Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Desarrollo de dispositivos Bluetooth LE basados en CC2540/41

ESCUELA POLITECNICA
SUPERIOR

Autor: Francisco Javier García-Calvo García

Tutor/es: José Manuel Rodríguez Ascariz

2016

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Desarrollo de dispositivos Bluetooth LE basados en CC2540/41

Autor: Francisco Javier García-Calvo García

Tutor: José Manuel Rodríguez Ascariz

Presidente: Luciano Boquete Vázquez

Vocal 1: José Manuel Villadangos Carrizo

Vocal 2: José Manuel Rodríguez Ascariz

Calificación: …….
Fecha: ……

Resumen

Gracias a su autonomía y a su bajo coste cada vez son más los dispositivos basados en la tecnología Bluetooth LE, desde termómetros inalámbricos o sensores para medir la presión sanguínea a dispositivos deportivos como relojes con GPS o sensores para runners.

El trabajo que aquí se propone consistirá en un estudio de la especificación Bluetooth LE profundizando en el funcionamiento de toda su pila de protocolos y su uso para la creación de aplicaciones.

Finalmente, como ejemplo, se desarrollará una aplicación de medida de la distancia respecto a una baliza a partir de la potencia de recepción de los paquetes BLE transmitidos por ésta.

Palabras Clave: Bluetooth Low Energy, Protocolos, CC2540/41, Desarrollo aplicaciones, Transmisión inalámbrica.

Abstract

Thanks to its autonomy and its low cost, Bluetooth LE technology based devices are increasingly its use, from wireless thermometers or sensors for measuring blood pressure devices to sports watches with GPS or sensors for runners.

The work proposed here consist of a study of the Bluetooth LE specification deepen the operation of its entire protocol stack and its use to create applications.

Finally, as an example, an application for measuring the distance from a beacon using the reception power of the BLE packets transmitted by it will be developed.

Keywords: Bluetooth Low Energy, Protocols, CC2540/41, Applications Development, Wireless Transmission.

Índice general

Índice de figuras

Índice de tablas

Índice de listados de código fuente

Lista de Acrónimos

ATT	Attribute Protocol
ADV	Advertising Packet
BLE	Bluetooth Low Energy
GATT	General Attribute Profile
GAP	General Access Profile
L2CAP	Logical Link Control and Adaptation Protocol
SIG	Special Interest Group

Capítulo 1

Introducción

1.1 ¿Qué es Bluetooth Low Energy?

Es una tecnología de comunicación inalámbrica de bajo consumo que permite el intercambio de información entre dispositivos a cortas distancias.

Bluetooth Low Energy (**BLE**), creada por Bluetooth Special Interest Group (**SIG**) ha sido desarrollada como una tecnología complementaria al Bluetooth clásico con el fin de garantizar el bajo consumo, bajo coste, bajo ancho de banda y poca complejidad.

1.2 Antecedentes

Junio de 2010: Bluetooth SIG lanza en su versión 4.0 las especificaciones de Bluetooth Low Energy.

Diciembre de 2013: llega la actualización **Bluetooth 4.1**, actual referencia para cualquier desarrollo de productos BLE.

1.3 Especificaciones

Las especificaciones de Bluetooth (4.0 y superiores) definen dos tecnologías inalámbricas diferentes:

- **BR/EDR** (Bluetooth clásico): El estándar inalámbrico que ha ido evolucionando con las especificaciones de Bluetooth desde la versión 1.0.
- **BLE** (Bluetooth Low Energy): El estándar inalámbrico de bajo consumo introducido en la versión 4.0 de la especificación.

En el mercado podemos encontrar dispositivos que hacen uso de una sola o de ambas tecnologías.(modo dual).

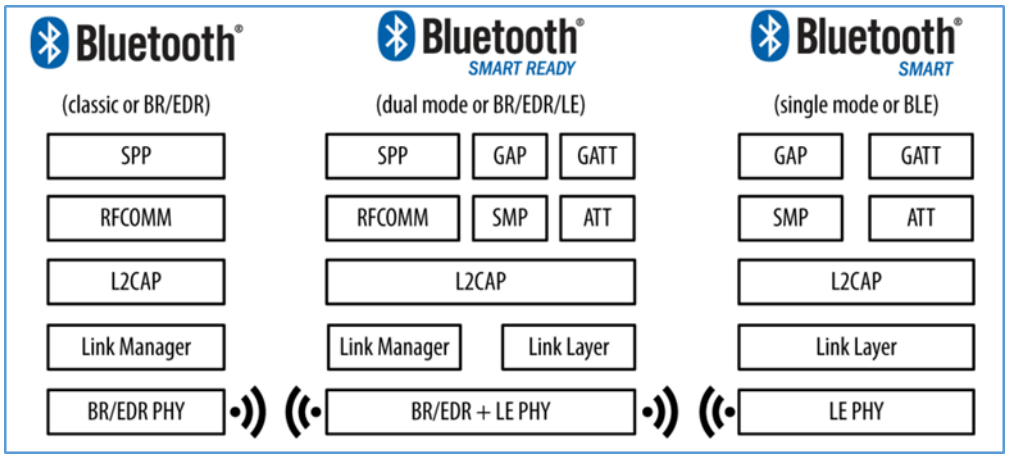


Figura 1.1: Diferentes estándares Bluetooth.

Los dos estándares de comunicación no son directamente compatibles, cualquier dispositivo Bluetooth anterior a la especificación 4.0 no puede comunicarse con un dispositivo BLE.

1.4 Topología de la red

Existen dos maneras mediante las que un dispositivo BLE puede comunicarse con otro, difusión (Broadcast) o conexión.

1.4.1 Difusión o Broadcast

Consiste en el envío de datos unidireccionalmente hacia cualquier dispositivo que esté dispuesto a escuchar. Se utiliza para enviar datos a más de un dispositivo a la vez.

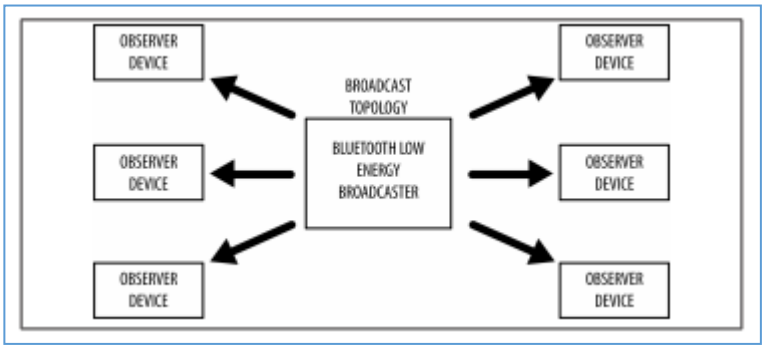


Figura 1.2: Esquema Broadcast.

[1.1],[1.2] Getting started with BLE. O´reilly[1]

En el modo difusión o broadcast hay dos roles definidos:

- **Difusor:** Envía periódicamente paquetes de anuncio (advertising packets) a cualquier dispositivo que pueda estar dispuesto a recibirlos.
- **Observador:** Continuamente escanea unas frecuencias preestablecidas con el fin de recibir cualquier paquete de anuncio que pueda estar siendo emitido en ese momento.

1.4.2 Conexión

Una conexión es un intercambio permanente y periódico de paquetes de datos sólo disponibles para los dos dispositivos involucrados en ella.

En el modo conexión hay definidos dos roles:

- **Periférico** (Slave): Difunde periódicamente paquetes de anuncio indicando su disponibilidad para aceptar conexiones y acepta las peticiones de conexión entrantes, una vez lograda la conexión intercambia datos con el dispositivo Central según la frecuencia dispuesta por éste.
- **Central** (Master): Continuamente escanea las frecuencias preestablecidas buscando paquetes de anuncio orientados a conexión y, cuando es posible, la inicia. Una vez que se ha logrado la conexión establece la frecuencia de intercambio de datos.

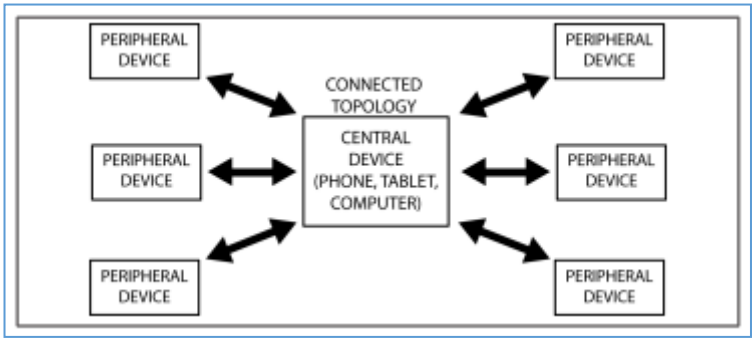


Figura 1.3: Esquema Conexión.

[1.3] Getting started with BLE. O´reilly[1]

1.5 Protocolos y perfiles

El stack de protocolos de bluetooth define como debe ser tratada la información para que pueda ser eficazmente transmitida de un dispositivo a otro, implementa aspectos como el formato de los paquetes, su multiplexación o la codificación y decodificación de los mismos.

Los perfiles definen cómo deben ser usados éstos protocolos para lograr un fin específico, existen dos tipos:

1.5.1 Perfiles Genéricos

Definidos en las especificaciones de SIG

- General Acces Profile (**GAP)**: define los roles, procedimientos y modos que permiten a los dispositivos descubrirse entre ellos, enviar paquetes de anuncio periódicamente (Difusión) y establecer y gestionar conexiones.
- General Attribute Profile (**GATT)**: define los procedimientos básicos que permiten a los dispositivos leer, escribir y descubrir elementos de datos entre ellos.

1.5.2 Perfiles Específicos

Son perfiles enfocados a un tipo de servicio muy específico, como el perfil de glucosa (transfiere el nivel de glucosa en sangre vía BLE) o el perfil de temperatura corporal. Son desarrollados tanto por SIG como por terceros y su funcionamiento está basado en el GATT.

Capítulo 2

Protocolos

La pila de protocolos de todo dispositivo BLE puede ser dividida en 3 partes diferenciadas: controlador, host y capa de aplicación.

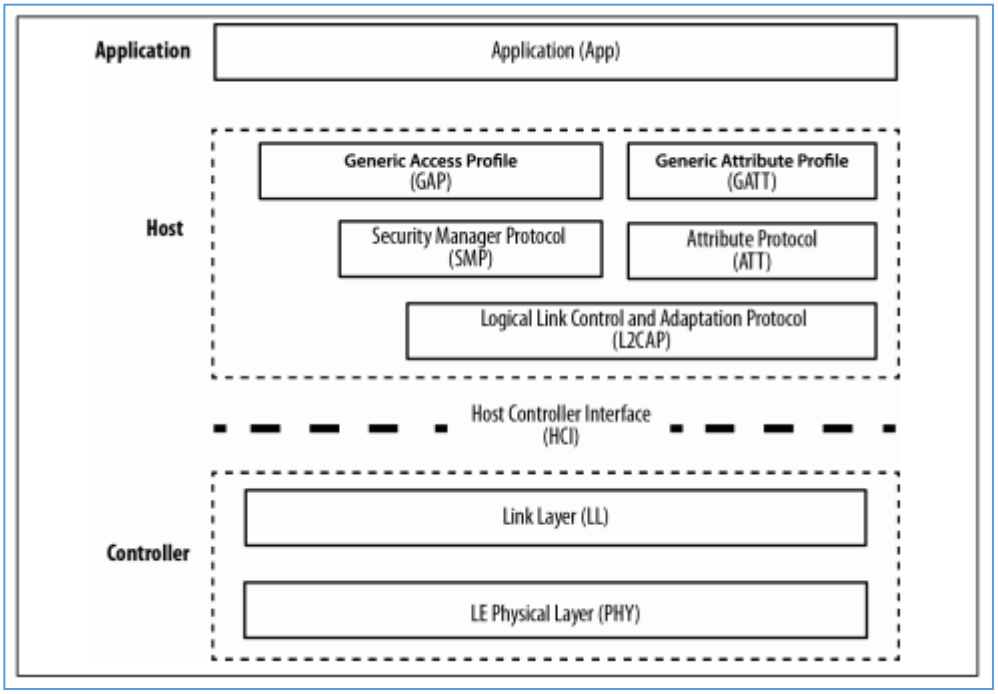


Figura 2.1 Esquema Stack Protocolos.

2.1 Controlador

Incluye las siguientes tres capas

- Capa física
- Capa de enlace
- Interfaz Host-Controlador (HCI)

2.1.1 Capa física

Contiene los circuitos capaces de modular y demodular las señales de radio.

Las comunicaciones se realizan en la banda de 2,4GHz dividiéndose en 40 canales, de 2,4GHz a 2,4835GHz siendo 37 de ellos dedicados a la transmisión de datos de conexión y 3 al envío de paquetes de anuncio y establecimiento de conexiones.

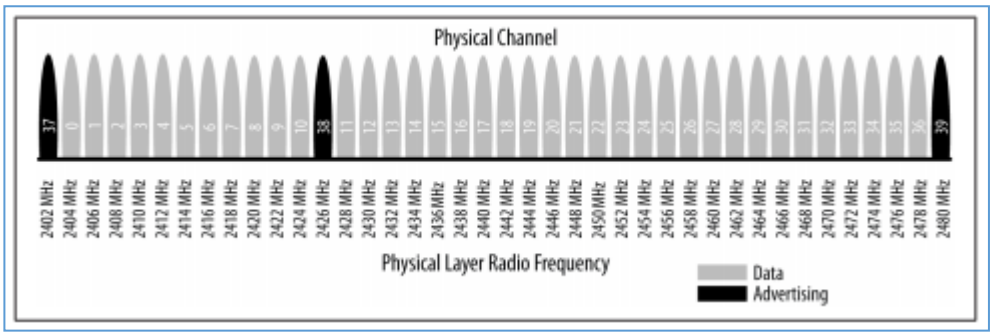


Figura 2.2: Canales BLE.

Para la elección del canal en el que se va a realizar la comunicación se emplea la técnica del salto de frecuencia (frequency hopping) siguiendo la siguiente fórmula:

$$canal = (canal_{actual} + salto) mod\ 37$$

El valor del campo salto se decide durante el establecimiento de la conexión y por lo tanto es diferente para cada una de las posibles conexiones. Esta técnica se utiliza para minimizar el efecto de cualquier interferencia de radio potencialmente presente en cualquiera de los canales de la banda de 2,4GHz, especialmente de WiFi y Bluetooth classic.

2.1.2 Capa de enlace

Es la responsable de difundir datos, escanear y crear y mantener conexiones. También es responsable de la estructura de los paquetes.

[2.2] Getting started with BLE. O´reilly [1]

Su funcionamiento es el de una máquina con 5 estados:

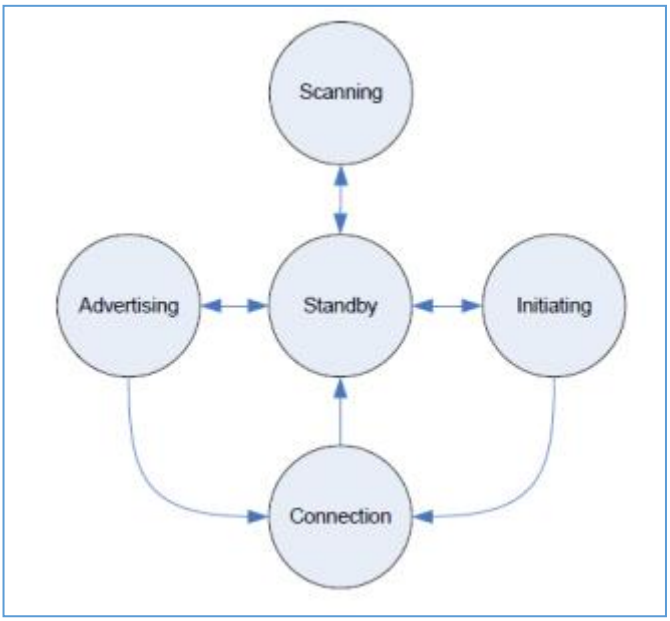


Figura 2.3 Máquina de estados GAP.

Standby: No se transmiten ni reciben ningún tipo de paquetes.

Advertising: Difunde continuamente paquetes *advertising* en los canales preestablecidos sin saber si serán recibidos por alguien. Cuanto mayor sea la frecuencia de emisión mayor será la probabilidad de que un dispositivo que se encuentre escaneando en ese canal reciba el paquete.

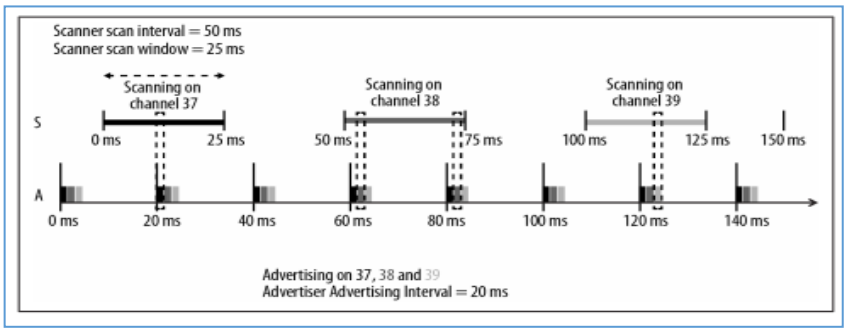


Figura 2.4: Ventanas de escaneo.

[2.4] Getting started with BLE. O´reilly [1]

Scanning: Busca paquetes *advertising*

- Escaneo pasivo: El dispositivo escucha en búsqueda de paquetes *advertising*.
- Escaneo activo: El dispositivo escucha en búsqueda de paquetes advertising y, tras recibir uno envía un Scan Request packet al anunciante, respondiéndole éste con un Scan Response packet, lo cual duplica la carga útil que el anunciante es capaz de enviar.

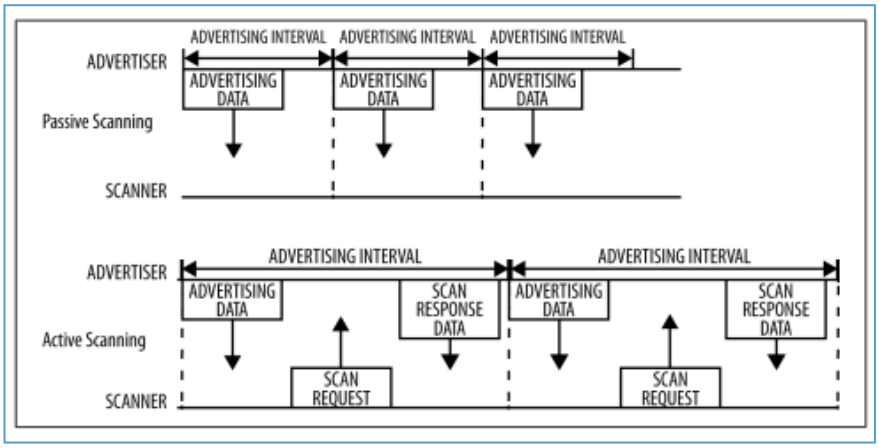


Figura 2.5: Esquema escaneo.

Initiating: Inicia una conexión con un anunciante.

Conexión: es el intercambio de datos entre un esclavo y un maestro en unos tiempos predeterminados, a cada uno de los intercambios se le denomina “evento de conexión”.

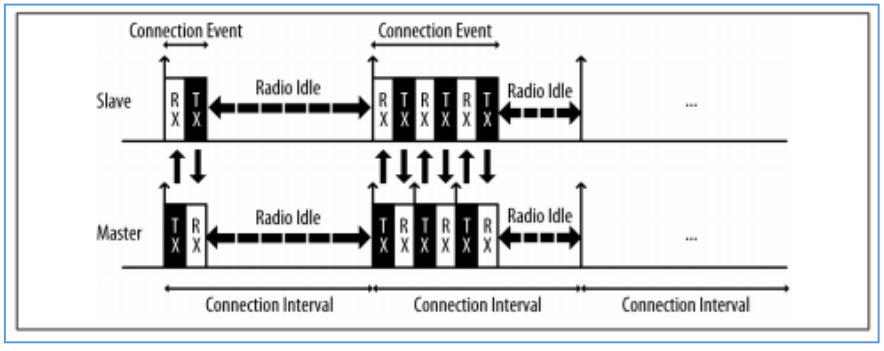


Figura 2.6: Eventos de conexión.

Para que se produzca una conexión entre dos dispositivos, el iniciador tiene que recibir un paquete *advertising* orientado a conexión y enviar una solicitud al anunciante donde se le indicarán los parámetros de esa conexión como la latencia, el intervalo de conexión y el timeout así como el salto de frecuencia que van a utilizar.

[2.5],[2.6] Getting started with BLE. O´reilly [1]

Una vez que el esclavo ha recibido el paquete de solicitud de conexión se considera que ambos dispositivos están conectados.

- **Intervalo de conexión:** Es el tiempo transcurrido desde el inicio de dos eventos de conexión consecutivos
- **Latencia:** Número de eventos de conexión que un esclavo puede decidir no atender antes de que se produzca una desconexión.
- **Timeout:** Tiempo máximo que puede transcurrir entre dos paquetes válidos recibidos para que no se pierda la conexión.

2.1.3 Estructura de un paquete BLE

Su función es el intercambio de información tanto bidireccional como unidireccional entre dispositivos.

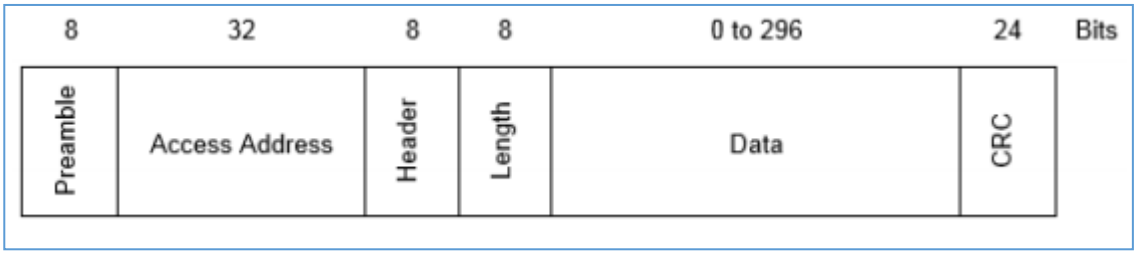


Figura 2.7: Esquema paquete BLE.

Preámbulo: Los primeros 8 bits son, o bien 01010101 o 10101010

Access Adress: Hay dos tipos:

- Advertising Access Adress: Se utiliza para difundir datos, su valor siempre es 0x8E89BED6
- Data Access Adress: Se utiliza en las conexiones y su valor varía para cada una de ellas.

Cabecera: Su valor depende del tipo de paquete, en los paquetes de anuncio indica el tipo de paquete que es.

Longitud (length): Indica la longitud del campo de datos del paquete.

Data (payload): Son los datos transmitidos, puede ser datos de anuncio, un Scan Response o bien datos de usuario enviados entre dos dispositivos conectados. Un paquete de anuncio tiene una carga útil de 31 bytes

CRC: Detección de errores.

[2.7] Wikipedia

2.1.4 Tipos de paquetes BLE

La cabecera de un paquete *advertising* nos da información sobre sus propiedades: conectividad, escaneabilidad y direccionalidad.

2.1.4.1 Conectividad

Según su capacidad de conexión los paquetes *advertising* pueden ser conectables o no conectables:

- **Conectables:** Un Scanner puede iniciar una conexión después de recibir de otro dispositivo un paquete *advertising* que sea conectable.
- **No conectable:** El Scanner no puede iniciar una conexión, estos paquetes *advertising* se usan principalmente para la difusión de datos.

2.1.4.2 Escaneabilidad

- **Escaneable:** Un Scanner podría emitir una petición de respuesta de escaneo (Scan Response) al recibir el paquete *advertising*.
- **No escaneable:** No se puede emitir una petición de respuesta de escaneo.

2.1.4.3 Direccionabilidad

- **Directo:** Dirigido a un Scanner en concreto, su carga útil contiene la dirección de ese dispositivo. Todo paquete *advertising* directo es por lo tanto conectable.
- **Indirecto:** Un paquete *advertising* indirecto no está dirigido a ningún Scanner en concreto y puede contener datos de usuario en su carga útil.

2.1.5 Ejemplo de declaración de un paquete advertising en el stack de TI

```
static uint8 advertData[] =
{
  // Flags; this sets the device to use limited discoverable
  // mode (advertises for 30 seconds at a time) instead of general
  // discoverable mode (advertises indefinitely)
  0x02,   // length of this data
  GAP_ADTYPE_FLAGS,
  DEFAULT_DISCOVERABLE_MODE | GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED,

  // service UUID, to notify central devices what services are included
  // in this peripheral
  0x03,   // length of this data
  GAP_ADTYPE_16BIT_MORE,      // some of the UUID's, but not all
  LO_UINT16( SIMPLEPROFILE_SERV_UUID ),
  HI_UINT16( SIMPLEPROFILE_SERV_UUID ),

};
```

Declaración paquete Advertising.

Éste paquete advertising corresponde a un dispositivo en modo periférico, sus dos primeros bytes nos indican mediante el uso de flags las características de conexión del dispositivo y los 3 últimos bytes los servicios disponibles.

Una vez que un dispositivo central ha recibido un paquete de anuncio puede solicitar un paquete de respuesta adicional (Scan Response packet) donde el periférico le enviaría datos complementarios como puede ser su nombre o los intervalos de conexión deseados.

A continuación se muestra la declaración de un paquete de scan response en el stack de TI:

```
static uint8 scanRspData[] =
{
  // nombre completo
  0x14,   // longitud del dato
  GAP_ADTYPE_LOCAL_NAME_COMPLETE,
  0x53,   // 'S'
  0x69,   // 'i'
  0x6d,   // 'm'
  0x70,   // 'p'
  0x6c,   // 'l'
  0x65,   // 'e'
  0x42,   // 'B'
  0x4c,   // 'L'
  0x45,   // 'E'
  0x50,   // 'P'
  0x65,   // 'e'
  0x72,   // 'r'
  0x69,   // 'i'
  0x70,   // 'p'
  0x68,   // 'h'
  0x65,   // 'e'
  0x72,   // 'r'
  0x61,   // 'a'
  0x6c,   // 'l'
// rango de intervalo de conexion
  0x05,   // tamaño del dato
  GAP_ADTYPE_SLAVE_CONN_INTERVAL_RANGE,LO_UINT16( DEFAULT_DESIRED_MIN_CONN_INTERVAL ),   // 100ms
  HI_UINT16( DEFAULT_DESIRED_MIN_CONN_INTERVAL ),LO_UINT16( DEFAULT_DESIRED_MAX_CONN_INTERVAL ),   // 1s
  HI_UINT16( DEFAULT_DESIRED_MAX_CONN_INTERVAL ),
  // Tx power level
  0x02,   // tamaño del dato
  GAP_ADTYPE_POWER_LEVEL,
  0       // 0dBm
};
```

Declaración paquete Scan Response.

Los 14 primeros bytes están dedicados al nombre del dispositivo codificado en hexadecimal, los 5 siguientes a los intervalos de conexión deseados por el periférico y los 2 últimos a su potencia de transmisión.

2.2 Host-Controller Interface

El HCI es un protocolo estándar que permite la comunicación entre host y controlador a través de una interfaz serie como puede ser UART, USB o SDIO.

2.3 Host

2.3.1 Logical Link Control and Adaptation Protocol (L2CAP)

Sirve como un protocolo de multiplexación, encapsula varios protocolos de las capas superiores en paquetes de BLE estándar (y viceversa).

También toma paquetes grandes de las capas superiores y los fracciona en pedazos que quepan en la carga útil máxima de 27 bytes de los paquetes BLE, en este sentido se podría decir que L2CAP es similar al protocolo TCP.

2.3.2 Security Manager

Provee a la pila de protocolos de Bluetooth la capacidad de generar e intercambiar llaves de seguridad.

2.3.3 Protocolo de Atributos (ATT)

Sigue una arquitectura cliente-servidor. El servidor tiene datos guardados en forma de atributos que el cliente puede escribir y leer.

El protocolo de atributos define cuatro campos de datos:

- **Handle:** Dirección de un atributo individual.
- **Type:** UUID, define el significado del valor.
- **Value:** El valor efectivo del dato.
- **Permission:** Indica si el atributo puede ser leído/escrito.

El protocolo también define 6 tipos de mensajes:

- **Request**: El cliente pide algo al servidor
- **Response:** El servidor envía un mensaje de respuesta ante una petición de un cliente.
- **Command:** El cliente envía una orden al servidor sin esperar una respuesta.
- **Notification:** El servidor notifica al cliente de un nuevo valor de un atributo (Sin confirmación)
- **Indication:** El servidor indica al cliente un nuevo valor, normalmente con confirmación.
- **Confirmation:** El cliente envía una confirmación ante una indicación del servidor.

2.3.4 Perfíl General de Atributos (GATT)

Define como debe usarse el ATT es decir, como se organizan los datos y como se intercambian entre aplicaciones.

2.3.5 Perfil General de Acceso (GAP)

Define los roles de broadcaster, observer, periférico y central, a su vez define los procedimientos de descubrimiento y establecimiento de conexiones.

Capítulo 3

GAP

El Perfil de Acceso General (GAP) establece las normas que deben seguir los dispositivos para poder descubrirse, difundir datos y establecer conexiones

Como ya se vio anteriormente el GAP define los siguientes aspectos de la interacción entre dispositivos:

3.1 Roles

Un dispositivo puede operar con uno o más roles a la vez, cada rol impone unas restricciones y define unos comportamientos. Ciertas combinaciones de estos roles son las que permiten a los dispositivos comunicarse y es el GAP el encargado de definir esas relaciones.

GAP especifica cuatro roles que puede adoptar un dispositivo para unirse a una red BLE:

- **Broadcaster:** Optimizado para aplicaciones que solamente transmiten paquetes con el fin de distribuir datos regularmente. Un dispositivo configurado como broadcaster envía periódicamente paquetes *advertising* con datos, éstos datos son accesibles a cualquier dispositivo que esté escuchando.
- **Observer:** Un dispositivo configurado como Observer se mantiene a la escucha para recibir datos dentro de paquetes *advertising* provenientes de otro dispositivo en modo broadcaster.
- **Central:** Un dispositivo Central recibe los paquetes *advertising* de otros dispositivos para después iniciar una conexión con el dispositivo deseado.
- **Peripheral:** Un dispositivo en rol Peripheral utiliza paquetes *advertising* para permitir a los dispositivos Central encontrarle y establecer una conexión con él.

El rol de un dispositivo se configura durante la inicialización del mismo.

3.2 Configuración del GAP

Las funcionalidades del GAP están definidas en el código de las librerías. Las cabeceras de sus funciones se encuentran en el archivo **gap.h**.

A la hora de desarrollar una aplicación utilizando el stack de TI no es necesario que se programe la inicialización del GAP, utilizando las plantillas facilitadas para cada rol simplemente hay que agregarlas al proyecto y configurar sus parámetros directamente desde la aplicación.

Los parámetros de la conexión se pueden modificar u obtener a través de las funciones **GAP_SetParamValue()** y **GAP_GetParamValue()**.

```
// Set the GAP Role Parameters
GAPRole_SetParameter( GAPROLE_ADVERT_ENABLED, sizeof( uint8 ), &initial_advertising_enable );
GAPRole_SetParameter( GAPROLE_ADVERT_OFF_TIME, sizeof( uint16 ), &gapRole_AdvertOffTime );

GAPRole_SetParameter( GAPROLE_SCAN_RSP_DATA, sizeof ( scanRspData ), scanRspData );
GAPRole_SetParameter( GAPROLE_ADVERT_DATA, sizeof( advertData ), advertData );

GAPRole_SetParameter( GAPROLE_PARAM_UPDATE_ENABLE, sizeof( uint8 ), &enable_update_request );
GAPRole_SetParameter( GAPROLE_MIN_CONN_INTERVAL, sizeof( uint16 ), &desired_min_interval );
GAPRole_SetParameter( GAPROLE_MAX_CONN_INTERVAL, sizeof( uint16 ), &desired_max_interval );
GAPRole_SetParameter( GAPROLE_SLAVE_LATENCY, sizeof( uint16 ), &desired_slave_latency );
GAPRole_SetParameter( GAPROLE_TIMEOUT_MULTIPLIER, sizeof( uint16 ), &desired_conn_timeout );
```

Setup parámetros conexión.

Mediante el uso de la función GAP_SetParamValue() se definen tanto el contenido de los paquetes de anuncio y scan response como los siguientes parámetros de conexión:

3.2.1 Parámetros de conexión

3.2.1.1 Connection Interval

Intervalo de conexión, es la cantidad de tiempo que pasa entre dos eventos de conexión, el valor se representa en unidades de 1.25 ms y su rango puede oscilar entre 6 (7.5ms) y 3200 (4.0s). Diferentes aplicaciones pueden requerir diferentes intervalos de conexión. Un intervalo muy largo conlleva un ahorro de energía pues el dispositivo puede entrar en sleep mode entre eventos de conexión, pero si tiene nuevos datos que mandar tendrá que esperar al próximo evento. Un intervalo de conexión muy corto tiene la ventaja de aumentar la oportunidad de enviar y recibir datos, pero eso conlleva un mayor consumo de energía.

3.2.1.2 Slave Latency

Éste parámetro da la opción al esclavo de saltarse un número determinado de eventos de conexión, si no tiene datos que mandar puede permanecer en sleep mode ahorrando energía sin perderse la conexión. El valor de Slave Latency representa el máximo número de eventos de conexión que se pueden ignorar, siendo su valor mínimo 0 (No se puede ignorar ningún evento) y su valor máximo 499; no obstante, el valor máximo no puede hacer que el intervalo efectivo de conexión supere los 16s.

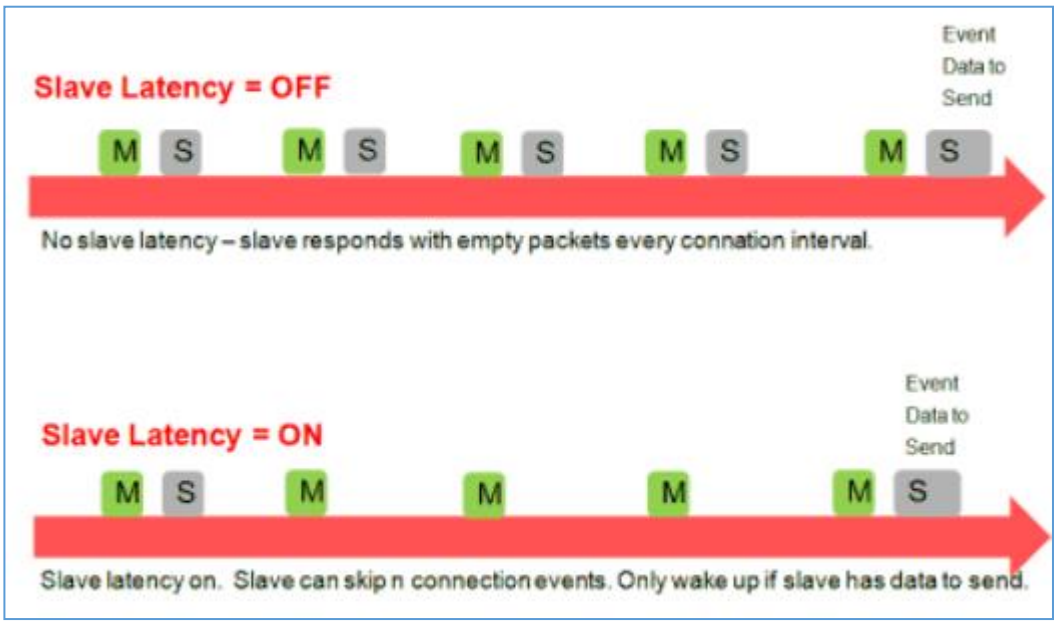


Figura 3.1: Slave Latency.

$$Effective\ Connection\ Interval = (Connection\ Interval) * (1 + Slave\ Latency))$$

3.2.1.3 Supervision Timeout

Es la cantidad máxima de tiempo entre dos eventos de conexión con éxito. Si se supera ésta cantidad el dispositivo considerará la conexión como perdida y volverá a un estado de no-conexión. El valor se representa en unidades de 10ms y su rango puede ir desde 10 (100ms) a 3200 (32s). El Timeout Tendrá que ser siempre mayor que el intervalo efectivo de conexión.

[3.1] Guía referencia CC2541 [2]

La elección de estos parámetros de conexión juega un papel muy importante en la optimización del consumo de un dispositivo BLE. A continuación se muestra un resumen general sobre las consecuencias de la elección de los parámetros:

Reducir el intervalo de conexión conlleva:

- Un incremento del consumo de potencia en los dos dispositivos.
- Aumento del rendimiento (throughput) en ambas direcciones
- Reducción del tiempo necesario para enviar datos de un lado a otro.

Aumentar el intervalo de conexión conlleva:

- Reducir el consumo de potencia en los dos dispositivos.
- Reducir el rendimiento (troughput) en ambas direcciones.
- Aumentar el tiempo necesario para enviar datos de un lado a otro.

Reducir el Slave Latency conlleva:

- Aumentar el consumo del dispositivo en rol Peripheral.
- Reducir la cantidad de tiempo que tarda un dato enviado por un Central en ser recibido por un Peripheral.

Aumentar el Slave Latency conlleva:

- Reducir el consumo de potencia del Peripheral durante aquellos periodos en los que no tiene datos que enviar al Central.
- Incrementar la cantidad de tiempo que tarda un dato enviado por un Central en ser recibido por el Peripheral.

3.2.2 Configuración del GAP en rol Peripheral

El rol Peripheral permite al dispositivo anunciarse, conectarse con un central y solicitar una serie de parámetros de conexión al dispositivo maestro.
Los prototipos de las funciones del API del perfil pheripheral pueden encontrarse en el archivo **pheripheral.h**. El API facilita dos funciones para obtener y configurar los parámetros del GAP.

Algunos de esos parámetros son:

GAPROLE_ADVERT_ENABLED: Habilita o deshabilita la difusión, su valor por defecto es TRUE

GAPROLE_ADVERT_DATA: Array que contiene los datos a enviar en los paquetes de anuncio.

GAPROLE_SCAN_RSP_DATA: Array que contiene los datos a enviar en los paquetes de Scan Response, se envía como respuesta a un observer o central que lo solicite.

GAPROLE_ADVERT_OFF_TIME: Indica cuánto tiempo ha de pasar para que un dispositivo que está en modo descubrimiento limitado vuelva a ser descubrible. Si su valor es 0 el dispositivo no volverá a anunciarse hasta que el parámetro **GAPROLE_ADVERT_ENABLED** vuelva a ser **TRUE**.

GAPROLE_PARAM_UPDATE_ENABLE: Habilita y deshabilita la solicitud de actualización automática de parámetros.

GAPROLE_MIN_CONN_INTERVAL: Valor mínimo deseado de intervalo de conexión, su valor por defecto es 80 (100ms).

GAPROLE_MAX_CONN_INTERVAL: Valor máximo deseado de intervalo de conexión, su valor por defecto es 3200 (4s).

GAPROLE_SLAVE_LATENCY: Valor deseado de Slave Latency, su valor por defecto es 0.

GAPROLE_TIMEOUT_MULTIPLIER: Valor deseado del Timeout, su valor por defecto es 1000(10s).

Para iniciar el funcionamiento del GAP hay que llamar, una vez configurados los parámetros, una sola vez a la función **GAPRole_StartDevice();**

También existe un parámetro para la habilitación de la actualización automática de parámetros, ajustando el parámetro **GAPROLE_PARAM_UPDATE_ENABLE** a **TRUE**. Si ésta característica está habilitada y el peripheral inicia una conexión cuyos parámetros están fuera de sus valores deseados, puede solicitar al central una actualización de los mismos. Siempre que esos parámetros sean acordes a la especificación BLE el central debe aceptarlos y cambiarlos.

Los parámetros de conexión deseados son fijados en la función **SimpleBLEPeripheral_Init();** y su valor puede ser cambiado fácilmente.

```
// Minimum connection interval (units of 1.25ms, 80=100ms) if automatic parameter update request is enabled
#define DEFAULT_DESIRED_MIN_CONN_INTERVAL     80

// Maximum connection interval (units of 1.25ms, 800=1000ms) if automatic parameter update request is enabled
#define DEFAULT_DESIRED_MAX_CONN_INTERVAL     800

// Slave latency to use if automatic parameter update request is enabled
#define DEFAULT_DESIRED_SLAVE_LATENCY         0

// Supervision timeout value (units of 10ms, 1000=10s) if automatic parameter update request is enabled
#define DEFAULT_DESIRED_CONN_TIMEOUT          1000

// Whether to enable automatic parameter update request when a connection is formed
#define DEFAULT_ENABLE_UPDATE_REQUEST         TRUE
```

Parámetros de conexión

3.2.3 Configuración del GAP en multi-rol Peripheral Broadcaster

El rol múltiple peripheral/broadcaster funciona prácticamente igual que el rol pheripheral aunque añade funcionalidades adicionales permitiendo al dispositivo operar en esos dos roles del GAP simultáneamente. Para utilizar esta multifuncionalidad hay que excluir los archivos **peripheral.c** y **pheripheral.h** y añadir los archivos **peripheralBroadcaster.c** y peripheralBroadcaster.h, además hay que definir el valor de preprocesado **PLUS_BROADCASTER**.

3.2.4 Configuración del GAP en rol Central

El rol central permite al dispositivo descubrir dispositivos que se estén anunciando, establecer conexiones y actualizar los parámetros de esas conexiones.

Las funciones principales del perfil de rol central se pueden encontrar en el archivo **central.h**, las funciones utilizadas para obtener y establecer parámetros son **GAPCentralRole_GetParameter();** y **GAPCentralRole_SetParameter();** respectivamente.

Para permitir la actualización de parámetros durante una conexión hay que establecer **GAPROLE_PARAM_UPDATE_ENABLE** a **TRUE** tal y como se configuró en el rol peripheral.

Capítulo 4

GATT

El perfil de atributos genérico (GATT) establece en detalle como intercambiar datos de usuario a través de una conexión BLE. Mientras que el GAP define la relación a bajo nivel entre dispositivos, el GATT se encarga estrictamente de los procedimientos de transferencia de datos.

Todo dato relevante para las aplicaciones o los usuarios debe tener un formato y ser empaquetado y enviado de acuerdo con las reglas del GATT.

4.1 Roles

En GATT, cuando dos dispositivos están conectados, cada uno toma uno de los siguientes roles:

Cliente: Escribe y lee datos del servidor.

Servidor: Recibe peticiones de los clientes y les responde, es el responsable de almacenar y hacer que los datos estén disponibles para los clientes, organizándolos en atributos.

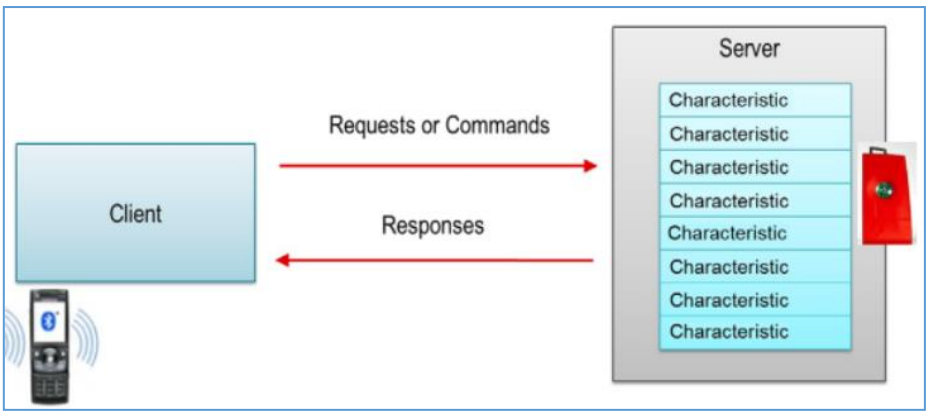


Figura 4.1 Esquema Cliente-Servidor.

Típicamente, los roles del GATT de cliente y servidor son independientes de los roles del GAP, periférico y central. Un Peripheral puede ser tanto un cliente GATT o un servidor GATT, lo mismo ocurre con un dispositivo Central. Un dispositivo puede también actuar como un cliente y un servidor a la vez.

4.2 Características y atributos del GATT

Los atributos son la entidad de datos más pequeña definida por GATT, son unidades de información direccionables que contienen datos de usuario. Toda información debe ser organizada de este modo. Las características son agrupaciones de datos en forma de atributos.

Una característica está formada por los siguientes atributos:

- **Characteristic Value:** El valor del dato.
- **Characteristic Declaration:** Almacena las propiedades, localización y tipo del valor de la característica.
- **Client Characteristic Configuration:** Permite al cliente o bien enviar la característica al cliente sin más (notificación) o enviar y esperar una confirmación.
- **Characteristic User Description:** Cadena ASCII que describe la característica.

Éstos atributos son almacenados en el servidor GATT dentro de una tabla de atributos, asociadas a cada atributo encontramos las siguientes propiedades:

Handle: identificador único de 16 bits para cada uno de los atributos de un servidor GATT, es la parte de cada atributo que lo hace direccionable, el número de handles disponibles en cada servidor GATT es de 0xFFFE (65535).

Type: Indica qué representa el dato del atributo. Su valor es un UUID (Universal Unique Identifier), algunos de los UUIDs son definidos por el SIG y otros por el usuario.

Permissions: Indica si un cliente puede acceder al dato del servidor y si es así, de qué manera.

- **NONE:** El atributo no puede ser leído ni escrito por el cliente.
- **READABLE:** Puede ser leído por el cliente.
- **WRITEABLE:** Puede ser escrito por el cliente.
- **READABLE AND WRITEABLE:** Puede ser leído y escrito por el cliente.

Value: Valor del dato en cuestión.

Handle	Type	Permissions	Value	Value length
0x0201	$UUID_1$ (16-bit)	Read only, no security	0x180A	2
0x0202	$UUID_2$ (16-bit)	Read only, no security	0x2A29	2
0x0215	$UUID_3$ (16-bit)	Read/write, authorization required	"a readable UTF-8 string"	23
0x030C	$UUID_4$ (128-bit)	Write only, no security	{0xFF, 0xFF, 0x00, 0x00}	4
0x030D	$UUID_5$ (128-bit)	Read/write, authenticated encryption required	36.43	8
0x031A	$UUID_1$ (16-bit)	Read only, no security	0x1801	2

Figura 4.2: Atributos.

4.3 Servicios GATT

Un servicio GATT es una colección de características, por ejemplo: el servicio de ritmo cardiaco contiene una característica de la medida del ritmo cardiaco y una característica de localización del sensor en el cuerpo. Se pueden agrupar varios servicios para formar un perfil, muchos perfiles sólo cuentan con un solo servicio por lo que los términos servicio y perfil son intercambiables.

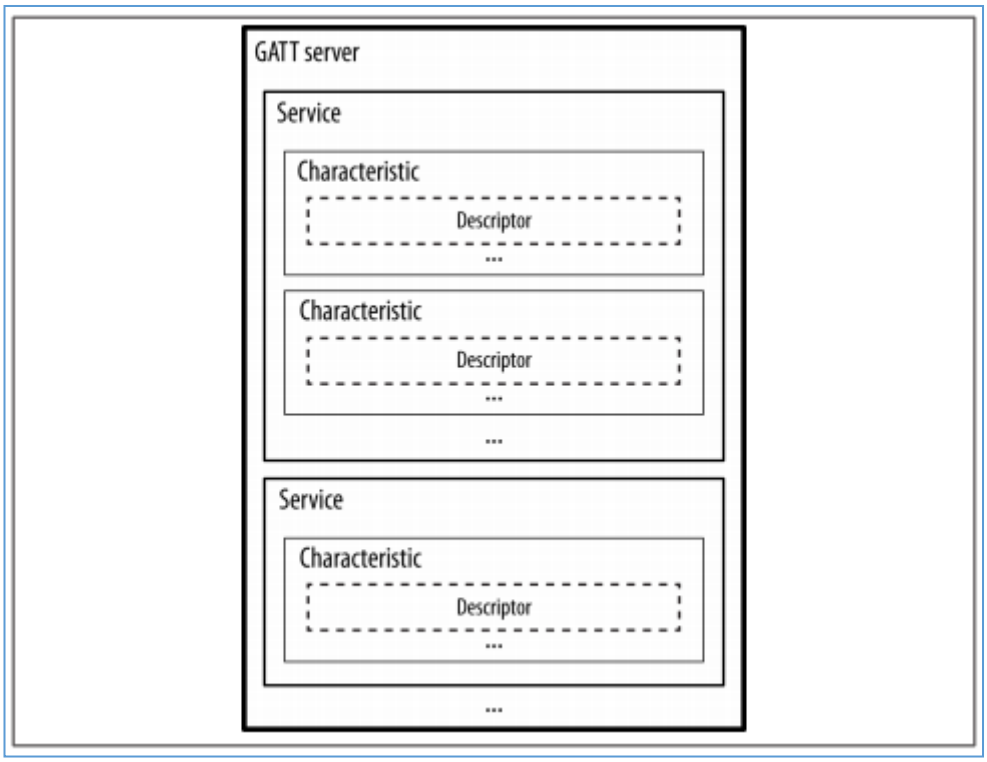


Figura 4.3: Servicios GATT.

[4.1],[4.2] Getting started with BLE. O´reilly [1]

Servicios: Los servicios de un servidor GATT agrupan datos en forma de características. Un servicio puede tener ninguna, una o más características. Cada servicio se diferencia de los demás a través de un UUID único de 16 bits.

Un servicio puede ser instanciado varias veces en un mismo servidor GATT.

Características: Las características son contenedores de datos, cada una contiene al menos dos atributos: La declaración de la característica y el valor de esa característica. Toda característica se encontrará siempre dentro de un servicio.

La aplicación SimpleBLEPeripheral tiene 4 servicios:

Mandatory GAP Service: Éste servicio contiene información como el nombre del dispositivo o identificador del fabricante. El servicio es parte del stack de protocolos de BLE y es obligatoria su presencia según la especificación de BLE. Su código fuente no es facilitado, está compilado dentro de la librería del stack.

Mandatory GATT Service: Éste servicio contiene información sobre el servidor GATT y también forma parte del stack de protocolos de BLE. Su presencia es obligatoria según la especificación de BLE y su código fuente tampoco es facilitado por el fabricante. Compilado en la librería del stack.

Device Information Service: Éste servicio contiene información del dispositivo como sus versiones de hardware, software y firmware o el nombre del fabricante. Es parte del stack y es configurado por la aplicación.

SimpleGATTProfile Service: Es un servicio de prueba para testear el funcionamiento del dispositivo. Su código fuente al completo se encuentra en los archivos **simpleGATTProfile.c** y **simpleGATTProfile.h**

4.4 Tabla de Atributos

A continuación se muestra la tabla de atributos del proyecto SimpleBLEPEripheral correspondiente al servicio SimpleGATTProfile:

Handle	Uuid	Uuid Description	Value	Properties
0x001F	0x2800	GATT Primary Service Declaration	F0:FF	
0x0020	0x2803	GATT Characteristic Declaration	0A:21:00:F1:FF	
0x0021	0xFFF1	Simple Profile Char 1	01	Rd Wr 0x0A
0x0022	0x2901	Characteristic User Description	Characteristic 1	
0x0023	0x2803	GATT Characteristic Declaration	02:24:00:F2:FF	
0x0024	0xFFF2	Simple Profile Char 2	02	Rd 0x02
0x0025	0x2901	Characteristic User Description	Characteristic 2	
0x0026	0x2803	GATT Characteristic Declaration	08:27:00:F3:FF	
0x0027	0xFFF3	Simple Profile Char 3		Wr 0x08
0x0028	0x2901	Characteristic User Description	Characteristic 3	
0x0029	0x2803	GATT Characteristic Declaration	10:2A:00:F4:FF	
0x002A	0xFFF4	Simple Profile Char 4		Nfy 0x10
0x002B	0x2902	Client Characteristic Configuration	00:00	
0x002C	0x2901	Characteristic User Description	Characteristic 4	
0x002D	0x2803	GATT Characteristic Declaration	02:2E:00:F5:FF	
0x002E	0xFFF5	Simple Profile Char 5		Rd 0x02
0x002F	0x2901	Characteristic User Description	Characteristic 5	

Tabla de atributos Simple Peripheral.

Primero se va a explicar el contenido de ésta tabla y posteriormente se explicará cómo configurar una tabla de atributos desde cero.

4.4.1 Contenido

En la tabla de atributos de SimpleGATTProfile observamos que el perfil contiene las siguientes 5 características:

SIMPLEPROFILE_CHAR1: Un valor de 1 byte que puede ser leído o escrito desde un dispositivo cliente.

SIMPLEPROFILE_CHAR2: Un valor de 1 byte que puede ser leído pero no escrito desde un dispositivo cliente.

SIMPLEPROFILE_CHAR3: Un valor de 1 byte que puede ser escrito pero no leído desde un dispositivo cliente.

SIMPLEPROFILE_CHAR4: Un valor de 1 byte que no puede ser leído o escrito directamente, puede ser configurado para enviar notificaciones a un dispositivo cliente.

SIMPLEPROFILE_CHAR5: Un valor de 5 bytes que puede ser leído pero no escrito desde un dispositivo cliente

.

A continuación se profundizará línea a línea en la tabla de atributos referenciando cada parámetro por su Handle:

0x001F: Este atributo sirve para declarar un nuevo servicio, su UUID está estandarizado (0x2800 correspondiente a **GATT_PRIMARY_SERVICE_UUID**). El valor del atributo se corresponde con el UUID del SimpleGATTProfile (Definido por TI, 0xFFF0).

0x0020: Este atributo sirve para declarar una característica **SIMPLEPROFILE_CHAR1**, puede entenderse la declaración de una característica como un puntero al valor de la propia característica. Su UUID está estandarizado y es 0x2803 correspondiente a **GATT_CHARACTER_UUID**.
El valor del atributo de declaración de característica consta de 5 bytes:

Byte 0: Propiedades de la característica **SIMPLEPROFILE_CHAR1**:
0x02: Lectura permitida del valor de la característica.
0x04: Escritura permitida del valor de la característica sin respuesta.
0x08: Escritura permitida del valor de la característica con respuesta.
0x10: Notificaciones del valor permitidas (Sin ACK).
0x20: Notificaciones del valor permitidas (Con ACK).
El valor 0xA0 significa que la característica puede ser leída (0x02) y escrita (0x08).

Bytes 1-2: Valor del Handle del valor de la característica **SIMPLEPROFILE_CHAR1**.

Bytes 3-4: UUID del valor de **SIMPLEPROFILE_CHAR1** (Definida por el usuario 0xFFF1).

0x0021: Este atributo es el valor de **SIMPLEPROFILE_CHAR1**. Su UUID es 0xFFF1 (definido por usuario). Este valor es el dato útil de la característica, tal y como indica su declaración de característica, se puede leer y escribir.

0x0022: Éste atributo es la descripción de usuario de **SIMPLEPROFILE_CHAR1**, su UUID es 0x290 (estandarizado). Su valor es una cadena de caracteres que describen la característica, en este caso: “Characteristic 1”.

0x0023-00x2F: Éstos atributos tienen la misma estructura que **SIMPLEPROFILE_CHAR1**.

0x002: Éste atributo es la configuración de característica de **SIMPLEPROFILE_CHAR4**, su UUID es 0x2902 (definida por bluetooth SIG). Escribiendo en este atributo el servidor GATT puede configurar las notificaciones (escribiendo 0x0001) o indicaciones (escribiendo 0x0002) de la **SIMPLEPROFILE_CHAR4**. Escribiendo 0x0000 se deshabilitan tanto las notificaciones como las indicaciones.

4.4.2 Creación de una tabla de Atributos

En este aparatado se va a crear la tabla de atributos correspondiente al ejemplo anterior paso a paso.

Recordemos que los atributos son la entidad de datos más pequeña definida por GATT, son unidades de información direccionables.

Todo atributo de la tabla deberá seguir la siguiente estructura (definida en gatt.h):

```
typedef struct attAttribute_t
{
  gattAttrType_t type; //!< Attribute type (2 or 16 octet UUIDs)
  uint8 permissions;   //!< Attribute permissions
  uint16 handle;       //!< Attribute handle - assigned internally by attribute server
  uint8* const pValue; //!< Attribute value - encoding of the octet array is defined in
                       //!< the applicable profile. The maximum length of an attribute
                       //!< value shall be 512 octets.
} gattAttribute_t;
```

Estructura atributo.

Sus elementos son los siguientes:

Type: Es el identificador (UUID) asociado a ese atributo, definido como:

```
typedef struct
{
  uint8 len;          //!< Length of UUID
  const uint8 *uuid; //!< Pointer to UUID
} gattAttrType_t;
```

Struct del UUID.

Su longitud (len) puede tener dos valores, **ATT_BT_UUID_SIZE** para atributos de 2 bytes o **ATT_UUID_SIZE** para atributos de 16 bits. *uuid es un puntero a un valor que puede ser definido por el usuario o un valor reservado definido por SIG.

Permissions: Éste valor indica de qué manera puede manejar el dato un cliente. Los posibles permisos están definidos en **gatt.h** y son los siguientes:

GATT_PERMIT_READ // El atributo se puede leer.
GATT_PERMIT_WRITE // El atributo se puede escribir.
GATT_PERMIT_AUTHEN_READ // La lectura requiere autentificación.
GATT_PERMIT_AUTHEN_WRITE // La escritura requiere autentificación.
GATT_PERMIT_AUTHOR_READ // La lectura requiere autorización.
GATT_PERMIT_ENCRYPT_READ // La lectura requiere encriptación.
GATT_PERMIT_ENCRYPT_WRITE // La lectura requiere encriptación.

Handle: Campo con el valor del handle. Se asignan secuencialmente por el servidor.

pValue: Puntero al valor del atributo.

4.4.2.1 Definición de un servicio

```
// Simple Profile Service
{
  { ATT_BT_UUID_SIZE, primaryServiceUUID }, /* type */
  GATT_PERMIT_READ,                         /* permissions */
  0,                                        /* handle */
  (uint8 *)&simpleProfileService            /* pValue */
},
```

Definición de un servicio.

Ésta es la definición del servicio de nuestro ejemplo (SimpleGATTProfile).

Type: UUID de 2 bytes, su identificador (UUID) es el correspondiente a un servicio primario (0x2800, definido por SIG).

Permissions: **GATT_PERMIT_READ**, la lectura de su valor está habilitada.

Handle: Sin valor, es asignado por el servidor GATT.

pValue: Puntero al valor del atributo, en este caso su valor es el UUID del servicio, definido por el usuario en **GATTprofile.h** (0xFFF0).

```
// Simple Profile Service UUID
#define SIMPLEPROFILE_SERV_UUID                0xFFF0
```

Identificador del nuevo servicio.

Una vez que tenemos definido un servicio podemos definir cuáles serán sus características.

4.4.2.2 Definición de una característica

El ejemplo consta de 5 características, se va a estudiar cómo definir una de ellas, como por ejemplo SIMPLEPROFILE_CHAR1:

Atributo de Declaración:

```
// Characteristic 1 Declaration
{
  { ATT_BT_UUID_SIZE, characterUUID },
  GATT_PERMIT_READ,
  0,
  &simpleProfileChar1Props
},
```

Declaración atributo.

Type: Es el UUID definido por SIG para una característica (characterUUID = 0x2803).

Permission: Un cliente debe poder leer el UUID, luego su valor es **GATT_PERMIT_READ**.

pValue: su valor se corresponde con las propiedades que tendrá la característica, en este caso son lectura y escritura permitidos.

```
// Simple Profile Characteristic 1 Properties
static uint8 simpleProfileChar1Props = GATT_PROP_READ | GATT_PROP_WRITE;
```

Propiedades característica.

Atributo de valor de una característica:

```
// Characteristic Value 1
{
  { ATT_BT_UUID_SIZE, simpleProfilechar1UUID },
  GATT_PERMIT_READ | GATT_PERMIT_WRITE,
  0,
  &simpleProfileChar1
},
```

Atributo de característica.

Type: Es el UUID definida por el usuario para la característica 1 (0xFFF1)

```
// Characteristic 1 UUID: 0xFFF1
CONST uint8 simpleProfilechar1UUID[ATT_BT_UUID_SIZE] =
{
  LO_UINT16(SIMPLEPROFILE_CHAR1_UUID), HI_UINT16(SIMPLEPROFILE_CHAR1_UUID)
};
```

UUID Caracteristica 1.

Estando su valor **SIMPLEPROFILE_CHAR1_UUID** definido en **simpleGATTProfile.h**:

```
#define SIMPLEPROFILE_CHAR1_UUID            0xFFF1
```

UUID Caracteristica 1.

Permission: Permitida lectura y escritura **GATT_PERMIT_READ | GATT_PERMIT_WRITE**

pValue: Puntero al valor real de la característica, está definido en el perfil:

```
// Characteristic 1 Value
static uint8 simpleProfileChar1 = 0;
```

Valor de característica.

Atributo de descripción de usuario:

```
// Characteristic 1 User Description
{
  { ATT_BT_UUID_SIZE, charUserDescUUID },
  GATT_PERMIT_READ,
  0,
  simpleProfileChar1UserDesp
},
```

User Description de Característica 1.

Da información al usuario sobre la naturaleza de la característica, como por ejemplo su nombre.

Type: Definido por SIG (charUserDescUUID = 0x2901).

Permission: Lectura permitida.

pValue: Puntero al valor que describe la característica, en este caso:

```
// Simple Profile Characteristic 1 User Description
static uint8 simpleProfileChar1UserDesp[17] = "Characteristic 1\0";
```

Valor del User Description.

Como ya vimos en **SIMPLEPROFILE_CHAR4** algunas características pueden requerir de una configuración adicional para manejar aspectos tales como el envío de notificaciones, para ello debemos añadir un atributo adicional con esa configuración:

Atributo de configuración de cliente de característica:

```
// Characteristic 4 configuration
{
  { ATT_BT_UUID_SIZE, clientCharCfgUUID },
  GATT_PERMIT_READ | GATT_PERMIT_WRITE,
  0,
  (uint8 *)simpleProfileChar4Config
},
```

Configuración Característica 4.

Type: Definido por SIG (clientCharCfgUUID = 0x2902).

Permissions: El cliente debe poder leer y escribir.

pValue: Puntero al array de configuración de clientes, la configuración es diferente para cada cliente:

```
// Simple Profile Characteristic 4 Configuration Each client has its own
// instantiation of the Client Characteristic Configuration. Reads of the
// Client Characteristic Configuration only shows the configuration for
// that client and writes only affect the configuration of that client.
static gattCharCfg_t simpleProfileChar4Config[GATT_MAX_NUM_CONN];
```

Array con información de cada conexión.

```
typedef struct
{
  uint16 connHandle; //!< Client connection handle
  uint8  value;      //!< Characteristic configuration value for this client
} gattCharCfg_t;
```

Estructura valor de configuración.

Recordemos que escribiendo en pValue el cliente puede configurar las notificaciones (0x0001) o las indicaciones (0x0002), escribiendo 0x0000 se deshabilitan ambas.

Una vez poblada la tabla de atributos (simpleProfileAttrTbl) debemos inicializarla en la aplicación:

```
// Initialize GATT attributes
GGS_AddService( GATT_ALL_SERVICES );            // GAP
GATTServApp_AddService( GATT_ALL_SERVICES );    // GATT attributes
DevInfo_AddService();                           // Device Information Service
SimpleProfile_AddService( GATT_ALL_SERVICES );  // Simple GATT Profile
```

Inicialización tabla atributos.

Siendo el valor que se pasa a la función el UUID del servicio a añadir o bien **GATT_ALL_SERVICES** (0xFFFFFFFF) para añadir todos los servicios existentes.

4.5 Definición de los callbacks

Los callbacks son funciones que utilizan los perfiles para pasar información a la aplicación. En éste proyecto el SimpleGATTProfile llama a una función callback siempre que un cliente GATT escribe o lee en un valor de una característica.

Estas funciones están definidas e inicializadas por la aplicación en **SimpleGATTProfile.c** y **SimpleBLEPeripheral.c**.

Solicitud de lectura de un cliente GATT: Cuando se produce una solicitud de lectura de un determinado atributo el stack de protocolos comprueba sus permisos y, si puede ser leída llama a la función callback del perfil, **simpleProfile_ReadAttrCB();**, definida en **SimpleGATTProfile.c**:

```
/*********************************************************************
 * @fn          simpleProfile_ReadAttrCB
 *
 * @brief       Read an attribute.
 *
 * @param       connHandle - connection message was received on
 * @param       pAttr - pointer to attribute
 * @param       pValue - pointer to data to be read
 * @param       pLen - length of data to be read
 * @param       offset - offset of the first octet to be read
 * @param       maxLen - maximum length of data to be read
 *
 * @return      Success or Failure
 */
static uint8 simpleProfile_ReadAttrCB( uint16 connHandle, gattAttribute_t *pAttr,
                                       uint8 *pValue, uint8 *pLen, uint16 offset, uint8 maxLen )
{
  bStatus_t status = SUCCESS;

  ...
    case SIMPLEPROFILE_CHAR1_UUID:
       *pLen = 1;
        pValue[0] = *pAttr->pValue;
        break;

  ...

    }
  }

  return ( status );
}
```

A1: Función callback de lectura de atributo.

Solicitud de escritura de un cliente GATT: Cuando se produce una solicitud de escritura de un determinado atributo el stack de protocolos comprueba sus permisos y, si puede ser escrita llama a la función callback del perfil, simpleProfile_WriteAttrCB, definida en SimpleGATTProfile.c:

```
/*********************************************************************
 * @fn      simpleProfile_WriteAttrCB
 *
 * @brief   Validate attribute data prior to a write operation
 *
 * @param   connHandle - connection message was received on
 * @param   pAttr - pointer to attribute
 * @param   pValue - pointer to data to be written
 * @param   len - length of data
 * @param   offset - offset of the first octet to be written
 *
 * @return  Success or Failure*/
static bStatus_t simpleProfile_WriteAttrCB( uint16 connHandle, gattAttribute_t *pAttr,
                                            uint8 *pValue, uint8 len, uint16 offset )
{
  ...
    switch ( uuid )
    {
      case SIMPLEPROFILE_CHAR1_UUID:
  ...
        //Write the value
        if ( status == SUCCESS )
        {
          uint8 *pCurValue = (uint8 *)pAttr->pValue;
          *pCurValue = pValue[0];
          notifyApp = SIMPLEPROFILE_CHAR1;
         break;
  ...
 // If a charactersitic value changed then callback function to notify application of change
  if ( (notifyApp != 0xFF ) && simpleProfile_AppCBs && simpleProfile_AppCBs->pfnSimpleProfileChange )
  {
    simpleProfile_AppCBs->pfnSimpleProfileChange( notifyApp );
  }
  return ( status );
}
```

A2: Función callback escritura de atributo.

En éste caso, si se ha producido algún cambio en algún atributo se llamará a la función de callback de aplicación para notificarle éste cambio (SimpleProfileChange), definida en **SimpleBLEPeripheral.c** indicando en qué característica se ha producido.

Función de callback de aplicación que es llamada cuando se produce algún cambio en algún atributo:

```
/*********************************************************************
 * @fn      simpleProfileChangeCB
 *
 * @brief   Callback from SimpleBLEProfile indicating a value change
 *
 * @param   paramID - parameter ID of the value that was changed.
 *
 * @return  none
 */
static void simpleProfileChangeCB( uint8 paramID )
{
  uint8 newValue;

  switch( paramID )
  {
    case SIMPLEPROFILE_CHAR1:
      SimpleProfile_GetParameter( SIMPLEPROFILE_CHAR1, &newValue );
      #if (defined HAL_LCD) && (HAL_LCD == TRUE)
        HalLcdWriteStringValue( "Char 1:", (uint16)(newValue), 10,  HAL_LCD_LINE_3 );
      #endif // (defined HAL_LCD) && (HAL_LCD == TRUE)
      break;
    case SIMPLEPROFILE_CHAR3:
      SimpleProfile_GetParameter( SIMPLEPROFILE_CHAR3, &newValue );

      #if (defined HAL_LCD) && (HAL_LCD == TRUE)
        HalLcdWriteStringValue( "Char 3:", (uint16)(newValue), 10,  HAL_LCD_LINE_3 );
      #endif // (defined HAL_LCD) && (HAL_LCD == TRUE)

      break;

    default:
      // should not reach here!
      break;
  }
}
```

A3: Función callback de cambio en atributo.

4.6 Funciones Get y Set

El perfil que contiene las características facilita dos funciones, Set y Get para que la aplicación pueda leer y escribir valores directamente.

Función para escribir atributos desde aplicación, definida en **SimpleGATTProfile.c**:

```
/*********************************************************************
 * @fn      SimpleProfile_SetParameter
 *
 * @brief   Set a Simple Profile parameter.
 *
 * @param   param - Profile parameter ID
 * @param   len - length of data to right
 * @param   value - pointer to data to write.  This is dependent on
 *          the parameter ID and WILL be cast to the appropriate
 *          data type (example: data type of uint16 will be cast to
 *          uint16 pointer).
 *
 * @return  bStatus_t
 */
bStatus_t SimpleProfile_SetParameter( uint8 param, uint8 len, void *value )
{
  bStatus_t ret = SUCCESS;
  switch ( param )
  {
    case SIMPLEPROFILE_CHAR1:
      if ( len == sizeof ( uint8 ) )
      {
        simpleProfileChar1 = *((uint8*)value);
      }
      else
      {
        ret = bleInvalidRange;
      }
      break;

   ...
  }
}
```

A4: Función SET.

Ejemplo de uso en aplicación (**SimpleBLEPeripheral.c**):

```
// Setup the SimpleProfile Characteristic Values
{
  uint8 charValue1 = 1;
  uint8 charValue2 = 2;
  uint8 charValue3 = 3;
  uint8 charValue4 = 4;
  uint8 charValue5[SIMPLEPROFILE_CHAR5_LEN] = { 1, 2, 3, 4, 5 };
  SimpleProfile_SetParameter( SIMPLEPROFILE_CHAR1, sizeof ( uint8 ), &charValue1 );
  SimpleProfile_SetParameter( SIMPLEPROFILE_CHAR2, sizeof ( uint8 ), &charValue2 );
  SimpleProfile_SetParameter( SIMPLEPROFILE_CHAR3, sizeof ( uint8 ), &charValue3 );
  SimpleProfile_SetParameter( SIMPLEPROFILE_CHAR4, sizeof ( uint8 ), &charValue4 );
  SimpleProfile_SetParameter( SIMPLEPROFILE_CHAR5, SIMPLEPROFILE_CHAR5_LEN, charValue5 );
}
```

Ejemplo de uso de función SET.

Función para leer atributos desde aplicación, definida en **SimpleGATTProfile.c**:

```
/*********************************************************************
 * @fn      SimpleProfile_GetParameter
 *
 * @brief   Get a Simple Profile parameter.
 *
 * @param   param - Profile parameter ID
 * @param   value - pointer to data to put.  This is dependent on
 *          the parameter ID and WILL be cast to the appropriate
 *          data type (example: data type of uint16 will be cast to
 *          uint16 pointer).
 *
 * @return  bStatus_t
 */
bStatus_t SimpleProfile_GetParameter( uint8 param, void *value )
{
  bStatus_t ret = SUCCESS;
 ...
  switch ( param )
  {
    case SIMPLEPROFILE_CHAR1:
      *((uint8*)value) = simpleProfileChar1;
      break;

...
  }

  return ( ret );
}
```

A5: Función GET.

Ejemplo de uso en aplicación (**SimpleBLEPeripheral.c**):

```
// Call to retrieve the value of the third characteristic in the profile
stat = SimpleProfile_GetParameter( SIMPLEPROFILE_CHAR3, &valueToCopy);
```

Ejemplo uso función GET.

4.7 Ejemplo del funcionamiento del servidor GATT

En este ejemplo se van a probar las opciones de lectura, escritura y notificación, para ello disponemos de un dispositivo *SensorTag* que será configurado en modo Peripheral y contará con las características definidas en el ejemplo de configuración del servidor GATT del apartado anterior, a ellas se accederá desde un dispositivo móvil actuando en modo Central.

Utilizando el software Packet Sniffer [4] de Texas Instruments se visualizará el contenido de los paquetes enviados entre el Peripheral y el Central.
Hay 4 características definidas:

Característica 1: Permisos de lectura y escritura.
Característica 2: Permiso de lectura.
Característica 3: Permiso de escritura.
Característica 4: Notificación.

Las características se inician con los siguientes valores:

```
// Setup the SimpleProfile Characteristic Values
{
  uint8 charValue1 = 1;
  uint8 charValue2 = 2;
  uint8 charValue3 = 3;
  uint8 charValue4 = 4;
  uint8 charValue5[SIMPLEPROFILE_CHAR5_LEN] = { 1, 2, 3, 4, 5 };
  SimpleProfile_SetParameter( SIMPLEPROFILE_CHAR1, sizeof ( uint8 ), &charValue1 );
  SimpleProfile_SetParameter( SIMPLEPROFILE_CHAR2, sizeof ( uint8 ), &charValue2 );
  SimpleProfile_SetParameter( SIMPLEPROFILE_CHAR3, sizeof ( uint8 ), &charValue3 );
  SimpleProfile_SetParameter( SIMPLEPROFILE_CHAR4, sizeof ( uint8 ), &charValue4 );
  SimpleProfile_SetParameter( SIMPLEPROFILE_CHAR5, SIMPLEPROFILE_CHAR5_LEN, charValue5 );
}
```

Inicialización de valor de características.

Característica 1: Con UUID 0xFFF1, permiso de lectura y escritura, Inicialmente tiene valor 1.

Vemos que efectivamente, tras realizar su lectura su valor es hexadecimal es 0x01.

P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		ATT_Read_Req		CRC	RSSI (dBm)	FCS
	+997273						LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode	AttHandle			
399	=68821005	0x06	0x25DA61D4	M->S	OK	L2CAP-S	2	0	0	0	7	0x0003	0x0004	0x0A	0x0025	0xB2C9FE	-34	OK

Paquete Read request.

P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		ATT_Read_Rsp		CRC	RSSI (dBm)	FCS
	+230						LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode	AttValue			
402	=69818739	0x16	0x25DA61D4	S->M	OK	L2CAP-S	2	0	1	0	6	0x0002	0x0004	0x0B	01	0x98A7E0	-54	OK

Paquete Read Response con valor 01.

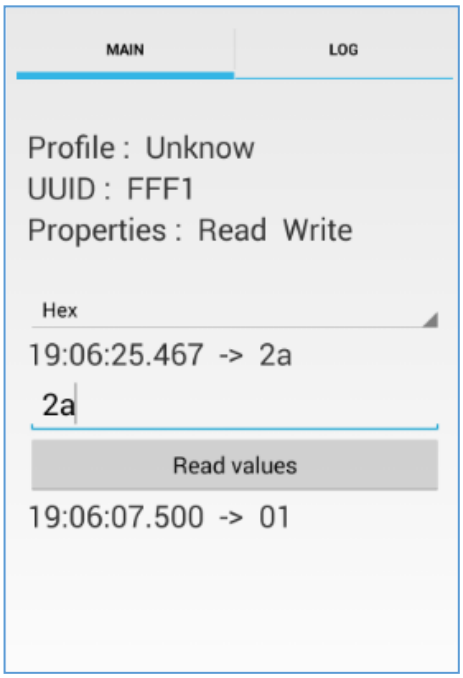
A continuación se va a escribir un nuevo valor, en este caso 0x2A:

P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		ATT_Write_Req			CRC	RSSI (dBm)	FCS
	+997273						LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode	AttHandle	AttValue			
494	=116701167	0x22	0x25DA61D4	M->S	OK	L2CAP-S	2	1	1	0	8	0x0004	0x0004	0x12	0x0025	2A	0x131A8F	-38	OK

Paquete Write Request con valor 2ª .

P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		ATT_Write_Rsp	CRC	RSSI (dBm)	FCS
	+231						LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode			
499	=117699363	0x0D	0x25DA61D4	S->M	OK	L2CAP-S	2	1	0	0	5	0x0001	0x0004	0x13	0x8DEF2C	-53	OK

Paquete Write Response confirmación de escritura.



Una vez escrito se procede a leer de nuevo el valor para comprobar que la acción ha sido realizada con éxito:

Paquetes de la operación de lectura:

P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		ATT_Read_Req		CRC	RSSI (dBm)	FCS
	+997273						LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode	AttHandle			
565	=153608793	0x22	0x25DA61D4	M->S	OK	L2CAP-S	2	0	0	0	7	0x0003	0x0004	0x0A	0x0025	0xB2C9FE	-37	OK

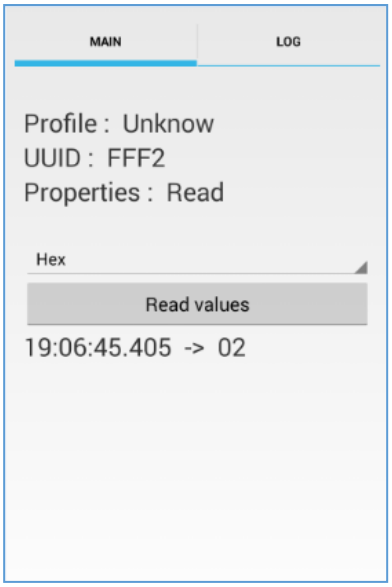
Paquete Read Request.

P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		ATT_Read_Rsp		CRC	RSSI (dBm)	FCS
	+231						LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode	AttValue			
570	=155604031	0x1D	0x25DA61D4	S->M	OK	L2CAP-S	2	0	1	0	6	0x0002	0x0004	0x0B	2A	0xA144A0	-58	OK

Paquete Read Response con valor 2A.

Vemos que efectivamente, en el campo AttValue el valor actual del atributo es 0x2A

Característica 2: Con UUID 0xFFF2, permiso de sólo lectura, tras realizar la operación vemos que efectivamente su valor es 0x02.



Paquetes de la operación de lectura:

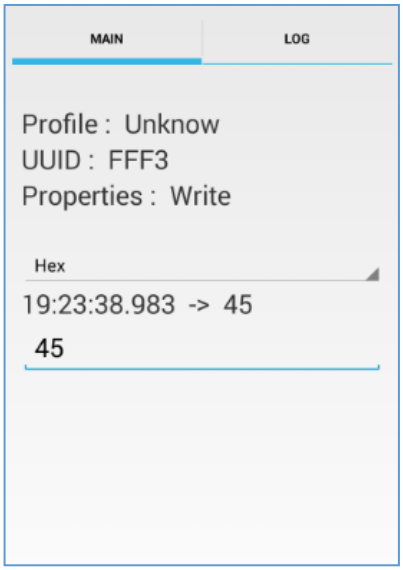
P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		ATT_Read_Req		CRC	RSSI (dBm)	FCS
	+997273						LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode	AttHandle			
606	=175553870	0x10	0x25DA61D4	M->S	OK	L2CAP-S	2	1	1	0	7	0x0003	0x0004	0x0A	0x0028	0xD7231F	-32	OK

Paquete Read Request.

P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		ATT_Read_Rsp		CRC	RSSI (dBm)	FCS
	+230						LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode	AttValue			
609	=176551604	0x20	0x25DA61D4	S->M	OK	L2CAP-S	2	1	0	0	6	0x0002	0x0004	0x0B	02	0x56DA2E	-66	OK

Paquete Read Response con valor 02.

Característica 3: Con UUID 0xFFF3, propiedad de solo escritura, en este caso se ha escrito el número 45:



Paquetes de la operación de escritura:

P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		ATT_Write_Req			CRC	RSSI (dBm)	FCS
	+997273						LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode	AttHandle	AttValue			
820	=285279266	0x00	0x25DA61D4	M->S	OK	L2CAP-S	2	1	1	0	8	0x0004	0x0004	0x12	0x002B	45	0x51F9A7	-34	OK

Paquete Write Request.

P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		ATT_Write_Rsp	CRC	RSSI (dBm)	FCS
	+231						LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode			
823	=286277001	0x10	0x25DA61D4	S->M	OK	L2CAP-S	2	1	0	0	5	0x0001	0x0004	0x13	0x8DEF2C	-73	OK

Paquete Write Response.

Característica 4: En esta característica se va a comprobar el funcionamiento del modo Notificación, en el cual el servidor GATT del Peripheral informará al cliente en el teléfono (Central) cada vez que se realice una acción de escritura en la característica.
Para realizar la comprobación se ha creado una función periódica en la aplicación que cada 5 segundos copiará el valor de la característica 3 en la característica 4:

Definición del periodo en ms:

```
// How often to perform periodic event
#define SBP_PERIODIC_EVT_PERIOD                   5000
```

Periodo función periódica.

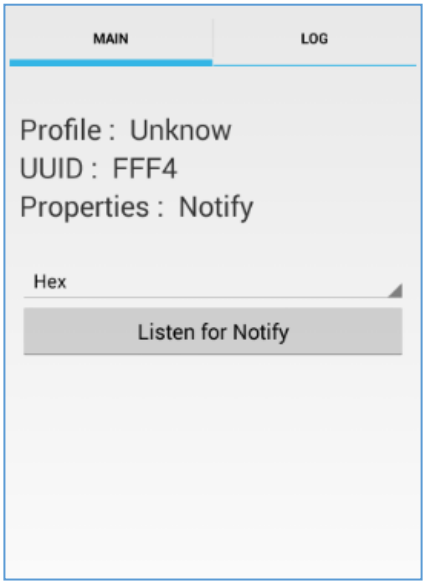
Función periódica:

```
static void performPeriodicTask( void )
{
  uint8 valueToCopy;
  uint8 stat;


  stat = SimpleProfile_GetParameter( SIMPLEPROFILE_CHAR3, &valueToCopy);

  if( stat == SUCCESS )
  {
    SimpleProfile_SetParameter( SIMPLEPROFILE_CHAR4, sizeof(uint8), &valueToCopy);
  }
}
```

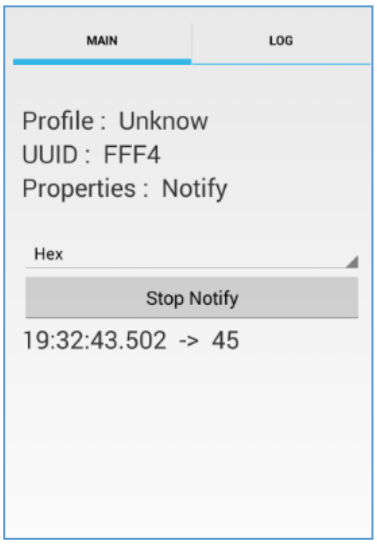
A6: función periódica. fuente: aplicación

Una vez copiado el valor, si la opción de notificación está habilitada el dispositivo Peripheral enviará un mensaje al Central con el nuevo valor.

Característica 4 con UUID 0xFFF4 y propiedad de notificación:



Activación de la notificación:



Proceso de activación de la opción de notificación: Como ya se vió en el apartado de la tabla de atributos (página 32), para activar la opción Notify hay que escribir un 0x0001 en el atributo Client Characteristic Configuration de la característica 4:

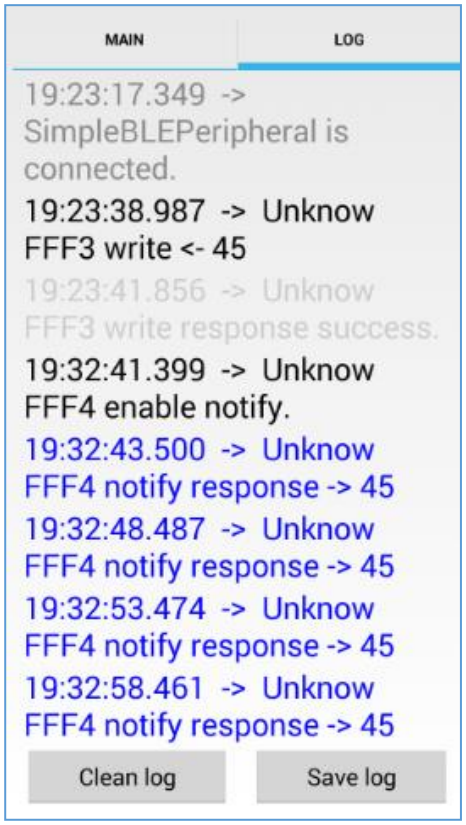
P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		ATT_Write_Req			CRC	RSSI (dBm)	FCS
	+997272						LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode	AttHandle	AttValue			
668	=207473983	0x0A	0x25DA61D4	M->S	OK	L2CAP-S	2	1	1	0	9	0x0005	0x0004	0x12	0x002F	01 00	0x044278	-49	OK

Paquete Write Request para la activación de la opción Notify (Valor 0001).

P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		ATT_Write_Rsp	CRC	RSSI (dBm)	FCS
	+231						LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode			
671	=208471718	0x1A	0x25DA61D4	S->M	OK	L2CAP-S	2	1	0	0	5	0x0001	0x0004	0x13	0x8DEF2C	-55	OK

Paquete Write Response de la activación de la opción Notify.

Observando el registro de logs se puede ver que cada 5 segundos se recibe un mensaje con el valor de la característica:



Vemos que se reciben un mensajes con el valor que se ha escrito anteriormente en la característica 3 (45):

P.nbr.	Time (us) +231	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		ATT_Handle_Value_Notify			CRC	RSSI (dBm)	FCS
							LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode	AttHandle	AttValue			
679	=212461732	0x10	0x25DA61D4	S->M	OK	L2CAP-S	2	1	0	0	8	0x0004	0x0004	0x1B	0x002E	45	0x1E55FD	-52	OK
689	=217449250	0x16	0x25DA61D4	S->M	OK	L2CAP-S	2	0	1	0	8	0x0004	0x0004	0x1B	0x002E	45	0x281F4D	-51	OK
699	=222436768	0x1C	0x25DA61D4	S->M	OK	L2CAP-S	2	1	0	0	8	0x0004	0x0004	0x1B	0x002E	45	0x1E55FD	-48	OK
709	=227424286	0x22	0x25DA61D4	S->M	OK	L2CAP-S	2	0	1	0	8	0x0004	0x0004	0x1B	0x002E	45	0x281F4D	-53	OK

Varios paquetes recibidos por notificación con valor 45.

Capítulo 5

Características Técnicas del CC2540/41

El CC2540 y el CC2541 son dos SoC para comunicación Bluetooth Low Energy diseñados por Texas Instruments. Optimizados para un bajo consumo permiten la comunicación entre dispositivos de una forma fiable y a un bajo precio.

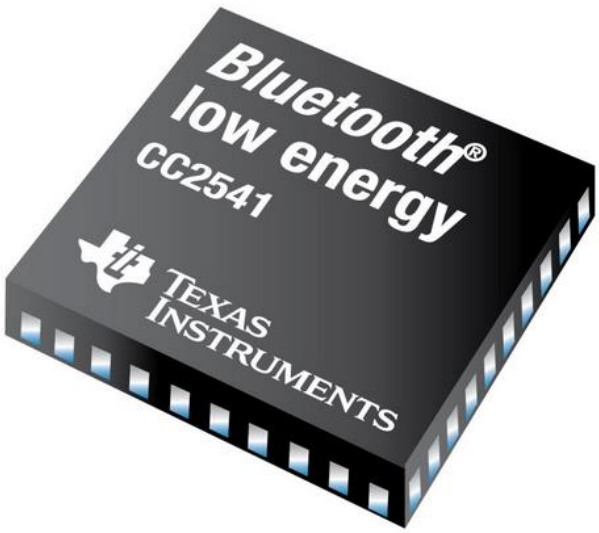
Ambos constan de un microcontrolador MCU 8051 y una RAM de 8 bytes, no obstante existen ciertas diferencias:

CC2541 no tiene soporte para USB
El CC2541 elimina el soporte para USB que tiene el CC2540 y en su lugar utiliza I2C en los mismos pins.

Mayor potencia de transmisión en CC2540
El CC2540 tiene una mayor potencia de transmisión (4dBm) comparados con los 0 dBm del CC2541, por lo tanto, en términos de rango de radio el CC2540 tiene un mejor comportamiento.

Mayor consumo de potencia en el CC2540
La mayor potencia de transmisión conlleva un mayor consumo, exactamente 27mA para una transmisión a 4 dBm frente a los 18.2mA del CC2541 cuando transmite a 0dBm.

Tanto los pines del CC2540 como los del CC2541 son compatibles, a excepción de aquellos dedicados a USB/I2C.



5.1 Pines

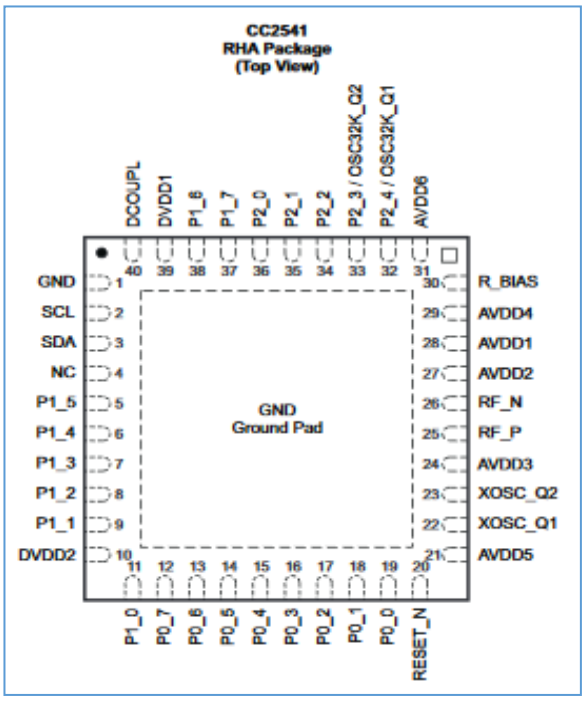


Figura 5.1: Pines CC2541. Texas Instruments

PIN DESCRIPTIONS

PIN NAME	PIN	PIN TYPE	DESCRIPTION
AVDD1	28	Power (analog)	2-V–3.6-V analog power-supply connection
AVDD2	27	Power (analog)	2-V–3.6-V analog power-supply connection
AVDD3	24	Power (analog)	2-V–3.6-V analog power-supply connection
AVDD4	29	Power (analog)	2-V–3.6-V analog power-supply connection
AVDD5	21	Power (analog)	2-V–3.6-V analog power-supply connection
AVDD6	31	Power (analog)	2-V–3.6-V analog power-supply connection
DCOUPL	40	Power (digital)	1.8-V digital power-supply decoupling. Do not use for supplying external circuits.
DVDD1	39	Power (digital)	2-V–3.6-V digital power-supply connection
DVDD2	10	Power (digital)	2-V–3.6-V digital power-supply connection
GND	1	Ground pin	Connect to GND
GND	—	Ground	The ground pad must be connected to a solid ground plane.
NC	4	Unused pins	Not connected
P0_0	19	Digital I/O	Port 0.0
P0_1	18	Digital I/O	Port 0.1
P0_2	17	Digital I/O	Port 0.2
P0_3	16	Digital I/O	Port 0.3
P0_4	15	Digital I/O	Port 0.4
P0_5	14	Digital I/O	Port 0.5
P0_6	13	Digital I/O	Port 0.6
P0_7	12	Digital I/O	Port 0.7
P1_0	11	Digital I/O	Port 1.0 – 20-mA drive capability
P1_1	9	Digital I/O	Port 1.1 – 20-mA drive capability
P1_2	8	Digital I/O	Port 1.2
P1_3	7	Digital I/O	Port 1.3
P1_4	6	Digital I/O	Port 1.4
P1_5	5	Digital I/O	Port 1.5
P1_6	38	Digital I/O	Port 1.6
P1_7	37	Digital I/O	Port 1.7
P2_0	36	Digital I/O	Port 2.0
P2_1/DD	35	Digital I/O	Port 2.1 / debug data
P2_2/DC	34	Digital I/O	Port 2.2 / debug clock
P2_3/ OSC32K_Q2	33	Digital I/O, Analog I/O	Port 2.3/32.768 kHz XOSC
P2_4/ OSC32K_Q1	32	Digital I/O, Analog I/O	Port 2.4/32.768 kHz XOSC
RBIAS	30	Analog I/O	External precision bias resistor for reference current
RESET_N	20	Digital input	Reset, active-low
RF_N	26	RF I/O	Negative RF input signal to LNA during RX Negative RF output signal from PA during TX
RF_P	25	RF I/O	Positive RF input signal to LNA during RX Positive RF output signal from PA during TX
SCL	2	I^2C clock or digital I/O	Can be used as I^2C clock pin or digital I/O. Leave floating if not used. If grounded disable pull up
SDA	3	I^2C clock or digital I/O	Can be used as I^2C data pin or digital I/O. Leave floating if not used. If grounded disable pull up
XOSC_Q1	22	Analog I/O	32-MHz crystal oscillator pin 1 or external clock input
XOSC_Q2	23	Analog I/O	32-MHz crystal oscillator pin 2

Figura 5.2: Tabla de Pines. Texas Instruments

5.2 Diagrama de bloques

Los módulos del diagrama de bloques pueden ser divididos en tres categorías: módulos relacionados con la CPU, módulos relacionados con la potencia, test y distribución de la señal de reloj y módulos relacionados con la radio. En la siguiente sección se explicarán cada uno de ellos.

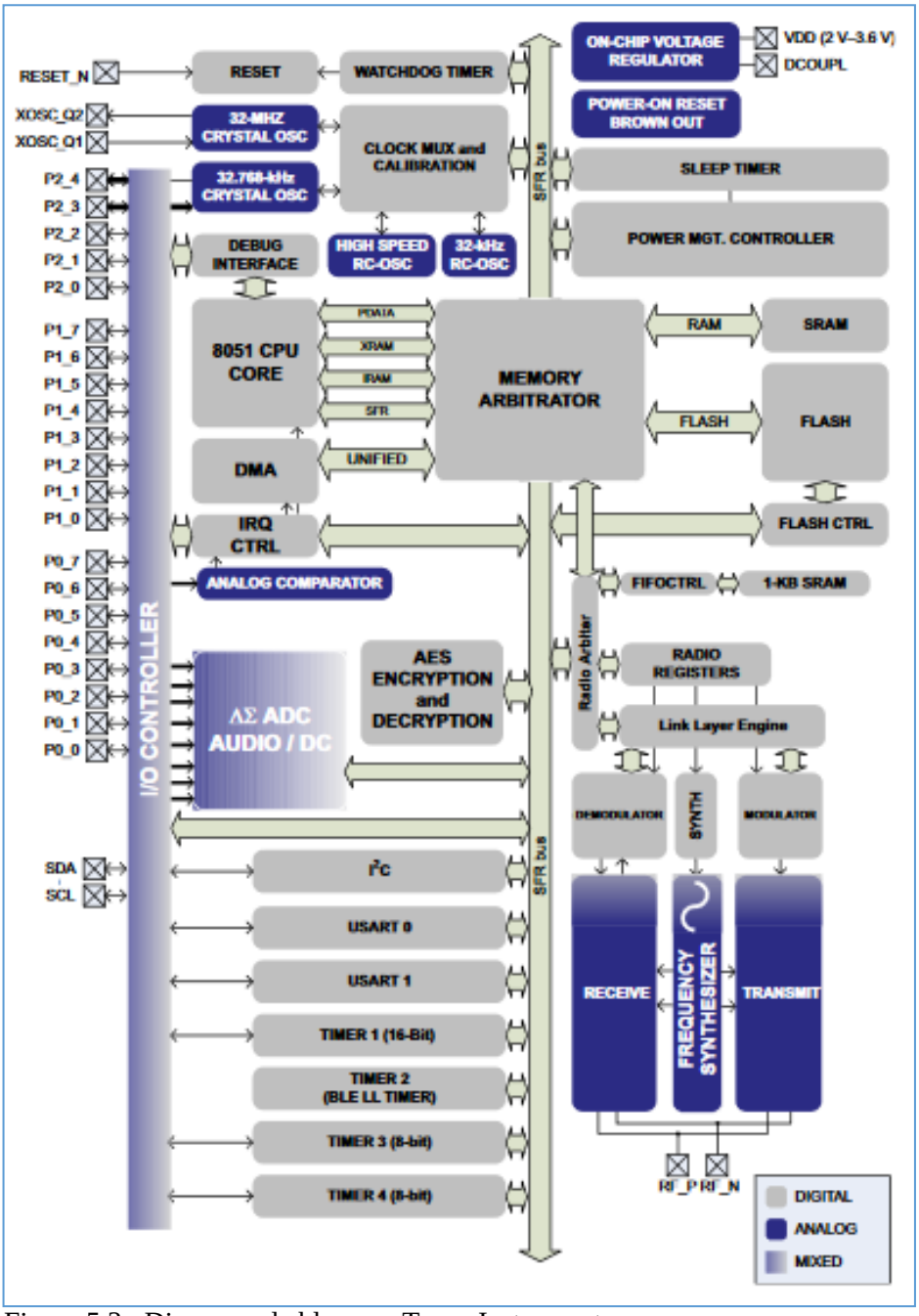


Figura 5.3: Diagrama de bloques. Texas Instruments

5.3 Descripción de los bloques

5.3.1 CPU y Memoria

- **Núcleo de la CPU**: Es un **8051**, consta de 3 buses de acceso de memoria (SFR, DATA y CODE/DATA), una interfaz de depuración y una unidad de interrupción con 18 entradas.
- **Memory Arbiter:** Conecta la CPU y el controlador de DMA con las memorias físicas y todos los periféricos mediante un bus SFR.
- **Bus SFR:** Conecta todos los periféricos hardware con el memory arbiter, además permite el acceso a los registros de radio en el banco de registros de radio pese a que éstos ya están mapeados en el espacio de memoria de XDATA.
- **SRAM de 8KB**: Mantiene su contenido aunque la parte digital se apague.
- **128/265 KB flash block**: Es la memoria programable no volátil del dispositivo.

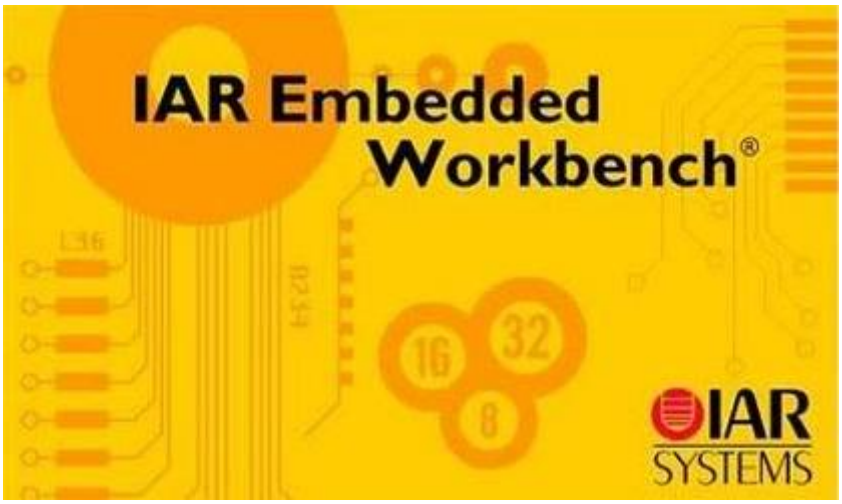
5.3.2 Periféricos

- **Controlador de interrupciones**: Maneja un total de 18 fuentes de interrupción, divididas en 6 grupos, cada uno de los cuales tiene asociado una de las 4 prioridades de interrupción disponibles.
- **Interfaz de depuración**: implementa una interfaz serie para depuración. Permite borrar o programar la memoria flash, parar o comenzar la ejecución del programa de usuario, ejecutar instrucciones en el 8051, habilitar breakpoints y ejecutar el programa paso a paso.
- **Controlador I/O**: Es el responsable de gestionar los pines de entrada y salida de propósito general.
- **Sleep timer**: Se ejecuta continuamente en todos los modos de operación, su aplicación típica es como contador o bien como timer de wake-up para salir del modo de ahorro de energía.
- **Watchdog timer**: Permite al CC2540/CC2541 resetearse si el firmware se cuelga.
- **Timer 1:** timer de 16 bits con funcionalidad de timer/contador/PWM. Consta de un prescaler programable y 5 canales de captura, cada uno con un comparador de 16 bits, además los canales pueden ser usados como generadores de PWM.
- **Timer 2:** timer de 40 bits, consta de un contador de 16 bits y contador de overflow de 24 bits.
- **Timer 3 y 4:** timers de 8 bits con funcionalidad de timer/contador/PWM, consta de un prescaler programable y un canal de contador con un valor de comparación de 8 bits que puede ser usado para la generación de PWM.
- **USART 0 y USART 1**: Pueden configurarse como SPI o UART
- **AES encryption/decryption core**: Permite al usuario encriptar y desencriptar datos usando el algoritmo AES con llaves de 128 bits.
- **ADC:** Con resolución configurable de 7 o 12 bits, permite trabajar en un rango de frecuencias entre 4 y 30KHz, puede automatizar el proceso de toma de muestras periódico.
- **I2C:** Para la conexión digital de periféricos mediante 2 pines, permite operar como master o slave.

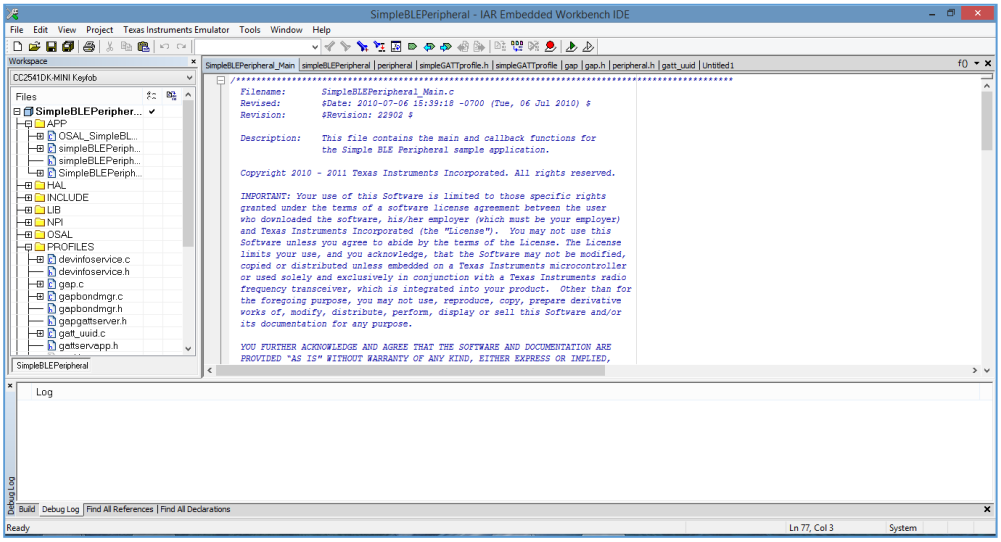
Capítulo 6

IAR Embedded Workbench

El software diseñado para el CC2540/41 es desarrollado utilizando la herramienta Embedded Workbench para 8051 de la compañía IAR [9].



A continuación se muestra una captura de pantalla del entorno de trabajo:



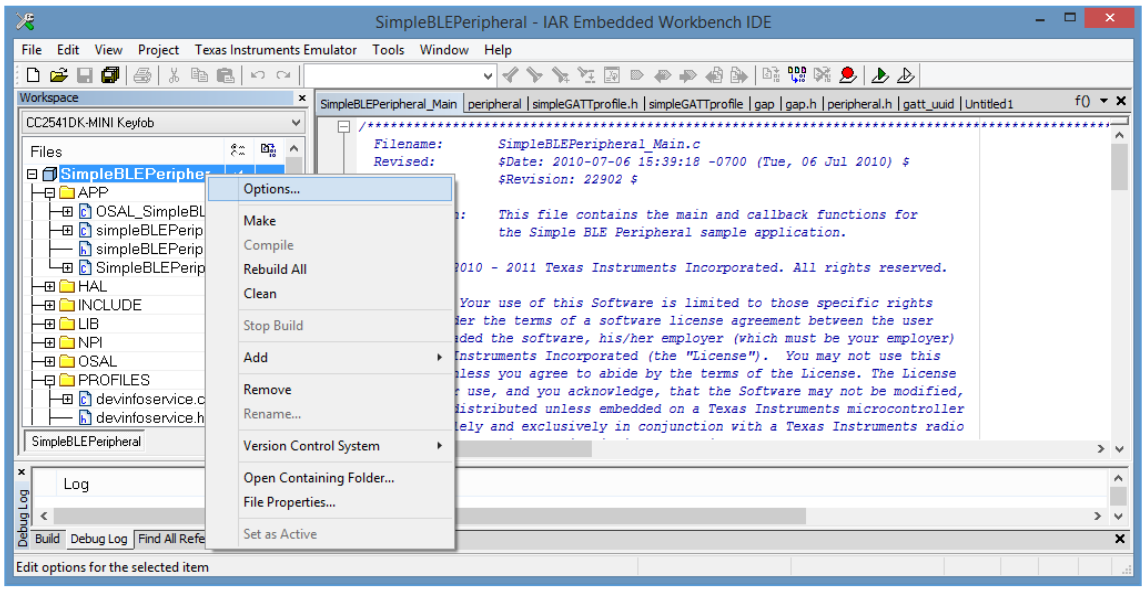
Ventana de trabajo IAR Embedded Workbench.

A la izquierda podemos observar los archivos de nuestro proyecto distribuidos en carpetas (Aplicación, perfiles..). En la ventana principal está el código a ejecutar y abajo la ventana de log donde se mostrarán los errores o warnings que se produzcan durante la compilación.

6.1 Opciones

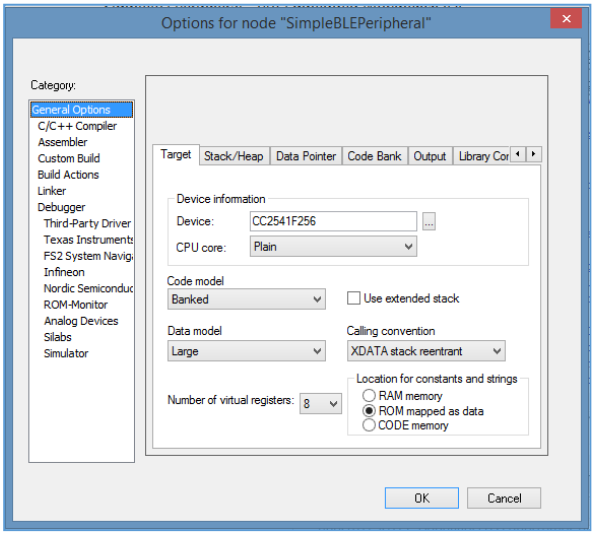
Para ver las opciones de un proyecto:

Primero Click derecho en el nombre del proyecto del que cuelgan todas las carpetas:



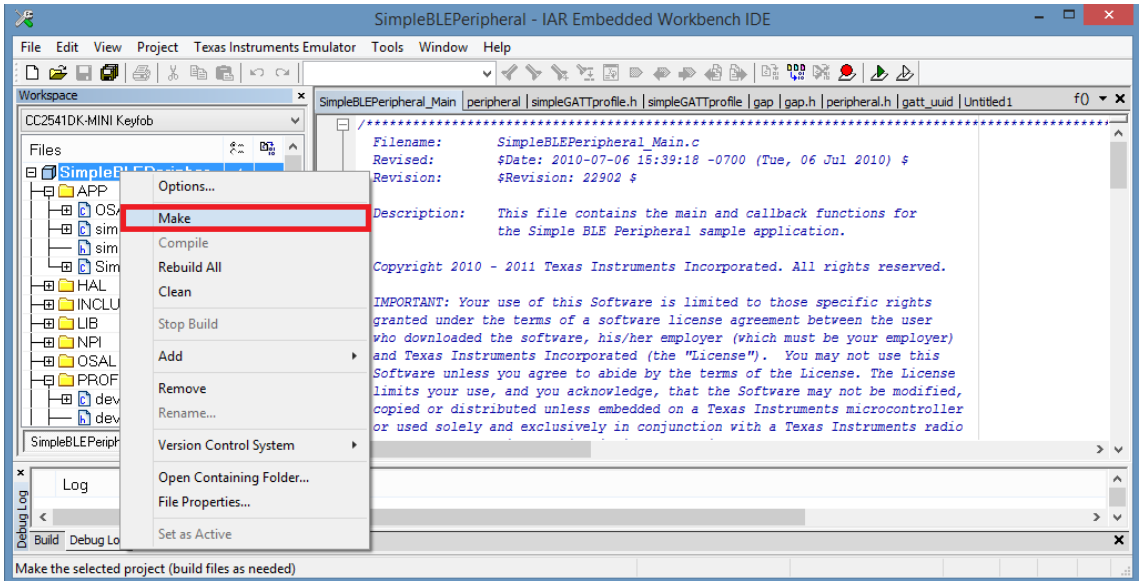
Menú opciones.

Una vez clickemos en opciones saldrá una ventana mostrando las opciones del proyecto donde podremos configurar con qué dispositivo vamos a trabajar, cuál será el debugger a utilizar (en nuestro caso CC Debugger) o si querremos ejecutar el código en modo simulación entre otras opciones.



6.2 Construir un proyecto

Primero debemos hacer click derecho en el nombre del proyecto:



Opción Make.

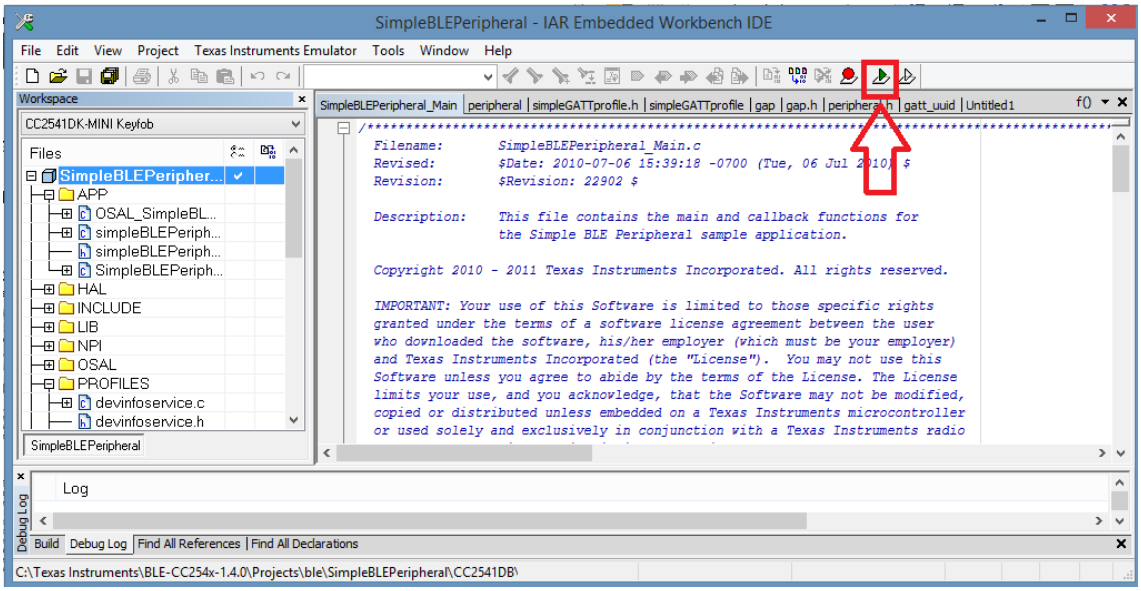
A continuación seleccionamos la opción Make o bien presionamos F7, esta acción compilará el código fuente, linkeará todos los archivos y construirá el proyecto, cualquier error o warning que se produzca aparecerá en la ventana de abajo.

6.3 Depuración de un proyecto

Para depurar un proyecto primero conectaremos el dispositivo con el CC2540/41 mediante el CC Debugger [7] a un puerto USB del PC.

CC Debugger.

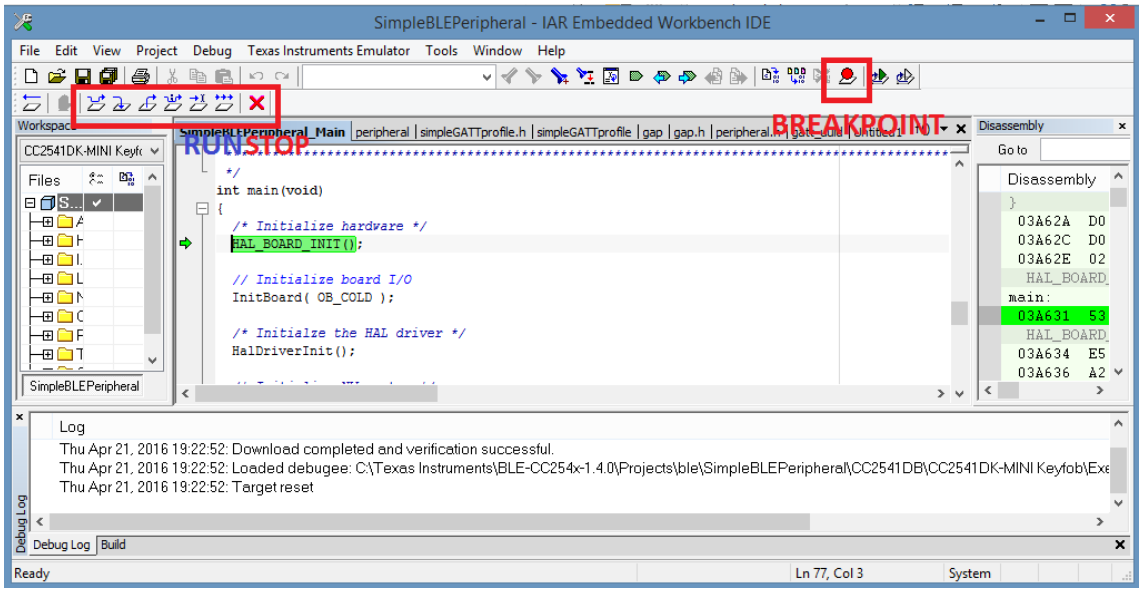
Acto seguido pulsaremos el botón de depuración en la esquina superior derecha de la ventana de trabajo:



Opción depuración.

Esto descargará el programa en la placa.

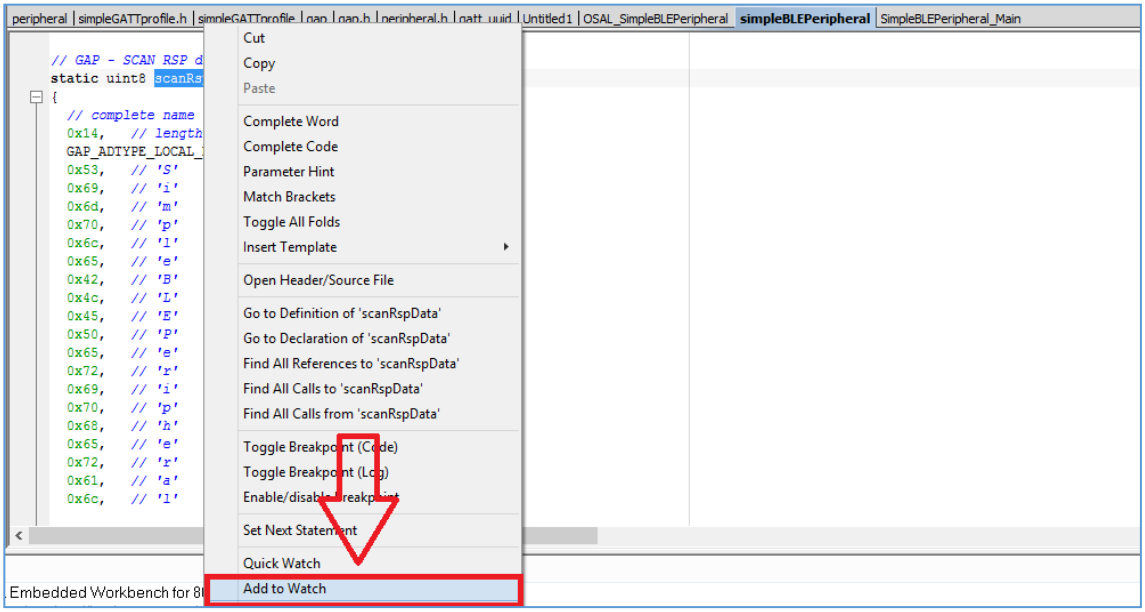
A continuación podremos ejecutar el programa presionando el botón run, ejecutarlo paso a paso o hasta el siguiente breakpoint:



Opciones de ejecución y creación de breakpoints.

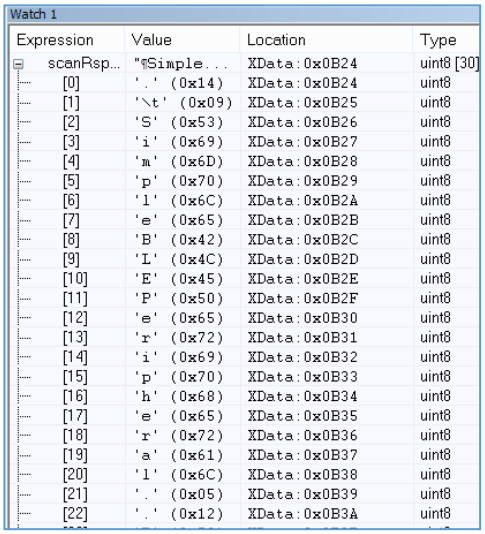
6.4 Ventanas de Watch

Podemos ver el valor de cualquier variable en tiempo real haciendo click sobre su nombre y seleccionando la opción Add to Watch:



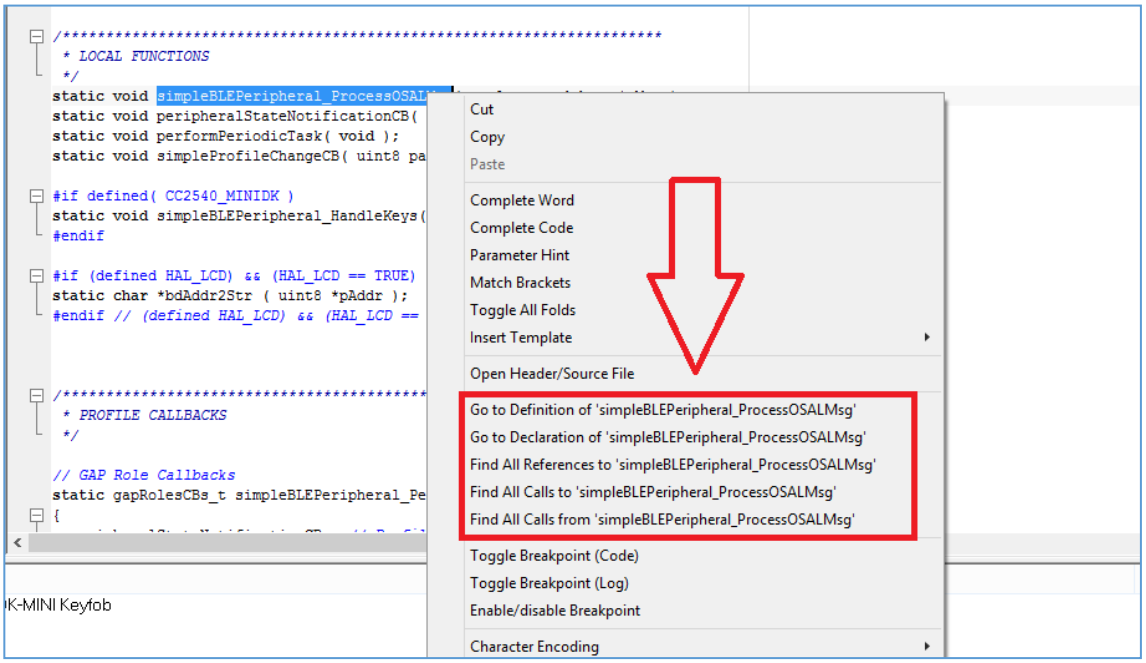
Opción Add to Watch.

Esto nos abrirá una ventana Watch donde podremos añadir todas las variables de interés que se quieran observar, en este caso se muestra el contenido de la variable scanRsponse, la cual contiene el valor del paquete de Scan Response a enviar por el dispositivo:



Ventana Watch.

Otra opción muy útil nos permite, seleccionando una variable o función, ir directamente al lugar en el que se declaró o buscar desde que archivos es llamada o referenciada:



Opciones de búsqueda.

Capítulo 7

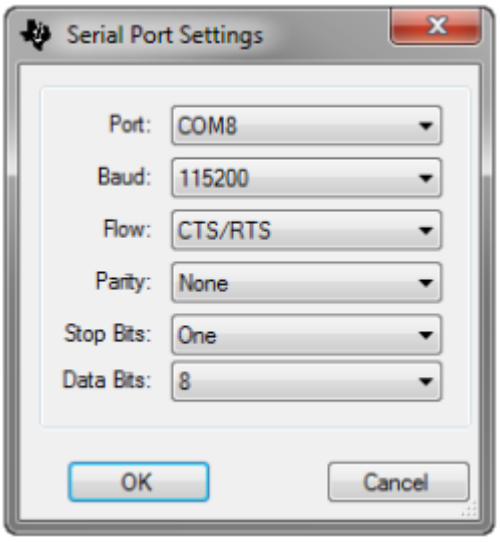
BTOOL

BTOOL [3] es una herramienta desarrollada por Texas Instruments que permite establecer conexiones con dispositivos BLE y, de ese modo, comprobar su correcto funcionamiento.

Consta de un dispositivo hardware llamado USB Dongle con el que se comunica a través del puerto serie del PC.

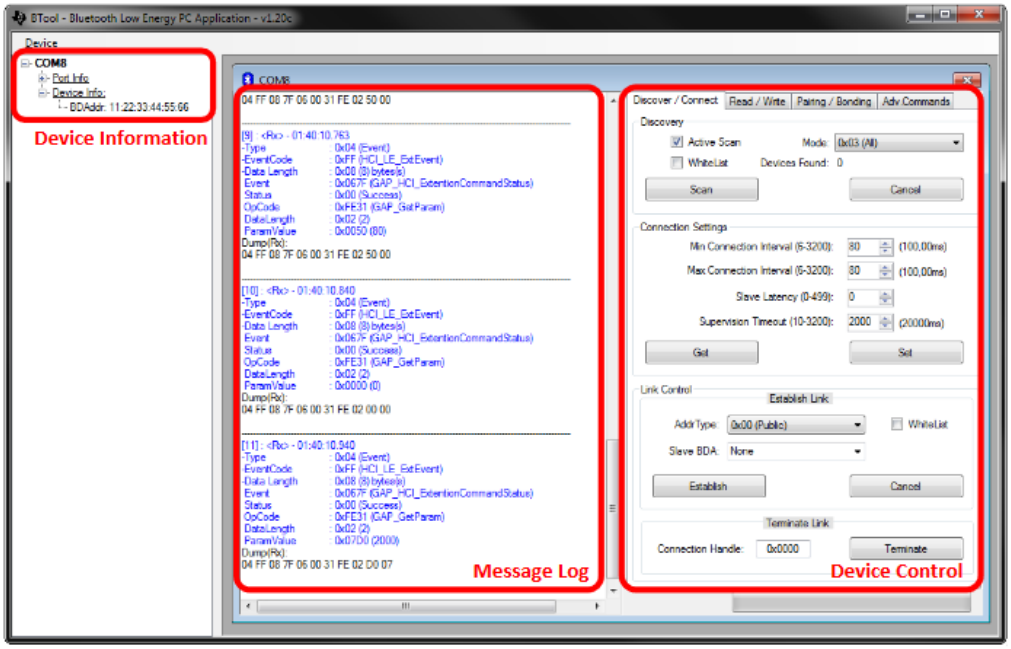
7.1 Inicio de la aplicación

Lo primero que se pide al abrir el programa es configurar el puerto serie a través del que se realizará la comunicación con el dispositivo USB Dongle:



Configuración Puerto Serie

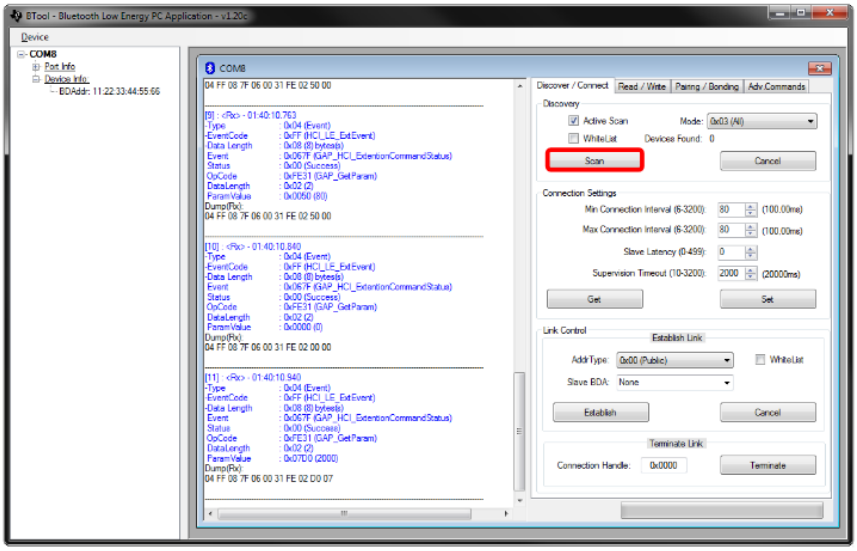
Una vez realizada la conexión se mostrará la siguiente pantalla:



La pantalla está dividida en 3 secciones, la parte izquierda muestra información sobre el estado de la conexión con el USB Dongle, la parte central es el registro del Log de las comunicaciones entre el PC y el USB Dongle y la parte izquierda contiene las opciones de control del dispositivo.

7.2 Creación de una conexión entre el USB Dongle y un dispositivo BLE

Una vez realizada la configuración del puerto serie el USB Dongle está listo para descubrir dispositivos BLE que estén en modo Advertising, en la parte de control de la pantalla se presionará el botón “Scan”:

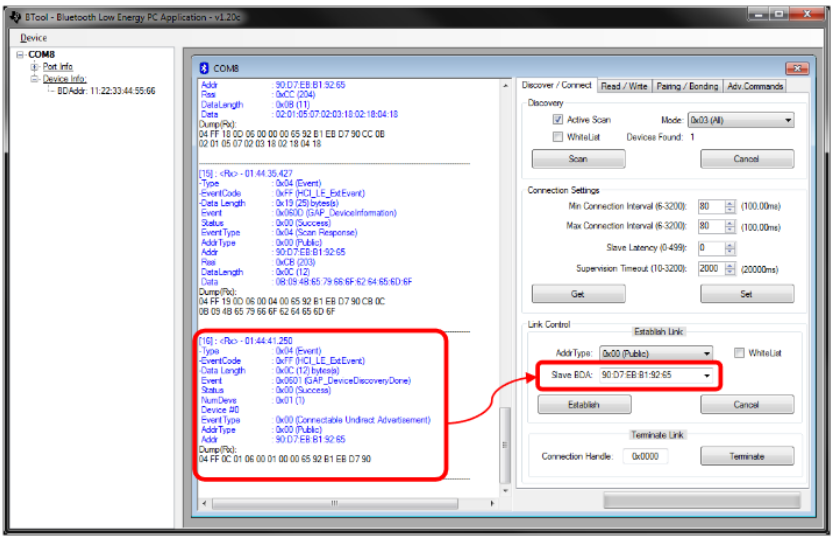


El USB Dongle comenzará a escanear en busca de otros dispositivos BLE, conforme vayan siendo descubiertos la pantalla de log irá mostrándolos, el proceso de escaneo terminará tras 10 segundos o al apretar el botón de “Cancel”.

```
------------------------------------------------------------------------
[13] : <Tx> - 07:57:29.572
-Type              : 0x01 (Command)
-OpCode            : 0xFE04 (GAP_DeviceDiscoveryRequest)
-Data Length       : 0x03 (3) byte(s)
 Mode              : 0x03 (3) (All)
 ActiveScan        : 0x01 (1) (Enable)
 WhiteList         : 0x00 (0) (Disable)
Dump(Tx):
0000:01 04 FE 03 03 01 00                                .......
------------------------------------------------------------------------
```

```
------------------------------------------------------------------------
[15] : <Rx> - 07:57:32.557
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x14 (20) bytes(s)
 Event           : 0x060D (1549) (GAP_DeviceInformation)
 Status          : 0x00 (0) (Success)
 EventType       : 0x00 (0) (Connectable Undirect Advertisement)
 AddrType        : 0x00 (0) (Public)
 Addr            : B4:99:4C:64:30:D5
 Rssi            : 0xD4 (212)
 DataLength      : 0x07 (7)
 Data            : 02:01:05:03:02:F0:FF
Dump(Rx):
0000:04 FF 14 0D 06 00 00 00 D5 30 64 4C 99 B4 D4 07 .........0dL....
0010:02 01 05 03 02 F0 FF                            .......
------------------------------------------------------------------------


------------------------------------------------------------------------
[16] : <Rx> - 07:57:32.573
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x2B (43) bytes(s)
 Event           : 0x060D (1549) (GAP_DeviceInformation)
 Status          : 0x00 (0) (Success)
 EventType       : 0x04 (4) (Scan Response)
 AddrType        : 0x00 (0) (Public)
 Addr            : B4:99:4C:64:30:D5
 Rssi            : 0xD3 (211)
 DataLength      : 0x1E (30)
 Data            : 14:09:53:69:6D:70:6C:65:42:4C:45:50:65:72:69:70:
                   68:65:72:61:6C:05:12:50:00:20:03:02:0A:00
Dump(Rx):
0000:04 FF 2B 0D 06 00 04 00 D5 30 64 4C 99 B4 D3 1E ..+......0dL....
0010:14 09 53 69 6D 70 6C 65 42 4C 45 50 65 72 69 70 ..SimpleBLEPerip
0020:68 65 72 61 6C 05 12 50 00 20 03 02 0A 00       heral..P. ....
------------------------------------------------------------------------
```

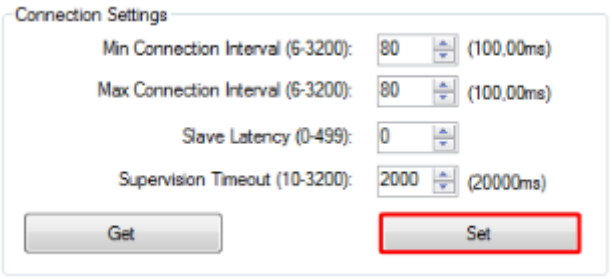
La dirección de todo dispositivo escaneado aparecerá en la sección Slave BDA de la ventana de control:



7.3 Selección de parámetros de conexión

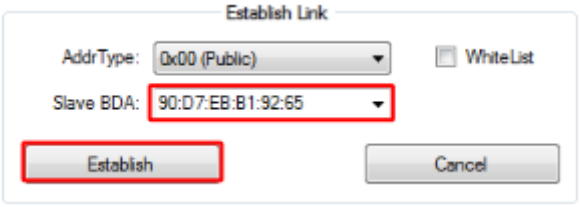
Antes de establecer una conexión se pueden configurar los parámetros deseados para la misma, los valores por defecto son 100 ms de intervalo de conexión, 0 de slave latency y 20s de supervisión timeout. Una vez establecidos hay que clickar en el botón “Set” para hacerlos efectivos.

Los parámetros han de definirse antes de que se establezca la conexión, cambiarlos durante una conexión activa no tendrá ningún efecto.



7.4 Establecimiento de conexión

Para establecer una conexión hay que seleccionar la dirección del dispositivo a conectar y hacer click en el botón “Establish”.




```
------------------------------------------------------------------------
[20] : <Tx> - 07:57:49.186
-Type            : 0x01 (Command)
-OpCode          : 0xFE09 (GAP_EstablishLinkRequest)
-Data Length     : 0x09 (9) byte(s)
 HighDutyCycle   : 0x00 (0) (Disable)
 WhiteList       : 0x00 (0) (Disable)
 AddrTypePeer    : 0x00 (0) (Public)
 PeerAddr        : B4:99:4C:64:30:D5
Dump(Tx):
0000:01 09 FE 09 00 00 00 D5 30 64 4C 99 B4          ........0dL..
------------------------------------------------------------------------
```

Una vez establecida la conexión en la ventana de log aparecerá un mensaje de evento de tipo “GAP_EstablishLink” con un valor de 0x00 (Success) en su campo de “Status”

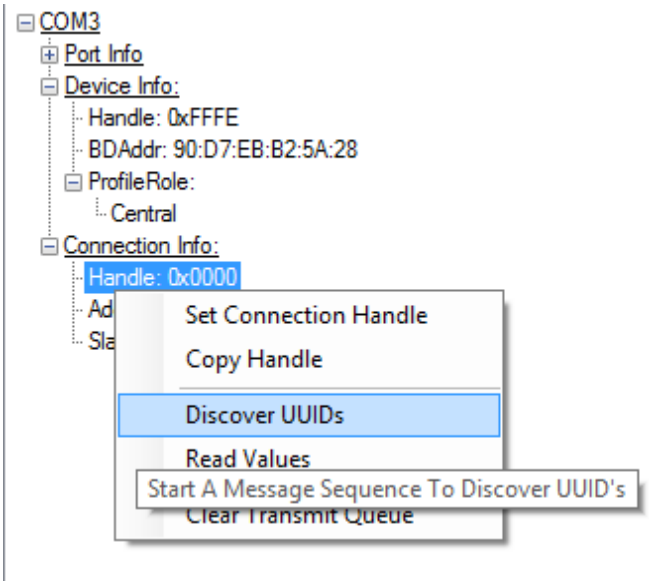
```
------------------------------------------------------------------------
[24] : <Rx> - 07:57:55.081
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x13 (19) bytes(s)
 Event           : 0x0605 (1541) (GAP_EstablishLink)
 Status          : 0x00 (0) (Success)
 DevAddrType     : 0x00 (0) (Public)
 DevAddr         : B4:99:4C:64:30:D5
 ConnHandle      : 0x0000 (0)
 ConnRole        : 0x50 (80) ()
 ConnInterval    : 0x0000 (0)
 ConnLatency     : 0xD000 (53248)
 ConnTimeout     : 0x0007 (7)
Dump(Rx):
0000:04 FF 13 05 06 00 00 D5 30 64 4C 99 B4 00 00 50 ........0dL....P
0010:00 00 00 D0 07 00                               ......
------------------------------------------------------------------------
```

En la ventana de información de dispositivo se podrán ver los datos referentes al dispositivo conectado:

Descubrimiento de todos los servicios de un dispositivo.

```
------------------------------------------------------------------------
[28] : <Tx> - 07:58:06.691
-Type            : 0x01 (Command)
-OpCode          : 0xFD90 (GATT_DiscAllPrimaryServices)
-Data Length     : 0x02 (2) byte(s)
 ConnHandle      : 0xFFFE (65534)
Dump(Tx):
0000:01 90 FD 02 FE FF                               ......
------------------------------------------------------------------------
```

Haciendo click derecho en el apartado Handle de la ventana de información del dispositivo podemos solicitarle el envío de todos los UUIDs de los servicios disponibles:



El resultado sería el siguiente

ConHnd	Handle	Uuid	Uuid Description	Value	Value Description	Properties
0x0000	0x0023	0x2800	GATT Primary Service Declaration	F0:FF		
0x0000	0x0024	0x2803	GATT Characteristic Declaration			
0x0000	0x0025	0xFFF1	Simple Profile Char 1			
0x0000	0x0026	0x2901	Characteristic User Description			
0x0000	0x0027	0x2803	GATT Characteristic Declaration			
0x0000	0x0028	0xFFF2	Simple Profile Char 2			
0x0000	0x0029	0x2901	Characteristic User Description			
0x0000	0x002A	0x2803	GATT Characteristic Declaration			
0x0000	0x002B	0xFFF3	Simple Profile Char 3			
0x0000	0x002C	0x2901	Characteristic User Description			
0x0000	0x002D	0x2803	GATT Characteristic Declaration			
0x0000	0x002E	0xFFF4	Simple Profile Char 4			
0x0000	0x002F	0x2902	Client Characteristic Configuration			
0x0000	0x0030	0x2901	Characteristic User Description			

En la tabla, como ejemplo, se observan los servicios disponibles en perfil Simple BLE Pheripheral, el cual está ejecutándose en el dispositivo conectado.

También se pueden leer todos los valores de las características, así como sus permisos, de una sola vez, seleccionando la opción “Read Values” en la ventana del dispositivo:

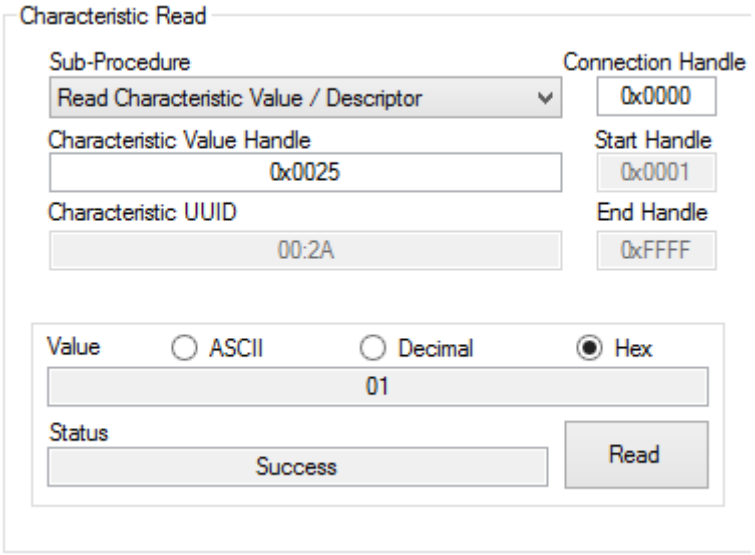
Tras ésta operación el resultado sería el siguiente:

ConHnd	Handle	Uuid	Uuid Description	Value	Value Description	Properties
0x0000	0x0023	0x2800	GATT Primary Service Declaration	F0:FF		
0x0000	0x0024	0x2803	GATT Characteristic Declaration	0A:25:00:F1:FF		
0x0000	0x0025	0xFFF1	Simple Profile Char 1	01		Rd Wr 0x0A
0x0000	0x0026	0x2901	Characteristic User Description	Characteristic 1		
0x0000	0x0027	0x2803	GATT Characteristic Declaration	02:28:00:F2:FF		
0x0000	0x0028	0xFFF2	Simple Profile Char 2	02		Rd 0x02
0x0000	0x0029	0x2901	Characteristic User Description	Characteristic 2		
0x0000	0x002A	0x2803	GATT Characteristic Declaration	08:2B:00:F3:FF		
0x0000	0x002B	0xFFF3	Simple Profile Char 3			Wr 0x08
0x0000	0x002C	0x2901	Characteristic User Description	Characteristic 3		
0x0000	0x002D	0x2803	GATT Characteristic Declaration	10:2E:00:F4:FF		
0x0000	0x002E	0xFFF4	Simple Profile Char 4			Nfy 0x10
0x0000	0x002F	0x2902	Client Characteristic Configuration	00:00		
0x0000	0x0030	0x2901	Characteristic User Description	Characteristic 4		

7.5 Lectura de una característica

A partir de su Handle:

Por ejemplo, para leer el valor de la característica 1, cuyo handle es 0x0025 tal y como se ve en la tabla anterior, debemos seleccionar la opción Read Characteristic Value en la ventana de lectura y presionar el botón “Read”.



Si aparece el mensaje “Sucess” la operación habrá sido correcta y en la ventana de arriba tendremos el valor de esa característica, en éste caso 01.

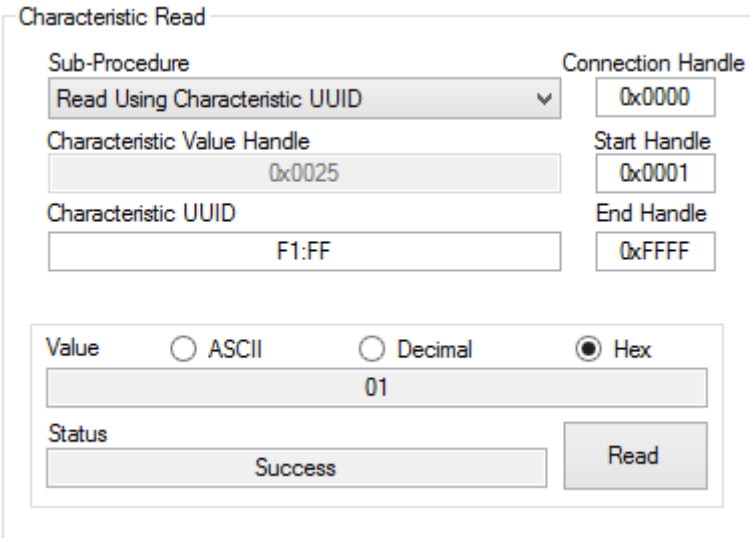
En la ventana de log, donde podemos ver los mensajes intercambiados entre el USB Dongle y el dispositivo tendremos los siguientes paquetes:

```
------------------------------------------------------------------------
[301] : <Tx> - 08:06:24.991
-Type            : 0x01 (Command)
-OpCode          : 0xFD8A (GATT_ReadCharValue)
-Data Length     : 0x04 (4) byte(s)
 ConnHandle      : 0x0000 (0)
 Handle          : 0x0025 (37)
Dump(Tx):
0000:01 8A FD 04 00 00 25 00                        ......%.
------------------------------------------------------------------------
[302] : <Rx> - 08:06:25.022
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x06 (6) bytes(s)
 Event           : 0x067F (1663) (GAP_HCI_ExtentionCommandStatus)
 Status          : 0x00 (0) (Success)
 OpCode          : 0xFD8A (GATT_ReadCharValue)
 DataLength      : 0x00 (0)
Dump(Rx):
0000:04 FF 06 7F 06 00 8A FD 00                     .........
------------------------------------------------------------------------
[303] : <Rx> - 08:06:26.976
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x07 (7) bytes(s)
 Event           : 0x050B (1291) (ATT_ReadRsp)
 Status          : 0x00 (0) (Success)
 ConnHandle      : 0x0000 (0)
 PduLen          : 0x01 (1)
 Value           : 01
Dump(Rx):
0000:04 FF 07 0B 05 00 00 00 01 01                  ..........
------------------------------------------------------------------------
```

El primero es la petición de lectura GATT_ReadCharValue conteniendo el valor del handle a leer y el último es la respuesta ATT_ReadRsp conteniendo el valor de la lectura (01).

A partir de su UUID:

También podemos leer una característica a partir de su UUID, en éste caso, el UUID de la característica 1 es FFF1 (Definido por el usuario en SimpleBLEPeripheral)
Para efectuar la lectura hay que seleccionar la pestaña “Read Using Characteristic UUID” e introducir el UUID a leer:



Vemos que el valor leído sigue siendo 01.

En la ventana de log se generan los siguientes mensajes:

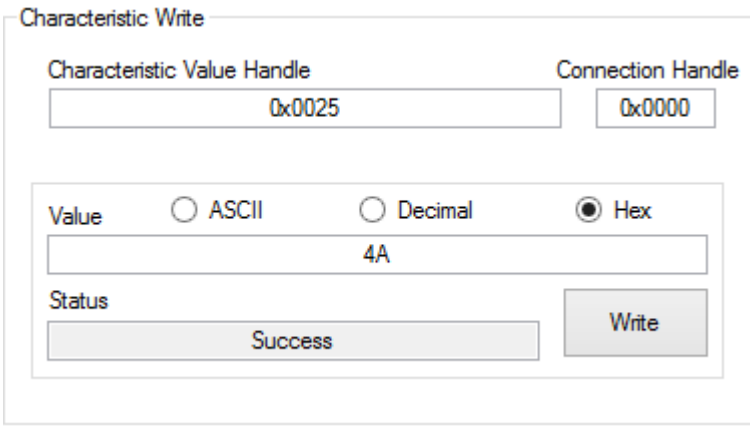
```
------------------------------------------------------------------------
[313] : <Tx> - 08:28:14.367
-Type            : 0x01 (Command)
-OpCode          : 0xFDB4 (GATT_ReadUsingCharUUID)
-Data Length     : 0x08 (8) byte(s)
 ConnHandle      : 0x0000 (0)
 StartHandle     : 0x0001 (1)
 EndHandle       : 0xFFFF (65535)
 Type            : F1:FF
Dump(Tx):
0000:01 B4 FD 08 00 00 01 00 FF FF F1 FF             ............
------------------------------------------------------------------------
[314] : <Rx> - 08:28:14.414
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x06 (6) bytes(s)
 Event           : 0x067F (1663) (GAP_HCI_ExtentionCommandStatus)
 Status          : 0x00 (0) (Success)
 OpCode          : 0xFDB4 (GATT_ReadUsingCharUUID)
 DataLength      : 0x00 (0)
Dump(Rx):
0000:04 FF 06 7F 06 00 B4 FD 00                      .........
------------------------------------------------------------------------
```

```
[319] : <Rx> - 08:28:26.119
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x0A (10) bytes(s)
 Event           : 0x0509 (1289) (ATT_ReadByTypeRsp)
 Status          : 0x00 (0) (Success)
 ConnHandle      : 0x0000 (0)
 PduLen          : 0x04 (4)
 Length          : 0x03 (3)
 Handle          : 0025
 Data            : 01
Dump(Rx):
0000:04 FF 0A 09 05 00 00 00 04 03 25 00 01          ..........%..
-------------------------------------------------------------------------
[320] : <Rx> - 08:28:27.963
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x06 (6) bytes(s)
 Event           : 0x0509 (1289) (ATT_ReadByTypeRsp)
 Status          : 0x1A (26) (The Procedure Is Completed)
 ConnHandle      : 0x0000 (0)
 PduLen          : 0x00 (0)
Dump(Rx):
0000:04 FF 06 09 05 1A 00 00 00                      .........
-------------------------------------------------------------------------
```

7.6 Escritura de una característica

Para escribir una carácteristica hay que indicar cuál es su Handle, en éste caso, para escribir en la característica 1 indicaremos que su Handle es 0x0025, una vez hayamos introducido el valor a escribir en la ventana se presiona el botón “Write”, en éste caso el valor a escribir será 0x4A.

Si aparece el mensaje Sucess la escritura habrá sido correcta.

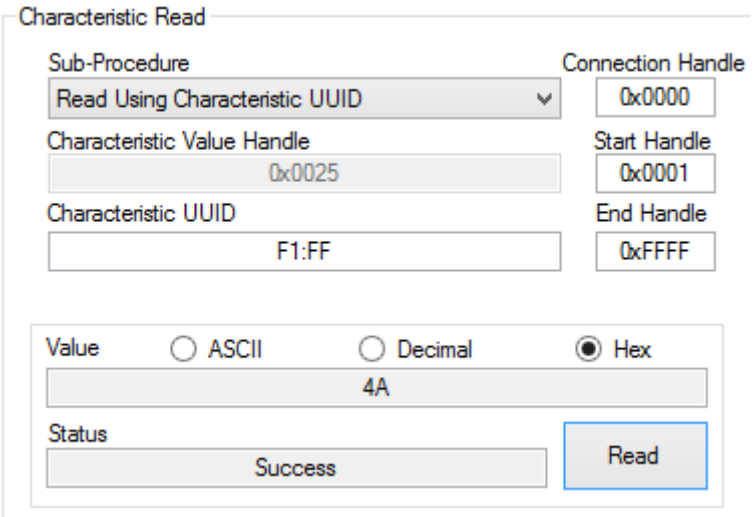


En la ventana de Log podremos ver los siguientes mensajes intercambiados entre Dongle y dispositivo correspondientes a una operación de escritura:

```
------------------------------------------------------------------------
[339] : <Tx> - 08:40:23.216
-Type            : 0x01 (Command)
-OpCode          : 0xFD92 (GATT_WriteCharValue)
-Data Length     : 0x05 (5) byte(s)
 ConnHandle      : 0x0000 (0)
 Handle          : 0x0025 (37)
 Value           : 4A
Dump(Tx):
0000:01 92 FD 05 00 00 25 00 4A                 |     ......%.J
------------------------------------------------------------------------
[340] : <Rx> - 08:40:23.247
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x06 (6) bytes(s)
 Event           : 0x067F (1663) (GAP_HCI_ExtentionCommandStatus)
 Status          : 0x00 (0) (Success)
 OpCode          : 0xFD92 (GATT_WriteCharValue)
 DataLength      : 0x00 (0)
Dump(Rx):
0000:04 FF 06 7F 06 00 92 FD 00                       .........
------------------------------------------------------------------------
[341] : <Rx> - 08:40:24.950
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x06 (6) bytes(s)
 Event           : 0x0513 (1299) (ATT_WriteRsp)
 Status          : 0x00 (0) (Success)
 ConnHandle      : 0x0000 (0)
 PduLen          : 0x00 (0)
Dump(Rx):
0000:04 FF 06 13 05 00 00 00 00                       .........
------------------------------------------------------------------------
```

El primer mensaje es un comando GATT_WriteCharValue con el valor a escribir (4A), el último es el mensaje de confirmación ATT_WriteRsp.

Una vez escrita la característica podemos volver a leer su valor para comprobar que efectivamente éste ha cambiado:

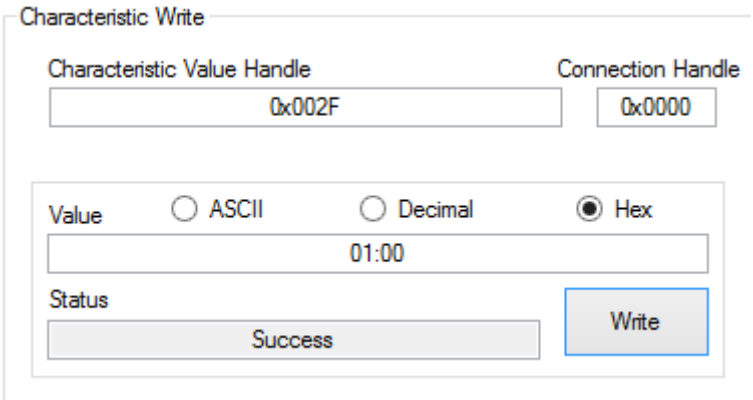


```
------------------------------------------------------------------------
[346] : <Tx> - 08:42:49.534
-Type            : 0x01 (Command)
-OpCode          : 0xFD8A (GATT_ReadCharValue)
-Data Length     : 0x04 (4) byte(s)
 ConnHandle      : 0x0000 (0)
 Handle          : 0x0025 (37)
Dump(Tx):
0000:01 8A FD 04 00 00 25 00                        ......%.
------------------------------------------------------------------------
[347] : <Rx> - 08:42:49.550
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x06 (6) bytes(s)
 Event           : 0x067F (1663) (GAP_HCI_ExtentionCommandStatus)
 Status          : 0x00 (0) (Success)
 OpCode          : 0xFD8A (GATT_ReadCharValue)
 DataLength      : 0x00 (0)
Dump(Rx):
0000:04 FF 06 7F 06 00 8A FD 00                     .........
------------------------------------------------------------------------
[348] : <Rx> - 08:42:50.956
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x07 (7) bytes(s)
 Event           : 0x050B (1291) (ATT_ReadRsp)
 Status          : 0x00 (0) (Success)
 ConnHandle      : 0x0000 (0)
 PduLen          : 0x01 (1)
 Value           : 4A
Dump(Rx):
0000:04 FF 07 0B 05 00 00 00 01 4A                  .........J
------------------------------------------------------------------------
```

Habilitación de notificaciones en una característica:

Si queremos habilitar la opción Notify en una característica que lo permita para que, periódicamente el dispositivo nos envíe información de su valor, hemos de escribir 0001 en el campo Client Característic Configuration de la misma:

ConHnd	Handle	Uuid	Uuid Description	Value	Value Description	Properties
0x0000	0x002D	0x2803	GATT Characteristic Declaration	10:2E:00:F4:FF		
0x0000	0x002E	0xFFF4	Simple Profile Char 4	03		Nfy 0x10
0x0000	0x002F	0x2902	Client Characteristic Configuration	00:00		
0x0000	0x0030	0x2901	Characteristic User Description	Characteristic 4		



Una vez habilitada la notificación en la ventana de log aparecerán todos los mensajes recibidos desde el dispositivo:

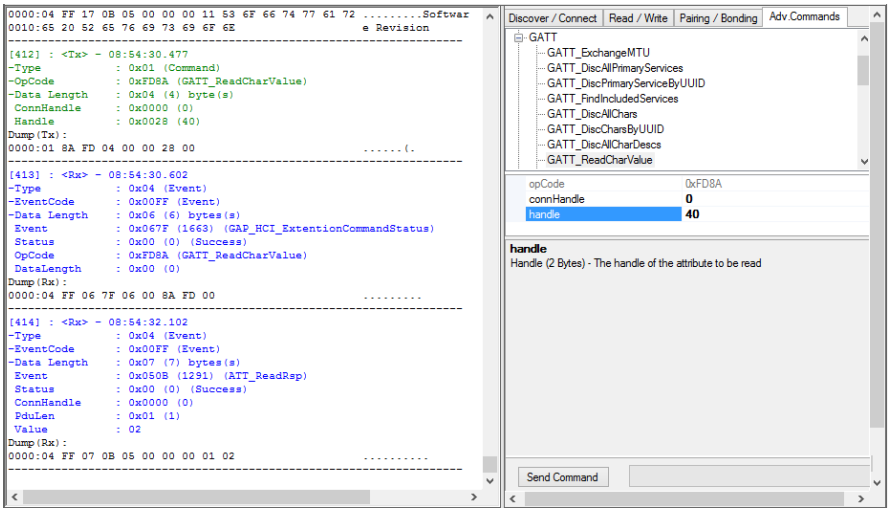
```
------------------------------------------------------------------------
[349] : <Tx> - 08:44:17.247
-Type            : 0x01 (Command)
-OpCode          : 0xFD92 (GATT_WriteCharValue)
-Data Length     : 0x06 (6) byte(s)
 ConnHandle      : 0x0000 (0)
 Handle          : 0x002F (47)
 Value           : 01:00
Dump(Tx):
0000:01 92 FD 06 00 00 2F 00 01 00                      ....../...
------------------------------------------------------------------------
[350] : <Rx> - 08:44:17.403
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x06 (6) bytes(s)
 Event           : 0x067F (1663) (GAP_HCI_ExtentionCommandStatus)
 Status          : 0x00 (0) (Success)
 OpCode          : 0xFD92 (GATT_WriteCharValue)
 DataLength      : 0x00 (0)
Dump(Rx):
0000:04 FF 06 7F 06 00 92 FD 00                         .........
------------------------------------------------------------------------
[351] : <Rx> - 08:44:18.950
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x06 (6) bytes(s)
 Event           : 0x0513 (1299) (ATT_WriteRsp)
 Status          : 0x00 (0) (Success)
 ConnHandle      : 0x0000 (0)
 PduLen          : 0x00 (0)
Dump(Rx):
0000:04 FF 06 13 05 00 00 00 00                         .........
------------------------------------------------------------------------

------------------------------------------------------------------------
[352] : <Rx> - 08:44:23.107
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x09 (9) bytes(s)
 Event           : 0x051B (1307) (ATT_HandleValueNotification)
 Status          : 0x00 (0) (Success)
 ConnHandle      : 0x0000 (0)
 PduLen          : 0x03 (3)
 Handle          : 0x002E (46)
 Value           : 03
Dump(Rx):
0000:04 FF 09 1B 05 00 00 00 03 2E 00 03                ............
------------------------------------------------------------------------
[353] : <Rx> - 08:44:27.952
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x09 (9) bytes(s)
 Event           : 0x051B (1307) (ATT_HandleValueNotification)
 Status          : 0x00 (0) (Success)
 ConnHandle      : 0x0000 (0)
 PduLen          : 0x03 (3)
 Handle          : 0x002E (46)
 Value           : 03
Dump(Rx):
0000:04 FF 09 1B 05 00 00 00 03 2E 00 03                ............
------------------------------------------------------------------------
```

```
[354] : <Rx> - 08:44:32.953
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x09 (9) bytes(s)
 Event           : 0x051B (1307) (ATT_HandleValueNotification)
 Status          : 0x00 (0) (Success)
 ConnHandle      : 0x0000 (0)
 PduLen          : 0x03 (3)
 Handle          : 0x002E (46)
 Value           : 03
Dump(Rx):
0000:04 FF 09 1B 05 00 00 00 03 2E 00 03              ............
--------------------------------------------------------------------------
[355] : <Rx> - 08:44:38.104
-Type            : 0x04 (Event)
-EventCode       : 0x00FF (Event)
-Data Length     : 0x09 (9) bytes(s)
 Event           : 0x051B (1307) (ATT_HandleValueNotification)
 Status          : 0x00 (0) (Success)
 ConnHandle      : 0x0000 (0)
 PduLen          : 0x03 (3)
 Handle          : 0x002E (46)
 Value           : 03
Dump(Rx):
0000:04 FF 09 1B 05 00 00 00 03 2E 00 03              ............
```

Se observa que todos los paquetes Notify contienen el mismo valor de la carácteristica 4, en éste caso 03.

Lista de comandos.

En la pestaña Adv Commands disponemos de un listado completo de todos los comandos disponibles para cada nivel del stack de BLE, seleccionando cualquiera de ellos podemos crear un paquete “customizado” con el contenido que nosotros queramos, como ejemplo se va a crear un paquete de solicitud de lectura, Seleccionando el paquete GATT_ReadCharValue se completan los campos del Handle deseado a leer y se presiona el botón “Send Command”, a la izquierda podemos ver el resultado con el paquete enviado y su respuesta:



Capítulo 8

Ejemplo de aplicación: BEACON

Un beacon o baliza es un dispositivo de bajo consumo que emite una señal broadcast. Utiliza la tecnología Bluetooth de bajo consumo (BLE) para transmitir mensajes o avisos directamente a un dispositivo móvil sin la necesidad de una sincronización de los aparatos.

A diferencia del GPS, los beacons pueden ser utilizados para la localización dentro de un entorno cerrado. Existen numerosas aplicaciones como el marketing, siendo uno de los usos mayoritarios. Otras como el servicio a clientes basadas en la localización o asistencia personalizada.

8.1 Medición del RSSI en los Advertising Packets

Para la medida de la potencia de la señal de los paquetes de *advertising* se ha configurado un dispositivo en modo Broadcaster que se comporta como una baliza, enviando continuamente paquetes que serán recibidos por un dispositivo en modo Observer.

Poblando una white list en el Observer con la dirección de todas las balizas a las que se desea escuchar, obtendremos la potencia de la señal de todos los paquetes recibidos para posteriormente poder calcular la distancia a las que se encuentran, ignorando todo aquel paquete que no provenga de una de las balizas del sistema, esto requiere de un "Observador continuo", si bien éste modo no está especificado en el stack BLE de TI, mediante software es fácilmente realizable.

Se han configurado dos dispositivos, un *SensorTag* configurado como baliza y un CC2541 de la placa de pruebas como Observer (para poder transmitir vía UART los datos de los paquetes recibidos).

8.1.1 Sensor Tag como baliza (Rol Broadcaster)

Existen diferentes protocolos de balizas BLE en el mercado (iBeacon, AltBeacon, Tessel Beacon…). Una característica común a muchos de ellos es la transmisión dentro del paquete de la potencia de la señal recibida a 1 metro del dispositivo para posteriormente comparar ese valor con el RSSI del paquete y poder calcular la distancia aproximada.

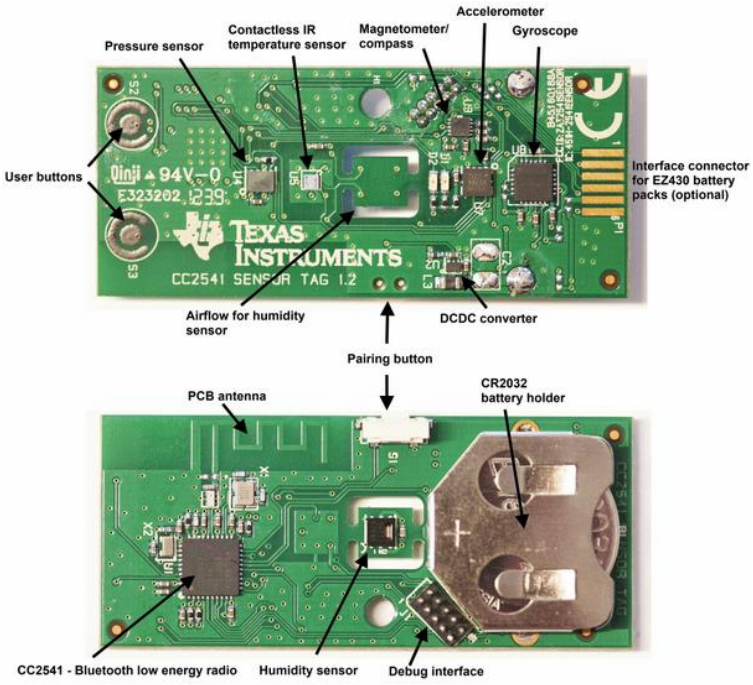
Para la creación del *advertising packet* que va a emitir el dispositivo Broadcaster (Sensor Tag), se hen seguido las especificaciones de la siguiente web [5]:

http://www.pubnub.com/blog/create-a-tessel-beacon-ibeacon-with-a-ble-module-tutorial-overview

Scan Response Data														
AD Structure 1			AD Structure 2											
0x02	0x01	0x1A	0x1B	0xFF	0xE0	0x00	0xBE	0xAC	0x0C	...	0xBB	0x00	0x09	0x00	0x06	0xBA	0x00
Remaining length	AD type	Data	Remaining length	AD Type	Manufacturer ID		Beacon Prefix		UUID (16 bytes)			Major		Minor		TX Power	
2	Flags	Flag	27	Manufacturer Specific	224: Google							9		6		-70	

Paquete Beacon. Tessel.

El SensorTag [6] es un dispositivo creado por Texas Instruments para el desarrollo de aplicaciones inalámbricas, consta de varios sensores y un CC2541 que será programado a través del CC Debugger.



Ésta es la información que se enviará en cada *Advertising Packet*, sigue básicamente la filosofía de las iBeacon de Apple, variando el código del fabricante, lo importante es recibir la información de la potencia a 1 metro (la cual varía según el dispositivo, -70dB en el ejemplo).

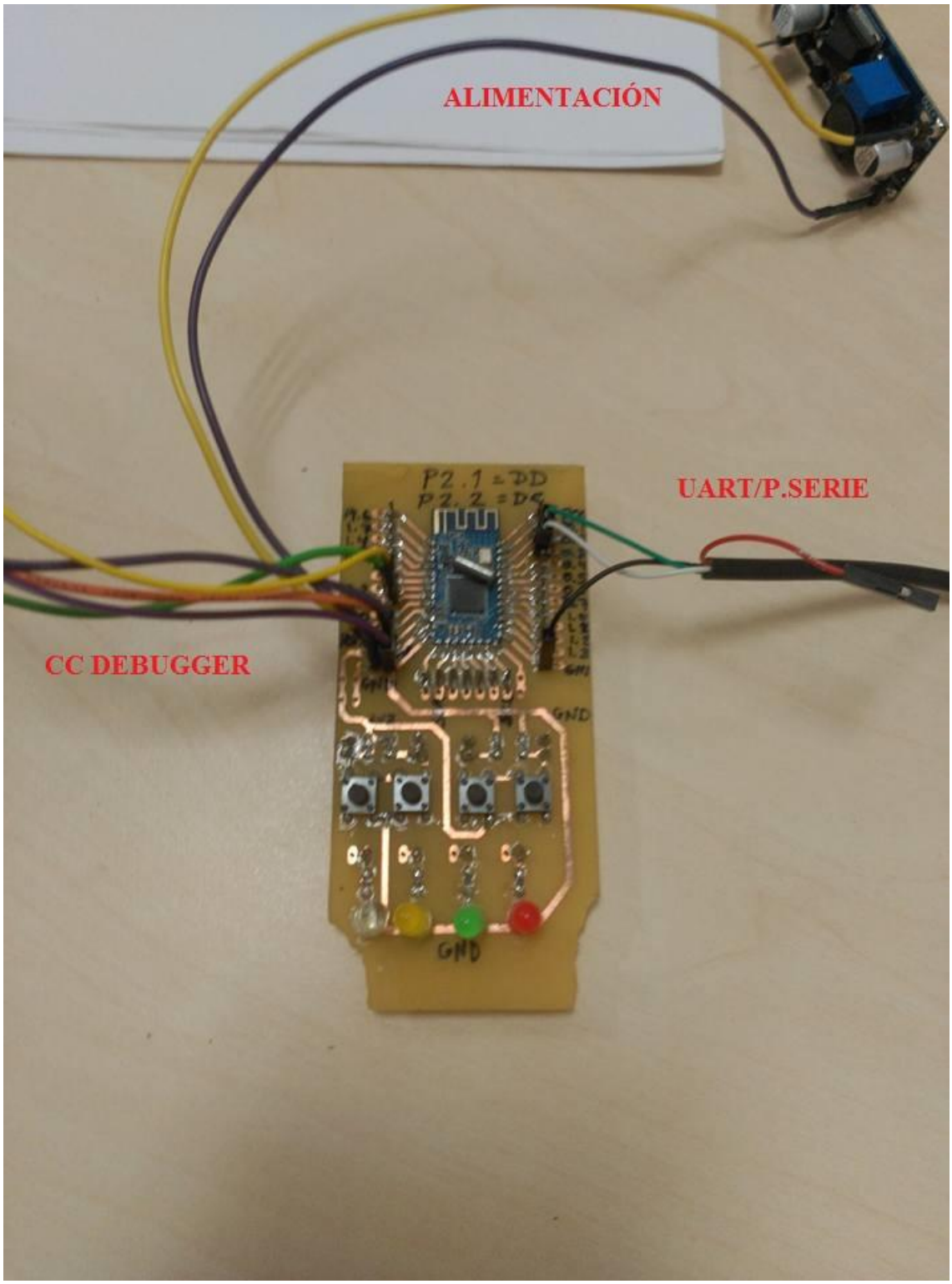
```
static uint8 advertData[] =
{
  0x02,   // length of this data
  GAP_ADTYPE_FLAGS,
  0x1A,  //GAP_ADTYPE_FLAGS_GENERAL,
  0x1B,  //longitud de la baliza
  GAP_ADTYPE_MANUFACTURER_SPECIFIC, // 0xFF para fabricante
  0x47,
  0x00,//esto es supuestamente de bluegiga
  0xBE,
  0xAC,
  0x01,
  0x02,
  0x03,
  0x04,
  0x05,
  0x06,
  0x07,
  0x08,
  0x09,
  0x10,
  0x11,
  0x12,
  0x13,
  0x14,
  0x15,
  0x16,
  0x00,
  0x09,
  0x00,
  0x06,
  0xBA,
  0x00
};
```

Paquete adv. aplicación

Como UUID de la baliza será una secuencia de números del 1 al 16, los cuatro siguientes bytes sirven para identificar a la baliza dentro de un conjunto mayor de balizas.

8.1.2 CC2541 de la tarjeta de pruebas (Modo Observer)

Se han creado dos tarjetas de pruebas para el desarrollo de aplicaciones, constan cada una de un CC2541, varios pines de entrada y salida así como 4 LEDs y 4 Switches. Son programadas usando el CC Debugger.



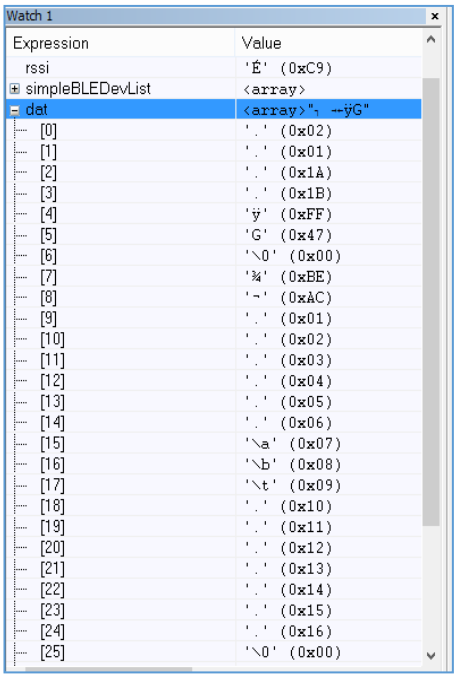
Se ha configurado un observador continuo para capturar todos los paquetes de las diferentes balizas, para ello hay que reiniciar el escaneo cada vez que se acaba su timeout (hay que llamar a la función **GAPObserverRole_StartDiscovery();** cada vez que se produce un evento del tipo **GAP_DEVICE_DISCOVERY_EVENT**).

Cada vez que se "escucha" un nuevo paquete se produce un evento del tipo **GAP_DEVICE_INFO_EVENT**, es en ese momento cuando se captura la información del payload data y se obtiene el RSSI del mismo:

```
case GAP_DEVICE_INFO_EVENT:
  {

    osal_memcpy( data, pEvent->deviceInfo.pEvtData, pEvent->deviceInfo.dataLen );

    osal_memcpy( dat,data, pEvent->deviceInfo.dataLen );

    rssi = pEvent->deviceInfo.rssi;

    simpleBLEAddDeviceInfo( pEvent->deviceInfo.addr, pEvent->deviceInfo.addrType );


  }
  break;
```

A7: Evento Info Device.

El valor del RSSI se encuentra dentro de la estructura deviceInfo.rssi y los datos del paquete en deviceInfo.pEvtData, de éste modo los datos capturados y el valor del RSSI de un paquete son los siguientes:



Lectura paquete desde el IAR Compiler

Coinciden con el contenido del Adv Packet configurado en el Broadcaster.

En la siguiente tabla se recogen los valores de RSSI para 5 paquetes a diferentes distancias del Broadcaster:

Juntos		**30 cm**		**60 cm**		**1 metro**	
Hex	**dB**	**Hex**	**dB**	**Hex**	**dB**	**Hex**	**dB**
0xDA	-38 dB	0xD1	-47 dB	0xB5	-75 dB	0xB8	-72 dB
0xD2	-46 dB	0xD8	-40 dB	0xC3	-45 dB	0xAA	-86 dB
0xD0	-48 dB	0xD3	-45 dB	0xBA	-70 dB	0xB6	-74 dB
0xD5	-43 dB	0xD8	-40 dB	0xC3	-61 dB	0xBE	-66 dB
0xDC	-36 dB	0xD3	-45 dB	0xB7	-73 dB	0xB9	-71 dB

Tabla 8.1: RSSI de 5 paquetes distintos

Haciendo una media de los 5 paquetes:

Distancia	**RSSI**
Juntos	-42,2 dB
30 cm	-43,4 dB
60 cm	-64,8 dB
1 m	-73,8 dB

Tabla 8.2: Media del RSSI de los paquetes capturados

Nota: Los datos en hexadecimal están en complemento a dos, para obtener su valor en dB hay que restarle 256 a su valor decimal.

Finalmente se envían los datos de cada paquete recibido vía UART al PC para posteriormente poder realizar los cálculos de obtención de la distancia. Los resultados capturados por el puerto COM3 son los siguientes:

```
[20/09/2015 19:12:46] Read data (COM3)
    be ac 35 cf                                          ¾¬5Ï
[20/09/2015 19:12:47] Read data (COM3)
    be ac 35 d0                                          ¾¬5Ð
[20/09/2015 19:12:47] Read data (COM3)
    be ac 35 d1                                          ¾¬5Ñ
[20/09/2015 19:12:47] Read data (COM3)
    be ac 36 d1                                          ¾¬6Ñ
[20/09/2015 19:12:48] Read data (COM3)
    be ac 36 d1                                          ¾¬6Ñ
[20/09/2015 19:12:48] Read data (COM3)
    be ac 36 cf                                          ¾¬6Ï
[20/09/2015 19:12:49] Read data (COM3)
    be ac 37 cf                                          ¾¬7Ï
```

Lectura puerto COM PC

Los dos primeros bytes son el código BEAC del protocolo Tessel Beacon enviados en los bytes 7 y 8 de los *advertising packets*, el tercer byte es un contador de una función periódica del broadcaster, enviado en el byte 10 del paquete y el último byte corresponde con el RSSI del paquete.

8.2 Recepción de paquetes enviados desde varias beacons simultáneamente

El objetivo es la captura por parte de un dispositivo en modo Observer de los paquetes enviados por varias beacons.

El dispositivo Observer está continuamente realizando una operación de descubrimiento (Discovery). La duración de la operación de descubrimiento es configurable por software, actualmente el dispositivo realiza un Scan cada 400 ms, produciéndose un evento del tipo **GAP_DEVICE_INFO_EVENT** cada vez que se recibe un nuevo paquete, en ese momento se envía vía UART la información contenida en en su payload así como su RSSI.

Cuando se termina una operación de discovery (400ms) se produce un evento del tipo **GAP_DEVICE_DISCOVERY_EVENT**, iniciándose en ese momento el siguiente discovery mediante la función **GAPObserverRole_StartDiscovery();**

Se han programado dos beacons, modificando el contenido de sus paquetes para otorgar un identificador a cada una de ellas.

El byte 9 de cada uno de los paquetes se corresponde a ese identificador, siendo 0x55 para una y 0xFF para otra. Para añadir más dispositivos al sistema simplemente bastaría con crear un nuevo identificador.

```
// GAP - Advertisement data (max size = 31 bytes, though this is
// best kept short to conserve power while advertisting)
static uint8 advertData[] =
{
  0x02,   // length of this data
  GAP_ADTYPE_FLAGS,
  0x1A,  //GAP_ADTYPE_FLAGS_GENERAL,
  0x1B,  //longitud de la baliza
  GAP_ADTYPE_MANUFACTURER_SPECIFIC, // 0xFF para fabricante
  0x47,
  0x00,//esto es supuestamente de bluegiga
  0xBE,
  0xAC,
  0x55,//este va a ser el identificador(FF,55)
  0x02,
  0x03,
```

Identificador baliza.

Como se ha visto antes, se envían vía UART 3 bytes de cada paquete, el 7, 8 y 9, siendo el 7 y 8 el código BEAC de beacon y el 9 el identificador así como el RSSI del paquete en cuestión.

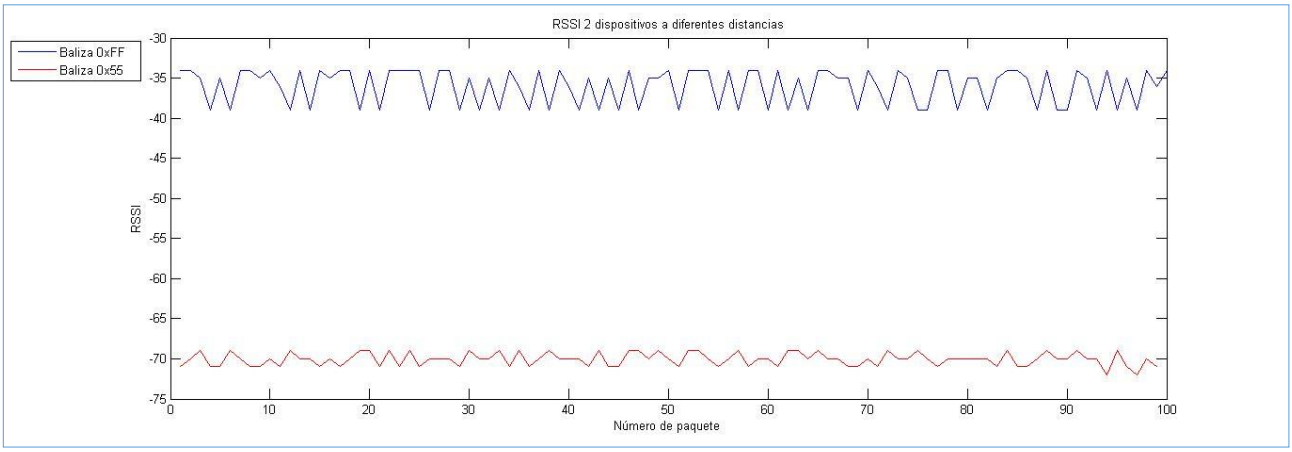
Para leer éstos datos se ha creado un script de matlab que separa cada RSSI según a que dispositivo corresponda, para posteriormente mostrar el valor de su RSSI gráficamente.

```
close all;
resultbaliza1 = 0
resultbaliza2 = 0
newobjs = instrfind ; fclose(newobjs);
serialUart=serial('COM4', 'BaudRate', 19200);
set(serialUart, 'InputBufferSize', 4); %numero de bytes en buffer
set(serialUart, 'Parity', 'none');
set(serialUart, 'DataBits', 8);
set(serialUart, 'StopBit', 1);
i=1
j=1
k=1
fopen(serialUart);
while i<20
databin = fread(serialUart);
if databin(3) == 255;%Baliza con identificador 0xFF
resultbaliza1(j) = databin(4) - 256;
plot(resultbaliza1,'b')
hold on
j = j+1;
pause(0.1)
elseif databin(3) == 85;%Baliza con identificador 0x55
resultbaliza2(k) = databin(4) - 256;
plot(resultbaliza2,'r')
hold on
k = k+1;
pause(0.1)
end
i = i+1;
end
fclose(serialUart);
```

A8: Script MATLAB.

Las gráficas se actualizan con cada iteración por lo que es bastante interesante ver el valor de los RSSI en tiempo real.

A continuación se muestra una gráfica de los resultados para 200 iteraciones del script de la captura de paquetes de dos balizas (100 paquetes de cada una), una situada junto al Observer y otra a unos 2 metros:



RSSI de 2 balizas a diferentes distancias.

8.3 Posicionamiento y balizas BLE

Se dispone de 3 balizas BLE y un receptor, el objetivo es calcular la distancia aproximada del receptor a cada una de las balizas para así poder encontrar su posición aproximada.

8.3.1 Cálculo de la distancia a partir del RSSI

En BLE el cálculo de distancias a partir del RSSI no es algo exacto, la potencia de señal recibida es muy susceptible a cambios en el entorno, posición de las antenas, orientación, obstáculos... En todas las medidas que se han realizado el valor de RSSI oscila considerablemente por lo que es necesario hacer algún tipo de filtrado para intentar estabilizarlo, lo más simple es tomar las máximas muestras posibles y calcular su media.

La siguiente fórmula permite calcular el valor de la distancia en metros a partir del RSSI [8]:

$$RSSI(dBm) = -10n\log_{10}(d) + A$$

Siendo:

d: Distancia en metros.
A: Potencia de la señal recibida en dBm a 1 metro del dispositivo.
n: Constante de propagación en el aire n=2.

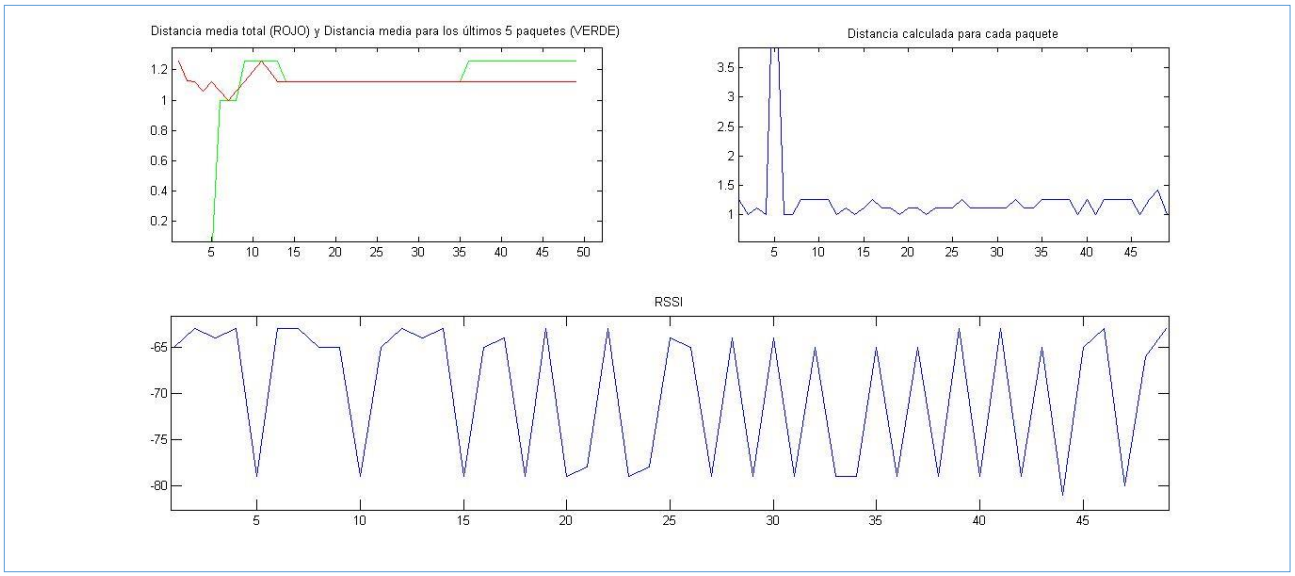
Lo primero es conocer cuál es la potencia de señal recibida a 1 metro del dispositivo, para ello se realiza una medición del RSSI a esa distancia para cada una de las balizas, ya que ese valor varía de una a otra.

Se ha medido el RSSI de cada baliza durante 500 paquetes recibidos y luego se ha realizado su media obteniendo los siguientes resultados:

BALIZA	RSSI
BALIZA AZUL (Dir 0xFF)	-63 dBm
BALIZA ROJA (Dir 0x55)	-65 dBm
BALIZA VERDE (Dir 0x26)	-61 dBm

Tabla 8.3: Media del RSSI para 500 paquetes según baliza

Una vez obtenido el valor de A y manteniendo las mismas condiciones del entorno en el momento de la medición procedo a aplicar la fórmula para comprobar su validez:



RSSI y distancia.

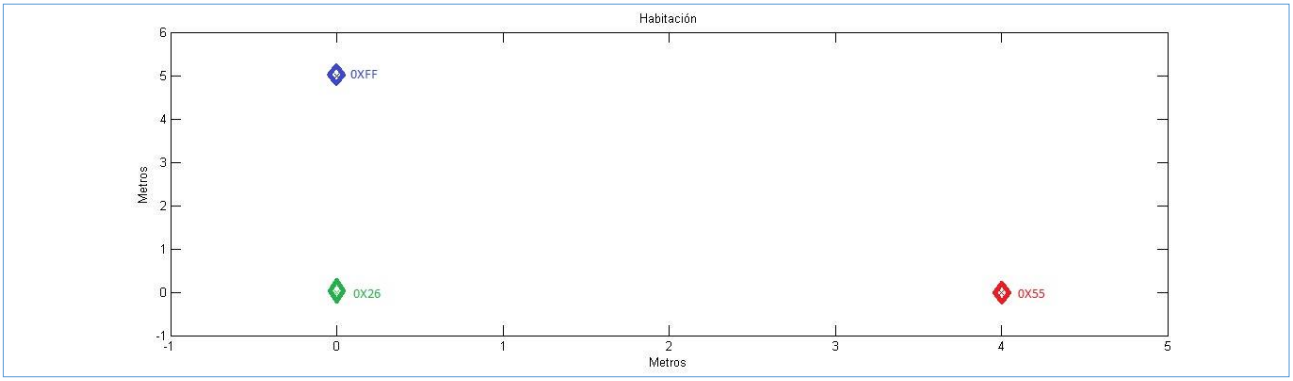
A pesar de producirse picos en el cálculo de la distancia instantánea (gráfica superior derecha) en la distancia media total obtenemos un valor bastante cercano a 1 metro (gráfico rojo en la gráfica superior izquierda)

Para intentar suavizar más la gráfica de la distancia se ha programado el script de matlab para que rechace todos aquellos valores que se desvíen más de un 80% de la media.

8.3.2 Cálculo de la posición aproximada

Disponiendo de 3 balizas y pudiendo hallar un valor de la distancia aproximado del dispositivo a cada una de ellas mi idea es intentar calcular una posición aproximada del mismo.

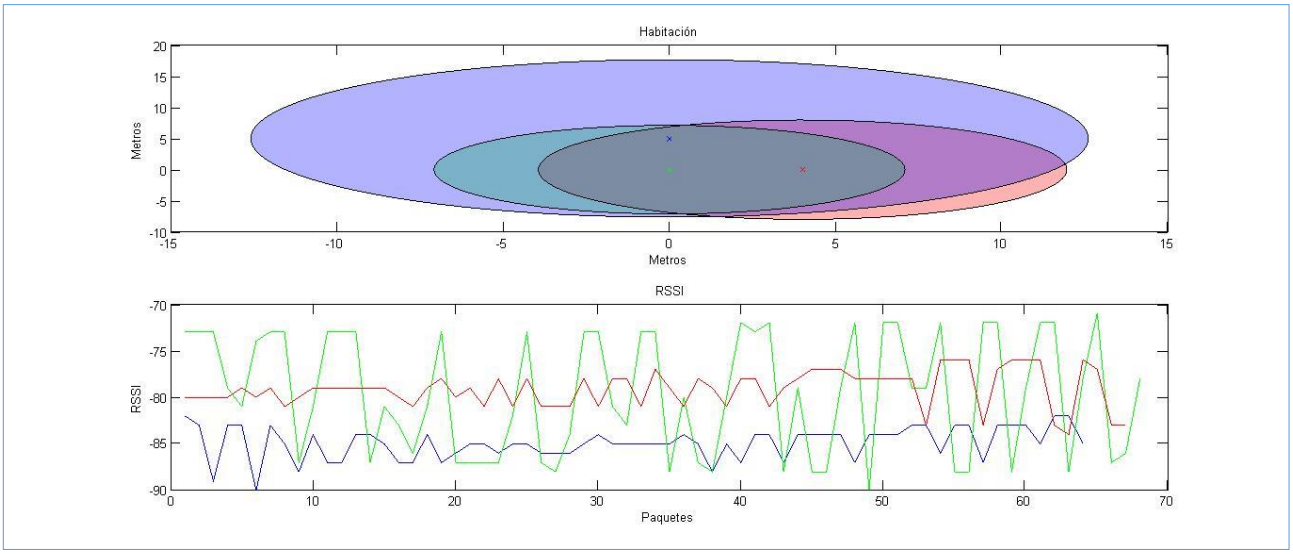
Para ello se han situado las 3 balizas en una habitación de la siguiente manera:



Disposición balizas.

Se ha desarrollado un script en matlab para trazar un circulo alrededor de cada baliza cuyo radio sea el valor de la distancia aproximada a esa baliza, de ese modo en aquella zona en la que las circunferencias se solapen será donde con mayor probabilidad se encontrará el receptor.

Situando el receptor entre las balizas verde y roja se ha obtenido el siguiente resultado:



RSSI y localización aproximada.

Realizando así las medidas no se puede calcular una posición dentro del área delimitado por las tres balizas, si es cierto que el receptor se encuentra dentro del área más oscurecida pero no es suficiente.

Disponiendo de más espacio entre balizas y de un número mayor de ellas podría mejorarse el resultado.

8.3.3 Conclusiones

Según la habitación donde estén y los elementos o personas que haya en ella el valor de la potencia recibida a 1 metro varía por lo que hay que estar modificando los valores del script para poder hallar los valores de la distancia, las medidas se han realizado en un garaje con una moto y muchos trastos de por medio..., También puede que influya la situación de las balizas, siempre se han dejado en el suelo, orientándolas frontalmente desde una pared a una determinada altura los resultados mejorarían.

Pliego de condiciones

Requisitos de Hardware

- PC Compatible HP 5500U i7
- Smartphone Android 4.4.4
- SensorTAG
- USB Dongle
- Placas de desarrollo
- CC Debugger

Requisitos de Software

- Sistema operativo Windows 10 Home
- Procesador de textos Microsoft Word 2016
- Compilador IAR Embedded Workbench para 8051 versión 8.30
- TI Packet Sniffer
- BTool
- MATLAB versión R2012

Presupuesto

En este apartado se adjunta un desglose de los costes estimados para este proyecto en cuanto a hardware, software y mano de obra se refiere

Concepto	*Precio unitario*	*Cantidad*	*Amortización 6 meses al 26%*	*Subtotal*
PC HP	699 €	1	90.87 €	90.87€
SmartPhone Android	199 €	1	25.87 €	25.87€
SensorTag	29 €	2	7.54 €	7.54€
USB Dongle	49 €	1	6.37 €	6.37€
Placas de desarrollo	20 €	2	No aplica	40€
CC Debugger	49 €	1	6.37 €	6.37€
Subtotal final				**177.02€**

Tabla B.1: Costes de Hardware

Concepto	*Precio unitario*	*Cantidad*	*Amortización 6 meses al 26%*	*Subtotal*
S.O. Windows 10	135€	1	17.55€	17.55€
IAR Embedded Workbench	2995€	1	389.35€	389.35€
MATLAB Student	69€	1	8.97€	8.97€
BTool	0€	1	No aplica	0€
TI Packet Sniffer	0€	1	No aplica	0€
Subtotal final				**415.87€**

Tabla B.2: Costes de Software

Concepto	*Precio unitario*	*Cantidad*	*Subtotal*
Ingeniería	90€	320	28800€
Mecanografía	40€	30	1200€
Subtotal final			**30000€**

Tabla B.3: Costes de mano de obra

Concepto	*Subtotal*
Costes de Hardware	**177.02€**
Costes de Software	**415.87€**
Costes de mano de obra	**30000€**
Subtotal final	**30592.89€**

Tabla B.4: Costes de ejecución material

Gastos generales y beneficio industrial

Los gastos generales y el beneficio industrial son desembolsos obligatorios que proceden de la utilización de las instalaciones de trabajo y los ingresos generados a nivel de manufactura. Para estas cuentas se estima un porcentaje del 16% sobre el coste total de ejecución material:

Concepto	*Subtotal*
Costes de ejecución material	30592.89€
Porcentaje estimado	16%
Subtotal final	**4894.86€**

Tabla B.5: Gastos generales y beneficio industrial

Presupuesto de ejecución por contrata

En este presupuesto se realiza el sumatorio de los resultados de los costes de ejecución material, los gastos generales y el beneficio industrial.

Concepto	*Subtotal*
Presupuesto de ejecución material	12.562,89€
Gastos generales y beneficio industrial	4894.86€
Subtotal final	**17457.75€**

Tabla B.6: Presupuesto de ejecución por contrata

Importe total del presupuesto

En último término, se calcula el presupuesto final del proyecto a partir de los costes totales de ejecución por contrata más el porcentaje de IVA del 21% aplicado.

Concepto	*Subtotal*
Presupuesto de ejecución por contrata	17457.75€
21% de IVA aplicado	3666.13€
Total final	**21123.88€**

Tabla B.7: Importe total del presupuesto

El importe total del presupuesto asciende a: 21123.88€ **(Veintiún mil ciento veintitrés euros)**

Apéndice

En este apartado se van a comentar los principales archivos de los que consta este trabajo

- SimpleBLEPeripheral.c Código fuente del proyecto utilizado para el desarrollo de aplicaciones en rol Peripheral.

- SimpleBLECentral.c Código fuente del proyecto utilizado para el desarrollo de aplicaciones en rol Central.

- SimpleBLEObserver.c Código fuente del proyecto utilizado para el desarrollo de aplicaciones en rol Observer.

- SimpleBLEBroadcaster.c Código fuente del proyecto utilizado para el desarrollo de aplicaciones en rol Broadcaster.

- SimpleGATTProfile.c: Código fuente del protocolo GATT con la configuración de la tabla de atributos y funciones callback.

Bibliografía

[1] Kevin Towmsend, Carles Cufi, Akiba, Robert Davidson, “Getting started with Bluetooth Low Energy”. Ed O`Reilly 2014. pp 1-74

[2] CC2540 and CC2541 Bluetooth low energy Software Developer´s Reference Guide. Texas Instruments. http://www.ti.com/lit/ug/swru271g/swru271g.pdf

[3] Bluetooth Low Energy CC2540/41 Mini Development Kit User´s Guide. Texas Instruments. http://www.ti.com/lit/ug/swru270c/swru270c.pdf

[4] SmartRF Packet Sniffer User´s Manual. Texas Instruments. http://www.ti.com/lit/ug/swru187g/swru187g.pdf

[5] Sunny Gleason & Norvan Sahiner, “Create a Tessel Beacon with a BLE Module”. https://www.pubnub.com/blog/2015-04-07-create-a-tessel-beacon-ibeacon-with-a-ble-module-tutorial-overview/

[6] SensorTag User Guide. Texas Instruments. http://processors.wiki.ti.com/index.php/SensorTag_User_Guide

[7] CC Debugger User´s Guide. Texas Instruments. http://www.ti.com/lit/ug/swru197h/swru197h.pdf

[8] Erin Lau, Boon, Lee, Seung-Chul Lee, Wang-Young Chung. Dongseo University. Busan. Korea.”Enhanced RSSI-Based high accuracy real-time user location tracking system for indoor and outdoor enviroments” pp3

[9] IAR Embedded Workbench IDE User Guide. IAR Systems AB. http://supp.iar.com/FilesPublic/UPDINFO/004916/arm/doc/EWARM_UserGuide.ENU.pdf

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá