Universidad Zaragoza
1542

Trabajo Fin de Grado

Sistema de adquisición de datos cinemáticos de bajo consumo

Autor/es

Raúl Viñals Mariñosa

Directores

Daniele Bibbo, Università Degli Studi di Roma Tre
Sandra Baldassarri, Universidad de Zaragoza

Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Junio 2015

Agradecimientos

Dapprima, voglio ringraziare a tutte le persone del BioLab3, ho imparato tantissime cose con loro e mi hanno fatto sentire come a casa dall'inizio. Specialmente, ringraziare al mio direttore del progetto, Daniele Bibbo, per confidare in me e darmi questa opportunità. A Ivan Bernabucci, per i suoi buoni consigli di programmazione. E naturalmente a mi amico Antonino per la sua constante aiuta e preoccupazione nel mio lavoro.

Ya en Zaragoza, quiero agradecer enormemente a mi padre por todo el tiempo que ha dedicado conmigo al proyecto. Nunca dejaré de aprender de ti papá. Gracias!

También quiero darle las gracias a Sandra, mi directora en Zaragoza, por su tiempo y consejos.

A Rafa Ferrer, fundador de la empresa Neki, por su ayuda y tiempo en los momentos más difíciles.

Por último, agradecer a todas las personas que de alguna manera han hecho posible que casi pueda decir que soy ingeniero, a mi madre, mi hermano, mi novia y mis amigos.

Zaragoza, 24 de Junio de 2015

Resumen

Este proyecto se plantea en la Università degli Studi Roma Tre, más concretamente en el *BioLab3*, laboratorio de electrónica centrado en aplicaciones biomédicas como la rehabilitación física o el alto rendimiento deportivo.

Una parte fundamental en las aplicaciones para monitorizar el movimiento humano consiste en la adquisición de datos inerciales, magnéticos y dinámicos a través de sensores portátiles situados por el cuerpo.

Los datos medidos por los sensores deben ser transmitidos a un dispositivo con capacidad de almacenamiento y procesado, como un PC o un *smartphone*. Hasta el momento, el *BioLab3* enviaba los datos a través de Bluetooth clásico. La principal limitación de esa tecnología es su elevado consumo, que sumado al de los sensores limitaba notablemente la autonomía del conjunto.

Surge de esta forma la necesidad de desarrollar un prototipo de adquisición y transmisión de datos inerciales y magnéticos de bajo consumo. Desde un principio se elige Bluetooth Low Energy (BLE) como alternativa, en parte por la experiencia previa del *BioLab3*, pero fundamentalmente por su soporte tanto en PCs como *smartphones* y por su elevado crecimiento en los últimos años en aplicaciones biomédicas.

El proyecto parte de cero y comienza con una investigación de mercado para elegir los componentes del sistema más competitivos. En la parte de adquisición se selecciona el MPU-9250 de Invensense, un IMU (Inertial Measurement Unit) de última generación que combina un acelerómetro, un giroscopio y un magnetómetro. Para soportar la transmisión la elección se decanta por el módulo Bluetooth Low Energy BL600, de Laird Technologies, en base a su bajo consumo y reducidas dimensiones.

Se ha precisado un elevado esfuerzo inicial y durante todo el proyecto de estudio de la tecnología BLE y de familiarización con el hardware y con el software de los dos extremos de comunicación.

Se han diseñado placas a las que soldar cada componente para facilitar el conexionado de los pines implicados. Se ha montado cada componente en un circuito operativo, que permite además comunicar el módulo BLE y el IMU a través de una interfaz I2C. Para cargar el software en el BL600 y depurar código se ha conectado el módulo BLE al PC a través de la puerta UART y un circuito adaptador de tensiones.

Una vez operativo el hardware, se han desarrollado paralelamente dos aplicaciones que intercambian datos a partir del estándar Bluetooth Low Energy: el "Servidor BLE de Adquisición de Datos" y el "Cliente BLE". La primera aplicación se encarga de la adquisición y transmisión de datos, controlando los sensores (MPU-9250) y el módulo BLE (BL600) y se ha programado en *smartBASIC* (lenguaje nativo del BL600). La segunda aplicación es un sistema de visualización, procesado y almacenamiento de datos alojada en un *smartphone* y programada en lenguaje Java sobre sistema operativo Android.

Se han transmitido con éxito los valores medidos, alcanzando así el objetivo inicial. Se han añadido mejoras no planificadas, como el control remoto del periodo de muestreo o el registro de datos en ficheros almacenados en la memoria externa del *smartphone*. Se ha conseguido transmitir valores a través de tres modos de comunicación BLE diferentes: indicación y lectura, donde los datos viajan desde el módulo BLE hasta el *smartphone*; y escritura, en cuyo caso viajan en sentido contrario.

Además, se ha determinado el periodo de muestreo mínimo alcanzable, relacionándolo con un parámetro crítico de la capa BLE de enlace denominado Intervalo de Conexión. Finalmente, se ha realizado un análisis del consumo energético del sistema en función del periodo de muestreo y se ha estimado la autonomía alcanzable con pilas de botón.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Raúl Viñals Mariñosa,

con nº de DNI 73024451A en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) _____, (Título del Trabajo) Sistema de Adquisición de Datos Cinemáticos de Bajo Consumo

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 25 de Junio de 2015

Fdo: Raúl Viñals Mariñosa

Contenidos

Anexos

CAPÍTULO 1

Introducción

Resumen

En este capítulo se describe cómo surge el proyecto en el Biolab3, laboratorio perteneciente a la Università degli Studi Roma Tre, cuales son las motivaciones que llevan a su desarrollo y los objetivos fijados.

Posteriormente se describe la estructura en Capítulos y Anexos de esta memoria.

1.1. Contexto y motivación.

El *Biolab3* es una laboratorio de ingeniería biomédica integrado en la Università degli Studi Roma Tre[1]. Este proyecto está relacionado con sus líneas de investigación relativas al análisis del movimiento humano. En esta línea el *Biolab3* colabora con médicos, fisioterapeutas y entrenadores para contribuir en técnicas de rehabilitación física o aumento de rendimiento deportivo [1] [2].

Una buena opción para monitorizar estas actividades son los sensores MEMS (*Microelectromechanical Systems)* de última generación, debido a su precisión, pequeño tamaño, bajo coste y consumo y gran versatilidad.

Para analizar los datos medidos por los sensores es necesario establecer una comunicación con algún dispositivo con capacidad de procesado, almacenamiento y visualización, como puede ser un PC o un *smartphone*. En algunas actividades es posible la transmisión por cable, pero en la mayoría es imprescindible la transmisión inalámbrica, mucho más cómoda y resistente en general.

El *BioLab3* ya había trabajado con tecnología inalámbrica para trasmitir datos, como Bluetooth clásico. En el caso de monitorizar el movimiento humano durante largos periodos de tiempo es necesario disponer de unidades portátiles de gran autonomía. El consumo de los sensores portátiles sumado al del dispositivo Bluetooth no alcanzaba la autonomía suficiente para poder trabajar con comodidad. De esta forma se planteó en el *BioLab3* como objetivo general diseñar un sistema

1. http://biolab.uniroma3.it/

inalámbrico capaz de adquirir, transmitir, procesar y mostrar datos cinemáticos, con el menor consumo energético posible en el dispositivo de sensores.

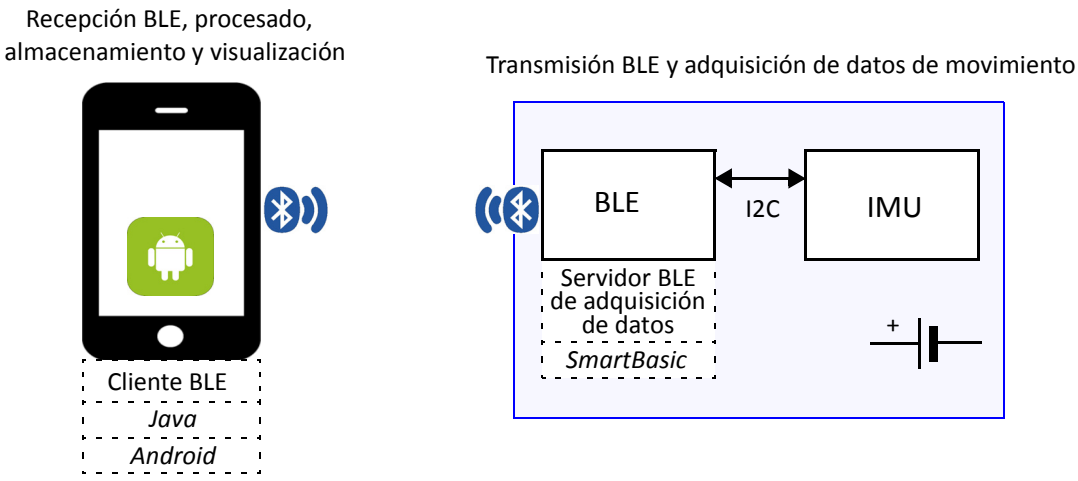
Existen distintas alternativas, pero para la transmisión inalámbrica se elige desde un principio la reciente tecnología Bluetooth Low Energy (BLE). Frente a otras posibilidades de bajo consumo, como ZigBee, BLE dispone de soporte tanto en PCs como en *smartphones* (IOs y Android) y su implantación está creciendo enormemente en los últimos años. Por ejemplo, las predicciones de la consultora ABI Research hablan de la venta de 60 millones de unidades BLE en 2019[2].

Ya existen en el mercado algunos dispositivos con una funcionalidad similar a la que buscamos con este proyecto, como es el SensorTag de Texas Instruments, dispositivo que mide datos inerciales, magnéticos y ambientales y los envía a un *smartphone* a través de BLE. Sin embargo, el *Biolab3* busca diseñar un dispositivo "*open-source*" sobre el que se puedan realizar todas las modificaciones que se deseen para el desarrollo de futuras aplicaciones.

1.2. Objetivos del sistema

En la Figura 1.1 se muestran los componentes hardware y software que van a formar el sistema.

FIGURA 1.1. Vista general del sistema.



A la derecha se muestra la parte de transmisión BLE y adquisición de datos de movimiento, compuesta por:

- **IMU** (*Inertial Measurement Unit*). Dispositivo electromecánico que mide la aceleración, velocidad angular y orientación combinando un acelerómetro, un giroscopio y un magnetómetro.
- **Módulo BLE**. Por un lado se encargará de recoger los datos inerciales medidos por la IMU vía I2C, y por otro lado de enviarlos por BLE al dispositivo de destino.

2. "BLE Tags and the Location of Things". Report AN-1755. Released 3Q 2014. Pages 17. Tables 6. One Issue Price US$2,500.00. www.abiresearch.com/market-research/product/1018920-ble-tags-and-the-location-of-things/

- **Servidor BLE de adquisición de datos**. El software que se tiene que desarrollar para interaccionar con el *smartphone* que se ejecuta en el módulo BLE, dónde, en nuestro caso, existe un microprocesador ARM que ejecuta un intérprete de *smartBASIC*, lenguaje de programación utilizado.
- **Batería**. Alimenta los circuitos.

A la izquierda se muestra la parte de recepción BLE, procesado, almacenamiento y visualización, compuesta por:

- **Dispositivo comercial con capacidad BLE**. Un *smartphone* con S.O. Android y versión 4.3 o superior.
- **Cliente BLE/Java/Android**. Aplicación que se ejecutará en el *smartphone* con capacidad de procesar, visualizary almacenar los datos recibidos y configurar el dispositivo de adquisición.

Las requisitos del sistema se concentran en la parte de adquisición:

- Bajo consumo. Una pila de botón debería ser suficiente para alimentar al dispositivo durante varias semanas de uso intensivo en laboratorio.
- Reducidas dimensiones. Aunque no es objetivo de este trabajo lograr una integración y encapsulado definitivo, hay que verificar que los componentes por separado van a permitir un formato de muy poco volumen.
- La comunicación entre BLE e IMU se realiza a través del bus de comunicaciones en serie I2C.

1.3. Estructura de la memoria

En el capítulo 2 se presenta un breve resumen de los aspectos más importantes de la Tecnología Bluetooth Low Energy, esencial para poder entender el trabajo que se ha realizado.

El capítulo 3 se centra en la parte hardware que se ha llevado a cabo. Se justifica la elección de un modelo concreto tanto de IMU como de módulo BLE, su montaje y conexionado.

El capítulo 4 se centra en el IMU escogido: el MPU-9250 de Invensense. Se ven los bloques funcionales que conforman el MPU-9250.

En el capítulo 5 se describe desde un punto de vista tanto hardware como software el módulo BLE seleccionado, el BL600 de Laird Technologies. Se detallarán los bloques funcionales que lo componen así como el entorno de desarrollo necesario para cargar aplicaciones en el BL600.

El capítulo 6 describe el software que se ha desarrollado en los dos protagonistas de la conexión Bluetooth LE, el Servidor BLE de adquisición de datos cargado en el BL600 y el Cliente BLE ejecutado en el *smartphone*.

El capítulo 7 presenta un estudio experimental de las limitaciones en el periodo de muestreo y del consumo energético del sistema.

Por último, en el capítulo 8 se analizan las principales conclusiones extradidas del desarrollo del proyecto, así como las posibles líneas futuras.

La memoria finaliza con una serie de anexos. El anexo A está dedicado a la carga de trabajo y desarrollo del proyecto en el tiempo. El anexo B profundiza en aspectos de la tecnología BLE. El anexo C detalla los registros utilizados del MPU-9250 y explica una serie de ejemplos de aplicación. Por último, el anexo D es un informe intermedio de progreso presentado en el *BioLab3*.

CAPÍTULO 2

Tecnología Bluetooth Low Energy

Resumen

La programación de la comunicación Bluetooth Low Energy entre cliente y servidor es uno de los aspectos claves de este proyecto. Por eso en este capítulo se revisan aspectos importantes de la tecnología BLE. Se explican sus rasgos principales, la pila del protocolo de comunicaciones que incluye cualquier dispositivo BLE y las distintas formas de comunicación entre varios dispositivos.

Posteriormente se describen los conceptos de protocolo y perfil. En concreto se explicará el papel que juegan el protocolo de atributos (ATT) y los perfiles genéricos de acceso (GAP) y de atributos (GATT) en el establecimiento de la comunicación y el intercambio de datos entre dispositivos BLE.

2.1. Introducción

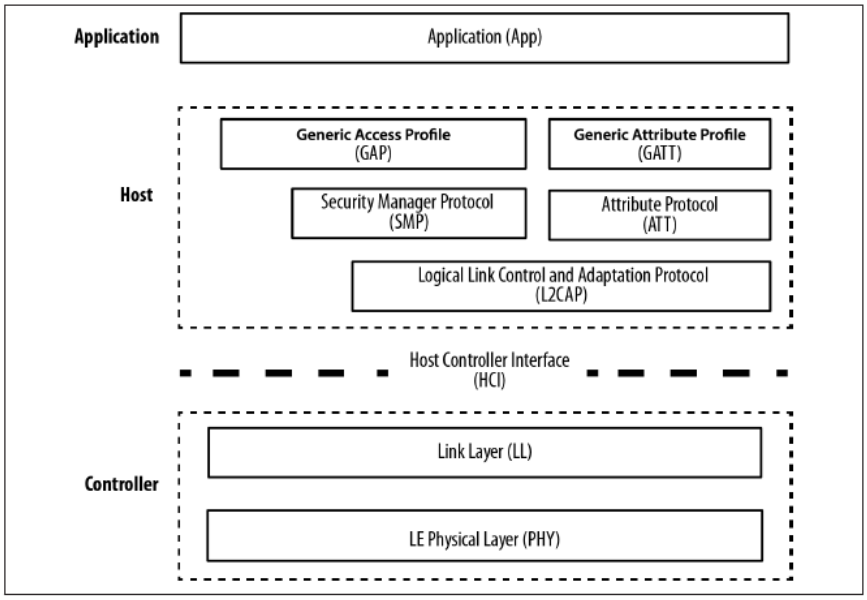
En 2010, se presenta Bluetooth Low Energy (BLE), también conocido como Bluetooth Smart, dentro de la versión 4.0 de la especificación *Bluetooth 4.0 Core Specification* [3][4]. Cubre tanto el estándar clásico Bluetooth como el nuevo BLE. Los dos estándares no son del todo compatibles, un dispositivo con una versión de Bluetooth anterior a la 4.0 no puede comunicarse con un dispositivo BLE.

Los principales componentes de un dispositivo BLE son los siguientes (ver Figura 2.1):

- *Application*: el software desarrollado por el usuario que opera de acuerdo al protocolo BLE.
- *Host*: las capas más altas de la pila del protocolo BLE. Suelen estar implementadas en forma de firmware y/o librerías en los dispositivos correspondientes.
- *Controller*: las capas más bajas de la pila del protocolo BLE, incluyendo la radio. Suele estar implementado por firmware y por dispositivos electrónicos.

Además, la especificación define un estándar de comunicación entre *Host* y *Controller*, el denominado *Host Controller Interface (HCI)*, para garantizar la interoperabilidad entre *Hosts* y *Controllers* fabricados por distintas compañías.

FIGURA 2.1. Pila del protocolo BLE ([4]).



En los apartados siguientes se describirán los perfiles GAP y GATT y el protocolo ATT, claves para el desarrollo de nuestra aplicación, dejando de lado las capas más bajas de BLE.

2.1.1 Características de BLE

BLE no se diseña como una alternativa al resto de tecnologías inalámbricas, y Bluetooth clásico, WiFi o ZigBee siguen teniendo su lugar. La Tabla 2.1 muestra las principales características de BLE frente a Bluetooth clásico.

TABLA 2.1. Comparativa entre Bluetooth clásico y BLE

	Classic Bluetooth	**BLE**
Data Throughput	1-3 Mbps	125 Kbps
Typ Operating Range	50 m	30 m[a]
Max Current	30 mA	15 mA
Sleep Current	-	1 μA
Modulation	GFSK	GFSK
Max Output Power	20 dBm	10 dBm

a. Hasta 100 m en modo *broadcast*.

Como se puede observar, BLE permite intercambiar paquetes de datos pequeños (de 20 bytes como máximo) a baja velocidad (125 Kbps). Su alcance típico ronda los 30 metros, pero la mayoría de aplicaciones solo precisan entre 2 y 5 metros. Aumentar el alcance requiere mayor potencia de emisión, con el consiguiente aumento de consumo y disminución de la vida de la batería.

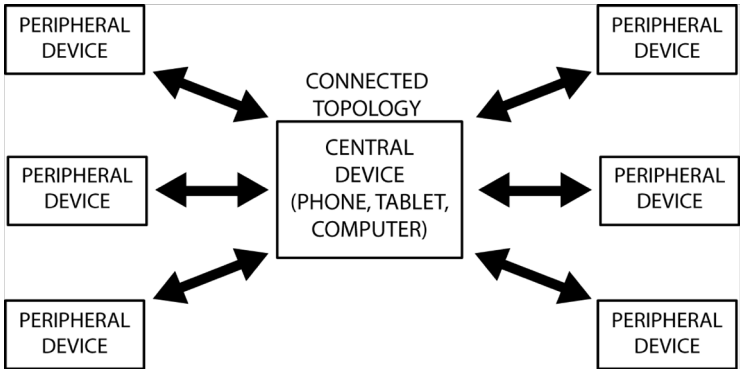
2.1.2 Comunicación BLE

Un dispositivo BLE puede comunicar de dos formas, mediante difusión (*broadcasting*) o mediante conexión (*connections*). Cada una posee limitaciones y ventajas:

- *Broadcasting and Observing*. Un dispositivo BLE que juega el papel de "*Broadcaster*" envía paquetes de datos periódicamente. Otros dispositivos BLE denominados "*Observers*" exploran periódicamente para recibir estos paquetes. Este modo de comunicación está pensado para enviar continuamente el mismo paquete de datos.
- *Connections*. La comunicación basada en conexión permite transmitir datos en ambas direcciones (ver Figura 2.2). El dispositivo *central* o *master* explora en busca de paquetes publicitarios de conexión que envían periódicamente los *peripherals* o *slaves*. Cuando el *master* lo considera apropiado inicia la conexión que deberá aceptar el *slave*. Una vez la conexión ha sido establecida, el *master* lleva la iniciativa y la temporización en el intercambio de datos.

La forma de comunicación utilizada entre el módulo BLE (*peripheral*) y el *smartphone* (*central*) es la conexión (*connection*).

FIGURA 2.2. Topología de la comunicación basada en conexión, sacado de [4].



Durante el proceso de establecimiento de una conexión *master* y *slave* negocian tres parámetros denominados parámetros de conexión de vital importancia durante la futura transmisión de datos:

- Intervalo de conexión: el tiempo entre dos eventos de conexión consecutivos. Puede variar desde 7,5 ms hasta 4 s. Con el menor valor obtenemos la máxima velocidad de datos pero también el consumo más elevado.
- *Slave latency*: el número de eventos de conexión que el *slave* puede dejar de escuchar al *master* sin poner en riesgo una desconexión.
- Tiempo de supervisión de la conexión: el tiempo máximo permitido entre la recepción de dos paquetes de datos consecutivos antes de que se pierda la conexión.

Se habla de negociación de parámetros porque durante el proceso de conexión el *peripheral* realiza una sugerencia de estos tres parámetros, pero es el *central* el que finalmente decidirá el valor de cada uno de ellos.

2.1.3 Protocolos y Perfiles

Los protocolos son los pilares básicos utilizados por los dispositivos BLE. Son las capas que definen formato de paquetes, encaminamiento, multiplexado, codificación y decodificación. Los protocolos permiten el envío eficiente de datos entre dispositivos BLE.

En cambio, los perfiles son "segmentos verticales" de funcionalidad que soportan modos de operación genéricos (GAP y GATT), requeridos por todos los dispositivos, o modos de operación particulares. En otras palabras, los perfiles definen como utilizar los protocolos para conseguir un objetivo genérico o particular.

Los perfiles genéricos aseguran la interoperabilidad entre dispositivos de distintos fabricantes. Han sido definidos por la especificación y se incluyen en cualquier dispositivo BLE. Los más importantes son los siguientes:

- *Generic Access Profile* (GAP). Define los roles, procedimientos y modos que permiten a los dispositivos retransmitir datos, descubrir a otros dispositivos, establecer y dirigir conexiones y negociar niveles de seguridad mediante el protocolo SMP. Este perfil es obligatorio para todos los dispositivos BLE. Es, esencialmente, la capa de control más alta.
- *Generic Attribute Profile* (GATT). Se encarga del intercambio de datos entre dispositivos BLE una vez ha sido establecida la conexión. El GATT define los modos y procedimientos que permiten a los dispositivos escribir y leer datos. Es la capa de datos más alta. El GATT se basa en el protocolo de atributos (Attribute Protocol, ATT).

El *Attribute Protocol* (ATT) es un protocolo de tipo petición/respuesta entre *client/server* basado en atributos. El *client* es el dispositivo que pide información al *server*, y el *server* es el que se la suministra. El protocolo es secuencial: hasta que el *server* no ha respondido a la petición del *client*, el *client* no debe iniciar una nueva petición.

Los servidores contienen datos siempre organizados en atributos. Un atributo está formado por:

- un **valor**, de 20 bytes como máximo.
- un identificador de 16 bits que utiliza el *client* para acceder al valor (*handle*).
- un identificador de 16 o 128 bits que especifica el tipo y la naturaleza del valor (UUID, *Universally unique identifier*).
- un conjunto de permisos relativos al valor.

El protocolo ATT define tres categorías principales de intercambio de datos entre *client* y *server*:

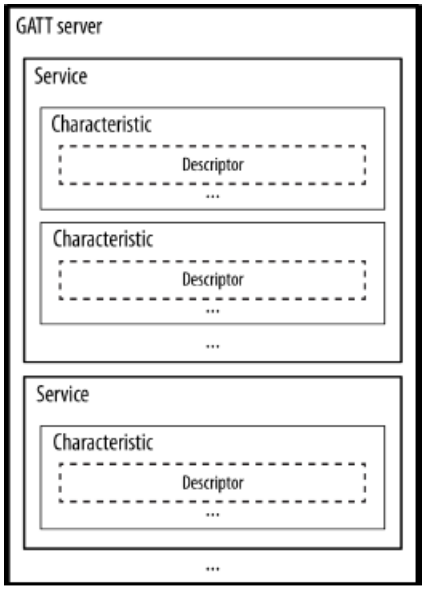
- operación de **lectura**, iniciada por el *client* para obtener el valor de uno o mas atributos. Existen varios tipos, pero el más usado es el esquema petición/respuesta (*Read Request/Response*). El *client* obtiene el valor del atributo a partir del *handle*.
- operación de **escritura**, iniciada por el *client* para fijar el valor de uno o más atributos. El *client* escribe en el valor de un atributo y espera respuesta del *server*.
- operación iniciada desde el *server* para enviar al *client* el valor de uno o mas atributos. El *server* puede comunicar cambios de valores sin necesidad de que el *client* tenga que malgastar ancho de banda y energía en encuestar al *server*. Existen dos tipos:

  - **indicación**, el *server* envía el valor y *handle* de un atributo y espera respuesta del *client* en forma de confirmación (*acknowledgement, ACK*).

  - **notificación**, equivalente a la indicación pero sin ACK.

El perfil GATT añade más funcionalidad y establece una **jerarquía** para organizar los atributos en lo que se conoce como ***GATT Server*** (ver Figura 2.3). Los atributos de un GATT *server* se agrupan en **servicios**, cada uno de los cuales puede contener una o más **características**. Las características, a su vez, pueden incluir cero o más **descriptores**. Todos los atributos pertenecientes a un GATT *server* deben están incluidos en una de estas tres categorías, servicio, característica o descriptor.

FIGURA 2.3. Jerarquía de atributos en un GATT Server ([4]).



En el ANEXO B, "Perfil GATT" se explica con mas detalle el perfil GATT y el papel de los servicios y las características en la programación de una aplicación BLE.

CAPÍTULO 3

Sistema BLE-IMU de transmisión y adquisición de datos

Resumen

En este capítulo se justifica la elección del chip MPU-9250 de InvenSense como IMU y del módulo BL600 de Laird Technologies como módulo BLE en concordancia con los requisitos planteados inicialmente.

Posteriormente, se describe la parte hardware desarrollada: montaje, conexionado y comunicación entre IMU, módulo BLE y PC.

3.1. Elección de los componentes

La elección del IMU y el módulo BLE se realiza en función a los requerimientos planteados al inicio del proyecto por el *BioLab3* (ver apartado 1.2.):

La elección del IMU se basó en la experiencia previa del *BioLab3*. Se decidió probar un dispositivo de última generación perteneciente a la familia MPU-9x5x, desarrollada por InvenSense [6]. Se trata de dispositivos de 9 ejes, que combinan un acelerómetro de 3 ejes, un giroscopio de 3 ejes, una brújula de 3 ejes, y un Procesador de Movimiento Digital (DMP). En el momento de la elección, InvenSense ofrecía dos nuevos dispositivos: el MPU-9150 y el MPU-9250, ver Tabla 3.1.

TABLA 3.1. MPU-9150 vs. MPU-9250.

	MPU-9150	**MPU-9250**
Full Scale Range	Accel: 16g	16g
	Gyro: 2000º/s	2000º/s
	Compass: 1200 uT	4800 uT
Operating Current (9 axis)	4,3 mA	3,7 mA
Package Dimmensions	4x4x1 mm	3x3x1 mm
Power Supply	2,4 - 3,6 V	2,4 - 3,6 V
Retail Price	$6,62	$6,62

Por el mismo precio se elige el MPU-9250 debido a su menor consumo, dimensiones más pequeñas y mayor rango en los datos magnéticos.

Debido a que el *Biolab3* no había trabajado con tecnología BLE, la elección del módulo BLE fue un poco más complicada. Se realizó una investigación de mercado para comparar los dispositivos más competitivos de los principales fabricantes. La Tabla 3.2 compara los módulos estudiados.

TABLA 3.2. Comparativa de dispositivos BLE.

	BR-LE4.0-SA Blueradios	**BLE113** BlueGiga	**OLP425** U-blox	**PAN 1721** Panasonic	**BL600** Laird Tech.
Power Consumption [mA]	27 TX[a] 19,6 RX[b]	18,2 TX 17,9 RX	18 TX 18 RX	14,3 TX 14,7 RX	10,5 TX 12 RX
Power Supply [V]	2 - 3,6	2 - 3,6	2 - 3,6	2 - 3,6	1,8 - 3,6
Dimensions [mm]	11,8x17,6x1,9	15,75x9,15x2,1	14,8x22,3x3,2	14,5x8,2x3	19x12,5x3
Volume [mm^3]	394,6	302,6	1056,1	356,7	731,25
I2C	OK	OK	OK	OK	OK

a. Transmisión
b. Recepción

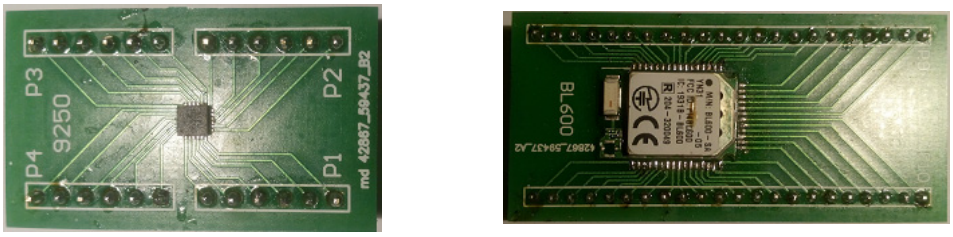
Todos ellos soportan comunicación I2C, mas ventajosa que SPI. Se escoge el BL600-SA de Laird Tech. por ser el más competitivo en relación consumo/dimensiones [7]. Otro punto, aparentemente positivo, que nos llevan a su elección es su lenguaje de programación, denominado *smartBASIC* (ver apartado 5.2.1) ya que se publicita como un lenguaje fácil de entender y completo.

3.2. Hardware y conexionado

El *BioLab3* realizó el diseño de una placa para el MPU y otra para el BL600 con *Altium Designer*, a la que se pudieran soldar los dos dispositivos. De esta forma se consigue prolongar con pistas eléctricas los pines de los componentes para facilitar el conexionado, ver Figura 3.1.

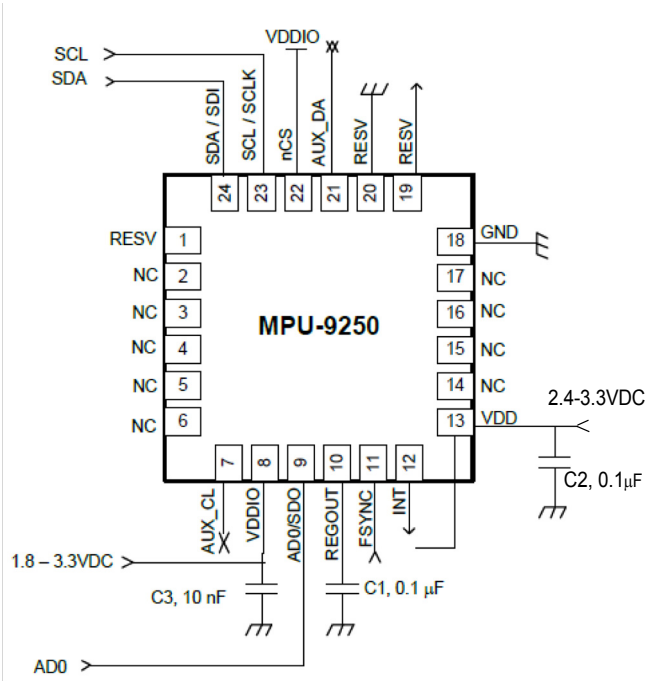
Se sueldan los extremos de las pistas a unas patillas para poder conectar los componentes a una placa eléctrica. Por otro lado, se sueldan los dos dispositivos a su respectiva placa mediante una pistola de aire caliente. En Zaragoza se repitió el proceso con otro juego de placas y chips, pero esta vez con la ayuda de una empresa que utilizó horno de soldadura.

FIGURA 3.1. MPU-9250 (izq.) y BL600 (der.).



La Figura 3.2 representa el conexionado de los pines del MPU-9250 que se ha llevado a cabo para el correcto funcionamiento del bus I2C.

FIGURA 3.2. Pines y conexión MPU-9250 para la operación I2C [8].



Pin Number	Pin Name	Pin description
1	RESV	Connect to VDD
7	AUX_CL	I2C Master serial clock
8	VDDIO	Digital I/O supply voltage
9	AD0	I2C Slave Address LSB,
10	REGOUT	Regulator filter
11	FSYNC	Connect to GND
12	INT	Interrupt digital output.
13	VDD	Power supply voltage
18	GND	Power supply ground
19	RESV	Not connected
20	RESV	Connect to GND
21	AUX_DA	I2C master serial data.
22	nCS	Connect to VDD
23	SCL	I2C serial clock
24	SDA	I2C serial data
2-6,14-17	NC	Not connected

La Tabla 3.3 muestra los pines utilizados para el correcto funcionamiento del BL600.

TABLA 3.3. Pines relevantes en el conexionado del BL600 [9].

Pin #	Name	Function	Alter. Fun.	Direction	Pull-up/down	Comment
1	GND					Connect to GND
9	SIO_7	DIO		IN	Pull-down	
10	VCC					Connect to VCC
12	SIO_8	DIO	I2C SDA	IN	Pulll-up	
13	SIO_9	DIO	I2C SCL	IN	Pull-up	
22	nRESET			IN		Connect to GND
40	nAutoRUN	DIO		IN	None	Select between the two BL600 operating modes

En el apartado 5.2.1 se explicará la funcionalidad de los pines 9 (SIO_7) y 40 (nAutoRUN).

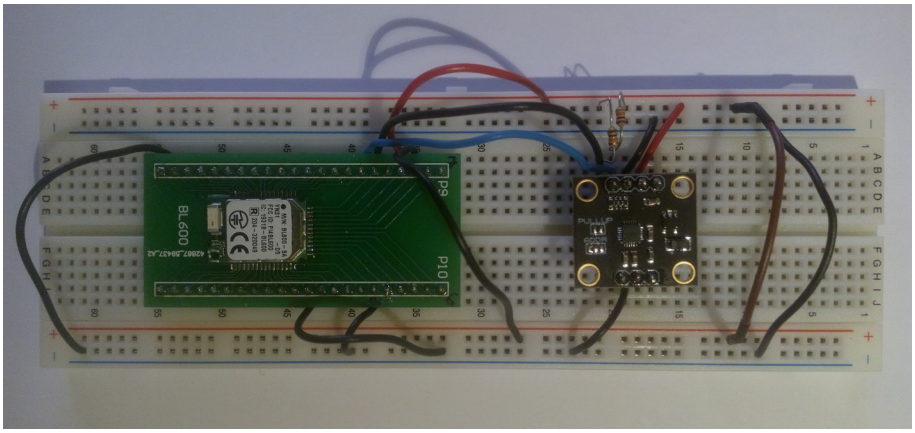
3.3. Comunicación I2C entre el MPU-9250/BL600.

Se selecciona el bus I2C, frente a SPI, como protocolo de comunicación entre el IMU y el módulo BLE por su menor cantidad de cables, facilidad de programación y por la posibilidad de

direccionar mas dispositivos dentro del IMU. Ambos chips soportan velocidades de datos I2C de 100 Kbps y 400 Kbps. Se ha trabajado siempre con la velocidad menor para minimizar el consumo.

El módulo BL600 solo puede ser configurado como *master I2C*, mientras que el MPU-9250 juega el rol de *slave I2C*. Ninguno de los chips tiene resistencias de *pull-up*, así que se añaden dos resistencias de 10K. La Figura 3.3 muestra el conjunto MPU-9250/BL600 montado en placa.

FIGURA 3.3. Conexionado MPU-9250[a]/BL600.



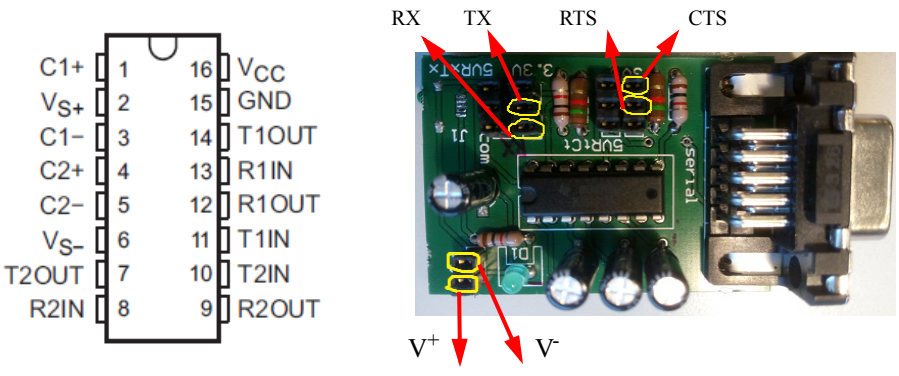
a. Además de la placa diseñada por el *BioLab3* se ha utilizado una placa desarrolla por Drotek (MPU-9250 Breakout Board) que facilita el conexionado de los pines.

3.4. Comunicación BL600 - PC

EL BL600 dispone de un entorno de desarrollo que permite la comunicación con PC a través del puerto serie (ver ANEXO D), con el fin de cargar el firmware en el módulo y depurar código.

Se ha utilizado el circuito integrado MAX232 [10], montado sobre una tarjeta diseñada por el *BioLab3*, (ver Figura 3.4) que adapta los niveles de tensión del BL600 al estándar RS232 de 5V. MAX232 y BL600 se conectan con los 4 pines estándar (TX, RX, CTS y RTS) y masa común.

FIGURA 3.4. Pines del MAX232 (izq.)[11] y tarjeta de montaje del *BioLab3* (der.).



CAPÍTULO 4

Descripción y programación del multichip MPU-9250

Resumen

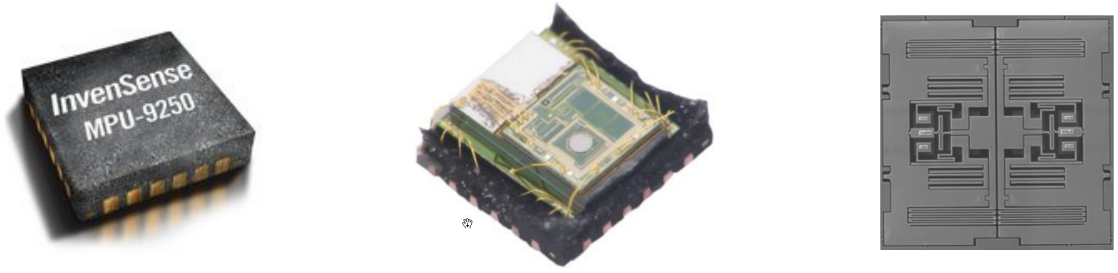
En este capítulo se presentan los aspectos más importantes del MPU-9250: funcionalidad, diagrama de bloques y descripción de los bloques de mayor trascendencia en el desarrollo del proyecto.

En el ANEXO C, "Mapa de registros utilizados y ejemplos de aplicación" se detallan los registros utilizados y una serie de ejemplos de aplicación.

4.1. Funcionalidad

El MPU-9250 de InvenSense (ver Figura 4.1) es un módulo multichip que consiste en dos *dies* integradas en un único empaquetamiento QFN (Quad-flat-no-leads) de 3x3x1 mm [8]. El primer *die* es un acelerómetro de 3 ejes y un giroscopio de 3 ejes. El otro *die* incluye el magnetómetro de 3 ejes AK8963 de *Asahi Kasei Microdevices Corporation* [12]. En definitiva, el MPU-9250 es un dispositivo de seguimiento de movimiento de 9 ejes que combina un acelerómetro, un giróscopo, un magnetómetro y un Procesador de Movimiento Digital (DMP).

FIGURA 4.1. Vista normal e ingeniería inversa[1] del multichip MPU-9250.



(a) Fotografía del multichip

(b) Ingeniería inversa: destape

(c) Ingeniería inversa: acelerómetro

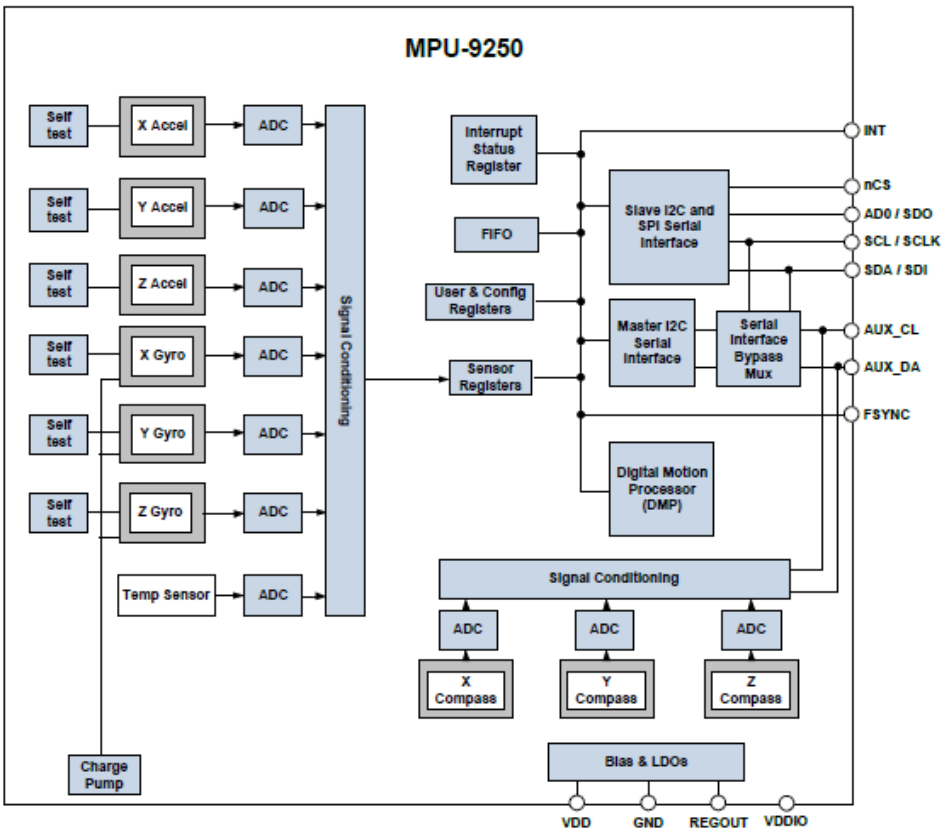
1. Reverse Costing Analysis report by System Plus Consulting. MEMS 9-Axis IMU, 150 pages, March 2014. Prices: full report: 2.990€, technology report: 1,990€.

4.1.1 Diagrama de bloques

El MPU-9250 se puede desglosar en los siguientes bloques funcionales (ver Figura 4.2):

- Giroscopio de 3 ejes con conversor analógico/digital (ADC) de 16 bits y acondicionamiento de señal.
- Acelerómetro de 3 ejes con ADC de 16 bits y acondicionamiento de señal.
- Magnetómetro de 3 ejes con ADC de 16 bits y acondicionamiento de señal.
- Procesador de Movimiento Digital (DMP).
- Interfaz primaria de comunicación I2C y SPI.
- Interfaz auxiliar I2C para sensores externos (*3rd party sensors*).
- Reloj.
- Registros de datos para los sensores.
- Memoria FIFO.
- Interruptores.
- Sensor digital de temperatura.
- *Self-test* para los tres sensores.

FIGURA 4.2. Diagrama de bloques del MPU-9250 ([8]).



A continuación se describen los bloques funcionales clave para el desarrollo del proyecto:

Giroscopio de 3 ejes con ADC de 16 bits y acondicionamiento de señal. Consiste en tres sensores microelectromecánicos independientes de velocidad angular, los cuales detectan la rotación alrededor de los ejes X, Y y Z. Cuando estos MEMS giran alrededor de alguno de los ejes, el Efecto Coriolis produce una vibración que es detectada por una condensador de *pickoff*. La señal resultante es amplificada, demodulada y filtrada para producir un voltaje proporcional a la velocidad angular. Pueden programarse cuatro rangos de medida: ±250, ±500, ±1000, o ±2000 grados por segundo.

Acelerómetro de 3 ejes con ADC de 16 bits y acondicionamiento de señal. Utiliza separadamente masas de prueba para cada eje de forma que un desplazamiento a lo largo de uno de los ejes induce un desplazamiento diferencial en la correspondiente masa de prueba, que es detectado por un sensor capacitivo. Cuando el dispositivo está situada en una superficie plana el acelerómetro medirá 0 g en el eje X y Y, y +1 g en el eje Z debido a la gravedad. Pueden programarse cuatro rangos de medida: ±2*g*, ±4*g*, ±8*g*, o ±16*g*.

Magnetómetro de 3 ejes con ADC de 16 bits y acondicionamiento de señal. Utiliza tecnología de sensores Hall de elevada susceptibilidad para detectar el magnetismo terrestre en los ejes X, Y y Z. Posee un rango de medida de ±4800 μT.

Interfaz primaria de comunicación I2C. El MPU-9250 siempre actúa como esclavo. La dirección I2C del MPU-9250 a la que se debe acceder desde el BL600 es 1101000**0**b[2] o 1101001**1**b. El bit menos significativo se determina con el pin 9 del chip (AD0). Este pin se ha fijado a 0 por lo que la dirección del MPU con la que trabajaremos es 1101000 (0x68[3]).

Interfaz auxiliar I2C. El MPU-9250 posee una línea auxiliar I2C para comunicarse con sensores externos o con el magnetómetro. Como puede observarse en la Figura 4.2, el magnetómetro es independiente del acelerómetro y del giróscopo, y está conectado directamente con esta línea auxiliar (AUX_CL y AUX_DA). El bus auxiliar dispone de dos modos operativos:

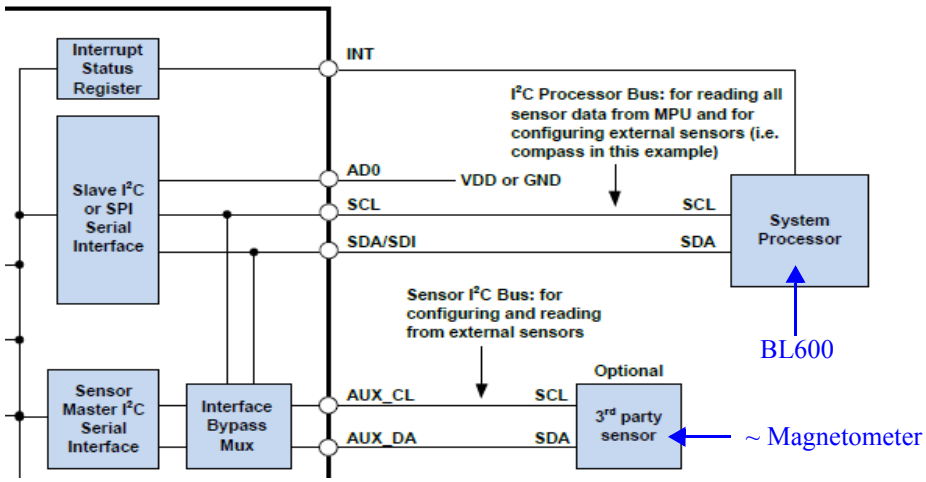
- *I2C Master Mode*: El MPU-9250 actúa como *master* en la comunicación con cualquier sensor externo conectado a la línea auxiliar.
- *Pass-Through Mode*: El MPU-9250 conecta a través de un multiplexor la línea primaria (SDA y SCL) con la línea auxiliar (AUX_CL y AUX_DA), permitiendo al BL600 comunicarse directamente con un sensor externo o con el magnetómetro.

Ambos modos son programables. Este último es el que se ha utilizado para comunicar directamente al BL600 con el magnetómetro. En este modo la dirección I2C del magnetómetro es 0x0C.

La Figura 4.3 muestra la solución I2C adoptada, en la que el BL600 se comunica con el MPU-9250 a través de la línea primaria y con un sensor externo a través de la línea auxiliar haciendo uso del *Pass-Through Mode*.

2. El sufijo “b” indica base de numeración binaria.
3. El prefijo “0x” indica base de numeración hexadecimal.

FIGURA 4.3. Solución I2C adoptada ([8]).



Aunque el magnetómetro no es exactamente un sensor externo, ya que está incluido en el bloque del MPU-9250, el modo de comunicación entre BL600 y magnetómetro es equivalente al presentado en la figura entre el *System Processor* y el 3rd *party sensor*.

En el *ANEXO C, "Mapa de registros utilizados y ejemplos de aplicación"* se puede consultar la siguiente información relacionada con el MPU-9250:

- Registros utilizados del MPU-9250 en la aplicación que se ejecuta en el Servidor BLE de adquisición de datos: *a*) valores de aceleración, velocidad angular y magnetismo; *b*) control de los rangos de medida; *c*) habilitación del modo *pass-through*; *d*) estado y control del magnetómetro y *e*) valores de calibración para ajustar las medidas de magnetismo (*Sensitivity Adjustment Values*).
- Dos ejemplos de aplicación que forman parte de la aplicación Servidor BLE de adquisición de datos. Uno de ellos usa un timer para controlar la puerta I2C y la lectura de datos. El siguiente muestra los pasos necesarios para activar el magnetómetro.

CAPÍTULO 5

Descripción y programación del Módulo BL600

Resumen

El presente capítulo expone tanto desde un punto de vista hardware como software los aspectos más importantes del módulo BL600.

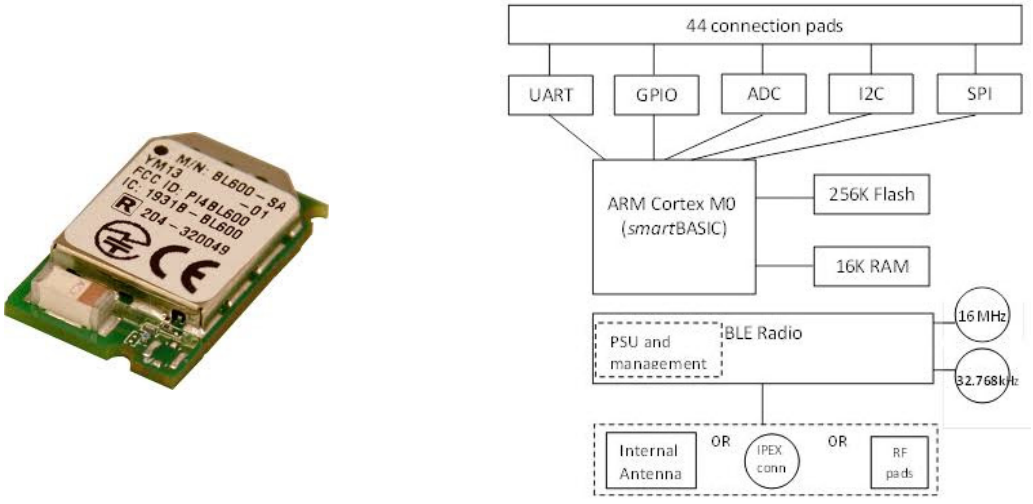
En primer lugar se describen los bloques funcionales del BL600 que intervienen en la comunicación BLE, en la ejecución de programas y en la interacción con periféricos.

Posteriormente se presenta el entorno de desarrollo utilizado para depurar código y cargar firmware en el BL600. Se describen brevemente las características de smartBASIC, el lenguaje de programación "nativo" de los módulos BL600. Finalmente se comenta la aplicación "UWTerminal", que permite interactuar con el BL600 a través del puerto serie.

5.1. Bloques funcionales del módulo BL600

La Figura 5.1 muestra el aspecto físico (izq.) y el diagrama funcional del BL600 (dcha.).

FIGURA 5.1. Fotografía y diagrama funcional del BL600.



A continuación se detallan los siguientes bloques del BL600: procesador, memorias, UART, I2C, y *timers*.

Procesador ARM Cortex M0. El Cortex M0 es el procesador de 16 bits más sencillo desarrollado por ARM. No soporta directamente aritmética en coma flotante ni maneja memoria virtual. Por ello ejecuta instrucciones con muy poco gasto energético y es ideal como acompañante en aplicaciones BLE. Su papel es ejecutar el intérprete de *smartBASIC*. El intérprete, a su vez, ejecutará la aplicación "Servidor BLE de adquisición de datos".

Memoria RAM. Memoria de 16KB donde se almacenan las variables de programa.

Memoria Flash. Memoria de 256KB que almacena el intérprete (residente) de *smartBASIC* y también el programa escrito en *smartBASIC* (aunque por razones de espacio se carga una versión binaria, previamente compilada en el PC de desarrollo).

UART (puerto serie). Dispone de 4 líneas (RX, TX, CTS, RTS). Se ha utilizado para cargar en el BL600 las aplicaciones *smartBASIC* a través del MAX232, ver 3.4.

I2C. El BL600 solo puede ser configurado como *master I2C* con la condición adicional de que solo haya un *master* en el bus. La velocidad de datos puede seleccionarse entre 100 Kbps y 400 Kbps. Se escoge 100 Kbps para minimizar consumo. Las resistencias de *pull-up* se añaden externamente.

***Timers* (temporizadores).** Por un lado, el BL600 dispone de 8 *timers* de 8-bits con resolución de 976 µs que son controlados únicamente por funciones de *smartBASIC*. Su uso más frecuente es la generación de eventos. Por otro lado, existe un noveno *timer* de 31 bits, denominado *Tick Timer*, que incrementa su valor cada 1 ms. Tiene una precisión de ±488 µs.

5.2. Entorno de desarrollo

5.2.1 Lenguaje SmartBASIC

El módulo BL600 soporta directamente *smartBASIC* [14], un lenguaje interpretado derivado de *BASIC* y desarrollado por Laird Tech. El intérprete[1] reside en memoria, y ejecuta programas codificados en un lenguaje intermedio que denominan "*bytecode*". El proceso de traducción de texto ASCII *smartBASIC* (extensión ".sb") a *bytecode* (extensión ".uwc") se realiza desde un PC Windows con un compilador cruzado libre. Laird Tech. no proporciona herramientas específicas para editar/verificar el programa fuente, por lo que hemos optado por el editor de texto gratuito *Notepad++*.

Para facilitar la programación de aplicaciones BLE, *smartBASIC* ofrece un buen soporte a la ejecución dirigida por eventos y también un conjunto amplio de rutinas incluidas en el intérprete residente (*core language built-in routines*).

Un programa *smartBASIC* contiene los siguientes bloques:

- Declaración e inicialización de variables.
- Definición de procedimientos y funciones.
- Declaración de **eventos** que llaman a las rutinas anteriores.

1. Laird Tech. lo denomina *run-time engine*.

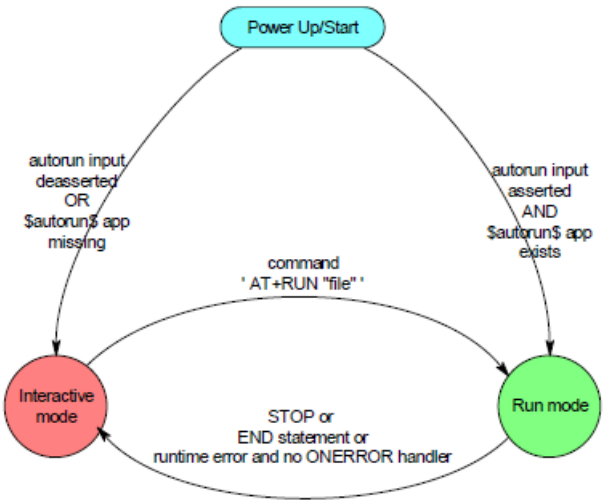
- Código principal, equivalente al *main()* en lenguaje C.

El programa termina con la sentencia WAITEVENT. Cuando el intérprete la alcanza, espera a que suceda alguno de los **eventos** declarados, activa las rutinas asociadas a esos eventos, vuelve al estado de espera, y así sucesivamente.

Cualquier plataforma *smartBASIC* dispone de dos modos operativos (ver Figura 5.2):

- **Modo interactivo**: permite interacción entre PC y BL600 a través del puerto serie. En el PC se ejecuta la aplicación *UWTerminal*, que utiliza el clásico estándar de comandos AT[2] para *modems* de comunicaciones. Desde este modo puede cargarse en la memoria flash una aplicación *smartBASIC*. El pin 40 (*nautoRUN*) debe estar a "1", y el pin 9 (SIO_7) a "0".
- **Modo *Run-time***: la conexión eléctrica de los dos pines anteriores debe ser la opuesta. En este modo, en cuanto el módulo BL600 recibe alimentación, se busca un fichero cuyo nombre empiece por *$autorun$* y si se encuentra se ejecuta. Por tanto, si se quiere ejecutar automáticamente el programa fuente *Mi_Ejecutable.sb* hay que nombrarlo como *$autorun$Mi_Ejecutable.sb*

FIGURA 5.2. Modos del módulo BL600 y transiciones ([14]).



5.2.2 UWTerminal [15]

El *UWTerminal* es una aplicación para PC Windows que sirve de entorno de desarrollo y de emulación de terminal. Permite invocar al compilador, cargar aplicaciones compiladas *smartBASIC* y depurar código en ejecución en el módulo BL600. Previamente es necesario conectar eléctricamente el puerto serie según se explicó en el apartado 3.4. (PC ↔ MAX232 ↔ BL600). En el ANEXO D, "Informe de Progreso. Sistema de Adquisición para el BioLab3" se pueden consultar más detalles acerca del *UWTerminal*.

2. Los comandos AT (*attention*) se desarrollan como interfaz de comunicación con un MODEM para su configuración y control.

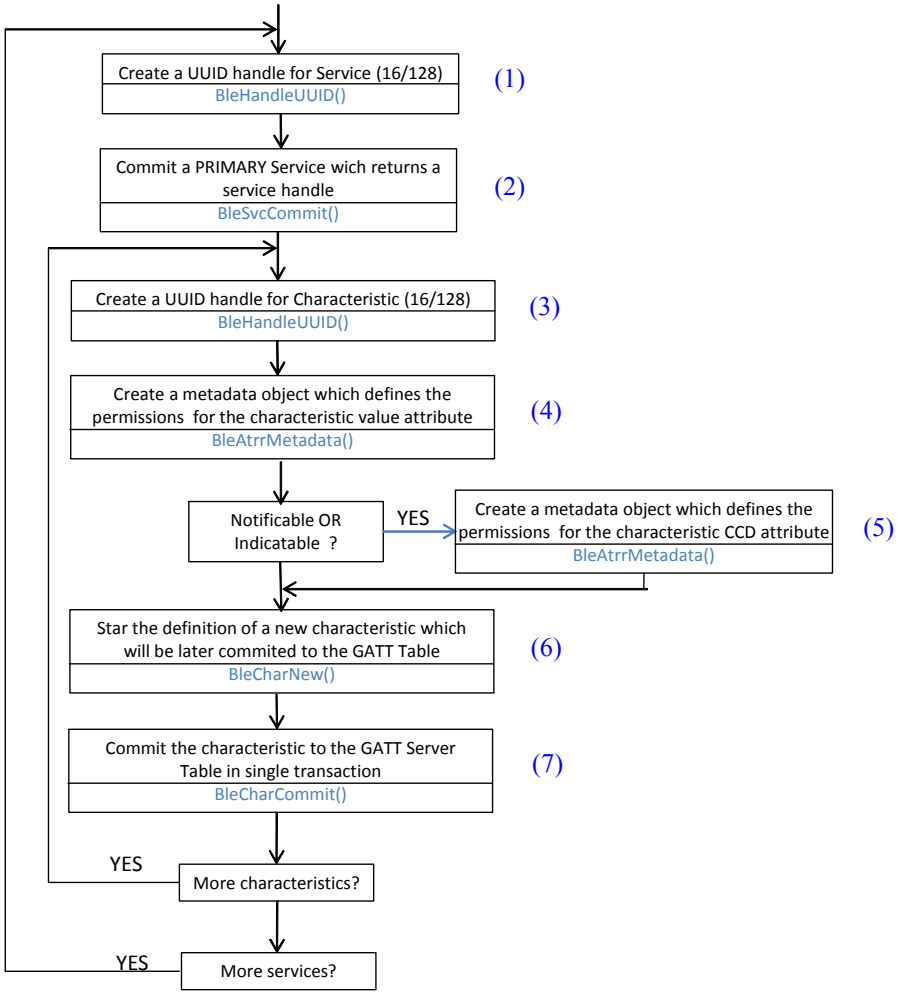
5.3. GATT Server en smartBASIC

A continuación se describen los aspectos más importantes que incluye *smartBASIC* en relación a la tecnología BLE como la creación del GATT Server a partir de la definición de servicios y características o los eventos más importantes generados durante la conexión BLE.

5.3.1 Creación del GATT Server

El siguiente esquema describe los pasos necesarios para crear un *GATT server*. Las funciones que aparecen son propias de *smartBASIC*, pero el esquema es el mismo con independencia del entorno de desarrollo y el módulo BLE sobre el que se programe.

FIGURA 5.3. Esquema de creación del *GATT server* ([14]).



En los dos primeros bloques del esquema se registra el servicio en la *GATT Table*. Todos los **servicios** y **características** de la aplicación son propios, es decir, no han sido desarrollados previamente por Bluetooth SIG. Por ello, el UUID que se pasa a la primera función es de 128 bits (1). Para generar UUIDs aleatoriamente se ha utilizado el siguiente enlace: `http://www.guidgenerator.com/online-guid-generator.aspx`. La segunda función ingresa el **servicio** en la *GATT*

Table (2). El *handle* que devuelva esta función, *BleSvcCommit()*, es el que se usará para interactuar con el servicio.

Para las **características** el proceso es parecido, pero antes de ingresar la característica en la *GATT Table* se definen los permisos del valor del atributo de la característica (4). Si se quiere trabajar con operaciones de **notificación** o **indicación**, además de indicarlo en las propiedades de la característica, se tiene que añadir a la característica que se quiere enviar un descriptor denominado *Client Characteristic Configuration Descriptor* (CCCD) (ver ANEXO B, "Perfil GATT", apartado B.7., "Descriptores"). El CCCD se inicializa y se le da los permisos correspondientes con (5).

El bloque (6) permite definir las propiedades de la característica, asignar a la característica el UUID que se ha creado con (3) y añadir el CCCD si previamente se ha definido con (5).Análogamente a los servicios, ingresamos la característica en la *GATT Table* (7).

5.3.2 Eventos

El intérprete de *smartBASIC* responde a una serie de eventos “interrumpiendo” a la sentencia WAITEVENT y ejecutando una rutina que podemos programar. Esta posibilidad va a ser muy útil para respetar el protocolo ATT, ya que nos permite imponer **un orden temporal estricto** en las acciones de nuestra aplicación. Los principales eventos utilizados en la aplicación son los siguientes:

- EVCHARVAL: este evento ocurre cuando el *client* escribe en una característica del *server*. En ese momento se llama a la función correspondiente, que recibe tres parámetros, unos de los cuales es el *handle* que se creó cuando se ingresó la característica. Los otros dos indican el *offset* y la longitud de datos del valor de la característica.
- EVCHARCCD: es lanzado cuando el *client* escribe en el CCCD de una característica para habilitar las **notificaciones** o **indicaciones** en esa característica. De nuevo, uno de los parámetros que recibe la función correspondiente es el *handle* de la característica.
- EVCHARHVC: ocurre cuando un valor enviado vía **indicación** recibe el *acknowledgement (ACK)* esperado por parte del *client*. Como siempre, uno de los parámetros que recibe la función correspondiente es el *handle* de la característica.
- EVBLEMSG: informa a la aplicación sobre eventos relevantes en la comunicación BLE. Por ejemplo, cuando se establece una conexión o se produce una desconexión inesperada.
- EVTMRn: es lanzado cuando el *timer* “n” expira.

Remarcar la importancia de los tres primeros eventos en la etapa de desarrollo y depuración de código. Gracias al evento y al *handle* de la característica, podemos saber en todo momento el estado de una característica en particular.

CAPÍTULO 6

Servidor BLE de adquisición de datos y Cliente BLE

Resumen

Este capítulo se centra en el software desarrollado en cada uno de los protagonistas de la comunicación BLE: "Servidor BLE de adquisición de datos", escrito en smartBASIC y cargado en el BL600; y "Cliente BLE", escrito en Java y realizado en el entorno de desarrollo Android Studio. Ambos parten de ejemplos desarrollados por Laird Tech. pero han tenido que ser modificados prácticamente en su totalidad.

Por último se presenta el entrelazado temporal de peticiones/respuestas entre smartphone y BL600 donde se aprecia la interacción temporal entre Servidor y Cliente.

6.1. Servidor BLE de adquisición de datos

El servidor está escrito en *smartBASIC* y se carga en el BL600 a través del puerto serie. Está compuesta de un programa principal y de una serie de librerías.

El programa principal es de tipo *$autorun$*, una vez cargado en la memoria flash se ejecuta automáticamente al alimentar el BL600. Importa las siguientes librerías:

- *custom.GATTserver.sblib*: incluye todas las funciones que inicializan y registran todos los servicios y características. Se trata de la única librería que se ha modificado, creando el *GATT Server* descrito en el apartado 5.3.1.
- *ble.sblib*: contiene definiciones para facilitar la lectura y escritura del código.
- *gap.service*: inicializa y gestione el perfil GAP.
- *device.information.service.sblib*: contiene la información básica del BL600 a la que el *client* podrá acceder cuando explora (dirección, nombre...).
- *connection.manager.sblib*: gestiona aspectos de la conexión entre *client* y *server*.

6.1.1 GATT Server: *custom.GATTserver.sblib*

El GATT Server es el componente principal del Servidor BLE de adquisición de datos, donde se crean e inicializan todos los servicios y características que almacenan los datos a los que accederá el Cliente BLE.

La *GATT Table* consta de cuatro servicios, tres corresponden a los sensores del MPU-9250 y el cuarto corresponde al control del periodo de muestreo. Los servicios de sensor tienen una caracte-

rística, que contiene el valor medido en cada periodo de muestreo. Para enviar estos valores de forma continua al Cliente BLE decidimos que la operación ATT más recomendable era la **indicación**. Cada vez que se lee un sensor se envía una indicación al Cliente BLE, evitando así la encuesta continua, con el consiguiente ahorro de consumo. La otra opción era la **notificación**, en la cual el *client* no envía confirmación. Hemos preferido la primera opción para garantizar la correcta recepción de todos los datos enviados.

El primer servicio creado es el servicio de la **aceleración** (ver Tabla 6.1).En la primera fila de la tabla se muestra el atributo de declaración de servicio primario de UUID = 0x2800 (ver ANEXO B). En el valor de este atributo está almacenado otro UUID, en este caso el correspondiente a la aceleración ("ced9d913 ...")[1]. Este UUID nos permitirá ingresar el servicio en el GATT *server.*

TABLA 6.1. Acceleration Service, cuatro atributos.

UUID	Nombre	Valor	Permisos	Descripción
0x2800	GATT_PRIMARY_SERVICE	Acceleration Service UUID: ced9d91366924a1287d56f2764762b2a	READ	Declaración del Servicio de Aceleración
0x2803	GATT_CHARACTERISTIC_UUID	Acceleration UUID: “ced8d9...” Acceleration handle: chAccMeas Properties: indicate (0x02)	READ	Declaración de la Característica de Aceleración
“ced8d9...”	ACCEL_MEASUREMENT_CHARACTERISTIC	X:Y:Z accelerations (6 bytes, integer)	None	Valores medidos de aceleración Unidades: g
0x2902	CLIENT_CHARAC_CNFG_UUID	00:00 (2bytes)	READ/WRITE	00:01 permite indicación 00:00 impide indicación

El siguiente atributo corresponde a la declaración de la característica. El valor de su atributo incluye dos campos además del UUID propio de la característica. Uno es el *handle* de la característica de la aceleración. Su utilidad ya fue descrita en 5.3.2. El otro campo tiene 8 bits que fijan las propiedades de la característica, en este caso la de **indicación**.

El tercer atributo almacena en su valor las medidas de aceleración. Este valor será el que se envíe vía **indicación** al *client*. Tiene una longitud de 6 bytes (2 por coordenada). Este atributo no requiere ningún permiso. El último atributo es un CCCD (*Client Characteristic Configuration Descriptor*), obligatorio en el caso de operaciones de indicación. Para habilitar la **indicación** el *client* tiene que escribir 0x01 en el valor del atributo.

El siguiente servicio creado es el relativo al **giroscopio**, ver Tabla 6.2.

TABLA 6.2. Gyroscope Service, cuatro atributos.

UUID	Nombre	Valor	Permisos	Descripción
0x2800	GATT_PRIMARY_SERVICE	Gyr. Service UUID: “ced7d913669...”	READ	Gyroscope Service Declaration
0x2803	GATT_CHARACTERISTIC_UUID	Properties: indicate(02) Gyr. handle: chGyrMeas Gyr. UUID: “ced6d9...”	READ	Gyroscope Characteristic Declaration
“ced6d9...”	GYR_MEASUREMENT_CHARACTERISTIC	X:Y:Z Coordinates (6 bytes, integer)	None	Gyroscope Measurements Values Units: º/s
0x2902	CLIENT_CHARAC_CNFG_UUID	00:00 (2bytes)	READ/WRITE	00:01 to enable indication 00:00 to disable

1. Generado aleatoriamente, ver 5.3.1.

El servicio del magnetismo contiene una característica más para almacenar los coeficientes de corrección de cada coordenada (*Sensitivity Adjustment Values*, ASA). Se trata de constantes calibrados en fábrica, de forma individual para cada MPU-9250 (ver Anexo B, apartado C.2.4). El valor de los coeficientes (8 bits cada uno) se envía únicamente al principio del programa mediante una comunicación de tipo *Read*. El cálculo del campo magnético ajustado se realiza en el Cliente BLE.

TABLA 6.3. Magnetism Service, seis atributos.

UUID	Nombre	Valor	Permisos	Descripción
0x2800	GATT_PRIMARY_SERVICE	Mag. Service UUID: "ced5d913669..."	READ	Magnetism Service Declaration
0x2803	GATT_CHARACTERISTIC_UUID	Properties: indicate(02) Mag. handle: chMagMeas Mag. UUID: "ced4d9..."	READ	Magnetism Characteristic Declaration
"ced4d9..."	MAG_MEASUREMENT_CHARACTERISTIC	X:Y:Z Coordinates (6 bytes)	None	Magnetism Measurements Values Units: μT
0x2902	CLIENT_CHARAC_CNFG_UUID	00:00 (2bytes)	READ/WRITE	00:01 to enable indication 00:00 to disable
0x2803	GATT_CHARACTERISTIC_UUID	Properties: read(40) Mag. handle: chMagASA Mag. UUID: "ced3d9..."	READ	Magnetism Characteristic Declaration
"ced3d9..."	MAG_ADJUSTMENT_CHARACTERISTIC	ASAX:ASAY:ASAZ Coordinates (3 bytes, integer)	READ	Magnetism Sensitivity Adjustment Values Units: μT

Por último, se ha definido el servicio del **periodo de muestreo**, Tabla 6.4. El valor del periodo de muestreo es un número natural de 2 bytes que el Cliente envía al Servidor a través de una comunicación de tipo *Write*. El valor por defecto es de 1000 ms.

TABLA 6.4. Sampling Period Service, tres atributos

UUID	Nombre	Valor	Permisos	Descripción
0x2800	GATT_PRIMARY_SERVICE	Period Service UUID: "ced2d913669..."	READ	Period Service Declaration
0x2803	GATT_CHARACTERISTIC_UUID	Properties: write(10) Period handle: chPeriodValue Period UUID: "ced1d9..."	READ	Period Characteristic Declaration
"ced1d9..."	PERIOD_MEASUREMENT_CHARACTERISTIC	Integer 2 bytes	WRITE	Period Value Units: ms

6.1.2 Esquema del Servidor BLE de Adquisición de Datos

El programa principal se encarga de inicializar el MPU-9250, abrir el puerto I2C, realizar la lectura de los sensores y llamar a las funciones que crean e inicializan los servicios y características. Al final se ejecuta la sentencia WAITEVENT que queda en espera de ser interrumpida por algún evento (ver apartado 5.2.1.). En la Figura 6.1 se muestra el esquema gráfico del Servidor.

En primer lugar se ejecuta la función *InitSensor()* que inicializa el MPU-9250. El *Timer0* recurrente comienza a contar, se abre el puerto I2C y se fija el modo *by-pass* para poder leer directamente del magnetómetro AK8963. Posteriormente se ejecuta la función *OnStartup()*. El programa principal acude a la librería *custom.GATTserver.sblib* para crear, registrar e inicializar todos los servicios y características.

Cuando el intérprete de *smartBASIC* alcanza la sentencia WAITEVENT el procesador queda a la espera de recibir eventos.

El primer evento lanzado al alimentar el BL600 es EVCHARCCD (0), cuando el *client* habilita las **indicaciones**. Es necesario un proceso de secuencialización, llevado a cabo por el *client* (ver 6.3.).

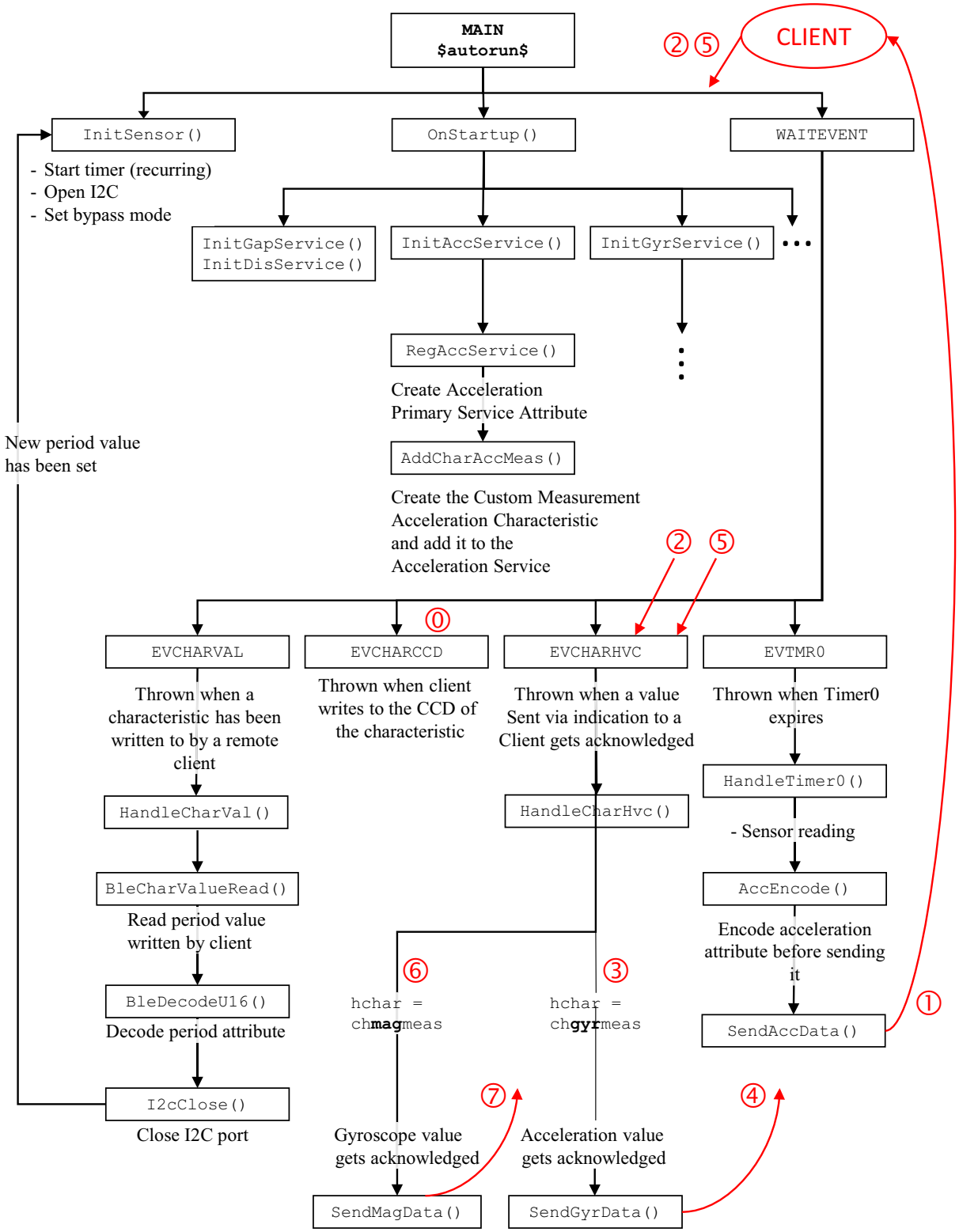
Mientras tanto, el *Timer0* va contando. Cuando expira se lanza el evento EVTMR0. En este punto se comprueba si se han habilitado las indicaciones para las tres características. Si es así, se leen los tres sensores. En caso contrario aparece el error correspondiente. Una vez leídos los sensores se codifica la aceleración y se envía su valor vía **indicación** (1). La codificación consiste en transformar el valor medido a formato *little endian*[1], que es el requerido por la especificación Bluetooth. Codificamos los valores como enteros de 16 bits en complemento a dos.

Una vez el *server* ha enviado al *client* la aceleración, este responde con una confirmación (2). Cuando el *server* recibe esta confirmación se lanza el evento EVCHARHVC. A continuación se llama a la función *HandlerCharHvc()*, que recibe como parámetro el *handle* de la característica que se ha enviado. Se comprueba si este *handle* coincide con el de la aceleración (3). En caso correcto, se envía el valor de la velocidad angular vía indicación (4). Se repite el mismo proceso para enviar el magnetismo (5, 6, 7). De este modo, conseguimos una secuencialización en el envío de datos con indicaciones.

Cuando el usuario introduce el periodo de muestreo desde el *smartphone*, es decir, cuando el *client* escribe sobre la característica del periodo de muestro, es lanzado el evento EVCHARVAL. Se lee su valor y se cierra el puerto I2C. Se vuelve a llamar a la función *InitSensor()*, donde el *Timer0* se reprograma con el nuevo valor y se repite todo el proceso de comunicación por indicación con el nuevo periodo. No es necesario volver a habilitar las **indicaciones**.

1. *Little endian* es un formato de almacenamiento de un objeto *multibyte* en memoria que coloca el byte de menos peso (LSB) en la dirección de memoria menor.

FIGURA 6.1. Esquema gráfico del Servidor BLE de Adquisición de Datos.



6.2. Cliente BLE

6.2.1 Entorno de desarrollo

Como entorno de desarrollo se ha escogido Android Studio, un entorno integrado (IDE) lanzado por Google y basado en IntelliJ IDEA [16]. Google escogió como lenguaje de desarrollo oficial Java [17]. También es necesario para empezar el desarrollo en Android una serie de librerías y herramientas, el SDK (*Software Development Kit*) de Android; y la máquina virtual de Java, contenida en el JDK (*Java Developmet Kit)*.

En cuanto a Bluetooth Low Energy, Google introdujo soporte software por primera vez en la versión Android 4.3 (API 18) [18]. El dispositivo Android solo puede jugar el rol de *central* y de *client* en la comunicación BLE. Se ha trabajado con un Samsung Galaxy S4 mini y con un Samsung Galaxy S3.

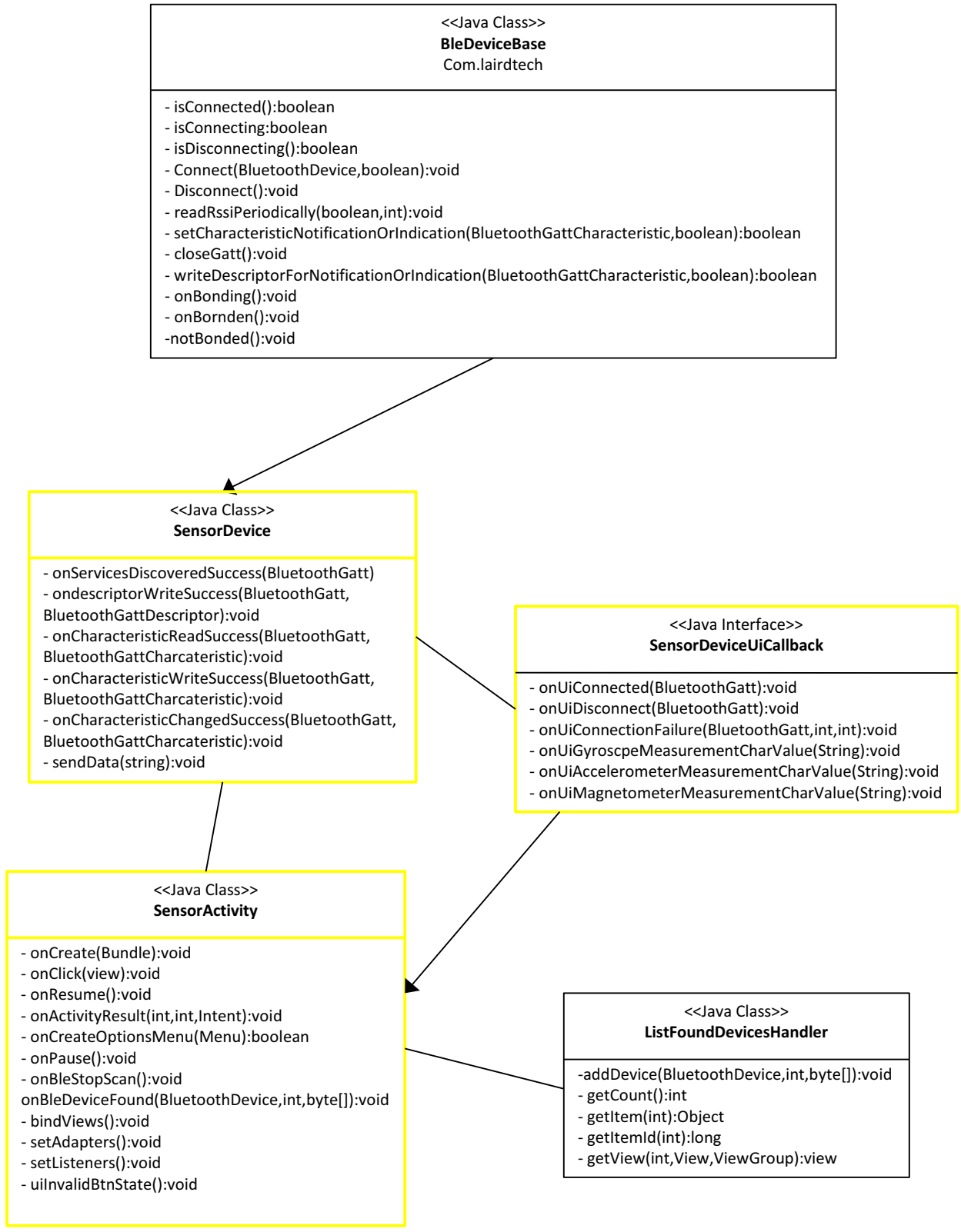
6.2.2 Cliente BLE

La aplicación Android que se ha desarrollado se ha realizado en base al ejemplo *HeartRateBleDemo* desarrollado por Laird Technologies.

La aplicación Cliente BLE juega el papel de *central* o *master* durante el establecimiento de la conexión. Una vez que ha sido establecida, el Cliente BLE juega el rol de GATT *client* y será el que lleve la iniciativa durante el intercambio de datos.

La siguiente figura muestra un diagrama de las clases más importantes de la aplicación. Las líneas que las unen representan una relación mutua. Dentro de cada clase se han añadido los métodos más importantes. Las clases en amarillo son las que han sido modificadas en gran medida.

FIGURA 6.2. Diagrama de clases



Las principales clases y librerías que se han utilizado y modificado son las siguientes:

- ***SensorActivity.*** Esta clase extiende a la clase *Activity.* Las actividades interactúan con el usuario, se encargan de crear ventanas o interfaces donde podamos situar nuestra UI (*User Interface*). Más concretamente, *SensorActivity* es la actividad principal (*main activity)* del proyecto. Es la encargada de gestionar la ventana que aparece al abrir la aplicación (ver Figura 6.3).

FIGURA 6.3. Ventana principal de la aplicación Cliente BLE

Los principales elementos son dos botones (*Scan* y *Send*); un *EditText* con formato numérico, donde se introduce el periodo de muestreo; y una serie de *TextView* para mostrar las medidas.

La clase *SensorActivity* se encarga de enlazar estos recursos de la interfaz de usuario con las variables de nuestro programa. También es la responsable de mostrar en pantalla el valor de los sensores y el nombre y dirección del dispositivo al que se ha conectado, así como de enviar el periodo de muestreo y explorar la existencia de dispositivos BLE.

- ***BleDeviceBase.*** Clase definidas por Laird Tech. e incluida en el paquete *com.lairdtech.bt.ble.* Contiene todos los métodos necesarios para iniciar una conexión BLE con un dispositivo remoto y para comunicarse con él una vez establecida la conexión.
- ***SensorDevice.*** Esta clase es sobre la que más se ha trabajado y la más importante en la comunicación con el módulo BLE. Contiene la definición e inicialización de todos los servicios y características de nuestro proyecto, así como el UUID de cada uno de ellos.

  Se encarga de, *i)* habilitar las indicaciones en el *Server* en las tres características cinemáticas de forma secuencial, escribiendo un 1 en el valor del atributo del CCC Descriptor, *ii)* leer la característica que contiene los valores de calibración del magnetismo, *iii)* decodificar los valores recibidos vía indicación y realizar los cálculos necesarios, y *iv)* escribir en la característica del periodo muestreo el valor introducido por el usuario.

- ***SensorDeviceUiCallback.*** Gestiona todos los *callbacks*[1]de la aplicación. Informa del éxito o fracaso de la escritura o lectura de una característica o descriptor, indica si alguna operación GATT ha fallado o si el GATT *client* se ha desconectado del *server*.
- ***ListFoundDevicesHandler.*** Responsable de almacenar y mostrar todos los dispositivos BLE que se han encontrado durante el proceso de exploración. Muestra el nombre del dispositivo, su dirección y la "fuerza" de la señal (RSSI).

Una interesante mejora que se ha realizado y que gestiona ***SensorActivity*** consiste en crear y escribir un fichero de texto que almacena los datos inerciales y magnéticos en la memoria externa del teléfono (tarjeta microSD). Puede cambiarse el directorio de almacenamiento para usar la memoria interna del teléfono. El nombre del fichero es la fecha en el momento de abrir la aplicación (*yyyy_MM_dd_HH_mm_ss.txt*). En cada línea se escribe el tiempo actual en el momento de recibir la aceleración, en formato UTC (precisión de milisegundos) y los 9 valores medidos.

6.3. Entrelazado temporal de peticiones y respuestas

La Figura 6.4 ilustra la secuencia temporal de peticiones-respuestas entre el *Cliente BLE* y el *Servidor BLE de adquisición de datos.* Al ser alimentado, el módulo BLE comienza a enviar paquetes publicitarios de conexión (0). Estos paquetes contienen AD *Types* (*Advertising Data Types*). El BL600 envía el nombre del dispositivo, su dirección y la fuerza de la señal (RSSI).

Por otro lado, se debe pulsar el botón de *Scan* de la aplicación Android para que el *central* comience a explorar los paquetes publicitarios (1). Los dispositivos que explora el Cliente BLE se añaden a una lista que muestra el *smartphone* gestionada por *ListFoundDevicesHandler*. Para iniciar una conexión con alguno de ellos se pulsa sobre su nombre. En este momento el *smartphone* envía una petición de conexión privada al módulo BLE (2). Éste acepta la conexión y detiene el envío paquetes publicitarios (3). Durante el proceso de conexión se produce la negociación de los parámetros de conexión (ver 2.1.2). Desde el Servidor BLE se puede sugerir el valor de estos parámetros, pero el Cliente BLE decidirá finalmente su valor.

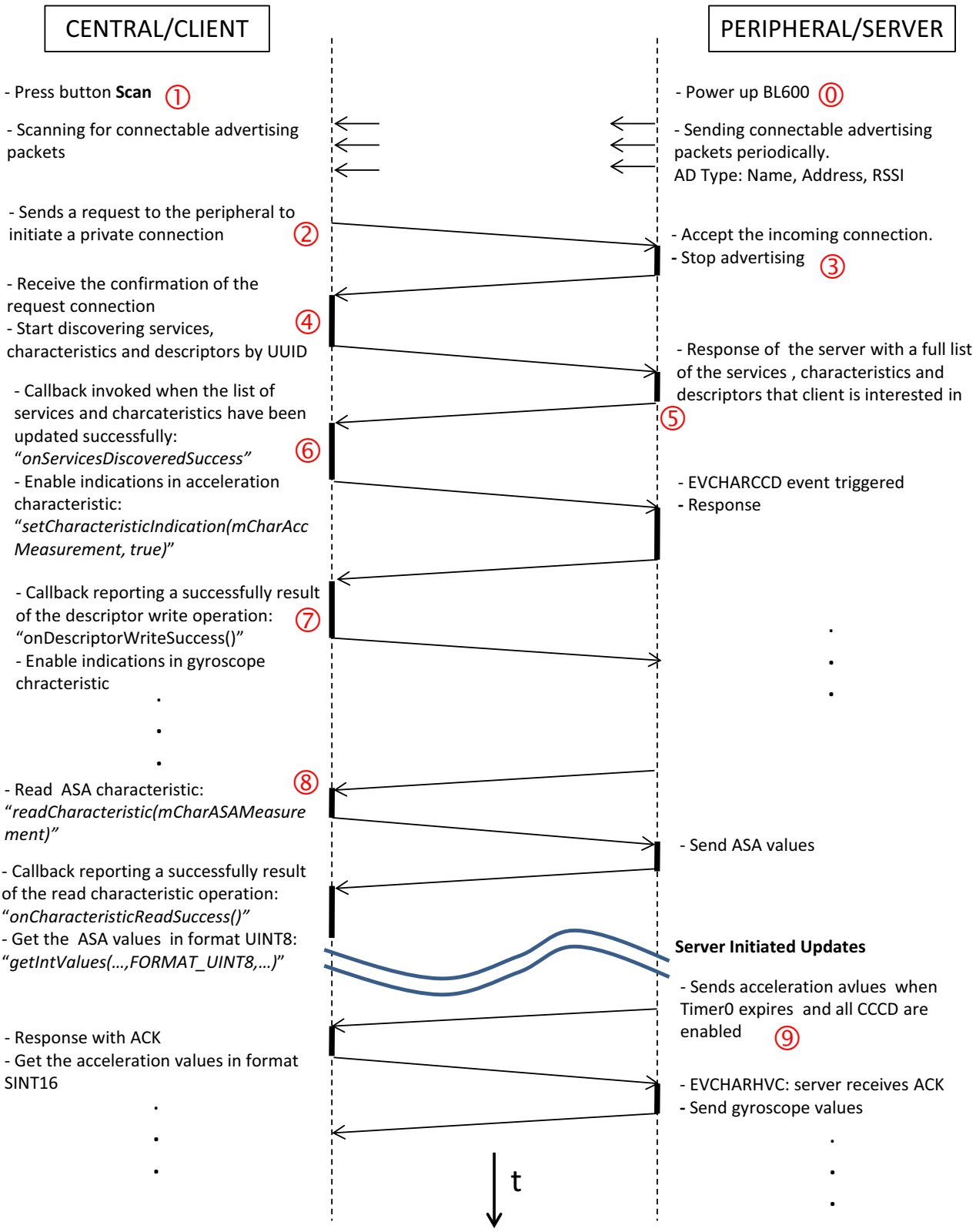
Tras establecer la conexión, el *client* comienza un intercambio de paquetes (4) para determinar la cantidad, situación y naturaleza de todos los atributos de interés contenidos en el GATT *server*. Este proceso en Android se realiza con la función: `discoverServices()`. El *server* responde con una lista completa de todos los atributos que le interesan al *client* (5).

Una vez la operación ha finalizado con éxito, se lanza el *callback* `onServicesDiscoveredSuccess(BluetoothGatt gatt)`. En este punto se pueden recuperar todos los servicios y características con las funciones `getService()` y `getCharacteristic()`. De esta manera, ya se puede trabajar con los servicios y características en la app Android.

Es dentro de este *callback* cuando el *client* habilita las **indicaciones** en el CCCD de la característica de la aceleración del *server* (6). Cuando la respuesta del *server* llega al *client* se lanza un nuevo *callback:* `onDescriptorWriteSuccess(BluetoothGatt gatt,BluetoothGattDescriptor descriptor)`, en el cual se habilitan las **indicaciones** en la característica de la velocidad angular (7).

1. Los *callbacks* (literalmente, devolución de llamadas) informan de la finalización de una tarea o acción y si ésta ha finalizado con éxito.

FIGURA 6.4. Entrelazado temporal de peticiones/respuestas.



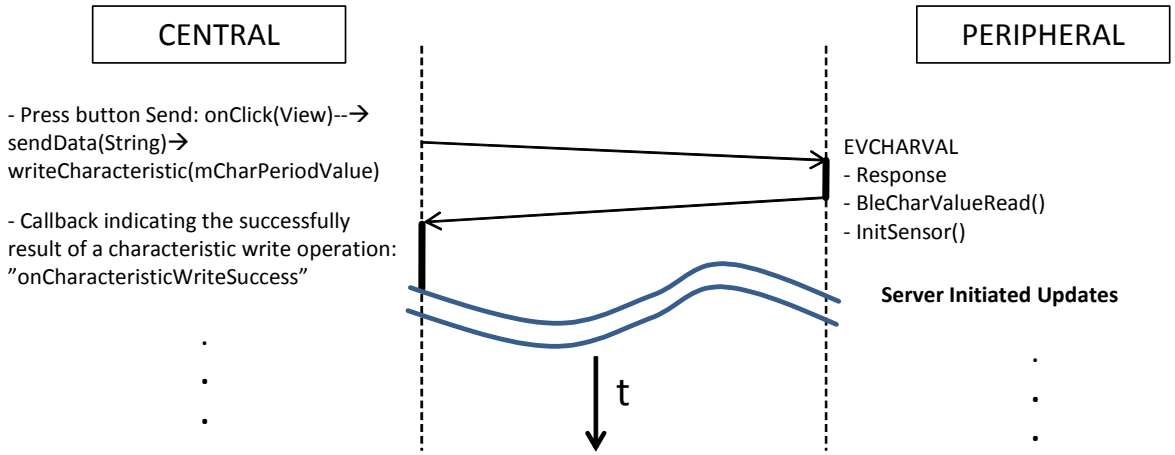
Se repite el mismo proceso para el magnetismo. Cuando se lanza el *callback* de la característica del magnetismo se realiza la lectura de la característica de los coeficientes de calibración del magnetismo (8).

Se consigue de esta forma la secuencialización necesaria en el protocolo ATT. Un error que se cometió inicialmente fue el no respetarla: antes de que el *client* recibiera el primer *callback* ya se había enviado la siguiente petición de habilitación de las **indicaciones**. El *client* nunca puede estar esperando más de un *callback* a la vez.

Las líneas curvas indican un cambio en el sentido de la comunicación. El *client* deja de llevar la iniciativa y es el *server* el que comienza a enviar los datos vía **indicación** al *client* (9). Hasta que éste no recibe el ACK del *client* no envía la siguiente indicación.

El proceso de indicación continúa hasta que se produce una desconexión, hay un fallo en la comunicación, o se envía un nuevo valor del periodo de muestreo desde el *smartphone*. La Figura 6.5 ilustra el entrelazado temporal en el envío del periodo de muestreo.

FIGURA 6.5. Entrelazado temporal al enviar el periodo de muestreo



Al introducir el valor del periodo de muestreo y pulsar el botón *Send* se lanza el método `onClick(View)`, que lee el valor, lo codifica con formato *FORMAT_UINT16* y lo envía al *server* con el método `writeCharacteristic(mCharPeriodValue)`. Cuando el *server* recibe el valor se llevan a cabo las operaciones descritas en el apartado 6.1.2 y responde al *client*. En el momento que la respuesta en forma de confirmación del *server* llega al *client* se lanza el *callback* `onCharacteristicWriteSuccess(BluetoothGatt gatt, BluetoothGattCharacteristic ch)`.

De nuevo cambia el sentido de la comunicación y el *server* comienza a enviar los valores de los sensores vía **indicación** con el nuevo periodo de muestreo.

CAPÍTULO 7 *Medidas experimentales*

En este capítulo se fijan los límites alcanzables en el periodo de muestreo, obtenidos experimentalmente variando el intervalo de conexión (IC) y comprobando los datos almacenados en los ficheros de datos.

Para los periodos alcanzables se ha realizado un estudio del consumo energético del sistema. Se ha medido la potencia media en el MPU-9250, en el BL600 y en el sistema global (ambos chips más las líneas I2C) con un watímetro de precisión. Con los resultados obtenidos se ha realizado una estimación de la vida útil de dos pilas de botón de 3V.

7.1. Límites en el periodo de muestreo

El análisis de los ficheros de texto permite determinar el periodo de muestreo mínimo, que está muy relacionado con el intervalo de conexión (IC, ver 2.1.2). Sugiriendo un IC de unos 100 ms se observa que el periodo de muestreo mínimo es de unos 600 ms. Si forzamos al sistema a trabajar a frecuencias mayores vemos que no responde y se mantiene en esos 600 ms. En cambio, sugiriendo un IC de 10 ms el sistema consigue muestrear a unos 300 ms. Recordar que durante un periodo de muestreo el *server* tiene que enviar tres características vía indicación y el *client* recibirlos, responder con un ACK, procesarlos, mostrarlos por pantalla y almacenarlos en el fichero.

Al realizar una investigación más profunda respecto al IC se ha llegado a las siguientes conclusiones [19][20]:

- Aunque el protocolo BLE permite teóricamente un intervalo de conexión mínimo de 7,5 ms, cada *smartphone* tiene un límite algo mayor en función de su *stack* BLE y su sistema operativo. Más concretamente, el Samsung Galaxy S3 utilizado en el laboratorio de Zaragoza posee un IC mínimo de unos 50 ms.
- Como ya se discutió, en una indicación el *server* envía el valor correspondiente y el *client* responde con un ACK en forma de confirmación. Se ha concluido que valor y ACK viajan en ICs distintos. Es decir, en cada indicación transcurren dos ICs. En el caso de tres características:

$$T_{muestreo} \geq 6 \cdot IC$$

Por tanto el mínimo periodo de muestreo que se puede alcanzar comunicando mediante indicaciones es de unos 300 ms.

Esta conclusión responde a la observación experimental. Con un IC de unos 100 ms no se puede muestrear a menos de 600 ms. En el otro caso, el “Servidor BLE de adquisición de datos” le suge-

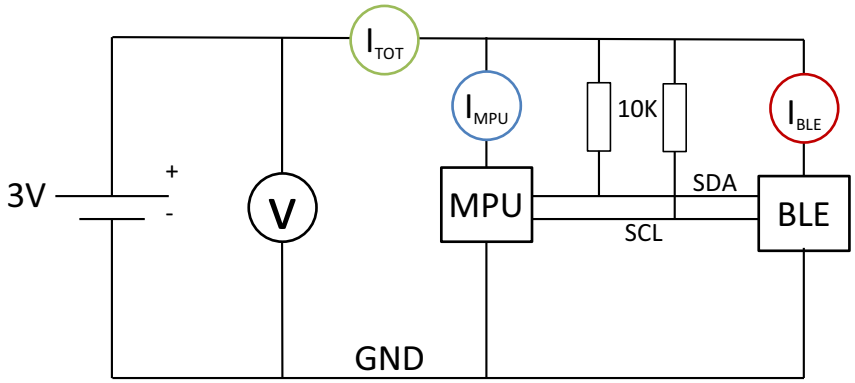
ría al *central* un IC de unos 10 ms. Pero es el *central* el que finalmente fija los parámetros de conexión. De esta forma elegía el IC más cercano a los 10 ms, es decir, su límite máximo de 50 ms. Con un IC de 50 ms alcanzamos un periodo de muestreo de 300 ms.

7.2. *Análisis del consumo del sistema*

Se ha medido la potencia media consumida por el MPU-9250, por el BL600 y por el conjunto, utilizando el watímetro de precisión WT210 de *Yokogawa* [21].

La Figura 7.1 representa la colocación las sondas de voltaje e intensidad para medir la potencia media de los componentes y del sistema.

FIGURA 7.1. Colocación de sondas V e I para cada componente.



El terminal *WTViewerFree* del watímetro ha permitido almacenar y visualizar en el PC los datos de consumo a través del puerto serie RS232.

Se han realizado medidas de la potencia media en función de la frecuencia de muestreo, variándola desde el *smartphone* (20 medidas para cada frecuencia). Se ha realizado un experimento con un IC de 100 ms (ver Figura 7.2) y un otro con el IC mínimo de 50 ms (ver Figura 7.3).

FIGURA 7.2. Gráfica de potencia media vs. frecuencia de muestreo para un IC de 100 ms. La frecuencia de 0 Hz mide la actividad del BL600 enviando paquetes publicitarios.

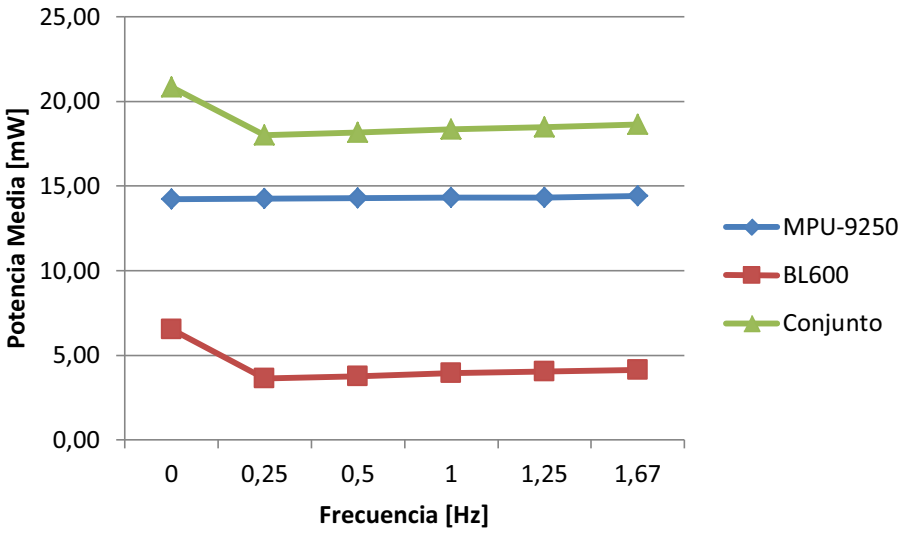
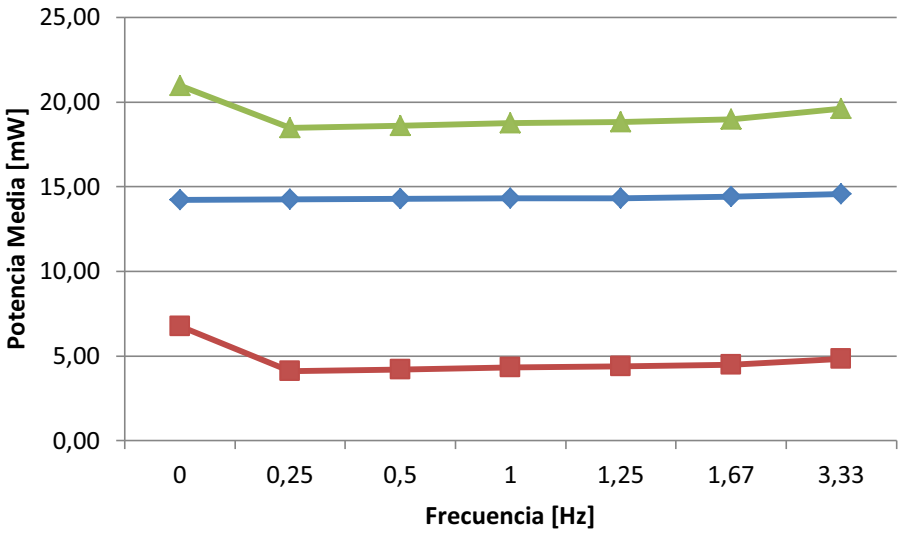


FIGURA 7.3. Gráfica de potencia media vs. frecuencia de muestreo para un IC de 50 ms. La frecuencia de 0 Hz mide la actividad del BL600 enviando paquetes publicitarios.



De los experimentos realizados se concluye:

- Aproximadamente 2/3 de la potencia total media se deben al MPU-9250. Las medidas se han realizado con los tres sensores funcionando.
- La potencia consumida es prácticamente constante con la frecuencia de muestreo debido a que se ha trabajado con frecuencias muy bajas. Se realizó un primer experimento llegando a valo-

res de 25 Hz donde si se podía apreciar una relación lineal entre ambos parámetros pero se desestimaron las medidas porque el sistema no podía alcanzar tanta velocidad.

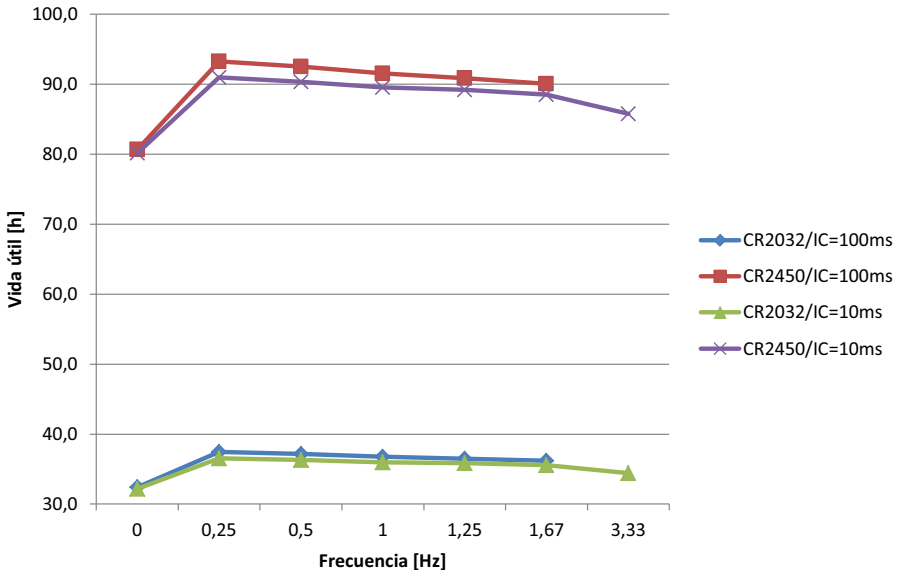
- El IC no afecta a la potencia consumida por el MPU-9250. Sí que afecta ligeramente al BL600, apreciándose una diferencia de algo menos de 1 mW entre las medidas tomadas con un ICs de 100 ms y de 50 ms.
- Se ha observado que existe una diferencia de 1 mW entre la potencia total y la suma de las potencias consumidas por el MPU-9250 y el BL600. Esta diferencia se debe a las resistencias de *pull-up* del bus I2C, ya que en la medida de la potencia de los componentes por separado no se ha considerado la corriente que circula por las líneas SDA y SCL (ver Figura 7.1).
- Es interesante analizar porqué la potencia consumida por el BL600 cuando se está anunciando (en la Figura 7.2 y la Figura 7.3 el consumo cuando se está anunciando corresponde a la medida a 0 Hz) es mayor a la potencia consumida transmitiendo. Al parámetro que debemos acudir es al intervalo publicitario (*advertising interval*), es decir, el intervalo de tiempo entre dos eventos publicitarios. A este parámetro se le puede dar un valor entre 20 ms y 10240 ms. El BL600 a 20 ms consume unos 800uA, con un intervalo de 10240 ms consume unos 3uA. Se ha trabajado a 25 ms. Este parámetro podría ser aumentado para reducir notablemente el consumo. Tampoco es conveniente un valor demasiado alto, ya que se tardaría demasiado tiempo en establecer la conexión.

Por último, se ha realizado una estimación de la vida útil (ver Figura 7.4) en función de la frecuencia de muestreo y del IC de la pila de botón CR2032 (3 gramos, 225 mAh) y la CR2450 (6,2 gramos, 560 mAh) de Panasonic (Litio + MnO_2) [22]. La vida útil la hemos calculado como el cociente entre la carga de la batería y la intensidad consumida por el conjunto alimentado a 3V.

En la parte superior se ve la vida útil para las pilas de mayor carga (CR2450), con las que se obtendrían unas 90 horas de trabajo ininterrumpido. En el caso de las pilas más pequeñas (CR2032) se conseguirían unas 37 horas.

Se puede apreciar que para la misma pila y frecuencia de muestreo la vida útil es algo mayor en el caso de utilizar el intervalo de conexión de 100 ms. También se ve que a mayor frecuencia disminuye la vida útil de la pila.

FIGURA 7.4. Vida útil en función de la frecuencia de muestreo y el intervalo de conexión



CAPÍTULO 8

Conclusiones y posibles mejoras

8.1. Conclusiones

En este proyecto se ha diseñado, construido y programado un prototipo de adquisición y transmisión de datos inerciales y magnéticos de bajo consumo.

Se han desarrollado dos sistemas que cooperan a través de un radioenlace que sigue el estándar *Bluetooth Low Energy* (BLE). El sistema de recepción, procesado, visualización y almacenamiento se aloja en un *smartphone*. El sistema de adquisición y transmisión de datos se basa en un módulo BLE (BL600 de *Laird Tech.*) y en un multichip con sensores de movimiento de última generación (MPU-9250 de *Invensense*), ambos programados en *smartBASIC*, una variante de Basic orientada a la programación por eventos. La carga y depuración se realiza desde PC a través del puerto serie.

Se han seleccionado los componentes del sistema de adquisición y transmisión, buscando minimizar el consumo de energía. Se han construido varios prototipos combinando placas de circuito impreso diseñadas por el *BioLab3* (MPU-9250, BL600 y MAX232) en una placa de prototipado. En caso de ser necesario, sería factible un encapsulado de reducido tamaño y peso. A los dos componentes principales habría que añadir dos resistencias externas y la pila de botón.

Por otro lado se han escrito los programas necesarios en los dos sistemas. Se ha programado paralelamente el *Servidor BLE de adquisición de datos*, cargado en el BL600, y el *Cliente BLE*, ejecutado en un *smartphone* con sistema operativo Android.

El *Servidor BLE de adquisición de datos* lee los datos medidos en los sensores y los almacena en un conjunto de servicios y características siguiendo la estructura que define el perfil GATT. El *Cliente BLE* recibe estos datos, los procesa, los muestra por pantalla y los escribe en un fichero. Es posible cambiar dinámicamente el periodo de muestreo desde el *smartphone*. Se han conseguido intercambiar datos mediante tres modos de comunicación distintos: indicación, lectura y escritura; en los dos primeros los datos viajan de *server* a *client* y en el último de *client* a *server*.

El sistema ha cumplido con los requisitos planteados inicialmente y funciona correctamente. Se han añadido varias mejoras no planificadas, como el control del periodo de muestreo y el almacenamiento de los datos en ficheros de la memoria externa del *smartphone*. Esta última mejora significa disponer de todos los datos medidos, incluyendo en cada línea el tiempo de recepción de cada medida. Permitirá al *BioLab3* analizar el movimiento "*off-line*".

Estudiando dichos ficheros de datos se observó que el sistema no respondía a periodos de muestreo inferiores a cierta cota. Se ha investigado la causa, determinando que la limitación se debe al intervalo de conexión y al protocolo envío/confirmación de la comunicación por indicación. Para el rango de los periodos de muestreo alcanzables se ha realizado un análisis del consumo energético, concluyendo que la autonomía del sistema, aunque mejorable, parece suficiente.

El prototipo no queda cerrado a la aplicación descrita, si no que el software desarrollado puede ser utilizado como una base o plantilla para otros proyectos relacionados con sensores portátiles y comunicación inalámbrica BLE.

Si bien la autonomía del sistema es un poco menor a la que esperábamos, se ha demostrado que la tecnología BLE consume realmente poco y que la elección del módulo BLE en este aspecto ha sido correcta. El BL600 tan solo consume unos 4 mA de media muestreando cada segundo. El consumo del sistema se debe principalmente al MPU-9250. Este consume aproximadamente dos tercios de la potencia total del sistema. El sensor que más consume con diferencia es el giróscopo.

Por otro lado, el lenguaje *smartBASIC* ha resultado ser de gran facilidad de construcción. No obstante, se han puesto en evidencia ciertas carencias. Por ejemplo, *smartBASIC* tan solo soporta de dos tipos de variables: *integer* (32 bits) y *string*. En etapas de desarrollo habrían sido útiles tipos como el *float*.

Debido a la falta de medios y a la escasa información encontrada en *Internet* el desarrollo y depuración de la aplicación ha sido bastante difícil. Por cuestiones económicas, no se pudo adquirir el kit de desarrollo del BL600 (Development Kit BL600 [18]), que habría facilitado la depuración de código. Se recomienda su adquisición para futuros desarrollos.

8.2. Posibles mejoras

Una posibilidad de reducción de consumo consistiría en crear tres características, una para cada sensor, con permisos de escritura, que actuaran a modo de interruptores (ON, OFF) de cada uno de los sensores. De esta forma podríamos tener activos únicamente los sensores necesarios. Según el Datasheet del MPU-9250 [8], el giroscopio consumo unos 3.2 mA, el acelerómetro unos 450 μA y el magnetómetro unos 280 μA. Si no fuera necesaria la velocidad angular en nuestra aplicación el consumo se reduciría enormemente.

Otra importante mejora sería poder conectar varios periféricos al mismo *central*. De esta forma podríamos controlar desde el Cliente BLE varios Servidores BLE (diferentes sistemas de adquisición de datos) y recibir datos de todos ellos a la vez. Esta funcionalidad sería muy interesante para las aplicaciones del *BioLab3*.

Si la aplicación lo requiere, se podría aumentar notablemente la velocidad de envío de datos cambiando indicaciones por notificaciones. En estas últimas, el *client* no envía confirmación (ACK). En cada intervalo de conexión se pueden enviar varias notificaciones (depende del *smartphone* utilizado, pero al menos permiten 4 notificaciones). Aumentaría el consumo y se perdería algo de precisión ya que no se advierte de la recepción del dato por parte del *smartphone*, pero seguramente se podría alcanzar un periodo de muestreo de unos 50 ms. En caso de emprender esta modificación, se recomienda adquirir un analizador de protocolos BLE de bajo coste.

Por último, tal y como se nombró en el apartado 4.1.1 el MPU-9250 contiene embebido un procesador de movimiento digital (DMP). Este recoge los datos del acelerómetro, giroscopio y magnetómetro y los combina y procesa a través de una serie de algoritmos de fusión de movimiento. Invensense no facilita que algoritmos de fusión utiliza su DMP, pero posiblemente se trate de un filtro de Kalman. La información resultante del DMP es más precisa y completa que la obtenida con cada sensor por separado. Al proveer datos ya procesados, el DMP puede significar un ahorro de consumo y recursos en el procesador de destino, además de simplificar la temporización.

ANEXO A

Carga y desarrollo del proyecto

A.1. Gestión del tiempo

El proyecto se ha desarrollado desde noviembre de 2014 hasta junio de 2015, en dedicación a tiempo parcial. En el diagrama de Gantt que se presenta en la Tabla A.1 se pude observar la distribución en el tiempo de las distintas tareas.

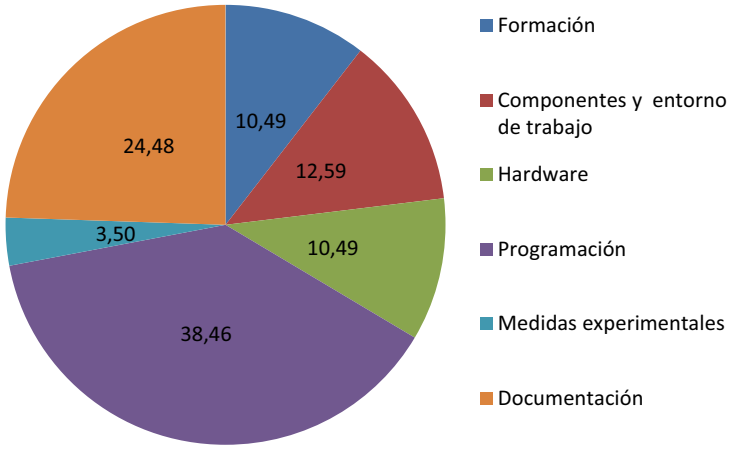
TABLA A.1. Distribución de tareas

Tareas	Horas	Nov	Dic	Ene	Feb	Mar	Abr	May	Jun	Total
FORMACIÓN										45
Estudio Java y Android	25									
Estudio tecnología BLE	20									
FAMILIARIZACIÓN COMPONENTES Y ENTORNOS										54
MPU-9250	4									
BL600 (hw +sw)	14									
Programación SmartBasic	16									
Pruebas BL600 - PC (UART)	5									
Pruebas BL600 - smartphone	10									
HARDWARE										45
Elección de componentes	9									
Montaje placas	18									
puesta en marcha	18									
PROGRAMACIÓN										165
Servidor BLE de adquisición de datos	81									
Cliente BLE	84									
MEDIDAS EXPERIMENTALES										15
Intervalo de conexión	5									
Estudio de consumo	10									
DOCUMENTACIÓN Y ORGANIZACIÓN										105
Informes de progreso	17									
Redacción de la memoria	76									
Reuniones directores de proyecto	12									
										429

A.2. Esfuerzo invertido

Este proyecto ha supuesto un total de 429 horas de trabajo. En el diagrama circular de la Figura A.1se puede observar el porcentaje de horas dedicado a cada tarea. En la Figura A.1 se muestra más detalladamente la cantidad de horas invertidas en las actividades que componen cada tarea principal. La tarea que mayor número de horas ha supuesto corresponde a la programación de las dos aplicaciones BLE descritas en el Capítulo 6.

FIGURA A.1. Porcentaje de horas dedicadas a cada tarea.



ANEXO B

Perfil GATT

Resumen

El propósito de este anexo es profundizar en los aspectos más importantes del Generic Attribute Profile (GATT).

Se describen de nuevo con más detalle los atributos y los roles que define el GATT (que coinciden con client y server del ATT).

Posteriormente se explican cómo están agrupados los atributos en un GATT Server a través de servicios, características y descriptores, fundamentales en el desarrollo del software del proyecto.

El GATT define un modelo de datos y procedimientos que permiten a los dispositivos descubrir, otros dispositivos, escribir, leer y enviar datos. Es, esencialmente, la capa más alta de datos de BLE. El GATT utiliza y está basado en el protocolo. El GATT además define como los datos están organizados de forma jerárquica en secciones llamadas servicios, formados por piezas de datos relacionadas denominadas características.

B.3. Roles

A diferencia de los roles que se definen durante el proceso de conexión (*central* vs. *peripheral*), los roles que define el GATT tienen sentido cuando ya se ha establecido una conexión entre dos dispositivos. Estos roles, que son los mismos que se han explicado en el ATT, determinan quién habla a quién y quién lleva la iniciativa en una conexión:

- *Client*: Envía peticiones al *server* y recibe respuestas de este. El *client* "necesita" la información de la que dispone el *server*. Como de antemano el *client* no sabe nada de los atributos de los que dispone el *server*, comienza descubriendo la presencia y naturaleza de estos a través del *handle* y el UUID. Una vez se ha completado este proceso, el *client* puede empezar a leer atributos o a escribir en ellos, además de recibir actualizaciones iniciadas por el server (indicaciones y notificaciones).
- *Server*: Recibe peticiones del *client* y le envía respuestas. Es el responsable de almacenar y hacer que todos los datos estén a disposición del *client*, en forma de atributos.

B.4. Atributos

Los atributos son las entidades de datos más pequeñas definidas por el GATT (y por el ATT). Ambos solo pueden trabajar con atributos, por lo que toda la información debe estar organizada de esta forma. Los atributos están siempre localizados en el *server*, a los que puede acceder y modificar el *client*. Un atributo está formado por los siguientes campos:

Handle. Es un identificador de 16 bits. El *client* debe descubrir en primer lugar el *handle* del atributo, para luego poder acceder a su valor. Es la parte del atributo que lo hace direccionable y localizable. Su rango va desde 0x0000 hasta 0xFFFF.

UUID (*Universally unique identifier*). Es un identificador que determina la naturaleza y tipo del atributo. Si ha sido definido por la especificación Bluetooth se utiliza un formato acortado de 16 bits. En otro caso posee una longitud de 128 bits.

Valor del atributo. Es la parte del atributo a la que el *client* puede acceder, leyendo o escribiendo.

Permisos. Especifican que operaciones ATT pueden ser ejecutadas en el valor de cada atributo. Existen permisos de lectura, escritura, encriptación y autorización del atributo.

La Tabla B.2 muestra un conjunto de atributos contenidos en un GATT *server*. Cada fila representa un único atributo, mientras que cada columna representa las diferentes partes constituyentes del atributo.

TABLA B.2. Atributos representados como una tabla

Handle	Type	Permissions	Value	Value length
0x0201	$UUID_1$(16b)	Read only	0x180A	2
0x0202	$UUID_2$(16b)	Write only	0x2A29	2
0x0215	$UUID_3$(16b)	Read/write encryption required	36.43	8
0x030C	$UUID_4$(128b)	Read/write, authorization required	{0xFF, 0x00}	2

B.5. Servicios

Los servicios son colecciones de características y relaciones a otros servicios que determinan una parte del funcionamiento del dispositivo. Cada servicio perteneciente a un GATT *server* comienza con un atributo que marca el inicio del servicio, denominado declaración del servicio.

La Tabla B.3, que representa un GATT server compuesto de dos servicios, sirve para entender mejor este concepto:

TABLA B.3. Ejemplo de atributo de declaración de un servicio en un server

Handle	UUID	Permissions	Value	Value length	Description
0x0100	0x2800	Read only	Service A UUID	2 or 16 bytes	Service A declaration
...	...	...	...	...	Service A details
0x0150	0x2800	Read only	Service B UUID	2 or 16 bytes	Service B declaration
...	...	...	...	...	Service B details

La piedra angular de un servicio es el atributo con UUID igual a 0x2800 [5]. Este marca el inicio del servicio. Todos los atributos que le sigan pertenecerán a este servicio hasta que otro atributo con UUID igual a 0x2800 sea encontrado. Cada atributo no sabe por sí mismo a que servicio pertenece, el GATT necesita un rango de *handles* para poder localizarlos, y esos rangos solo son descubiertos si existe una UUID de 0x2800. Aquí es donde el valor del *handle* adquiere importancia. En la tabla anterior, para que un atributo pertenezca al Servicio A, su handle debe estar dentro del rango 0x0100 y 0x0150.

El valor del atributo de la declaración del servicio determina de que servicio se trata. De esta forma, cada declaración de servicio contiene dos UUIDs: 0x2800 como el UUID del atributo, y otro almacenado en su valor que específica el servicio del que se trata (si es un servicio de batería, de presión sanguínea...).

En el siguiente link se puede ver el listado de servicios definidos por Bluetooth SIG: https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx

B.6. Características

Cada servicio GATT se compone de características que almacenan información útil para los servicios. Cada característica incluye al menos dos atributos: la propia declaración de la característica y el valor de la característica (ver Tabla B.4).

TABLA B.4. Declaración y valor de la característica

Handle	UUID	Permissions	Value
0x0001	0x2803	Read only	Handle: 0x0011 UUID: Characteristic UUID Properties
0x0011	Characteristic UUID	Any	Actual value

Una vez más, el UUID de la declaración de la característica está estandarizado, posee un valor de 0x2803 y determina el comienza de una característica. En el valor de este atributo hay un campo dedicado al UUID del valor de la característica, un campo para el *handle* del valor de la caracte-

rística y un último campo de 8 bits dedicado a definir las propiedades que tendrá la característica: *broadcast*, lectura, escritura sin respuesta, escritura, notificación, indicación, escritura con firma y escritura encolada.

Pero el atributo que realmente importa al programador es el valor de la característica. El tipo viene definido por su UUID: si se trata de una UUID aprobada por el SIG de Bluetooth (16 bits) o si ha sido creada por el usuario (*custom characteristic*, 128 bits).

Por último, el valor del atributo de la característica (*Actual value*) contiene los actuales datos a los que el *client* puede libremente acceder para intercambiar información. Este puede contener todo tipo de datos imaginables: desde temperaturas en grados hasta datos cinemáticos.

En el siguiente link se puede ver el listado de características definidas por Bluetooth SIG: https://developer.bluetooth.org/gatt/characteristics/Pages/Characteristics-Home.aspx

B.7. Descriptores

Los descriptores son información adicional de una característica y su valor. Están situados siempre después del valor del atributo de la característica. Están formados por un único atributo, la declaración del descriptor de la característica.

Al igual que ocurre con los servicios y características, existen dos tipos de descriptores: aquellos definidos por el GATT y los realizados por el propio usuario.

El más importante y más usado es sin duda el *Client Characteristic Configuration Descriptor* (CCCD). Su función es simple: actúa como un interruptor, habilitando o deshabilitando las operaciones iniciadas por el server, pero solo sobre el valor de la característica en la que se encuentran. El valor del CCCD no es más que un campo de 2 bits: uno correspondiente a las **notificaciones** y otro a las **indicaciones**. Por ejemplo, para habilitar las notificaciones en una característica, el *client* tiene que escribir un 1 en el primer bit del CCCD.

ANEXO C

Mapa de registros utilizados y ejemplos de aplicación

Resumen

En este anexo se pueden consultar los registros que se han utilizado en la aplicación "Servidor BLE de adquisición de datos", tanto del MPU-9250 como del magnetómetro AK8963.

Por último se incluyen dos ejemplos prácticos sacados de la aplicación "Servidor BLE de adquisición datos" que hacen uso de la mayoría de registros mencionados. El primero utiliza un timer para controlar la lectura de datos del registro de la aceleración y el segundo realiza los pasos necesarios para activar el magnetómetro y poder leer sus valores.

C.1. Mapa de registros y descripciones del MPU-9250

A continuación se irán detallando cada uno de los registros utilizados del MPU-9250 [13].

C.1.1 Configuración girocopio: registro 27

El registro de configuración del giroscopio permite configurar el rango de medida de la velocidad angular (ver Tabla C.1).

TABLA C.1. Gyroscope Configuration Register. Address: 0x1B.

BIT	NAME	FUNCTION
[7]	XGYRO_Cten	X Gyro self-test
[6]	YGYRO_Cten	Y Gyro self-test
[5]	ZGYRO_Cten	Z Gyro self-test
[4:3]	GYRO_FS_SEL[1:0]	Gyro full scale Select: 00 = ±250 º/s 01 = ±500 º/s 10 = ±1000 º/s 11= ±2000 º/s
[2]	-	Reserved
[1:0]	Fchoice-b[1:0]	Used to by-pass DLPF

Con los bits 4 y 3 podemos configurar el rango de medidas del giroscopio. Según el valor escogido tendremos que dividir el valor de la velocidad angular por un determinado factor de escala. Esta operación se realiza en el Cliente BLE.

En la Tabla C.2 se aprecian los factores de escala según el rango de medidas que se utilicen.

TABLA C.2. Sensitivity Scale Factor for gyroscope.

GYRO_FS_SEL	VALUE	UNITS
0	131	LSB/(º/s)
1	65,5	LSB/(º/s)
2	32,8	LSB/(º/s)
3	16,4	LSB/(º/s)

Se ha trabajado con un rango de ±250º/s (GYRO_FS_SEL = 0).

C.1.2 Configuración acelerómetro: registro 28

El registro de configuración del giroscopio permite configurar el rango de medida de la velocidad angular (ver Tabla C.1).

TABLA C.3. Accelerometer Configuration Register. Address: 0x1C

BIT	NAME	FUNCTION
[7]	ax_st_en	X Accel self-test
[6]	ay_st_en	Y Accel self-test
[5]	az_st_en	Z Accel self-test
[4:3]	ACCEL_FS_SEL[1:0]	Accele full scale Select: 00 = ±2 g 01 = ±4 g 10 = ±8 g 11= ±16 g
[2:0]	-	Reserved

Con los bits 4 y 3 podemos configurar el rango de medidas del acelerómetro. Según el valor escogido tendremos que dividir el valor de la aceleración por un determinado factor de escala, cuyos valores se muestran en la siguiente tabla (ver Tabla C.4).

TABLA C.4. Sensitivity Scale Factor for accelerometer.

GYRO_FS_SEL	VALUE	UNITS
0	16384	LSB/g
1	658192	LSB/g
2	4096	LSB/g
3	2048	LSB/g

Se ha trabajado con un rango de ±2 g (ACCEL_FS_SEL = 0).

C.1.3 Habilitación del modo *by-pass*: registro 55

Este registro se usa para activar el modo *by-pass* cuando se quiere acceder al magnetómetro o a un sensor externo (ver Tabla C.5).

TABLA C.5. Bypass Enable Configuration Register. Address: 0x37

BIT	NAME	FUNCTION
...	...	...
[1]	BYPASS_EN	When asserted, the I2C master interface pins (ES_CL, ES_DA) will go into bypass mode when the I2C master interface is disabled
[0]	..	...

Poniendo el bit [1] a 1 accedemos al modo *by-pass* para poder leer los registros del magnetómetro (ver 4.1.1).

C.1.4 Medidas acelerómetro: registros 59 a 64

Este registro de 8 bits almacena el byte más significativo de las medidas de la aceleración (ver Tabla C.6), mientras que el siguiente almacena el byte menos significativo (ver Tabla C.7).

TABLA C.6. High byte of x-acceleration data register. Address: 0x3B

BIT	NAME	FUNCTION
[7:0]	ACCEL_XOUT_H	High byte of accelerometer X-axis data

TABLA C.7. Low byte of x-acceleration data register. Address: 0x43

BIT	NAME	FUNCTION
[7:0]	ACCEL_XOUT_L	Low byte of accelerometer X-axis data

La salida es en formato complemento A2. El proceso de juntar los dos bytes en una misma variable se realiza en el Servidor BLE de adquisición de datos, mientras que la interpretación del signo en complemento A2 se realiza en el cliente BLE.

El cálculo del valor de la aceleración en el eje X se realizaría a partir de la siguiente ecuación:

$$XAcceleration = (AccelXOut)/(AccelSensitivityScaleFactor) \quad \text{(EC C.1)}$$

Equivalente para los ejes Y y Z.

C.1.5 Medidas giroscopio: registros 67 a 72

TABLA C.8. High byte of x-gyroscope data register. Address: 0x44

BIT	NAME	FUNCTION
[7:0]	GYR_XOUT_H	High byte of the X-axis gyroscope output

TABLA C.9. Low byte of x-gyroscpe data register. Address: 0x3C

BIT	NAME	FUNCTION
[7:0]	GYR_XOUT_L	Low byte of the X-axis gyroscope output

La salida es en formato complemento A2. El proceso de juntar los dos bytes en una misma variable se realiza en el Servidor BLE de adquisición de datos, mientras que la interpretación del signo en complemento A2 se realiza en el cliente BLE.

El cálculo del valor de la velocidad angular en el eje X (equivalente para los ejes Y y Z) se realizaría a partir de la siguiente ecuación:

$$XAngularRate = (GYRXOUT)/(GyrSensitivityScaleFactor) \quad \text{(EC C.2)}$$

C.2. Mapa de registros y descripciones del magnetómetro

C.2.1 Registro de estado 1

El registro de estado del AK8963 (ver Tabla C.10) se utiliza para comprobar si en los registros que almacenan las medidas del campo magnético hay nuevos datos disponibles.

TABLA C.10. Status 1 register. Address: 0x02. Read only

BIT	NAME	FUNCTION
...	...	...
0	DRDY	Data ready: “0” normal, “1” data is ready DRDY bit turns to 1 when data is ready in single measurement mode. It returns to “0” when measurement data register is read

Antes de leer los registros de los valores medidos por el magnetómetro debemos comprobar que DRDY está a “1”.

C.2.2 Datos medidos: HXL a HZH

Estos 6 registros de 8 bits almacenan los valores medidos por el magnetómetro.

TABLA C.11. Measurement Data Registers.

Address	Name	B7	...	B0
0x03	HXL: X-axis measurement data lower byte	HX7		HX0
0x04	HXH: X-axis measurement data higher byte	HX15		HX8
0x05	HYL: Y-axis measurement data lower byte	HY7		HY0
0x06	HYH:Y-axis measurement data higher byte	HY15		HY8
0x07	HZL: Z-axis measurement data lower byte	HZ7		HZ0
0x08	HZH: Z-axis measurement data higher byte	HZ15		HZ8

Los datos son almacenados en complemento A2 y formato *Little Endian*.

C.2.3 Registro de control 1

El registro de control (ver Tabla C.12) permite fijar la longitud de salida de los valores medidos por el magnetómetro y seleccionar el modo de medida.

TABLA C.12. Control 1 register. Address: 0x0A.

BIT	NAME	FUNCTION
[4]	BIT	Output bit setting: "0" 14-bit output, "1" 16-bit output
[3:0]	MODE	"0000": Power-down mode "0001": Single measurement mode "0010": Continuous measurement mode 1 ...

Antes de realizar una lectura de los registros de datos medidos del magnetómetro debemos seleccionar el *single measurement mode* y formato de salida de 16-bits de longitud (Ver el ejemplo del capítulo 4.3.2).

C.2.4 Registro de coeficientes de ajuste de sensibilidad

Estos tres registros de 8 bits cada uno almacenan el coeficiente de ajuste de sensibilidad de cada coordenada (ver Tabla C.13).

TABLA C.13. Sensitivity Adjustment Values Register. Read only

Address	Name	B7	...	B0
0x10	ASAX: Magnetic sensor X-axis sensitivity adjustment value	COEFX7		COEFX0
0x11	ASAY Magnetic sensor Y-axis sensitivity adjustment value	COEFY7		COEFY0
0x12	ASAZ Magnetic sensor Z-axis sensitivity adjustment value	COEFZ7		COEFZ0

Estos valores están calibrados para cada chip y están almacenados en una ROM de fusibles.

El ajuste de sensibilidad se realiza con la siguiente ecuación:

$$Hadj = H \times \left(\frac{(ASA - 128) \times 0,5}{128} + 1 \right) \qquad \text{(EC C.3)}$$

Donde H es el magnetismo medido por el sensor extraído del registro de datos medidos, ASA el coeficiente de ajuste de sensibilidad y $Hadj$ el valor del magnetismo una vez ajustado.

C.3. Ejemplos de aplicación

C.3.1 Abrir puerta I2C y leer valor de aceleración cuando el *timer* expira

Tras definir e inicializar las variables se ejecuta en primer lugar la función que inicializa el sensor (3): el *timer* recurrente comienza a contar (5) y se abre la puerta I2C (6). Cuando el *Timer0* expira es llamada la función *HandlerTimer0* (8), que realiza la lectura del byte mas significativo (11) y menos significativo (12) de cada componente de la aceleración. Por último, el MSB se desplaza 8 bits a la izquierda y se junta con el LSB en una misma variable (13).

```
  // Initialize the name of the registers
1 DIM MPU9250ADRESS, handle
 // Set the address of the MPU-9250
2 MPU9250ADRESS = 0x68

3 sub InitSensor()
4 dim rc
 // poll sensor on timer n°0 every 1000ms. Recurrent Timer
5 TimerStart(0,10000,1)

 // Open I2C Peripheral
6 rc = I2cOpen(100000, 0, handle)
7 endsub

8 function HandlerTimer0() as integer
9 dim rc
10dim acc_x_MSB, acc_x_LSB, acc_y_MSB, acc_y_LSB, acc_z_MSB,

 //Read acceleration
 // Read the data on the X component of the accelerometer
11 rc = I2cReadReg8(MPU9250_ADDRESS, 0x3B, acc_x_MSB)
12 rc = I2cReadReg8(MPU9250_ADDRESS, 0x3C, acc_x_LSB)

 // Read the data on the Y component of the accelerometer
 ...
 // Read the data on the Z component of the accelerometer
 ...
 //Shift the MSB 8 positions to right and OR with LSB
13 acc_x = ((acc_x_MSB << 8) | acc_x_LSB )
...
14 endfunc 1

 // Equivalent to main() in C
 // Enable synchronous event handlers

 //This  event is thrown when Timer0 expires
15 OnEvent  EVTMR0 call HandlerTimer0

 //Initialise the accelerometer sensor
16 InitSensor()

17 WaitEvent // Wait for a synchronous event
```

C.3.2 Activar el magnetómetro

Se selecciona en primer lugar el modo *by-pass* para poder acceder directamente al magnetómetro (1). A través del registro de control (0x0A) se selecciona el modo de medida única y se fija la salida de los valores del magnetismo en una longitud de 16 bits (2). Cuando el valor DRDY, "Data Ready" sea igual a 1 (3, 4, 5) realizamos la lectura del campo magnético de cada coordenada (6). Por último volvemos a seleccionar el modo de medida única para poder realizar la siguiente lectura del magnetómetro (7).

```
 //Set by-pass (Pass-Through) mode for the magnetometer
1 rc = I2cWriteReg8(MPU9250_ADDRESS, 0x37, 0x02)

//Request first magnetometer single measurement and set 16-bit output
2 rc = I2cWriteReg8(AK8963_ADDRESS, 0x0A, 0x11)

3 DO
 //Read register status 1 and wait for DRDY: "Data ready"
4  rc = I2cReadReg8(AK8963_ADDRESS, 0x02, DRDY)
5 UNTIL ( DRDY & 0x00000001 ) == 0x00000001

//Read X,Y,Z magnetic values
6 rc = I2cReadReg8(AK8963_ADDRESS, 0x03, h_x_LSB)
//...

//Request next magnetometer single measurement and set 16-bit output
7 rc = I2cWriteReg8(AK8963_ADDRESS, 0x0A, 0x11)
```

ANEXO D *Informe de Progreso. Sistema de Adquisición para el BioLab3*

Rapporto sistema di acquisizione

Raúl Viñals Mariñosa
Marzo 2015

1. Introduzione

Il progetto che viene descritto a continuazione sviluppa un sistema di acquisizione di dati cinematici attraverso un sensore magneto-inerziale, vedi Figura 1. Le parti che formano il sistema sono:

- **Sensore magneto-inerziale (IMU)**: Dispositivo elettronico che misura la accelerazione, velocità ed orientazione attraverso la combinazione di un accelerometro, un giroscopio ed un magnetometro. Disporre di interfaccia I2C. In nostro caso utilizzeremo il MPU-9250 di *InvenSense* [1].
- **Modulo Bluetooth**: Si incarica por un lato di raccogliere i dati inerziali del MPU via I2C e in secondo luogo di trasmetterli via Bluetooth. Abbiamo scelto il BL-600 SA di *Laird Technologies* [4].
- **Dispositivo di visualizzazione dei dati**: Riceve i dati via Bluetooth e li mostra. Ad esempio un cellulare con tecnologia Bluetooth Low Energy.
- **Batteria**: Fornisce i voltaggi adeguati.

- **Oscillatore**: genera la segnale di clock per l'insieme BL600 y MPU-9250.

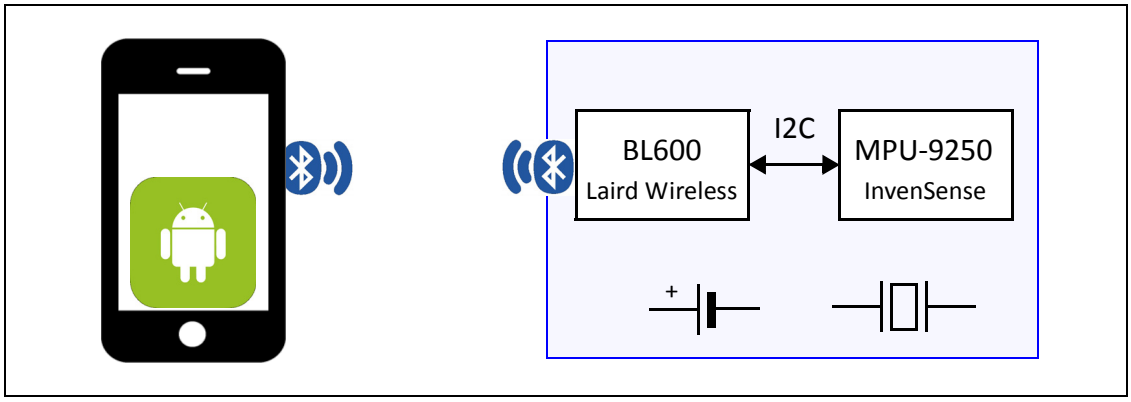


Figura 1. Elementi del sistema.

2. Scelta modulo Bluetooth Low Energy [4].

Ai fini della applicazione che si vuole ottenere se hanno fissato i principali caratteristiche che deve avere il modulo Bluetooth:

- Basso consumo, piccole dimensione e I2C interfaccia.

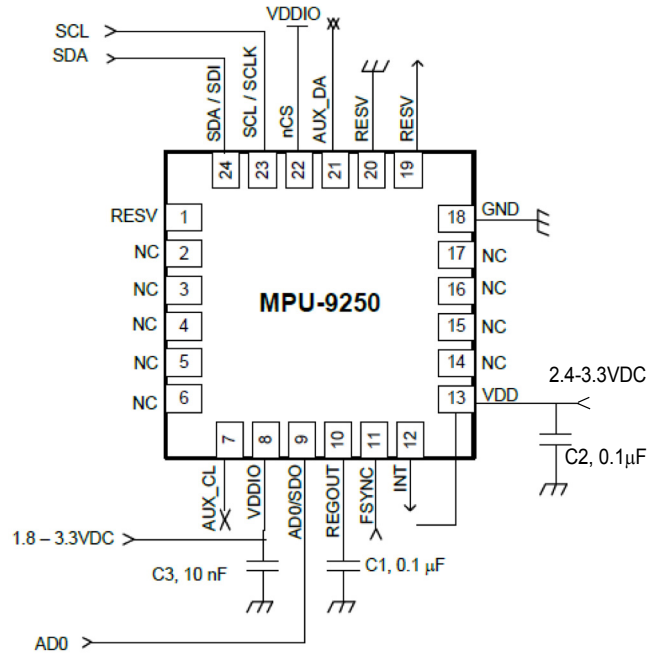
Dopo una ricerca di mercato per studiare e comparare le principale moduli Bluetooth se h a scelto il modulo Bluetooth BL600 SA di *Laird Technologies*. Il BL600 possiede la migliore relazione consumo/dimensione e interfaccia I2C. La Tabella 1 mostra le diverse alternative comparate e le loro caratteristiche principali.

	BR-LE4.0-S3A BlueRadios	**BLE113** Bluegiga	**OLP425** u-blox	**PAN 1721** Panasonic	**BL600** Laird Technologies
Power consumption [mA]	18,2 TX 17,9 RX 0,235 PM1	18,2 TX 17,9 RX 0,270 PM1	18 TX 18 RX	14,3 TX 14,7 RX	10,5 TX
Power supply [V]	V_{DD}=[2-3,6]	V_{DD}=[2-3,6]	V_{DD}=[2-3,6]	V_{DD}=[2-3,6]	V_{DD}=[1,8-3,6]
Dimensions [mm]	11,8x17,6x1,9	15,7x9,15x2,1	14,8x22,3x3,2	14,5x8,2x3	19x12,5x3
Volume [mm^3]	394,6	302,6	1056,1	356,7	731,25
I2C interface	OK	OK	OK	OK	OK

Tabella 1. Comparazione moduli Bluetooth.

3. Sensore magneto-inerziale MPU 9250 [1].

Il tipico circuito operativo che se deve montare per la operazione I2C e la descrizione dei pin più importanti appariscono nella Figura 2.



Pin Number	Pin Name	Pin description
1	RESV	Connect to VDD
7	AUX_CL	I2C Master serial clock
8	VDDIO	Digital I/O supply voltage
9	AD0	I2C Slave Address LSB, connect to GND
10	REGOUT	Regulator filter capacitor connection
11	FSYNC	Connect to GND
12	INT	Interrupt digital output. Open drain
13	VDD	Power supply voltage
18	GND	Power supply ground
19	RESV	Not connected
20	RESV	Connect to GND
21	AUX_DA	I2C master serial data. No connect
22	nCS	Connect to VDD
23	SCL	I2C serial clock
24	SDA	I2C serial data
2-6,14-17	NC	Not connected

Figura 2. Pin e connessione per la operazione I2C nel chip MPU 9250.

4. Modulo Bluetooth BL 600

4.1. Connessione BL600-PC

Prima di iniziare la trasmissione di dati via I2C, ci abbiamo comunicato con il modulo Bluetooth attraverso l'*UW Terminal* di *Laird Technologies*. È un terminale installato sul computer che permette comunicare con il modulo Bluetooth, girare programmi smartBasic e cominciare a imparare i fondamenti di *smartBasic* senza la necessità di altre h ardware. I requisiti sono i seguenti:

- Windows XP o posteriore.
- UW Terminal v6.21, o posteriore.
- Text Editor. Noi abbiamo utilizzato Notepad++.
- Manuale *smartBasic* [5].
- Adattatore porta seriale MAX232 [2].

Il adattatore della porta seriale contiene il MAX232 di *Maxim* che adatta i livelli di tensione del BL600 al standard RS232 di 5 V, vedere Figura 3.

- The MAX232 device is a dual driver/receiver that includes a capacitive voltage generator to supply TIA/EIA-232-F voltage levels from a single 5-V supply.
- Each receiver converts TIA/EIA-232-F inputs to 5-V TTL/CMOS levels. These receivers have a typical threshold of 1.3 V, a typical hysteresis of 0.5 V, and can accept ±30-V inputs. Each driver converts TTL/CMOS.input levels into TIA/EIA-232-F levels.

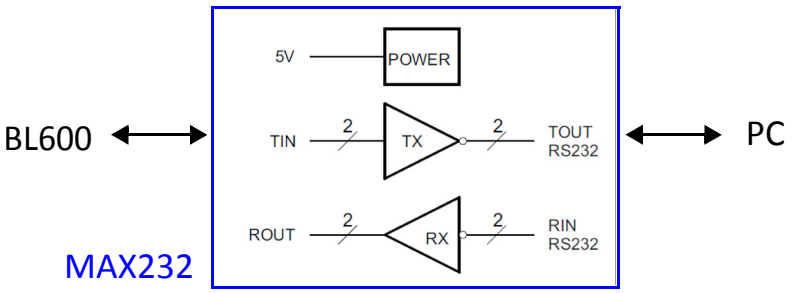


Figura 3. Descrizione basica del dispositivo MAX232.

Le Figura 4 rappresenta la configurazione dei pin del MAX232[3] e la configurazione della scheda del BioLab nella quale va montato il MAX232. La tensione di alimentazione del MAX232 è de 5V.

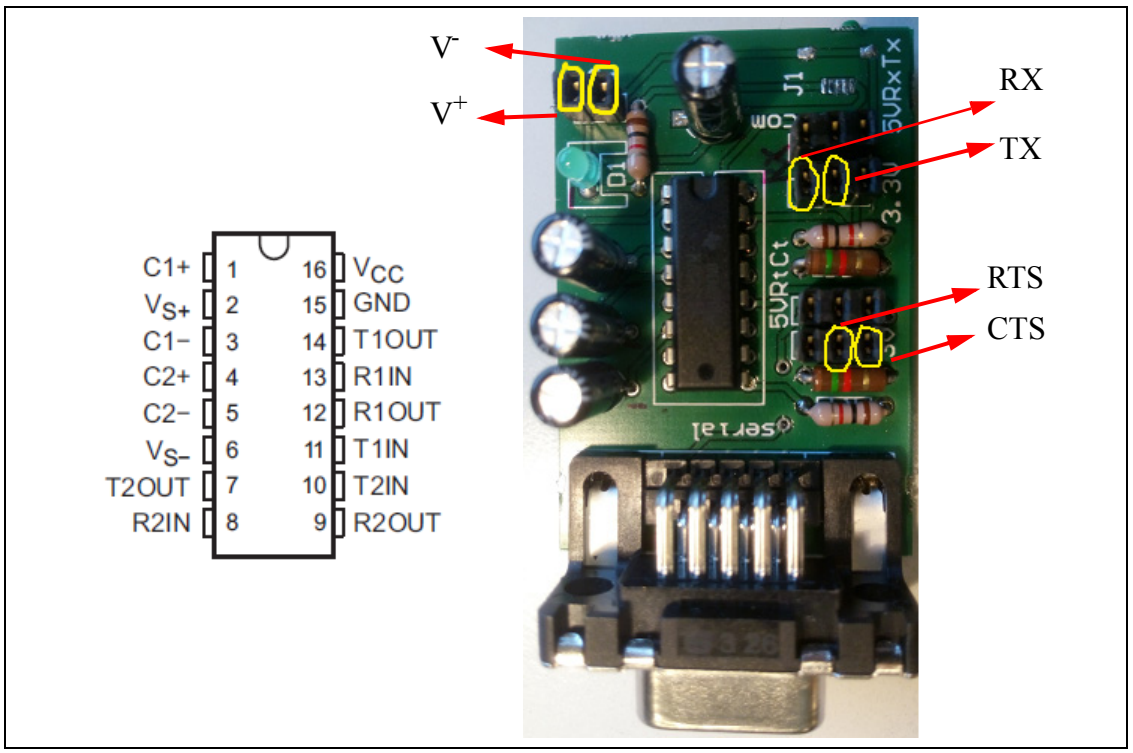


Figura 4. MAX232 e scheda.

4.2. Connessione BL600-MAX232

La Figura 5 mostra le connessione tra i circuiti BL600 y MAX232.

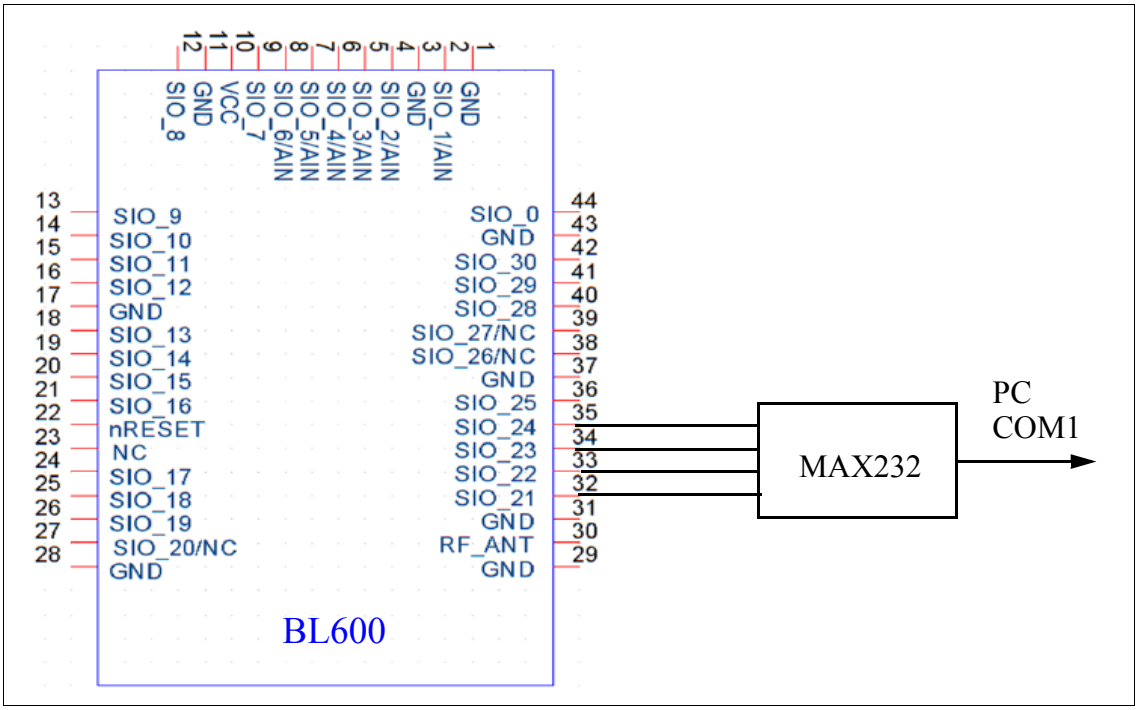


Figura 5. Connessione BL600-MAX232-PC

Le pin SIO_21 (UART TX), SIO_22 (UART RX), SIO_23 (UART RTS), SIO_24 (UART CTS) del BL600 devono essere collegati con le corrispondenti pin del MAX232.

I pin implicati nella comunicazione serie con il BL600 si possono vere sulla Tabella 2.

Pin #	Pin Name	Default Function	Alternat. Function[a]	Default Direction[b]	Pull-up Pull-down [b]	Comment
1	GND					Connect to GND
9	SIO_7	DIO		IN	Pull-down	
10	VCC					Connect to VCC
22	nRESET			IN		Connect to GND
32	SIO_21	DIO[c d]	UART TX[e]	OUT	Set high in FW	UARTCLOSE() selects DIO functionality and UARTOPEN() selects UART comms behaviour Connect to the MAX232
33	SIO_22	DIO	UART RX	IN	Pull-up	
34	SIO_23	DIO	UART RTS	OUT	Set low in FW	
35	SIO_24	DIO	UART CTS	IN	Pull-down	
36	SIO_25	DIO		IN	None	Laird Devkit: **UART_DTR** via CON12
40	nAuto-RUN	DIO		IN	None	Select between the two BL600 operating modes[f]

Tabella 2. Pin implicati BL600-MAX232

a. Secondary function is selectable in smartBASIC application.

b. smartBASIC runtime engine FW 1.5.66.0 (Apr2014) has DIO (Default Function) INPUT pins, have by default PULL-UP enabled. This was done to avoid floating inputs (which can also cause current consumption in low power modes (e.g. StandbyDoze) to drift with time.

c. DIO = Digital Input or Output. I/O voltage level tracks VCC.

d. DIO or AIN functionality is selected using the GpioSetFunc() function in smartBASIC.

e. I2C, UART, SPI controlled by xxxOPEN() functions in smartBASIC.

f. Pin nAutoRUN is an input with active low logic. The operating mode is selected by nAutoRUN pis status:
   - If low(0V), runs $autorun$ if it exists.
   - If high(VCC), runs via at+run (and “file name” of application)

4.3. UWTerminal v6.21 [6].

Una volta collegato tutto correttamente già possiamo cominciare a utilizzare l’*UWTerminal*:

- Aprire l’*UWTerminal* nel computer.
- Inserire la porta COM che si va a utilizzare e gli altri parametri:
  - Baudrate 9600
  - Parity None
  - Stop Bits 1

- Data Bits 8
- Handshaking CTS/RTS

Cliccando OK appare lo schermo principale dell'*UWTerminal*. È importante fare attenzione alle 4 tipi di LEDs indicatori situati nella parte superiore dello schermo. Questi mostrano lo stato delle linee di controllo del RS-232, vedere Figura 6.

Figura 6. UWTerminal tabs and status lights.

In relazione a la figura:

- CTS: *Clear to send*. Il verde indica che il modulo è pronto per ricevere dati.
- DSR: *Data set ready.*
- DCD: *Data Carrier Detect*.
- RI: *Ring Indicate*.

Se CTS sta verde intanto DSR, DCD e RI stano rosse significa che il modulo sta funzionando correttamente e che stiamo in condizioni di iniziare la comunicazione. Nel caso cui CTS diventa rosso fare speciale attenzione a che i pin del MAX232 stiano ben collegati.

Una volta che abbiamo verificato che CTS sta verde possiamo cominciare a interagire con il modulo Bluetooth attraverso una serie di comandi interattivi, dei quali i più tipici sono le seguenti:

- AT- Ritorna 00 se il modulo sta lavorando correttamente.
- AT I 3 - Mostra la revisione del firmware del modulo.
- AT I 4 - Mostra la direzione MAC del modulo.
- AT+DIR - Ritorna una lista di tutte le applicazioni caricati nel modulo. Si non è caricato nessun firmware risponde con "00"
- AT+DEL "filename" - Elimina una applicazione del modulo.
- AT+RUN "filename" - Fa partire una applicazione che già è caricata nel modulo. Per compilare una applicazione il smart Basic file deve essere salvato nella stessa cartella che il *UWTerminal*.
- ATZ - Resetea la CPU.

Per caricare un firmware nella modalità interattiva (PIN 40 = 1; PIN 9 = 0): dalla sch ermata dell'*UW Terminal* bisogna premere il tasto destro del mouse e scegliere l'opzione "XCompile + Load".

Per far girare un programma nella modalità *autorun*:

- Bisogna salvare il file del firmware con la scrittura “$autorun$name_firmware.sb”.
- Essere in modalità interattiva per caricare il firmware “$autorun$name_firmware.sb” tramite la UART.
- Una volta che il file è stato caricato, tramite il comando AT + DIR otteniamo la risponda visiva sull’UW Terminal: “$autorun$”.
- Spegniamo il modulo BL600 rimuovendo la batteria o la alimentazione esterna.
- Impostiamo le connessioni: PIN 40 = 0; PIN 9 = 1.
- Riaccendiamo il modulo e a questo punto il programma parte automaticamente.

Tutti i codici firmware *smartBasic* da caricare sul modulo BL600 tramita l’*UWTerminal* devono essere salvati con estensione “.sb” e nella stessa cartella dove si trova l’*UWTerminal* (smartBASIC_Sample_Apps). Le librerie vanno salvate con estensione “.sblib”.

Nel caso che il modulo non risponda ai comandi AT assicurarsi che il pin SIO_7 e il pin nAutoRUN (SIO_28) non stiano entrambi HIGH (VCC). In questo caso la UART è connessa con il servizio Virtual Serial Port e quindi il modulo BL600 non risponderà ai comandi AT ed i applicazioni non potranno essere caricati sul modulo. Nella modalità *Run mode* il pin nAutoRUN deve stare low (0V) e il pin SIO_7 high (VCC). Invece, nella modalità interattiva, nAutoRUN deve stare high e il pin SIO_7 low.

Vcc

4.4. Connessione BL600-MPU9250

Per poter iniziare la comunicazione del BL600 con il MPU9250 basta collegare i pin della interfaccia I2C (I2C_SDA, I2C_SCL) di ognuno dei dispositivi. Gli altri pin devono stare collegati esattamente come si ha descritto sopra. Le resistenze di pull-up delle linee SDA e SCL non sono forniti nel moduli, quindi si devono collegare esternamente, vedere Figura 7.

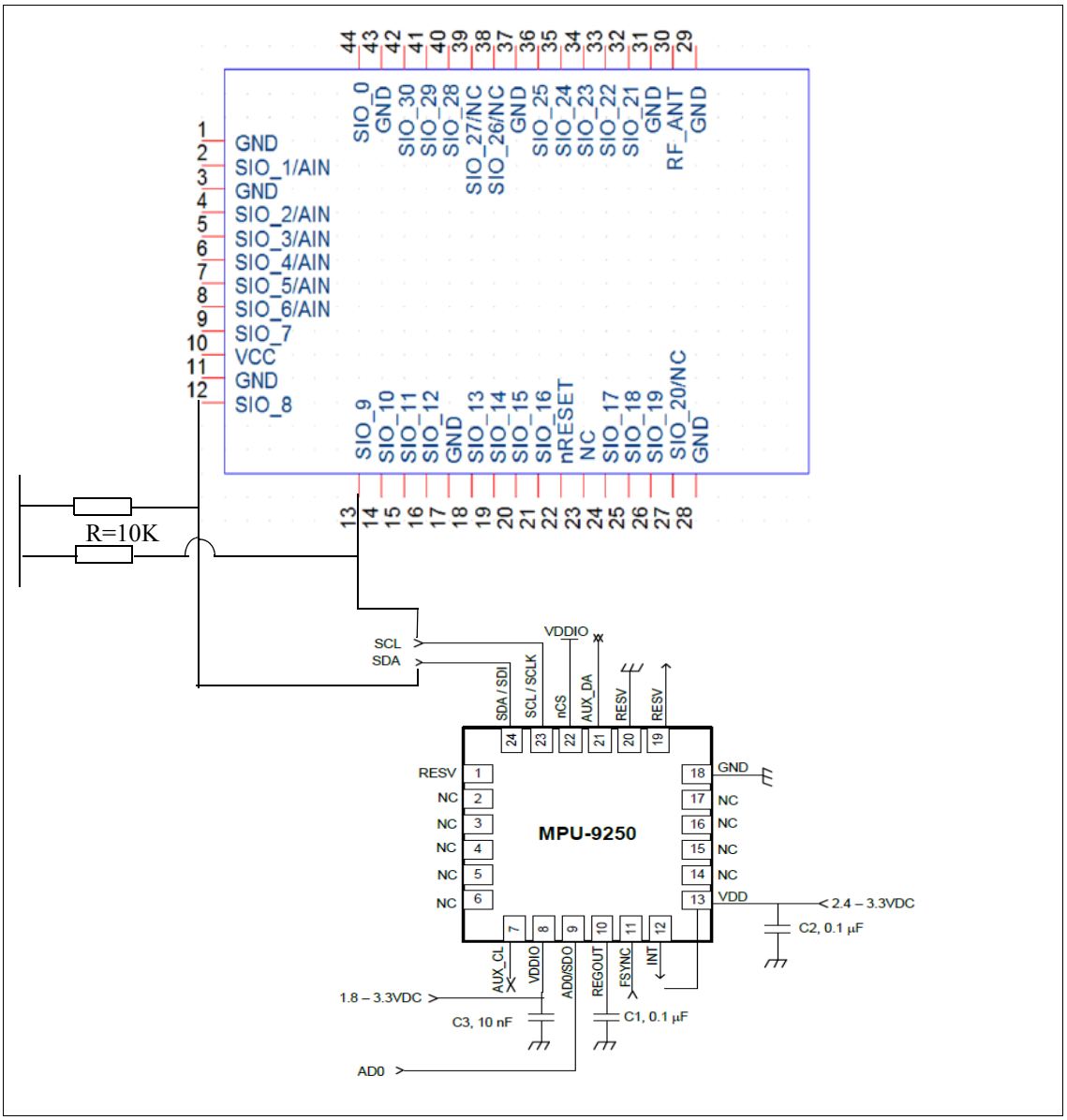


Figura 7. Connessione MPU9250 - BL600 via I2C

La prima prova che conviene fare per comprovare che la connessione è stata corretta è attraverso il registro *WHO_AM_I* del MPU9250 (Tabella 3), che indica al utente qual dispositivo viene accesso (il MPU).

BIT	NAME	FUNCTION
[7:0]	WHOAMI [a]	Register to indicate to user which device is being accessed

Tabella 3. WHOAMI register[8]

a. This register is used to verify the identity of the device. The contents of *WHO_AM_I* register is an 8-bit device ID. The default value of the register is 0x71.

Il codice smartBasic è il seguente:

```
// Initialize the name of the registers
DIM WHOAMI, MPU9250ADRESS

// Set the address of the registers
MPU9250ADRESS = 0x68                              // MPU-9250 address
WHOAMI = 0x75                                     // WHO_I_AM register address

// Open I2C Peripheral
rc = I2cOpen(100000, 0, handle)
IF rc != 0 THEN
  PRINT "\nFailed to open I2C interface with error code "; INTEGER.H' rc
ELSE
  PRINT "\nI2C open success"
ENDIF

// Read the data from WHO_I_AM register to see if the sensor is working
properly
rc = I2cReadReg8(MPU9250ADRESS, WHOAMI, c)

IF c == 0x71 THEN
  PRINT "\nMPU9250 is online..."; INTEGER.H' c
ELSE
  PRINT "\nFailed"; INTEGER.H' c
ENDIF
// Initialize the name of the registers
DIM WHOAMI, MPU9250ADRESS

// Set the address of the registers
MPU9250ADRESS = 0x68                              // MPU-9250 address
WHOAMI = 0x75                                     // WHO_I_AM register address

// Open I2C Peripheral
rc = I2cOpen(100000, 0, handle)
IF rc != 0 THEN
  PRINT "\nFailed to open I2C interface with error code "; INTEGER.H' rc
ELSE
  PRINT "\nI2C open success"
ENDIF

// Read the data from WHO_I_AM register
// to see if the sensor is working properly
rc = I2cReadReg8(MPU9250ADRESS, WHOAMI, c)

IF c == 0x71 THEN
  PRINT "\nMPU9250 is online..."; INTEGER.H' c
ELSE
  PRINT "\nFailed"; INTEGER.H' c
ENDIF
```

5. Software/Firmware

La prima demo che abbiamo fatto legge i dati della accelerazione dei registri del MPU attraverso il BL600 via I2C e li invia via Bluetooth al dispositivo di visualizzazione. Abb iamo utilizzato come dispositivo di visualizzazione il *Samsung Galaxy S4 mini* del Biolab dove già è caricata la *app Acceleration BLE Demo.*

Una volta che il hardware sia bene impostato, como viene descritto sopra, stiamo in condizioni di caricare il firmware sul BL600 per far girare la applicazione principale. Le due file che abbiamo fatto sono le seguenti:

- *$autorun$accelerometer.sb*: È il file principale, il equivalente al *main()* in C. Apre la porta I2C e fa la lettura dei registri di accelerazione del MPU-9250. Contiene delle lib rerie necessarie per il corretto funzionamento della applicazione.
- *custom.accelerometer.sblib*: È una libreria contenuta nel file principale. Gestisce la creazione del GATT Server e la forma dei attributi che inviamo via Bluetooth.

Per caricare il firmaware sul BL600 e visualizzare i dati della accelerazione sul smartph one:

- Accendere il modulo Bluetooth e aprire il *UW Terminal* nella cartella *BL600_&_MPU9250_DEMO*. Ricordare dobbiamo stare nella modalità interattiva (PIN 40 = 1; PIN 9 = 0).
- Caricare il firmaware *$autorun$accelerometer.sb* che si trova nella stessa cartella dell'*UW Terminal* (tasto destro, XCompile + Load).
- Spegniamo il modulo.
- Cambiamo a modalità *AutoRUN* impostando le connessioni: PIN 40 = 0; PIN 9 = 1.
- Apriamo la app Acceleration BLE Demo sul Samsung.
- Riaccendiamo il modulo e facciamo "Scan" sulla app del amrtphone. I dati della accelerazione cominceranno a arrivare sia all'*UW Terminal* sia al smartphone.

6. Riferimenti.

[1] InvenSense. *MPU-9250 Product Specification Revision* 1.0. 17 Gennaio 2014.

[2] http://es.wikipedia.org/wiki/MAX232.

[3] Texas Instruments. *MAX232x Dual EIA-232 Drivers/Receivers*. February 1989. Revised November 2014.

[4] Laird Technologies. *Single Mode Bluetooth Low Energy (BLE) Module. Hardware Integration Guide Version 2.2 (Part # BL600-SA, BL600-SC, BL600-ST, BL620-SA, BL620-SC, BL620-ST)*. 2014.

[5] Laird Technologies. BL600 smart BASIC Module. User Manual. Release 1.5.66.0. 28 Mar 2014 (Softdevice 6.0.0).

[6] Laird Technologies. UWTerminal *SmartBasic*. Quick Start Guide. v1.1. Document: CONN-QSG-UWTerminal_smartBasic_v1.0.

[7] www.ti.com/tool/CC2541-SENSORTAG-IBEACON-RD?keyMatch=Sensor-Tag&tisearch=Search-EN-Everything.

[8] InvenSense. *MPU-9250 Register Map and Descriptions Revision 1.4* Settembre 2013.

Referencias

[1] B. Fida, I. Bernabucci, D. Bibbo, S. Conforto, and M. Schmid. "*Varying behavior of different windows sizes on the classification of static and dynamic physical activities from a single accelerometer"*. Med Eng Phys 2015. 37(7):705-11.

[2] A. Protto. "*Design and development of a self-powered inertial/magnetic node for motor activities classification and qualification"*. *Biolab3* PhD Project. Personal communication.

[3] Specification of the Bluetooth System. Covered Core Package Version: 4.0, Junio 2010.

[4] K. Townsend, C. Cufi, Akiba, and R. Davidson. **Getting Started with Bluetooth Low Energy**. O'Reilly, 2014.

[5] https://epxx.co/artigos/bluetooth_gatt.php
(accessed 25 June 2009)

[6] www.invensense.com/products/motion-tracking/9-axis/
(accessed 25 June 2009)

[7] www.lairdtech.com/products/bl600-series
(accessed 25 June 2009)

[8] InvenSense. *MPU-9250 Product Specification Revision* 1.0. 17 Enero de 2014.

[9] Laird Technologies. *Single Mode Bluetooth Low Energy (BLE) Module. Hardware Integration Guide Version 2.2 (Part # BL600-SA, BL600-SC, BL600-ST, BL620-SA, BL620-SC, BL620-ST)*. 2014.

[10] http://es.wikipedia.org/wiki/MAX232
(accessed 25 June 2009)

[11] Texas Instruments. *MAX232x Dual EIA-232 Drivers/Receivers*. February 1989. Revised November 2014.

[12] Asahi Kasei Microdevices Corporation, AKM. Datasheet AK8963. *3-Axis Electronic Compass*. 2013. MS1356-E-02.

[13] InvenSense. *MPU-9250 Register Map and Descriptions Revision 1.4* Septiembre de 2013.

[14] Laird Technologies. BL600 smartBASIC Module. User Manual. Release 1.5.70.0. 28 Mar 2014 (Softdevice 6.0.0).

[15] Laird Technologies. UWTerminal *SmartBasic*. Quick Start Guide. v1.1. Document: CONN-QSG-UWTerminal_smartBasic_v1.0.

[16] https://developer.android.com/sdk/index.html
(accessed 25 June 2009)

[17] J. Friesen. **Learn Java for Android Development**. Second Edition. Apress. 2013

[18] https://developer.android.com/guide/topics/connectivity/bluetooth-le.html (accessed 25 June 2009)

[19] Q.-D. Ho, and T. Le-Ngoc. "*iVS: An intelligent end-to-end vital sign capture platform using smartphones.*" Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th. IEEE, 2014.

[20] C.Gomez, J.Oller, and J.Paradells. "*Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology.*" Sensors 12.9 (2012): 11734-11753.

[21] H. Nakanishi, N. Nishigaki, K. Tachibana, and T. Shinagawa. *WT210/WT230 Digital Power Meters*. Yokogawa.

[22] Primary Lithium Cells $LiMnO_2$. Sales program and technical handbook. Varta.

[23] Laird Technologies. *Bluetooth Low Energy Development Kit (DVK). User Manual. Version 1.1. Version 5 of DVK-BL600-SA*. September 2014