POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Biomedica

Orientamento Strumentazione Biomedica

Development of a wearable actigraph with Bluetooth Low Energy link and implementation of a dedicated Android App

Relatore

Marco Knaflitz

Studente

Alberto Marchesa

Correlatore

Daniele Fortunato

Index

Introduction 1

1 Device overview 2

2 BGM121 4

2.1 BLE fundamentals 4

2.1.1 Protocol Stack 5

2.1.2 Communication 8

2.2 System models 12

2.3 Simplicity Studio 13

2.3.1 GATT definition 13

2.3.2 Application code 14

2.4 Project outline 17

3 SAM E70 19

3.1 Parallel Input/Output Controller (PIO) 20

3.1.1 Hardware 20

3.1.2 User interface 20

3.2 Universal Asynchronous Receiver Transmitter (UART) 23

3.2.1 Hardware 23

3.2.2 User Interface 26

4 Firmware 28

4.1 Block diagram 28

4.2 Flow chart 29

4.2.1 UART Module 29

4.2.2 UART interrupt 32
4.2.3 App handle event 33
5 Android App 34
5.1 App fundamentals 34
5.1.1 Components 34
5.1.2 Intent 35
5.1.3 Resources 35
5.1.4 Manifest file 36
5.2 Android Studio 37
5.2.1 Project structure 37
5.2.2 Layout editor 38
5.3 Actigraph App 39
5.3.1 Manifest file 39
5.3.2 Application codes 40
6 Conclusion 43
References 44

Introduction

An actigraph is a medical device the size of a watch, worn on upper or lower limb, which contains at least an accelerometer to record movements the unit undergoes. Until early 2000's, actigraphy was used mainly as a method to assess insomnia, circadian rhythm disorders or excessive sleepiness [1]. Since then, progresses made on sensor's dimension and consumption, as well as data analysis algorithms, increased the number of application fields. Human Activity Recognition (HAR) is an active research area that classifies user activities from inertial data and it finds application in medicine, athletics, lifestyle monitoring and human/computer interaction. All modern consumer electronics, like smartphones, smartwatches and personal activity trackers, integrates most advanced Magneto-Inertial Measurement Units (MIMUs), therefore they represents a good platform for HAR application. Nevertheless, devices available nowadays on the market have some important limitations [2]. First, they classify only a small subset of user activities, mostly outdoor sport activities, while a considerable amount of our life is lived indoor, performing simple activities. This is true especially for the elderly and subjects who need cardiac or neuromuscular rehabilitation, given the importance of physical activity monitoring**.** Secondly, devices just acquire signals: to process data and classify activities, an external software is needed. Devices from ActiGraph company represents a good solution to the first limitation. These actigraphs have the purpose of monitoring activities in the medical field, especially for advanced age and sick subjects, providing accurate information and parameters concerning basic daily activities. However, an external software to classify data is still needed. With this background, in 2017, Politecnico di Torino, in collaboration with Medical Technology s.r.l., developed an actigraph with an on-board software for daily activities recognition. Afterwards, in 2018, this device was modified, by Fabio Bolognesi's thesis project, in order to improve computing skills, data storage and power consumption. The aim of this thesis now is to take a step forward on this project by installing a Bluetooth Low Energy (BLE) module in the device's hardware, modifying the device's firmware and developing a dedicated Android App to control the acquisition and display data with a smartphone. Such upgrade makes this device a good candidate for many others application fields because it is lightweight, easy-to-wear, cost-effective, computation-reliable and basically, can be used everywhere as it needs no external PC nor internet connection. To better understand how this novel device works, Bluetooth Low Energy technology and Android App main principles will be introduced in the following chapters, as well as changes made in device's firmware.

1 Device overview

The block diagram of this new actigraph is reported in Figure 1. The device consists of a SAME70 family microcontroller, a LSM9DS1 MIMU with a 3-axis accelerometer, a 3-axis gyroscope and a 3-axis magnetometer for the magneto-inertial signal acquisition. There is also an SD card for saving raw data and recognized activities and a BGM121 SiP Bluetooth module for wireless communication. Data can be transferred or by USB 2.0 port or by BGM121 SiP Bluetooth module for wireless communication.

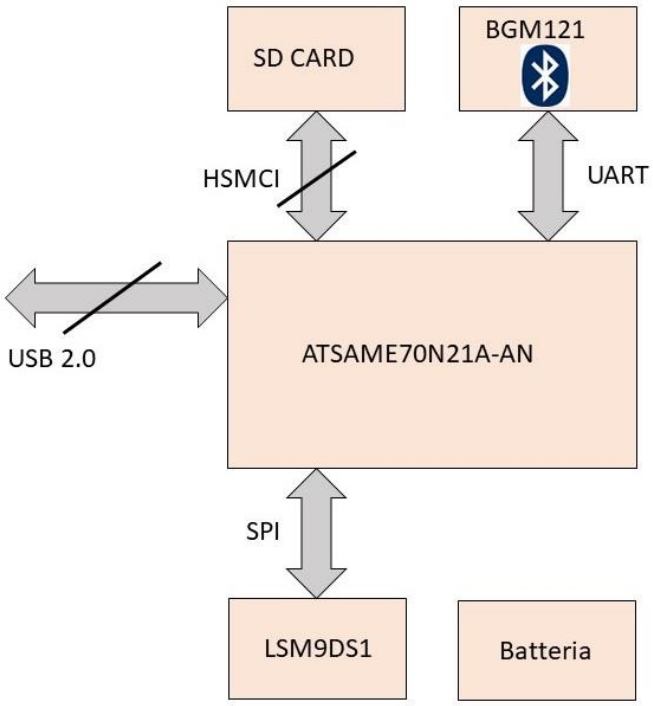


Figure 1- Actigraph Hardware Block Diagram

The device is a two-state machine: one for the data acquisition, data saving and activity classification, and the other for the results transmission (Figure 2). The on-board software is able to recognize 5 activities: walking, upright standing, resting (both sitting and lying), ascending and descending stairs. The classification is obtained through a decision tree (DT) algorithm developed by the Politecnico di Torino, stored inside the device. DT algorithm has been implemented in the firmware as a set of nested "if- then" statements. This property determines a strong reduction of the computational costs, reducing the power requirements of the system [3].

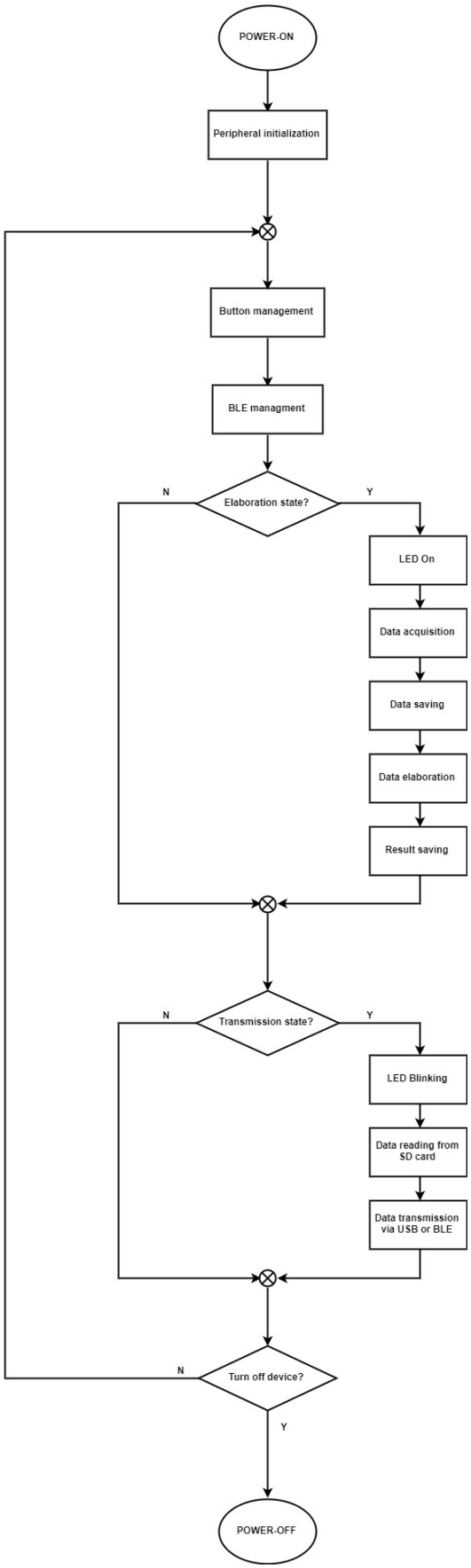


Figure 2- Block diagram of the two state wireless actigraph

2 BGM121

In order to get familiar with the BGM module, *SLWSTK6101C Wireless Starter Kit* of *Silicon Labs* has been used (Figure 3). It contains:

- Main board;
- Radio board BRD4300A with BGM111;
- Radio board BRD4302A with BGM121;
- Expansion board;
- USB cable.

Figure 3-SLWSTK6101C Wireless Starter Kit from Silicon Labs

2.1 BLE fundamentals

In 2001 *Nokia* started developing a wireless technology meant to be low energy and cost-effective [5], then launched on the market in 2006 by the name of *Wibree* [6]. In 2010, Bluetooth Special Interested Group (SIG) included this technology as part of the Bluetooth® version 4.0 Core Specification as *Bluetooth Low Energy (BLE)* [7]. To better understand how BLE works, the architecture and basic principles of BLE will be introduced in the following.

2.1.1 Protocol Stack

The protocol stack consists of three main blocks: Application, Host and Controller. The App is the direct interface with the user and helps the interoperability among devices of different manufacturers. The Host represents the software section of the protocol stack, while the Controller is the hardware ones (Figure 4).

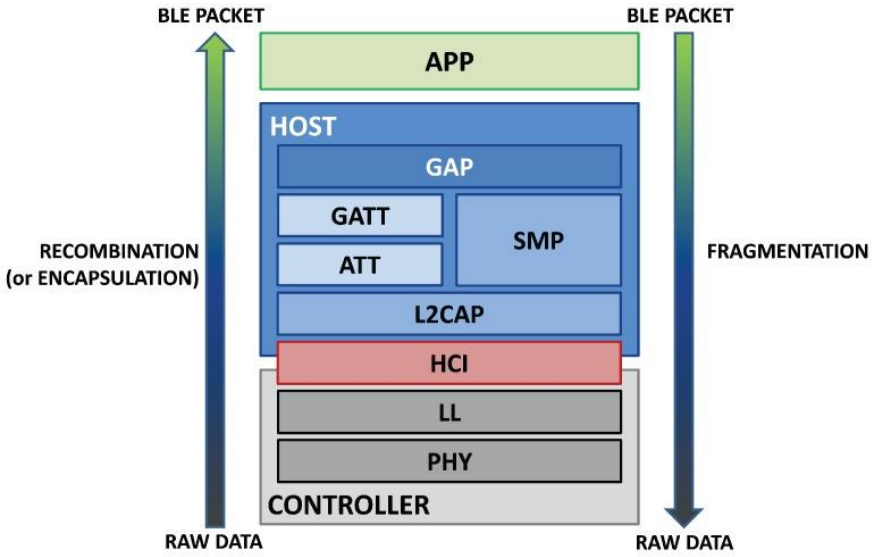


Figure 4-BLE protocol stack [8]

Physical Layer (PHY)

LE radio works in the unlicensed 2.4 GHz ISM (Industrial Scientific Medical) band. The BLE band goes from 2402 MHz to 2480 MHz and it is divided in 40 channel separated with 2 MHz. Three channels (37-39) are the primary advertisement channels, whereas the remaining ones (0-36) are secondary advertising channels and can be used also as data channels once connection is established [9]. The physical layer data rate is 1Mpbs or 2 Mpbs (later introduced in Bluetooth specification 5.0).

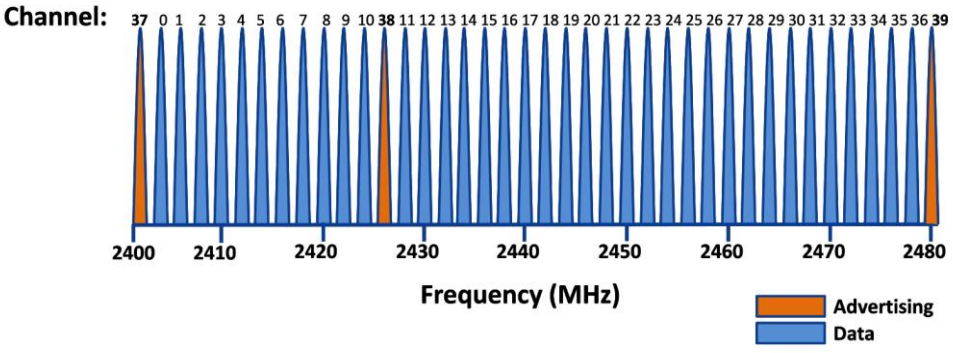


Figure 5- BLE frequency channels [8]

Link Layer (LL)

It is a combination of both software and hardware intended to avoid an overload of the CPU. The HW section provides a first level of control and data structure over the raw radio operations. The SW section defines the type of communication by managing the link state of the radio (standby, advertise, scan, initiate, connect and synchronise). Furthermore, it defines the four roles a device can assume: *Master*, *Slave, Advertiser* and *Scanner*. Further information will be found in 2.1.2 section.

Host Controller Interface (HCI)

It defines a set of commands and events for Controller and Host to communicate with each other. It hides the real time requirement of the former from the complexity but less timing-stringent protocols of the latter [10].

Logical Link Control and Adaptation Protocol (L2CAP)

A protocol that merges raw data into a BLE packet (encapsulation) and vice versa (fragmentation).

Security Manager Protocol (SMP)

It provides the ability to create and exchange security keys for a safe connection and it defines two roles during the establishment of a connection: *Initiator* and *Responder*. Such roles correspond respectively to *Master* and *Slave* defined in LL. Further information can be found in 2.1.2 section.

The following layers are fundamental, as they represents the entry point for the application to interact with the protocol stack.

Attribute Protocol (ATT)

ATT defines the roles of *Client* and *Server* for a given device. The latter stores data, while the former requests it (Figure 6). These data are called attributes and ATT is the protocol to access them.

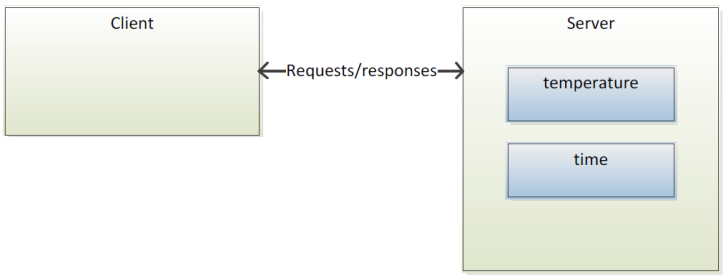


Figure 6-Device's roles defined by ATT protocol [11].

Each attribute is characterized by a Handle, an UUID, a set of permission and the value. The Handle is a 16-bit value assigned by each server to its own attributes, in order to allow a client to reference it. The Universal Unique Identifier (UUID) identifies the attribute type, that is the kind of data contained in the value. Two types of UUID are used: a globally unique 16-bit UUID defined by Bluetooth SIG in the core specification and a manufacturer-specific 128-bit UUID. Permission specifies which ATT operations (read, write, indicate, notify) can be executed on the attribute and the security requirements (encryption, authentication, authorization) for that operation. Permissions of a given attribute are defined by a higher layer specification, and are not discoverable using the Attribute protocol [12].

Generic Attribute Profile (GATT)

It represents the cornerstone of BLE data transfer as it defines how data is organized and exchanged between applications. GATT encapsulates the ATT and uses its roles (client, server) to transfer attributes. Furthermore, it defines a structure to organize attributes in a reusable and practical way, a common framework for all GATT-based profiles (Figure 7).

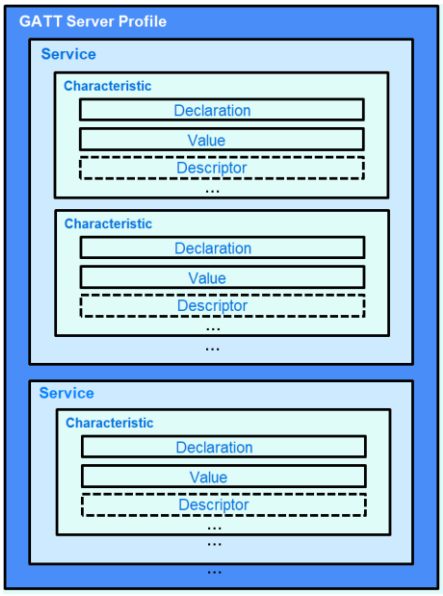


Figure 7-GATT Data Hierarchy. Adapted from [13]

The profile represents the top level of the hierarchy and it is composed of services and characteristics. Services are containers that group related attributes (characteristics) to achieve a particular function or feature. In turn, characteristics include at least two attributes: the declaration, which provides metadata related to user data, and the value, which is the actual user data. Additionally, the third ones is the descriptor, an attribute for additional info and proprieties that defines how the value can be handled by the client. An example of data structure and communication scheme of a Heart Rate

Profile is given in Figure 8. For ease of viewing, just Heart Rate service is illustrated on the left side of Figure 8.

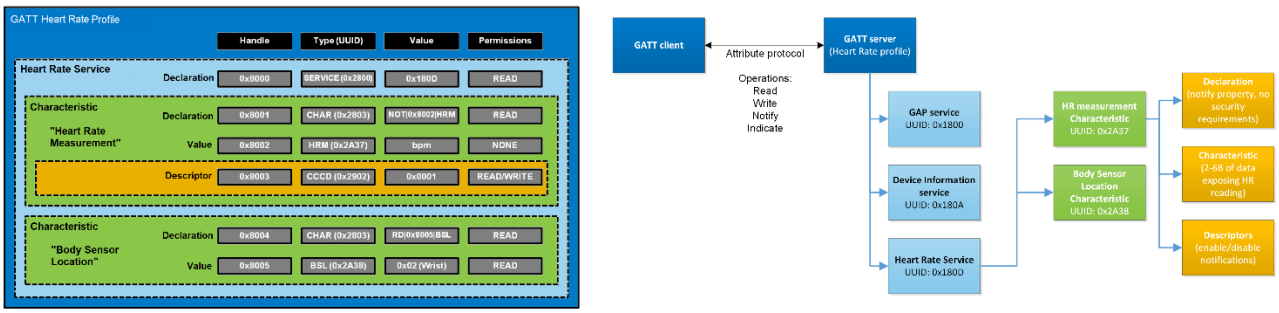


Figure 8-Data structure and communication scheme of a Heart Rate Profile

Generic Access Profile (GAP)

GAP is on top of the stack, as it controls how devices interoperate with each other at low level, outside the protocol stack. It provides access to the link layer operations (mentioned above in the LL section) and it defines roles, modes and procedure to regulate the communication. Roles defined by GAP allow devices to have a radio that either transmit or receive or both. The roles are: *Broadcaster*, *Observer, Peripheral* and *Central* (further information can be found in the next section). A device can support more than one role, but only one role at a time can be assumed.

2.1.2 Communication

While classic version of Bluetooth, called BR/EDR (Bit Rate/ Enhanced Data Rate), is characterized by a short-range and continuous wireless connection, BLE establishes short bursts of long-range radio connections [11]. BLE devices communicate in two different modalities: Broadcasting and Connection.

Broadcasting

Broadcasting is a one-way type communication; it is the only way to transmit data to several peers at the same time. Although broadcasting results the fastest communication mechanism, there is no security for data. For this reason, it is not suitable for certain applications. In Broadcasting, devices can assume two roles defined in GAP:

- *Broadcaster*: periodically sends advertisement packets, it uses the *Advertiser* role of LL.
- *Observer*: periodically scan for incoming advertisement packets, it uses the *Scanner* role of LL.

The two scanning modes are shown in Figure 9. In passive scanning mode, the *Scanner* just listen for incoming advertisement packets, while in active mode, once the packet has been received, it sends a scan request packet to the *Advertiser* in order to receive further information. The scan response doubles the data payload of the advertisement event, but no data are sent from the *Scanner* to the *Advertiser* (one-way communication).

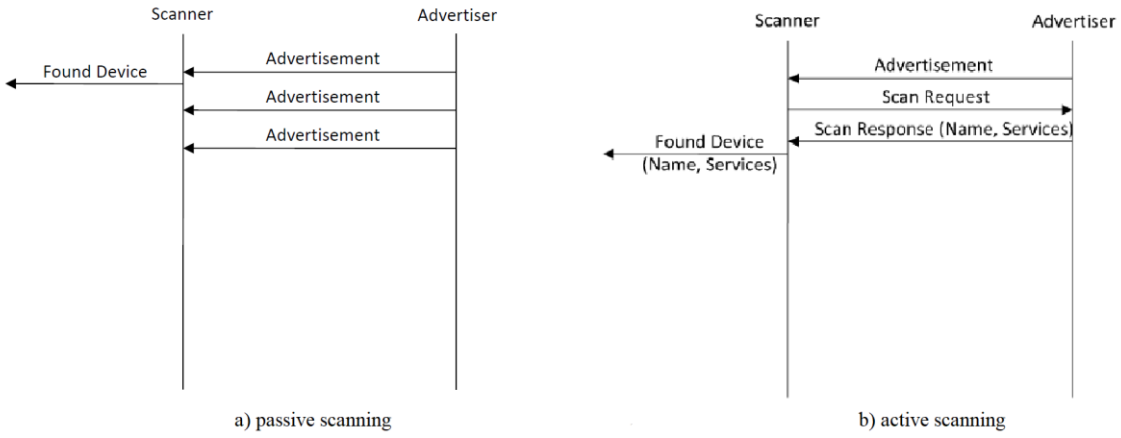


Figure 9- Scanning modes. Adapted from [11]

The *Broadcaster* sends out advertisement packets simultaneously from the 3 advertisement channels (37-39) at a given *advInterval*, while the *Observer* switches the advertisement channel each *scanInterval* and listens for incoming advertisement for a given *scanWindow* (Figure 10). The lower the *advInterval*, the higher the probability of a packet to be received, although power consumption rises.

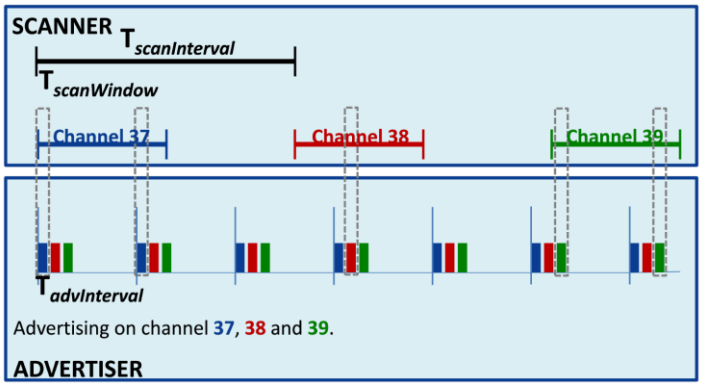


Figure 10- Time parameters for broadcasting mode [8]

Connection

Connection is a bi-directional, private and secure communication between two devices. In this case the roles defined by GAP are:

- *Central*: scans for connectable advertisement packets and initiates the connection. It uses the *Master* role of LL.
- *Peripheral*: sends connectable advertising packets and accepts connections. It uses the *Slave* role of LL.

As shown in Figure 11, connection consists of several steps.

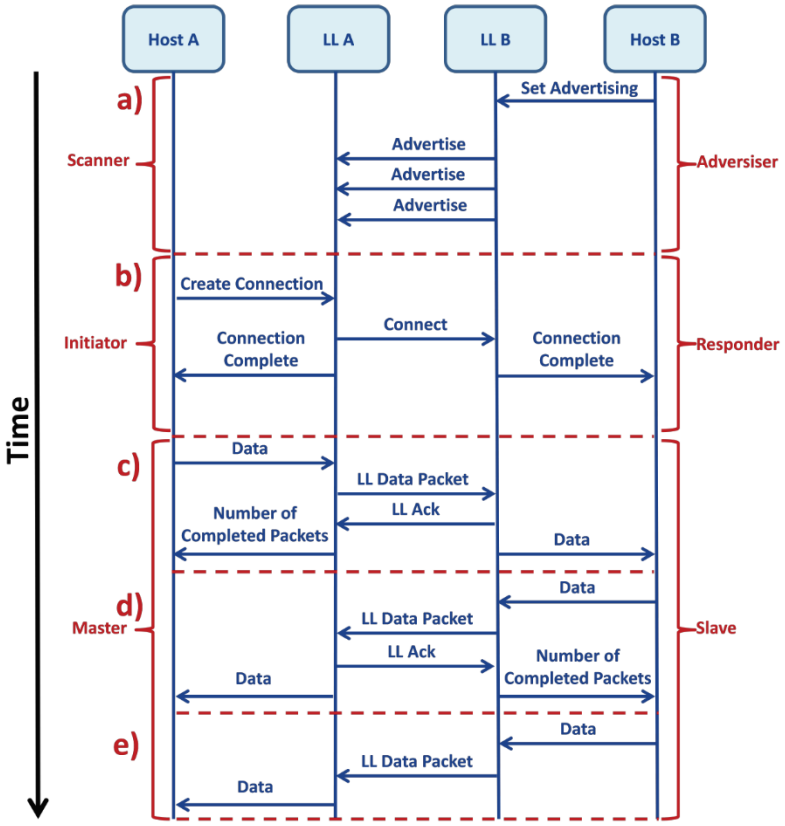


Figure 11- Connection between two BLE devices. a) Broadcasting mode, b) Connection establishment, c) Data transfer from master to slave in a round-trip communication, d) Data transfer from slave to master in a round-trip communication, e) Data transfer from slave to master in a one-way communication

At first, devices are always working in broadcasting mode: the *Advertiser* sends a message to its LL in order to sends out connectable advertisement packets. When the *Scanner* receives these packets, it is ready to initiate the connection (Figure 11-a). The *Initiator* sends a message to create the connection, firstly to its LL and then to the *Responder*. Once the connection is established, the two LLs send a confirmation message to their own Host layer (Figure 11-b). Devices can communicate in two ways when connected: in a round-trip communication, ACK packets are sent back to the *Host* layer to report if data were transmitted correctly (Figure 11-c/d); in a one-way communication, no ACK packets are involved (Figure 11-e). Several time parameters are used to describe the connection: a *connectionEvent* is the amount of time in which devices exchange data packets, while *Radio Idle* represents the time in which the communication is off. The time between two consecutive connection events is the *connectionInterval* and the *connectionSupervisionTimeout* represents the maximum time without receiving two packets before the connection is lost. Figure 12 shows time parameters of a round-trip communication (ACK is transmitted right after the data packet).

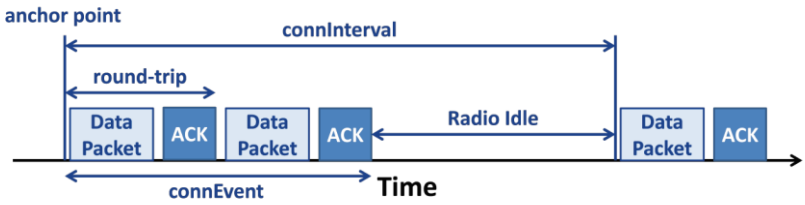


Figure 12- Time parameters in a round-trip communication [8]

2.2 System models

The *BGM121 Blue Gecko Bluetooth*® SiP Module of *Silicon Labs* allows two system models: System on Chip (SoC) and Network Co-Processor (NCP).

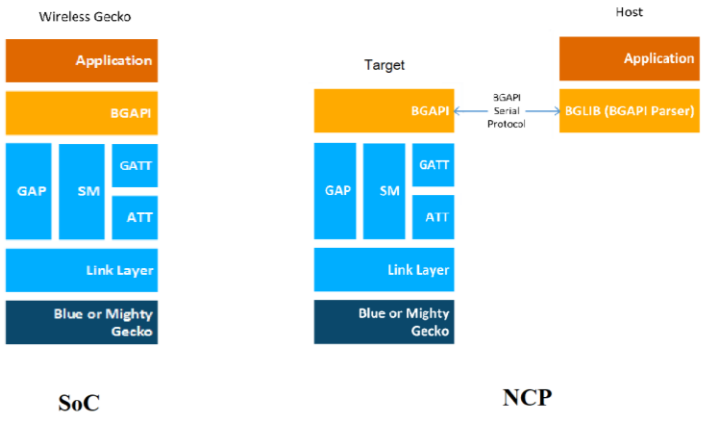


Figure 13- SoC vs NCP System Models [14].

As shown in Figure 13, application, host and controller codes run on the same Wireless MCU in a SoC system, while in NCP mode, BLE stack is split in two blocks: Host and Target. The *Host* represent an external MCU where the Application runs, while host and controller runs on the BMG121 MCU, that is the *Target*. Host and Target communicates via UART serial interface and the communication is defined by BGAPI, a custom binary protocol from *Silicon Labs*. Such serial protocol is lightweight and carries the BGAPI commands from the Host to the Bluetooth stack and responses and events from the Bluetooth stack back to the Host. The Host code is developed on top of BGLib, an ANSI C reference implementation of the BGAPI binary protocol (Figure 14).

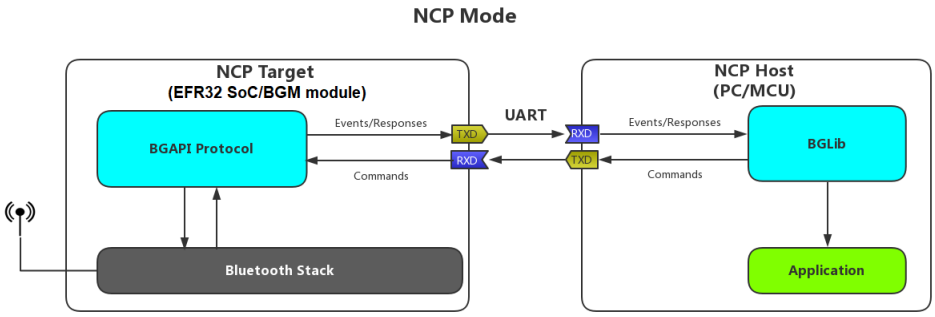


Figure 14-NCP mode. Adapted from [15].

2.3 Simplicity Studio

Simplicity Studio is the Eclipse-based IDE of *Silicon Labs* to program the BGM module. It includes powerful suite of tools to develop, debug and analyse applications. It comes with some precompiled examples to get started with the BGM module, avoiding development from scratch. Project build flow will be now shown.

2.3.1 GATT definition

When developing an application, the starting point is defining its GATT. To do so, the first option is to create an XML file and then convert it into .c and .h files. The second option is the GATT Editor, a strong graphical tool for designing the GATT. (Figure 15).

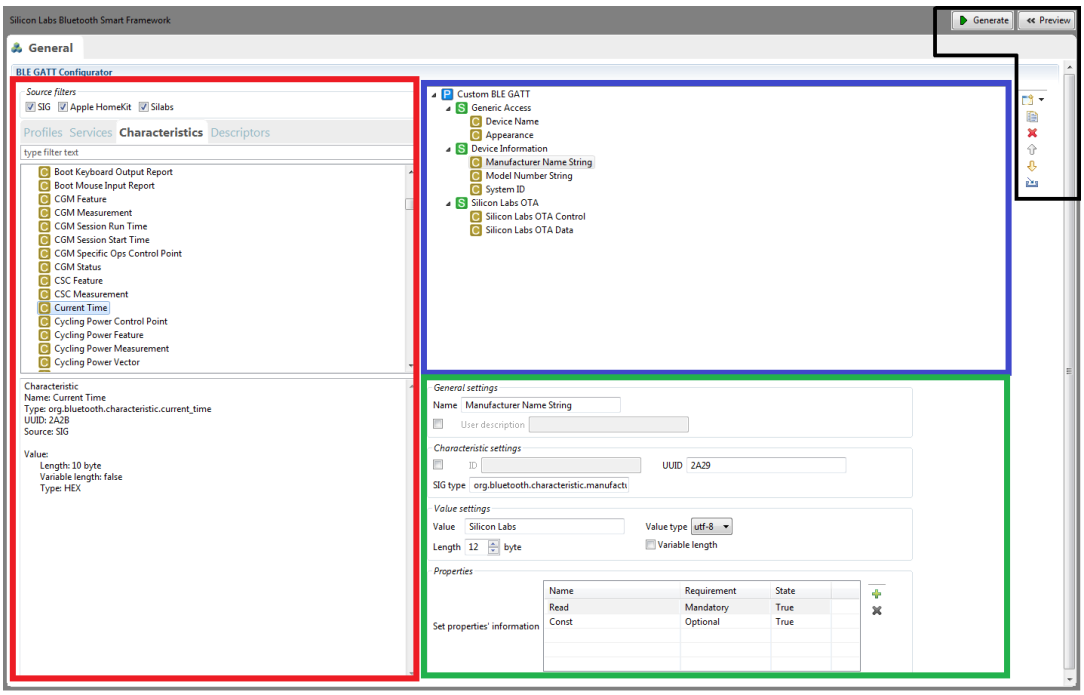


Figure 15- GATT Editor

A list of predefined Profiles/Services/Characteristics/Descriptors is shown in the left panel (red section). These items can be added to the current GATT database (blue section) by simply drag and drop the desired ones. New items can be created too from the tools menu (black section), despite previous item, these ones will be characterized by a 128-bit UUID (Protocol Stack in ATT section) An options menu is provided in the lower right panel (green section) allowing the user to modify name, value, ID and properties of the items. Once definition has been completed, the GATT database is generated and gatt.xml, gatt_db.c and gatt_db.h files are created. In gatt_db.h, constant values are assigned to each characteristic and service id (handles) to reference it in the application.

2.3.2 Application code

Since the Bluetooth stack is an event-driven architecture, the core of every application code, both SoC and NCP mode, is to handle events coming from the stack and to send command to the stack.

SoC code

In SoC mode, a basic application is made of an event listener, followed by an event handler. The BGM121 listens for events in a main while loop, in a block or non-blocking fashion (Figure 16). Respectively, the *gecko_wait_event()* function waits for events coming from the Bluetooth stack and blocks until an event is received, while the *gecko_peek_event()* function checks for events and return NULL if no events are coming from the stack.

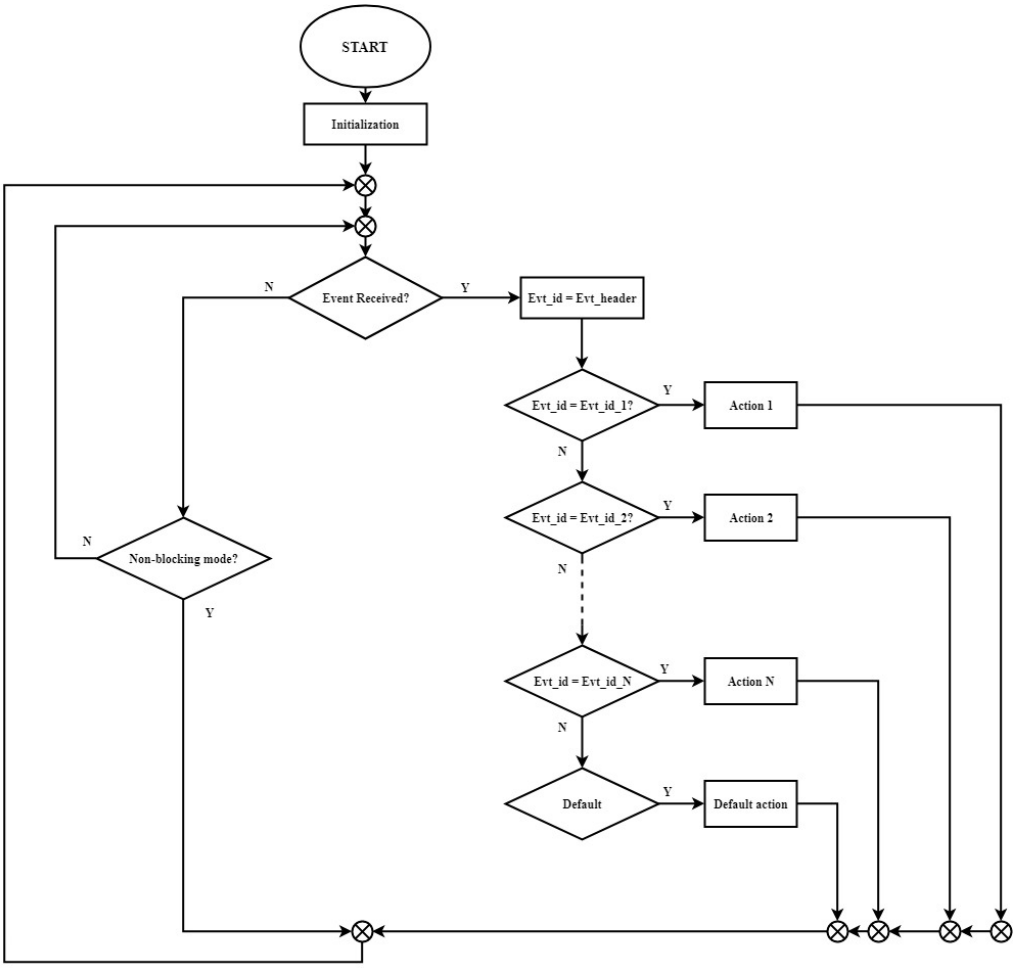


Figure 16- Block diagram of a basic SoC application code

These functions return a pointer to a gecko_cmd_packet structure holding the received event. The event handler is a switch structure where each case represents an event id and the input is the message header of the received event.

Every message is composed as Table 1. The first fourth bytes represent the header, the basic information of the message. The first byte defines the message type: 0x20 for command and response and 0xA0 for events, the second byte represent the payload length. The third byte defines the class: message might be used, for example, to manage connection (GAP class), write a characteristic (GATT server class) or to access and query the BGM module (System class). The fourth byte is used to distinguish between all the message within one class. All the other bytes represents the message payload and its length is variable. The full list of events, commands and responses can be found in the *Bluetooth Software API Reference Manual.*

Byte	Values	Name	Description
0	0x20: command 0x20: response 0xA0: events	hilen	Message type
1	0x00 – 0xFF	lolen	Payload length
2	0x00 – 0xFF	class	Message class
3	0x00 – 0xFF	method	Message id
4 -255	Message specific	Message specific	Payload

Table 1- BGAPI packet structure

NCP code (Target side)

When in NCP mode, the application runs on an external MCU (Host) while the BMG is used basically as an antenna (Target). The communication takes place through UART serial link: the Host sends commands to the Target, while the Target sends responses and events to the Host. In Figure 17 is shown the block diagram of a basic code which runs on the Target. The main loop starts with forwarding the commands from the UART Rx queue to the stack in order to be handled. Then the BGM listens for events coming from the stack in a non-blocking way. When an event is received, if it is not handled locally, it goes to the UART Tx queue in order to be ready to be transmitted. If no more events are received, the target sends out the enqueued events in order to be handled by the host MCU.

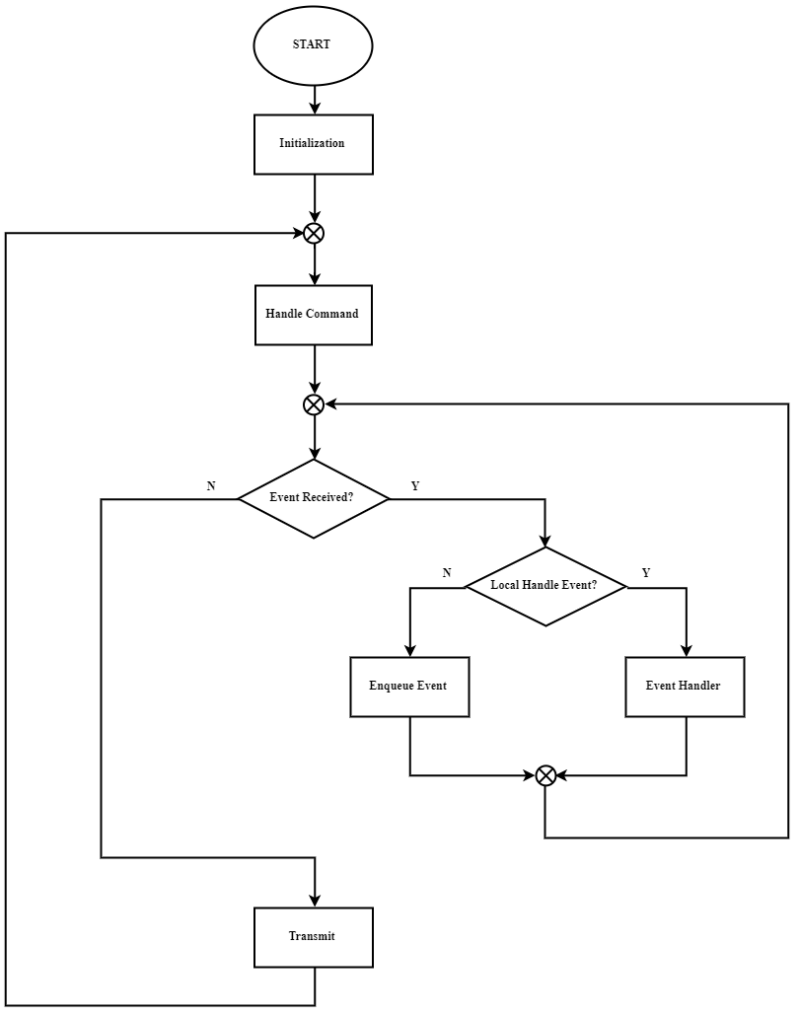


Figure 17- Block diagram of a basic NCP Target application code

2.4 Project outline

Since the aim of this project is to combine the strong computational feature of the actigraph's MCU with the BLE functionality of the Wireless MCU, the BGM121 will be used in NCP mode. In this way the new actigraph is composed of SAME70 and BMG121, communicating each other through a UART serial link. From the point of view of storage and data flow, the actigraph plays the role of Server while the smartphone the Client ones (both defined in ATT protocol) and they exchange data with a request/response pattern (Figure 18).

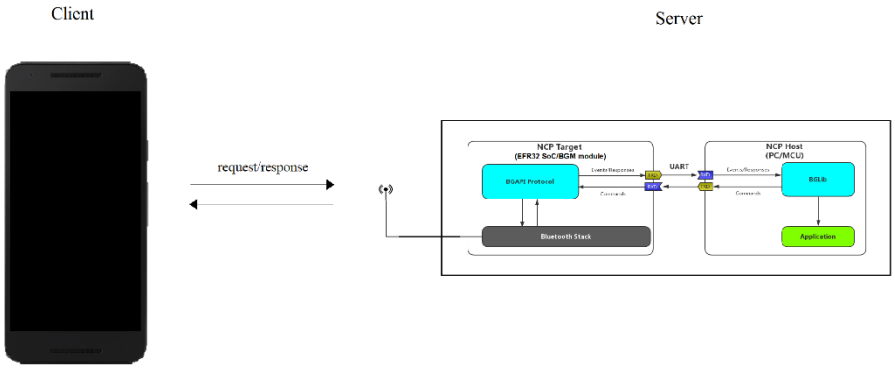


Figure 18- Project data transfer topology

From the point of view of the communication management, the actigraph plays the role of the Slave while the smartphone plays the role of the Master. The roles played by the two devices before the connection is established are defined in Table 2 and explained in section 2.1.2

Communication	Smartphone	Actigraph
Broadcasting	Scanner	Advertiser
Connection Establishment	Initiator	Responder
Connection	Master	Slave

Table 2- Roles played by smartphone and actigraph during a connection

The custom GATT Server profile defined for this project is shown in Figure 19. It consists of three services:

- Generic Access service gives generic information about the device, its characteristics are readonly;
- Acquisition service contains the I/O characteristic to start and stop the acquisition and the Data characteristic to save acquisition result, both are readable and writable;
- User Data service is used to save personal info given by the patient from the android application, its characteristics are readable and writable.

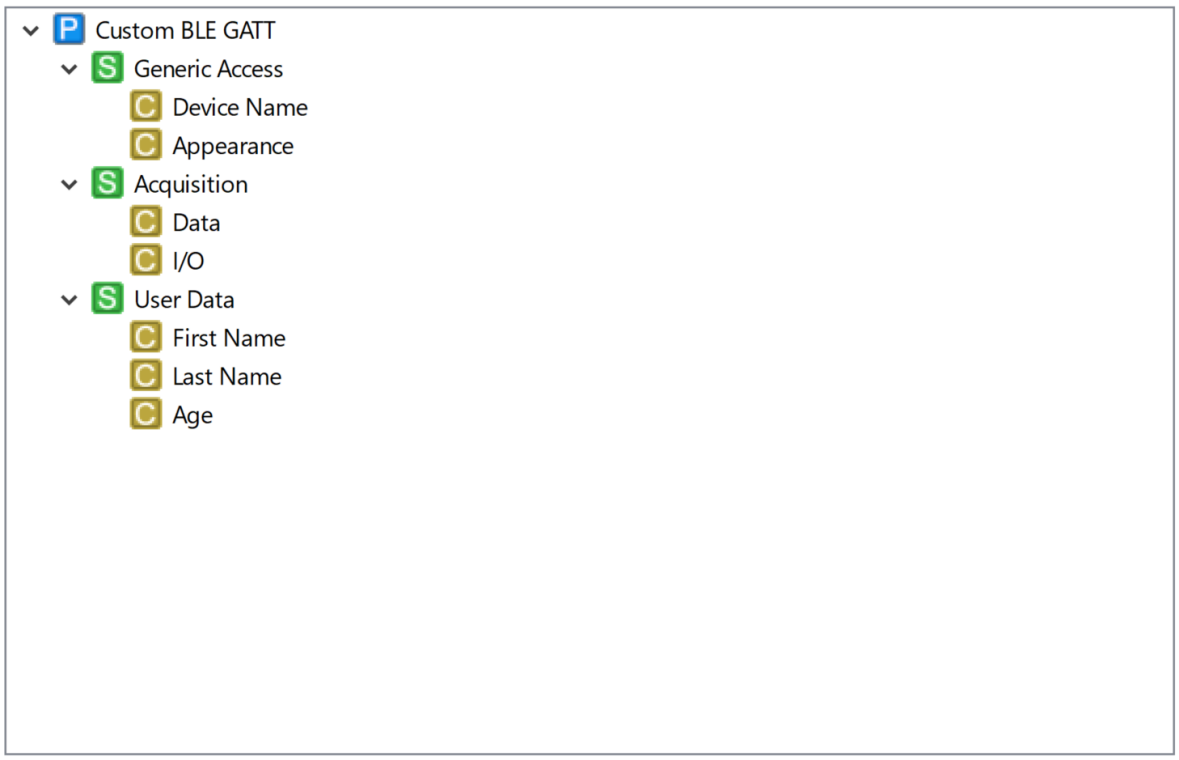


Figure 19- Custom GATT Server Profile

3 SAM E70

SAM E70 is the Host MCU of the NCP system, described in the previous chapter. It is a 32-bit microcontroller based on ARM Core cortex M7 by Atmel® Corporation. In order to establish a UART serial communication, the Atmel® SAM E70 XPL evaluation board has been used (Figure 20).

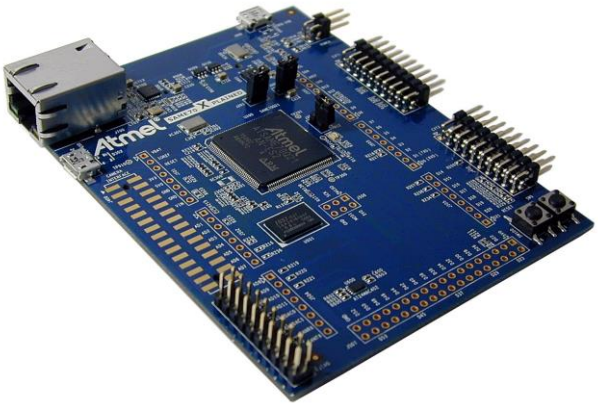


Figure 20- Atmel® SAM E70 XPL

To connect the LSM9DS1 to the SAME70 XPL evaluation board, the PROTO1 Xplained pro extension board sensor has been used (Figure 21).

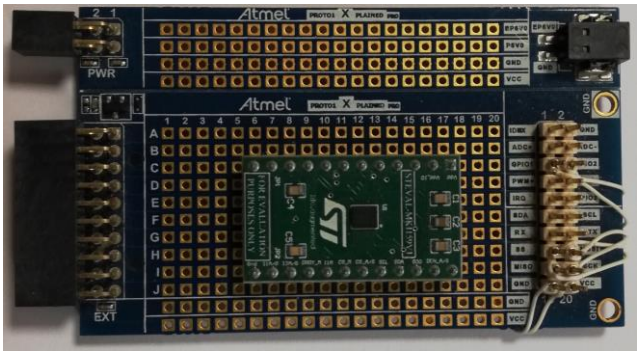


Figure 21- PROTO1 Xplained Pro with LSM9DS1

3.1 Parallel Input/Output Controller (PIO)

PIO is the module that manages up to 32 fully programmable I/O lines; each of these can be used as a general-purpose I/O or be assigned to one of the four embedded peripheral (A, B, C, D).

3.1.1 Hardware

The MCU provides five PIO controller: PIOA with 32 fully programmable I/O lines, PIOB with 12 fully programmable I/O lines, PIOC with 32 fully programmable I/O lines, PIOD with 32 fully programmable I/O lines and PIOE with 6 fully programmable I/O lines. PIO controller communicates with several modules (Figure 22).

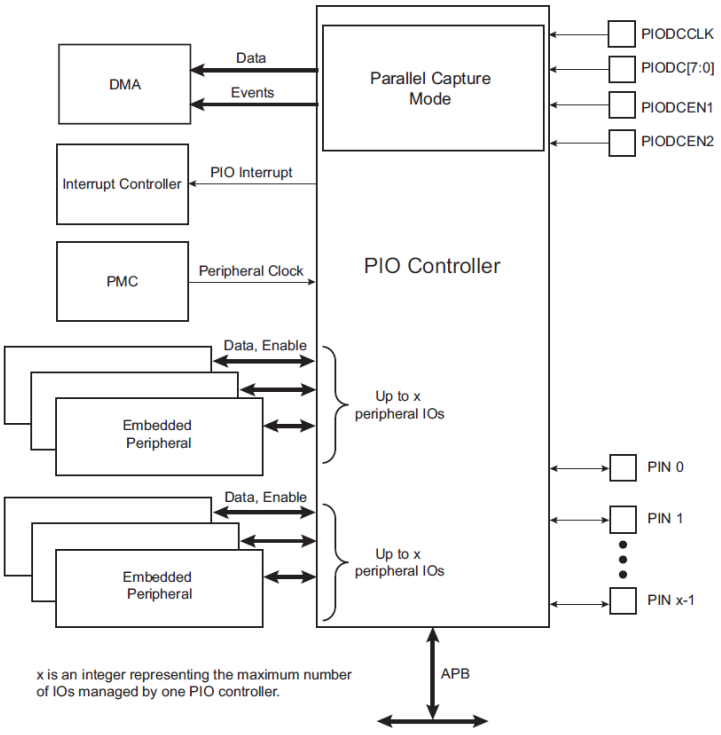


Figure 22- PIOx block diagram [16].

3.1.2 User interface

As first step, pins control must be defined. PIO_PER and PIO_PDR are the registers to enable or disable the PIO from controlling the pins respectively. These registers have 32 bit corresponding to the 32 I/O lines of each PIO controller (A, B, C, D, E). Setting certain bit of PIO_PER enables the PIO control, disabling the peripheral control of the corresponding pin. Vice versa writing 1 in the bits of PIO_PDR (Figure 23).

Name: PIO_PER
Offset: 0x0000
Property: Write-only

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

Bit	31	30	29	28	27	26	25	24
	P31	P30	P29	P28	P27	P26	P25	P24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	P23	P22	P21	P20	P19	P18	P17	P16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	P15	P14	P13	P12	P11	P10	P9	P8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
	P7	P6	P5	P4	P3	P2	P1	P0
Access								
Reset								

Bits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 – P PIO Enable

Value	Description
0	No effect.
1	Enables the PIO to control the corresponding pin (disables peripheral control of the pin).

Name: PIO_PDR
Offset: 0x0004
Property: Write-only

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

Bit	31	30	29	28	27	26	25	24
	P31	P30	P29	P28	P27	P26	P25	P24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	P23	P22	P21	P20	P19	P18	P17	P16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	P15	P14	P13	P12	P11	P10	P9	P8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
	P7	P6	P5	P4	P3	P2	P1	P0
Access								
Reset								

Bits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 – P PIO Disable

Value	Description
0	No effect.
1	Disables the PIO from controlling the corresponding pin (enables peripheral control of the pin).

Figure 23- PIOx Enable and Disable Register. Adapted from [16].

PIO_PER and PIO_PDR are write-only registers, to know the status of the I/O lines the status register must be check. PIO_PSR bits are set to 0 if peripheral control is enabled, while I s set to 1 if PIO control is enabled (Figure 24).

Name: PIO_PSR
Offset: 0x0008
Property: Read-only

Bit	31	30	29	28	27	26	25	24
	P31	P30	P29	P28	P27	P26	P25	P24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	P23	P22	P21	P20	P19	P18	P17	P16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	P15	P14	P13	P12	P11	P10	P9	P8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
	P7	P6	P5	P4	P3	P2	P1	P0
Access								
Reset								

Bits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 – P PIO Status

Value	Description
0	PIO is inactive on the corresponding I/O line (peripheral is active).
1	PIO is active on the corresponding I/O line (peripheral is inactive).

Figure 24- PIOx Status Register [16].

When a pin is disable from PIO control, it must be assigned to one of the embedded peripherals (A, B, C, D) because each pin can assume a different functionality according to the peripheral. PIO_ABCDSR1 and PIO_ABCDSR2 are the register for this task; Table 3 shows how the selection is achieved.

PIO_ABCDSR1 Pinx	PIO_ABCDSR2 Pinx	Peripheral selected
0	0	A
1	0	B
0	1	C
1	1	D

Table 3- PIO_ABCDSR1 and PIO_ANCDSR2 Peripheral selection

3.2 Universal Asynchronous Receiver Transmitter (UART)

UART is the module that provides a universal asynchronous serial link. In this project UART1 will be used for the serial communication between the MCU (Target) and the BGM121 module (Host).

3.2.1 Hardware

The UART operates in Asynchronous mode only and supports only 8-bit character handling (with parity). It's made up of a receiver and a transmitter that operate independently, and a common baud rate generator and it has no clock pin (Figure 25).

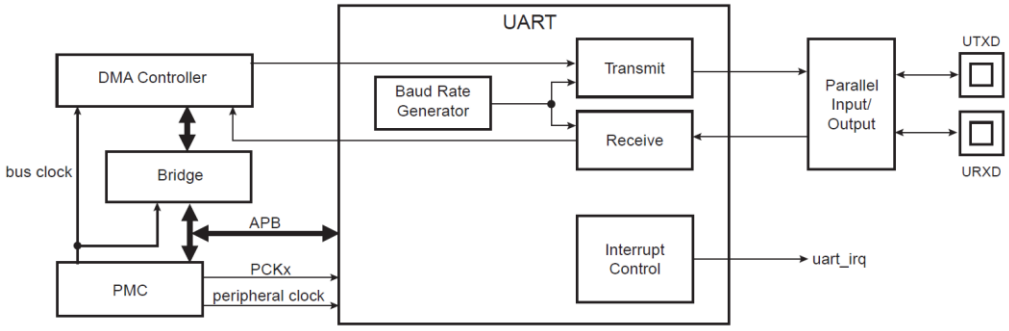


Figure 25- UARTx Block Diagram [16]

The baud rate generator define the bit period to both receiver and transmitter. Baud rate is essential in an asynchronous communication as the transmitter send data to the receiver without a clock signal, then they must agree on timing parameters in advance. In SAME70 baud rate is calculated as Equation 1, where the Source Clock could be the peripheral clock or a Programmable Clock Output (PCKx) and CD is the Clock Divisor.

$$BD = \frac{SOURCE\ CLOCK}{16 * CD}$$

Equation 1- Baud rate formula

The receiver detects the start of a received character by sampling the URXD signal until it detects a valid start bit. A low level (space) on URXD is interpreted as a valid start bit if it is detected for more than seven cycles of the sampling clock, which is 16 times the baud rate. A space which is 7/16 of a bit period or shorter is ignored and the receiver continues to wait for a valid start bit. When a valid start bit has been detected, the receiver samples the URXD at the theoretical midpoint of each bit. It is assumed that each bit lasts 16 cycles of the sampling clock (1-bit period) so the bit sampling point

is eight cycles (0.5-bit period) after the start of the bit. The first sampling point is therefore 24 cycles (1.5-bit periods) after detecting the falling edge of the start bit. Each subsequent bit is sampled 16 cycles (1-bit period) after the previous one.

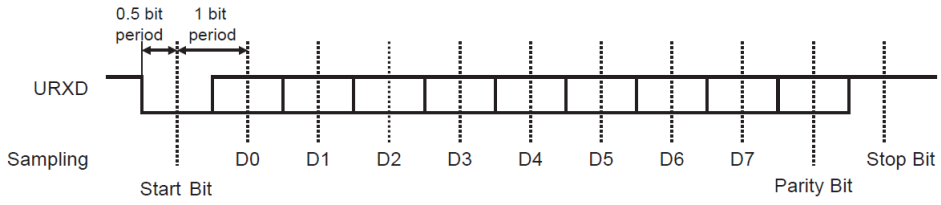


Figure 26- Character reception [16]

When a complete character is received, it is transferred to the Receive Holding Register (UART_RHR) and the RXRDY status bit in the Status Register (UART_SR) is set. The bit RXRDY is automatically cleared when UART_RHR is read.

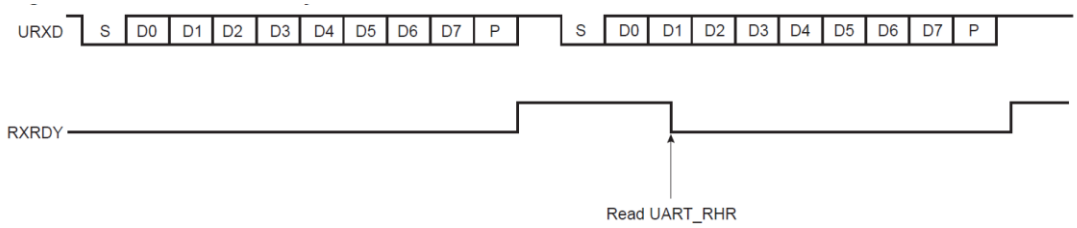


Figure 27- Receiver ready [16]

If the UART_RHR has not been read when a new character has been received, the Overrun bit in UART_SR (OVRE) is set and will be cleared only when the receiver will be reset, that is set the RSTSTA bit in the Control Register.

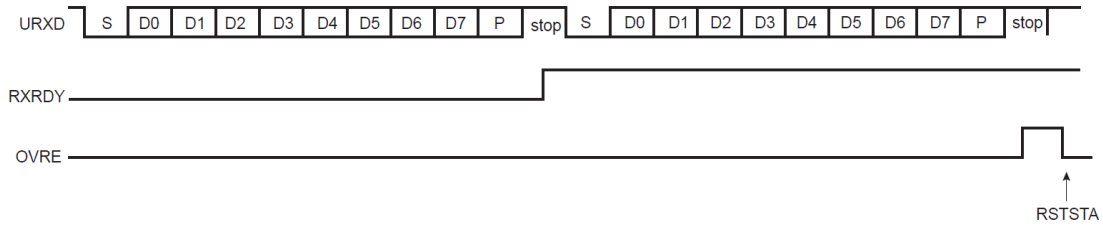


Figure 28- Receiver overrun [16]

The transmitter drives the UTXD pin at the selected baud rate, the line is driven depending on the character format: one start bit at level 0, 8 data bits, one optional parity bit and one stop bit at level 1 (Figure 29).

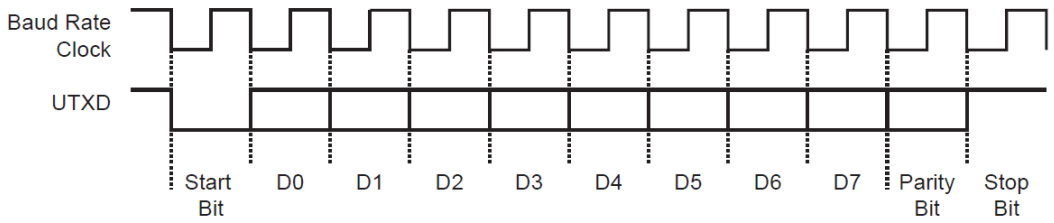


Figure 29- Character transmission example with parity mode enabled [16].

The transmission starts when the programmer writes in the UART_THR, and after the written character is transferred from UART_THR to the internal shift register. The TXRDY bit remains high until a second character is written in UART_THR. As soon as the first character is completed, the last character written in UART_THR is transferred into the internal shift register and TXRDY rises again, showing that the holding register is empty. When both the internal shift register and UART_THR are empty, i.e., all the characters written in UART_THR have been processed, the TXEMPTY bit rises after the last stop bit has been completed (Figure 30).

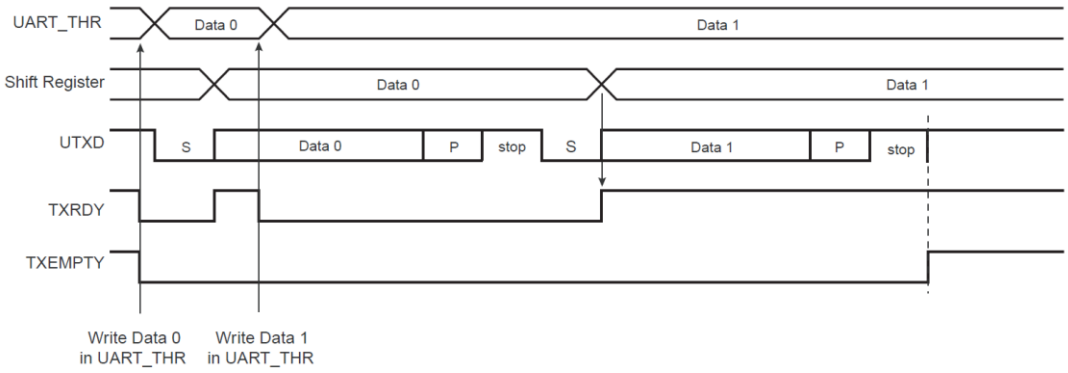


Figure 30- Transmission workflow [16]

3.2.2 User Interface

As first step UART set up must be performed. UART Mode Register allow the user to select the Baude Rate Source Clock: if BRSSRCCK bit is set the baud rate is driven by a PMC-programmable Clock (PCK), otherwise is driven by the peripheral clock.

Name: UART_MR
Offset: 0x04
Reset: 0x00000000
Property: Read/Write

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	CHMODE[1:0]			BRSRCCK	PAR[2:0]			
Access	R/W	R/W		R/W	R/W	R/W	R/W	
Reset	0	0		0	0	0	0	

Bit	7	6	5	4	3	2	1	0
				FILTER				
Access				R/W				
Reset				0				

Figure 31- UART Mode Register [16]

As mentioned in Equation 1, Baud Rate is given by two parameters. While in UART_MR the source clock is set, in UART_BRGR the Clock Divisor is selected. With 16 bit, CD value goes from 0 to 65535, if 0 is selected the Baud Rate is disabled.

Name: UART_BRGR
Offset: 0x20
Reset: 0x00000000
Property: Read/Write

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	CD[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	CD[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Figure 32- UART Baud Rate Generator Register [16]

In UART Control Register if RSTSTA bit is set, PARE, FRAME, CMP and OVRE bit in the Status Register are reset. TXDIS and TXEN bit disable and enable the transmitter respectively, while RXDIS and RXEN bit act on the receiver. If TXDIS and RXDIS are set meanwhile a character is being processed, the disabling occurs right after the character has been completed. RSTTX and RSTRX reset transmitter and receiver but potential process are aborted.

Name: UART_CR
Offset: 0x00
Reset: –
Property: Write-only

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
				REQCLR				RSTSTA
Access				W				W
Reset				–				–

Bit	7	6	5	4	3	2	1	0
	TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX		
Access	W	W	W	W	W	W		
Reset	–	–	–	–	–	–		

Figure 33- UART Control register [16]

UART_IER and UART_IDR enable or disable the interrupt for several control events described in Hardware. Once one of these events occurs and the respective bit in UART_IER is set, the same bit in the Status Register rises (Figure 34).

Name: UART_SR
Offset: 0x14
Reset: 0x00000000
Property: Read-only

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	CMP						TXEMPTY	
Access	R						R	
Reset	0						0	

Bit	7	6	5	4	3	2	1	0
	PARE	FRAME	OVRE				TXRDY	RXRDY
Access	R	R	R				R	R
Reset	0	0	0				0	0

Figure 34- UART Status Register [16]

4 Firmware

UART communication was developed separately from the previous device firmware. Therefore, only the firmware concerning the UART communication will be shown in the following. The UART module was the added to the actigraph firmware by PhD student Daniele Fortunato. Further information about elaboration state and transmission state can be found in [2].

4.1 Block diagram

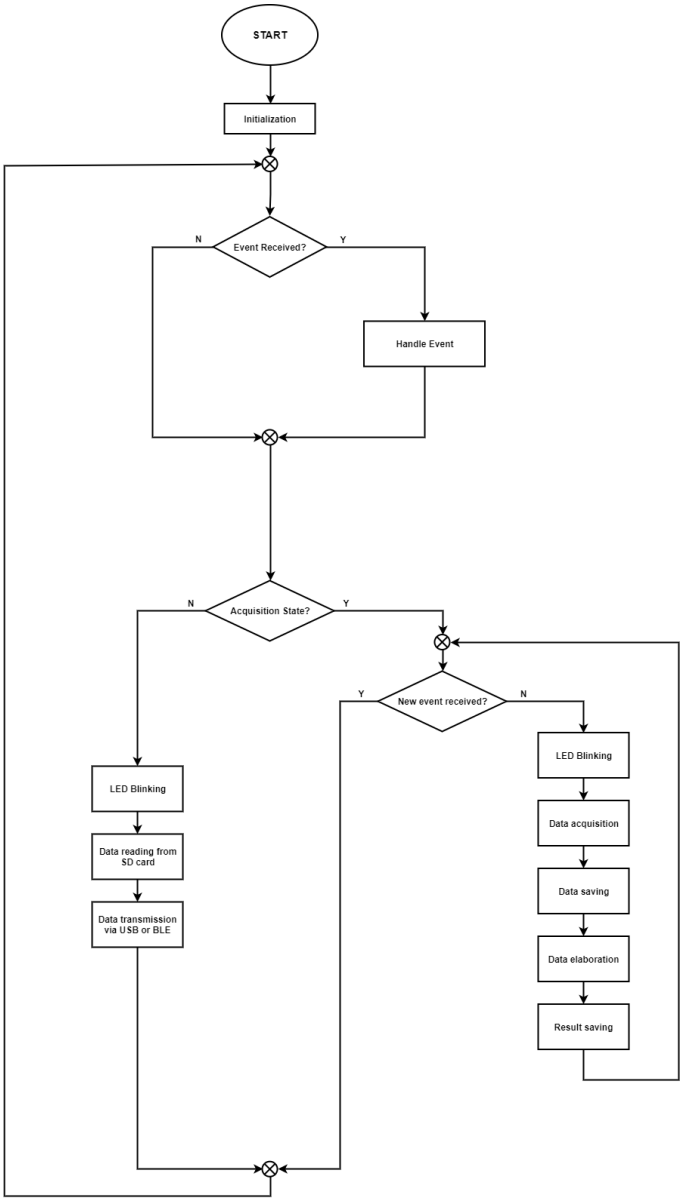
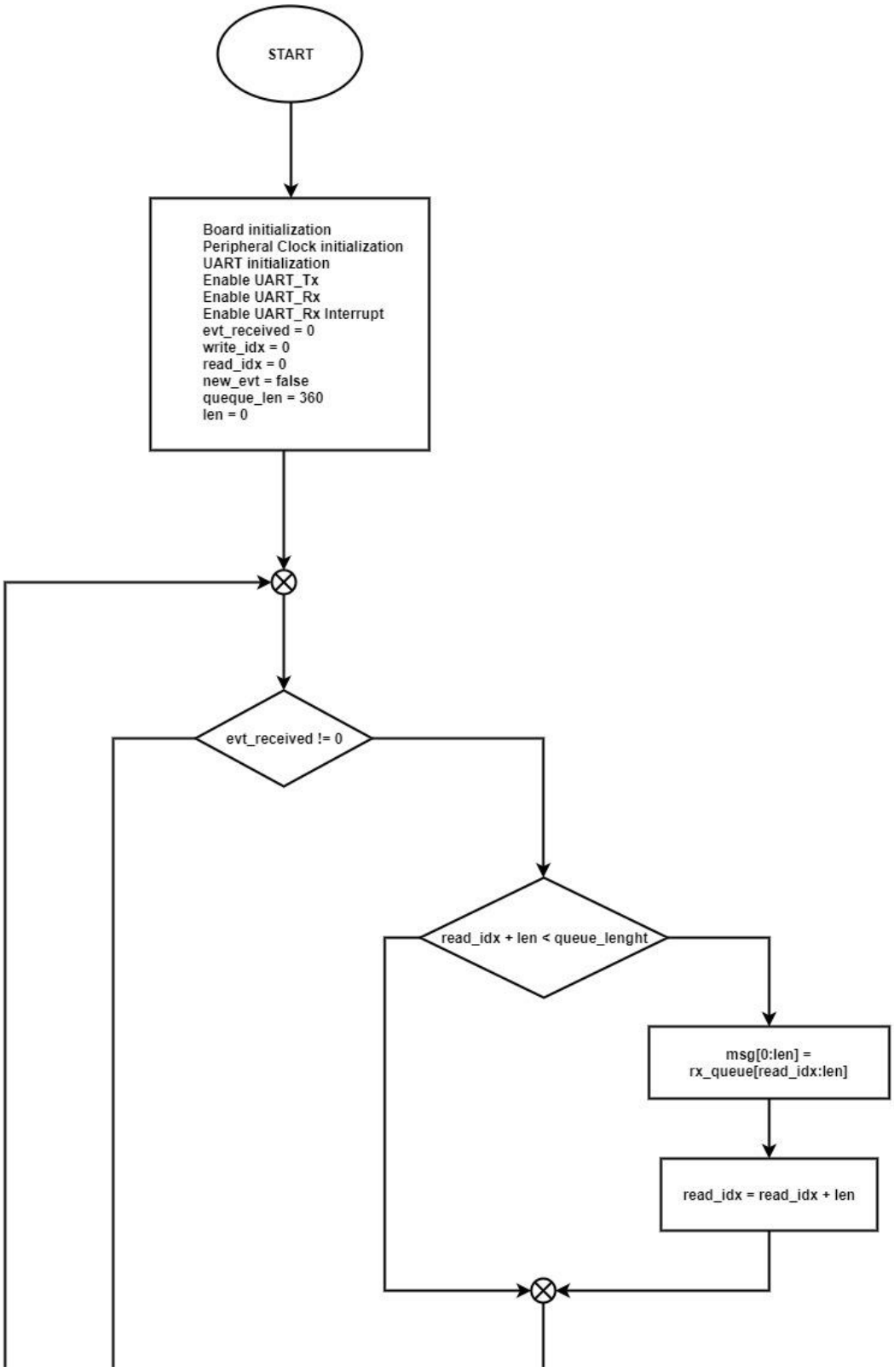
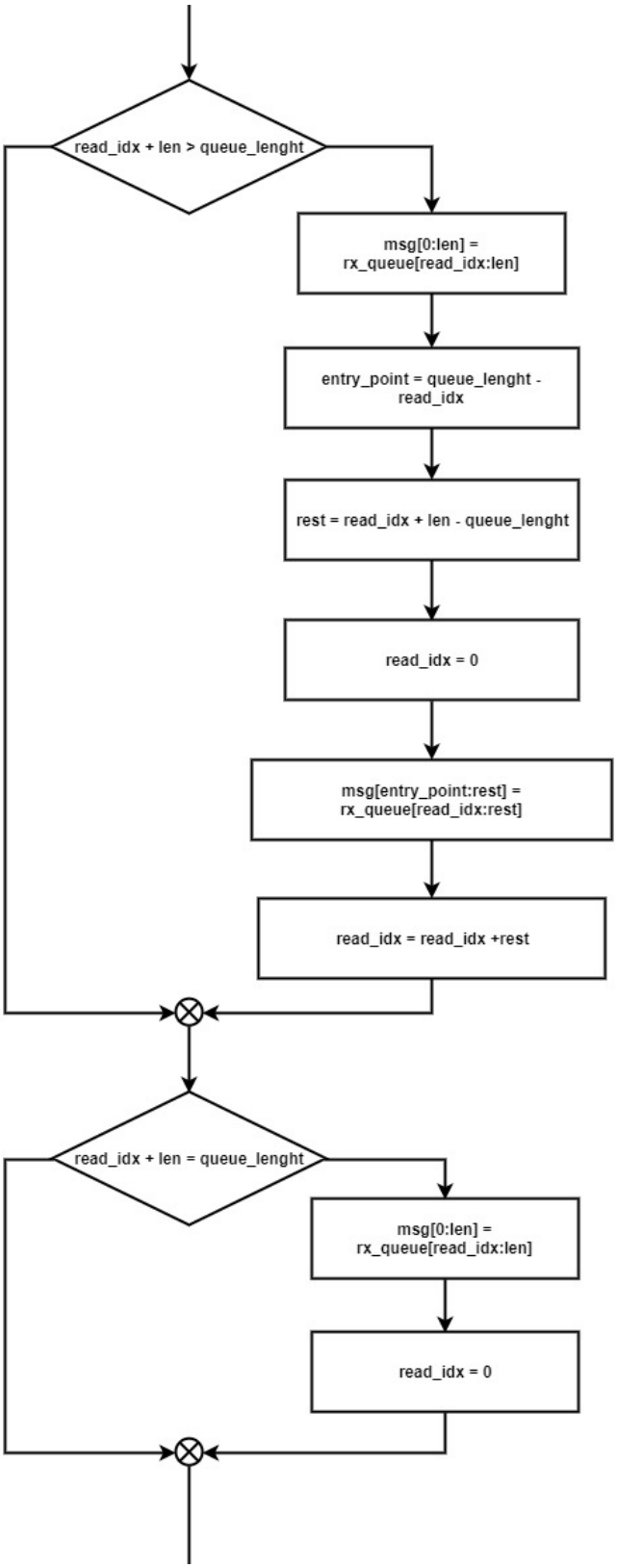


Figure 35- Block diagram of the UART module firmware

4.2 Flow chart

4.2.1 UART Module




read_idx + len > queue_lenght
msg[0:len] =
rx_queue[read_idx:len]
entry_point = queue_lenght -
read_idx
rest = read_idx + len - queue_lenght
read_idx = 0
msg[entry_point:rest] =
rx_queue[read_idx:rest]
read_idx = read_idx +rest
read_idx + len = queue_lenght
msg[0:len] =
rx_queue[read_idx:len]
read_idx = 0

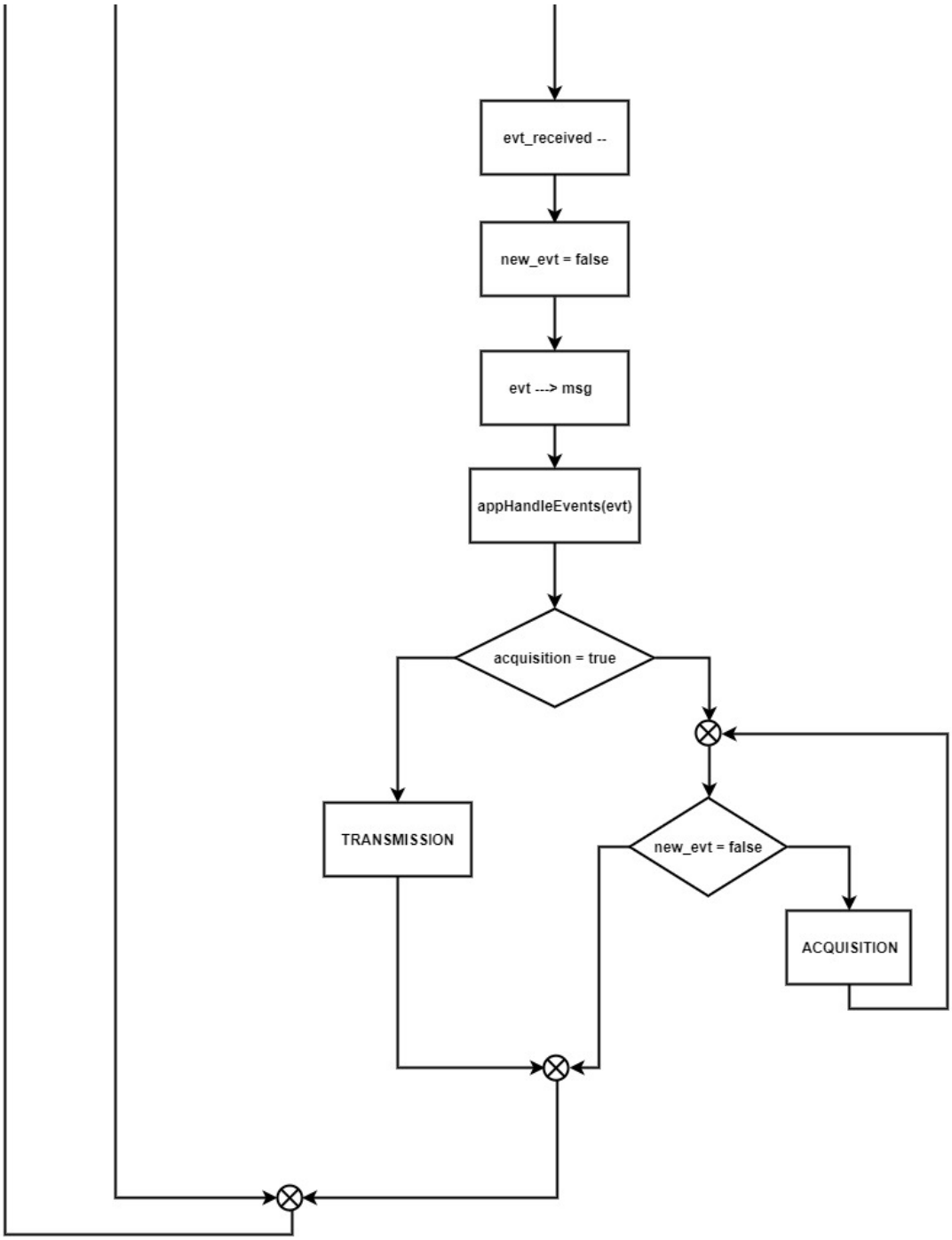


Figure 36- Flow chart of the UART module

4.2.2 UART interrupt

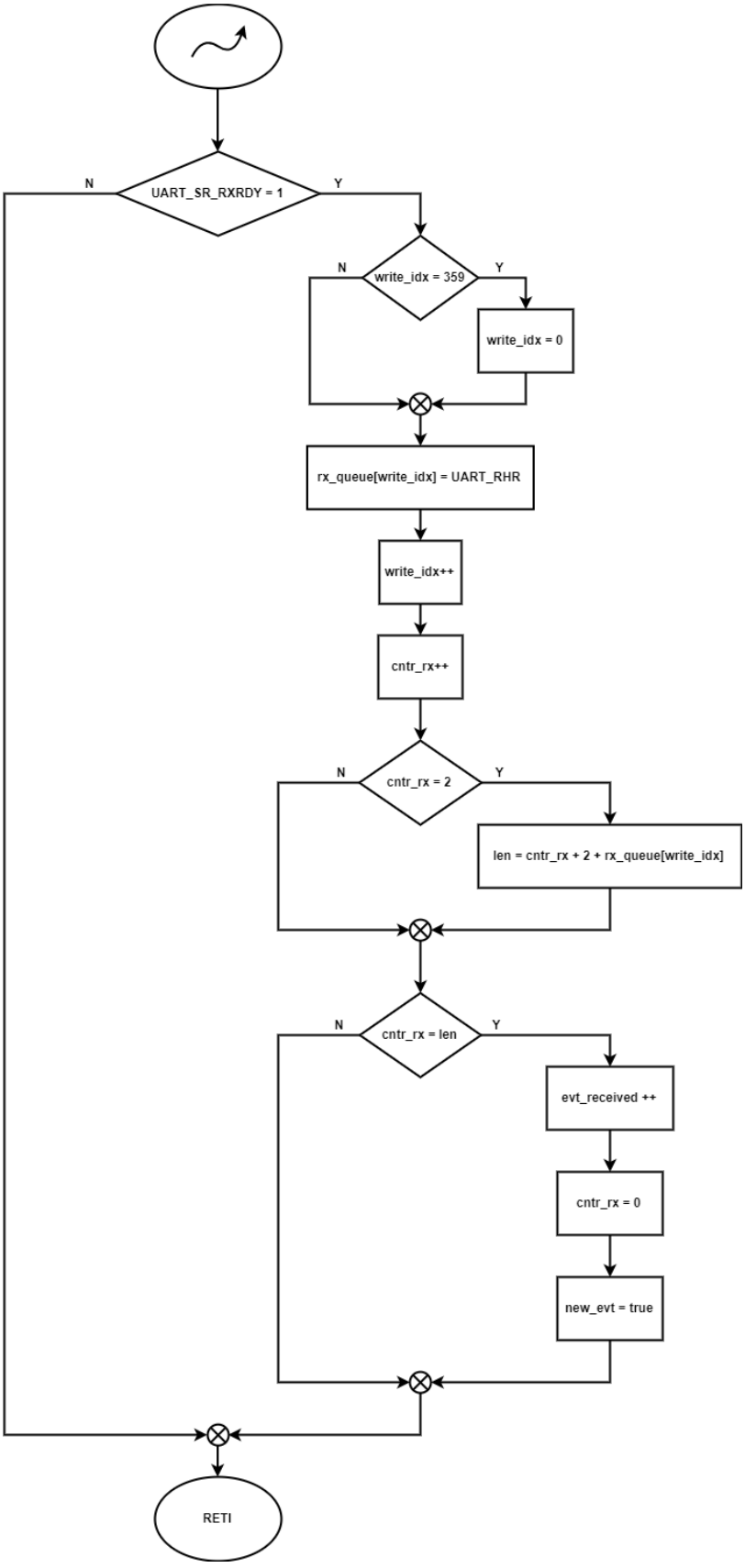


Figure 37- UART Interrupt flow chart

4.2.3 App handle event

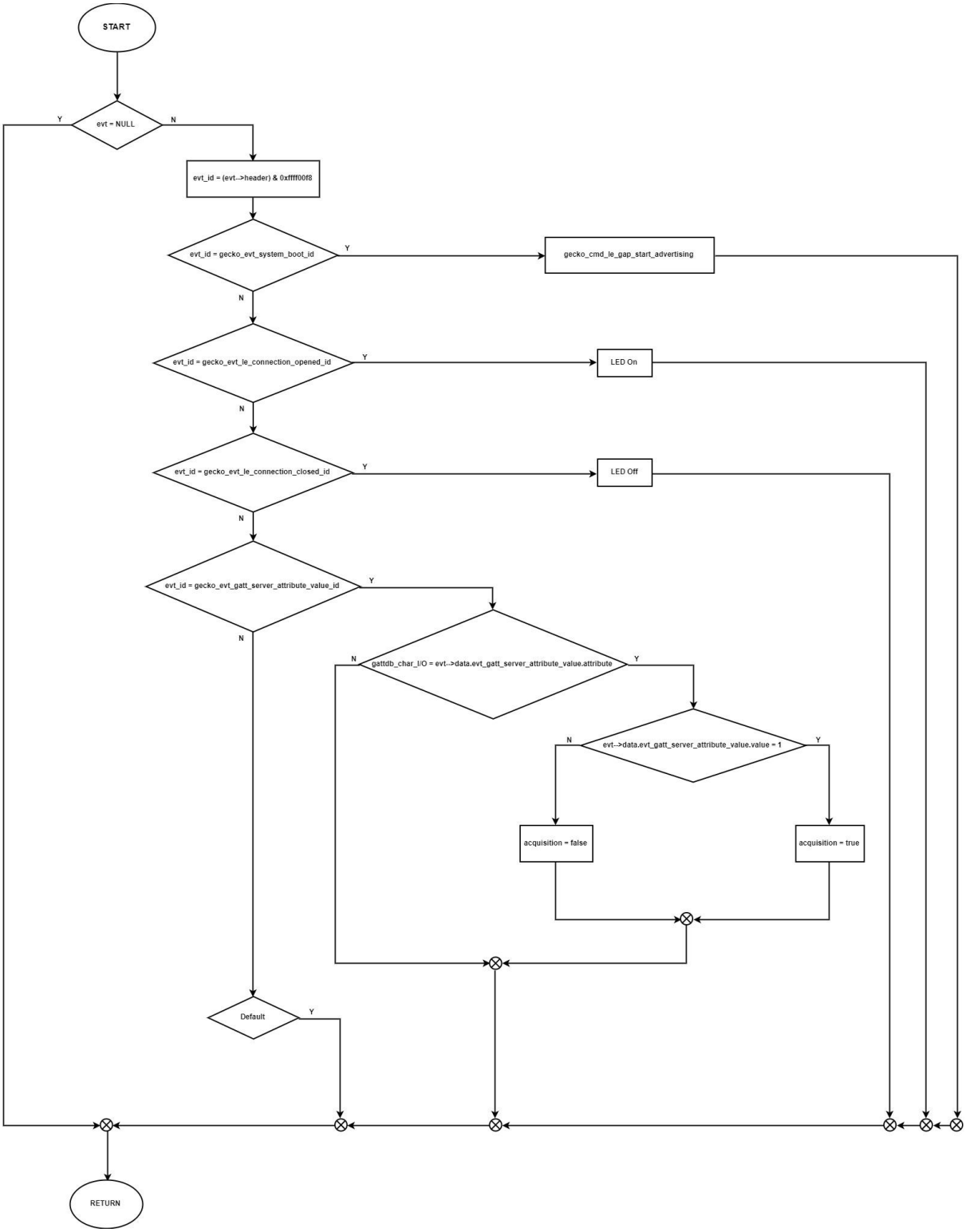


Figure 38- AppHandleEvents flow chart

5 Android App

The Android App implementation, for the actigraph control, requires three main software tools: JDK (Java Development Kit) for Java programming, an IDE (Integrated Development Environment) for a rapid development and the Android SDK (Software Development Kit), an ensemble of documents, libraries, debugger and emulator to create the applications.

5.1 App fundamentals

In order to better understand how an app works, the basic elements are going to be illustrated in the following.

5.1.1 Components

Components are the main building blocks of an Android application: they are entry points through which the user or the system can enter the app. Activities, Services, Broadcast receivers, Content providers are the four types of components, each one performing a distinct task and having a different lifecycle.

Activity

An activity is a screen in which a specific user interface has been drawn; it is the entry point to interact with the user. Most apps are made of several activities, the main activity is the first one to be launched when an app starts. After, it can launches other activities and so on (further information can be found in 5.1.2).

Service

Unlike activities, services work continuously and for a long time in the background with no interaction with the user. These components increase the app reactivity, as services usually organize data to be shown in the activities.

Content provider

A content provider shares data between applications. By default, an android app does not allow others apps to access its data. Such data are stored in the file system, in a SQLite database or on the web and the content provider allows other apps to read or modify those data. This component establishes a correct and secure communication between apps, ensuring application storage isolation.

Broadcast receiver

Broadcast receivers enable the system to deliver events to the app, outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements, such as an incoming message or a low battery notification. Therefore, these components stand for an instantaneous management of special circumstances.

5.1.2 Intent

Activities, Service and Broadcast receivers can request an action from another component, or another app, through an Intent, an asynchronous message managed by the system. There are two types of intents: explicit and implicit. Explicit intents define a specific component to act on and are typically used for components of the same app because the class name is already known. When different apps interact, implicit intents are used as they act on a specific type of component; system finds the appropriate component to start by comparing the contents of the intent with the intent filter, an expression declared in the receiving app that specifies the type of intents that the component would like to receive. Most common use cases are: starting an activity, starting a service and deliver a broadcast. An example of how an implicit intent starts an activity is shown in Figure 39.

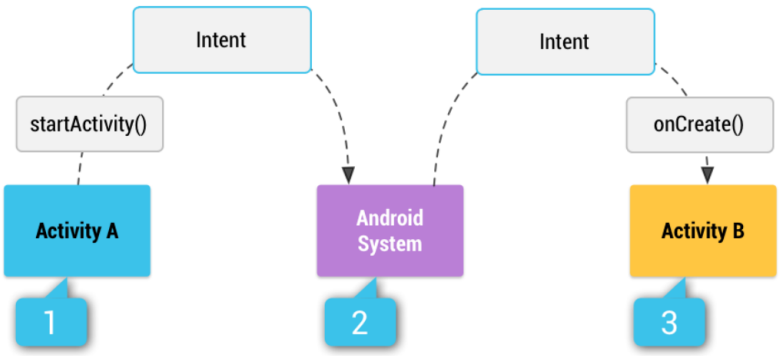


Figure 39- Scheme to start an activity with an implicit intent [17].

5.1.3 Resources

Besides codes, an android app is made up also by resources, such as xml files and images. Resources make it easy to update the visual representation of the app without modifying the code. Main resources are: layouts, values and drawables. Layouts are xml files that define the user interface layout of activities. Values refer to strings, integers and colors defined in xml files and used by the code or by other resources. Drawables are images files (.png, .jpg or .gif) and xml files which define the design of UI elements such as buttons, texts, checkboxes and so on.

5.1.4 Manifest file

All the elements discussed above are declared in the manifest file, an xml file which describes the app essential information. Such information is mandatory in order to make the android build tools (Android SDK), the Android operating system and Google Play, work correctly**.** Besides components, intents and resources, the manifest file declares other elements too.

Requirements

Not every Android device provides the same features and capabilities. In order to prevent installation on devices that lack features needed by the app, it is important to clearly define which kind of devices are supported by the app by declaring device and software requirements in the manifest file. Most of these declarations are informational only and the system does not read them, but external services such as Google Play do read them in order to provide filtering for users when they look for apps in the store.

Permissions

An android app works in isolation from the system and the others apps (sandbox), permissions are then needed if the app wants to access to user data as well as hardware features, such as camera or Bluetooth.

5.2 Android Studio

The IDE used in this project is Android Studio, as it is the official one used for Google's Android operating system [18].

5.2.1 Project structure

An Android Studio project contains everything that defines the workspace for an app, from source code and assets, to test code and build configurations. A project is divided into modules, discrete units of functionality. A module is a collection of source files and build settings that can be independently built, tested and debugged [19].

Project and Android view

The actual file structure is shown in the Project view (Figure 40-a). There are directories for each module and a grandle directory that defines the build configuration that applies to all modules. Each module directory contains build outputs (build/), private libraries (libs/), all code and resources files (src/) and its specific build configurations (build.grandle). A less appropriate but more intuitive file structure is given by the Android view (Figure 40-b). It is organized by modules and file types and it is the default one. Here all the project's build configuration files are shown in the Grandle Script section and each app module includes the Manifest file, the Java codes and the resources.

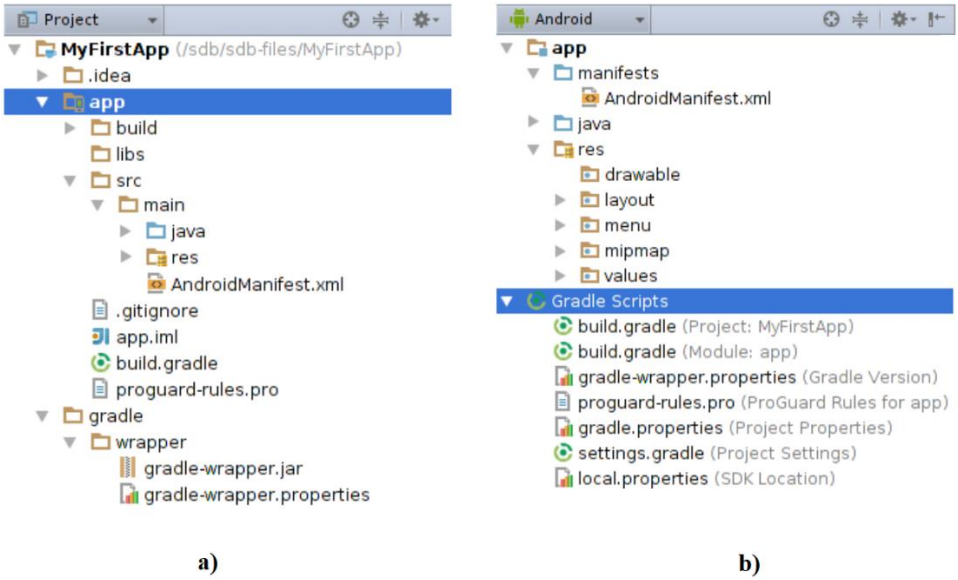


Figure 40- a) Android Studio's Project view, b) Android Studio's Android view

5.2.2 Layout editor

As mentioned before, activities are Java files through which the user interacts with the app. Each activity represents a screen with a User Interface (UI) drawn in it. However, the UI design is not defined in the java code, it is defined in a layout resource file. Such xml file is then associated to the correct activity in its java code. To design the UI there is the layout editor, a tool that compiles the xml file automatically by simply dragging-and-dropping the desired views in the screen. The UI has a hierarchy structure of Viewgroup and View object. A ViewGroup is a special view that contains and controls other views called “children”. A View is a class representing the basic building block of the UI (button, text, checkbox, etc.). It occupies a rectangular area on the screen and it is responsible for click-event handling.

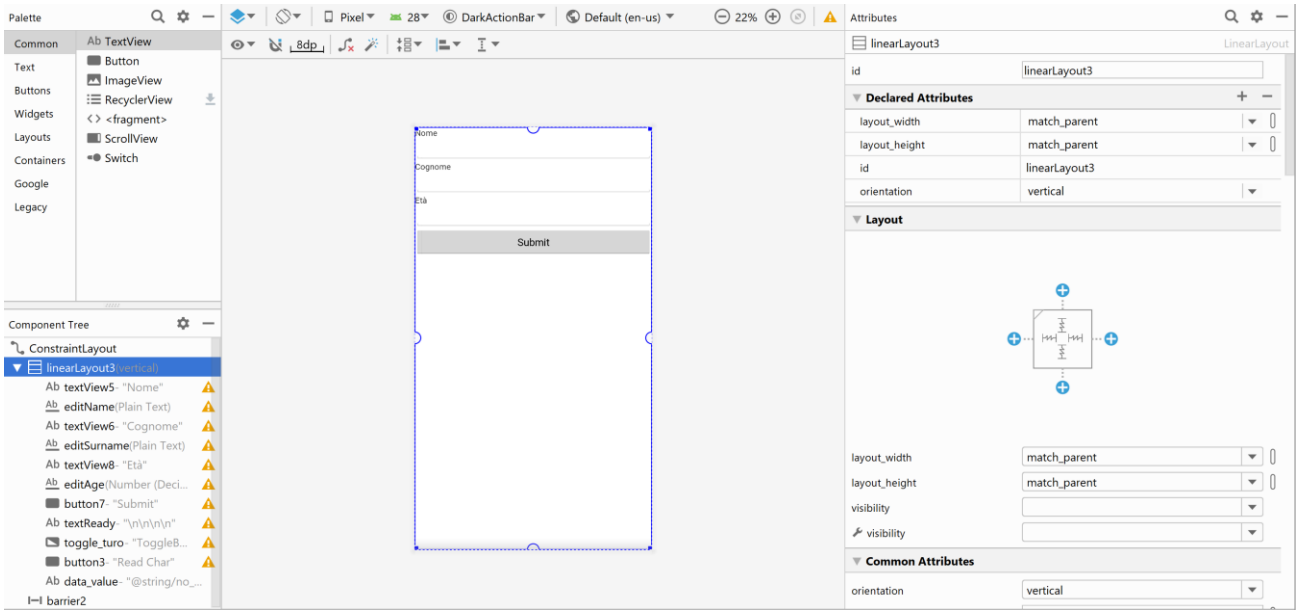


Figure 41- Layout editor

Each View object has an onClick propriety, which can be associated to a method implemented in the java code of the respective activity class. In this way, when an element (or component?) of the UI is clicked, the system calls the selected method.

5.3 Actigraph App

The app developed in this project starts from the "Android BluetoothLeGatt Sample" code by Google's developers [20]. This sample shows a list of available BLE devices and provides an interface to connect, display data and display GATT services and characteristics supported by the devices. It creates an Android Service for managing connection and data communication with a GATT server hosted on a given Bluetooth LE device. The activities communicate with the service, which in turn interacts with the Bluetooth LE API. The app has been modified to be more user friendly and more suitable to control the actigraph.

5.3.1 Manifest file

First, in the manifest file, requirements and permissions are declared. Requirements set the app available to BLE-capable devices only. If smartphones do not support the BLE functionality, it cannot install the app.

```
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
```

To use the Bluetooth feature, three permissions must be declared: BLUETOOTH to perform any communication task, BLUETOOTH_ADMIN to initiate discovery and manipulate settings and ACCESS_FINE_LOCATION to return scan results.

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

After that, the application element is declared, it contains attributes and sub-elements. Attributes define the app icon, name and theme, while sub-elements declare the application components, in this case just activities and the service.

5.3.2 Application codes

In the following, the sequence diagrams of the android app will be illustrated. Be aware, such diagrams do not represent every single step involved in the java codes, just the main ones are highlighted for ease of viewing and understanding.

Main Activity

`MainActivity` is the launcher activity. Once the user launches the app, the UI is set by loading the *activity_main.xml* layout file. When the user clicks on the "Find Device" button, the system calls the *onFindDevice()* method. Such method starts the `DeviceScanActivity`: first, it creates an intent object to be delivered to the activity and then, it calls the *startActivity()* method. The system receives the call and starts an instance of `DeviceScanActivity`. During the layout design, in order to make the button respond to the click action, the onClick propriety of the "Find Device" button was edited by selecting the *onFindDevice()* method (further information can be found in 5.2.2).

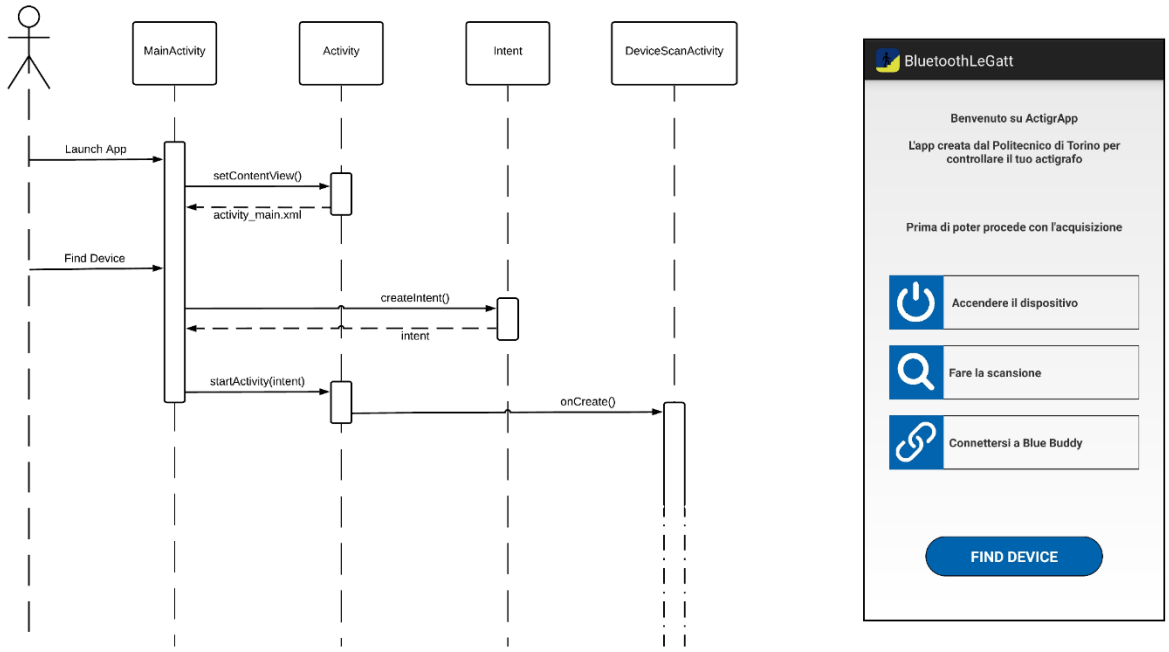


Figure 42- MainActivity Sequence diagram and screenshot

Device Scan Activity

When the `DeviceScanActivity` starts, the action bar, to control the scan, is set in the UI. Then, if Bluetooth is not already on, the system requests the user to enable it. When the scan starts, all the discovered devices are saved in a list and then added to the layout to be showed to the user. After a predefined time interval, the scan stops automatically. When the user selects one of the devices to connect to it, its info are retrieved from the saved list and then are added to the intent as extra values. The intent is delivered to the system in order to start the `DeviceControlActivity`.

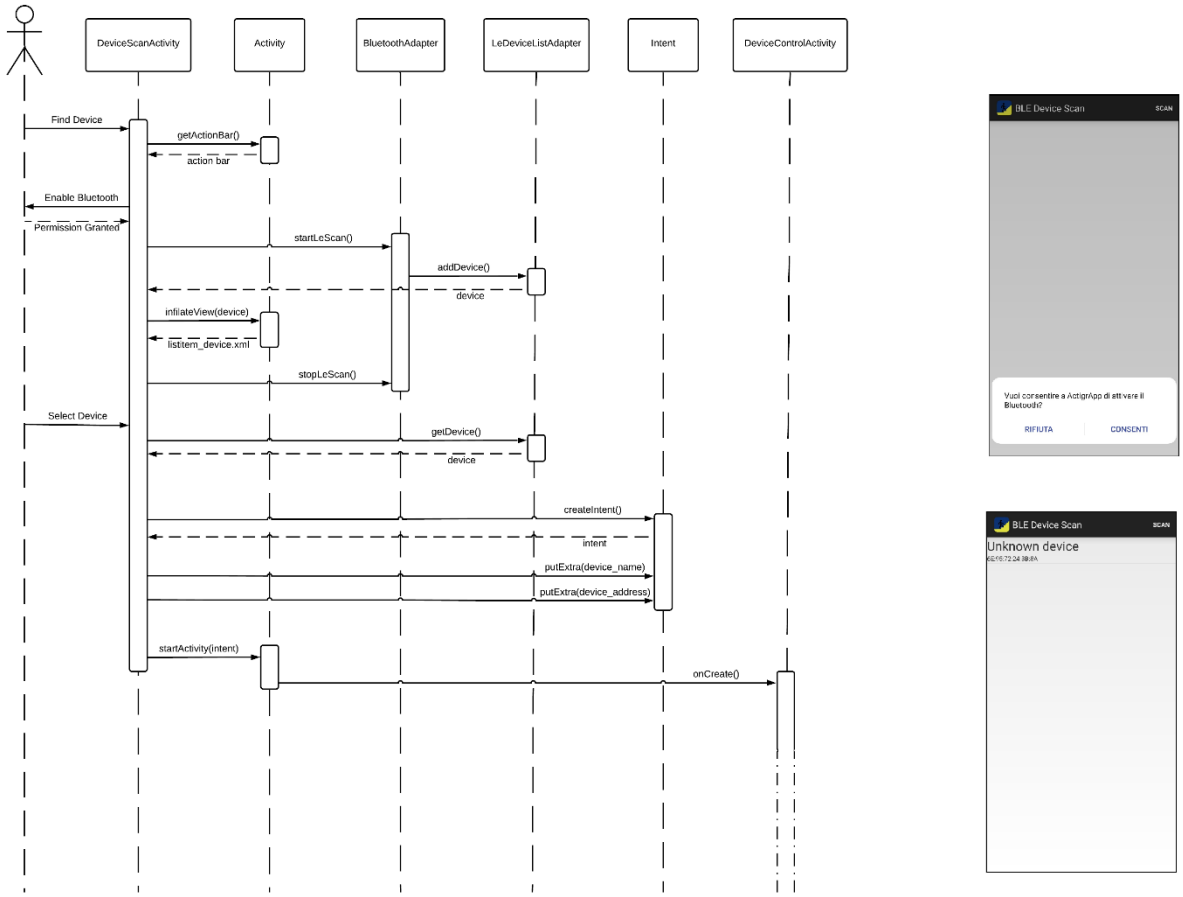


Figure 43- DeviceScanActivity Sequence diagram and screenshots

DeviceControlActivity

When the `DeviceControlActivity` starts: it sets the layout and gets the intent that started the activity in order to extract info concerning device name and address. Furthermore, it creates an intent to bind the `BluetoothLeService`, a service which interacts with the BLE API. Then, the activity connects to the device through the service. In order to start the acquisition, the system requires the user to fill a simple form with his data (name, last name and age). When the Submit button is clicked, the user data are written in the GATT Server database, more accurately, in the characteristics of the UserData service. Once the writing is successful, the Toggle button is made visible. When the toggle button is clicked on, system writes a hexadecimal value in the I/O characteristic belongings to the Acquisition service of the GATT Server profile. The 0x01 value is written to start the acquisition, while, the 0x00 value is written to stop it. When the acquisition is stopped, the Result button is made visible in order to read result data from the GATT Server database.

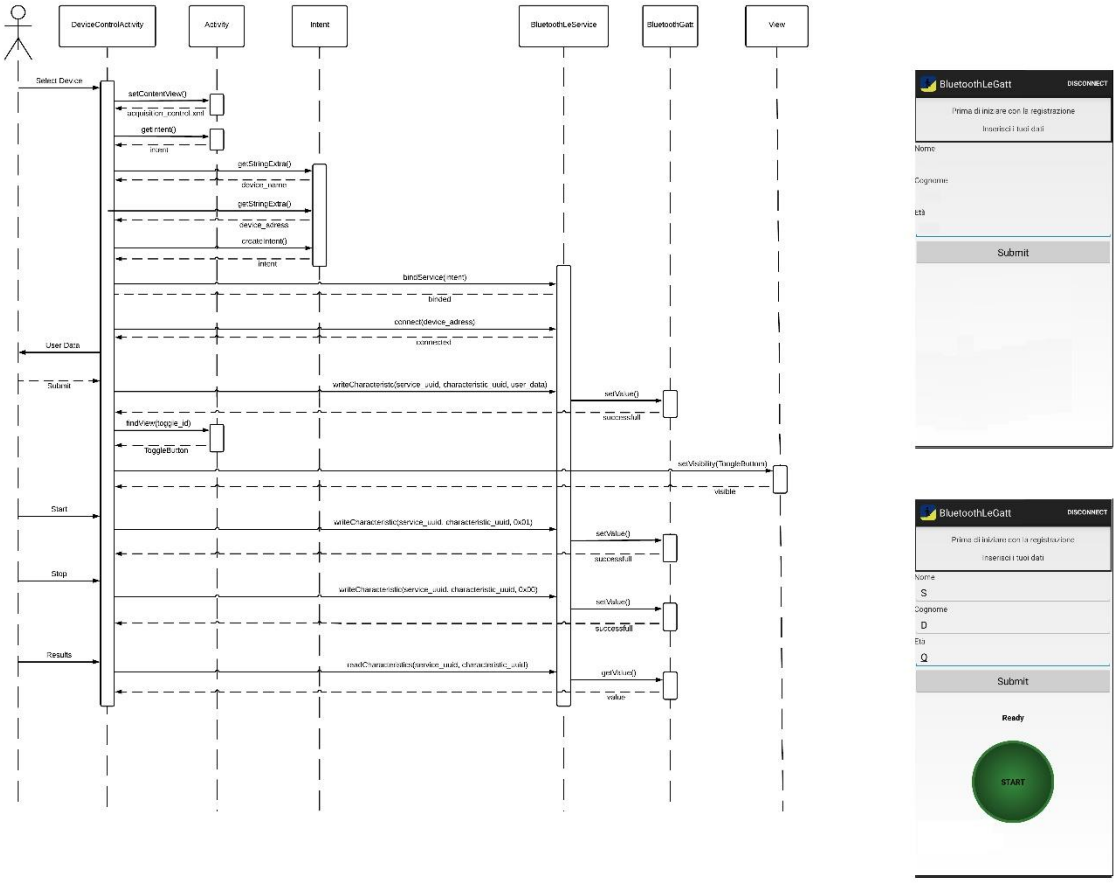


Figure 44- DeviceScanActivity Sequence diagram and screenshots

6 Conclusion

The aim of this thesis was to develop a wireless actigraph to be remote controlled. Therefore, the goal has been achieved. The integration of the BGM121 in the device has been successful: the firmware developed to communicate with the BLE has been easily integrated in the previous firmware; furthermore the communication flow can be easily modified according to custom use cases. The implementation of the android app allows the remote control, making the actigraph a 100% portable device. Future improvements are:

- power consumption analysis and tuning of the communication parameters;
- android studio graph development in order to show HAR diagram directly in the smartphone;
- app modification in order to connect more than one device to a single smartphone.

References

[1] S. Ancoli-Israel, R. Cole, C. Alessi et al. “The role of actigraphy in the study of sleep and circadian rhythms.” *SLEEP* vol 26, pp. 342-92, 2003

[2] F. Bolognesi “Programming and Development of a wearable device based on IMU unit for on board processing”, Master thesis, Polytechnic of Turin, Italy, 2018

[3] G. De Leonardis, D. Fortunato, et al. "An innovative microprocessor-based system for Human Activity Recognition: A fast and reliable classification algorithm." *Gait & Posture* vol. 66, pp S14-S15, 2018

[4] Sushma M. “Wibree technology”. *International Journal of Recent Trends in Engineering & Research*, vol. 3, pp. 497-502, April 2017.

[5] The Future of Things “Nokia’s Wibree and the Wireless Zoo”. Internet: https://thefutureofthings.com/3041-nokias-wibree-and-the-wireless-zoo/

[6] BBC “Bluetooth rival unveiled by Nokia”. Internet: http://news.bbc.co.uk/2/hi/technology/5403564.stm

[7] Wikipedia “Bluetooth Low Energy”. Internet: https://en.wikipedia.org/wiki/Bluetooth_Low_Energy

[8] J. Tosi, F. Taffoni, et al. “Performance Evaluation of Bluetooh Low Energy: A Systematic Review”. *Sensors* vol 17, Sept. 17

[9] BluetoothSIG. “Vol 1: Architecture & Terminology Overview, Part A: Architecture. In *Specification of the Bluetooth® System, Covered Core Package Version 5.1*” Kirkland, WA, USA, The Bluetooth Special Interest Group, 2019

[10] R. Davidson, K. Townsend et al. *Getting Started with Bluetooth Low Energy.* Sebastopol, CA, USA. O’Reilly Media, 2014

[11] Silicon Labs, “UG103.14: Bluetooth® LE Funfamentals”

[12] BluetoothSIG. “Vol 6: Core System Package [Host volume], Part F: Attribute protocol (ATT). In *Specification of the Bluetooth® System, Covered Core Package Version 5.1*” Kirkland, WA, USA, The Bluetooth Special Interest Group, 2019

[13] Microchip Developer “Attribute and Data Hierarcy”. Internet: https://microchipdeveloper.com/wireless:ble-gatt-data-organization

[14] Silicon Labs, “QSG139: Getting Started with *Bluetooth*® Software Development”

[15] Silicon Labs, “KBA_BT_1602: NCP Host Implementation and Example”. Internet: https://www.silabs.com/community/wireless/bluetooth/knowledge-base.entry.html/2018/01/18/ncp_host_implementat-PEsT

[16] Microchip, “SAM E70/S70/V70/V71 Family Data Sheet”

[17] Android Developers, “Intent and Intent Filters”. Internet: https://developer.android.com/guide/components/intents-filters.html

[18] Android Developers, “Meet Android Studio”. Internet: https://developer.android.com/studio/intro

[19] Android Developers, “Projects overview”. Internet: https://developer.android.com/studio/projects/index.html

[20] GitHub “Android BluetoothLeGatt”. Internet: https://github.com/googlesamples/android-BluetoothLeGatt