_Init failed.\n");
}

      /*The expansion board is configured as a peripheral device.
        Android app will be the central device*/
```

```
if (bnrg_expansion_board == IDB05A1) {
  ret = aci_gap_init_IDB05A1(GAP_PERIPHERAL_ROLE_IDB05A1, 0, 0x07
      , &service_handle, &dev_name_char_handle, &
      appearance_char_handle);
}
else {
  ret = aci_gap_init_IDB04A1(GAP_PERIPHERAL_ROLE_IDB04A1, &
      service_handle, &dev_name_char_handle, &
      appearance_char_handle);
}

if(ret != BLE_STATUS_SUCCESS){
  PRINTF("GAP_Init failed.\n");
}

      /*Set device name*/
ret = aci_gatt_update_char_value(service_handle,
    dev_name_char_handle, 0,
                                 strlen(name), (uint8_t *)name);

if(ret){
  PRINTF("aci_gatt_update_char_value failed.\n");
  while(1);
}

      // REVISAR
ret = aci_gap_set_auth_requirement(MITM_PROTECTION_REQUIRED,
                                   OOB_AUTH_DATA_ABSENT,
                                   NULL,
                                   7,
                                   16,
                                   USE_FIXED_PIN_FOR_PAIRING,
                                   123456,
                                   BONDING);
if (ret == BLE_STATUS_SUCCESS) {
  PRINTF("BLE Stack Initialized.\n");
}

PRINTF("SERVER: BLE Stack Initialized\n");

      // REVISAR POR QUE ES NECESARIO
ret = Add_Acc_Service();

if(ret == BLE_STATUS_SUCCESS)
  PRINTF("Acc service added successfully.\n");
else
  PRINTF("Error while adding Acc service.\n");

ret = Add_Environmental_Sensor_Service();

if(ret == BLE_STATUS_SUCCESS)
  PRINTF("Environmental Sensor service added successfully.\n");
else
  PRINTF("Error while adding Environmental Sensor service.\n");
```

```
/* Set output power level */
ret = aci_hal_set_tx_power_level(1,4);

while(1)
{
              /*This method, when executed, calls HCI_Event_CB at
                   sensor_service.c
              with a BT packet read as an argument.
              This method starts a chain of processes to enable
                  continuous data sending through BLE*/
  HCI_Process(); // HCI = (Host Controller Interface)
  User_Process();

              if(MAX30100_readRegister(
                  MAX30100_REG_INTERRUPCION_STATUS)&
                  MAX30100_PULSO_LISTO){

                    // We request infrarred data
                    max30100_buffer[0] = MAX30100_readRegister(
                        MAX30100_FIFO_DATOS);
                    max30100_buffer[1] = MAX30100_readRegister(
                        MAX30100_FIFO_DATOS);
                    /* In HR mode, bytes 3 and 4 of FIFO are
                        returning zeros, due to previous
                        deactivation of
                     * red led. Then we must read 3 and 4 byte
                         in any case to increment read pointer
                         of FIFO and
                     * avoid to read wrong data */
                    max30100_buffer[2] = MAX30100_readRegister(
                        MAX30100_FIFO_DATOS);
                    max30100_buffer[3] = MAX30100_readRegister(
                        MAX30100_FIFO_DATOS);

                    // Relevant data is on the first two bytes
                    intensidad = max30100_buffer[0];
                    intensidad = max30100_buffer[1]|intensidad
                        <<8;


                    /* If there is something to read */
                    if(intensidad > 20000){

                          /* We calculate heart rate
                              obtaining the number of samples
                              between minimums
                           * and then, supposing a certain
                               value of miliseconds between
                               samples, the
                           * value of beats per minute is
                               obtained */
                          if(intensidad > intensidad_anterior
```

```
                                    && cambio == 0){
                                        //Calculamos hr en latidos
                                            por minuto
                                        hr_temp += 60000.0/(
                                            num_subida*260);
                                        cambio = 1;
                                        num_subida = 1;
                                        ind_media++;
                                    }else if(intensidad <
                                        intensidad_anterior && cambio ==
                                         1){
                                        cambio = 0;
                                        num_subida++;
                                    }else{
                                        num_subida++;
                                    }

                                    // We perform an average of
                                        measures to estimate the BPM
                                    if(ind_media == 100){
                                        hr = hr_temp/ind_media;
                                        hr_temp = 0;
                                        ind_media = 0;
                                    }

                                /* If not there is no heart rate to show */
                                }else{
                                        hr = 0;
                                }

                                intensidad_anterior=intensidad;

                        }

  }
}

/* Funcion necesaria para que se conecte el dispositivo. */
void User_Process()
{
  if(set_connectable){
    setConnectable();
    set_connectable = FALSE;
  }

}


float obtainTemperature(){

            /* Wait for the end of conversion */
    /*  Before starting a new conversion, the current state of the
        peripheral
                                must be checked; if it is busy you
```

```
                                    need to wait for the end of
                                    current
        conversion before starting a new one.*/
    HAL_ADC_PollForConversion(&AdcHandle, 10);

    /* Check if the continuous conversion of regular channel is
        finished */
    if ((HAL_ADC_GetState(&AdcHandle) & HAL_ADC_STATE_REG_EOC) ==
        HAL_ADC_STATE_REG_EOC)
    {
      /* Get the converted value of regular channel */
      uwADCxConvertedValue = HAL_ADC_GetValue(&AdcHandle);

                        /*We obtain the value of output voltage in
                            mV (VREF = 3.3V)*/
                        VTAO = uwADCxConvertedValue*3300/4095;

                        /* Temperature in celsius degrees is
                            obtained using a third order function to
                             be more accurate
                                 in the range between -10 and 110
                                     Celsius degrees */
                        temperature = a*(VTAO^3) + b*(VTAO^2) + c*
                            VTAO + d;
    }

                return temperature;
}

int obtainHR(){

                return hr;

}


/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
static void Error_Handler(void)
{
  /* Infinite loop */
  while(1)
  {
                BSP_LED_Toggle(LED2);
    HAL_Delay(1000);
  }
}


/****END OF FILE****/
```

- **sensor_service.c:** Este archivo es el que se encarga de crear los servicios y características BLE que vamos a usar en nuestro sistema. En este caso tendremos dos características: temperatura y frecuencia cardíaca. Además, este archivo se encarga de actualizar los valores de las características cuando se solicite una lectura por parte del dispositivo al que se encuentre conectada la tarjeta de evaluación. El código completo se encuentra a continuación:

```
/*
  * @file     sensor_service.c
  * @author   Jose Carlos Montesinos
  * @version V1.0.0
  * @date     22-May-2017
  * @brief    Add a service with its characteristics.
        *           Periodically update the characteristics' value.
  */

#include "sensor_service.h"
#include "main.h"

volatile int connected = FALSE;
volatile uint8_t set_connectable = 1;
volatile uint16_t connection_handle = 0;
volatile uint8_t notification_enabled = FALSE;

// Test variables to see how data is modified on each call
int sum1 = 0;
int sum2 = 0;

uint16_t sampleServHandle, TXCharHandle, RXCharHandle; //NECESARIO
    ??
uint16_t accServHandle;
uint16_t envSensServHandle, tempCharHandle, HRCharHandle,
    humidityCharHandle;

#define COPY_UUID_128(uuid_struct, uuid_15, uuid_14, uuid_13,
    uuid_12, uuid_11, uuid_10, uuid_9, uuid_8, uuid_7, uuid_6,
    uuid_5, uuid_4, uuid_3, uuid_2, uuid_1, uuid_0) \
do {\
    uuid_struct[0] = uuid_0; uuid_struct[1] = uuid_1; uuid_struct
        [2] = uuid_2; uuid_struct[3] = uuid_3; \
        uuid_struct[4] = uuid_4; uuid_struct[5] = uuid_5;
            uuid_struct[6] = uuid_6; uuid_struct[7] = uuid_7; \
            uuid_struct[8] = uuid_8; uuid_struct[9] = uuid_9;
                uuid_struct[10] = uuid_10; uuid_struct[11] = uuid_11
                ; \
                uuid_struct[12] = uuid_12; uuid_struct[13] =
                    uuid_13; uuid_struct[14] = uuid_14; uuid_struct
                    [15] = uuid_15; \
}while(0)

  #define COPY_ACC_SERVICE_UUID(uuid_struct)  COPY_UUID_128(
      uuid_struct,0x02,0x36,0x6e,0x80, 0xcf,0x3a, 0x11,0xe1, 0x9a,0
```

```
      xb4, 0x00,0x02,0xa5,0xd5,0xc5,0x1b)

  #define COPY_ENV_SENS_SERVICE_UUID(uuid_struct)  COPY_UUID_128(
      uuid_struct,0x42,0x82,0x1a,0x40, 0xe4,0x77, 0x11,0xe2, 0x82,0
      xd0, 0x00,0x02,0xa5,0xd5,0xc5,0x1b)
  #define COPY_TEMP_CHAR_UUID(uuid_struct)          COPY_UUID_128(
      uuid_struct,0xa3,0x2e,0x55,0x20, 0xe4,0x77, 0x11,0xe2, 0xa9,0
      xe3, 0x00,0x02,0xa5,0xd5,0xc5,0x1b)
  #define COPY_HR_CHAR_UUID(uuid_struct)          COPY_UUID_128(
      uuid_struct,0xcd,0x20,0xc4,0x80, 0xe4,0x8b, 0x11,0xe2, 0x84,0
      x0b, 0x00,0x02,0xa5,0xd5,0xc5,0x1b)


/* Store Value into a buffer in Little Endian Format */
#define STORE_LE_16(buf, val)    ( ((buf)[0] =  (uint8_t) (val)
    ) , \
                                   ((buf)[1] =  (uint8_t) (val>>8)
                                       ) )

/**
 * @brief  Add an accelerometer service using a vendor specific
     profile.
 *
 * @param  None
 * @retval tBleStatus Status
 */
tBleStatus Add_Acc_Service(void)
{
  tBleStatus ret;

  uint8_t uuid[16];

  COPY_ACC_SERVICE_UUID(uuid);
  ret = aci_gatt_add_serv(UUID_TYPE_128,  uuid, PRIMARY_SERVICE, 7,
                          &accServHandle);
  if (ret != BLE_STATUS_SUCCESS) goto fail;

fail:
  PRINTF("Error while adding ACC service.\n");
  return BLE_STATUS_ERROR ;

}


/**
 * @brief  Add the Environmental Sensor service.
 *
 * @param  None
 * @retval Status
 */
tBleStatus Add_Environmental_Sensor_Service(void)
{
  tBleStatus ret;
  uint8_t uuid[16];
```

```
uint16_t uuid16;
charactFormat charFormat;
uint16_t descHandle;

COPY_ENV_SENS_SERVICE_UUID(uuid);
ret = aci_gatt_add_serv(UUID_TYPE_128,  uuid, PRIMARY_SERVICE,
    10,
                        &envSensServHandle);
if (ret != BLE_STATUS_SUCCESS) goto fail;

/* Temperature Characteristic */
COPY_TEMP_CHAR_UUID(uuid);
ret =  aci_gatt_add_char(envSensServHandle, UUID_TYPE_128, uuid,
    2,
                         CHAR_PROP_READ, ATTR_PERMISSION_NONE,
                         GATT_NOTIFY_READ_REQ_AND_WAIT_FOR_APPL_RESP
                             ,
                         16, 0, &tempCharHandle);
if (ret != BLE_STATUS_SUCCESS) goto fail;

charFormat.format = FORMAT_SINT16;
charFormat.exp = -1;
charFormat.unit = UNIT_TEMP_CELSIUS;
charFormat.name_space = 0;
charFormat.desc = 0;

uuid16 = CHAR_FORMAT_DESC_UUID;

ret = aci_gatt_add_char_desc(envSensServHandle,
                             tempCharHandle,
                             UUID_TYPE_16,
                             (uint8_t *)&uuid16,
                             7,
                             7,
                             (void *)&charFormat,
                             ATTR_PERMISSION_NONE,
                             ATTR_ACCESS_READ_ONLY,
                             0,
                             16,
                             FALSE,
                             &descHandle);
if (ret != BLE_STATUS_SUCCESS) goto fail;

/* Heart Rate Characteristic */
if(1){ //FIXME
  COPY_HR_CHAR_UUID(uuid);
  ret =  aci_gatt_add_char(envSensServHandle, UUID_TYPE_128, uuid
      , 3,
                           CHAR_PROP_READ, ATTR_PERMISSION_NONE,
                           GATT_NOTIFY_READ_REQ_AND_WAIT_FOR_APPL_RESP
                               ,
                           16, 0, &HRCharHandle);
  if (ret != BLE_STATUS_SUCCESS) goto fail;
```

```
    charFormat.format = FORMAT_UINT16;
    charFormat.exp = -5;
    charFormat.unit = UNIT_HRURE_BAR; // Edit for HR units
    charFormat.name_space = 0;
    charFormat.desc = 0;

    uuid16 = CHAR_FORMAT_DESC_UUID;

    ret = aci_gatt_add_char_desc(envSensServHandle,
                                 HRCharHandle,
                                 UUID_TYPE_16,
                                 (uint8_t *)&uuid16,
                                 7,
                                 7,
                                 (void *)&charFormat,
                                 ATTR_PERMISSION_NONE,
                                 ATTR_ACCESS_READ_ONLY,
                                 0,
                                 16,
                                 FALSE,
                                 &descHandle);
    if (ret != BLE_STATUS_SUCCESS) goto fail;
  }

   return BLE_STATUS_SUCCESS;

fail:
  PRINTF("Error while adding ENV_SENS service.\n");
  return BLE_STATUS_ERROR ;

}

/**
 * @brief  Update temperature characteristic value.
 * @param  Temperature in tenths of degree
 * @retval Status
 */
tBleStatus Temp_Update(int16_t temp)
{
  tBleStatus ret;

  ret = aci_gatt_update_char_value(envSensServHandle,
      tempCharHandle, 0, 2,
                                   (uint8_t*)&temp);

  if (ret != BLE_STATUS_SUCCESS){
    PRINTF("Error while updating TEMP characteristic.\n") ;
    return BLE_STATUS_ERROR ;
  }
  return BLE_STATUS_SUCCESS;

}

/**
```

```
 * @brief  Update HR characteristic value.
 * @param  int32_t HR in BPM
 * @retval tBleStatus Status
 */
tBleStatus HR_Update(int32_t HR)
{
  tBleStatus ret;

  ret = aci_gatt_update_char_value(envSensServHandle, HRCharHandle,
      0, 3,
                                   (uint8_t*)&HR);

  if (ret != BLE_STATUS_SUCCESS){
    PRINTF("Error while updating TEMP characteristic.\n") ;
    return BLE_STATUS_ERROR ;
  }
  return BLE_STATUS_SUCCESS;

}

/**
 * @brief  Puts the device in connectable mode.
 *         If you want to specify a UUID list in the advertising
    data, those data can
 *         be specified as a parameter in aci_gap_set_discoverable
    ().
 *         For manufacture data, aci_gap_update_adv_data must be
    called.
 * @param  None
 * @retval None
 */
/* Ex.:
 *
 *  tBleStatus ret;
 *  const char local_name[] = {AD_TYPE_COMPLETE_LOCAL_NAME,'B','l
    ','u','e','N','R','G'};
 *  const uint8_t serviceUUIDList[] = {AD_TYPE_16_BIT_SERV_UUID,0
    x34,0x12};
 *  const uint8_t manuf_data[] = {4,
    AD_TYPE_MANUFACTURER_SPECIFIC_DATA, 0x05, 0x02, 0x01};
 *
 *  ret = aci_gap_set_discoverable(ADV_IND, 0, 0, PUBLIC_ADDR,
    NO_WHITE_LIST_USE,
 *                                 8, local_name, 3,
    serviceUUIDList, 0, 0);
 *  ret = aci_gap_update_adv_data(5, manuf_data);
 *
 */
void setConnectable(void)
{
  tBleStatus ret;

const char local_name[] = {AD_TYPE_COMPLETE_LOCAL_NAME,'S','T','M',
    '3','2',' ','B','o','a','r','d'};
```

```
  /* disable scan response */
  hci_le_set_scan_resp_data(0,NULL);
  PRINTF("General Discoverable Mode.\n");

  ret = aci_gap_set_discoverable(ADV_IND, 0, 0, PUBLIC_ADDR,
      NO_WHITE_LIST_USE,
                                 sizeof(local_name), local_name, 0,
                                     NULL, 0, 0);
  if (ret != BLE_STATUS_SUCCESS) {
    PRINTF("Error while setting discoverable mode (%d)\n", ret);
  }
}

/**
 * @brief  This function is called when there is a LE Connection
     Complete event.
 * @param  uint8_t Address of peer device
 * @param  uint16_t Connection handle
 * @retval None
 */
void GAP_ConnectionComplete_CB(uint8_t addr[6], uint16_t handle)
{
  connected = TRUE;
  connection_handle = handle;

  PRINTF("Connected to device:");
  for(int i = 5; i > 0; i--){
    PRINTF("%02X-", addr[i]);
  }
  PRINTF("%02X\n", addr[0]);
}

/**
 * @brief  This function is called when the peer device gets
     disconnected.
 * @param  None
 * @retval None
 */
void GAP_DisconnectionComplete_CB(void)
{
  connected = FALSE;
  PRINTF("Disconnected\n");
  /* Make the device connectable again. */
  set_connectable = TRUE;
  notification_enabled = FALSE;
}

/**
 * @brief  Read request callback.
 * @param  uint16_t Handle of the attribute
 * @retval None
 */
void Read_Request_CB(uint16_t handle)
```

```
{

        if(handle == tempCharHandle + 1){
    int16_t data;
    //data = 270 + ((uint64_t)rand()*15)/RAND_MAX; //sensor
        emulation
                //data = 27 + sum1;
                data = obtainTemperature()*10;
    Temp_Update(data);
                //sum1++;
  }
  else if(handle == HRCharHandle + 1){
    int32_t data;
//     data = 100000 + ((uint64_t)rand()*1000)/RAND_MAX;
                //data = 98 + sum2;
                data = obtainHR();
    HR_Update(data);
                //sum2++;
  }

  //EXIT:
  if(connection_handle != 0)
    aci_gatt_allow_read(connection_handle);
}

/**
 * @brief  Callback processing the ACI events.
 * @note   Inside this function each event must be identified and
     correctly
 *         parsed.
 * @param  void* Pointer to the ACI packet
 * @retval None
 */
void HCI_Event_CB(void *pckt)
{
  hci_uart_pckt *hci_pckt = pckt;
  /* obtain event packet */
  hci_event_pckt *event_pckt = (hci_event_pckt*)hci_pckt->data;

  if(hci_pckt->type != HCI_EVENT_PKT)
    return;

  switch(event_pckt->evt){

  case EVT_DISCONN_COMPLETE:
    {
      GAP_DisconnectionComplete_CB();
    }
    break;

  case EVT_LE_META_EVENT:
    {
      evt_le_meta_event *evt = (void *)event_pckt->data;
```

```
      switch(evt->subevent){
      case EVT_LE_CONN_COMPLETE:
        {
          evt_le_connection_complete *cc = (void *)evt->data;
          GAP_ConnectionComplete_CB(cc->peer_bdaddr, cc->handle);
        }
        break;
      }
    }
    break;

  case EVT_VENDOR:
    {
      evt_blue_aci *blue_evt = (void*)event_pckt->data;
      switch(blue_evt->ecode){

      case EVT_BLUE_GATT_READ_PERMIT_REQ:
        {
          evt_gatt_read_permit_req *pr = (void*)blue_evt->data;
          Read_Request_CB(pr->attr_handle);
        }
        break;
      }
    }
    break;
  }
}

/****END OF FILE****/
```

- **main.h:** Este archivo se trata del fichero de cabecera de *main.c*. En él se definen constantes relativas al funcionamiento de la comunicación a través del puerto I2C, así como la estructura de algunas funciones que se utilizarán durante la ejecución de nuestro proyecto. El código completo se encuentra a continuación:

```
/* Define to prevent recursive inclusion */
#ifndef __MAIN_H
#define __MAIN_H

/* Includes */
#include "stm32l0xx_hal.h"
#include "stm32l0xx_nucleo.h"

/* Definition for I2Cx clock resources */
#define I2Cx                            I2C1
#define RCC_PERIPHCLK_I2Cx              RCC_PERIPHCLK_I2C1
#define RCC_I2CxCLKSOURCE_SYSCLK        RCC_I2C1CLKSOURCE_SYSCLK
#define I2Cx_CLK_ENABLE()               __HAL_RCC_I2C1_CLK_ENABLE()
#define I2Cx_SDA_GPIO_CLK_ENABLE()      __HAL_RCC_GPIOB_CLK_ENABLE
    ()
#define I2Cx_SCL_GPIO_CLK_ENABLE()      __HAL_RCC_GPIOB_CLK_ENABLE
    ()

#define I2Cx_FORCE_RESET()              __HAL_RCC_I2C1_FORCE_RESET
    ()
#define I2Cx_RELEASE_RESET()
    __HAL_RCC_I2C1_RELEASE_RESET()

/* Definition for I2Cx Pins */
#define I2Cx_SCL_PIN                    GPIO_PIN_8
#define I2Cx_SCL_GPIO_PORT              GPIOB
#define I2Cx_SDA_PIN                    GPIO_PIN_9
#define I2Cx_SDA_GPIO_PORT              GPIOB
#define I2Cx_SCL_SDA_AF                 GPIO_AF4_I2C1

/* Size of Transmission buffer */
#define TXBUFFERSIZE                      (COUNTOF(aTxBuffer) - 1)
/* Size of Reception buffer */
#define RXBUFFERSIZE                      TXBUFFERSIZE

/* Exported macro */
#define COUNTOF(__BUFFER__)   (sizeof(__BUFFER__) / sizeof(*(
    __BUFFER__)))

float obtainTemperature(void);
int obtainHR(void);

#endif /* __MAIN_H */

/*****END OF FILE****/
```

Estos archivos también se explicaron anteriormente en las Secciones 3.4.2.3 y 3.4.4. No se menciona el código relativo a los archivos *mpl.c* ya que se encuentra descrito previamente en la Sección 3.4.2.3 y tampoco se mencionan los archivos de cabecera, ya que se considera que a nivel de comprensión del proyecto no es relevante.

Por último, se menciona a continuación, por si fuera de utilidad para el futuro desarrollador, el código que permite leer la temperatura del sensor Heart Rate Click. Se adjunta el código del archivo *main.c* a continuación:

```
/* Includes */
#include "main.h"
#include "mpl.h"

/* Private define */
/* Uncomment this line to use the board as master, if not it is used as
    slave */
//#define MASTER_BOARD
#define I2C_ADDRESS        0x30F

/* I2C TIMING Register define when I2C clock source is SYSCLK */
/* I2C TIMING is calculated in case of the I2C Clock source is the
    SYSCLK = 32 MHz */
//#define I2C_TIMING     0x10A13E56 /* 100 kHz with analog Filter ON,
    Rise Time 400ns, Fall Time 100ns */
#define I2C_TIMING      0x00B1112E /* 400 kHz with analog Filter ON,
    Rise Time 250ns, Fall Time 100ns */

#define INTERVALO_LECTURAS 1000
#define CANTIDAD_LECTURAS_MEDIA 10
#define PIN_INTERRUPCION 7 /* Usando una placa Arduino Leonardo */

uint8_t max30100_buffer[4]; /* Una operacion de lectura del MAX30100
    devuelve como maximo 4 bytes (dos enteros de 16 bits) */
enum bul {False=0,True}; // 1
enum bul interrupcion_activa = False;
//bool interrupcion_activa=false;
float temperatura_actual;
float temperatura_media;
float temperatura_total=0;
unsigned int cantidad_lecturas=1; /* Empieza en la lectura numero 1 e
    incrementa despues de calcular la media */
unsigned long cronometro;

/* I2C handler declaration */

I2C_HandleTypeDef I2cHandle;


/* Buffer used for transmission */
uint8_t aTxBuffer[] = " ****I2C_TwoBoards communication based on Polling
    ****  ****I2C_TwoBoards communication based on Polling****  ****
    I2C_TwoBoards communication based on Polling**** ";
```

```
/* Buffer used for reception */
uint8_t aRxBuffer[RXBUFFERSIZE]; // SON NECESARIOS LOS BUFFERS??

/* Private function prototypes */
void SystemClock_Config(void);
static void Error_Handler(void);

/* Private functions */

/**
  * @brief  Main program
  * @param  None
  * @retval None
  */
int main(void)
{
  HAL_Init();

  /* Configure the system clock to 32 MHz */
  SystemClock_Config();

  /* Configure LED2 */
  BSP_LED_Init(LED2);


  /* Configure the I2C peripheral */
  I2cHandle.Instance             = I2Cx;
  I2cHandle.Init.Timing          = I2C_TIMING;
  I2cHandle.Init.OwnAddress1     = I2C_ADDRESS;

//  I2cHandle.Init.AddressingMode  = I2C_ADDRESSINGMODE_10BIT;

        /* Muy importante ya que la direccion del dispositivo es de 7
           bits */
        I2cHandle.Init.AddressingMode  = I2C_ADDRESSINGMODE_7BIT;
  I2cHandle.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  I2cHandle.Init.OwnAddress2     = 0xFF;
  I2cHandle.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  I2cHandle.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  I2cHandle.Init.NoStretchMode   = I2C_NOSTRETCH_DISABLE;

  if(HAL_I2C_Init(&I2cHandle) != HAL_OK)
  {
    /* Initialization Error */
    Error_Handler();
  }

  /* Enable the Analog I2C Filter */
  //HAL_I2CEx_ConfigAnalogFilter(&I2cHandle,I2C_ANALOGFILTER_ENABLE); //
      NECESARIO??


// Habilitamos la interrupcion de lectura lista
  MAX30100_writeRegister(MAX30100_REG_INTERRUPCION,
```

```
      MAX30100_TEMPERATURA_LISTA);
  /* Infinite loop */
  while (1)
  {
                HAL_Delay(1000);
                // Solicitamos lectura de temperatura
                MAX30100_writeRegister(MAX30100_REG_CONF_MOD,
                    MAX30100_LECTURA_TEMPERATURA);
                HAL_Delay(1000); //Esperamos que se realice la lectura
                max30100_buffer[0] = MAX30100_readRegister(
                    MAX30100_TEMPERATURA_INT);
                HAL_Delay(100);
                max30100_buffer[1] = MAX30100_readRegister(
                    MAX30100_TEMPERATURA_FRAC);
                HAL_Delay(100);
                temperatura_actual=max30100_buffer[0]+(float)
                    max30100_buffer[1]/16.0; /* Parte entera mas
                    dieciseisavos de grado */
  }
}


/* FIN CODIGO USUARIO */


void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct ={0};
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};

  /* Enable Power Control clock */
  __HAL_RCC_PWR_CLK_ENABLE();

  /* The voltage scaling allows optimizing the power consumption when
      the device is
     clocked below the maximum system frequency, to update the voltage
         scaling value
     regarding system frequency refer to product datasheet.  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /* Disable Power Control clock */
  __HAL_RCC_PWR_CLK_DISABLE();

  /* Enable HSE Oscillator */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLSource    = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLState     = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLMUL       = RCC_PLL_MUL4;
  RCC_OscInitStruct.PLL.PLLDIV       = RCC_PLL_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct)!= HAL_OK)
  {
    /* Initialization Error */
    while(1);
```

```
  }

  /* Select PLL as system clock source and configure the HCLK, PCLK1 and
       PCLK2
     clocks dividers */
  RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK |
     RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1)!= HAL_OK)
  {
    /* Initialization Error */
    while(1);
  }
}

/**
  * @brief  I2C error callbacks.
  * @param  I2cHandle: I2C handle
  * @note   This example shows a simple way to report transfer error,
      and you can
  *         add your own implementation.
  * @retval None
  */
void HAL_I2C_ErrorCallback(I2C_HandleTypeDef *I2cHandle)
{
  /** Error_Handler() function is called when error occurs.
    * 1- When Slave don't acknowledge it's address, Master restarts
        communication.
    * 2- When Master don't acknowledge the last data transferred, Slave
        don't care in this example.
    */
  if (HAL_I2C_GetError(I2cHandle) != HAL_I2C_ERROR_AF)
  {
    Error_Handler();
  }
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* Error if LED2 is slowly blinking (1 sec. period) */
  while(1)
  {
    BSP_LED_Toggle(LED2);
    HAL_Delay(1000);
  }
}
```

```
#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line
     number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and
     line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
         line) */

  /* Infinite loop */
  while (1)
  {
  }
}
#endif



/*****END OF FILE****/
```

Código de la aplicación Android

- **ScanActivity.java:** En este archivo se encuentra el código necesario para comprobar si el Smartphone del usuario es compatible con Bluetooth Low Energy y para escanear en busca de dispositivos con los que emparejarse. El código completo se encuentra a continuación:

```
        package com.example.jose.tfm_app;

import android.app.Activity;
import android.app.ListActivity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothManager;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.Handler;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;


//Activity to scan BLE devices and display them.
public class ScanActivity extends ListActivity {
    private LeDeviceListAdapter mLeDeviceListAdapter;
    private BluetoothAdapter mBluetoothAdapter;
    private boolean mScanning;
    private Handler mHandler;

    private static final int REQUEST_ENABLE_BT = 1;
    // Stops scanning after 10 seconds.
    private static final long SCAN_PERIOD = 10000;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getActionBar().setTitle(R.string.title_main_activity);
        mHandler = new Handler();

        // It is determined if BLE is supported on the device. If
            not the app ends.
        if (!getPackageManager().hasSystemFeature(PackageManager.
            FEATURE_BLUETOOTH_LE)) {
```

```
            Toast.makeText(this, R.string.ble_not_supported, Toast.
                LENGTH_SHORT).show();
            finish();
        }

        // Initializes a Bluetooth adapter.
        final BluetoothManager bluetoothManager =
                (BluetoothManager) getSystemService(Context.
                    BLUETOOTH_SERVICE);
        mBluetoothAdapter = bluetoothManager.getAdapter();

        // Checks if Bluetooth is supported on the device. If not
            the app ends.
        if (mBluetoothAdapter == null) {
            Toast.makeText(this, R.string.
                error_bluetooth_not_supported, Toast.LENGTH_SHORT).
                show();
            finish();
            return;
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        if (!mScanning) {
            menu.findItem(R.id.menu_stop).setVisible(false);
            menu.findItem(R.id.menu_scan).setVisible(true);
            menu.findItem(R.id.menu_refresh).setActionView(null);
        } else {
            menu.findItem(R.id.menu_stop).setVisible(true);
            menu.findItem(R.id.menu_scan).setVisible(false);
            menu.findItem(R.id.menu_refresh).setActionView(
                    R.layout.actionbar_progress);
        }
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_scan:
                mLeDeviceListAdapter.clear();
                scanLeDevice(true);
                break;
            case R.id.menu_stop:
                scanLeDevice(false);
                break;
        }
        return true;
    }

    // At init or resuming app, the list of available devices is
        filled.
```

```
@Override
protected void onResume() {
    super.onResume();

    // Ensures Bluetooth is enabled on the device.  If
        Bluetooth is not currently enabled,
    // fire an intent to display a dialog asking the user to
        grant permission to enable it.
    if (!mBluetoothAdapter.isEnabled()) {
        if (!mBluetoothAdapter.isEnabled()) {
            Intent enableBtIntent = new Intent(BluetoothAdapter
                .ACTION_REQUEST_ENABLE);
            startActivityForResult(enableBtIntent,
                REQUEST_ENABLE_BT);
        }
    }

    // Initializes list view adapter.
    mLeDeviceListAdapter = new LeDeviceListAdapter();
    setListAdapter(mLeDeviceListAdapter);
    scanLeDevice(true);
}

// If the user does not enable the bluetooth the app finishes.
@Override
protected void onActivityResult(int requestCode, int resultCode
    , Intent data) {
    // User chose not to enable Bluetooth.
    if (requestCode == REQUEST_ENABLE_BT && resultCode ==
        Activity.RESULT_CANCELED) {
        finish();
        return;
    }
    super.onActivityResult(requestCode, resultCode, data);
}

// If the app is paused then the scan is stopped and the list
    comes empty.
@Override
protected void onPause() {
    super.onPause();
    scanLeDevice(false);
    mLeDeviceListAdapter.clear();
}

// It starts DataShowActivity to connect with the selected
    device and show
// the characteristics of their services
@Override
protected void onListItemClick(ListView l, View v, int position
    , long id) {

    final BluetoothDevice device = mLeDeviceListAdapter.
        getDevice(position);
```

```
        if (device == null) return;
        final Intent intent = new Intent(this, DataShowActivity.
            class);
        intent.putExtra(DataShowActivity.EXTRAS_DEVICE_NAME, device
            .getName());
        intent.putExtra(DataShowActivity.EXTRAS_DEVICE_ADDRESS,
            device.getAddress());

        if (mScanning) {
            mBluetoothAdapter.stopLeScan(mLeScanCallback);
            mScanning = false;
        }

        startActivity(intent);

    }


    // This function have the functionality to start/stop scanning
        depending on the boolean enable
    private void scanLeDevice(final boolean enable) {
        if (enable) {
            // Stops scanning after a pre-defined scan period (10
                Secs default).
            // To preserve battery
            mHandler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    mScanning = false;
                    mBluetoothAdapter.stopLeScan(mLeScanCallback);
                    invalidateOptionsMenu();
                }
            }, SCAN_PERIOD);

            mScanning = true;
            mBluetoothAdapter.startLeScan(mLeScanCallback);
        } else {
            mScanning = false;
            mBluetoothAdapter.stopLeScan(mLeScanCallback);
        }
        invalidateOptionsMenu(); //Restarts Scan/Stop action bar
            menu
    }


    // Adapter for holding devices found through scanning.
    private class LeDeviceListAdapter extends BaseAdapter {
        private ArrayList<BluetoothDevice> mLeDevices;
        private LayoutInflater mInflator;

        public LeDeviceListAdapter() {
            super();
            mLeDevices = new ArrayList<BluetoothDevice>();
            mInflator = ScanActivity.this.getLayoutInflater();
```

```
        }

        public void addDevice(BluetoothDevice device) {
            if(!mLeDevices.contains(device)) {
                mLeDevices.add(device);
            }
        }

        public BluetoothDevice getDevice(int position) {
            return mLeDevices.get(position);
        }

        public void clear() {
            mLeDevices.clear();
        }

        @Override
        public int getCount() {
            return mLeDevices.size();
        }

        @Override
        public Object getItem(int i) {
            return mLeDevices.get(i);
        }

        @Override
        public long getItemId(int i) {
            return i;
        }

        @Override
        public View getView(int i, View view, ViewGroup viewGroup)
            {
            ViewHolder viewHolder;
            // General ListView optimization code.
            if (view == null) {
                view = mInflator.inflate(R.layout.activity_scan,
                    null);
                viewHolder = new ViewHolder();
                viewHolder.deviceAddress = (TextView) view.
                    findViewById(R.id.device_address);
                viewHolder.deviceName = (TextView) view.
                    findViewById(R.id.device_name);
                view.setTag(viewHolder);
            } else {
                viewHolder = (ViewHolder) view.getTag();
            }

            BluetoothDevice device = mLeDevices.get(i);
            final String deviceName = device.getName();
            if (deviceName != null && deviceName.length() > 0)
                viewHolder.deviceName.setText(deviceName);
            else
```

```
                viewHolder.deviceName.setText(R.string.
                    unknown_device);
            viewHolder.deviceAddress.setText(device.getAddress());

            return view;
        }
    }


    /** Device scan callback. is an implementation of the
        BluetoothAdapter.LeScanCallback,
     which is the interface used to deliver BLE scan results*/
    private BluetoothAdapter.LeScanCallback mLeScanCallback =
            new BluetoothAdapter.LeScanCallback() {

                @Override
                public void onLeScan(final BluetoothDevice device,
                    int rssi, byte[] scanRecord) {
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            mLeDeviceListAdapter.addDevice(device);
                            mLeDeviceListAdapter.
                                notifyDataSetChanged();
                        }
                    });
                }
            };


    static class ViewHolder {
        TextView deviceName;
        TextView deviceAddress;
    }
}
```

- **DataShowActivity.java:** Este archivo es el encargado de conectar con el dispositivo seleccionado en la actividad anterior, solicitar la lectura de las características que encuentre en sus servicios y mostrar su valor en la interfaz de la actividad. Además permite habilitar y deshabilitar el envío continuo de datos y conectar y desconectar del dispositivo. Cabe mencionar que muchas de las funciones que utiliza están definidas en el servicio *BLEService.java*. El código completo de este archivo se define a continuación:

```
        package com.example.jose.tfm_app;

import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattService;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
```

```
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.app.Activity;
import android.os.Handler;
import android.os.IBinder;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

public class DataShowActivity extends Activity {

    private final static String TAG = DataShowActivity.class.
        getSimpleName();

    public static final String EXTRAS_DEVICE_NAME = "DEVICE_NAME";
    public static final String EXTRAS_DEVICE_ADDRESS = "
        DEVICE_ADDRESS";

    private TextView mConnectionState;
    private TextView mTempField;
    private TextView mHumField;
    private String mDeviceName;
    private String mDeviceAddress;

    private boolean buttonFlag = false;

    private BLEService mBluetoothLeService;

    private boolean mConnected = false;

    private List<BluetoothGattService> gattServices;

    // UUIDs for test purposes. Must be modified with final data
    public static String TEST_SERVICE = "42821a40-e477-11e2-82d0
        -0002a5d5c51b";
    public static String TEST_TEMP = "a32e5520-e477-11e2-a9e3-0002
        a5d5c51b";
    public static String TEST_HUM = "cd20c480-e48b-11e2-840b-0002
        a5d5c51b";
    public final static UUID UUID_TEST_TEMP = UUID.fromString(
        TEST_TEMP);
    public final static UUID UUID_TEST_HUM = UUID.fromString(
        TEST_HUM);
    public final static UUID UUID_TEST_SERVICE = UUID.fromString(
```

```
        TEST_SERVICE);

    List<BluetoothGattCharacteristic> charList = new ArrayList<>();


    // Code to manage Service lifecycle.
    private final ServiceConnection mServiceConnection = new
        ServiceConnection() {

        @Override
        public void onServiceConnected(ComponentName componentName,
            IBinder service) {
            mBluetoothLeService = ((BLEService.LocalBinder) service
                ).getService();
            if (!mBluetoothLeService.initialize()) {
                Log.e(TAG, "Unable to initialize Bluetooth");
                finish();
            }
            // Automatically connects to the device upon successful
                start-up initialization.
            mBluetoothLeService.connect(mDeviceAddress);
        }

        @Override
        public void onServiceDisconnected(ComponentName
            componentName) {
            //mBluetoothLeService.disconnect();
            mBluetoothLeService = null;
        }
    };

    // Handles various events fired by the Service.
    // ACTION_GATT_CONNECTED: connected to a GATT server.
    // ACTION_GATT_DISCONNECTED: disconnected from a GATT server.
    // ACTION_GATT_SERVICES_DISCOVERED: discovered GATT services.
    // ACTION_DATA_AVAILABLE: received data from the device.  This
        can be a result of read
    //                        or notification operations.
    private final BroadcastReceiver mGattUpdateReceiver = new
        BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
            if (BLEService.ACTION_GATT_CONNECTED.equals(action)) {
                mConnected = true;
                updateConnectionState(R.string.connected);
                invalidateOptionsMenu();
            } else if (BLEService.ACTION_GATT_DISCONNECTED.equals(
                action)) {
                mConnected = false;
                updateConnectionState(R.string.disconnected);
                invalidateOptionsMenu();
                clearUI();
            } else if (BLEService.ACTION_GATT_SERVICES_DISCOVERED.
```

```
                equals(action)) {

                // Initial obtention of data when services are
                    discovered
                obtainData();

            } else if (BLEService.ACTION_DATA_AVAILABLE.equals(
                action)) {

                // If there is available data, it is displayed on
                    the UI
                displayData(intent.getStringExtra(BLEService.
                    EXTRA_ID), intent.getStringExtra(BLEService.
                    EXTRA_DATA));

                // A list is needed in order to ensure that all
                    characteristics are read sequentially
                // Otherwise there will be losses of data
                charList.remove(charList.get(charList.size() - 1));

                if (charList.size() > 0)
                    mBluetoothLeService.readCharacteristic(charList
                        .get(charList.size()-1));

            }
        }
    };

    private void clearUI() {

        mTempField.setText(R.string.no_data);
        mHumField.setText(R.string.no_data);

    }

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

        final Intent intent = getIntent();
        mDeviceName = intent.getStringExtra(EXTRAS_DEVICE_NAME);
        mDeviceAddress = intent.getStringExtra(
            EXTRAS_DEVICE_ADDRESS);

        // Sets up UI references.
        ((TextView) findViewById(R.id.device_address)).setText(
            mDeviceAddress);
        mConnectionState = (TextView) findViewById(R.id.
            connection_state);
        mTempField = (TextView) findViewById(R.id.temp_value);
        mHumField = (TextView) findViewById(R.id.hum_value);

        Button boton_inicio = (Button) findViewById(R.id.
```

```
            button_init);
        //Enables continuous acquisition of data
        boton_inicio.setOnClickListener(new View.OnClickListener()
            {
            @Override
            public void onClick(View v) {

                if(!buttonFlag && mConnected){
                    buttonFlag = true;
                    obtainContinuousData();
                }

            }
        });

        Button boton_fin = (Button) findViewById(R.id.button_stop);
        //Disables continuous acquisition of data
        boton_fin.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                if(buttonFlag && mConnected){
                    buttonFlag = false;
                }

            }
        });


        getActionBar().setTitle(mDeviceName);
        getActionBar().setDisplayHomeAsUpEnabled(true);

        Intent gattServiceIntent = new Intent(this, BLEService.
            class);
        bindService(gattServiceIntent, mServiceConnection,
            BIND_AUTO_CREATE);
    }

    @Override
    protected void onResume() {
        super.onResume();
        registerReceiver(mGattUpdateReceiver,
            makeGattUpdateIntentFilter());
        if (mBluetoothLeService != null) {
            final boolean result = mBluetoothLeService.connect(
                mDeviceAddress);
            Log.d(TAG, "Connect request result=" + result);
        }

    }

    /*Creates a thread to read data from the STM32 Board every
        second*/
    private void obtainContinuousData() {
```

```
        Thread streamLector = new Thread(){
            @Override
            public void run() {

                try{

                    while(buttonFlag == true){
                        obtainData();
                        Thread.sleep(1000);
                    }

                }catch (Exception e) {}
            }
        };

        streamLector.start();

    }


    /*This method is capable to read characteristics values from
        the STM32 Board sequentially
    * First the service wanted is located. Then the characteristics
         wanted are also obtained
    * Afterwards those characteristics are added to a list to be
        read sequentially to not to
    * lose data.
    * Finally, data is read one by one*/
    private void obtainData(){

        gattServices = mBluetoothLeService.getSupportedGattServices
            ();

        // Loops through available GATT Services.
        for (BluetoothGattService gattService : gattServices){

            // This code checks the service that is being used
            if(UUID_TEST_SERVICE.equals(gattService.getUuid())){

                charList.add(gattService.getCharacteristic(
                    UUID_TEST_TEMP));
                charList.add(gattService.getCharacteristic(
                    UUID_TEST_HUM));
            }

        }

        mBluetoothLeService.readCharacteristic(charList.get(
            charList.size()-1));

    }
```

```
@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(mGattUpdateReceiver);
}

@Override
protected void onDestroy() {
    super.onDestroy();
    unbindService(mServiceConnection);
    mBluetoothLeService = null;
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.data_show_menu, menu);
    if (mConnected) {
        menu.findItem(R.id.menu_connect).setVisible(false);
        menu.findItem(R.id.menu_disconnect).setVisible(true);
    } else {
        menu.findItem(R.id.menu_connect).setVisible(true);
        menu.findItem(R.id.menu_disconnect).setVisible(false);
    }
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch(item.getItemId()) {
        case R.id.menu_connect:
            mBluetoothLeService.connect(mDeviceAddress);
            return true;
        case R.id.menu_disconnect:
            buttonFlag = false;
            mBluetoothLeService.disconnect();
            return true;
        case android.R.id.home:
            onBackPressed();
            return true;
    }
    return super.onOptionsItemSelected(item);
}

private void updateConnectionState(final int resourceId) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mConnectionState.setText(resourceId);
        }
    });
}


/*Displays the data read on the UI. Data is distinguished by
```

```
        and ID*/
    private void displayData(String id, String data) {
        if (id != null && data != null) {
            if(id.equals("1")){
                mTempField.setText(data);
            }else{
                mHumField.setText(data);
            }

        }
    }



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

}
```

- **BLEService.java:** En este archivo, que se trata de un servicio Android en vez de una actividad, se definen múltiples funciones que van a hacer uso de la API Bluetooth Low Energy de Android para descubrimiento de dispositivos, lectura de servicios y características y otro tipo de funcionalidades. el código completo se encuentra a continuación:

```
        package com.example.jose.tfm_app;

import android.app.Service;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattDescriptor;
import android.bluetooth.BluetoothGattService;
import android.bluetooth.BluetoothManager;
import android.bluetooth.BluetoothProfile;
import android.content.Context;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;


public class BLEService extends Service {

    private final static String TAG = BLEService.class.
        getSimpleName();

    private BluetoothManager mBluetoothManager;
    private BluetoothAdapter mBluetoothAdapter;
    private String mBluetoothDeviceAddress;
    private BluetoothGatt mBluetoothGatt;
    private int mConnectionState = STATE_DISCONNECTED;

    private static final int STATE_DISCONNECTED = 0;
    private static final int STATE_CONNECTING = 1;
    private static final int STATE_CONNECTED = 2;

    public final static String ACTION_GATT_CONNECTED =
            "com.example.jose.tfm_app.ACTION_GATT_CONNECTED";
    public final static String ACTION_GATT_DISCONNECTED =
            "com.example.jose.tfm_app.ACTION_GATT_DISCONNECTED";
    public final static String ACTION_GATT_SERVICES_DISCOVERED =
            "com.example.jose.tfm_app.
                ACTION_GATT_SERVICES_DISCOVERED";
    public final static String ACTION_DATA_AVAILABLE =
            "com.example.jose.tfm_app.ACTION_DATA_AVAILABLE";

    //REVISAR
    public final static String EXTRA_ID =
            "com.example.jose.tfm_app.EXTRA_ID";
    public final static String EXTRA_DATA =
            "com.example.jose.tfm_app.EXTRA_DATA";

    public static String TEST_TEMP = "a32e5520-e477-11e2-a9e3-0002
        a5d5c51b";
    public static String TEST_HUM = "cd20c480-e48b-11e2-840b-0002
        a5d5c51b";
    public final static UUID UUID_TEST_TEMP = UUID.fromString(
        TEST_TEMP);
    public final static UUID UUID_TEST_HUM = UUID.fromString(
        TEST_HUM);



    // Implements callback methods for GATT events that the app
        cares about.  For example,
    // connection change and services discovered.
    private final BluetoothGattCallback mGattCallback = new
        BluetoothGattCallback() {
        @Override
        public void onConnectionStateChange(BluetoothGatt gatt, int
```

```
            status, int newState) {
        String intentAction;
        if (newState == BluetoothProfile.STATE_CONNECTED) {
            intentAction = ACTION_GATT_CONNECTED;
            mConnectionState = STATE_CONNECTED;
            broadcastUpdate(intentAction);
            Log.i(TAG, "Connected to GATT server.");
            // Attempts to discover services after successful
                connection.
            Log.i(TAG, "Attempting to start service discovery:"
                +
                    mBluetoothGatt.discoverServices());

        } else if (newState == BluetoothProfile.
            STATE_DISCONNECTED) {
            intentAction = ACTION_GATT_DISCONNECTED;
            mConnectionState = STATE_DISCONNECTED;
            Log.i(TAG, "Disconnected from GATT server.");
            broadcastUpdate(intentAction);
        }
    }

    @Override
    public void onServicesDiscovered(BluetoothGatt gatt, int
        status) {
        if (status == BluetoothGatt.GATT_SUCCESS) {
            broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED);
        } else {
            Log.w(TAG, "onServicesDiscovered received: " +
                status);
        }
    }

    @Override
    public void onCharacteristicRead(BluetoothGatt gatt,
                                     BluetoothGattCharacteristic
                                         characteristic,
                                     int status) {
        if (status == BluetoothGatt.GATT_SUCCESS) {
            broadcastUpdate(ACTION_DATA_AVAILABLE,
                characteristic);
        }

    }

    @Override
    public void onCharacteristicChanged(BluetoothGatt gatt,
                                        BluetoothGattCharacteristic
                                            characteristic) {

        broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
    }
};
```

```
private void broadcastUpdate(final String action) {
    final Intent intent = new Intent(action);
    sendBroadcast(intent);
}

/*Method that reads the formatted data received by the STM32
    Board
* It is then sent to DataShowActivity to be displayed on the UI
    */
private void broadcastUpdate(final String action,
                             final BluetoothGattCharacteristic
                                 characteristic) {
    final Intent intent = new Intent(action);

    int format = -1;

    // There are only two parameters for test purposes. Must be
        modified with final data
    if(UUID_TEST_TEMP.equals(characteristic.getUuid())){

        format = BluetoothGattCharacteristic.FORMAT_SINT16;
        float temp = characteristic.getIntValue(format, 0);
        temp = temp/10;
        intent.putExtra(EXTRA_ID, "1");
        intent.putExtra(EXTRA_DATA, String.valueOf(temp) + "\
            degreeC");


    }else if (UUID_TEST_HUM.equals(characteristic.getUuid())){

        format = BluetoothGattCharacteristic.FORMAT_UINT16;
        final int hum = characteristic.getIntValue(format, 0);
        intent.putExtra(EXTRA_ID, "2");
        intent.putExtra(EXTRA_DATA, String.valueOf(hum) + "lpm
            ");
    }

    sendBroadcast(intent);
}

public class LocalBinder extends Binder {
    BLEService getService() {
        return BLEService.this;
    }
}

@Override
public IBinder onBind(Intent intent) {
    return mBinder;
}

@Override
```

```
    public boolean onUnbind(Intent intent) {
        // After using a given device, you should make sure that
            BluetoothGatt.close() is called
        // such that resources are cleaned up properly.  In this
            particular example, close() is
        // invoked when the UI is disconnected from the Service.
        close();
        return super.onUnbind(intent);
    }

    private final IBinder mBinder = new LocalBinder();

    /**
     * Initializes a reference to the local Bluetooth adapter.
     *
     * @return Return true if the initialization is successful.
     */
    public boolean initialize() {
        // For API level 18 and above, get a reference to
            BluetoothAdapter through
        // BluetoothManager.
        if (mBluetoothManager == null) {
            mBluetoothManager = (BluetoothManager) getSystemService
                (Context.BLUETOOTH_SERVICE);
            if (mBluetoothManager == null) {
                Log.e(TAG, "Unable to initialize BluetoothManager."
                    );
                return false;
            }
        }

        mBluetoothAdapter = mBluetoothManager.getAdapter();
        if (mBluetoothAdapter == null) {
            Log.e(TAG, "Unable to obtain a BluetoothAdapter.");
            return false;
        }

        return true;
    }

    public boolean connect(final String address) {
        if (mBluetoothAdapter == null || address == null) {
            Log.w(TAG, "BluetoothAdapter not initialized or
                unspecified address.");
            return false;
        }

        // Previously connected device.  Try to reconnect.
        if (mBluetoothDeviceAddress != null && address.equals(
            mBluetoothDeviceAddress)
                && mBluetoothGatt != null) {
            Log.d(TAG, "Trying to use an existing mBluetoothGatt
                for connection.");
            if (mBluetoothGatt.connect()) {
```

```
            mConnectionState = STATE_CONNECTING;
            return true;
        } else {
            return false;
        }
    }

    final BluetoothDevice device = mBluetoothAdapter.
        getRemoteDevice(address);
    if (device == null) {
        Log.w(TAG, "Device not found.  Unable to connect.");
        return false;
    }
    // We want to directly connect to the device, so we are
        setting the autoConnect
    // parameter to false.
    mBluetoothGatt = device.connectGatt(this, false,
        mGattCallback);
    Log.d(TAG, "Trying to create a new connection.");
    mBluetoothDeviceAddress = address;
    mConnectionState = STATE_CONNECTING;
    return true;
}


/**
 * Disconnects an existing connection or cancel a pending
     connection. The disconnection result
 * is reported asynchronously through the
 * {@code BluetoothGattCallback#onConnectionStateChange(android
     .bluetooth.BluetoothGatt, int, int)}
 * callback.
 */
public void disconnect() {
    if (mBluetoothAdapter == null || mBluetoothGatt == null) {
        Log.w(TAG, "BluetoothAdapter not initialized");
        return;
    }
    mBluetoothGatt.disconnect();
}

/**
 * After using a given BLE device, the app must call this
     method to ensure resources are
 * released properly.
 */
public void close() {
    if (mBluetoothGatt == null) {
        return;
    }
    mBluetoothGatt.close();
    mBluetoothGatt = null;
}
```

```
    /**
     * Request a read on a given {@code BluetoothGattCharacteristic
         }. The read result is reported
     * asynchronously through the {@code BluetoothGattCallback#
         onCharacteristicRead(android.bluetooth.BluetoothGatt,
         android.bluetooth.BluetoothGattCharacteristic, int)}
     * callback.
     *
     * @param characteristic The characteristic to read from.
     */
    public void readCharacteristic(BluetoothGattCharacteristic
        characteristic) {
        if (mBluetoothAdapter == null || mBluetoothGatt == null) {
            Log.w(TAG, "BluetoothAdapter not initialized");
            return;
        }
        mBluetoothGatt.readCharacteristic(characteristic);
    }




    public List<BluetoothGattService> getSupportedGattServices() {
        if (mBluetoothGatt == null) return null;

        return mBluetoothGatt.getServices();
    }

}
```

Hay más explicaciones sobre el código Android en la Sección 3.3.4.3.

Apéndice II. Manual de uso del sistema

Este último apéndice, se trata de un manual de uso del sistema, es decir, en él se van a resumir inicialmente los elementos que componen el sistema y cómo interactúan entre ellos, para finalmente detallar un ejemplo de uso del sistema paso a paso, para comprender las distintas funcionalidades que se han implementado en el sistema desarrollado en este TFM. Se describen brevemente a continuación los elementos que componen el sistema:

- **Tarjeta de evaluación STM32L053R8:** Esta tarjeta va a hacer de eje central del sistema. Por un lado va a recoger los datos medidos por los sensores y por otro los va a transmitir a la aplicación Android a través de BLE. Más detalles de esta tarjeta pueden encontrarse en la Sección 3.2.
- **Módulo de expansión BLE X-NUCLEO-IDB05A1:** Este módulo de expansión va a ser el encargado de ampliar la funcionalidad de la tarjeta de evaluación, aportándole la posibilidad de transmitir y recibir datos a través de Bluetooth Low Energy. Va a conectarse a la tarjeta de evaluación haciendo uso de sus conectores tipo Arduino. Más información de este módulo puede encontrarse en la Sección 3.3.3.1.
- **Sensor de frecuencia cardíaca:** Este sensor va a medir la intensidad reflejada de sus leds rojo e infrarrojo y con esos datos podrán inferirse datos de frecuencia cardíaca y de SpO2. Tiene además un sensor de temperatura. Los datos medidos se enviarán a la tarjeta de evaluación haciendo uso del puerto I2C. Más detalles de este sensor pueden encontrarse en la Sección 3.4.2.
- **Sensor de temperatura:** Este sensor va a realizar medidas continuas de temperatura que va a enviar a través de su puerto de salida que estará conectado al ADC de la tarjeta de evaluación. Dispone de un enable para activar o desactivar su salida. Más detalles de este sensor pueden encontrarse en la Sección 3.4.3.
- **Aplicación Android:** Esta aplicación es la encargada de escanear en busca de dispositivos disponibles, en este caso la tarjeta de evaluación, y emparejarse con ellos, así como de leer sus servicios y características y mostrarlos en tiempo real a través de su interfaz. Más detalles sobre esta aplicación pueden encontrarse en la Sección 3.3.4.

En esencia, el sistema funciona de la siguiente manera: se obtienen las medidas de la temperatura y la frecuencia cardíaca del usuario utilizando los sensores, se envían a través de BLE a la aplicación Android y ésta las muestra en su interfaz al usuario en tiempo real y de forma continua. A continuación, se describe un ejemplo de uso de este sistema paso a paso para tener una mejor comprensión sobre cómo hacer funcionar al sistema y cuáles son las funcionalidades que ofrece.

Lo primero que se debe hacer es preparar el sistema para su uso, para ello hay que seguir los siguientes pasos:

- Conectar todo el hardware de la manera descrita en la Figura 4.10, colocando lo primero de todo el módulo de expansión BLE en los conectores tipo Arduino de la tarjeta de evaluación.
- Cargar el proyecto en la tarjeta de evaluación, usando el entorno de desarrollo *ARM Keil*, de la manera descrita anteriormente en el Apéndice I, o bien cargar el proyecto utilizando otro entorno de desarrollo o un archivo *Makefile*. Desde este momento se estarán leyendo los datos a través de los sensores y enviándose a través de BLE.
- Finalmente, se abre la aplicación Android para comenzar a hacer uso del sistema. En este punto se supone que ya se ha instalado la aplicación en el Smartphone del usuario.

Con el sistema preparado, se comienza a utilizar el sistema. Lo primero que aparece es, en caso de no tener el Bluetooth activado, una solicitud para activarlo como puede verse en la Figura 4.11. Si se elige la opción del no, la aplicación se cerrará ya que no se puede utilizar sin tener el Bluetooth activado.

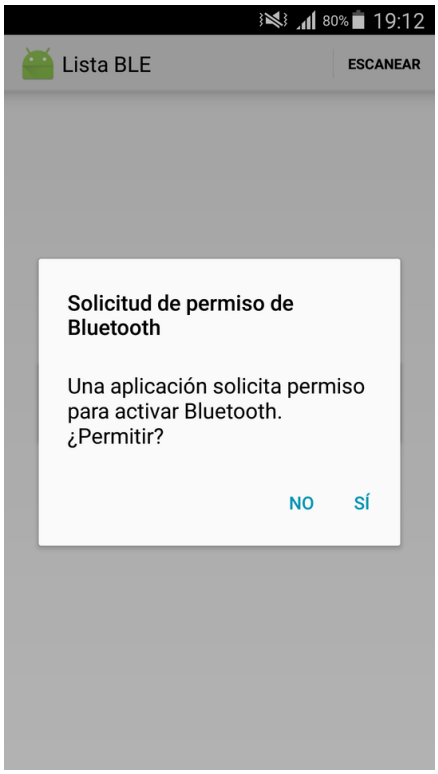

Figura 4.11: Solicitud de activación del Bluetooth

Si se pulsa sobre el sí, se activará el Bluetooth y se comenzará a escanear en busca de dispositivos disponibles para conexión, que se irán mostrando en la lista como puede verse en la Figura 3.15. En esta misma figura, puede verse que hay un botón

en la parte superior derecha para iniciar o detener el escaneo de dispositivos. Para establecer conexión con un dispositivo de los de la lista, hay que pulsar sobre él. Hay que tener en cuenta que no será posible conectar a otro dispositivo que no sea la tarjeta de evaluación que aparecerá en la lista con el nombre *STM32 Board*.

Una vez seleccionada la tarjeta de evaluación, se abrirá una nueva actividad, mostrándose una interfaz como la que aparece en la Figura 3.16. Puede verse ver que se muestra diversa información que se resume a continuación:

- **Dirección de dispositivo:** Dirección MAC del dispositivo con el que se ha establecido conexión.
- **Estado:** Informa de si el estado de conexión con el dispositivo es conectado o desconectado.
- **Temperatura:** Muestra el valor de la característica de temperatura.
- **Frecuencia cardíaca:** Muestra el valor de la característica de la frecuencia cardíaca. Si no se pone el dedo sobre el sensor MAX30100 se devolverá como valor 0 lpm hasta que se apoye un dedo y puedan tomarse medidas reales.

Además de la información que se muestra, existe la posibilidad de interactuar con esta interfaz a través de dos botones que se encuentran debajo de la información y un menú que se encuentra en la parte superior derecha de la actividad. Se resume a continuación la funcionalidad de cada uno de estos tres elementos:

- **Botón Obtener datos continuos:** Inicialmente, esta actividad hace una petición individual de lectura de los valores de las características al iniciarse. Este botón habilita la opción de actualizar los valores de las características de forma continua, para mostrar los datos que se están obteniendo de forma real. Se actualizan los valores cada segundo, aunque este intervalo puede ser fácilmente modificado en el código.
- **Botón Detener datos continuos:** Este botón deshabilita la opción de actualizar los valores de las características de forma continua.
- **Menú Conectar/desconectar:** Estos botones permiten desconectarnos y volver a conectarnos al dispositivo objetivo en cualquier momento.

Finalmente, mencionar que si se pulsa el botón de atrás del Smartphone del usuario o el botón de la parte superior izquierda, se volverá a la actividad *ScanActivity.java* donde podrán buscarse nuevos dispositivos y establecer conexión con ellos.