VŠB - Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra kybernetiky a biomedicínského inženýrství

Víceúčelový senzor pro analýzu pohybů člověka

Multipurpose Sensor for Human Movement Analysis

2016 Bc. Jan Foltyn

VSB - Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Cybernetics and Biomedical Engineering

Diploma Thesis Assignment

Student:	**Bc. Jan Foltyn**
Study Programme:	N2649 Electrical Engineering
Study Branch:	3901T009 Biomedical Engineering
Title:	Multipurpose Sensor for Human Movement Analysis Víceúčelový senzor pro analýzu pohybů člověka
The thesis language:	English

Description:

1. Getting knowledge of human motion monitoring problematics and Bluetooth Low Energy standard.
2. Research of contemporary solutions of this problematics.
3. Design and realization of firmware for recently build human motion sensoric platform, with use of Bluetooth Low Energy communication.
4. Design and realization of a uniform communication protocol for use with sensoric platform and another applications, capable of low power consumption improvement.
5. Software for wireless configuration of sensoric platform, real-time measurement of received data and their storage.
6. Tests and measurements with complete prototype of human motion sensoric platform.
7. Summary of achieved results.

References:

[1] GUPTA, Naresh C. *Inside Bluetooth low energy.* Boston: Artech House, 2013, 1 online zdroj (422 pages). Artech House mobile communications series. ISBN 978-1-60807-580-5.
[2] HEYDON, Robin. *Bluetooth low energy: the developer's handbook.* Upper Saddle River, NJ: Prentice Hall, 2012, 345 p. ISBN 978-0132888363.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **Ing. Martin Černý, Ph.D.**

Date of issue: 01.09.2015
Date of submission: 29.04.2016

doc. Ing. Jiří Koziorek, Ph.D.
Head of Department

prof. RNDr. Václav Snášel, CSc.
Dean

Declaration

„I hereby declare that this thesis is my own work. I stated all materials and publications that I used."

The date of thesis submission: 1. 7. 2016

Jan Foltyn

Acknowledgment

Firstly my thanks goes to leader of my thesis Ing. Martin Černý, Ph.D. for his help and patience. Then I wish to express gratitude to my colleagues from Vigilio S.A. company for their help during my internship in France. And finally to my family and fricnds, for their support during my studies.

Abstrakt

Tento dokument popisuje víceúčelové zařízení pro analýzu lidského pohybu. Obsahuje popis použitých technologií a senzorů, použitých k měření fyzikálních veličin, spojených s analýzou pohybu člověka. Dále popisuje základy technologie Bluetooth Low Energy. Praktickou částí práce byl vývoj firmwaru mikropočítače, určený ke sběru dat, konfiguraci použitých senzorů a dále PC software pro Windows, určený k monitorování naměřených dat v reálném čase a jejich zápisu do souboru. Tento dokument by měl sloužit i jako průvodce pro vývojáře, kteří budou projekt upravovat či dále vyvíjet. Popisuje navržené ovladače senzorů, Bluetooth komunikace, hardwarových nastavení a jiné důležité kroky pro zprovoznění této senzorické platformy, jako je například nastavení vývojového softwaru tohoto zařízení.

Klíčová slova

Inerciální měřící jednotka, Bluetooth Low Energy, akcelerometr, magnetometr, gyroskop, senzor úhlové rychlosti, barometr, Bluegiga, nízký odběr, STM32

Abstract

This document describes a multi-purpose device for analysis of human body movement. It contains a description of technologies used in measurement of some physical variables associated with a human body movement, as well as an explanation of Bluetooth Low Energy communication basics. Practical part of work was a development of embedded software for data acquisition, sensors configuration, BLE module service and a Windows PC software for real-time monitoring and data storage. This document should also serve as a step by step guide for developer, who is going to continue in this work. It describes operation of sensor drivers, BLE communication, hardware settings, PC software and also some necessary steps, like setting the programming interface, making user able to run the embedded software and improve this device.

Key Words

Inertial Measurement Unit, Bluetooth Low Energy, accelerometer, magnetometer, gyroscope, angular velocity sensor, barometer, Bluegiga, low power, STM32

Shortcuts and symbols

BLE	Bluetooth Low Energy
BT	Basic Rate
COM	Communication Port (virtual serial port)
EDR	Enhanced Data Rate
fc	Connection frequency
FHSS	Frequency Hopping Spread Spectrum
FIFO	Firs In First Out memory
fs	Sampling frequency
GATT	Generic Attribute Protocol
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
IMU	Inertial Measurement Unit
ISM	Industrial Scientific and Medical (frequency band)
ISR	Interrupt Service Routine
LE	Low Energy
MCU	Micro Controller Unit
MEMS	Micro Electro Mechanical Systems
SWD	Serial Wire Debug
UUID	Universal Unique Identifier

List of Figures

List of Tables

Table of contents

Introduction

Aim of this project was to develop device with multiple sensors, which is able to communicate with remote PC or mobile device via Bluetooth Low Energy, so as a development of PC software for real-time data monitoring and storing the measured data into files for another research uses. This device was developed in cooperation with Vigilio S. A. company, and was called Activ'Platform.

In general, Activ'Platform has to be device for research uses, tests and should also serve as a prototype for development of some commercial solutions in the future. Therefore this device has to be able to get from its sensors and peripherals as much as possible. Configuration of device will be done wirelessly as well as transmission of collected data. Firstly we should mention, that there are lot of devices on the market, designed for measurement of human body motion. Some of them can be used for research, allowing user to change settings of sensors or data transmission using their software development kits. But in this project, we want to go even further inside the data acquisition unit and control the process from the start. This is the only way how we can achieve the best possible optimization in the future. Activ'Platform differs from conventional products by amount of measured physical variables. It is available to measure acceleration, magnetic field, angular velocity, altitude (atmospheric pressure) and is prepared for implementation of human temperature sensor. All those parameters can be measured by one hardware platform, which is the main goal of this project. Hardware part was mainly designed by Vigilio company and except some necessary changes made after trials and tests with prototype, it is not disclosed in this document.

Practical part of this document contains description and algorithms of Activ'Platform embedded software, hardware configuration and operation of drives written for on-board sensors. Because this work should serve as a basic developer guide, some important C functions are referenced and explained in this project. Most of them can be used in the future, especially sensor drivers, completely written or modified for Cortex M0 controllers. Embedded software was written in Keil uVision 5 and this work also includes a guide how to properly set the programming interface for microcontroller and J-link debugger. Last part of work is a functional description of C# software for monitoring and saving data. And of course, simple user guide for this application and references to important functions.

1. Used technologies

First step of measurement device development is usually a definition of what we want to measure and how precise the results should be. Naturally second step lead us to find if our concept is physically achievable with an existing technical equipment. If it's not, we must return to the starting point and reconsider it. Next step is usually to find most suitable sensors considering the price etc. But this project was started from a different point of view. Sensors, used in Activ'Platform are nowadays commonly used for human body motion measurement and a fact that this technology is still more often implemented in mobile devices and consumer electronics, makes them even more available on the market. At this point we don't have to search for the available technologies, but select sensors that can offer us enough information for a reasonable price. And a meaning of this project was to build a platform that can be used for multiple purposes in medical research. Therefore, sensors were chosen by team of colleagues from VŠB-TU, Vigilio. S.A. and institute for nanotechnology of Lyon, according to their experiences and knowledge about human motion measurement and fall detection. Working with these sensors needs some theoretical basic about their physical principle, which will be explained underneath.

1.1. Accelerometry

Acceleration is a first time derivative of velocity. As a non-electrical physical unit, a specific method of conversion to electrical unit is needed and it is based on combination of Hook's law and Newton's second law. On the Figure 1 is an object of known mass, attached to the outer sensor structure by an elastic element. When the whole structure is accelerated in a sensitive direction of that elastic element a force is applied on it according to Hook's law, which results in displacement of the inner object. Acceleration value can be deduced from measured displacement and known constants with equation (1).

$$\boldsymbol{a} = \frac{\boldsymbol{k}}{\boldsymbol{m}} \cdot \Delta \boldsymbol{x} \tag{1}$$

spring stiffness: $k\left[\frac{N}{m}\right]$

mass of inner object: $m[kg]$

displacement: $\Delta x[m]$

acceleration: $a\left[\frac{m}{s^2}\right]$ $\left[\frac{N/m}{kg} \cdot m = \frac{kg \cdot m}{kg \cdot s^2} = \frac{m}{s^2}\right]$

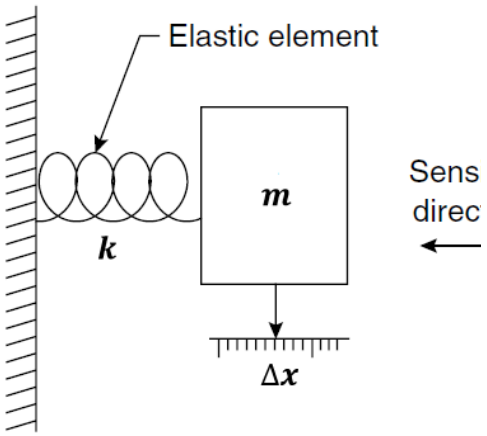

Figure 1 Working principle of accelerometer [1]

Accelerometers varies from the displacement sensor and type of connection between outer sensor structure and inner mass. There are several common types of displacement sensors like strain gauge sensors, piezoelectric sensors or capacitive sensors. Piezoelectric sensors are inexpensive, they can offer great endurance, but their precision is not sufficient for most of medical applications. Achievable precision of capacitive sensors is much better, while the endurance is still good and power consumption is so low that sensors can run on a coin cell battery with some low power microcontroller. Measured capacitance can be changed by changing the space between two plates of a capacitor. The space is called dielectric and its material has also significant influence on the resulting capacity. The fundamental principle of plate capacitor is given by equation (2).

$$\boldsymbol{C = \varepsilon_0 \cdot \varepsilon_r \cdot \frac{S}{d}} \tag{2}$$

vacuum permittivity	$\varepsilon_0 \left[\frac{F}{m}\right]$	
relative permittivity	ε_r [ ]	
cross sectional area of plate	$S\ [m^2]$	
dielectric	$d\ [m]$	
capacity	$C\ [F]$	$\left[\frac{F}{m} \cdot \frac{m^2}{m} = F\right]$

Principle of operation is clear, but construction of accelerometer needs a special technology, to build the sensor in smallest possible dimensions and do not let it affect precision. Today's accelerometers are using microelectromechanical systems (MEMS). MEMS systems are combination of mechanical and electrical components in micrometer scale, formed by semiconductors and microfabrication

technologies, when micro machine processing is used to place all the parts on a common silicon substrate. Picture below shows two types of MEMS accelerometer with a strain gauge and piezoelectric sensor [1] [2].

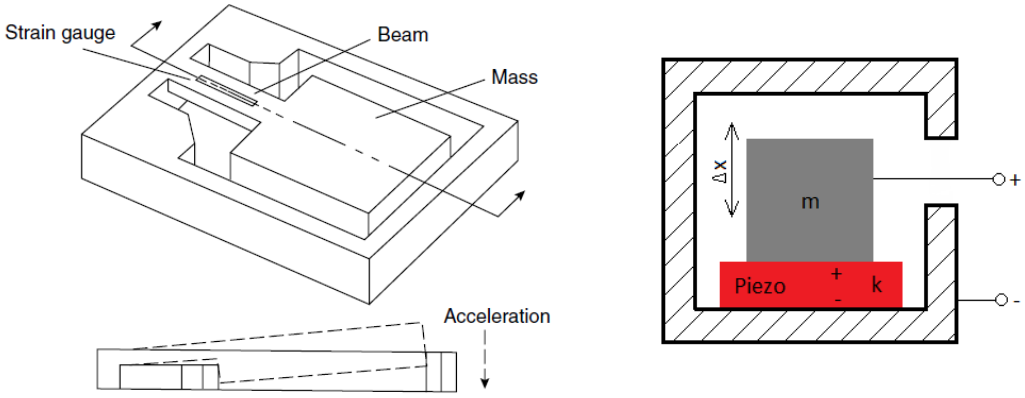

Figure 2 Construction of different accelerometer types [1]

1.2. Magnetometer

Most commonly used physical variable, related to magnetic field is a magnetic induction. There are two common types of its expression. Gauss G in CGS (Centimeter Gram Second) system and Tesla T in SI (Le Systeme International) system. CGS system is obsolete, however the expression of magnetic inductance in G is still commonly used by many producers of electronic components with high sensitivity. The reason is clear, because of the conversion factor 1T = 10000G. When a detection of really low magnetic inductance is required, the measurement scale will be in single units of Gauss. And due to conversion to digital integer number, it is much more practical to use G against T or mT.

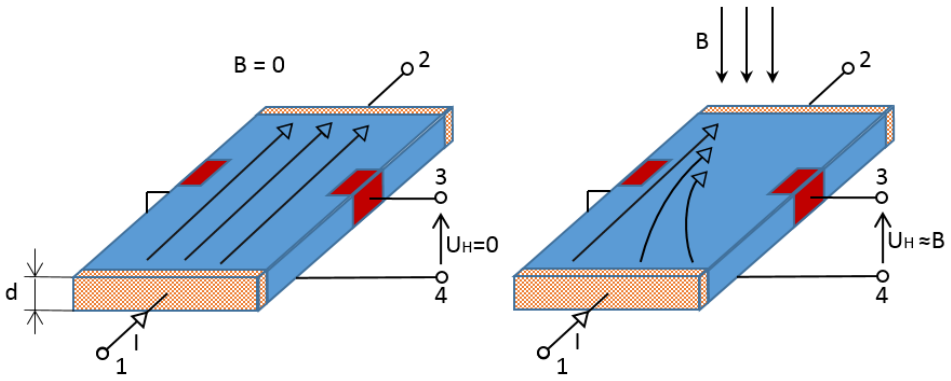

Figure 3 Hall effect sensor

Mostly used technology of low magnetic inductance measurement is a using of Hall effect sensor. Hall effect sensor is usually realized as a thin plate of semiconductor with two pairs of contacts.

First pair of wide contacts (1 and 2) is used to lead the current trough the sensor and second pair of thinner plates (3 and 4) serves as an output source of Hall voltage. When the sensor is not influenced by magnetic field, current flows equally through the plate. In a magnetic field, inductance affects charge carriers by the force, perpendicular to their direction, causing the separation of electric current on one side of sensor plate. Resulting difference of potentials is an output Hall voltage expressed by the equation (3).

$$\boldsymbol{U_H = h \cdot \frac{B \cdot I}{d}} \tag{3}$$

material constant	$h \left[\frac{m^3}{A \cdot s}\right]$	
magnetic inductance	$B[T]$	$\left[T = \frac{Wb}{m^2}\right]$
electric current	$I\ [A]$	
plate thickness	$d\ [m]$	
output Hall voltage	$U_H\ [V]$	$\left[\frac{m^3}{A \cdot s} \cdot \frac{T \cdot A}{m} = \frac{m^3}{A \cdot s} \cdot \frac{V \cdot s \cdot A}{m^3} = V\right]$

Modern Hall effect sensors are provided as standalone parts with internal source of current and analog voltage output. The main differences between them are sensitivity, and power consumption. Sensor used in Activ'Platform has three hall sensors put in different orientation, therefore it provides measurement of magnetic field in three axes X, Y and Z. Modern magnetometers are often using specific circuits to lower the power consumption while the sensitivity should be unaffected [3].

1.3. Gyroscope

Using term gyroscope can be a little confusing, because originally the name gyroscope means a device with spinning disc whose orientation in space is unaffected by angle rotation of outer objects, holding the gyroscope. But this meaning of gyroscope is an angular velocity sensor, sometimes called angular accelerometer.

These angular velocity sensors are based on measurement of Coriolis force of a rotating object. A breakthrough in this technology came again with development of MEMS. MEMS are nowadays the best and possibly the only wise choice for all applications with size limitations. There are several types of microelectromechanical structures used as angular velocity sensors, so we'll focus on explanation of ST Microelectronics technology applied in Activ'Platform. Just like the accelerometer uses object of known mass to deduce acceleration from its displacement, an angular velocity sensors are using moving or rotating objects of known mass with devices for Coriolis force measurement. ST Microelectronics developed innovative solution, with just one driven mass, for angular velocity measurement of all three axes. The Mass inside sensor shown on Figure 4 consists of four basic elements labeled from M1 to M4. During the operation, they are moving inward and outward in horizontal plane, on a certain frequency. Red and yellow arrows are showing the impact of Coriolis force to rotation of three axes.

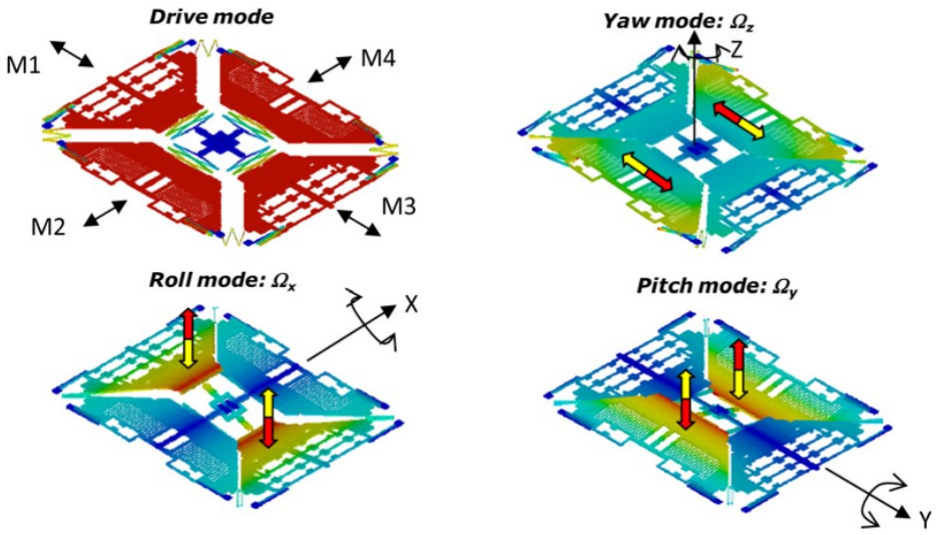

Figure 4 ST Microelectronics angular velocity sensor [4]

Figure shows the effects of three types of rotational movement, represented as pitch, roll and yaw. Each of them results in different motion of mass inside the sensor. During the rolling movement (rotation around X axis), elements M1 and M3 are moving in vertical plane and in opposite directions. If elements were moved in equal directions, applied movement would be linear shift in direction of Z axis and not the rolling around X axis. Pitch mode effect has the same principle applied on elements M2 and M4. Rotation around Z axis has no vertical effect, but results in horizontal movement of M2 and M4 in opposite direction. Thanks to differential character of capacitive sensing, the element's movement in equal direction (M2 and M4 or M1 and M3) will not result in the change of capacity. Therefore this type of sensor is able to reject linear acceleration, what makes it more resistant to shocks and vibrations.

Conventional products use three single driving structures, or one single plus one dual driving structure. Sometimes they are even operating on different frequencies, the mismatch between frequencies will cause noise while ST technology has a great advantage of noise reduction [2] [4].

1.4. Barometer

Purpose of atmospheric pressure measurement in biomedical sensor is to estimate the absolute altitude or altitude changes. Dependence between pressure and altitude is exponential, estimation can be done using formula (4). For more precise estimation of altitude, a temperature compensation (5) is generally recommended.

$$\boldsymbol{P_h = P_0 \cdot e^{-\frac{h}{h_0}}} \tag{4}$$

pressure at altitude h	$P_h[Pa]$
pressure at sea level	$P_0[Pa]$
altitude	$h[m]$
sea level altitude	$h_0[m]$

$$P_h = P_0 \cdot e^{-\frac{m \cdot g \cdot h}{k \cdot T}} \tag{5}$$

pressure at altitude h	$P_h[Pa]$	$\left[Pa \cdot \frac{\frac{kg \cdot m^2}{s^2}}{\frac{J}{K} \cdot K} = Pa \right]$
pressure at sea level	$P_0[Pa]$	
mass of the air molecule	$m[kg]$	
standard gravity	$g\left[\frac{m}{s^2}\right]$	
altitude	$h[m]$	
Boltzmann's constant	$k\left[\frac{J}{K}\right]$	$\left[\frac{J}{K} = \frac{kg \cdot m^2}{K \cdot s^2}\right]$
temperature	$T[K]$	

Pressure sensors are necessary part of a medical equipment. There are different types of sensors for different applications, but the main part of it is always a diaphragm. Diaphragm deflection caused by the pressure applied is measured, using available methods. Methods are still the same, including strain gauges, capacity sensor or piezoelectric sensor. Then a detailed construction depends on the character of measured environment. Diaphragm has two sides, so basically one side of diaphragm has to be exposed to a reference pressure and the other side to measured environment. Sensors of pressure that is significantly higher, than atmospheric pressure, can use atmospheric pressure as a reference, but in case of atmospheric pressure measurement, a vacuum chamber is used as a source of reference pressure. However creation of absolute vacuum in a sensor is not possible, hence the pressure level in a chamber has to be reasonably low, as it is well known that warming of pressure residues results in chamber expansion.

Since the evolution of semiconductors, pressure transducers can be made on single chip, where a part of the semiconductor material is used as a diaphragm. Barometric sensor producers emphasize especially factory calibration of sensors and endurance. Presuming that barometers will be exposed to adverse conditions such as humidity changes, sensors are often equipped with stainless chassis. Whenever the basic unit of pressure is Pascal, according to SI system, many clients and producers use different units such as Bar o PSI (pounds per square inch), overview of unit conversion is shown below [1] [5].

$1Pa = 0{,}145038 \cdot 10^{-3} PSI$

$1Bar = 14{,}5038\, PSI$

$1PSI = 6{,}89476 \cdot 10^{3} Pa$

$1Bar = 10^{5} Pa$

$1Bar = 10 mH_2O = 10^{4} mmH_2O$ [5]

2. Bluetooth technology basics

Bluetooth is technology that allow us transmit and receive data wirelessly between two or more devices, using one frequency band. This technology uses ISM (industrial scientific and medical) frequency band at approximately 2,4GHz. It is not easy to operate in this band, because almost every object can affect the transmission and wave propagation on that high frequencies is bad. But on the other side, there are no limitations and fees for ISM band in whole world, which is significant advantage for a developers. But also, it means that other technologies and wireless devices can use this frequency band as well. Bluetooth Company came with elegant solution of this issue. The carrier frequency of transmitter and receiver is changing during the transmission, therefore the interferences between devices are highly reduced. Bluetooth technology uses one device configured as a master and multiple devices that can be configured as slaves. Master controls the connection with slaves. After start of Bluetooth device it is master who is searching for other BT devices, this procedure is called scanning. Slaves are also called advertisers, they are able to transmit advertising to a scanner (master), who can start the connection [6].

2.1. Bluetooth Standard specifications

Precise frequency band for BT technology is from 2400 to 2483.5MHz. This band is divided into 79 channels of 1MHz width. Plus on the edges of BT freq. are two unused frequency intervals called guard band to comply „out of band" regulations for each country.

As we know, BT technology supports using of multiple devices in one band. The way to do this is called FHSS (frequency hopping spread spectrum). It is based on fast jumping, or hopping, between tight frequency channels using pseudo random pattern, which has to be known by receiver and also by transmitter. By transmitting on one random channel for very short time and then jumping to another one, BT technology avoids the risk of transmission interference. Standard of Bluetooth FHSS is 1600 Jumps per second.

Maximal theoretical speed of Bluetooth transmission is 1Mbit per second, but the real speed depends on speed of used transmitter and receiver. There are few basic categories to sort BT products. First and popularly discussed is the difference between Bluetooth Classic and Bluetooth Smart. Bluetooth Classic is the first technology developed by Bluetooth Company. We can meet with lot of final products using this technology in headphones, smart phones, laptops etc. Bluetooth Classic is the technology described by already mentioned parameters. But Bluetooth Smart technology starts to be popularly used nowadays. It can be found under name Bluetooth 4.0 and higher (Therefore Bluetooth 3.0 and lower is always BT classic). Common mistake is, that the name Bluetooth Smart is often swapped with the technology Bluetooth Low Energy. But the difference is that BT Smart is name of whole technology, containing BT Classic (called also BR, Basic rate) and BT Low Energy. And then it depends on BT Smart device, if it's using Classic and also LE or just one of them. These devices are called single mode devices, or dual mode devices according to a fact, that single mode device has just BR or just LE. But dual mode device consists of BR and LE together. This is just a simple example, because LE and BR are most commonly used BT technologies, but BT developed also other ones, like Enhanced data rate (EDR) to enhance maximum speed of BT to more than one Mbit per second. For applications where is no need

for huge data transmissions, Bluetooth Low Energy (single mode) device can be used, to lower the power consumption as much as possible [6] [7].

2.2. Low Energy specifications

Some Bluetooth Low Energy (BLE) specifications differs from Bluetooth Classic. Firstly, considering much lower data rate, there is no need to use so many channels for frequency hopping, so the BT frequency band is then divided to just 40 channels instead of 79. From which 37 channels are used for data transmission and 3 channels are used for advertising. Advertising is a procedure necessary for devices to find each other and will be better described below. BLE device from the look of link layer, can be described as a state machine with following states: standby, advertising, scanning, initiating and connection. One device can only scan (master procedure) or advertise (slave procedure) in one time.

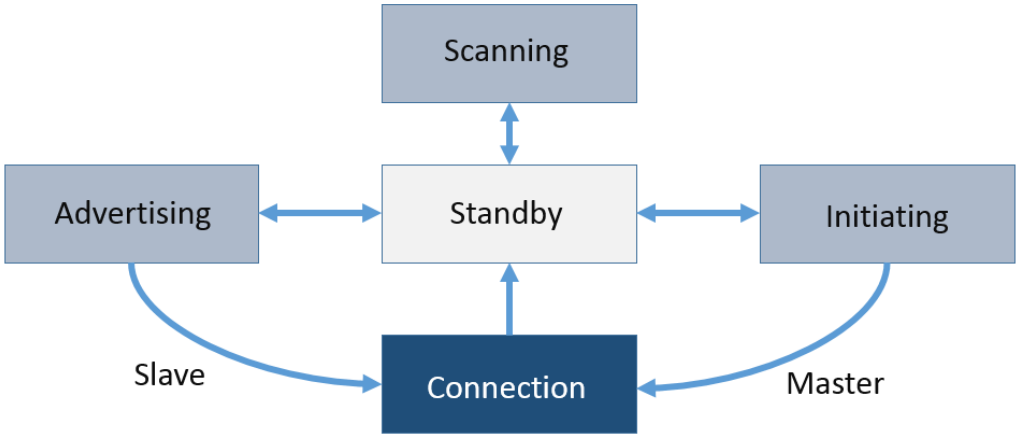

Figure 5 Link layer state machine

In a **standby state**, the device is not sending or receiving any packets, the standby state can be entered anytime.

In the **advertising state**, the device (called advertiser) is sending advertising packets and if the master is set to active scanning mode, it can also send responses and some additional info, when the master sends a request. The advertising state is usually started by setting the slave device „discoverable, connectable."

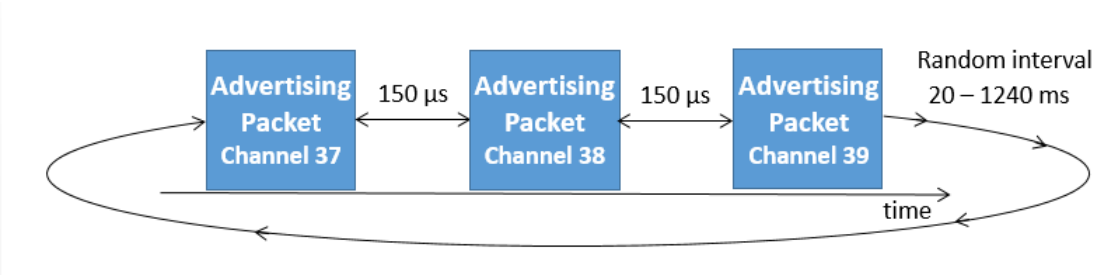

Figure 6 Advertising procedure

Advertising packets are sent by three. The latency between them is set to 150µs and latency between groups of three is random interval from 20 to 1240ms, to eliminate interferences of more advertising devices on three advertising channels.

Scanning state is a state in which the master is receiving advertising packets from advertising devices. In active scanning mode the scanner is listening to advertising packets and then is asking advertisers for more information, such as device friendly name, supported services, application data, etc.

Connection is established by master, after he sends the connection request packet with appropriate parameters. In this state, master is called initiator because he's initiating connection. Connection is considered open, if both devices continue in sending packets. Slave always has to send some response, when he receives a packet from master [7].

2.3. LE connection parameters

As mentioned before, the connection is established by sending connection request packet. The packed contains necessary information of connection parameters. The 3 most important parameters for Bluetooth Low Energy are, connection interval, slave latency and supervision timeout.

Connection interval is the time between two connection events. To save energy, BLE devices are not communicating all the time, but just in connection events. So bigger interval between those events will save more energy but decrease data rate. No matter if device has data to send, it has to wait until next connection event. The interval can be set from 7.5ms to 4sec. Therefore the maximal frequency of refreshing data in Bluetooth LE is 133.33Hz. In BLE113 configuration, the interval is set as multiple of 1.25ms, so the number for configuration can be from 6 to 3200.

Slave latency is an amount of connection events, that slave can skip to save energy. After he receives packet from master and he has no data to send, he doesn't have to send any packet as many times as it is defined with this parameter. It shall be an integer in the range of 0 to ((supervision timeout/conn. interval) - 1) and also less than 500.

Supervision timeout is interval between two connection events, before the connection is considered as lost. It shall be multiple of 10ms in range of 100ms to 32s and also larger than: (1+slave_latency) · connection interval.

If remote device is not able to set required parameters, it will send connection parameter update request to initiator or master (depends on connection status, these parameters can be also changed during connection) [7].

2.4. Generic Attribute Profile

GATT is BLE data protocol. It defines shape of data and possible operations with them. So it contains identification data, description, permissions how to manage data and in the end it contains data to be transmitted. GATT define the roles for devices called client and server. Client is usually the master and his task is to send requests to server which contains data. The server (usually slave) contains data stored in GATT database and provides them to client on request. Basic parts of GATT are services, characteristics and attributes. Let's start from attribute.

Attribute stays on the end of GATT and contains value, handle and type information. Handle is a 16bit value used to find attribute in protocol. And it is generated automatically after creation of GATT profile. Attribute value type information is written by UUID (universal unique identifier). They define the physical value of data stored in GATT. Hierarchy of attributes characteristics and services is shown on Figure 7.

Characteristic can be described as a different look on attribute, but it contains more data. Therefore characteristic is higher in the GATT protocol. Inside characteristic is a description (additional info), characteristic UUID and permissions. 16bit identifiers are defined as a standard of Bluetooth and can be found on their developer portal. UUID can be also 128bit and 128bit UUIDs have optional value for user while 16bit identifiers should be respected and not changed for other purposes than defined by Bluetooth Company. Permissions part contains information about access to attribute data. There are 4 possibilities: read, write, indicate and notify. Read and write is pretty clear. These permissions are written from the look of client. So usually master can read or write to remote server. While read or write procedure is controlled by client, notify and indicate enables server to transmit values itself. This is useful for real time measurements. Difference between indicate and notify is, that indicate is acknowledged. But in this case acknowledge in BLE takes more connection intervals, so it slows down data rate to half.

On the top of attribute profile is a **service**. Characteristics are sorted into the services and one service can have more characteristics. Service has also his own UUID. By listing in GATT profile (for example in Bluegiga BLE GUI) you can see more UUIDs. They are called declarations. Declarations are used to differ characteristics, services and other parts of GATT [6] [7].

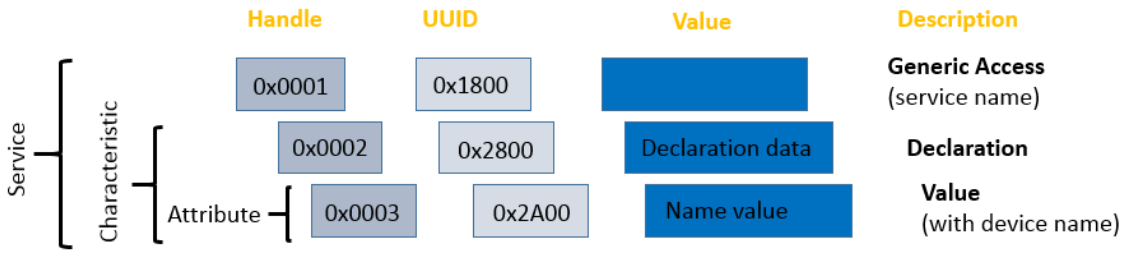

Figure 7 GATT structure

3. Activ'Platform overview

This device is now able to control and collect data from three modules, which means five sensors in total. An Analog devices accelerometer, barometer and module from ST Microelectronics containing accelerometer, magnetometer and gyroscope. For controlling whole unit, low consumption ARM based microcontroller is used. For Bluetooth communication was chosen Bluegiga BLE113, which is Bluetooth Low Energy single mode end product. Here is an overview with overall description of Activ'Platform basic components:

- **Accelerometer ADXL362**
- **Barometer MS5611**
- **Bluetooth module Bluegiga BLE113**
- **Inertial sensor LSM9DS0**
- **Microcontroller STM32L052**

Communication between sensors and microcontroller is established via SPI bus and BLE113 uses communication via 2 wire UART. The idea is to configure device and sensors according to setting, written in PC program for monitoring data. Then the configuration is sent to microcontroller via Bluetooth. Microcontroller, after receiving configuration data, does the configuration of all onboard sensors. Temporarily unused sensors are turned off to save energy. Activ´Platform is small device using lithium battery as a power supply. That's why the software and hardware for this device were developed with respect to low power consumption. Measured data are collected from chosen sensors and stored in packets, which are continuously transmitted via Bluetooth. On the site of PC, Bluegiga BLED112 dongle is used as Bluetooth master. The dongle is connected to computers USB port and act as a COM port. PC application for Windows is controlling the dongle communication, storage of data to text file and also shows measured values on real-time monitor.

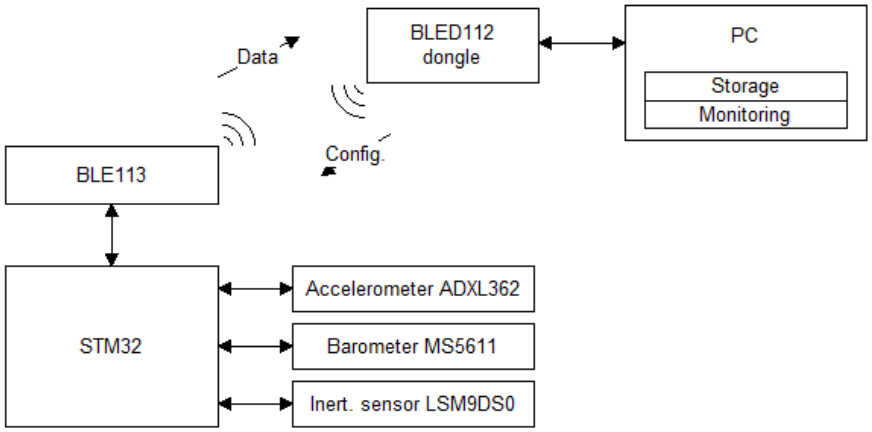

Figure 8 Activ'Platform and remote control

3.1. Conventional devices research results

The use of inertial sensors in many applications grown significantly with the evolution of MEMS. Also evolution of wireless technologies in free rent bands helped to a development of wireless battery powered inertial sensors. In medical systems these sensors can be used for human (muscle) activity observation, or fall detection. In digital imaging and image acquisition, inertial sensors are used in optical stabilizers. In automotive industry they are useful to estimate the location without presence of GPS signals. In mobile devices MEMS sensors allows control via gestures and checking the device orientation. Or animators are using them to record real movements that can be applied on models, programmed in PC. So inertial sensors are widely used, while solutions and sensor types are different. Sensors chosen for the Activ'Platform can provide even high resolution, sampling frequency and ultra-low power consumption (in different settings) compared to products available on the market. This gives Activ'Platform great advantage, because the firmware can be modified for relatively high performance or low power consumption. Of course every technology is facing to some limitations, specified at the end of this chapter, but still Activ'Platform hardware design holds on, compared to products available on the market. In the list below is a summary of different inertial sensors for medical application with their features.

X-IO X-IMU

- Sampling frequency range up to 512Hz (divisions of 1kHz)
- Accel. resolution range up to ±8g
- Magnet. resolution range up to ±8.1G
- Gyro. resolution range up to ±2000dps
- Wireless technology Bluetooth Classic

X-IO solution is using Bluetooth classic for wireless data transmission and has wide range of sampling frequency setting, which doesn't seems to be necessary, but that is just another way of sensor configuration. Advantage is SD card support, for long-term measurement. System is supplied by Li-Pol battery rechargeable from USB port and power consumption can grow up to 150mA due to selected data rate. SDK with examples for PC software is available [8].

X-IO X-BIMU

- Sampling frequency range up to 128Hz (256Hz without wireless)
- Accel. resolution range up to ±16g
- Gyro. resolution range up to ±2000dps
- Wireless technology 802.15.4 XBee

X-BIMU is a low power twin of X-IMU unit. Power consumption during wireless transmission is 25mA, which is much better, than in case of X-IMU unit. But this device faces to some important limitations. For example, magnetometer cannot be used with wireless transmission, because of its issues

with on board algorithms. And maximal sampling frequency also decreases when wireless transmission is turned on [9].

STT_IBS

- Sampling frequency range 125-250Hz
- Accel. resolution range Selectable: ±2g and ±8g
- Magnet. resolution range ±12G
- Gyro. resolution range ±2000dps
- Wireless technology Bluetooth 2.0

Advantage of this device is C# and C++ SDK that can be obtained from producer. But note that this SDK is implemented on s PC, changes of embedded software in the platform are not possible. Also using of Bluetooth 2.0 and relatively high sampling frequencies does not support power saving features. SD card is also not implemented. Producer does not state a resolution ranges of all sensors [10].

Inertia ProMove Mini

- Sampling frequency range 1kHz (accel. and gyro.), 100Hz (magn.), 25Hz (barom.)
- Accel. resolution range selectable: ±2g, ±4g, ±8g, ±16g
- Magnet. Resolution range ±49.12G
- Gyro. resolution range selectable: ±250, ±500, ±1000, ±2000dps
- Barometer range 260 -1260hPa
- Wireless technology Bluetooth Smart

This technology has a great advantage of Bluetooth Smart dual mode device. Which means that user has an option of low power transmission or high performance. This IMU has also a barometer, which is still not so common for conventional products. Device has 2GB of flash memory, USB port for wired data transmission and battery charging and offers four hours of battery life in full wireless streaming mode [11].

Cometa Wave Track

- Sampling frequency range 100Hz (quaternions), 256Hz (raw data)
- Wireless technology Custom solution

Cometa Company focus on wireless EMG and IMU sensor solutions since 2001. Their Wave Track system can offer probably smallest, inductive rechargeable, waterproof inertial sensor on the market including wireless data acquisition, internal memory and 6 hours of battery life. No software development kit can be obtained from the producer. This system does not allow sampling frequency or resolution settings, but interesting is the selectable output of quaternion data format [12].

Arduino 10 DOF MEMS IMU Sensor

- Maximum sampling freq. 3.2kHz (accel), 160Hz (magnet.)
- Accel. resolution range selectable: ±2g, ±4g, ±8g, ±16g
- Barometer range 300 -1100hPa
- Magnet. resolution range ±8G
- Gyro. resolution range ±2000dps

Although this is not a complex measurement solution, it is an interesting example for development, so it is included in the list. This board contains just inertial sensors, barometer and needs to be connected to a data acquisition unit. Here in research it is used as an example, why a complex solution was developed instead of buying some intermediate product. Nowadays, these product are very popular because of their ready to use hardware and software character. Single modules are using standard communication interfaces, so they can be assembled together into more complex devices. In short, we can say that for research use of device, which is not limited by size, speed and power consumptions, this can be a good choice. But assuming that our hardware, has to be assembled in one sensor with size limitations, must be able to achieve the lowest possible power consumption and either high performance, this kind of solution would not be really efficient for our application [13].

Summary

Comparing these technologies, it was found, that every solution faces to some technical limitations. Only one system shown, has a low power mode option, while it is still using wireless interface. This is thanks to dual mode Bluetooth Smart device. Resolution ranges of devices in the list are mostly the same, as they match the requirements of IMU. User selectable sampling frequency in whole range does not seems to be se desirable, because ultra-low power sensors have usually one low power oscillator and fixed dividers and so user can choose sampling frequency from several possible values. In other words, complexity does not make a device better.

These devices usually run on Li-Pol rechargeable batteries. Activ'Platform is using coin cell CR battery. And for development of commercial solution, it is always easier to select approved standard type battery, because these types satisfies international standards even for using in healthcare. Since the material of most batteries is toxic, another advantage of coin cells is a steel package, compared to most of Li-Pol batteries, which doesn't have solid package at all. And finally self-discharge. When a device is used as a fall detector, battery requirements are high capacity and long life. Lithium batteries have lowest self-discharge of known types.

Some of the solutions available on market provides software development kits, so the user can program his own algorithms on a PC. But access to the embedded code is always forbidden as it is company know how. So when the control of data acquisition and data processing is required in the embedded system, we have no other choice but develop our own hardware and embedded software.

3.2. Hardware prototype

The development of Activ'Platform was started from BLE communication, so first hardware used, was BLE113 development kit, running via BGSCRIPT and then via BGAPI after connection with ATMEGA microcontroller. One of old versions of Vigifall device vas used for this prototype. After the BLE communication and BGAPI was running, turn came to sensor drivers development. During the working internship in Vigilio S.A. in Grenoble, a hardware prototype of Active'Platform was built, using STM32L053 discovery board, barometric sensor soldered on evaluation board, LSM9DS0 IMU breakout board and ADXL362 accelerometer mounted on board of Vigifall fall detection sensor. This hardware prototype was used to modify proposed design of Activ'Platform schematic. It was found during the development that some peripherals has to be set differently. Every sensor has a different principle of operation and sometimes their methods of data acquisition are not clear. For example ST Microelectronics IMU LSM9DS0 contains three sensors in one package. They are using one SPI bus with two slave selects pins and three interrupt outputs. Naturally it seems like every sensor has one interrupt pin, but it doesn't. One sensor works as a standalone device and two other are using just one interrupt output together. Next example is BLE113 control. Documentation overview of the module says, that BGAPI can be controlled just via UART without flow control. But then it was found, that additional pin for wakeup is required. UART cannot wake BLE113 even after reset. These are examples, why the original schematic was improved after thorough study of device components.

After all drivers were prepared and tested, a prototype board was designed. Because of small dimension and character of components, the prototype was assembled in the factory. In a Table 1 below is a list of MCU inputs and outputs assigned to pins and ports of used sensors. This setting was tested on development board and the applied on prototype design.

Prototype board is a four layer PCB of a thickness 0.8mm in a shape of triangle. The board is supplied from a coin cell CR2032 battery or CR2240 battery. Battery is mounted on a separated board and connection between the boards is arranged by miniature SMD four pin connector with contact spacing of 1.25mm. Which type of battery will be used depends on a battery grip board, connected to a sensor. As a power solution during development, a wire was soldered right to supply input, because SWD interface used for debug connection does not have a supply wire. However JTAG debuggers do have a 5V power supply pin and Activ'Platform has a regulator that can handle 5V, so the debugger supply was used to save batteries during development process.

Table 1 STM32 IO connections

Sensor pin:	STM32 pin:	Note:
ADXL362		
MOSI	PA7	SPI1
MISO	PA6	SPI1
CLK	PA5	SPI1
CS	PB10	
INT1	PB1	
INT2		unused (leave unconnected)
VccAcc	PB2	
MS5611		
MOSI	PA7	SPI1
MISO	PA6	SPI1
CLK	PA5	SPI1
CS	PB12	
Vcc	PA8	
LSM9DS0		
MOSI	B15	SPI2
MISO	B14	SPI2
CLK	B13	SPI2
CSG	A15	
INTG	NC	not used, is used to different interrupts than we need
DRDY	PB9	Gyroscope FIFO interrupt
DENG	NC	not used and not described in datasheet (no significant function)
INT1X	NC	not used, is used for different interrupts than needed
INT2X	PB8	Accelero FIFO and Magnetometer data ready interrupts
Vcc	PB3	
CSX	PA12	
BLE113		
TX	PA9	
RX	PA10	
V_EN	PA11	
P_0.0 wakeup	PB4	

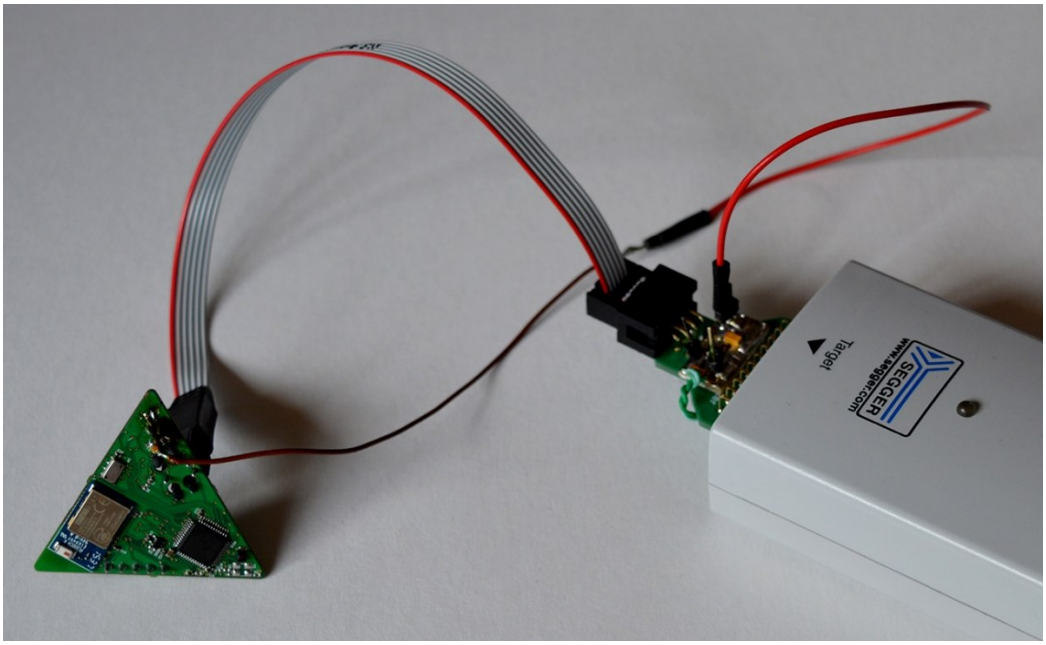

Figure 9 Activ'Platform Prototype (connection with J-Link)

Debug interface is connected to the board via 6pin needle adapter, produced by SEGGER. Using this adapter, a board does not require additional connectors for debug. Needle adapter is attached to the board interface with plastic clips and connected with golden spring contacts, directly to arranged pads. Brown (and red) wire is a power supply, connected to voltage regulator assembled on JTAG to SWD adapter. This adapter was modified for prototyping board to save batteries during development. Red arrows on Figure 10 Asembled prototype of Activ'Platform the orientation of axes.

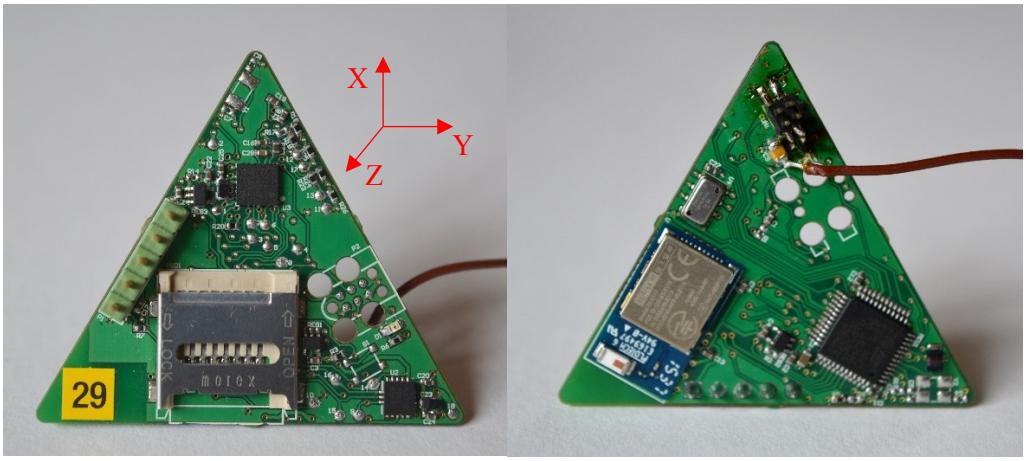

Figure 10 Asembled prototype of Activ'Platform

4. Ble113

BLE113 is Bluetooth Low Energy single mode end product device from company Bluegiga. End product in Bluetooth terminology means, that device consist of all necessary things to run with BT. Single mode product means, that even if device is Bluetooth Smart (4.0) it works only in low energy mode. BLE113 uses specific tool for programming and configuration. The controlling microcomputer on this module is Texas Instruments CC2540, so as programming hardware the Texas Instruments CC Debugger is used. Then module can be controlled by several ways.

- **BGSCRIPT** is simple language developed by Bluegiga technologies. It contains simple commands to handle Bluetooth, module GPIOs and other peripherals. Disadvantage is, that this language is not very flexible and cannot be used in more complex solutions.
- **BGAPI** is opportunity to control module via commands send through UART. List of commands is the same as in BGSCRIPT language, but when nothing happens, the module is in sleep mode and more sophisticated operations as well as controlling BLE113 are done with external microcontroller.
- **CSDK**. Bluegiga also provides C language Software Developer Kit for users, who wants to improve embedded software. But SDK is provided only on demand and requires additional tool for code writing. IAR Embedded Workbench for 8051 core.

A controlling via BGAPI was chosen for this project, therefore STM32 microcontroller is sending commands to BLE113 via UART. Unfortunately with increasing throughput, BGAPI is not able to work properly because of command/response nature of control. Waiting for the response requires some time and also raw commands containing data are unnecessarily long even without data, hence UART is fulfilled with kilobytes of needless data. Generally, when module works as a cable replacement, there is no need for complex command structure. An optimization could be done by creation of custom firmware for BLE113. The SDK database was already obtained from Bluegiga company, but improving of embedded software in Bluetooth module is for now just one subject to improve in the future. For now UART has to be set to quite high baud rate, to prevent loss of data.

According to notes from Bluegiga technologies. Maximal achievable throughput of this module is approx. 60kbit/s. That means if we are using minimal connection intervals 7.5ms, BLE113 can send 450bits per one interval which is 56.25 Bytes. After tests, with a development board, it was found, that achievable rate is little higher without an error (60B of data can be sent in one connection interval). Hardware configuration of the module offers three mode of throughput. Performance, balanced and power, explained in section 4.2 GATT configuration. To obtain maximum speed, Bluegiga module is set to performance mode and it is able to send three GATT characteristics with 20 octets of data during one connection interval [14] [7].

4.1. BLE113 hardware configuration

The Bluegiga module uses specific XML files, which allows us to easily configure BLE113 module, using parts of code defined by company firmware. In case of using BGAPI to control module, configuration can be done with Bluegiga BLE SW Update Tool, provided by Bluegiga company for free. This software works with several files that can be edited with notepad or another XML editor, including file hardware.xml and gatt.xml. File gatt.xml serves to configure GATT profiles for BLE113 module and will be described in next chapter. If using BGAPI, it is necessary to set specific hardware configuration for the module. XML files are matched together and downloaded into BLE113 via Texas Instruments CC Debugger. Activ'Platform hardware has five holes implemented for BLE113 programming (see part BLE113_PROG part on the board).

BGAPI is using two wire UART, with baud rate adjustable up to 2Mbit per second. Because two wire UART has only Rx and Tx wires and no flow control implemented, direct memory access (DMA) must be set. This is done by enabling packet mode for UART. It is also necessary to set wakeup pin. If there is low level on wakeup pin, the module is sleeping and does not receives BGAPI commands. High level sets external interrupt to wake up module from the sleep mode. When wakeup pin is set high, module generates hardware_io_port_status event and then desired command can be send via BGAPI, but wakeup pin must be held high, at least until the first byte of command response is received. The screenshot of Activ'Platform hardware configuration XML file is on Figure 11. All XML configuration strings for BLE113 module are described in document Bluetooth Smart Module Configuration Guide included in this project and it is also available on Bluegiga Technologies websites after user registration [15] [7].

```
<?xml version="1.0" encoding="UTF-8"?>
- <hardware>
    <sleeposc ppm="30" enable="true"/>
    <usb enable="false" endpoint="none"/>
    <txpower bias="5" power="15"/>
    <usart endpoint="api" mode="packet" flow="false" baud="230400" alternate="1" channel="1"/>
    <wakeup_pin enable="true" pin="0" port="0"/>
    <port pull="down" tristatemask="0" index="0"/>
    <port pull="down" tristatemask="0" index="1"/>
    <port pull="down" tristatemask="0" index="2"/>
    <pmux regulator_pin="7"/>
</hardware>
```

Figure 11 HW configuration XML file

4.2. GATT configuration

As mentioned before, BLE is using attribute profile to sort data. Therefore the only way to transmit data is to write them to GATT database. GATT configuration is written in gatt.xml and after program compilation, Bluegiga BLE SW Update Tool creates text file with list of configured attributes and their automatically generated handles. Handle is 16bit number for attribute identification. Firstly and before the configuration itself, we need to know, what kind of data will be transferred through BLE.

Bluetooth Low Energy is not developed to transfer huge amount of data with high speed and therefore the communication works specifically to save energy. However BLE can operate in three modes according to purpose of using the module. The modes are called Performance Balanced and Power. Power mode optimizes the energy consumption by sending just one packet during connection interval. But even if this setting minimizes the power it limits throughput. The performance mode maximizes throughput by loading new packets into transmission buffer and sending them as soon as the previous packets were successfully transmitted. But this setting increases power consumption. Balanced mode is trying to achieve the best throughput and also saves energy by sending just packets that fits into 128B transmission buffer. But not just this setting affects power consumption. The module consumes most of the energy while transmitting, so it depends mainly on how often module is transmitting. This property is set by connection interval and other connection parameters. Purpose of the Activ'Platform, is to achieve highest possible speed to transmit data from as many sensors as possible, with sufficient sampling frequency and resolution, but also to have possibility to limit the data transfers and save energy. So during the development of high data rate firmware, ability of low power consumption modes is remembered. Basically low power depends on used hardware, hence the hardware is designed to send data intervals and use the time between intervals to save energy. The same technique can be applied on MCU, but firstly it was necessary to develop working firmware.

Ideology of BLE is to have data in attribute database. They are well sorted and accessible, but real access to data in GATT needs some time. Which means that if we want to send lot of different data values stored in different characteristics, we need time to access them. This is very inefficient for our solution, because Activ'Platform measures lot of different data that should be stored in different characteristics and services. For example: accelerometer should have its service with characteristics for each axis as well as barometer or thermometer should have their own services. Unfortunately accessing (read/write) to all those characteristics takes too much time and it is impossible to write more than three characteristics during one connection interval, if we are using shortest connection interval of 7,5ms. This problem is properly described in chapter 7.

Because it is impossible to create characteristic, according to BLE recommendations. A custom characteristics were created (more information in section 7.3). When a developer is not able to set UUID from known UUID's in BT database, 128bit UUID has to be chosen. There are some free UUID generators on the internet, considering that with 128bit random value, a risk of non-unique identifier is negligible. However the final GATT configuration for Activ'Platform is described in Table 2 and guide, how to configure GATT with all parameters is written in document Bluetooth Profile Toolkit Developer Guide. Basically the tab shows us structure of GATT used for Activ'Platform and this structure is programmed to gatt.xml file which then configures GATT in module itself [7] [16] [17].

Table 2 Activ'Platform GATT configuration

service	Handle	UUID	characteristic	Data Size	Permissions
Generic Access Profile	-	0x1800	-	-	-
	0x03	0x2A00	name (senddatademo)	-	read
	0x05	0x2A01	Appearance	2B	read
ActivPlatformData	-	128bit	-	-	-
	0x08	128bit	Data1	20B	notify
	0x0C	128bit	Data2	20B	notify
	0x10	128bit	Data3	20B	notify
ActivePlatformConfig	-	128bit	-	-	-
	0x15	128bit	configChar	5B	read/write

128bit UUID's are user defined.

Reasons to create customized GATT were explained, but let's concentrate on the structure itself. Two additional services were created. First service is default and necessary service called Generic Access Profile, because it contains GAP information. Also device name can be written by user into this service and it will appear as a result of active scanning. First custom service is a data storage. Every byte transmitted from Activ'Platform is written here, to one of three characteristics. Data size of each characteristic is limited to 20B, since it is a longest Write message achievable with BGAPI. Handles were automatically generated and all permissions are set to notify, hence BT won't send acknowledgment messages to every received packet. Notice that every custom characteristic or service has the 128 UUID. Third service is a configuration service. To see how configuration bytes looks like, see the section 7.1. Five bytes represents configuration message of three parts. First part says which sensors are turned on or off. Second part is used to determine sampling frequency and third part sets the resolution ranges. Permissions of configuration characteristic are read and write, since the characteristic can be read or either written by remote device.

4.3. BGAPI and commands

One of ways to control BLE113 is by BGAPI, controlling the module with commands sent via UART. Module is configured to support two wire UART. Microcontroller inside the module has direct memory access, so to control received message's data it needs just one more byte with information about message length. This byte is called payload and it is always on start of the transmitted message (command). There are three types of messages. Command, response and event. Command sends data to

module from external microcontroller. The module will proceed the command, and send back response message with additional information about the command execution. If the command contain some parameters, they are usually written in response to make sure that they were correctly set. Also each response has two bytes with error code. If they are all zero, no problem occurred. If not, the value inside specifies the problem. List of error codes and their meanings is written in API Reference document. BLE113 event is usually not caused by external microcontroller but some events are also sent instead of response. Good example can be reset command, because after reset, the system generates boot event and so there is no need to send extra response after reset command.

Each message has first four bytes called packet header. The packet header specifies message type, payload, command class and ID. Message type says if the message is command, response or event. Payload length byte contains number of bytes in message without packet header and first payload byte (the one for DMA control). Class divides commands into the groups according to their purpose such as connection, system, GATT, hardware etc. And finally ID specifies the command or event itself.

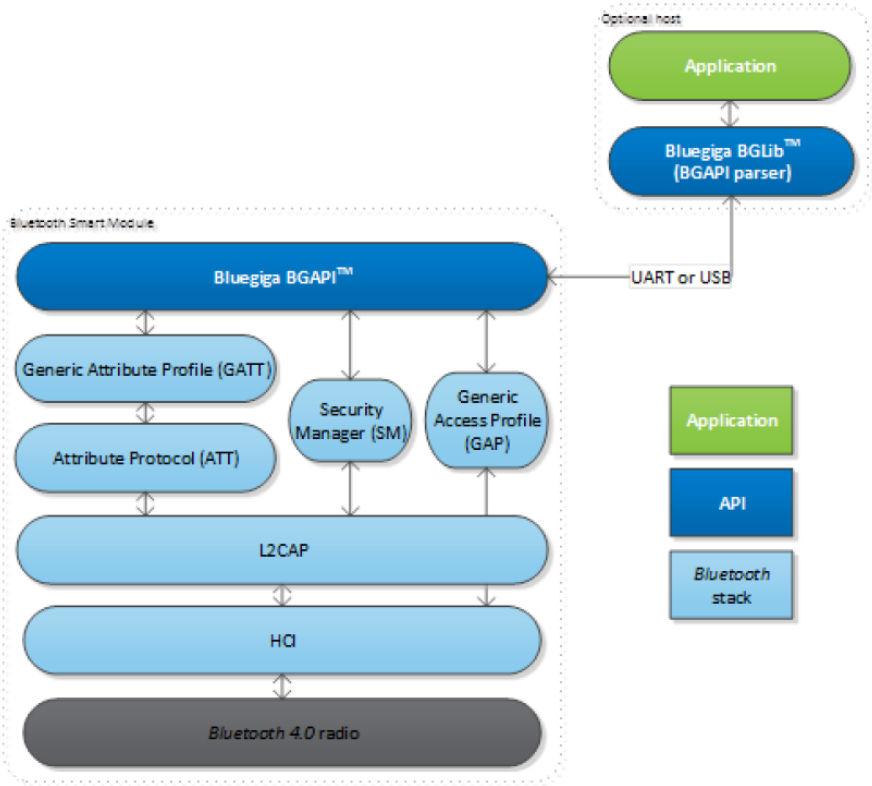

Figure 12 BGAPI layer [7]

So after the start of whole device, BLE113 is in sleep mode. Microcontroller sets module's wakeup pin to logic high and after this, the module sends event via UART. Then the MCU will send command to configure module's advertising state. Setting it discoverable and connectable means that the module will start sending advertising packets. Advertising is running until the connection is established or until it is turned off. Connection is established by remote master, by sending connection request packet. After this, the module will send event via UART, telling us it's connected. And then we are able to transmit data with Bluetooth LE. The connection between master and slave happens every

connection interval, when slave has new data. Data are stored in GATT, microcontroller is writing them there with Write commands, sent via UART. If master and slave are disconnected, BLE113 generates disconnect event, to inform microcontroller, it cannot send data now. After disconnection event, whole procedure starting from „set advertising mode" is repeated.

Figure 13 shows principle of BGAPI control and structure of packet header. First byte sent via UART tells if message is command, event or response. Second byte specifies the message payload (number of bytes in message without message header). Third byte selects class of the command. Class is a number, expressing character of BLE113 layers, processing the message. Last byte of header (together with ID byte) specify exact command, response or event that is executed.

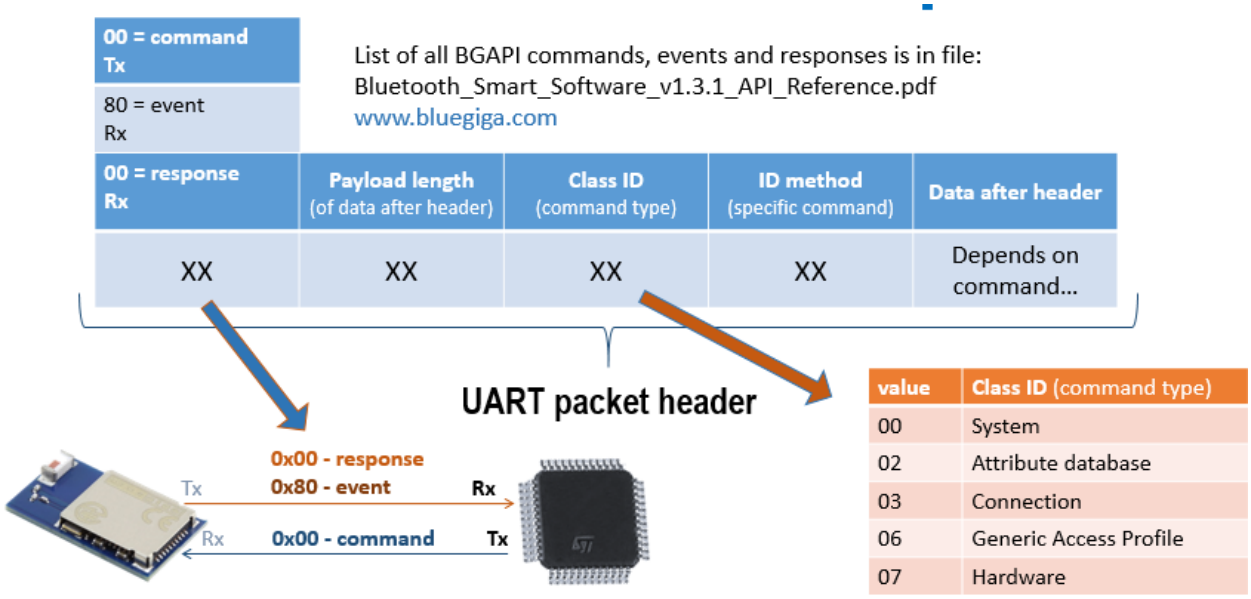

Figure 13 BGAPI message structure

The list below explains some basic commands, responses and events, used for module operation. BLED112 dongle drive, connected to remote PC USB, is using the same API, since BLED112 dongle has the same MCU. So don't let you to be confused if some command mentioned is a master command although the BLE113 module mounted on Activ'Platform is a slave. Please note, that just packed headers are listed, with a simple message explanation, because of better understanding of what is in the Activ'Platform firmware. Payload shown in a packet header is a minimal payload of a message and will increase with increasing amount of data in the message. Response packet headers are the same as appropriate command headers, except the payload byte. That's why response structure is also not present in the list. Responses can be found in API reference right under appropriate command documentation, what makes them easy to find. Also purpose of this section is to document, which commands were used in the firmware, not to transcribe API reference. For a whole message structure, error codes and meaning of other additional values, please refer to BGAPI Reference [7].

Commands/responses

- **Set Mode (advertiser)**
  By calling of this command and setting BLE113 as discoverable and connectable, an advertising mode is selected.

Type	Payload	Class	ID
0x00	0x02	0x06	0x01

  Response: returns zero value if the advertising mode has been entered. Non-zero value is an error code.

- **Set Mode (scanner)**
  By calling of this command and setting BLED112, a scanning mode is selected.

Type	Payload	Class	ID
0x00	0x01	0x06	0x01

  Response: returns zero if the scanning mode is started. Non-zero value is an error code.

- **Connect Direct (scanner)**
  This command shall be used to initiate connection between master and remote slave. Command requires setting of connection parameters referred in section 2.3 and the remote device's address.

Type	Payload	Class	ID
0x00	0x0F	0x06	0x03

  Response: returns zero if the connection was successful. Non-zero value is an error code.
- **Write (to GATT)**
  This command is used for writing attributes to attribute database. 20byte can be sent in a single Write command.

Type	Payload	Class	ID
0x00	0x04	0x02	0x00

  Response: returns zero if the attribute was successfully written. Non-zero value is an error code.

Events

- **Attribute Value (server)**
  Is generated by GATT server, when he receives a value from remote client.

Type	Payload	Class	ID
0x80	0x07	0x02	0x00

- **Attribute Value (client)**
  Is generated by GATT client, when he receives a value has passed from GATT server to client. Usually generated after Read by Handle command.

Type	Payload	Class	ID
0x80	0x05	0x04	0x05

- **Disconnected**

  When BLE is disconnected. Contains a reason of disconnection event.

Type	Payload	Class	ID
0x80	0x03	0x03	0x04

- **Connection status**

  Usually processed after the connection is initiated. Data after header contain connection parameters explained in section 2.3 of this document.

Type	Payload	Class	ID
0x80	0x10	0x03	0x00

5. STM32L05

STM32L05 is ARM core microcontroller optimized for ultra-low energy consumption. As a brain of Activ'Platform it controls all device peripherals via SPI and UART. As a microcontroller designed for low consumption it needs specific configuration. All peripherals inside controller can be turned on or off by switching their clock. For example if there is no use for port B in our application, we can turn it off and save current, which would supply the port B circuits. The same thing works for SPI UART and other peripherals and they are turned off by default. However application of the low power MCU's has a great advantage of saving the energy, even a programming of simple firmware with no respect to low consumption is more difficult compared to conventional MCU. Things that must be kept on mind during embedded software development on low power MCU are described in this section.

5.1. Setting the programming interface

Embedded software was developed in Keil uVision 5. Keil is a professional environment for embedded software development. Keil provides a freeware licence with 32kB code size limitation. This code size is sufficient for approx. 23kB of Activ'Platform code size.

For every embedded software project, it is necessary to have documentation and libraries for used MCU. Because there are too many companies and different MCU's, software environment cannot contain libraries for every type, hence they must be downloaded from internet database. Keil solved this with a feature called Pack Installer, which is available directly from Keil environment. Doesn't matter if user is going to change old project or create new a one, download of appropriate software pack is necessary. So the pack for STM32L0XX has to be installed.

After the software pack installation and project creation, project options must be configured for proper operation of MCU and debugger. In this project J-Link EDU tool from SEGGER Company was used as a programmer and debugger. SEGGER requires a basic drivers for operation. One of tools automatically installed with those drivers is J-Link Commander. JTAG standard interface provides 5V supply as mentioned in section 3.2, which was used to power the prototype. Power must be enabled by typing „power on" to J-Link Commander. Before setting of project options, command line can be used to check, if the interface between J-Link and MCU is running. To do that, type „connect" to the command line and follow the instructions. Required parameters are Cortex M0 device, SWD interface and rest can be left default [18].

When our MCU is powered and the connection between J-Link works, we can continue with project options configuration in Keil, by selecting Project and Options for Target or using Alt+F7 shortcut. In the first Tab (Device), selection of STM32L052C8 is required. In the second tab (Target) on Figure 14 is a setting of addresses, where the program will be stored. There are three settings of STM32 bootloader, two addresses in FLASH memory and one in SRAM are supported. Mostly used Boot setting is from the address 0x8000000 with size 0x10000. Since this setting is done with a hardware pull down resistor, be sure to type correct address and size in this tab [19].

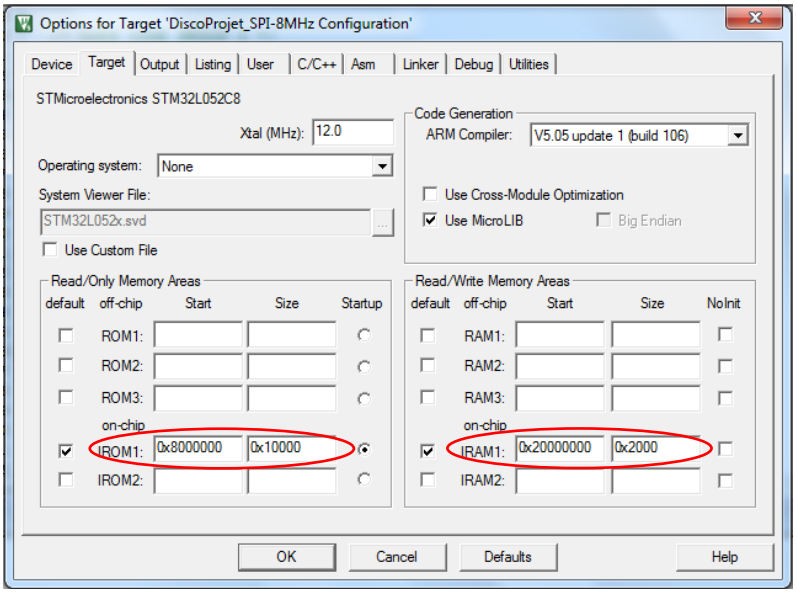

Figure 14 Setting the FLASH memory adress in target options

Next necessary setting is in tab C/C++. When the code is translated to assembler language, MCU must store lot of information for debugger. If debug mode is not needed, this code is not necessary and MCU saves lot of memory space. In the situation when debug mode is required and code is too large, higher **optimization level** can be set, to disable access of debugger to lower program layers. Therefore some memory is stored and debug mode is supported in higher program levels. But from the start make sure that optimization level is set to zero, because higher optimization level disables debug mode in most HAL drivers. Also HAL driver definition must set for correct type of MCU.

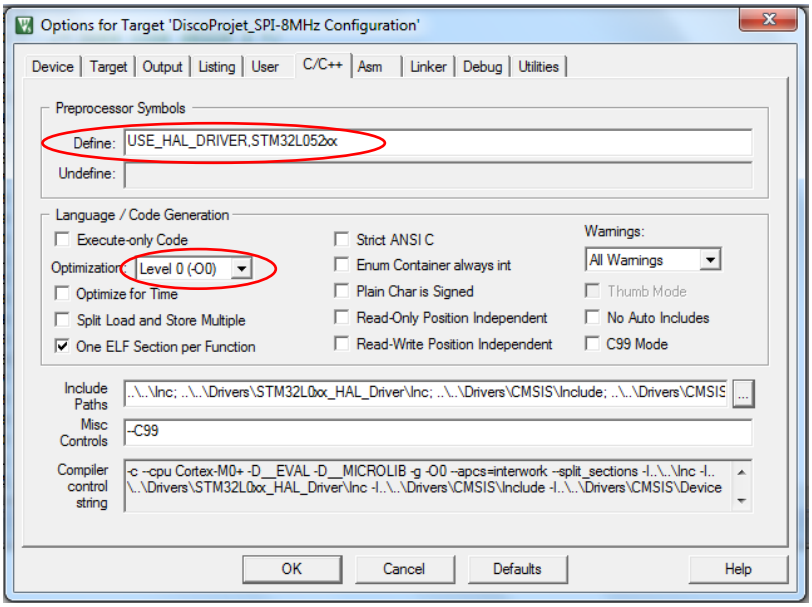

Figure 15 Setting the HAL drives and optimization level

Finally a device and interface for program and debug can be set. In a debug tab, shown on Figure 16, choose the correct device (J-Link) instead of simulator and go to the settings.

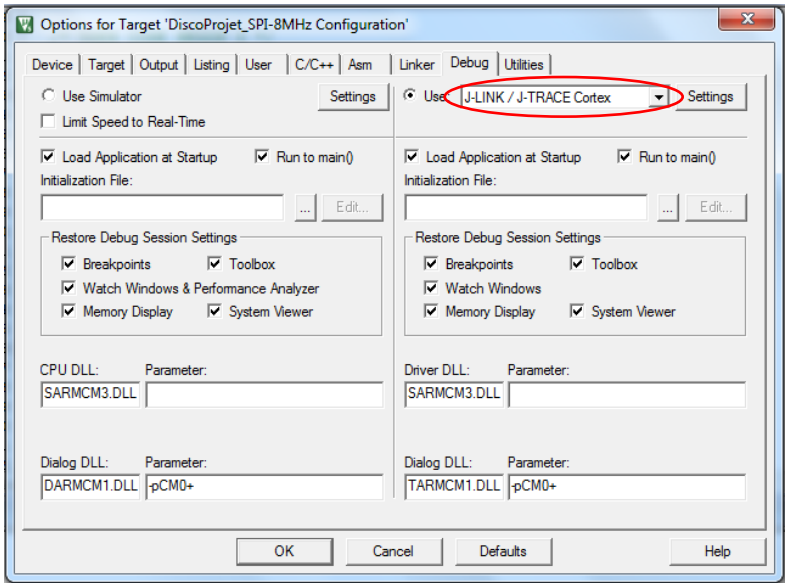

Figure 16 Programmer/debugger setting

If the J-link is connected, a serial number and device name appears in a group box (Figure 17). Default interface is JTAG. It has to be changed to SWD. Automatic detection of clock is not recommended, because it was found that it can run relatively slow, compared with fixed setting. Very important is to set „Download to Flash option," since the flash is set as a program memory. When the MCU is connected to J-Link, a device appears in SW Device group box. Additional specification of MCU type (STM32L052C8) can be required by pop up window.

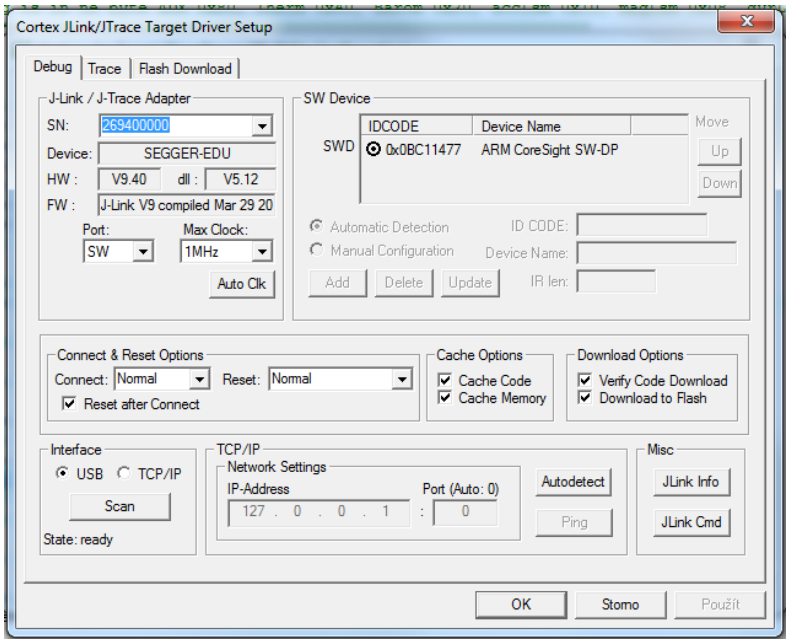

Figure 17 Interface settings

5.2. Initialization of hardware and peripherals

There are two ways to configure STM32L05 hardware and peripherals. First is a direct modification of MCU registers and the second is to use HAL drivers. When we see the configuration code programmed with use of HAL drivers, the code is easier to understand, because it doesn't use just numbers and register values. But also HAL drivers have some drawbacks. They usually use structures to store configuration settings before the configuration itself. When some variable of the structure isn't set, it can be replaced by default value, hence it is necessary to set whole structure before calling of HAL configuration function, or the additional setting will be necessary for correct operation. For example the low consumption microcontrollers have opportunity to set speed of GPIO. And if we are setting GPIO speed in STM32 using HAL driver, the driver is not able to set highest speed. That means, the driver does not allow us to use serial communication with high speed otherwise we will set GPIO speed directly by writing registers. Exploring the internet and trying HAL drivers, it was found that this is not only weakness of those drivers and using them should be always revised.

STM microelectronics provides software STM Cube that allows us to generate configuration code, but the code is generated to software's main file and it is using HAL drivers. After some experiences with STM Cube it was decided to write configuration file manually. HAL drivers are used just for parts with approved functionality. Activ'Platform configuration file is separated from rest of code for better portability (io_configuration.c).

C functions in main.c

- **void PeripheralSetting()**

Calling this function does all the basic hardware and peripheral initialization. To change parameters of hardware setting, io_configuration.c file has to be modified. To configure the STM32 hardware, combination of HAL drivers and SFR's is used, because even if HAL driver do not offer all necessary settings against SFR's, but working with HAL drivers is easier and the code looks more organized.

- **SPI_HandleTypeDef GetSpi1()**

Or SPI_HandleTypeDef GetSpi2() returns SPI handle structure of configured SPI bus. This structure is an input parameter for every function using SPI drivers. Because each sensor driver requires the operation of SPI, it is necessary to get this structure before sensor setting and then use it in drive initialization functions.

- **uint16_t ConnIntChoose(uint16_t freq, uint8_t onoff)**

Input parameters of the function are freq and onoff, received from configuration packet. Function returns 16bit value for timer 2 period setting. Next program step should be calling of Data Amounts function and Timer2Init.

- **void DataAmounts(uint16_t freq)**

This function sets global variables adxCBamount, accCBamunt and gyrCBamount, telling us how many values must be read from ring buffers during one connection interval. Input parameter freq is still the same parameter used in configuration packet. Amount of values sent during one connection interval can be set just for accelerometer ADXL362 and LSM9DS0's accelerometer and gyroscope. Other sensors are not using higher than maximal connection frequency.

- **void Timer2Init(uint16_t freq)**

During Interrupt sequence routines of timer 2, measured data are written to GATT. Timer 2 interval is one third connection interval length, because of ability to send three characteristics in one connection interval (three write to GATT commands).

6. Sensors

Activ'Platform collects data from three separated sensors. Barometer MS5611, accelerometer ADXL362 and inertial sensor LSM9DS0. They're all communicating with microcontroller via SPI interface on two separated busses. Barometer and Accelerometer are connected to SPI2 while LSM9DS0 is on SPI1.

6.1. Barometer MS5611

Barometer MS5611 is able to measure pressure up to maximal precision of 0,012hPa, altitude with 10cm precision and ADC output value is in 24bit resolution. The sensor is designed for low energy consumption and works on demand. That means if the sensor is not activated by some action of SPI, it is in sleep mode. Fastest conversion time of sensor is 1ms, but conversion time depends on ADC precision. Conversion time with highest precision takes approximately 10ms. And higher precision also increases power consumption.

MS5611 is calibrated from the factory and contains PROM memory with calibration data. They are used to calculate precise pressure value from measurement data. Standard procedure for measurement according to datasheet is to turn on the sensor. Then read 16 byte PROM memory and store coefficients. The memory is read by two bytes, coefficients are 16bit integers [20].

After storage of coefficients, the barometer is ready to use. ADC value is obtained by reading its register. It is the same command as for reading PROM but for different addresses. The sensor has five registers for temperature and five for pressure. Each of those five registers are for different precision or OSR (oversampling rate, because precision of ADC depends on precision time). When command for reading register is sent, controller has to wait some time to let sensor make the conversion. Conversion time for individual OSR is specified in datasheet. After that, result is sent back to microcontroller where calculation of precise pressure is done. The method how to obtain temperature compensated pressure and how to calculate it with calibration data is shown on Figure 19 Pressure calculation procedure . C language implementation of this code is shown on figure below. This code sample is taken directly from Activ'Platform barometer driver.

```
//************************************************
//          Compensated presure calculation
// According to datasheet MS5611 page 7 (diagram)
//************************************************
signed int CompensatedPressureCalc(uint32_t tempD2, uint32_t pressD1)
{
  signed int dt, temp, pressure;
  long double sense, offset;

  dt = tempD2 - ((int)regC[5]*256); //Difference between actual and reference temperature
  temp = 2000+(dt*((double)regC[6]/(8388608))); //Actual temperature with 0,01°C resolution
  offset = (double)regC[2]*(65536) + ((double)regC[4]*dt)/(128); //Offset at actual temperature
  sense = ((int)regC[1])*(32768) + ((double)(regC[3])*dt)/(256); //Sensitivity at actual temperature

  pressure = ((((pressD1 * sense)/(2097152)) - offset)/(32768)); //Temerature compensated pressure 0,01hPa resolutior
  // (example) value 100009 = 1000,09 hPa
  return pressure;
}
```

Figure 18 Temperature compensated pressure calculation code

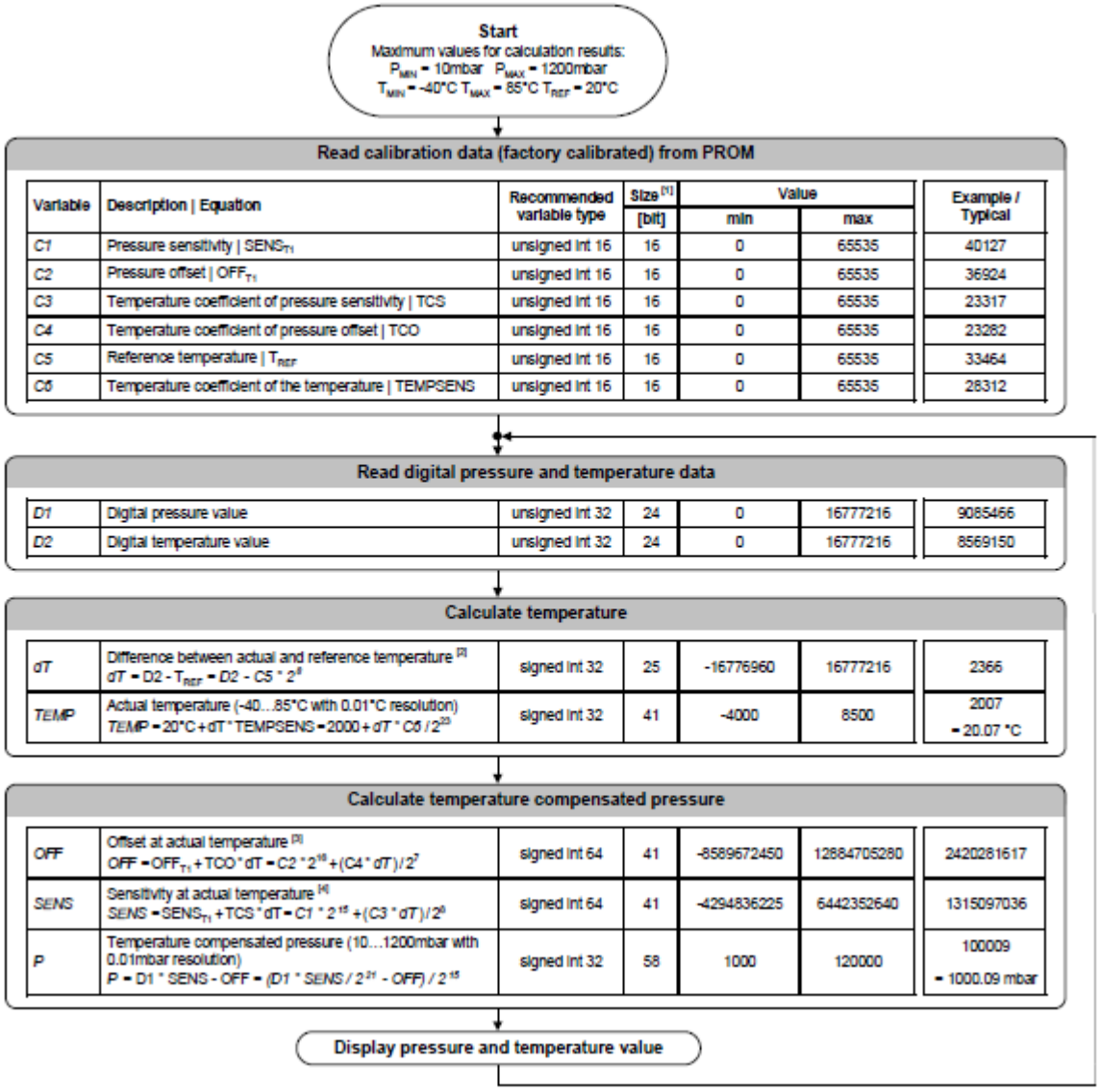

Figure 19 Pressure calculation procedure [20]

C functions

- **void Power_Barometer(bool onoff)**

This function enables the power, according to input parameter of Boolean type. If bool onoff is false, then the power is disabled.

- **void InitBarometer(SPI_HandleTypeDef *hspiInit)**

Initialization of barometer specifies the SPI bus used for communication, provides reset with necessary wakeup delay due to datasheet recommendations and reads internal PROM memory. PROM memory in this sensor contains necessary factory calibration coefficients used for pressure calculation. hspiInit pointer lead us to struct with information about chosen SPI bus. There are 2 SPI busses configured on active platform, see file io_configuration.c. To get appropriate structure, call function SPI_HandleTypeDef GetSpi1().

- **uint32_t PressureMeasurement()**

Does all necessary thing to return the 32 bit unsigned integer pressure result. To obtain the result, it is using two functions below. They can be used also separately, to obtain raw data or just pressure calculation.

- **uint32_t ReadADC(int8_t cmd, uint8_t regType)**

For raw data readings, two things has to be specified. Firstly the oversampling mode (cmd) and type of register to read (0xD1, or 0xD2). This driver uses HAL_Delay functions as a delay after reset and waiting for ADC conversion. Basically HAL drivers delay function is simple and it's working, but with a software development and interrupt complexity growing, it might be wise to replace this delay function with some more robust one.

- **signed int CompensatedPressureCalc(uint32_t tempD2, uint32_t pressD1)**

Calculation of temperature compensated pressure was already described in the text. Input parameters are pressure and temperature 32bit ADC conversion result.

6.2. Accelerometer ADXL362

The ultralow power 3 axis accelerometer had already optimized driver which was gotten from some version of Vigi'Fall. Basically the accelerometer works alone, after configuration and start of the measurement it collects data according its registers settings. To achieve lowest current consumption, accelerometer can store data in FIFO memory and the memory can be read all at once with one SPI command. That means we don't have to read every single value individually and save some energy. There are other features as well, for example we don't have to measure all axes or we can choose from different sample rates from really low 3,125Hz to high 400Hz for precise fast measurements. Most of the sensors available on the market do not provide higher sampling frequencies that 400Hz, because higher data rates are not suitable for implemented I2C bus. The driver from Vigi'Fall was programmed

for use with ATMEGA microcontroller and so it was modified for STM32. Configuration of accelerometer is set directly to fs=100Hz, resolution ±2g and should be modified to comply requirements received from configuration data stored in GATT. The sampling frequency and resolution is set by Filter Control Register.

After configuration and start of measurement accelerometer starts to fill FIFO with new values and there are several options to read them. Of course it is possible to read FIFO anytime we want, but it would be quite pointless. Accelerometer has two interrupt pins which can be used to let us know about FIFO status, for example full memory, data ready or watermark. In case of „data ready," ADXL362 will put INT1 or INT2 pin to unstable state, if there are any new data in memory. Interruption pin is in unstable state until interruption is cleared by reading the data or status register. FIFO watermark is a number of accelerometer samples in the memory after which the sensor will send an interrupt. This number for FIFO watermark is stored in FIFO Samples Register and FIFO Control Register. FIFO can store up to 512 bytes. It is wise to change size of this watermark with changing sampling frequency. Because to get 512 bytes with lowest rate takes too much time and then it would be impossible to make real time monitoring of those data. Activ'Platform is using FIFO watermark and so after watermark interrupt on INT1 pin, microcontroller sends command to read all new FIFO samples. To make a compromise for FIFO setting, watermark was set to 180B, that means 30 samples, which is the same number as LSM9DS0 maximum FIFO size [21].

The sequence of received data after Read command is easy to overlook in ADXL datasheet and so, the principle is emphasized here once again. Values are 12bit ADC values, so in serial communication they need to be transferred in two bytes. First transmitted is LSB and other byte contain 4 bits of data, two bits of axis information and two bits are unused (called sign extension). Therefore it is highly recommended to always read even number of bytes from FIFO. Otherwise it can lead to loosing data from one axis. Table 3 below shows order of mentioned bits and specifies how axis information works. After ADXL FIFO is read, values are sorted to MCU driver's receive buffer in order X_LSB, X_MSB, Y_LSB, Y_MSB, Z_LSB, Z_MSB and the sign extension is cleared as well as data types. The reason is to have data received from different sensors in the same format and order. Since LSM9DS0 is using this format automatically because of 16bit precision, ADXL driver had to be adapted.

Table 3 ADXL362 FIFO buffer data format [21]

B15	B14	B13	B12	B11	B10	B9	B8
Data Type: 00: X-Axis 01: Y-Axis 10: Z-Axis 11: Temp		Sign Extension		MSB	Data		

B7	B6	B5	B4	B3	B2	B1	B0
Data							LSB

C functions

- **void Power_Accelero(unsigned char temp)**

Accelerometer is powered on or off. No delays are applied. Supply pin setting is written directly in the function.

- **bool Init_Accelero(SPI_HandleTypeDef *spi)**

Again the initialization of SPI and ADXL362 registers configuration. Probably the most important function of the driver. This function is opened to improvements of resolution and sampling frequency settings. Actually the fs is set to 100Hz and resolution to 2g. Please find FILTER _CTL register in ADXL362 datasheet [21].

- **bool Start_Acq_Accelero()**

After the initialization, when all registers are properly set, a measurement can be started. Calling of this function will start data acquisition. To temporarily stop acquisition, function **bool Stop_Acq_Accelero()** is also implemented.

- **uint8_t* Read_Data_Accelero_100Hz()**

After the FIFO contains 180bytes of data, watermark interrupt is generated. When STM receives a change on external interrupt pin, this function is used to read 180 bytes from FIFO memory and returns pointer to address of an array with received values. Values are sorted in order X_LSB, X_MSB, Y_LSB, Y_MSB, Z_LSB and Z_MSB. 30 values multiplied by 6 bytes makes 180 bytes in total. Sign extension bits are cleared to make the format of data same as in case of LSM9DS0 sensor.

6.3. Inertial sensor LSM9DS0

This sensor consist of 3 axis accelerometer, gyroscope and magnetometer in one package. But this sensor acts like two SPI devices. One with gyroscope and the second one with an accelerometer and magnetometer. Accelerometer and gyroscope have their own FIFO memories which can be read the same as reading single value from register. But principle of use is similar to accelerometer ADXL362. Firstly we must configure sensor's registers to obtain optimal function for our application. The configuration registers are individual for gyroscope and for accelerometer with magnetometer, but they have similar addresses. After proper configuration the measurement can be started. Again after the sensor collects some data, it can change logic state on interruption pin but be careful about one thing. Accelerometer with magnetometer has two interruption pins called INTX1 and INTX2, while gyroscope has pin INTG and DRDY for the same purpose. DRDY pin for gyroscope can handle more interrupts than just data ready event including FIFO watermark, overflow etc. This is not different in case of second part of sensor, but accelerometer and magnetometer interrupts used for Activ'Platform are on single pin INTX1. Therefore after each interrupt from accelerometer or magnetometer, we have to check which one was it by reading interrupt registers.

The configuration is also made using received configuration data. And the driver has most of the functions written twice. One for gyroscope part of module and second for accelerometer with magnetometer. They require different slave select pin and register addresses. The difference in function between LSM9DS0 and ADXL362 is checking source of generated interrupt and reading FIFO by many Read commands instead of one command to read all memory. The magnetometer does not have a FIFO, because it uses slower sampling rates than gyroscope and accelerometer. Another important note about magnetometer is, that it cannot be really turned off when accelerometer is active. When magnetometer is not required it is set to power down mode. The reason is using of the same SPI bus, slave select and interrupt pin.

C functions

- **void Power_LSM(bool onoff)**

  LSM sensor is powered on or off. Supply pin setting is written directly in the function.

- **void LSM9DS0_Init(SPI_HandleTypeDef *hspiInit)**

Selection of SPI bus. SPI handle definition structure is used by SPI HAL drivers implemented in this software.

- **bool LSM9DS_ConfigureGyro(uint16_t freq, uint8_t resolution)**

Input parameters of a gyroscope setting are integers assembled from configuration packet array bytes. The function returns true if the gyroscope setting succeed. Default setting is a power down mode, so when the gyroscope is not required for the operation, this function doesn't have to be called.

- **bool LSM9DS_ConfigureAccel(bool accOn, bool magOn, uint16_t freq, uint8_t resolution)**

Frequency and resolution range setting are the same input parameters from configuration packet array as were used in ConfigureGyro function. Another input parameters are bool values, specifying if accelerometer or magnetometer shall be active. If the settings were processed and checked, function returns true bool value.

- **uint8_t ReadRegX(uint8_t reg)**

Reads a single register from accelerometer and magnetometer registers. Input parameter is an 8bit address of selected register. For list of all registers, see LSM9DS0 datasheet [4].

- **uint8_t* AccelReadFIFO(uint8_t banks)**

Reads selected number of values from accelerometer FIFO memory, specified by input parameter **banks**. Parameter banks may take values from 1 to 30, since FIFO can store 30 values. Function returns a pointer to the buffer with received data. Size of this buffer is 180bytes. **GyroscopeReadFIFO** function has the same character, but is programmed for gyroscope interface.

- **uint8_t* MagnetRead()**

  Function reads magnetometer data. Returns a pointer to a buffer with six bytes of data.

7. Embedded software

Embedded software was developed as a relatively simple data acquisition unit with no respect to setting of low power modes or other difficult power saving features. They were kept on mind, during development of drivers for used sensors, but this acquisition unit presumes high sample rates to drive BLE module to its maximal performance, hence low power features were omitted in main code. Figure 20 shows general algorithm of program main loop. Hardware initialization was already described in section 5.2. After initialization, advertising mode on BLE113 is turned on. Then it waits, till it receives configuration packet via Bluetooth so it can configure all required sensors. Sensors can run even during the configuration but they are not receiving any interrupt. To start data acquisition, FIFO memory of each used sensor must be erased. Last step to start measurement is enabling of timer 2. Timer two arranges writings to GATT and data collection from sensor (ring) buffers.

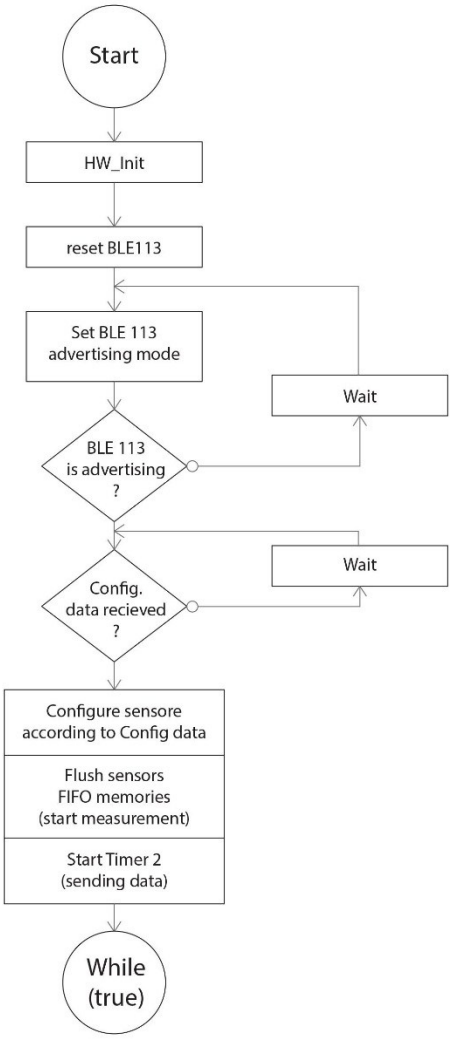

Figure 20 BLE and acquisition start algorithm

7.1. Device configuration

This section describes, how the wireless configuration of Activ'Platform is done. In configuration packet, Figure 21. There are 5 Bytes of data including one called OnOff, two bytes for sampling freq. information and one for sensor resolutions. The OnOff tells us which sensor will be turned on or off. Extension byte was originally used to specify powered axes, but it was decided that it has no meaning, since we want to measure mostly all axes. And in case of modification, it can be done directly in appropriate sensor diver without increasing complexity of software and controll. In freq. bytes, there are two or three bit numbers which specifies the frequency of some sensor. In case of resolution byte, it is 2 bit number, which can set resolution range of some sensors. So each sensor can have maximally 8 different sampling frequencies and 4 different resolution ranges. This solution is very convenient because it includes all possible settings of sampling frequencies and resolutions, using just three bytes.

OnOff:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ADXL	Thermom.	Barometer	acc LSM	mag LSM	gyr LSM	Unused	Unused	Unused (extension)							

Freq:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ADXL 2	ADXL 1	ADXL 0	Therm. 1	Therm. 0	Baro. 1	Baro. 0	AccLSM.2	AccLSM.1	AccLSM.0	MagLSM. 2	MagLSM. 1	MagLSM. 0	GyrLSM. 2	GyrLSM. 1	GyrLSM. 0

Resolution:

1	2	3	4	5	6	7	8
ADXL 1	ADXL 0	acc LSM 1	acc LSM 0	mag LSM 1	mag LSM 0	gyr LSM 1	gyr LSM 0

Figure 21 Configuration packet overall structure

After the configuration packet is stored in GATT, BLE113 sends the event of GATT Attribute Value (see section 4.3). MCU reads the event and stores the configuration packet to public array. After that, all sensors are set, according to this packet (if the setting is implemented in appropriate driver). When all sensors are configured and started, changes in configuration characteristic should not be done. Moreover it would be useless, because sensors setting is implemented just after first receiving of configuration packet and cannot be done during operation.

While Figure 21 shows overall structure of configuration packet, Table 4 Sampling frequency coding and Table 5 explains meaning of values written in the packet. Onoff byte does not need a table

because the meaning of every bit is clear even from the overall structure. When a bit set high, then a sensor which belongs to this bit will be turned on and vice versa.

Table 4 Sampling frequency coding

***fs* binary code** 2,1,0	***fs* [Hz]**					
	ADXL362	**Thermo.**	**Barometer**	**Accel. LSM**	**Magn. LSM**	**Gyro. LSM**
000			3,125	3,125	3,125	95
001			6,25	6,25	6,25	190
010	12,5		12,5	12,5	12,5	390
011	25		25	25	25	
100	50	out of range		50	50	
101	100			100		
110	200			200		
111	400			400		

Table 5 Resolution coding

resolution binary code 1,0	**sensor resolution range**			
	ADXL362	**Accel. LSM**	**Magn. LSM**	**Gyro. LSM**
	[g]	[g]	[G]	[°/s]
00	2	2	2	245
01	4	4	4	500
10	8	8	8	2000
11		16	12	

7.2. Data acquisition

Both accelerometer ADXL362 and inertial sensor LSM90DS0 drivers have function, which returns a pointer of buffer array with new values read from FIFO memory. Because writing values to GATT and reading FIFO memories must run simultaneously, a ring buffer was programmed. Using a standard array as a buffer would be unstable solution. Because if the GATT writing algorithm doesn't read all bytes from buffer before next FIFO interrupt, loss of data occurs. Ring buffer (or circular buffer) has a great advantage of writing data, when the buffer is not read yet. Ring buffer requires two pointers for operation. Activ'Platform is using two unsigned 8bit integers as pointers (they specify written or read cell of a ring buffer, they are not a pointer types). Size of circular buffer is set to 256 bytes. Thanks to utilization of 8bit integers, reaching of maximal value will cause them to overflow, so the ring buffer is working right from the principle of data types and there is no need to control pointers circulation manually.

Figure 22 Writing and reading the ring buffer shows the relationship between ring buffers and the rest of the code. Ring buffer can be accessed after two events. The external interrupt, telling about

full FIFO memory. When the FIFO is read, code in ISR stores data in circular buffer, shifts write register and increments size byte. Second event that requires aces to circular buffer is Timer 2 interrupt. Timer 2 ISR calls a function to fill dtsBuffer (data to send buffer). Size of dtsBuffer array is 56 bytes, which is the maximum amount of data sent during one connection interval. Average frequency of reading must be always higher than a frequency of writing.

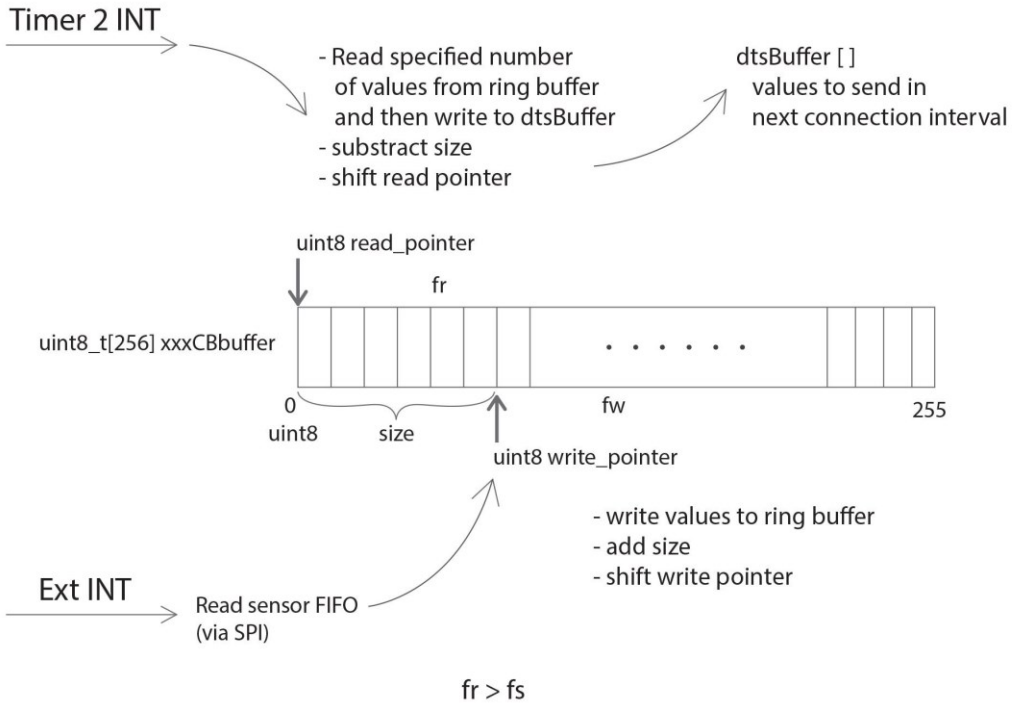

Figure 22 Writing and reading the ring buffer

Because highest sampling frequency is 400Hz and highest connection frequency just 133.3Hz it is obvious, that by reading just one value from ring buffer the fr >fs condition won't be satisfying. So when chosen sampling frequency is higher than maximum connection frequency (of any sensor), more values must be read. Table specifies a number of **values** to read in one connection interval and the connection interval length **Δtc** for a selected sampling frequency **fs max**. C code representation of this table is mediated by DataAmounts function (in section 5.2). Number of values sent in one connection interval (called data amount in the software) is individual for each sensor, due to its sampling frequency.

Table 6 Amount of data to send and connection interval setting

***fs max* [Hz]**	***Δtc* [ms]**	***values***
3,125	240	1
6,25	120	1
12,5	60	1
25	30	1
50	15	1
100	7,5	1
200	7,5	2
400	7,5	4

Algorithm on a Figure 23 shows external interrupt service from GPIO. As it was mentioned before, external interrupt is generated when a FIFO memory of some sensor is fulfilled or filled to a watermark (ADXL driver, section 6.2). When the interrupt is generated, system checks which sensor generated the interrupt and reads appropriate FIFO memory using programmed drivers. Because LSM9DS0 accelerometer and magnetometer have the same interrupt pin, one LSM9DS0 register bust be read to identify which sensor generated the interrupt.

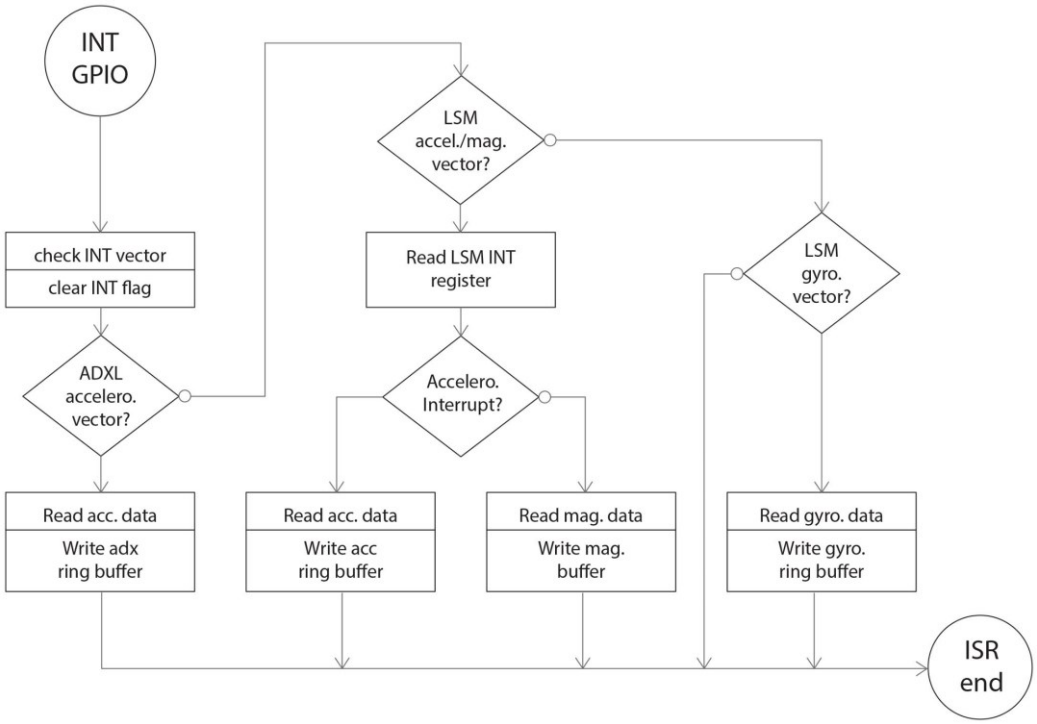

Figure 23 External interrupt service routine (ISR)

Timer 2 interrupt service handles writing to a GATT protocol. To achieve maximal throughput via Bluegiga BLE113 module. Three Write commands with 20B of payload are executed during one connection interval. Structure of data stored in those three commands is explained in next section. That's why the period between timer 2 interrupts is always one third of a selected connection interval Δtc. In a first of three interrupts in row, a dtsBuffer (data to send buffer) is filled if there are some data to write. Then a first Write command, containing first18Bytes from dtsBuffer, is sent via UART. Function that fills the dtsBuffer has an index variable, telling us how many bytes is stored in the buffer. According to this index, it is decided if second and third Write commands will be executed. Simplified structure of timer 2 ISR is shown on Figure 24, assuming that all Write commands will be executed. Real C algorithm uses a specific function for Write command and this function recognizes if the command Write has to be sent or if the dtsBuffer is emptied.

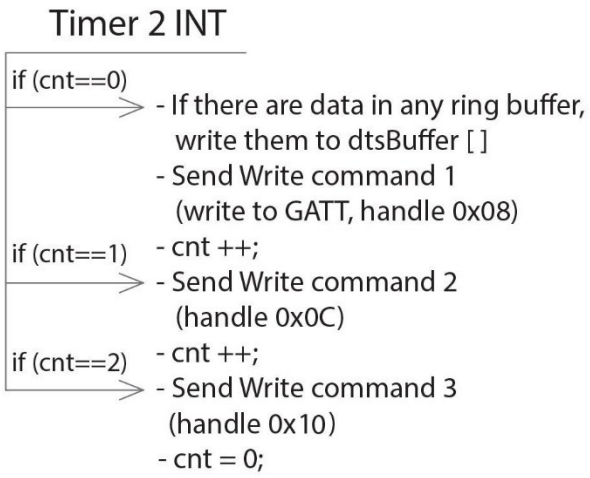

Figure 24 Timer 2 ISR

7.3. Communication protocol

The idea of GATT structure is to have many characteristics and attributes, each for one sensor and its parameter. When this idea is applied, every parameter can be found in GATT protocol under its unique handle. Therefore data are well sorted, and whole GATT structure reminds a table of actually received data. But Activ'Platform can contain 4 sensors (3 implemented). So GATT protocol would have 4 services for data transmission. Three of them would have more than one attribute, meaning accelerometer, magnetometer and gyroscope with three axes. That means 4 services and 15 characteristics. With increasing speed it would be impossible and also not efficient to execute 15 commands for writing an attribute to GATT. Also receiving of another 15 Attribute Value Changed events wouldn't be nice solution.

Because of the previously mentioned problem and fact, that only three characteristics can be used for data transmission in one connection interval with highest speed (already explained in section 4.2), the communication protocol and system of transmitting data had to be written to use just three characteristics for all transmitted data. Therefore, previous services, one for each sensor, were replaced with just one service, with three characteristics (see Table 2 Activ'Platform GATT configuration). Each characteristic contains 20 bytes of data. First byte of each characteristic will contain 8bit sequence number used for synchronization, incremented with every connection interval. And the second byte of first characteristic is a „packet contain" byte, for received packet decoding. Transmitted data are sorted in fixed order and a packet contain byte tells us, which data from which sensor are present in upcoming characteristics with equal sequence number. Low level in this byte means, that packet does not contain this value and vice versa.

1	2	3	4	5	6	7	8
Acc. ADXL	Thermom.	Barometer	Acc. LSM	Mag. LSM	Gyr. LSM	Unused	Unused

Figure 25 Packet contain byte, from a bit point of view

The communication protocol overview is shown on the picture below. First characteristic will contain seq. number, packet contain byte and then 18 bytes of data. If data are bigger than 18B, the rest of them is written into next one or two characteristics. Second and third characteristic also have a sequence number to control, if all received data belongs to each other. Whole picture is a GATT service and single tabs (Data1, Data2, and Data3) represents characteristics. These characteristics can be read by handle stored in GATT.

Data 1	1B	1B	18B
	Seq. Num.	Packet contain	Data
Data 2	1B	19B	
	Seq. Num.	Data	
Data 2	1B	19B	
	Seq. Num.	Data	

60B total length, 56B payload

Figure 26 GATT data service

All values are little endian, sorted in this order: Axis X LSB, Axis X MSB, Axis Y LSB, Axis Y MSB, Axis Z LSB, Axis Z MSB and repeated as many times as the protocol requires. Amount of data sent during one connection interval was specified in section 7.2.

Communication protocol was tried on experimental data (saw tooth signal). An array of values, incremented in every connection interval, was programmed and transmitted via Bluetooth LE, using this communication protocol. Then after storing the data PC text file, it was checked that all values are written on appropriate place due to communication protocol rules.

8. Pc data acquisition and monitoring

For a wireless configuration of Active'Platform, collecting and data monitoring, C# software was developed. Using Microsoft Visual Studio 2013, it is possible to program form application for Windows. Visual Studio form represents GUI, where standard Windows objects and controls can be used. Visual Studio has set of programmed controls and other objects that can be added to form application and configured by developer. Then the program looks like standard application used in Windows operating system. This application works with BLED112 dongle connected to USB port on PC.

Application is switching between two forms (between two windows). One controls Bluetooth connection, configuration of Activ'Platform and data saving settings. Second form serves as a monitor to show measured data. The code is quite complex and there is no place for a whole description. Moreover most of the code handles service of GUI components, which is not considered as a part of main algorithm. Operation of basic parts will be explained as well as overall function. Some Important C function are explained in section 8.2, the rest is commented in the software source code. Many functions are throwing status bar reports about their execution and result. This feature should serve to user and also to a developer to understand a code sequence.

8.1. BLED112 dongle

BLED112 dongle is a Bluetooth Smart single mode Low Energy device. This dongle drive has the same core as the BLE113 module. It is designed into a shape of little flash memory drive. After plug into a USB port, it can be controlled with the same commands as BLE113 module, since the interface used is a serial port. After the dongle is plugged to USB port, it can be found in Device Manager as a COM port. If the PC operating system reports an issue during hardware installation, an additional driver can be required. Driver can be obtained from Bluegiga websites or offline from attachments of this document. BLED112 service slightly differs from BLE113. Firstly, because the dongle drive is master (or client). And secondly because COM of flow control. No additional byte to allocate direct memory access is required. So commands are one byte shorter compared to UART communication between BLE113 and MCU.

So after start the COM port has to be opened. Then a scanning mode is set on the dongle drive. Again the module is using commands, responses and events. In the scanning mode, after the module receives some advertising packet, it is sent to PC as an event. After an advertising packet is processed, it is possible to pick up a BLE device from a GUI to send a connection request.

8.2. Software description

Great advantage of Visual Studio C# software is a use of prepared controls and classes. There are system methods to use serial port, buffers, FIFO, writing data to file etc. Each incoming message to connected COM port generates an event. Data are stored in input buffer and after reading them, it is

necessary to decode what that message means or what kind of data was received. Therefore there is a code for sorting received data. Each message is stored as an instance of the custom class called Received data and important parts of the message are stored as properties such as message **type**, **data**, **class** and **ID**. Message type says whether message is event or response, command class and ID specifies the purpose of received message so it can be processed according to its purpose. Code for sorting messages may look little confusing, but all it does is a comparison of message class, ID and message type with stored values, to decide which method should be called to process the message or to perform an appropriate response. Best way to understand is to cooperate with Bluegiga API reference guide and find message's ID and classes (some of them explained in section 4.3).

So after start of the application, user will put dongle to a scanning state by clicking scan button, which sends command for setting module to scan state. Incoming events from module are processed to obtain addresses, names and other parameters of found devices. They are stored as an instances of the class called Devices and displayed in list box object in application setting GUI. After choosing one of them and clicking connect button. The command for sending connection request with data of chosen device is executed. When the connection is initiated, BLE dongle receives a connection status packet telling us if the connection succeed, or if an error occurred. Error messages are written to a status bar, with an error code from Bluegiga API Reference and a possible solution. If the devices are connected, user sets the configuration of Activ'Platform, via GUI, confirmed by Measure button and the software will create appropriate configuration packet from given parameters. When the configuration packet is sent, setting GUI is hidden and replaced by measurement GUI. The first GUI is hidden but running, because the code inside contains a part for receiving and sorting data from dongle. Measurement form gets received data as a queue and draws them to appropriate charts. Meanwhile the configuration form also does writing into the file from the same data, but different buffer.

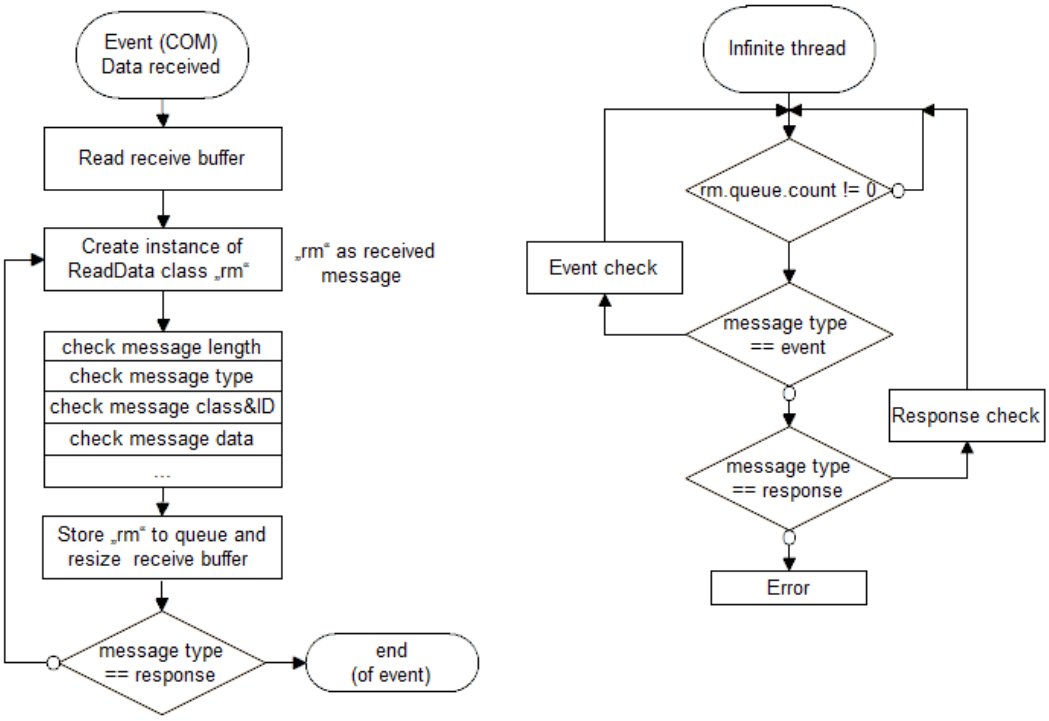

Figure 27 General algorithm of message processing

C# functions

- **public void DataReceived(object sender, SerialDataReceivedEventArgs e)**

Generated by an event, when a message is received from the COM port. ReceivedData class instance is created and called **rm** as a received message. Note that COM port can cut the API messages into different pieces, simple algorithm for solving this issue was programmed and it's commented in the code.

- **public void ControllLoop()**

The only function in the code that is using custom thread. This thread gets the received message instance and stores it inside two buffers. A monitorqueue for real-time monitor and the other called fifo. Received messages from fifo are processed during the timer 1 tick event and written to the text file. ControlLoop function also checks the message type and then calls a ResponseCheck or EvenCheck method.

- **public void btnStop_Click_1(object sender, EventArgs e)**

A configuration packet is formed and sent by this event. Setting GUI is hidden after this event.

- **public void DeviceChoose(byte[] adress, byte[] data)**

This function is called after the scanner receives an advertising packet. Structure of advertising packet is translated and received information, such as device name or device address, are shown in list view.

- **private void EventCheck(byte classID, byte messageID, byte[] data)**
- **private void ResponseCheck(byte classID, byte messageID, byte[] data)**

These two functions are called from the Control Loop. Source code of these functions is commented according BGAPI Reference. These functions were written to identify the packet header and to call an appropriate method for data processing. Occasionally to write massage on the status bar. Input parameters are selected parts of ReceivedData class.

- **private void ChartSetting(float percsize, int grcnt,int itemindex)**

Chart setting is executed after selection of measured variables and a "Measure" button click. Function is repeated several times and creates selected number of chart objects with necessary settings. Percsize variable sets the chart area size due to the number of charts, window size etc. All input parameters are calculated automatically in one of timer 1 tick events. Variable grcnt is a number of charts and itemindex is setting index of a chart.

8.3. Real time monitor

The real time monitor serves for visual control of received data. It is programmed using Visual Studio Chart class, available from .NET Framework 4.5. All charts are refreshed in Timer Monitor Tick event. Because all data can be stored in three received messages, they must be formed into one buffer before their processing. That buffer is called simply dataArray. A sequence number from received packet is stored in global variable and when the received message is processed, program checks if received sequence number is the same as the old one, stored in a variable. If it is the same sequence number, received data are added to dataArray buffer, behind the previously received characteristic. If the received sequence number differs from the old one, dataArray is processed and values identified in dataArray are drawn to appropriate chart object.

Real-time monitor is working in rolling mode, which means that every received value is drawn on the right border of the chart area. All points are shifted to the left and erased, after they exceed the left border of chart area. That will keep a constant number of points in chart area. This number of point can be changed during operation, with X Axis numeric up down control. Figure 28 shows an accelerometer test. Accelerometer was running with resolution range of 2g and 100Hz sampling frequency. Figure shows real-time measurement of standard gravity acceleration (1g) and changing the orientation of axes. Axis Y label is a value of acceleration in units of standard gravity, X axis label is a number of sample.

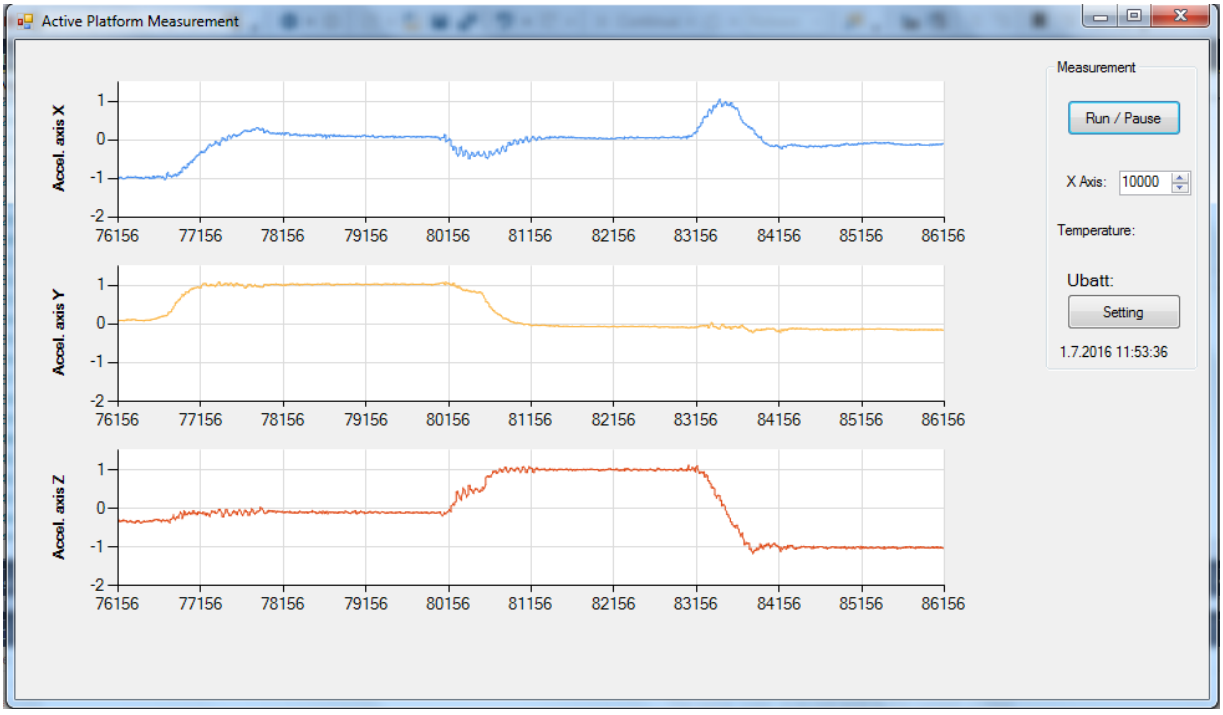

Figure 28 Measurement form Accelerometer

8.4. Storing the data

Data storage was solved as a text file, where values are split by tab character. This format is very suitable for Excel or Matlab import. File location can be changed in the setting GUI. If the file location setting is omitted, txt file is created in the folder of software's executable. User must be sure to put the executable into the folder with allowed access.

As mentioned in previous section, when a message is received (GATT Attribute Value event) it is processed by Received Data class. Then a created class instance is saved inside two buffers. One is programmed using Queue class and second with a List class. List contains data, which will be written to a text file. An algorithm used for processing received messages is almost the same as the one for real-time monitor. Additional explanation is not necessary to understanding of the code, since the difference is mainly in writing strings instead of creating chart points. Strings must be split by an adequate number of tab characters to obtain the structure suitable for Excel or Matlab.

Received data are stored in a string format corresponding to ADC signed 16bit integer value received from a sensor. It was decided to use this format because storage of converted values in double format requires an additional setting of decimal points before conversion to string, or the file doesn't look very organized. Text file header has three rows with name of every measured variable, sampling frequency value and resolution range.

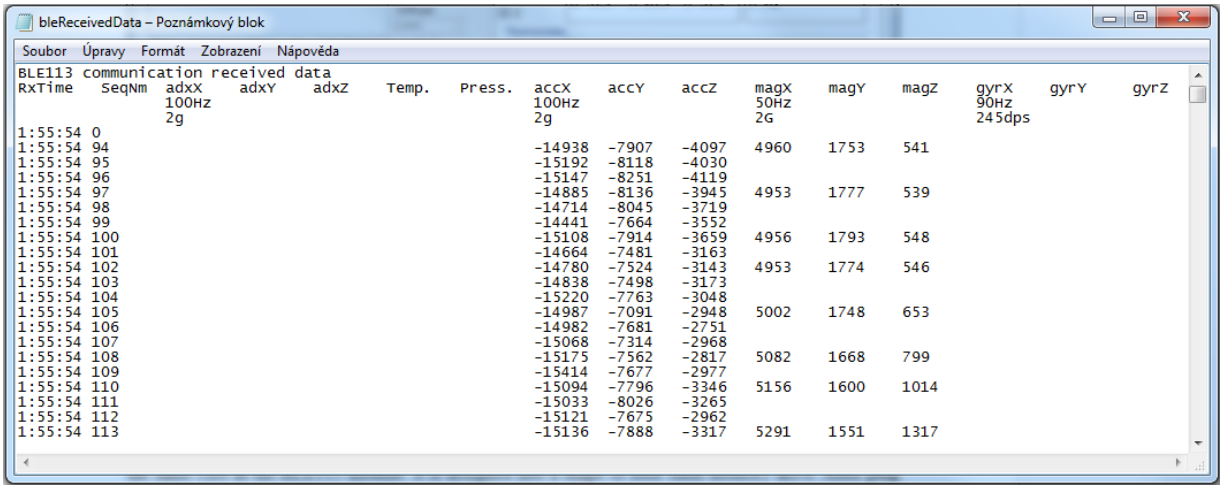

Figure 29 Measured data stored in .txt file

8.5. Application user guide

After the application is started, user sees the first configuration GUI. Firstly a COM port has to be opened, followed by software connection of BLED112 dongle and status bar message with connection result. If there's no dongle available in combo box, it is necessary to restart application or reconnect the dongle. Secondly a scanning mode has to be started by clicking the button scan. Active scanning is set by default. When active scanning isn't set, the device name will not be received, but replaced by default name.

In the left top corner is list box with a search results. There we can choose a desired device and initiate connection by clicking the connect button (scan button changed status to connect). In case of mistake, when desired device wasn't selected, we can click disconnect (the same button again) and repeat the procedure. When the connection between BLE113 and BLED112 is initiated, a configuration packet can be created by selecting parameters on the right side of the window or by choosing one of the presets. Presets can be simply modified by developer. To add preset, List View collection has to be modified using Visual Studio environment. Then a setting can be done in function PresetListView, by copying and modification of older preset code.

Counter in right down corner (highlighted in green) says, how many more bytes can the BLE handle and will not allow us to choose more data to send if counter exceeds 56 bytes. So counter value depends on number of chosen sensors or values and their sampling rates, but doesn't depends on resolution range. The resolution range's selection doesn't affect the bit precision of sensor's output.

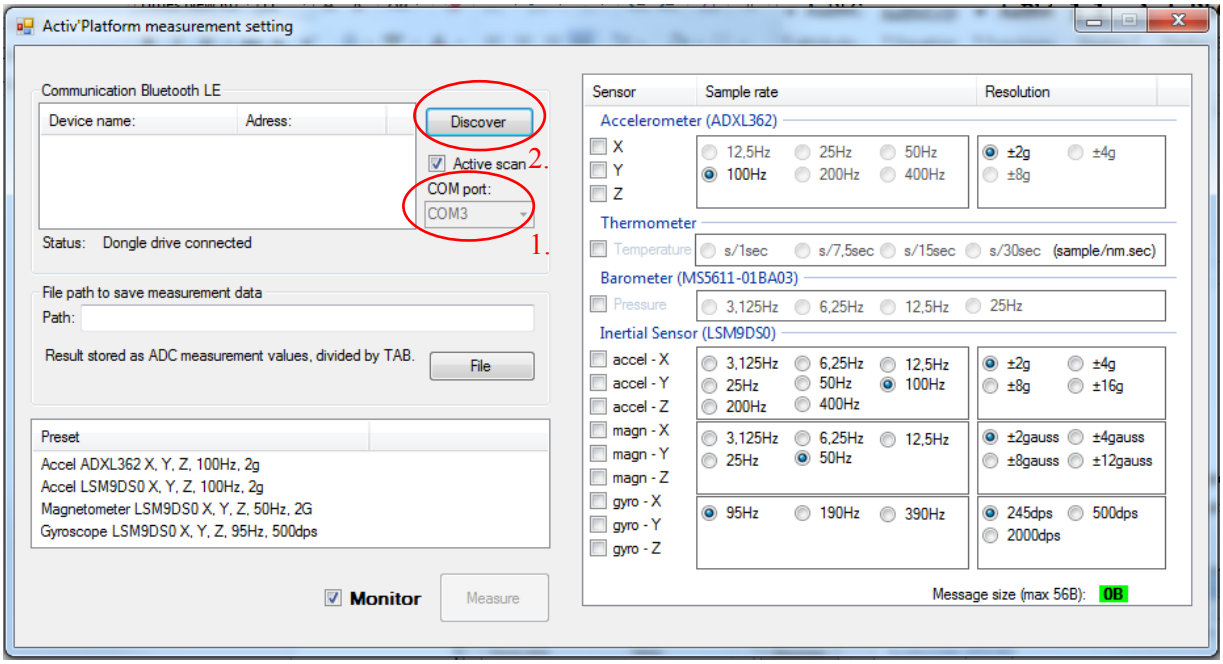

Figure 30 Application guide (COM port selection)

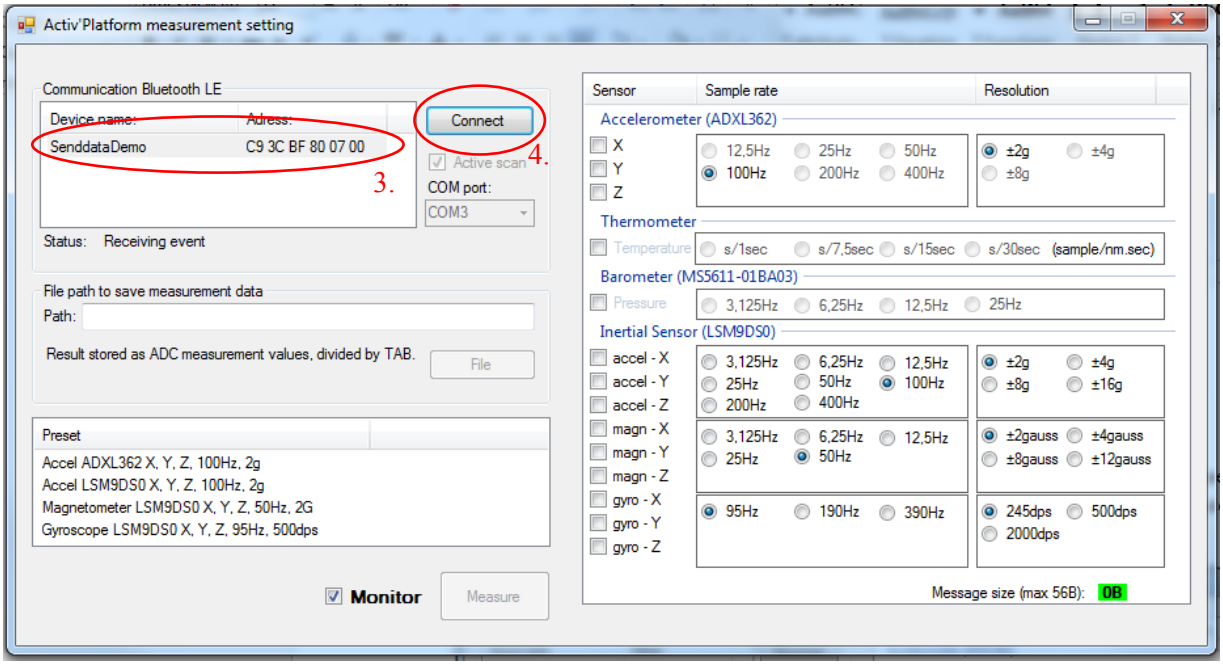

Figure 31 Application guide (BLE device connection)

After click on the Measure button, configuration packet is created, transmitted and stored into GATT and the measurement window is opened while the configuration window is closed. To enable data receiving, it is necessary to enable notification of characteristic, which is done by Run button. Then measurements starts. If device is disconnected by an accident or by action of slave. Software will throw a pop up window with a warning. To reconnect the device, user must return to setting form and repeat explained steps.

Real-time monitor can be turned on or off with a check box next to the measurement button. When real-time monitoring is disabled, measurement window will be still opened, but instead of charts, user sees reports about start and stop of saving data.

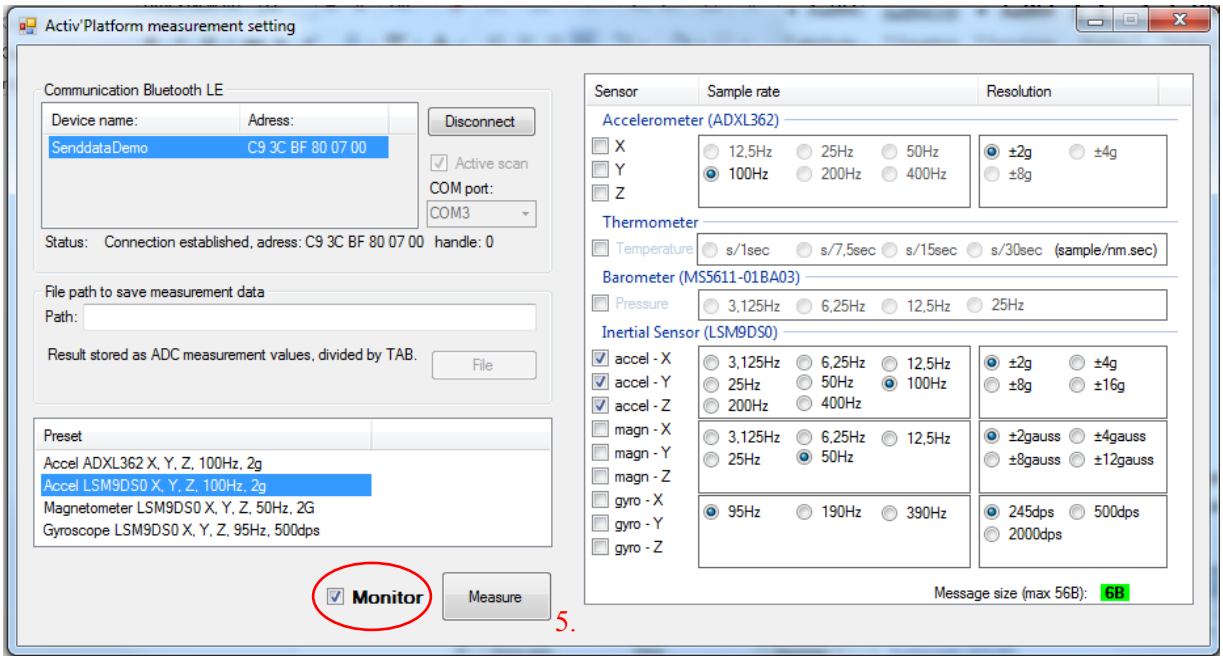

Figure 32 Application guide (device configuration)

9. Optimization and measurement

First prototype of Activ'Platform was built from separate components using prototype board. As a microcontroller, the STM32L053 Discovery board was used. Discovery board has internal ST-Link debugger, but it doesn't seems to be a good replacement of original ST-Link, because just operation itself sometimes caused unpredictable problems. Maybe due to STM Discovery board peripherals. Sometimes it is hard to find out if the problem is in embedded code, debugger or it's a HAL driver bug. This issue was consulted with colleagues in Vigilio S.A. and it was decided to use J-Link debugger, since it is professional product, that shouldn´t have this kind of problems. During development with J-Link, no issues of this character appeared. Most difficult part of a development is an unexpected problem or when the used technology is facing to some limitations. Solution of these parts takes most of the development time. Purpose of this section is to explain some of them and their solutions.

9.1. BLE113 throughput

While developing Activ'Platform firmware, the first idea was to use multiple characteristics in GATT to transmit data via Bluetooth. For that case, a service for every sensor from active platform and characteristic, for every sensor's measured parameter was created. For example service: accelerometer ADXL362, characteristic 1: axis X, characteristic 2: axis Y etc. Each characteristic contained appropriate number of bytes, according to sampling rate and sensor's resolution (meaning a bit resolution of the sensor's ADC, not a resolution range). For wireless transmission of data, Write command with appropriate handle and data must be send via UART. Each Write command is followed with response of BLE113 module as an acknowledgement of writing to GATT. BGAPI Reference manual says, that delay between first transmitted byte of a command and a first received byte of a response is from 0.8ms to 1.5ms. And it was measured that this delay stays in this interval, no matter what baud rate is set. So by using this GATT setting, we wouldn't be able to refresh GATT during one connection interval (using fc = 133.3Hz) even if high baud rate was set.

Therefore the communication protocol (section 7.3) was developed. System was tested with baud rate 57600kbps, the time interval between the start of transmission and the end of BLE113 response was 5.12ms. Using this baud, system was able to send only one Write command in 7.5ms period. Using baud rate 115200kbps, mentioned interval was approximately 2.4ms, but still not enough to achieve stable Bluetooth LE connection with maximal throughput. Even the setting of 230400kbps, which is the maximal baud rate for BLE113 module, tested by Bluegiga Company was not enough and therefore even higher baud rate was set. UART is now running with 460800kbps speed and it is able to send all three characteristics from communication protocol in one 7.5ms connection interval. The example of measurement is shown on Figure 33 Write to GATT command and response (baud 115200)

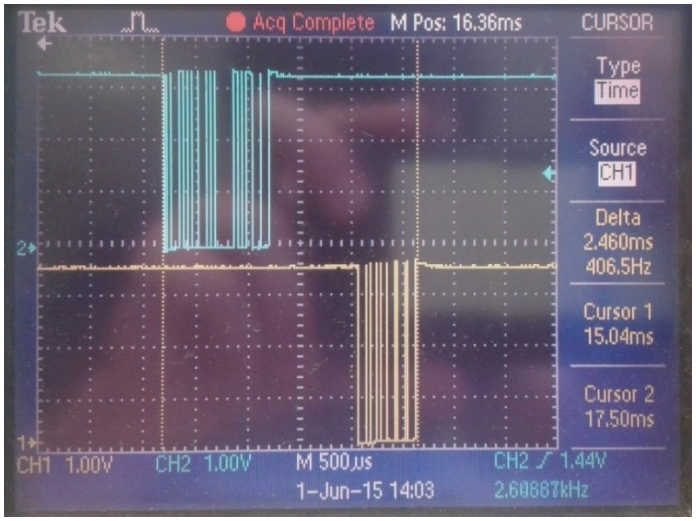

Figure 33 Write to GATT command and response (baud 115200)

9.1. STM32 Bootloader

ST Microelectronics Cortex M0 devices offer three memory options, where a code can be stored. Two locations in Flash memory and one in SRAM are available. But when a binary source code is downloaded into MCU a bootloader must be set to the same starting address, or the code will not execute. STM discovery board used as a first prototype of Activ'Platform, had a commonly used bootloader setting (0x8000000 starting address and 0x10000 size). This setting is made by hardware connection of pull down resistor to, to BOOT0 pin on MCU. When there's a pull up resistor on BOOT0 pin, additional setting is required. This additional setting bust be stored in specific registers and it wasn't found during development, where and how the appropriate setting can be done in C embedded software. Only a tool for ST-link debugger is provided with a guide for this setting. Because ST-Link debugger wasn't used for Activ'Platform prototype programming, more available setting was chosen. Unfortunately Activ'Platform prototype was assembled with a pull up resistor R5 instead of pull down R4 so the position of assembled resistor R5 had to be switched. This hardware setting will be changed in next version of prototype.

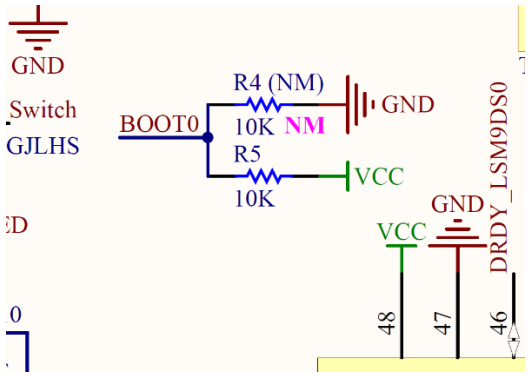

Figure 34 Boot setting

10. Results

Firstly, let's summarize parts of this project, step by step. Bluetooth technology principles and operation was studied, concentrating on LE option. With a sufficient knowledge of BLE operation, first tests on Bluegiga BLE113 module were mage, using BGSCRIPT language. When the module was able to transmit experimental data to the PC, an implementation of BGAPI was started. Then a first version of C# software was developed for BLE communication monitoring. And after this point, a real time monitor application was prepared and a turn came to embedded software development. Embedded software now contain all sensor drivers described in this document. It handles basic communication via Bluetooth LE and it is able to transmit experimental data with highest speed that this technology supports. Measurement drivers were tried out individually on the first prototype of Activ'Platform, assembled on a prototyping board. C# software was modified to meet the requirements of Activ'Platform configuration. Design of the first standalone prototype was consulted with colleagues in Vigilio company. Then the prototype was assembled in a factory and after solving some issues even this prototype was tried out on experimental data.

The overall Activ'Platform project isn't finished with this thesis, but now it contains necessary parts making the development easier in the future. Since sensor drivers were made with respect to low power, I suppose that the next part of the project will concentrate on this feature. The wireless configuration of the module was designed and tested on LSM9DS0 sensor. Designed structure of configuration characteristic in GATT contains unused extension byte, which can be used for possible modifications and improvements. An SD card driver wasn't part of this thesis. Therefore the extension byte in configuration characteristic can be used to turn off the Bluetooth after configuration and use SD card storage instead of wireless transmission. For the proper operation of whole unit, a long debugging will be necessary. Because of time limitation of this thesis, it cannot contain long-term tests and improvements of code robustness. This part of whole Activ'Platform project is focused mainly on BLE communication, PC software and hardware drivers for data acquisition.

Attachments included in the thesis are latest versions of embedded software, C# software and BLE113 configuration code.

References

[1] J. D. Bronzino, The Biomedical Engineering Handbook, Boca Raton: CRC/Taylor, 2006.

[2] M. Dadafshar, *Accelerometer and Gyroscopes Sensors: Operation, Sensing, and Applications,* 2014.

[3] M. Jan, Elektronika, 6 editor, Praha: Idea servis, 2008.

[4] S. Microelectronics, *Everything about STMicroelectronics 3-axis digital MEMS gyroscopes,* 1 editor, 2011.

[5] S. Devices, *Introduction to the principals of Smart Pressure Devices,* 2011.

[6] H. Robin, Bluetooth low energy, The Developer's Handbook, Upper Saddle River, NJ: Prentice Hall, 2012.

[7] Bluregiga Technologies, *BLE113 API reference,* 1.3 editor, 2014.

[8] „X-IO Technologies, X-BIMU," [Online]. Available: http://www.x-io.co.uk/products/x-bimu/.

[9] „X-IO Technologies, X-IMU," [Online]. Available: http://www.x-io.co.uk/products/x-imu/.

[10] „stt systems," [Online]. Available: http://www.stt-systems.com/products/inertial-motion-capture/stt-ibs/.

[11] „Inertia Technology," [Online]. Available: http://inertia-technology.com/promove-mini.

[12] „Cometa, WaveTrack," [Online]. Available: http://www.cometasystems.com/products/wavetrack-inertial-system.

[13] „Arfaduit, 10 DOF MEMS IMU Sensor," [Online]. Available: https://www.adafruit.com/product/1604.

[14] J. Rowberg, „[HOW-TO]: Maximize throughput with BLE modules," 2012. [Online]. Available: https://bluegiga.zendesk.com/entries/22400867--HOW-TO-Maximize-throughput-with-the-BLE112-BLED112.

[15] Bluegiga Technologies, *Configuration Guide,* 2014.

[16] Bluetooth, „Bluetooth Developer Portal," [Online]. Available: https://www.bluetooth.com/develop-with-bluetooth.

[17] Bluegiga Technologies, *Bluetooth Smart Profile Toolkit Developer Guide,* 3.4 editor, 2014.

[18] SEGGER Microcontroller, *J-Link / J-Trace User Guide,* Rev.0 editor, 2016.

[19] ST Microelectronics, *Ultra-low-power STM32L0x2 advanced ARM-based 32bit MCUs reference manual,* Rev.1 editor, 2014.

[20] Measurement Specialties, *MS5611-01BA03 Barometric Pressure sensor datasheet,* 2012.

[21] Analog Devices, *ADXL362 Micropower, 3-Axis, Digital Output MEMS Accelerometer Datasheet,* Rev. A editor, 2012.

List of Attachments on CD

Attachment I. BLED112D driver and BLE113 custom configuration files

Attachment II. Activ'Platform schematic, PCB drawings and production data

Attachment III. Embedded software project (Keil uVision 5)

Attachment IV. Data acquisition PC software (MS Visual Studio 2013)