Universitat Oberta de Catalunya

uoc.edu

Sistema Low Cost de Comunicación Bluetooth Low Energy

Julia León Jiménez
Máster en Ingeniería de Telecomunicación
Área de Electrónica

Nombre Consultor: Aleix López Antón
Nombre Profesor Responsable de la Asignatura: Carlos Monzo Sánchez

Junio 2021

Licencias alternativas (elegir alguna de las siguientes y sustituir la de la página anterior)

A) Creative Commons:

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento 3.0 España de Creative Commons

B) GNU Free Documentation License (GNU FDL)

Copyright © AÑO TU-NOMBRE.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free

C) Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Sistema Low Cost de Comunicación Bluetooth Low Energy*
Nombre del autor:	*Julia León Jiménez*
Nombre del consultor/a:	*Aleix López Antón*
Nombre del PRA:	*Carlos Monzo Sánchez*
Fecha de entrega (mm/aaaa):	06/2021
Titulación:	*Máster en Ingeniería de Telecomunicación*
Área del Trabajo Final:	*Área de Electrónica*
Idioma del trabajo:	*Español*
Palabras clave	*Bluetooth Low Energy, Arduino, nRF52*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Debido al auge del IoT (Internet of Things), cada vez es más usual encontrar dispositivos (bombillas del hogar, riego, altavoces…) que se controlan con una aplicación instalada en nuestro Smartphone. Estos dispositivos se conectan por Bluetooth entre ellos y con el dispositivo controlador, en este caso, nuestro Smartphone. Con el fin de hacer que el consumo de este sistema de comunicación sea el mínimo posible, nació el Bluetooth Low Energy, protocolo con un bajo nivel de potencia y un alcance de comunicaciones más que aceptable para esta tecnología.

El presente proyecto se centra en diseñar un sistema de comunicación Bluetooth Low Energy haciendo uso del ya conocido microcontrolador Arduino, en una de sus variantes, y un integrado SoC que recibe datos por alguna de sus interfaces de comunicación y los envía por Bluetooth.

En primer lugar, se realizará un estudio de los sistemas de comunicación Bluetooth Low Energy desarrollados que se encuentran en el mercado, para posteriormente poder realizar una comparación entre el sistema diseñado en este proyecto y los que ya existían en el mercado. Tras este estudio, se analizarán y seleccionarán los dispositivos Arduino y SoC que se utilizarán para el diseño del sistema de comunicación.

En segundo lugar, se diseñará la estructura hardware, formada por Arduino + SoC + todo el material electrónico necesario para completar y poner en funcionamiento el sistema.

A continuación, se desarrollará una interfaz software que será la encargada de recoger los datos que el SoC recibe por uno de sus periféricos de comunicación y envía por Bluetooth.

Por último, se diseñará un prototipo sobre el que se realizarán las pruebas de funcionamiento para comprobar que el diseño es óptimo en coste, consumo y funcionamiento.

Para concluir con el proyecto, se realizará una comparativa entre el sistema desarrollado y los sistemas de comunicación BLE existentes. Se compararán aspectos como: consumo, coste, tamaño, alcance...

Abstract (in English, 250 words or less):

Due to the rise of the IoT (Internet of Things), it is increasingly common to find devices (household light bulbs, irrigation, speakers ...) that are controlled with an application installed on our Smartphone. These devices are connected by Bluetooth with each other and with the controller device, in this case, our Smartphone. In order to make the consumption of this communication system as low as possible, Bluetooth Low Energy was born, a protocol with an enough low level of power and communication range for this technology.

This project focuses on designing a Bluetooth Low Energy communication system using a variant of Arduino microcontroller and an integrated SoC that receives data through one of its communication interfaces and sends it through Bluetooth.

First of all, a study of the Bluetooth Low Energy communication systems developed that are on the market will be carried out, in order to later be able to

make a comparison between the system designed in this project and those that already existed on the market. After this study, the Arduino and SoC devices that will be used for the design of the communication system will be analyzed and selected.

Then, the hardware structure will be designed, consisting of Arduino + SoC + all the electronic components necessary to complete and put the system into operation.

Then, a software interface will be developed that will be responsible for collecting the data the SoC receives through one of its communication peripherals and sends through Bluetooth.

Finally, a prototype will be designed on which functional tests will be carried out to verify that the design is optimal in cost, consumption and operation.

To conclude the project, a comparison will be made between the developed system and the existing BLE communication systems. Aspects such as: consumption, cost, size, range will be compared...

Índice

Lista de figuras

Listado de Tablas

1. Introducción

1.1 Motivación

Actualmente, los dispositivos portátiles comerciales (smartphones, ordenadores, tablets…) tienen entre sus características la tecnología Bluetooth, lo que permite que el dispositivo se comunique con otros dispositivos sin hacer uso de internet o red y, por tanto, sin coste.

Por otro lado, la aparición del IoT (Internet of Things), que está en pleno auge, hace aún más interesante el poder diseñar un sistema de comunicación Bluetooth que consuma poca energía y, además, sea low cost. Por ejemplo, con el diseño de este sistema de comunicación se podrían gestionar y conectar dispositivos del hogar (luces, persianas, cámaras…) con un único dispositivo de control.

Por último, personalmente, trabajar con Arduino e integrados compatibles con este microcontrolador es un campo de especial interés y motivación.

1.2 Objetivos

- Conocer las características y propiedades del Bluetooth Low Energy
- Estudio de los SoC disponibles en el mercado que integren la parte RF
- Desarrollar una interfaz que permita recoger los datos que el SoC envía por Bluetooth
- Desarrollar un prototipo que integre el software y el hardware seleccionado
- Realizar pruebas sobre el prototipo para demostrar su correcto funcionamiento
- Conclusiones sobre el proyecto: ¿Por qué elegir este desarrollo?

1.3 Estructura del proyecto

El proyecto consta de 9 capítulos en los que se describe el desarrollo del sistema completo.

En el primer capítulo se realiza un resumen sobre las motivaciones que llevaron a realizar el proyecto y se establecen los objetivos que se deben cumplir durante el desarrollo del proyecto.

El segundo capítulo, por un lado, describe el Bluetooth Low Energy, sus características y aplicaciones. Por otro lado, presenta algunos desarrollos comerciales en los que se utiliza el estándar BLE, que servirán de comparación para las conclusiones del proyecto.

En el tercer y cuarto capítulo, se describen la estructura hardware y software del proyecto. En primer lugar y tras estudiar las opciones del mercado, se seleccionan los componentes que formarán el sistema. En segundo lugar, se presentan los programas y aplicaciones a utilizar durante el proyecto y se desarrolla el firmware encargado de recoger datos de un sensor y enviarlos por Bluetooth.

En el quinto capítulo, se realiza el montaje del prototipo y se realizan pruebas sobre el mismo, para comprobar el correcto funcionamiento del sistema.

Por último, se presentan las conclusiones obtenidas tras la finalización del proyecto y se realiza una comparación entre los sistemas del mercado y el desarrollado en el proyecto.

2. Estado del Arte

En el presente capítulo de la memoria se desarrollará el estado del arte del proyecto, entendiéndolo como los avances tecnológicos en sistemas de comunicación que se han llevado a cabo en cuanto a los sistemas de comunicaciones haciendo uso de Bluetooth Low Energy.

En primer lugar, se desarrollarán las características y prestaciones del Bluetooth Low Energy, con el fin de conocer este estándar que está presente en nuestro día a día.

A continuación, se presentarán una serie de proyectos que se han llevado a cabo en este ámbito.

2.1 Bluetooth Low Energy

El Bluetooth es un estándar de comunicación inalámbrica de voz y datos entre dispositivos a corta distancia. Este estándar que, como el protocolo Bluetooth, divide en paquetes la información transmitida, funciona en el rango de los 2400MHz a los 2483.5MHz mediante un enlace en la banda ISM de los 2.4GHz.

En nuestro día a día están cada vez más presentes los dispositivos y accesorios controlados y monitorizados por Bluetooth, pero ¿Es necesario tener un consumo de energía como el del Bluetooth en dispositivos que no se controlan o monitorizan a cada segundo? Así apareció el subconjunto del protocolo Bluetooth conocido como Bluetooth Low Energy, también conocido como BLE o Bluetooth Smart.

A partir de 2013, los dispositivos smartphone fabricados vendrían con el estándar Bluetooth 4.0 y sus sucesores, que incluyen tanto el Bluetooth Classic como el Bluetooth Low Energy o Bluetooth Smart.

2.1.1 Características BLE

El Bluetooth Low Energy es un subconjunto del estándar Bluetooth que se caracteriza principalmente por tener un consumo bajo.

Las principales características [1] de este subconjunto son:

- Radio de transferencia máximo teórico: Menos de 100 metros.
- Tasa de bit: La tasa de bit puede tener 4 valores tabulados, 125 kb/s, 500 kb/s, 1 Mb/s o 2 Mb/s.
- Latencia: La latencia en aplicaciones con BLE es alrededor de 6 ms.

- Consumo medio: El consumo depende la aplicación en la que se esté usando el BLE, puede variar entre un mínimo de 0.01 W y un máximo de 0.5 W.

- Corriente pico de consumo: Menos de 15 mA.

- Seguridad: Uno de los aspectos que más importancia tiene últimamente en el ámbito tecnológico es la seguridad. En este aspecto, el Bluetooth Low Energy puede parecer vulnerable, a pesar de que cuenta con una fase de autenticación y cifrado de 128 bits, esto no es suficiente para la fase de emparejamiento que deben pasar los dispositivos antes de establecer una conexión segura. En la fase del emparejamiento se intercambia información no encriptada, se establece la comunicación y se intercambian las claves. En este punto es donde se puede encontrar una vulnerabilidad. A continuación, para comprender este punto, se describen brevemente los 4 tipos de emparejamiento que existen:

    - *Just Works*: Es el tipo de emparejamiento menos seguro, ya que las claves que intercambian los dispositivos en esta fase siempre son ceros.

    - *Fuera de banda (OOB)*: En este tipo de emparejamiento, algunos datos se comparten por medio de otro protocolo inalámbrico más seguro que el Bluetooth Low Energy, por ejemplo, Zigbee. Este emparejamiento no es usual encontrarlo en dispositivos o sensores pequeños, ya que solo presentan un protocolo de comunicación inalámbrica (BLE).

    - *Claves de paso*: Es el tipo de emparejamiento más utilizado en los dispositivos cotidianos. En el dispositivo máster se establece una clave de 6 dígitos que el usuario debe ingresar en los periféricos que desee conectar.

    - *Comparación numérica*: Este tipo de emparejamiento es similar al anterior, pero, en primer lugar, los dispositivos comparan los valores aleatorios que compartieron en el inicio del emparejamiento y generan una nueva clave de 6 dígitos que debe verificar el usuario. Este emparejamiento solo está disponible en Bluetooth Low Energy Secure Connections.

- Pila de protocolos [2]:

    - *Capa física (PHY)*: Se encarga de realizar la modulación y demodulación de las señales analógicas y su posterior conversión a señales digitales.

    - *Capa de enlace (LL)*: Gestiona las características de las comunicaciones, controla los cambios y define los roles.

- *Interfaz de control (HCI)*: Protocolo estándar que contiene los eventos y comandos para la comunicación serie entre un host y un controlador.

- *L2CAP*: Esta capa permite dar acceso y soporte a los protocolos ATT y SMP.

- *Security Manager Protocol (SMP)*: Protocolo que permite generar las claves de seguridad a intercambiar entre dos dispositivos.

- *Attribute Protocol (ATT)*: Protocolo con arquitectura cliente-servidor que permite el intercambio de información.

- *Generic Access Profile (GAP):* Esta capa permite que un dispositivo sea visible para el resto de los dispositivos y establece las normas para la conexión.

- *Generic Attribute Profile (GATT):* Capa que determina la manera en la que intercambian información dos dispositivos.

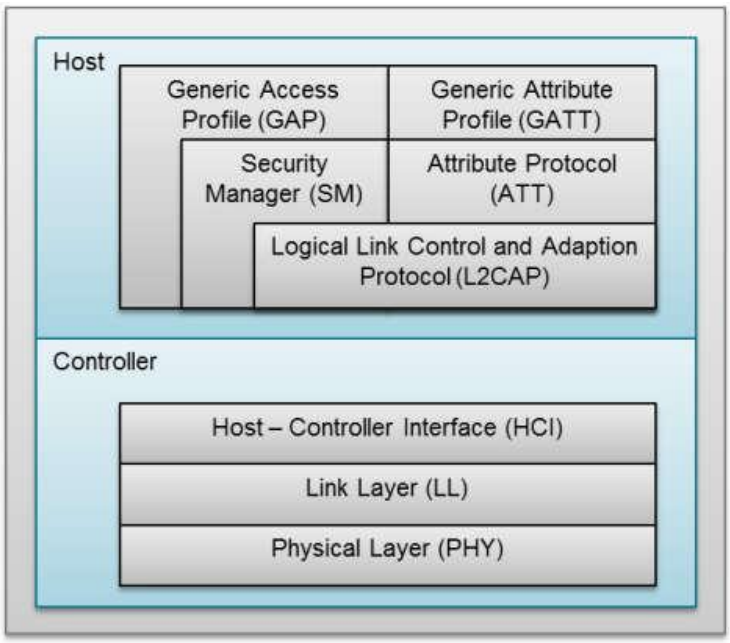

Ilustración 1: Pila de Protocolos BLE [2]

2.1.2 Versiones BLE

Desde que en 2013 Bluetooth incorporara tanto el Bluetooth Classic como el Bluetooth Low Energy en su protocolo, han creado varias versiones [5] del protocolo mejorando sus propiedades. Estas versiones se describen a continuación.

- Bluetooth 4.0: Primera versión del protocolo que integraba tanto el Bluetooth Classic como el Bluetooth Low Energy. Además, en esta

versión hubo un gran salto en el tamaño de los archivos que se podían transferir, pasando de 4 MB a 25 MB.

- Bluetooth 4.1: En esta segunda versión se mejoró la velocidad de transferencia. Por otro lado, se solucionó un problema de interferencia con el 4G.

- Bluetooth 4.2: Esta versión tuvo grandes ventajas. Por un lado, incorporó la posibilidad de utilizar Bluetooth para conectarse a internet a través del protocolo IPv6 y sólo permite la conexión y rastreo con dispositivos de confianza. Por otro lado, se mejora la velocidad de transferencia de datos y la capacidad de los paquetes que se envían.

- Bluetooth 5.0: En esta versión se duplica la velocidad (de 1 Mbps a 2 Mbps) y se aumenta el rango de transferencia perjudicando a la velocidad de transmisión, por lo que solo puede usarse una de estas mejoras a la vez. Además, se aumenta el tamaño del paquete a 255 bytes.

- Bluetooth 5.1: Además de mejorar la velocidad, permite la detección y la ubicación de otros dispositivos.

- Bluetooth 5.2 LE Audio: Incluye tres novedades importantes. La mejora del ATT conocida como Enhanced Attribute Protocol (EATT), el LE Power Control para optimizar la potencia de transmisión y el LE Isochronous Channels para enviar audio a varios dispositivos simultáneamente.

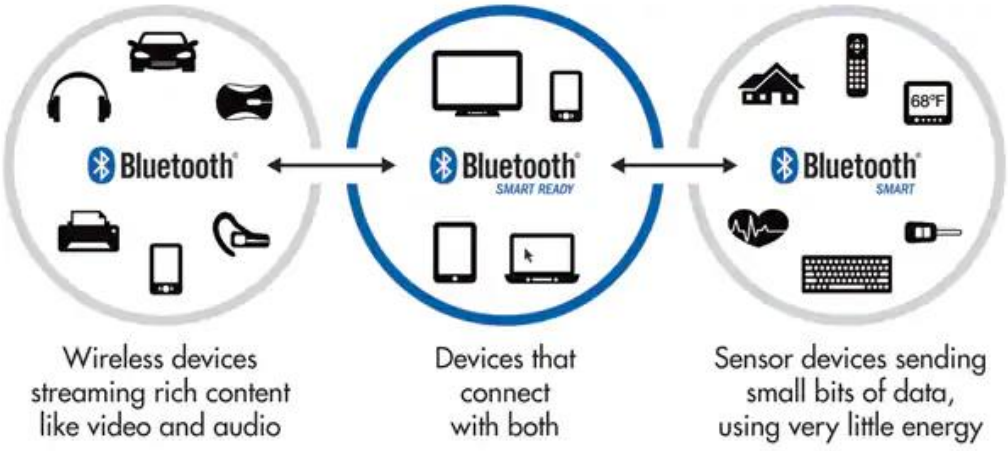

Ilustración 2: Versiones BLE [4]

2.1.3 Diferencias entre Bluetooth Classic y Bluetooth Low Energy

En el presente apartado se muestran las principales diferencias [1] entre las características del Bluetooth Classic y el Bluetooth Low Energy.

Tabla 1: Diferencias Bluetooth Classic y BLE

Características	Bluetooth Classic	BLE
Finalidad	Conexión entre dispositivos móviles, accesorios que requieren mayor energía	IoT, M2M, conexión con sensores y accesorios de baja energía
Tasa de transferencia	24Mb/s	1Mb/s
Consumo medio de energía	1W	0.01W-0.5W (depende del caso de uso)
Alcance máximo	100m	<100m
Tiempo de conexión	100ms	6ms
Modo de suspensión	-	Sleep Mode: Sólo se despierta cuando se recibe una solicitud de conexión

2.1.4 Aplicaciones BLE

Desde la inclusión del BLE en el protocolo Bluetooth son muchos los dispositivos y ámbitos en los que se ha hecho uso de este. A continuación, se muestran las aplicaciones más usuales.

- Domótica: Monitorización y control sobre sensores o actuadores colocados en el hogar. Por ejemplo, monitorización de la temperatura y posterior encendido/apagado del calefactor en función de esta.

- Sistemas de seguridad: Se utiliza para enviar mensajes de alarma cuando ocurren anomalías en el entorno.

- Localización Indoor: Técnica utilizada para conocer el posicionamiento de objetos o personas dentro de un edificio.

Ilustración 3: Aplicación BLE - Localización indoor [6]

2.1.5 ¿Por qué BLE?

Tras realizar un análisis sobre el BLE, parece intuitivo saber los motivos por los que se ha elegido este estándar para llevar a cabo las implementaciones anteriores:

- Bajo consumo de potencia: El BLE tiene unos requerimientos de potencia muy bajos.

- Durabilidad de la batería: Gracias al bajo consumo de potencia de esta tecnología, los dispositivos que lo utilizan tienen una mayor duración de la batería.

- Compatibilidad con iOS y Android: Las grandes plataformas como iOS, Android o Microsoft son compatibles con el BLE.

2.2 Sistemas desarrollados

En el mercado se encuentran numerosas aplicaciones que cuentan con un sistema de comunicación Bluetooth Low Energy, las más usuales son aplicaciones destinadas a salud, deporte, hogar o seguridad.

En este apartado, en primer lugar, se describe un módulo de comunicación BLE existente, similar al que se pretende desarrollar en el proyecto. Por otro lado, se presentan algunas de estas aplicaciones, que servirán de referencia para el sistema que se va a desarrollar en el presente proyecto.

2.2.1 Módulo Bluetooth de Baja Energía ISP1507

La empresa Insight SiP ha desarrollado este módulo haciendo uso del SoC nRF52832 de Nordic Semiconductor.

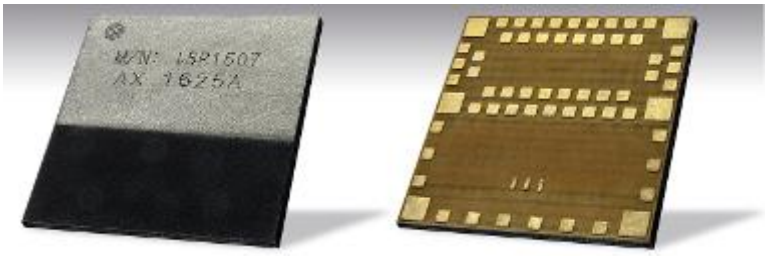

Ilustración 4: Módulo BLE ISP1507 [9]

El módulo ISP1507 es un módulo bluetooth de ultra bajo consumo que por su pequeño tamaño y su nivel de compactibilidad es muy útil para aplicaciones IoT médicas, fitness o del hogar. Su consumo permite una durabilidad de la batería de varios años.

Las principales características [8] del módulo se detallan a continuación:

- Voltaje de alimentación: 1.7 V - 3.6 V
- Transferencia de datos: 2 Mb/s
- Tipos de interfaces incluidas: I2C, SPI, UART, PDM
- Tamaño de la memoria: 64 kB SRAM y 512 kB FLASH
- Pines I/O: De 10 a 30 pines I/O de los cuales de 2 a 8 son ADCs

2.2.2 Wearables

Gracias al bajo consumo del BLE, la mayoría de wearables tienen una batería de larga duración a pesar de permanecer siempre conectados al *smartphone*. Además, el propio *smartphone* no consume apenas batería, punto a favor de esta tecnología, ya que la batería es una gran limitación de estos dispositivos móviles.

Además del bajo consumo, el BLE es la mejor opción para los wearables ya que los chips son de tamaño pequeño y el protocolo está integrado en los *smartphones*.

Ilustración 5: Wearables [4]

2.2.3 Sistema de monitorización cardíaca

Existen múltiples dispositivos de monitorización cardíaca que integran BLE entre sus características. Es el caso del dispositivo CARDIO UEES, publicado en la *Revista Ciencia UNEMI Vol. 9, Nº 20, Septiembre 2016* [7].

El diagrama de bloques de este dispositivo se muestra en la siguiente figura, donde el proyecto se divide en cinco etapas.

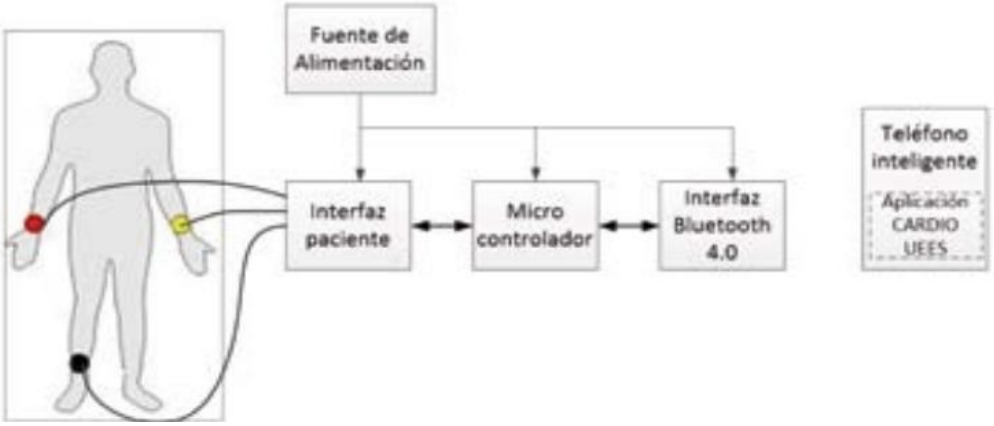

Ilustración 6: Sistema BLE - Sistema monitorización cardíaca

El módulo Bluetooth utilizado en este proyecto es el BL600 de LairdTech. Este módulo recibe a través de su interfaz UART los parámetros obtenidos de la interfaz del paciente y los envía a la aplicación móvil CARDIO UEES.

2.2.4 Control de iluminación en ambientes interiores

Gracias al Bluetooth Low Energy es posible controlar y monitorizar la iluminación del hogar mediante una aplicación móvil.

Un ejemplo de esta aplicación es la app de iGuzzini Smart Light, que permite controlar el encendido y apagado de la luminaria, los niveles de iluminación, color de la iluminación...

Ilustración 7: Sistema BLE – Control de iluminación

Por otro lado, existen dispositivos que son asistentes virtuales que incluyen un sistema de comunicación BLE que les permite comunicarse con accesorios o sensores para controlarlos o monitorizarlos.

3. Estructura Hardware

En el presente capítulo, se analizarán y seleccionarán los dispositivos hardware necesarios para poder llevar a cabo el desarrollo del Sistema de Comunicación BLE Low Cost.

3.1 Microcontrolador Arduino

Arduino es una plataforma open-source basada en software y hardware de fácil manejo. La compañía Arduino diseña placas de desarrollo hardware que, junto con otros dispositivos hardware como sensores o actuadores, permiten supervisar y controlar objetos.

Ilustración 8: Logotipo Arduino

Nació, en el año 2005, como herramienta didáctica en el Instituto IVREA (Italia). El objetivo del proyecto fue el de crear una herramienta sencilla de bajo costo útil para el desarrollo de aplicaciones digitales o prototipos, dirigida a estudiantes con poca experiencia en el ámbito de la programación y la electrónica. [11]

Las placas de Arduino han ido evolucionando desde placas simples de 8 bits hasta placas para aplicaciones IoT o impresión 3D. Actualmente, el número de placas comerciales de Arduino asciende a 26. Estas placas se programan en un lenguaje propio de Arduino haciendo uso del software IDE Arduino.

3.1.1 Características de Arduino

Las características de las placas Arduino son muy importantes a la hora de elegir la adecuada para un determinado proyecto, ya que una mala elección condiciona el desarrollo completo del mismo.

- Número de pines: En función del tipo de proyecto a realizar, es posible realizar un descarte de las placas de Arduino según tengan un gran número de pines o un número menor de pines de los requeridos.

- Tipo de microcontrolador: En las placas oficiales de Arduino se diferencian dos tipos de microcontroladores: ATMega AVR 8 bits o 32 bits y ARM de 32 bits.

- Memoria RAM: Está ligada al tipo de microcontrolador que incorpore la placa. Gestiona la rapidez con la que se procesan los datos.

- Memoria FLASH: Está ligada al tipo de microcontrolador que incorpore la placa. Según la complejidad del código que se va a programar para el funcionamiento del proyecto, es necesario que la placa tenga mayor almacenamiento para albergar dicho código.

3.1.2 Elección del modelo Arduino a utilizar

Con el fin de seleccionar la placa de Arduino que más se adecue a la funcionalidad del proyecto, se van a analizar las características [10] de tres de las placas más utilizadas en proyectos de ámbito BLE/IoT.

3.1.2.1 Arduino UNO Revisión 3

La placa Arduino UNO es la más comercializada de todas las placas desarrolladas por la compañía. A pesar de las limitaciones de memoria que presenta, es utilizada en gran cantidad de proyectos.

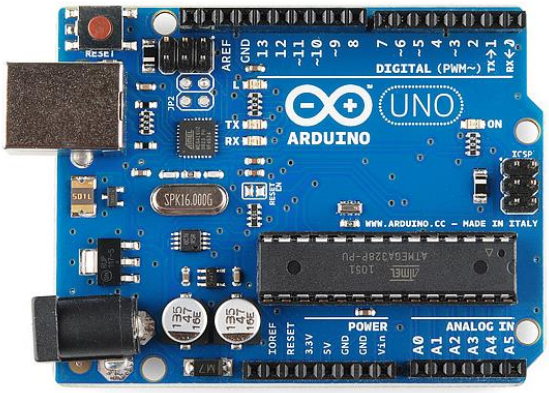

Ilustración 9: Arduino UNO

Tabla 2: Características Arduino UNO

Microcontrolador	ATmega328P
Alimentación (V)	5V
Pines digitales	14 (6 PWM)
Pines analógicos	6 ($I_{máx}$ = 40mA)
Memoria SRAM	2kB
Memoria Flash	32kB
Memoria EEPROM	1kB

3.1.2.2 Arduino MEGA Revisión 3

El modelo superior a Arduino UNO es el Arduino MEGA, ya que cuenta con mayor memoria, mayor número de pines tanto digitales como analógicos y un procesador de mejor calidad.

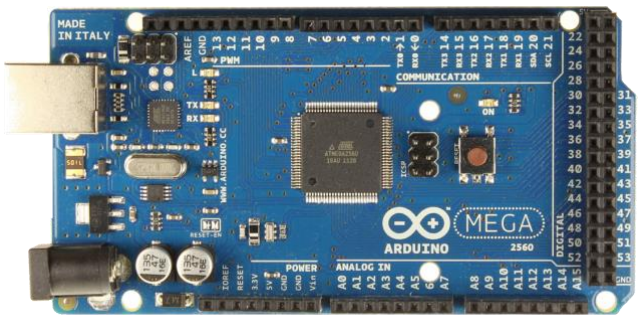

Ilustración 10: Arduino MEGA

Tabla 3: Características Arduino MEGA

Microchip	ATmega2560
Alimentación (V)	5V
Pines digitales	54 (15 PWM)
Pines analógicos	16 ($I_{máx}$ = 40mA)
Memoria SRAM	8kB
Memoria Flash	256kB
Memoria EEPROM	4kB

3.1.2.3 Arduino Nano

La placa Arduino Nano es la versión compacta creada por la compañía Arduino. En lugar de tener conector USB estándar, posee un conector mini-USB.

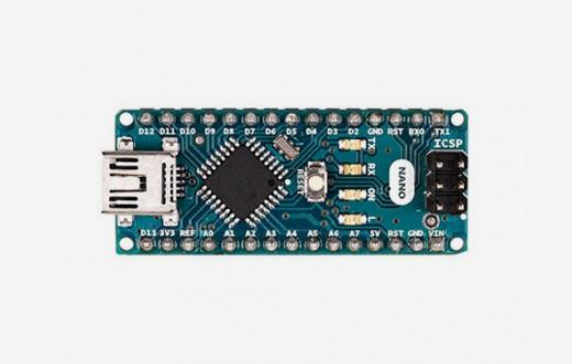

Ilustración 11: Arduino NANO

Tabla 4: Características Arduino NANO

Microchip	ATmega328
Alimentación (V)	5V
Pines digitales	22 (6 PWM)
Pines analógicos	8 ($I_{máx}$ = 40mA)
Memoria SRAM	2kB
Memoria Flash	32kB
Memoria EEPROM	1kB

Debido a sus prestaciones, el número de convertidores ADC y la calidad del procesador, la placa Arduino elegida para el desarrollo del proyecto será la placa Arduino MEGA Revisión 3.

3.2 SoC (System on Chip)

El componente principal del Sistema de Comunicación BLE será un sistema en un chip (SoC), que contiene múltiples componentes junto a un chip integrado conformando un sistema completo.

3.2.1 Elección del integrado a utilizar

Con el fin de seleccionar el SoC más adeacuado para el sistema a desarrollar, a continuación, se describen y analizan los *System on Chip* más utilizados para Bluetooth Low Energy.

3.2.1.1 Nordic Semiconductor nRF52 DK

El nRF52 DK [12] es un kit de desarrollo para aplicaciones de Bluetooth Low Energy haciendo uso del SoC nRF52832, aunque también se pueden realizar

desarrollos con el SoC nRF52810, al ser éste una variante con menos RAM, FLASH y recursos periféricos que el nRF52832.

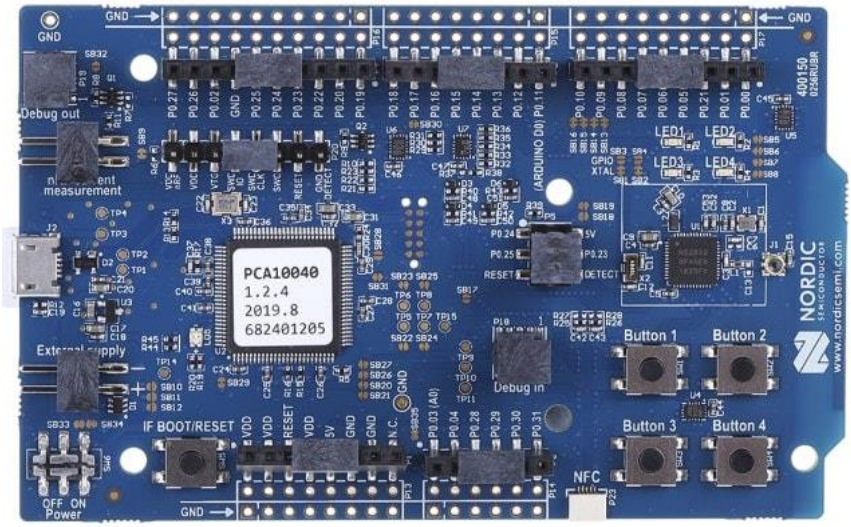

Ilustración 12: nRF52 DK

El SoC nRF52832, por un lado, integra el protocolo de radio de 2.4GHz, compatible con BLE, con el microprocesador ARM Cortex-M4F y, por otro lado, es un microcontrolador con las siguientes características:

Tabla 5: Características SoC nRF52832

Pines I/O	32
Interfaces	SPI, I2C, UART, PWM, ADC
Memoria RAM	64kB
Memoria FLASH	512kB
Protocolos soportados	BLE, NFC, ANT

En la siguiente imagen, *Ilustración 13*, se muestra el pinout del kit de desarrollo nRF52 DK, cuyas características más relevantes se exponen a continuación:

- Compatibilidad con el estándar y los módulos Arduino
- Posibilidad de depuración con el depurador J-Link SEGGER
- Antena NFC integrada
- Conector para medidas RF
- Puerto COM virtual por interfaz MCU

- 4 LEDs y 4 botones para interacción del usuario, conectados a pines I/O dedicados del kit de desarrollo

- La alimentación de esta placa puede ser:

    - USB: Los 5V del USB se regulan a 3.3V gracias a un regulador de voltaje integrado.

    - Fuente externa: 1.8V - 3.6V

    - Pila CR2032

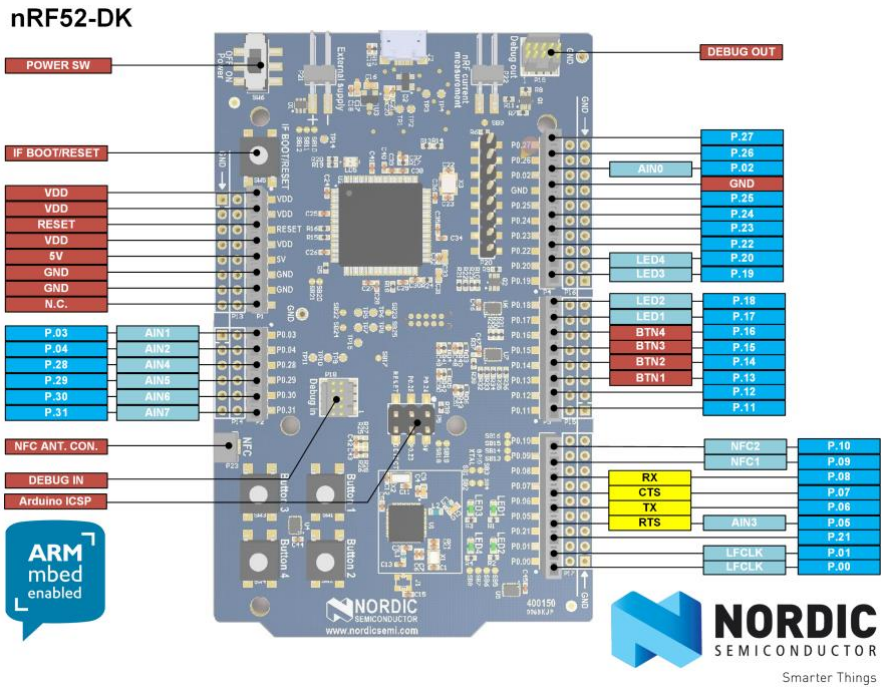

Ilustración 13: Pinout nRF52

En la página del fabricante se proporciona el siguiente diagrama de bloques, donde se muestra la conexión entre los distintos componentes y bloques que conforman el kit de desarrollo:

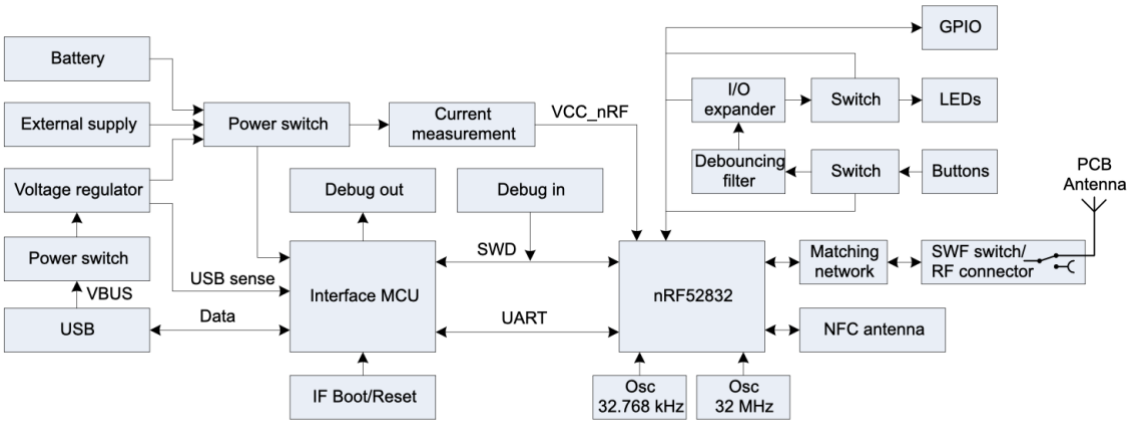

Ilustración 14: Arquitectura nRF52 DK

Para establecer comunicación con los dispositivos BLE, Nordic Semiconductor cuenta con dos aplicaciones móviles muy útiles y que servirán de apoyo en el desarrollo del proyecto, *nRF Toolbox* y *nRF Connect for Mobile*. Estas aplicaciones permiten configurar cualquier dispositivo Bluetooth de baja energía al que pueda conectarse el Smartphone que posee la aplicación y registrar eventos.

En cuanto al costo de este kit de desarrollo, el precio es de unos 40€ si se realiza la compra en un distribuidor oficial que lo entrega en un corto plazo (4-7 días).

El SoC estudiado es utilizado en múltiples ámbitos dadas sus características, como son: Wearables, domótica, periféricos para smartphones y ordenadores…

3.2.1.2 STMicroelectronics BlueNRG-2

El BlueNRG-2 es un sistema en chip (SoC) monomodo Bluetooth Low Energy de muy bajo consumo, compatible con el estándar Bluetooth 5.

Ilustración 15: BlueNRG-2 DK

Las características del SoC [13] son las siguientes:

Tabla 6: Características SoC BlueNRG-2

Alimentación	1.7V – 3.6V
Pines I/O	14, 15 o 26
Interfaces	SPI, I2C, UART, ADC
Memoria RAM	24kB
Memoria FLASH	256kB
Protocolos soportados	BLE

En la página del fabricante se proporciona el siguiente diagrama de bloques, donde se muestra la conexión entre los distintos componentes y bloques que conforman el SoC BlueNRG-2:

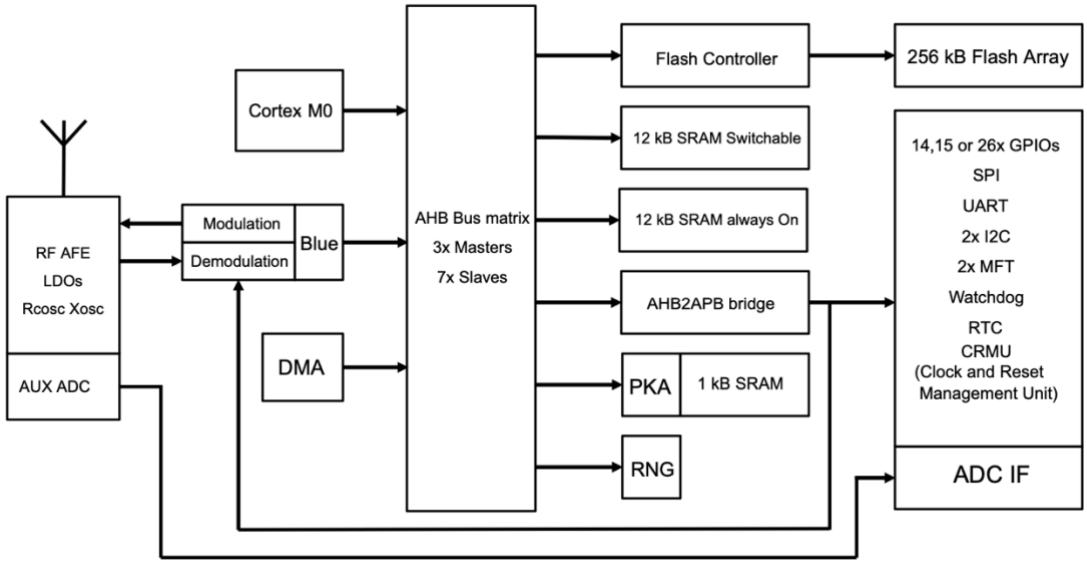

Ilustración 16: Arquitectura BlueNRG-2

El precio medio del kit de desarrollo es de 60€, si se pide en un distribuidor que lo entrega en un corto plazo (4-7 días).

Este SoC tiene múltiples aplicaciones en una de las tecnologías en auge actualmente, el Iot: Wearables, seguridad, control remoto, domótica, periféricos…

Ilustración 17: Ejemplo de aplicación BlueNRG-2

Teniendo en cuenta las características descritas de los dos SoC anteriores y su coste, el sistema en chip seleccionado para el desarrollo del proyecto es el nRF52832, ya que presenta mejores características en cuanto a memoria y soporte, y su precio es menor.

3.2.2 Hardware necesario para el integrado

Para poder completar el desarrollo hardware del sistema, es necesario contar con otros dos tipos de dispositivos: Uno o varios sensores, que recojan los datos que se van a enviar por Bluetooth por parte del SoC, y un dispositivo móvil que reciba estos datos.

El sensor puede variar en función de la aplicación en la que se utilice el sistema. En este caso, se han usado como ejemplo dos sondas de temperatura.

3.2.2.1 Sensor de Temperatura DS18B20

El dispositivo sensor de temperatura seleccionado es uno de los más utilizados en proyectos relacionados con Arduino e IoT donde se pretende medir la temperatura de líquidos, el sensor DS18B20. Una de las grandes ventajas de este sensor es que convierte internamente el dato analógico en digital.

Ilustración 18: Sensor de temperatura DS18B20

En la siguiente tabla se muestran las características [15] del sensor DS18B20:

Tabla 7: Características sensor DS18B20

Alimentación	3V – 5.5V
Precisión en la medida	±0.5ºC en el rango [-10, 85]ºC
Rango de medida	[-55, 125]ºC
Protocolo	1-wire
Precio	1€

La resolución del sensor es configurable de 9 a 12 bits. Por defecto, la resolución se establece en 12 bits, lo que se traduce en 0.0625ºC.

Para su correcto funcionamiento, es necesario conectar una resistencia pull-up de 4.7kΩ entre la tensión de alimentación y la línea de datos.

Este sensor destaca por utilizar el protocolo de comunicaciones 1-wire [16], que permite realizar la transmisión a través de un único bus en el que hay un maestro y varios esclavos. Gracias a esta estructura, es posible controlar varios dispositivos haciendo uso de un solo pin de la placa.

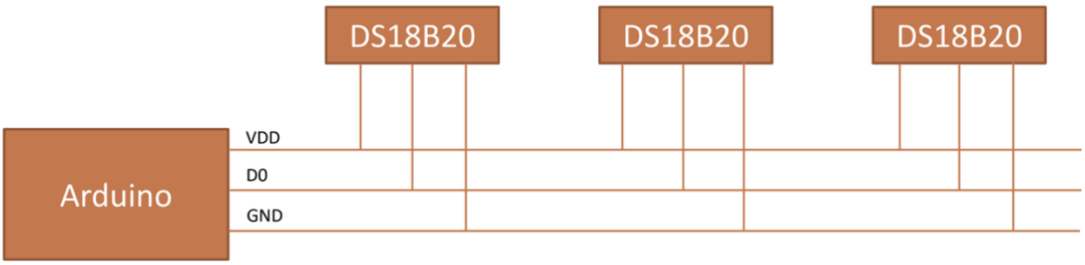

Ilustración 19: Bus del sensor DS18B20

Cada dispositivo cuenta con un identificador propio de 64 bits proporcionado en la fase de fabricación, por lo que es posible diferenciar cuál de los sensores conectados es el que está enviando los datos y decidir a cuál realizarle una petición.

Para poder iniciar una comunicación, es necesario realizar un reinicio del bus manteniendo la señal D0 a 0V durante 48 µs. Tras el reinicio, los dispositivos que estén conectados al bus deben indicar su presencia poniendo la señal D0 a 0V durante 60 µs.

Por otro lado, los datos se envían o reciben en bloques de 8 bits siguiendo la siguiente secuencia:

- Maestro: Si quiere enviar un bit a 1, mantiene la señal D0 a 0V menos de 15 µs. Si lo que quiere es enviarlo a 0, mantiene esta señal a 0V durante 60 µs.

- Esclavo: Si quiere enviar un bit a 1, mantiene la señal D0 a 5V. Si lo que quiere es enviarlo a 0, mantiene esta señal a 0V durante 60 µs.

3.2.2.2 Smartphone – Aplicación nRF Connect

El segundo dispositivo necesario para completar el sistema es un Smartphone, en el que se descargará la aplicación nRF Connect que proporciona el fabricante del SoC.

Ilustración 20: Icono aplicación nRF Connect

Esta aplicación, disponible tanto en la Google Play Store de Android como en el App Store de Apple, permite encontrar dispositivos BLE, configurarlos y comunicarse con ellos. En el capítulo 5, *Pruebas Prototipo*, se estudiarán en profundidad sus funciones.

3.3 Esquema General del Sistema de Comunicación

Tras la selección de los componentes que formarán el Sistema de Comunicación, se presenta el esquema general del mismo.

En primer lugar, de los sensores de temperatura DS18B20 se obtiene la temperatura ambiente, que será enviada al Arduino a través de uno de sus pines digitales, ya que este sensor cuenta con un conversor analógico-digital interno.

Una vez que el Arduino haya recogido los datos, los enviará a la placa nRF52 a través de la interfaz UART. El SoC nRF52832 se encargará de enviar por Bluetooth, a la aplicación nRF Connect, los datos de temperatura recogidos.

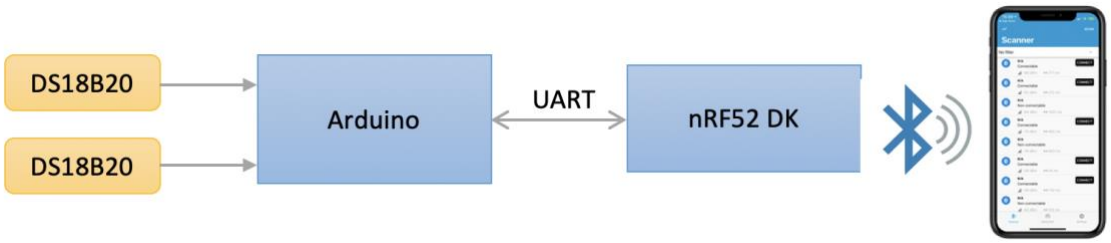

Ilustración 21: Esquema General del Sistema de Comunicación BLE

4. Estructura Software

Para poder llevar a cabo el desarrollo software del proyecto, es necesario contar con las herramientas de configuración y programación de los dispositivos hardware seleccionados en el apartado anterior.

A continuación, se describen estas herramientas software y el desarrollo software del proyecto.

4.1 IDE Arduino

Para la recogida de datos de la sonda de temperatura se hará uso de la placa Arduino MEGA, la cual se programa con el software desarrollado por la misma compañía, IDE Arduino. Esta aplicación de escritorio, que se puede obtener directamente de la página del fabricante, está disponible para los sistemas operativos habituales como son Windows, macOS o Linux.

En primer lugar, al abrir la aplicación, nos encontramos con la pantalla mostrada en la *Ilustración 22*, que corresponde a un sketch, archivo de código de Arduino. Este sketch contiene dos funciones principales, que se pueden complementar con las funciones desarrolladas para un proyecto particular:

- Función `setup`: En esta parte del cuerpo del sketch se recoge la inicialización de las variables que formarán parte del código. Además, se realizarán aquellas acciones que solo se deben ejecutar una vez durante el proceso.

- Función `loop`: Esta función la conforman aquellas acciones que se deben repetir de manera recurrente durante la ejecución del código.

Además de la barra de menú extendida usual de todas las aplicaciones, el programa cuenta con una barra de menú de acciones principales donde se pueden realizar las siguientes acciones: verificar si un sketch es correcto, cargarlo al módulo o placa conectada, crear un nuevo sketch, abrirlo o guardarlo y, por último, la ventana del puerto serie.

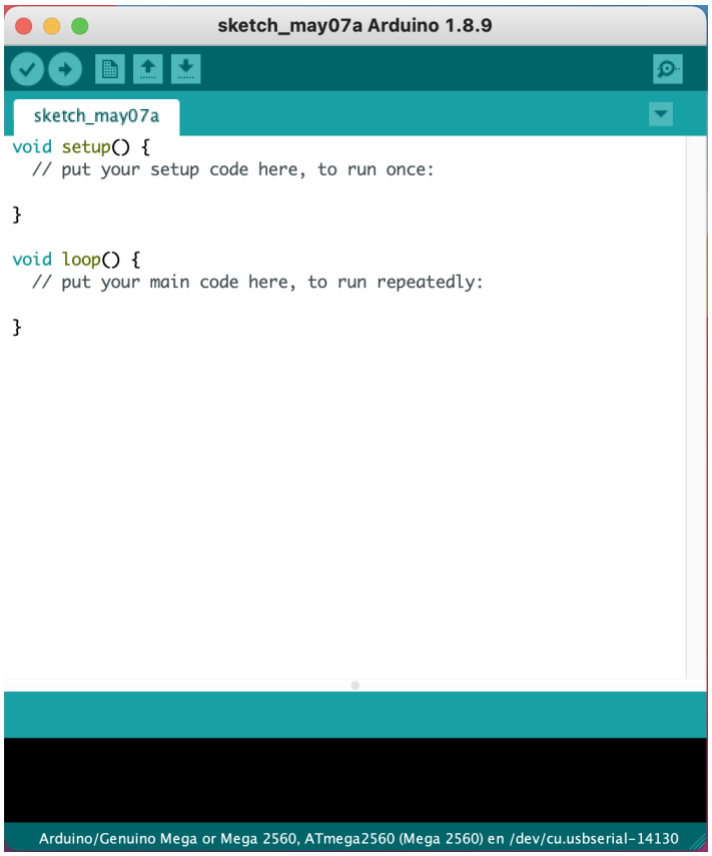

Ilustración 22: Pantalla principal IDE Arduino

4.1.1 Uso del sensor de temperatura DS18B20 en el IDE Arduino

Como se ha comentado anteriormente, el primer paso del sistema consiste en recoger las medidas tomadas por dos sondas de temperatura. Para hacer esto posible, es necesario incorporar al sketch las siguientes librerías [15]:

- DallasTemperature.h: Librería necesaria para la funcionalidad del sensor de temperatura. En esta librería se definen todas las funciones necesarias para establecer comunicación con los sensores, iniciar el bus de comunicación, leer datos de los sensores o especificar su formato (Grados centígrados, grados Kelvin…), entre otras.

- OneWire.h: Como se indicó en la descripción del sensor de temperatura, este cuenta con un protocolo de comunicaciones específico que requiere de esta librería para funcionar correctamente.

4.1.2 Uso de la placa nRF52 DK en el IDE Arduino

A pesar de que en el presente proyecto la programación de la placa nRF52 se hará a través del programa SEGGER, recomendado en la página del fabricante, se va a realizar una breve explicación de la configuración a realizar para poder hacer uso de la placa nRF52 DK desde el software IDE Arduino. [14]

En primer lugar, se debe añadir la URL para la gestión de la placa en las preferencias del programa.

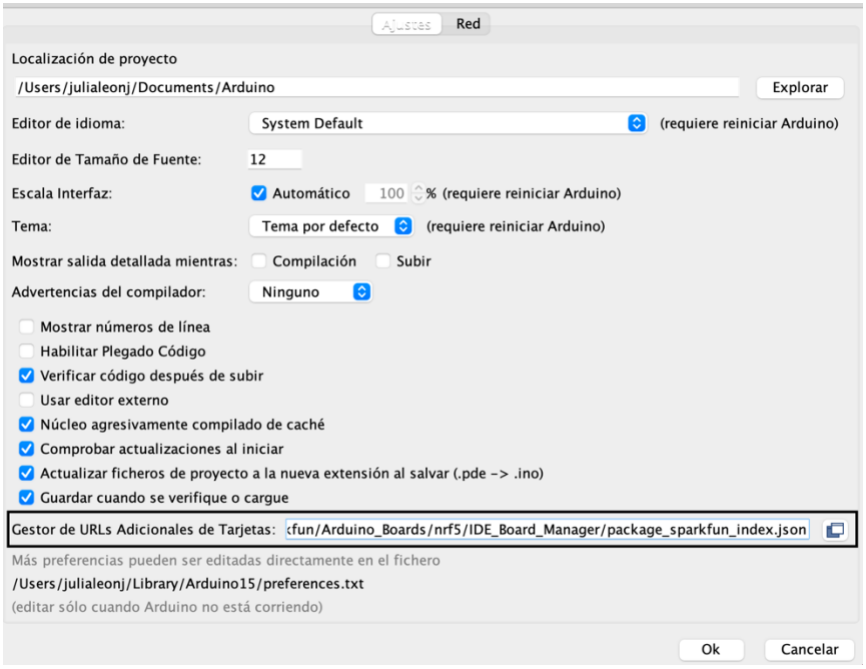

Ilustración 23: Configuración nRF52 en IDE Arduino

A continuación, se instala el gestor de tarjetas nRF52, para que sea posible enviar los datos recogidos por el Arduino al kit de desarrollo nRF52 DK.

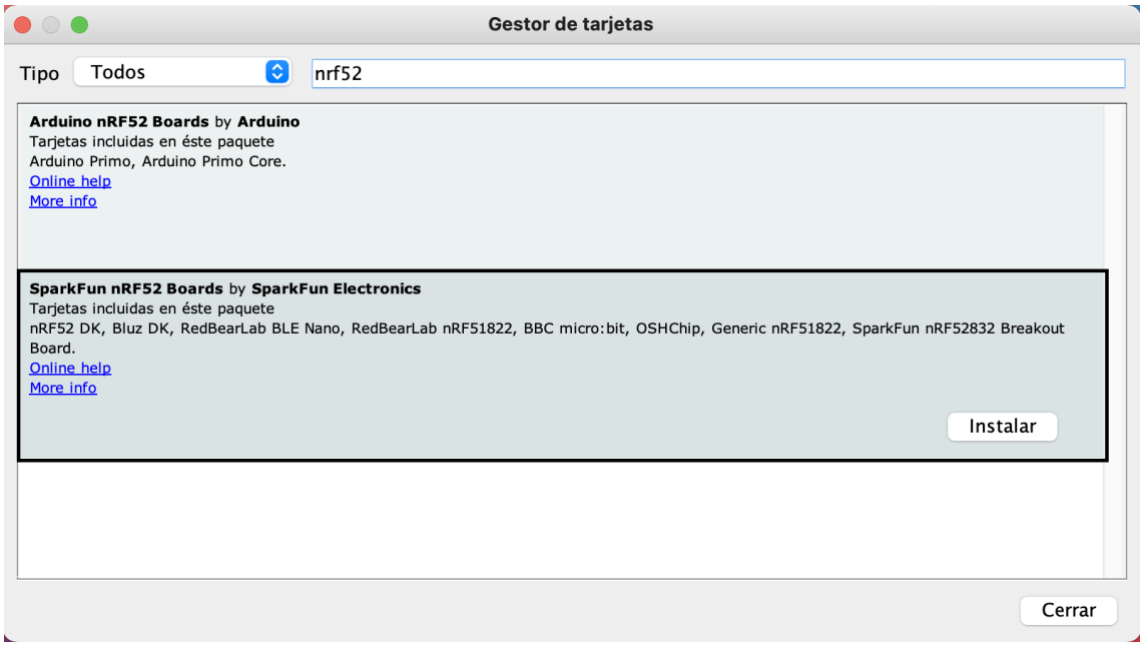

Ilustración 24: Instalación del gestor de tarjetas nRF52 DK

Una vez instalado, el programa permite seleccionar, como placa conectada al ordenador, la placa nRF52 DK.

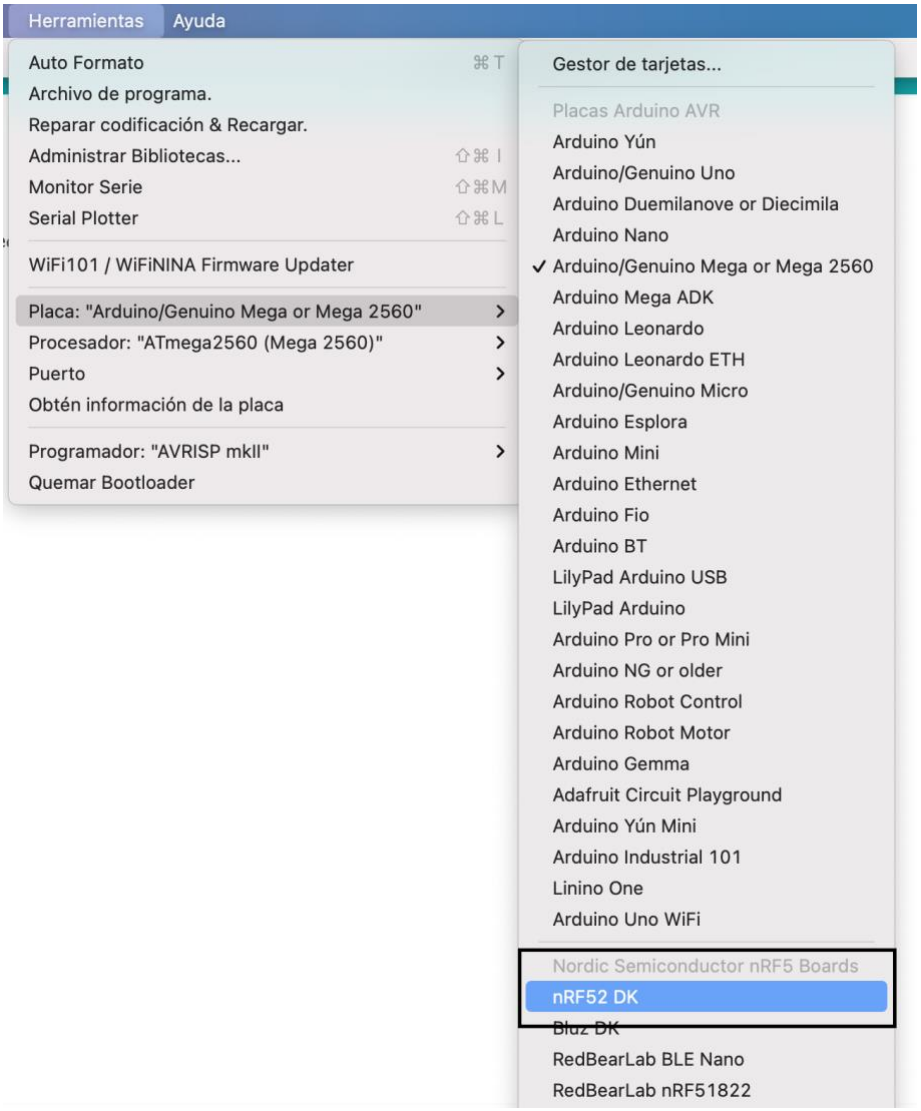

Ilustración 25: Selección de la placa nRF52

4.2 Sketch programado en Arduino para la recogida de datos

En la interfaz de recogida de datos programada en la placa Arduino, en primer lugar, se inicializa la UART y el bus de comunicación 1-wire de las sondas de temperatura, a través de las funciones `begin()` de las librerías *SoftwareSerial* y *DallasTemperature* respectivamente.

La recogida de datos funcionará independientemente del número de sondas que estén conectadas. Con el fin de conocer este dato, se hace uso de la función `getDeviceCount()`, que devuelve un entero indicando el número de sondas detectadas en el bus de comunicación.

Una vez iniciadas las comunicaciones y conocido el número de sondas de las que recolectar los datos, comienza la función `loop()` del sketch que, como se indicó en el apartado anterior, está formada por las acciones que se van a repetir en el código de manera recurrente. En primer lugar, con la función `requestTemperatures()` se envía a los sensores el comando para la toma de

temperatura. A continuación, a través de un bucle `for`, cuyas iteraciones varían en función del número de sondas, se envía por la UART un texto indicando la temperatura de cada uno de los sensores: "`Temperatura del sensor i: XXºC`"

Para obtener la temperatura de los sensores, la librería *DallasTemperature* proporciona dos maneras de llevarlo a cabo: a través del índice o a través de la dirección del sensor. Además, permite obtener la medida en grados Centígrados o en grados Fahrenheit. En este caso, se obtienen los datos por índice en grados Centígrados, haciendo uso de la función `getTempCByIndex(i)`.

En el capítulo *9. Anexos* se puede encontrar el código del programa completo.

4.3 Firmware para la placa nRF52 DK

En cuanto a la programación del kit de desarrollo nRF52 DK de Nordic Semiconductor, el software necesario es el entorno de desarrollo software nRF5 SDK y el programa SEGGER Embedded Studio.

- nRF5 SDK: Kit de desarrollo software. La versión utilizada en el desarrollo del sistema es la v17.0.2.

- SEGGER Embedded Studio: Entorno de desarrollo gratuito integrado para código en lenguaje C o C++. La versión utilizada en el desarrollo del sistema es la v5.42b.

Dentro del entorno nRF5 SDK se pueden encontrar librerías, documentación y ejemplos de las distintas funcionalidades para el kit de desarrollo que se utiliza en este proyecto, por tanto, esto será la base del desarrollo de la parte Bluetooth del sistema [17]. Por otro lado, se debe tener en cuenta el modelo PCB del kit de desarrollo que, como se puede observar en la *Ilustración 12*, el nRF52 DK incorpora el PCA10040. Para el dispositivo software, se ha escogido el SoftDevice s132, que indica la pila de protocolos BLE para maestro o esclavo del chip nRF52.

A continuación, se describen los pasos más importantes llevados a cabo en el archivo principal (*main.c*) del software implementado en el nRF52 DK. En el capítulo *9. Anexos* se puede encontrar el código completo.

Hay que destacar el servicio utilizado en el sistema, Nordic UART Service (NUS), servicio basado en GATT con características de transmisión y recepción de datos. Los datos recibidos por la UART, en formato texto, se envían por Bluetooth a la aplicación nRF Connect, instalada en un dispositivo móvil.

En primer lugar, se declara la instancia del servicio, `BLE_NUS_DEF(our_service, NRF_SDH_TOTAL_LINK_COUNT)`, donde el primer argumento es el servicio del sistema y el segundo argumento indica el número máximo de clientes que se pueden conectar al servicio en una misma instancia. Por otro lado, se declaran las instancias del módulo GATT,

`NRF_BLE_GATT_DEF(m_gatt)`, el módulo de escritura en cola, `NRF_BLE_QWR_DEF(m_qwr)`, y el módulo de *advertising* (publicación), `BLE_ADVERTISING_DEF(m_advertising)`.

A continuación, se definen parámetros de comunicación, como los intervalos máximo y mínimo de conexión o los reintentos de conexión. Por último, se define el UUID del servicio que, en este caso, está establecido en 0x0001 para el servicio NUS.

En la siguiente lista se detallan las funciones principales definidas en este archivo, así como su descripción:

- Función `timers_init`: Inicializa el módulo timer. Mediante esta función se controlan los temporizadores de conexión y comunicación.

- Función `gap_params_init`: Inicializa el módulo GAP, los parámetros GAP del dispositivo, establece los permisos y la apariencia de este. Además, en esta función se establece el nivel de seguridad de las tramas enviadas y recibidas que, en el caso del sistema del proyecto, se selecciona el modo `BLE_GAP_CONN_SEC_MODE_SET_OPEN`, donde no se requiere protección.

- Función `gatt_init`: Inicializa el módulo GATT, los parámetros GATT y establece el máximo tamaño permitido en la MTU.

- Función `uart_init`: Inicializa la UART y sus parámetros. Por un lado, se defininen los pines que se utilizan para la recepción y transmisión de datos, se desactiva el control de flujo, se define la tasa de baudios y la paridad. Por otro lado, se establecen los valores por defecto del FIFO como los valores máximos y mínimos del buffer.

- Función `advertising_init`: Inicializa el módulo de *advertising* y sus parámetros.

- Función `services_init`: Inicializa los servicios que se van a utilizar en el sistema. En este caso, los dos servicios a utilizar son el Nordic UART Service y el Queued Write Module.

- Función `conn_params_init`: Inicializa el módulo de comunicación y sus parámetros.

- Función `ble_evt_handler`: Gestiona los eventos de Bluetooth Low Energy.

- Función `gatt_evt_handler`: Gestiona los eventos de la librería GATT.

- Función `uart_event_handle`: Gestiona los eventos de la UART. Esta función recibe un carácter desde la UART y lo añade a la cadena hasta que recibe el carácter de nueva línea (\n) o se alcanza la máxima

longitud, momento en el que da por finalizada la cadena y la envía por Bluetooth.

- Función `advertising_start`: Pone en funcionamiento el *advertising* del dispositivo en modo `BLE_ADV_MODE_FAST`, es decir, se conecta a cualquier dispositivo.

- Función `idle_state_handle`: Se gestiona el estado IDLE del main. Si no hay ninguna operación de log pendiente, el dispositivo entra en modo reposo hasta que ocurra el próximo evento.

Por último, se encuentra la función `main` de la aplicación, donde, en primer lugar, se llama a todas las funciones de inicialización (`xxxx_init`) descritas anteriormente para inicializar todos los módulos y comunicaciones. A continuación, comienza el *advertising*, momento a partir del cual se recogerán todos los mensajes llegados de la UART y se enviarán por Bluetooth. Por último, se mantiene un bucle en el que el dispositivo entrará en modo reposo mientras no reciba información para enviar a través de Bluetooth.

El esquema que se muestra a continuación resume a grandes rasgos cómo se establece la comunicación entre dos dispositivos BLE [18]. En el caso del sistema del proyecto, el host A se corresponde con el Smartphone y el host B con el nRF52 DK.

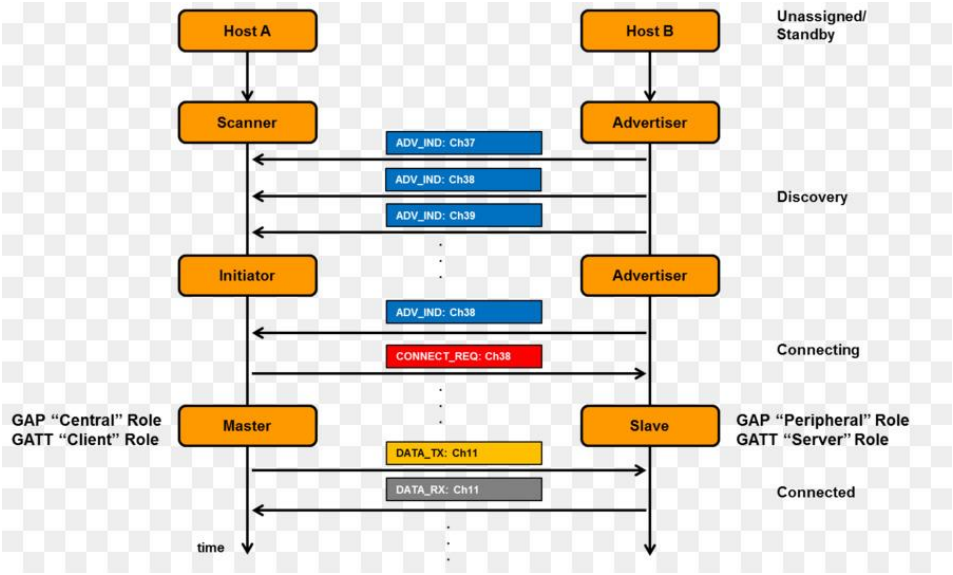

Ilustración 26: Diagrama de comunicación BLE

5. Pruebas Prototipo

Para realizar las pruebas sobre el sistema desarrollado, se ha elaborado un prototipo. Este prototipo se puede observar en la *Ilustración 27*, donde aparecen los elementos hardware seleccionados y descritos en el capítulo *3. Estructura Hardware*: Arduino MEGA Rev. 3, nRF52 DK y las sondas de temperatura DS18B20.

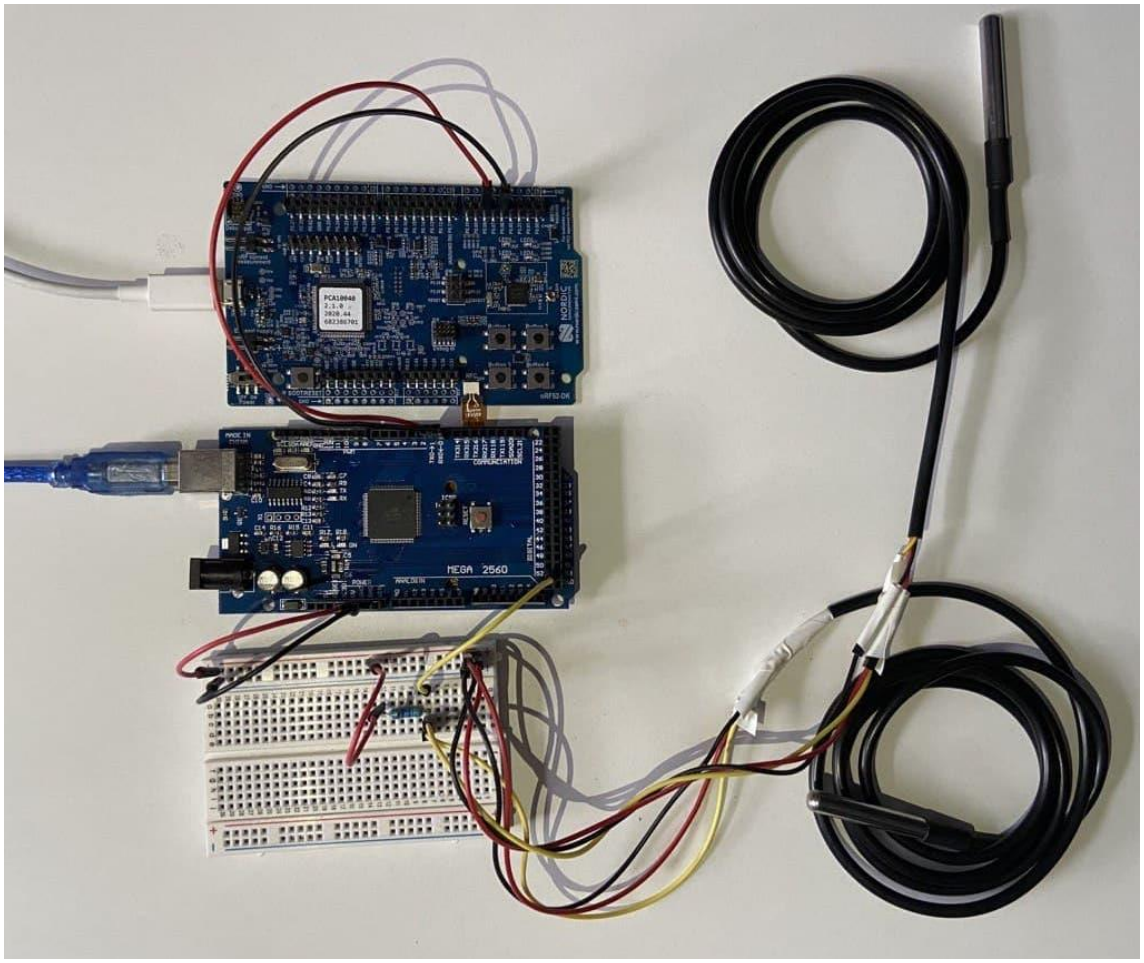

Ilustración 27: Prototipo Sistema BLE

Todas las pruebas se realizan con la aplicación móvil de *nRF Connect for mobile* de Nordic Semiconductor. Esta aplicación está disponible tanto para iOS como para Android en sus respectivas tiendas de aplicaciones.

5.1 Prueba de conectividad

En primer lugar, se va a comprobar que el dispositivo está disponible para comenzar la comunicación. Para ello, se abre la aplicación *nRF Connect* y se pulsa sobre SCAN, momento en el que comenzará el escaneo de dispositivos disponibles.

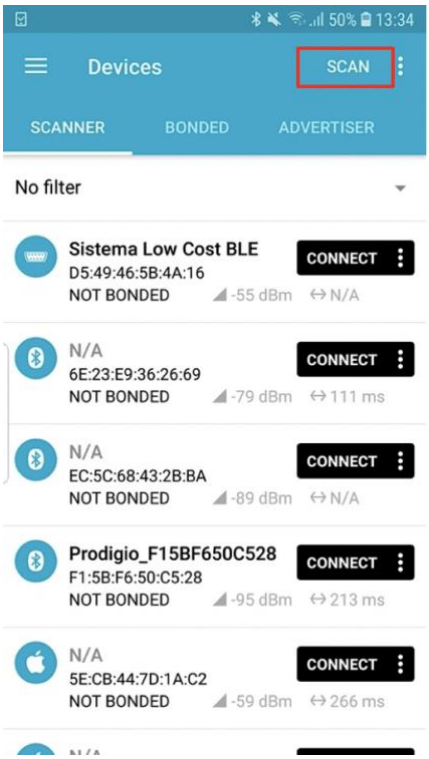

Ilustración 28: Escaneo de dispositivos

Una vez realizado el escaneo, el primer dispositivo que aparece es *Sistema Low Cost BLE* que, como se puede observar en el código del capítulo *9. Anexos*, es el nombre establecido para el dispositivo Bluetooth.

A continuación, se pulsa sobre el botón CONNECT y se establece la conexión entre el Smartphone y el Sistema BLE. En este momento, aparece en la pantalla la lista de atributos del dispositivo, que se agrupan en servicios. En este caso, aparece el servicio Nordic UART Service (NUS), que es el utilizado en el sistema.

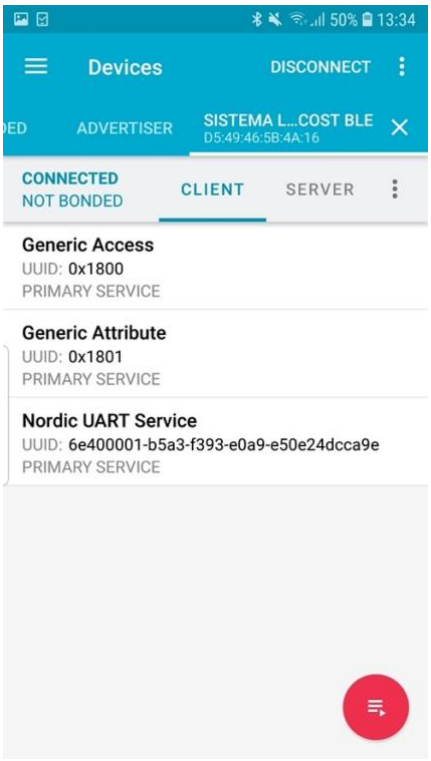

Ilustración 29: Conexión entre Smartphone y Sistema BLE

Para ver las características que posee un servicio, se hace clic sobre el nombre de este. En la *Ilustración 30* se muestran las características del servicio NUS que, visto desde el nRF52 DK, son:

- Característica RX: El nRF52 DK recibe información del Smartphone y la envía por la UART. En este sistema, la función que trata estos datos está deshabilitada ya que se pretende únicamente enviar por BLE los datos recibidos por la UART, pero se mantiene la característica para futuras modificaciones.

- Característica TX: Cuando están activas las notificaciones, el nRF52 DK envía por Bluetooth la información recibida por la UART como notificaciones.

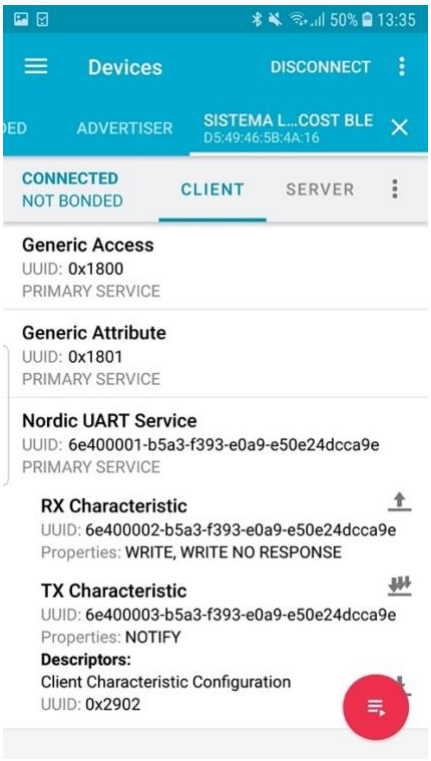

Ilustración 30: Características del servicio NUS

5.2 Prueba del Sistema Low Cost BLE completo

En el presente apartado se va a realizar la primera prueba sobre el dispositivo completo una vez realizada la conexión. Como se observa en la *Ilustración 30*, la característica TX del dispositivo tiene la propiedad NOTIFY y el atributo CCCD (Client Characteristic Configuration Descriptor).

Por defecto, los CCCDs vienen deshabilitados en la aplicación, por tanto, es necesario habilitarlos para poder ver en el log de la aplicación los mensajes recibidos por Bluetooth. Para habilitarlos, se debe hacer clic en los tres puntos de la derecha, al lado de SERVER, y pulsar sobre *Enable CCCDs*.

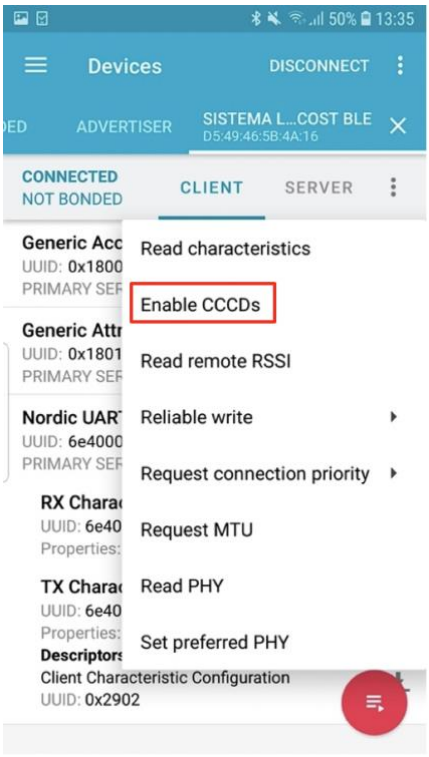

Ilustración 31: Habilitación de CCCDs

A continuación, se puede observar como el campo *Value* del CCCD ha tomado el valor “Notifications enabled”.

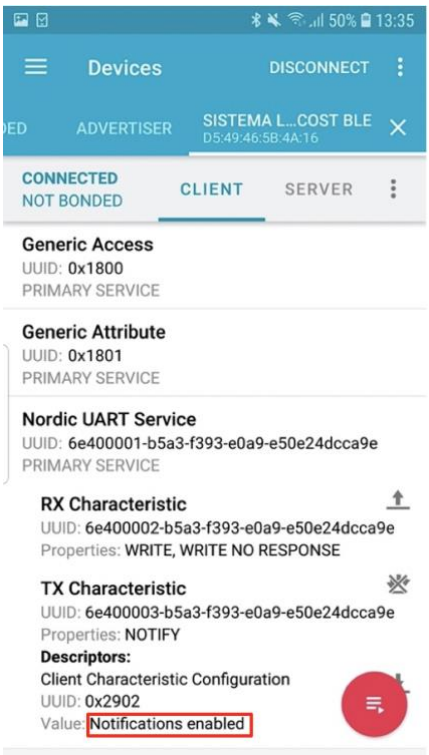

Ilustración 32: Notificaciones activadas

Una vez activadas las notificaciones, se accede al log de la aplicación, donde se podrán ver todos los mensajes intercambiados entre el Smartphone y el Sistema BLE, incluidos los mensajes de temperatura que se han recibido de Arduino por la UART del nRF52 DK y se han enviado por Bluetooth.

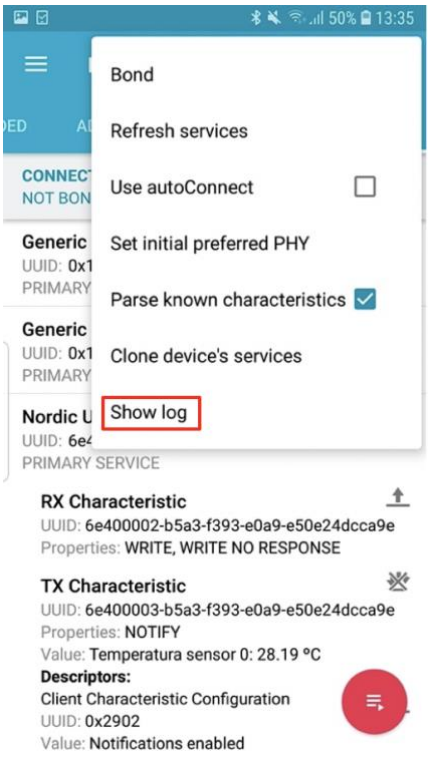

Ilustración 33: Mostrar log de nRF Connect

Como se observa en la siguiente imagen, el Smartphone está recibiendo mensajes con el valor de los sensores de temperatura que están conectados al Arduino. Cada vez que el Smartphone va a recibir un mensaje nuevo, recibe una notificación del tipo “Notification received from…” junto al UUID del servicio NUS.

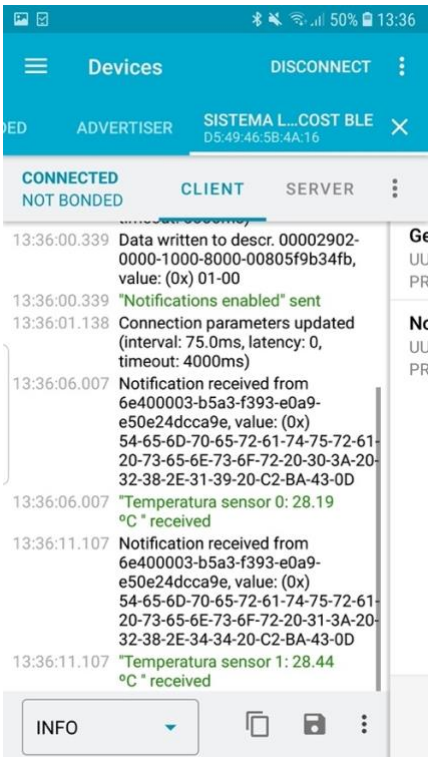

Ilustración 34: Log nRF Connect

Por otro lado, si se muestran en el log únicamente los mensajes de la capa de aplicación sólo aparecerán los mensajes de este tipo.

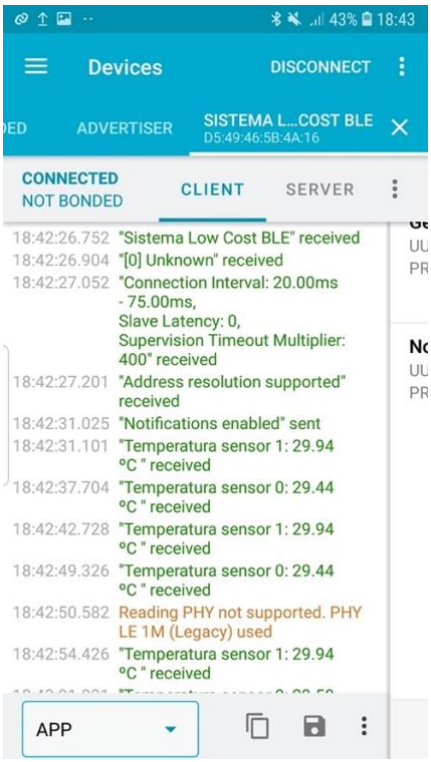

Ilustración 35: Log mensajes APP nRF Connect

5.3 Prueba de recogida del Sistema Low Cost BLE completo alterando la temperatura

La última prueba por realizar consiste en alterar la temperatura de una de las sondas conectadas. En particular, se aplicará calor para aumentar su temperatura y así observar en la aplicación *nRF Connect* como aumentan los grados Centígrados en las notificaciones recibidas del sensor número 1.

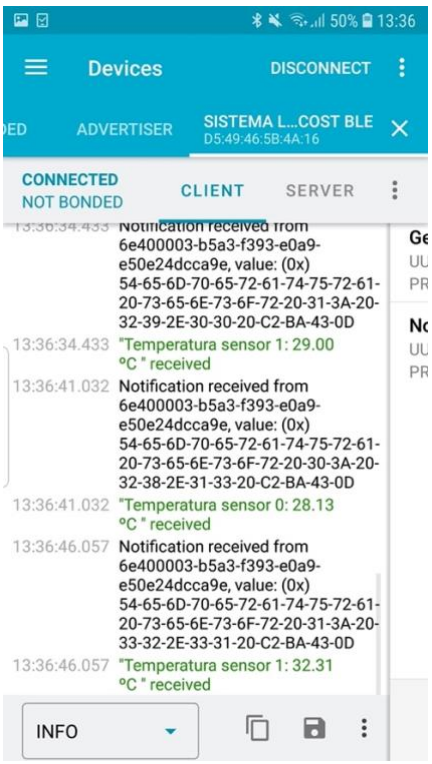

Ilustración 36: Log Aumento de Temperatura

6. Conclusiones

Una vez finalizado el diseño y desarrollo del Sistema de comunicación BLE Low Cost, se puede concluir que se ha alcanzo los objetivos establecidos en el primer capítulo del proyecto. Se ha estudiado a fondo el Bluetooth Low Energy, conociendo cómo se da la comunicación, sus parámetros y aplicaciones, se ha realizado un estudio de mercado en cuanto a los SoC disponibles y se ha desarrollado un prototipo funcional de un Sistema de comunicación BLE Low Cost.

6.1 Presupuesto total del desarrollo del Sistema

Los componentes hardware del prototipo se han adquirido en tiendas online low cost. En su mayoría, los componentes son de “Aliexpress”, a excepción del kit de desarrollo nRF52 DK que, por motivos de tiempo de espera, se adquirió a través de “RS Components”.

Tabla 8: Coste componentes

Componente	Coste
SEGGER Embedded Studio	Gratuito
nRF5 SDK	Gratuito
Arduino MEGA	7.43€
nRF52 DK	33.66€
Sondas DS18B20	1€/ud
Componentes Arduino	1.94€
Total	**45.03€**

Por otro lado, si se tiene en cuenta el trabajo realizado por el ingeniero, el coste total aumentaría en 7200€, asumiendo el contrato de un ingeniero con poca experiencia que cobra 15€/hora y trabaja 40h semanales durante 3 meses, tiempo que ha durado el total del desarrollo del proyecto.

Si el producto se comercializa, el trabajo realizado por el ingeniero sólo se ejecuta una vez, por lo que no se debe tener en cuenta para todas las unidades.

En conclusión, el sistema desarrollado es económico y cumple con las expectativas puestas tras analizar otros sistemas del mercado como los descritos en el capítulo 2. *Estado del Arte*. Se debe tener en cuenta que el tamaño del prototipo es superior al que se lograría alcanzar realizando un sistema integrado, ya que el kit de desarrollo nRF52 DK y el Arduino tienen un

tamaño grande que se podría minimizar utilizando sólo el SoC y una versión de Arduino más pequeña.

6.2 Mejoras futuras

Una vez finalizado el proyecto, se plantean mejoras futuras que pueden ayudar a realizar modificaciones sobre el sistema que lleven al desarrollo de otros sistemas aplicados a un ámbito en concreto.

- Se pueden añadir más sensores o actuadores conectados a Arduino de manera que se puedan supervisar y controlar hogares o parámetros de un ecosistema.

- Mejora de la estructura software con el fin de optimizar el espacio en memoria y su fluidez.

- Desarrollo de una aplicación propia para la supervisión y control de los sensores y actuadores.

- Desarrollo de un sistema de alarmas, de manera que sólo se generen mensajes cuando los parámetros supervisados se encuentren fuera del rango establecido.

7. Glosario

BLE	Bluetooth Low Energy
GATT	Generic Attribute Profile
GAP	Generic Acess Profile
PHY	Physical Layer
SoC	System on Chip
DK	Development Kit
SDK	Software Development Kit
IDE	Integrated Development Environment
UUID	Universally Unique ID
ADC	Convertidor Analógico-Digital
UART	Universal Asynchronous Receiver-Transmitter
NUS	Nordic UART Service
MUT	Maximum Transfer Unit

8. Bibliografía

[1] Características y aplicaciones BLE, diferencias con Bluetooth Classic
https://en.wikipedia.org/wiki/Bluetooth_Low_Energy

[2] Pila de protocolos BLE
https://smart-lighting.es/bluetooth-low-energy-introduccion-la-tecnologia/

[3] Wearables BLE
https://developex.com/blog/the-role-of-ble-in-wearable-technology/

[4] Imagen Wearables BLE
https://techcrunch.com/2015/06/10/a-periodic-table-of-wearable-technology/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAAE_YPcsbU4OgVlM55H3vXrQgfCqOfw3lZdYazGWpkqsoasHFsZtRbXnQNpgeQHrfw-ja_2wCfv_vqKF1RBphmEypLMIUFwm_N1ciPBILydxBg9mCkz0s-YDijM0TeGlnPNcaMq_Rbbv8bjwS17Wy15sDJzN2F_pt7GzzGfKcVnQ

[5] Versiones BLE
https://www.digikey.es/es/articles/bluetooth-low-energy-design-101-from-chipsets-to-protocol-stacks-to-modules
https://mundoaltavoces.com/bluetooth-todas-las-clases-y-versiones-en-que-se-diferencian/

[6] Imagen aplicación BLE
https://www.compartirwifi.com/blog/localizacion-en-interiores-a-traves-de-tecnologias-como-wifi-o-beacon/

[7] Monitor Cardíaco Portátil con Interfaz Bluetooth - Dialnet
https://dialnet.unirioja.es/descarga/articulo/5774772.pdf

[8] Módulo ISP1507
https://www.mouser.es/ProductDetail/Insight-SiP/ISP1507-AX-RS?qs=wd5RIQLrsJgv7KmXryj2YA%3D%3D

[9] Imagen módulo ISP1507
https://www.comunicacionesinalambricashoy.com/modulo-bluetooth-baja-energia/

[10] Información sobre Arduino y las placas
https://www.arduino.cc/en/Main/Products

[11] Arduino
https://es.wikipedia.org/wiki/Arduino

[12] nRF52 DK - Datasheet
https://www.nordicsemi.com/Software-and-Tools/Development-Kits/nRF52-DK

[13] Blue-NRG2
https://www.st.com/en/wireless-connectivity/bluenrg-2.html

[14] Arduino IDE con el kit nRF52 DK
https://jimmywongiot.com/2020/09/07/arduino-ide-with-nrf52-dk-board/

[15] Sonda DS18B20
https://programarfacil.com/blog/arduino-blog/ds18b20-sensor-temperatura-arduino/

[16] Explicación de 1-Wire en la UART
https://www.maximintegrated.com/en/design/technical-documents/tutorials/2/214.html

[17] Nordic Semiconductor - Ejemplo de servicio NUS
https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v14.0.0%2Fble_sdk_app_nus_eval.html

[18] Diagrama de comunicación BLE
https://www.freepng.es/png-1c4rqj/

9. Anexos

9.1 Código del sketch Arduino

```
#include <SoftwareSerial.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// Pin donde se conecta el bus 1-Wire
const int pinDatosDQ = 52;

// Objetos
OneWire oneWireObjeto(pinDatosDQ);
DallasTemperature sensorDS18B20(&oneWireObjeto);

// Variables
String str;

void setup() {
  // Se inicia la comunicacion serie entre Arduino y el DK nRF52
  Serial.begin(115200);

  // Se inicia el bus de comunicaciones
  sensorDS18B20.begin();
  delay(2000);

  // Se muestra el número sondas DS18B20 que hay conectadas al bus
  str = String("Sondas DS18B20 conectadas: ") + String(sensorDS18B20.getDeviceCount());
  Serial.println(str);
}

void loop() {
  // Se lee la temperatura de todos los sensores conectados al bus
  delay(1000);
  sensorDS18B20.requestTemperatures();

  // Se envía por la UART la temperatura medida por los sensores conectados al bus
  for (int i = 0; i < sensorDS18B20.getDeviceCount(); i++){
    // Se envía por la UART la temperatura del sensor con ID = 0
    str = String("Temperatura sensor ") + String(i) + String (": ") + String(sensorDS18B20.getTempCByIndex(i)) + String(" ºC");
    Serial.println(str);
    delay(5000);
  }
```

9.2 Archivo main.c SoC

```
#include <stdint.h>
#include <string.h>
#include "nordic_common.h"
#include "nrf.h"
#include "ble_hci.h"
#include "ble_advdata.h"
#include "ble_advertising.h"
#include "ble_conn_params.h"
#include "nrf_sdh.h"
#include "nrf_sdh_soc.h"
#include "nrf_sdh_ble.h"
#include "nrf_ble_gatt.h"
#include "nrf_ble_qwr.h"
#include "app_timer.h"
```

```
#include "ble_nus.h"
#include "app_uart.h"
#include "app_util_platform.h"
#include "bsp_btn_ble.h"
#include "nrf_pwr_mgmt.h"
#include "nrf_uart.h"
#include "nrf_log.h"
#include "nrf_log_ctrl.h"
#include "nrf_log_default_backends.h"

/**< Identificador configuración SoftDevice BLE. */
#define APP_BLE_CONN_CFG_TAG            1

/**< Nombre del dispositivo. */
#define DEVICE_NAME                     "Sistema Low Cost BLE"
/**< Tipo de UUID para el servicio Nordic UART Service (vendor specific). */
#define NUS_SERVICE_UUID_TYPE           BLE_UUID_TYPE_VENDOR_BEGIN

/**< Prioridad aplicación BLE. */
#define APP_BLE_OBSERVER_PRIO           3

/**< Intervalo de publicación (Unidades de 0.625ms). */
#define APP_ADV_INTERVAL                64

/**< Duración de la publicación (Unidades 10ms) */
#define APP_ADV_DURATION                18000

/**< Mínimo intervalo de conexión. */
#define MIN_CONN_INTERVAL               MSEC_TO_UNITS(20, UNIT_1_25_MS)
/**< Máximo intervalo de conexión. */
#define MAX_CONN_INTERVAL               MSEC_TO_UNITS(75, UNIT_1_25_MS)
/**< Latencia del esclavo. */
#define SLAVE_LATENCY                   0
/**< Timeout conexión supervisor (Par·metro GAP). */
#define CONN_SUP_TIMEOUT                MSEC_TO_UNITS(4000, UNIT_10_MS)
 /**< Tiempo evento primera llamada a sd_ble_gap_conn_param_update (5s). */
```

```
#define FIRST_CONN_PARAMS_UPDATE_DELAY  APP_TIMER_TICKS(5000)
/**< Tiempo entre cada llamada a sd_ble_gap_conn_param_update tras la primera llamada (30s). */
#define NEXT_CONN_PARAMS_UPDATE_DELAY   APP_TIMER_TICKS(30000)
 /**< Intentos antes de renunciar a la negociación de conexión. */
#define MAX_CONN_PARAMS_UPDATE_COUNT    3

/**< Código de error en stack dump. */
#define DEAD_BEEF                       0xDEADBEEF

/**< Tamaño del buffer TX UART. */
#define UART_TX_BUF_SIZE                256
/**< Tamaño del buffer RX UART. */
#define UART_RX_BUF_SIZE                256

#define NRF_SDH_BLE_TOTAL_LINK_COUNT    1

/**< Instancia servicio BLE NUS. */
BLE_NUS_DEF(our_service, NRF_SDH_BLE_TOTAL_LINK_COUNT);
/**< Instancia módulo GATT. */
NRF_BLE_GATT_DEF(m_gatt);
/**< Contexto módulo Queued Write.*/
NRF_BLE_QWR_DEF(m_qwr);
/**< Instancia módulo Advertising. */
BLE_ADVERTISING_DEF(m_advertising);

/**< Manejo de la conexión actual. */
static uint16_t   m_conn_handle         = BLE_CONN_HANDLE_INVALID;
/**< Longitud máxima de trama (bytes) para transmitir. */
static uint16_t   m_ble_nus_max_data_len = BLE_GATT_ATT_MTU_DEFAULT - 3;
/**< Universally unique service identifier. */
static ble_uuid_t m_adv_uuids[]         =
{
   {BLE_UUID_NUS_SERVICE, NUS_SERVICE_UUID_TYPE}
};
```

```
/* Función para inicializar los timers. */
static void timers_init(void)
{
    ret_code_t err_code = app_timer_init();
    APP_ERROR_CHECK(err_code);
}


/* Función para inicializar GAP. */
static void gap_params_init(void)
{
    uint32_t                err_code;
    ble_gap_conn_params_t   gap_conn_params;
    ble_gap_conn_sec_mode_t sec_mode;

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);

    err_code = sd_ble_gap_device_name_set(&sec_mode,
                                          (const uint8_t *) DEVICE_NAME,
                                          strlen(DEVICE_NAME));
    APP_ERROR_CHECK(err_code);

    memset(&gap_conn_params, 0, sizeof(gap_conn_params));

    gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
    gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
    gap_conn_params.slave_latency     = SLAVE_LATENCY;
    gap_conn_params.conn_sup_timeout  = CONN_SUP_TIMEOUT;

    err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
    APP_ERROR_CHECK(err_code);
}


/* Función para el manejo de errores del módulo Queued Write. */
static void nrf_qwr_error_handler(uint32_t nrf_error)
```

```
{
    APP_ERROR_HANDLER(nrf_error);
}


/* Función para inicializar los servicios. */
static void services_init(void)
{
    uint32_t           err_code;
    ble_nus_init_t     nus_init;
    nrf_ble_qwr_init_t qwr_init = {0};

    qwr_init.error_handler = nrf_qwr_error_handler;

    err_code = nrf_ble_qwr_init(&m_qwr, &qwr_init);
    APP_ERROR_CHECK(err_code);

    memset(&nus_init, 0, sizeof(nus_init));

    err_code = ble_nus_init(&our_service, &nus_init);
    APP_ERROR_CHECK(err_code);
}


/* Función para el manejo de eventos del módulo de Parámetros de Conexión. */
static void on_conn_params_evt(ble_conn_params_evt_t * p_evt)
{
    uint32_t err_code;

    if (p_evt->evt_type == BLE_CONN_PARAMS_EVT_FAILED)
    {
        err_code = sd_ble_gap_disconnect(m_conn_handle,
BLE_HCI_CONN_INTERVAL_UNACCEPTABLE);
        APP_ERROR_CHECK(err_code);
    }
}
```

```
/* Función para el manejo de errores del módulo de Par·metros de Conexión. */
static void conn_params_error_handler(uint32_t nrf_error)
{
    APP_ERROR_HANDLER(nrf_error);
}


/* Función para inicializar el módulo de Parámetros de Conexión. */
static void conn_params_init(void)
{
    uint32_t               err_code;
    ble_conn_params_init_t cp_init;

    memset(&cp_init, 0, sizeof(cp_init));

    cp_init.p_conn_params                  = NULL;
    cp_init.first_conn_params_update_delay = FIRST_CONN_PARAMS_UPDATE_DELAY;
    cp_init.next_conn_params_update_delay  = NEXT_CONN_PARAMS_UPDATE_DELAY;
    cp_init.max_conn_params_update_count   = MAX_CONN_PARAMS_UPDATE_COUNT;
    cp_init.start_on_notify_cccd_handle    = BLE_GATT_HANDLE_INVALID;
    cp_init.disconnect_on_fail             = false;
    cp_init.evt_handler                    = on_conn_params_evt;
    cp_init.error_handler                  = conn_params_error_handler;

    err_code = ble_conn_params_init(&cp_init);
    APP_ERROR_CHECK(err_code);
}


/* Función para el manejo de eventos de advertising. */
static void on_adv_evt(ble_adv_evt_t ble_adv_evt)
{
    uint32_t err_code;
```

```
    switch (ble_adv_evt)
    {
        case BLE_ADV_EVT_FAST:
            err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING);
            APP_ERROR_CHECK(err_code);
            break;
        default:
            break;
    }
}


/* Función para el manejo de eventos BLE. */
static void ble_evt_handler(ble_evt_t const * p_ble_evt, void * p_context)
{
    uint32_t err_code;

    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GAP_EVT_CONNECTED:
            NRF_LOG_INFO("Conectado");
            err_code = bsp_indication_set(BSP_INDICATE_CONNECTED);
            APP_ERROR_CHECK(err_code);
            m_conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
            err_code = nrf_ble_qwr_conn_handle_assign(&m_qwr, m_conn_handle);
            APP_ERROR_CHECK(err_code);
            break;

        case BLE_GAP_EVT_DISCONNECTED:
            NRF_LOG_INFO("Desconectado");
            m_conn_handle = BLE_CONN_HANDLE_INVALID;
            break;

        case BLE_GAP_EVT_PHY_UPDATE_REQUEST:
        {
            NRF_LOG_DEBUG("PHY update request.");
```

```
            ble_gap_phys_t const phys =
            {
                .rx_phys = BLE_GAP_PHY_AUTO,
                .tx_phys = BLE_GAP_PHY_AUTO,
            };
            err_code = sd_ble_gap_phy_update(p_ble_evt->evt.gap_evt.conn_handle, &phys);
            APP_ERROR_CHECK(err_code);
        } break;

        case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
            // Paridad no soportada
            err_code = sd_ble_gap_sec_params_reply(m_conn_handle,
BLE_GAP_SEC_STATUS_PAIRING_NOT_SUPP, NULL, NULL);
            APP_ERROR_CHECK(err_code);
            break;

        case BLE_GATTS_EVT_SYS_ATTR_MISSING:
            // No hay atributos almacenados
            err_code = sd_ble_gatts_sys_attr_set(m_conn_handle, NULL, 0, 0);
            APP_ERROR_CHECK(err_code);
            break;

        case BLE_GATTC_EVT_TIMEOUT:
            // Evento timeout desconexión Cliente GATT
            err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gattc_evt.conn_handle,
                                             BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
            APP_ERROR_CHECK(err_code);
            break;

        case BLE_GATTS_EVT_TIMEOUT:
            // Evento timeout desconexión Servidor GATT
            err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gatts_evt.conn_handle,
                                             BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
            APP_ERROR_CHECK(err_code);
            break;
```

```
        default:
            break;
    }
}


/* Función para inicializar el SoftDevice y el stack de BLE. */
static void ble_stack_init(void)
{
    ret_code_t err_code;

    err_code = nrf_sdh_enable_request();
    APP_ERROR_CHECK(err_code);

    // Configuración del stack BLE
    uint32_t ram_start = 0;
    err_code = nrf_sdh_ble_default_cfg_set(APP_BLE_CONN_CFG_TAG, &ram_start);
    APP_ERROR_CHECK(err_code);

    // Habilitar stack
    err_code = nrf_sdh_ble_enable(&ram_start);
    APP_ERROR_CHECK(err_code);

    NRF_SDH_BLE_OBSERVER(m_ble_observer, APP_BLE_OBSERVER_PRIO,
ble_evt_handler, NULL);
}


/* Función para el manejo de eventos de la librería GATT. */
void gatt_evt_handler(nrf_ble_gatt_t * p_gatt, nrf_ble_gatt_evt_t const * p_evt)
{
    if ((m_conn_handle == p_evt->conn_handle) && (p_evt->evt_id ==
NRF_BLE_GATT_EVT_ATT_MTU_UPDATED))
    {
        m_ble_nus_max_data_len = p_evt->params.att_mtu_effective - OPCODE_LENGTH -
HANDLE_LENGTH;
```

```
        NRF_LOG_INFO("Longitud de datos 0x%X(%d)", m_ble_nus_max_data_len,
m_ble_nus_max_data_len);
    }
    NRF_LOG_DEBUG("Cambio de ATT MTU completado. central 0x%x peripheral 0x%x",
                  p_gatt->att_mtu_desired_central,
                  p_gatt->att_mtu_desired_periph);
}


/* Función para inicializar GATT. */
void gatt_init(void)
{
    ret_code_t err_code;

    err_code = nrf_ble_gatt_init(&m_gatt, gatt_evt_handler);
    APP_ERROR_CHECK(err_code);

    err_code = nrf_ble_gatt_att_mtu_periph_set(&m_gatt,
NRF_SDH_BLE_GATT_MAX_MTU_SIZE);
    APP_ERROR_CHECK(err_code);
}


/* Función para el manejo de eventos de la UART. */
void uart_event_handle(app_uart_evt_t * p_event)
{
    static uint8_t data_array[BLE_NUS_MAX_DATA_LEN];
    static uint8_t index = 0;
    uint32_t       err_code;

    switch (p_event->evt_type)
    {
        case APP_UART_DATA_READY:
            UNUSED_VARIABLE(app_uart_get(&data_array[index]));
            index++;
```

```
            if ((data_array[index - 1] == '\n') ||
                (index >= m_ble_nus_max_data_len))
            {
                if (index > 1)
                {
                    NRF_LOG_DEBUG("Listo para el envìo por BLE");
                    NRF_LOG_HEXDUMP_DEBUG(data_array, index);

                    do
                    {
                        uint16_t length = (uint16_t)index;
                        err_code = ble_nus_data_send(&our_service, data_array, &length,
m_conn_handle);
                        if ((err_code != NRF_ERROR_INVALID_STATE) &&
                            (err_code != NRF_ERROR_RESOURCES) &&
                            (err_code != NRF_ERROR_NOT_FOUND))
                        {
                            APP_ERROR_CHECK(err_code);
                        }
                    } while (err_code == NRF_ERROR_RESOURCES);
                }

                index = 0;
            }
            break;

        case APP_UART_COMMUNICATION_ERROR:
            APP_ERROR_HANDLER(p_event->data.error_communication);
            break;

        case APP_UART_FIFO_ERROR:
            APP_ERROR_HANDLER(p_event->data.error_code);
            break;

        default:
            break;
```

```
    }
}


/*Función que inicializa el módulo UART. */
static void uart_init(void)
{
    uint32_t                   err_code;
    app_uart_comm_params_t const comm_params =
    {
        .rx_pin_no    = RX_PIN_NUMBER,
        .tx_pin_no    = TX_PIN_NUMBER,
        .rts_pin_no   = RTS_PIN_NUMBER,
        .cts_pin_no   = CTS_PIN_NUMBER,
        .flow_control = APP_UART_FLOW_CONTROL_DISABLED,
        .use_parity   = false,
        .baud_rate    = NRF_UART_BAUDRATE_115200
    };

    APP_UART_FIFO_INIT(&comm_params,
                       UART_RX_BUF_SIZE,
                       UART_TX_BUF_SIZE,
                       uart_event_handle,
                       APP_IRQ_PRIORITY_LOWEST,
                       err_code);
    APP_ERROR_CHECK(err_code);
}


/* Función que inicializa el módulo de advertising. */
static void advertising_init(void)
{
    uint32_t               err_code;
    ble_advertising_init_t init;

    memset(&init, 0, sizeof(init));
```

```
    init.advdata.name_type          = BLE_ADVDATA_FULL_NAME;
    init.advdata.include_appearance = false;
    init.advdata.flags              = BLE_GAP_ADV_FLAGS_LE_ONLY_LIMITED_DISC_MODE;

    init.srdata.uuids_complete.uuid_cnt = sizeof(m_adv_uuids) / sizeof(m_adv_uuids[0]);
    init.srdata.uuids_complete.p_uuids  = m_adv_uuids;

    init.config.ble_adv_fast_enabled  = true;
    init.config.ble_adv_fast_interval = APP_ADV_INTERVAL;
    init.config.ble_adv_fast_timeout  = APP_ADV_DURATION;
    init.evt_handler = on_adv_evt;

    err_code = ble_advertising_init(&m_advertising, &init);
    APP_ERROR_CHECK(err_code);

    ble_advertising_conn_cfg_tag_set(&m_advertising, APP_BLE_CONN_CFG_TAG);
}


/* Función que inicializa la gestión de la energìa. */
static void power_management_init(void)
{
    ret_code_t err_code;
    err_code = nrf_pwr_mgmt_init();
    APP_ERROR_CHECK(err_code);
}


/* Función para el manejo del estado IDLE del main. */
static void idle_state_handle(void)
{
    if (NRF_LOG_PROCESS() == false)
    {
        nrf_pwr_mgmt_run();
    }
```

```
}


/* Función que comienza con el advertising. */
static void advertising_start(void)
{
    uint32_t err_code = ble_advertising_start(&m_advertising, BLE_ADV_MODE_FAST);
    APP_ERROR_CHECK(err_code);
}


/* Función que inicializa el módulo de log de nRF. */
static void log_init(void)
{
    ret_code_t err_code = NRF_LOG_INIT(NULL);
    APP_ERROR_CHECK(err_code);

    NRF_LOG_DEFAULT_BACKENDS_INIT();
}


int main(void)
{

    // Se inicializan todos los módulos y servicios
    uart_init();
    log_init();
    timers_init();
    power_management_init();
    ble_stack_init();
    gap_params_init();
    gatt_init();
    services_init();
    advertising_init();
    conn_params_init();
```

```
    // Comienza la ejecución
    printf("\r\nUART comenzada.\r\n");
    advertising_start();

    for (;;)
    {
        idle_state_handle();
    }
}
```