The design and implementation of sensor communication protocol with connectivity adapter interfaces in nRF51822 embedded development platform
Abstract

Nowadays technologies such as the wireless connectivity, embedded devices and MEMS provide a possibility to build tiny compact devices which have sensing, computing and networking capabilities. The sensors connected to the embedded devices give new possibilities and environments for collecting sensor data and passing it to the networks or other devices. Wireless sensor node technology, as the name suggest, is purposely designed to collect sensor data from the sensor network cluster(s) or a single sensor. “The Internet of Things” defines the systems that consist of networks and networks of sensors, actuators, and smart objects which are interconnected together allowing communication with one another; some of which have decision making capabilities.

The wireless sensor networks and the Internet of Things provide several data centric protocols to access sensor data of sensor networks and the embedded devices. Data centric protocols are based on two paradigms request-response and publish-subscribe. The Publish and subscribe is quite popular in today’s solution because the same sensor data can be routed and published to several final processing units or the devices which are connected to the router gateways.

Sensor communication protocol with the connectivity bus adapter interfaces provides an easy and simple way to access sensor register over different kinds of connectivity. Protocol can be integrated to the embedded device which then provides the message passing with two connectivity adapter interfaces. A lower level connectivity interface provides a possibility to interact with the sensor via I2C of SPI buses. A high level interface provides a connection to and from the devices over the wired or wireless connections.

Keywords
Sensors, protocols, data centric, embedded systems

Supervisor
Ph.D., University Lecturer Raija Halonen
Foreword

When I started the master’s thesis I didn’t believe that I could manage such a tight schedule to conduct the research. I would like to thank Ph.D., University Lecturer Raija Halonen for the encouragement and guidance to research work. I would like to offer my gratitude for my family and team members for all the support and patience with adapting to such a tight schedule.

Juha Niemi

Oulu, May 3rd, 2016
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>BLE</td>
<td>Bluetooth Low Energy</td>
</tr>
<tr>
<td>DSRM</td>
<td>Design Science Research Methodology</td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose I/O</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>I2C</td>
<td>Inter-Integrated Circuit</td>
</tr>
<tr>
<td>L2CAP</td>
<td>Logical Link Control and Adaptation</td>
</tr>
<tr>
<td>MEMS</td>
<td>Microelectromechanical Systems</td>
</tr>
<tr>
<td>ODR</td>
<td>Output Data Rate</td>
</tr>
<tr>
<td>OSI</td>
<td>Open System Interconnection Model</td>
</tr>
<tr>
<td>PP</td>
<td>Publish-subscribe</td>
</tr>
<tr>
<td>RR</td>
<td>Request-Response</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>SOC</td>
<td>System on Chip</td>
</tr>
<tr>
<td>SA</td>
<td>Sensor/Actuator</td>
</tr>
<tr>
<td>TWI</td>
<td>Two Wired Interface</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver Transmitter</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>WSN</td>
<td>Wireless Sensor Network</td>
</tr>
</tbody>
</table>
Contents

Abstract ........................................................................................................................................... 2
Foreword .......................................................................................................................................... 3
Contents ........................................................................................................................................... 5
1. Introduction ................................................................................................................................. 6
  1.1 Purpose of study ...................................................................................................................... 6
  1.2 Research motivation ............................................................................................................... 8
  1.3 Research method and questions ........................................................................................... 8
  1.4 Structure of the thesis ............................................................................................................ 8
2. Earlier knowledge .......................................................................................................................... 9
  2.1 Embedded systems ............................................................................................................... 9
  2.2 MEMS technology .............................................................................................................. 10
  2.3 Protocols ................................................................................................................................ 10
    2.3.1 Sensor nodes .................................................................................................................... 12
    2.3.2 Internet of things ............................................................................................................. 14
3. Research approach ..................................................................................................................... 17
  3.1 Design science research ....................................................................................................... 17
  3.2 Design science framework ................................................................................................... 18
  3.3 Design science research methodology .................................................................................. 20
  3.4 Process implementation ....................................................................................................... 21
4. Development of artefact .......................................................................................................... 23
  4.1 Sensor evaluation kit concept ............................................................................................. 23
    4.1.1 Off the shelf approach .................................................................................................... 24
    4.1.2 Microcontroller approach .............................................................................................. 24
  4.2 Protocol development environment and system architecture ............................................. 25
    4.2.1 Development platform and sensors ............................................................................... 25
    4.2.2 Software tools and compiler .......................................................................................... 26
    4.2.3 System architecture ....................................................................................................... 26
  4.3 What was designed and developed ....................................................................................... 27
  4.4 Technical and functional requirements .................................................................................. 28
  4.5 Protocol design ..................................................................................................................... 29
    4.5.1 Structured view (Reference implementation) ................................................................. 29
    4.5.2 Connectivity bus adapter interfaces ............................................................................... 30
    4.5.3 Dynamic behaviour ......................................................................................................... 34
  4.6 Message formats .................................................................................................................... 38
    4.6.1 General message format ................................................................................................. 38
    4.6.2 Message header ............................................................................................................... 39
    4.6.3 Message variable part ...................................................................................................... 39
    4.6.4 Individual messages ........................................................................................................ 41
  4.7 Functional description .......................................................................................................... 42
  4.8 Demonstration ...................................................................................................................... 44
  4.9 Evaluation ............................................................................................................................... 50
5. Discussion and conclusions ....................................................................................................... 52
1. Introduction

Generally there are many sensor based systems, which provide access for sensor data over different types of communication channels, introduced in the sensor technology area. Wireless Sensor Network (WSN) technology was defined as a network of sensor nodes, which sense the environment and then communicate the information gathered over the wireless links. (Hunkeler, Truong, & Stanford-Clark, 2008)

Combining Microelectromechanical Systems (MEMS), wireless communication (Bluetooth low energy) and advanced electronics (microcontrollers) have given new capabilities for the development of the small smart-devices; having sensing, computing and both wired and wireless communication ability. Internet of Things (IoT) based technology provides new challenges for sensor technologies, because embedded environments are everywhere around us. Embedded devices provide new kinds of environments which generate enormous amounts of sensor data for storing, processing and presenting it in a seamless and interpretable form. (Gubbi, Buyya, Marusic, & Palaniswami, 2013) Radio-Frequency Identification (RFID) tags, sensors, actuators, and smart devices are able to interact with each other and are designed to work on top of the traditional networking. Currently the IoT is a rapidly growing market in the area of modern wireless telecommunication. There are still many technological challenges related to IoT, for example the interoperability of connected devices. Several industrial standardizations and body- research methods are involved in accomplishing the goal of fulfilling the highlighted technological requirements. (Atzori, Iera, & Morabito, 2010)

1.1 Purpose of study

The purpose of this study was to provide a simple protocol for accessing sensor data over wired and wireless connections in embedded device development platforms. The protocol was developed as a part of the sensor evaluation kit concept for enabling the microcontroller base device support for the evaluation kit. The main goal of the sensor evaluation kit concept was to provide a sophisticated way to demonstrate newly manufactured sensors for the customers. The second goal was to provide powerful tool for the sensor engineers in their daily driver and sensor algorithm software development phase. Protocol was also enabling the easy way to build demo application(s) to a PC without having to change the embedded device firmware software. The firmware and the evaluation kit concept was planned to be an open source in the future which might lead the sensor testing ecosystems around it.

This study was done at Kionix, Inc which is global MEMS inertial sensor manufacturer based in Ithaca, NY, USA. Kionix is a wholly owned subsidiary of ROHM Co., Ltd. of Japan. The company is developing sensor solutions and the study was done by the author as a part of the work aiming at improvements of the flexibility and usability of the sensor solutions in question. The results were peer reviewed by a selected group of the R&D team members. Kionix offers high-performance, low-power accelerometers, gyroscopes, and 6-axis combination sensors plus comprehensive software libraries that support a full range of sensor combinations, operating systems and hardware platforms. Kionix's newest product innovations include full-featured, low-power, 2x2x0.9mm accelerometers, an ultra-thin accelerometer at only 0.7mm high, and the breakthrough KMX61G, a high-performance, low-power, magnetometer-accelerometer device with
integrated sensor fusion software and auto-calibration algorithms to deliver the industry's first highly accurate gyroscope emulation.

A constructive approach was used to explore how to build a simple and portable connectivity independent protocol to control and access data of multiple sensors in a microcontroller based embedded development platform. Sensor settings were controlled and the data passed from the embedded device to a PC client application(s) e.g. via a universal serial bus (USB, emulated as a serial port) and using Bluetooth low energy (BLE). A prototype of the protocol was built and an example scenario of how to use the protocol was provided. Basic operations were included as read/write sensor data and polling General Purpose I/O (GPIO) line(s) were included in the protocol implementation.

The construction part of the study covers the prototype design, including protocol message definitions and implementation which was based on the earlier study papers related to embedded devices’, sensors and protocols. Protocol implementation was integrated as a part of the sensor evaluation kit. The evaluation kit concept is described in chapter 4. One of the objectives was to implement the protocol solution for a wearable device and to generalize the protocol library for an easy porting to other embedded device platforms. Embedded device development platforms provide libraries and communication stacks for message transferring to and from devices using USB, Bluetooth and Wi-Fi connections. These platforms also provide digital control interfaces for sensor control and data transfer over Inter-Integrated Circuit (I2C) and Serial Peripheral Interface (SPI) buses. The common libraries provided by the platforms can include for example scheduling, GPIO handling, UART and common definitions to build embedded applications.

The design aspects of the protocol development were reusability and portability due to today’s wide range of embedded development platforms. Figure 1 presents an overview of the protocol concept. The protocol engine for processing protocol messages and connectivity adapter interfaces for enabling message transfer from the connectivity bus2 adapter to connectivity bus1 adapter and vice versa.

![Figure 1. Interfaces of sensor communication protocol.](image-url)

In the example shown in Figure 1 the connectivity bus2 adapter interface is realized to provide Bluetooth connection to receive/and send sensor dedicated messages to the embedded device. Connectivity bus1 presents the adapter interface for the TWI connection, such as I2C, for communicating with the sensor device. When protocol
interfaces are implemented the protocol functionality becomes operative and the sensor can be remotely controlled and the data can be read from the device.

1.2 Research motivation

Most of the data centric protocols offered by the WSN and IoT technologies were difficult to understand and complex to implement. Those were designed to run on top of the TCP/IP or UDP protocols and they didn’t take into account the agnostic approach of their design. There are solutions in the market providing I2C/SPI host adapters with hardware and software support to connect a sensor to a PC. Those solutions are expensive, usually bound to the wired connections and set unwanted limitations such as voltage levels and operation speeds for sensor communication bus. (Nano River Technologies, 2016; Total Phase, 2016) There are many microcontroller base embedded device platforms exists where the sensor can be connected via digital interfaces (Arduino, 2016; Embedded Linux, 2016; iProtoxi, 2016). In this kind of an environment the sensor communication protocol provides a valuable solution and support for the sensor control and data transfer over the wireless and wired connections. A wide range of microcontroller base embedded platforms can be supported when porting the protocol. Minimal software development effort is needed for introducing a new platform for testing and developing.

1.3 Research method and questions

The research method which was used in this thesis is based on the constructionism aspect and is in line with the design science research approach. The design science was developed for the IS research because of the nature of the discipline. Traditional science tries to search, develop and verify theories and also predict the human behavior. In the Design science research, artifacts, in other words man-made objects, which don’t form naturally, are designed, constructed and evaluated. (Hevner et al., 2004)

This thesis strives to answer the following question:

- What kind of protocol provides a simple communication over wired and wireless communication connections with a sensor in embedded devices?

1.4 Structure of the thesis

Chapter 2 gives an overview of embedded systems, MEMS sensors and earlier studies of protocol development of this field. Chapter 3 describes the research methods and explains how this research was conducted. Chapter 4 describes the idea of the sensor evaluation kit concept and the environment, where the artifact is developed and includes detailed descriptions about the artifact and its evaluation. Chapter 5 presents the conclusion and the implications of the research.
2. Earlier knowledge

The system on chip is introduced because the whole computing and other electronical systems are tightly placed in a single chip providing the base for the present day embedded devices, inertial sensors included. The sensor communication protocol was developed as an application layer protocol so the WSN and IoT data centric protocols were studied. This chapter will provide basic information in order to understand the complexity of the sensor software develop environment. The primary research work concentrates in exploring the software data centric sensor protocols which are developed in WSN and IoT.

2.1 Embedded systems

In general, embedded systems refer to a certain type of computing system, the purpose of which is to perform some specific function. An embedded system usually contains a microprocessor, memory, GPIO, and some electronics and mechanical parts related to a specific function. Nowadays the embedded systems are based on the microcontrollers including the CPU with integrated memory and peripheral interfaces. (Wikipedia, 2016a) Another definition for the embedded systems is; collections of programmable parts connected to Application Specific Integrated Circuits (ASICs) and other digital components. The components interact continuously with the environment via sensors and actuators. The typical programmable parts are microcontrollers and digital signal processors (DSP). Embedded systems are designed with ad hoc approaches which are based on earlier experience and knowledge. (Balarin, 1997)

The SoC is an integrated circuit (IC) which integrates to all computing and other electronical systems into a single chip. It may contain analogue, digital, mixed-signal and, more often, radio-frequency functions. Below is an example block diagram of a simple system on the chip. (Wikipedia, 2016c) See Figure 2 as a simple example of the system on the chip.

![Figure 2. Generic overview of the SoC (Nordic Semiconductor, 2016b).](image-url)
In the SoC the processor may be standard or customized. It is dedicated to some special task, for example media processing. The SoC can include several processors and other generators of bus cycles, like DMA controllers. The interconnection of the processors can use a variety of mechanisms, such as shared memory and message delivering between the hardware entities. The MEMS sensors can be connected to the system on chip via digital interfaces such as the I2C and the SPI. The physical sensor interrupt lines are connected to the SoC via the GPIO system(s) offering interrupt detections. See Figure 2 system on chip provided hardware blocks. (University of Cambridge, 2014)

Firmware is an embedded software program which simplifies the communication with the required hardware functionality. The firmware software program is integrated to the hardware unit or integrated circuit with fixed configuration, which cannot be changed during the execution. The relationship between the hardware and the firmware is considered to be interdependent, and the qualification testing of the hardware requires that the firmware be resident on the hardware. (Summers, 2011)

2.2 MEMS technology

Nowadays the silicon micro sensor and micro–total analytical systems (TAS) devices are all encompassed as the Microelectromechanical systems (MEMS) technology. Generally the MEMS components/devices are from 20 micrometre to a millimetre in length i.e. 0.02 to 1.0. MEMS consist of a central unit for processing data and components which are interacting with its environment. They are called micro sensors. When the semiconductor device fabrication technology was invented and ready to use in small devices like MEMS they became practical. (Wikipedia, 2016b) MEMS devices are mostly fabricated to the silicon and are made by using techniques which are often derived from the microelectronics industry (Du & Bogue, 2007). MEMS device’s solution and application in the MEMS market is targeted to the automotive, aerospace, industrial process control, electronics instrumentations, office equipment’s, appliances and the telecommunication. MEMS technology is expected to grow and there are lot of opportunities in the commercial markets because of the low-cost, high-functionality and the small size of the devices. (Leondes, 2007)

MEMS technology has been successful in physical sensing context and a wide range of small devices have been developed; such as motion, microphone, pressure and air mass sensors (Du & Bogue, 2007). Inertial sensors are devices to generate the repose of physical motion. The physical motion can be linear movement or rotation, which is transformed into electrical signals. Accelerometers, magnetometers and gyroscopes are most common type of inertial sensors and are they based on MEMS technology. (Beeby, 2004)

2.3 Protocols

The Open Systems Interconnection model (OSI) defines a networking framework of how applications can communicate over the network. It defines and guides the vendors and developers to build digital communication products and software programs which interoperate together. (Kroon, 2014) The OSI model consists of seven layers; physical, data link, network, transport, session, presentation and the application. It should be noticed that today’s protocols do not always use the seven layers model; for example the TCP/IS using the 6-layer rather than the 7-layer model. In the following Figure 3 presents the OSI 7 layer and they are drawn so that the layer 1 is at the bottom and layer seven is at the top. (Briscoe, 2000)
The physical layer defines the mechanical, electrical and functional characteristic of the networks (Zimmermann, 1980). The purpose of the Data link layer is to provide a procedure to transfer data between network entities and also to detect possible errors in transmission. The typical protocols are PPP and the HDLC. The Network layer is responsible of maintaining and establishing the connections. For example the IP protocol is functional in this layer. (Briscoe, 2000) The Transport Layer ensures the data reliability and integrity. The intent is also to relieve the session layer. (Zimmermann, 1980) For example the TCP protocol functions in this layer (Briscoe, 2000). The Session layer provides the structure for controlling and communication (Zimmermann, 1980). It provides two presentation entities to exchange data with each other (Briscoe, 2000). In the Presentation layer application data is either packed or unpacked. The data, which is unpacked for the use of applications and the packed data, is for transferring. (Zimmermann, 1980) In the Application layer the applications (end user) and the application protocols are functional. For example the protocols FTP, TELNET and SMTP live in this layer. (Briscoe, 2000)

UART is the one of the oldest and still predominant asynchronous interface for data transfer. Some implementation of UART also provides the synchronous mode. When the UART interfaces are integrated to the microcontroller base device only the physical layer (layer 1) and the data link layer (layer 2) from the OSI model are implemented. The layer from 3 to 7 is often implemented in software applications. (Soffel, 2003)

The Bluetooth core specification 4.0 defines the Classic Bluetooth, Bluetooth high speed and the Bluetooth low energy. It defines three layers (OSI model) physical, logical and the Logical Link Control and Adaptation (L2CAP) layers. The Physical layer is used to transport one or more logical links which supports the asynchronous traffic. Data packets which are sent to the logical links are multiplexed onto a physical link. The logical layer is managing the physical connections between the devices. L2CAP layer provides a channel based abstraction to applications and services by multiplexing and de-multiplexing channels over logical links. (Rivero, 2013)

Different kinds of embedded devices such as smartphones, smartwatches, netbooks and tablets are exchanging information in shifting networking environments. Different kinds of networking environments require efficiency, mobility support, adaptability, reliability, security, and timeless quality properties. Most of the applications and

---

**Figure 3. The 7 layers of the OSI model (Briscoe, 2000).**
services are built on top of the middleware relaying the Request-Response (RR) and Publish-Subscribe (PS) paradigms for the information exchange. These kinds of paradigm base protocols are running in the application level in the irrespective low level protocol stack. (Rodríguez-Domínguez et al., 2012)

Those two are commonly used paradigms of building messaging protocols. The RR type of a model is evolved from the HTTP protocol. This is where the other side issues a request and the other side sends the response message. So the RR is widely used in the distributed systems where the sender is requesting information and the receiver is sending exactly the same information which was requested. The RR pattern can be characterized so that the data sender is pulling the data from the destination by sending a request message. The Publish and subscribe pattern does not need to know the actual endpoint where the data is sent. Usually there is a message broker which keeps a track of who is publishing and who wants to receive the data. Depending on the implementation of the protocol and the environment the publisher may not know how it is receiving the data (subscribe). There are also other messaging patterns such as point-to-point (P-P) or remote procedure call (RPC). (Moyer, 2015)

When the data transfer must be reliable and processed in a timely manner the RR paradigm is often considered and used. The RR model has a relay on the usually one-way request where the response is only the status message, but there are also requirements for the batch request where the several requests are batched together to improve the protocol efficiency. When supporting the mobility, non-blocking and on-to-many messaging distribution; such as event notification, the PP is widely used. In the protocol constructive point of view the software engineers will focus on communication semantics and the quality properties rather than focusing on the communication mechanisms themselves. The PP paradigm emulates the human procedure of publishing and subscribing. The Subscriber informs the interests in a certain matter and automatically receives information each time it is released. (Rodríguez-Domínguez et al., 2012)

2.3.1 Sensor nodes

Wireless Sensor Network technology is defined as a network of sensor nodes which sense the environment and then the communicated information is gathered using the wireless links. The WSN has become more and more prominent in the recent years from a technical and commercial point of view. Areas like industrial automation, environmental monitoring and transportation are potentially growing markets. The targeted applications and solutions require sensor data transferring and data collection alongside the traditional networks and the new transferring technologies. WSN technology means that the sensor node or nodes are integrated into the traditional network infrastructure via gateways. Battery operated sensor and actuator (SA) devices with unlimited storages and computing capabilities are sending data via gateways for the applications and servers for processing the data or storing the data for later analysis. (Hunkeler et al., 2008)

A sensor network consists of sensing (sensors, measurements), computing and communication elements. It works as a tool for observing and reacting to certain phenomena. Typical sensor base applications are data collection, monitoring and medical telemetry. Also controlling and activation type functionalities are tied to the sensor based application areas. Four basic components are involved in the sensor networks: remote or localized sensors, connection to network of peer to peer
connections, central point for data collections and computing resource for the data handling. (Sohraby, Minoli, & Znati, 2007)

One of the wireless links is a simplified version of the Bluetooth technology, Bluetooth low energy. It is designed for transmitting a small amount of data efficiently to other devices. Bluetooth low energy operates at 2.4 GHz ISM and allows 1 Mbit/data rates for up to 10 meter distances. As the name indicates, low energy version of the Bluetooth also provides a way to significantly save power. (Buratti, Conti, Dardari, & Verdone, 2009) BLE is based on a single hop solution which is more applicable to areas such as health care, consumer electronics, smart energy and security. The ZigBee, 6LoWPAN or Z-Wave was being developed form a networking and industrial point of view. Those nodes can also operate a data collection points and routers. (Gomez, Oller, & Paradells, 2012)

There are basically two types of sensor node architectures; the source node architecture and the hierarchical processing architecture. The sensor nodes are not just sending the raw data but they are also responsible for data fusion. The nodes are using their processing abilities locally and transmit only the subset of data. In the WSN architecture there are remote clusters which contain sensor nodes which might include single hop or multi hop nodes. The nodes are connected to the clustering node which can do the data processing before sending the data to the final processing node. (Sohraby et al., 2007)

In general a large number of sensor nodes exist in the sensor networks and so the identification in wireless networks might be difficult. Querying a particular set of the sensors often leads to repeated transmissions of data from the sensor nodes. The data centric protocols were valuable for selecting the particular set of forwarding sensor nodes and for the construction of the data aggregation form for the sensor data transmission. This varies from the traditional address base routing which is based on the nodes addresses. The data centric approach is based on a technique where the sink is sending queries in some particular cluster region. (Zia, 2015) Queries are based on attribute naming which describes the characteristic of the request data. Then the sink waits for a response to a query which was addressed in the selected region(s). (Akkaya & Younis, 2005)

The following list contains the most important WSN based system(s) data centric protocols:

- **SPIN** is the first data-centric protocol based on data negotiable between nodes to eliminate redundant data to produce energy savings. SPIN nodes can use three types of messages to communicate. The ADV advertises that the node has a new data to share. The REQ message is for when the data is needed to be received. The DATA-DATA message contains the actual data. SPIN family includes four protocols which have a different characteristic and target purpose. (Rehena, Roy, & Mukherjee, 2011)

- **Directed Diffusions** this protocol was important milestone in WSN data centric protocol development. It provided the scheme to get rid of the unnecessary operations in network layers and so saving the energy. Value pair concept was introduced for the data queries (messages). (Akkaya & Younis, 2005; Intanagonwiwat, Govindan, & Estrin, 2000) Rumor routing and GRB are variations of the directed diffusion protocols. (Akkaya & Younis, 2005)

- **CADR and IDSQ** are protocols which strive to be a general form of the directed...
diffusion. The idea is to maximize the information gain and minimize the latency and bandwidth. Activating sensors which are close to a particular cluster or processing node and specific the event/data which to receive. (Akkaya & Younis, 2005)

- COUGAR has a sensor database system where the sensor nodes can be selected by the leader node to perform aggregation and transmit the data to cluster gateway or the final processing node. (Akkaya & Younis, 2005)

- ACQUIRE this approach views the sensor network as a distributed database. These suites for complex queries include sub queries. If the sensor node cache information is not up-to-date the nodes gather information from the neighboring nodes. (Akkaya & Younis, 2005)

2.3.2 Internet of things

The Internet of Things is defined by the IEEE as; a system which consists of networks of sensors, actuators, and smart objects whose purpose is to interconnect all things including every day and industrial objects making them intelligent, programmable and capable of interacting with humans and each other. The core idea of the Internet of Things is to connect all things to the internet and allowing them communication with one another and making some decisions. The IoT is commonly seen a wearable solution; meaning “carrying devices” such as smart phones, smart watches, pedometers, and heart rate monitors. These devices can be connected to the other devices over Bluetooth for sending and receiving data notifications. It is clear that the popularity of these devices has increased in the complexity of connection and data delivery point of view so new protocols are needed to be developed into the IoT world. (Atzori et al., 2010)

Today’s wearable technology is a leading category of IoT devices which are carried with. These devices include inertial sensors such as accelerometers and gyroscopes. These devices are divided into two categories single purpose devices and multifunctional devices. The single purpose devices usually include the sensors and actuators which collect the sensor data and perform simple tasks or actions. The multifunctional devices are capable of collecting and processing for example sensor data in several sources and presenting them to the user or transferring those into other devices. Heart rate monitors and pedometers are examples of the single purpose devices and the smart watches and glasses of the multifunctional devices. (Lewson, 2015)

There are several protocols developed in the first evolution phase of Internet of things and the main purpose of those protocols is to communicate between the embedded devices (D2D) and also with the server entities (D2S). Also the S2S protocols are developed to share the device data between the services. There are variations of those protocols and they are adapted to the different needs. (Schneider, 2013)

The following list contains the Internet of Things protocols which are meant to for SA data transferring in the IoT networks:

- CoAP, the Constrained Application protocol is designed be a web based protocol for the use of constrained nodes and networks. The protocol provides the request and response paradigm between the application endpoints. CoAP protocol was designed in energy save sense and targeted to building automation
and other machine to machine applications. (Shelby, Hartke, & Bormann, 2014). The CoAP was particularly targeted for low power sensors for control and data transfer over the internet networks. (Wireless Sensor Networks Magazine, 2013) The CoAP is a client-server model based on the HTTP protocol. The protocol can act both roles; as the client and the server. Basically the CoAP is equivalent to the HTTP sending the request in the same way; requesting an action on a resource on a server. The CoAP handles these transactions over the UDP asynchronously. Protocol features; optimized efficient data transfer, uses proxies and provides interface for applications to use (Already using HTTP). The CoAP can send messages to one or more devices (multicast IP destination addresses) and also provide low header overhead and simple parsing, URI and content type support. The protocol was specially designed to 8-bit microcontrollers with small amounts of ROM and RAM. (Shelby et al., 2014)

- MQTT, the Message Queuing Telemetry Transport is an instant and broker-based publishing and subscribing type of protocol which runs over the TCP/IP. It was designed to be simple and easy to use and ideal for “push notification” types of messages. MQTT is widely used in the Android phone because of its scalability and simplicity. (Tang et al., 2013) Network addressing as communication between the sensor/actuator (SA) may be problematic because of the dynamic and temporary nature of the addresses. It is complex with large numbers of the sensor nodes connected to the WSN. The MQTT protocol was introduced collecting device data and transferring it to server. The protocol was designed especially for a machine-to-machine and for the mobile applications. MQTT is a client-server based solution where the client establishes the network connection to the server. The client publishes application messages that others might be interested in. Other client(s) can send subscribe requests to the subject of interest. Server side accepts the connections and messages and processes the PP messages. (Stanford-Clark & Truong, 2008)

- MQTT-SN, the Message Queuing Telemetry Transport protocol for sensor networks is an extension to the MQTT and was designed to low cost and low power SA devices and the network communication environment such as low bandwidth, high link failures and restriction of the message lengths. Differences to the MQTT protocol:
  - The connection message is split into two additional messages to deliver the will topic and will message to server.
  - Short message length and limited transmission bandwidth. The publish message is replaced by a short topic id; register procedure allows the client to register their topic name and receive a topic id. Topic ids can be used in subscribe and publishing messages.
  - Pre-defined topic ids and names are introduced. No registration needed.
  - Discovery procedure to help pre-configuration (address discovering).
  - Keep-alive procedure was defined for the support of sleeping clients.

The MQTT-SN was designed keeping in mind the agnostic of the networking service meaning that the network which supports a bi-directional data transfer between server and client nodes. (Hunkeler et al., 2008)

- DSS, The Data Distributed Service is a real time middleware protocol and API standard providing real-time, high-performance and interoperable data delivery
using the publishers and subscribes paradigm. The message brokers and servers are eliminated in the DSS. So it is simplifying the development and also minimizing latency and increasing the reliability. The protocol can manage small devices, and connect large sensor networks. (Object Management Group, 2016)
3. Research approach

The approach in this study is called the design science research approach. In this chapter this method is explained in more detail.

3.1 Design science research

Hevner et al. (2004) explains that the design science research is a science where artifacts are designed, constructed and somehow demonstrated and evaluated. Artifacts are things which are not encountered in nature, i.e. they are made by humans. Two processes can be identified within the design science: building process and evaluating process. The resulting products of the research are constructs, modes, methods and instantiations. The research should be replicated using the same definitions and settings as in the original implementation/constructon.

The purpose of information systems is to improve an organizations and individual effectiveness and efficiency in daily work and business. Information systems and organisations consist of structures, people, technologies and processes connected together. (Hevner et al., 2004) March and Smith defines the information technology as follows: Information technology forms data in a form that is meaningful to the recipient and performs real or perceived value in current prospective actions or decisions. (March & Smith, 1995) The information systems research is often carried out using the design science research method and it also applies theories from other disciplines such as economics, computer science, and the social science. Information systems try to solve problems which occur when organisations meet information technology. (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007)

Two research paradigms characterize the IS research: behavioral science and design science. Behavioral science guides the research and tries to explain or predict the phenomenon. Often the design science research tries to build and evaluate the artifacts from a business’ “need perspective”. In other words the paradigm of the behavioral-science is to find “what is true” and in contrast design science seeks- “what is effective”. An important thing which needs to be remembered is that the design science research is different than designing or building IS systems. (Hevner et al., 2004)
3.2 Design science framework

Hevner et al. (2004) presents the information systems research framework as understanding the relations between environments, IS research and knowledge base. See Figure 4 for the IS research framework.

The starting point of the research can be a “business need” or other phenomenon, which needs to be investigated. This is a so called “problem space”. This problem space can occur in the context of certain people, a community or an organization (environment). If the starting point of the research activities is “business needs” rather than some other phenomenon, the research projects are usually more successful. A knowledge base such as prior research on the relevant area, could serve as a valuable input both for the building and the evaluation process. This includes the data and methodologies which can be used to support research activities by providing a set of theories, instruments, constructs, models and frameworks. Example theories, models and frameworks are used in the definition phase of research and the methodologies in the phase of evaluating process. (Hevner et al., 2004)
Seven suggested guidelines for design science research presented by the Hevner et al. (2004) illustrated in Figure 5.

1. **Design as an artifact**: An IT Artifact is created to serve a purpose and it’s identifiable and viable. The Artifact addresses an important organizational problem. IT artifacts can be instantons, constructs, models or methods.

2. **Problem relevance**: Technology based solutions are provided and addressed to the relevant business problems. Knowledge and understanding are acquired as enablers for the research.

3. **Design evaluation**: In order to evaluate an artifact it needs to be demonstrated. Evaluation is a vital part of the design science research process.

4. **Research contributions**: Well-defined and conducted design science research needs to produce contributions to the designed artifact area. Three types of contributions are identified; the artifact itself, foundations, and the methodologies.

5. **Research rigor**: Using rigorous methods for constructing and evaluating the artifact is essential for reproducing the results. Constructed artifact often relies on theoretical models, so the rigor is derived from the knowledge base.

6. **Design as research process**: For finding the best and optimal design, the design science process is iterative and uses various methods and theories. The search
process is tried to find in order effective solution for the phenomenon or the problem under investigation.

7. **Communication of research:** Reporting the research must be addressed to an academic audience (rigorous) and to a professional audience (relevance).

### 3.3 Design science research methodology

An earlier research of design science has been inputted to develop the design science research methodology (DSRM) model. There has been the same core staring point in all design science studies; design and development. In some cases the research activities are divided into separated activities and in others it is seen as iterative process. There are also the different views for the evaluation of the design science research e.g. single act of demonstration or more a formal way which is tightly connected to some evaluation method which needs to be followed. DSRM model is a combination or a synthesis of the prior design science studies. (Peffers et al., 2006)

Peffers et al. (2006) introduced the DSRM process based on previous design science principles. The design research methodology is presented and conducted in this design science research. The methodology supports the commonly accepted design science frameworks. The DSRM includes practices, principles and procedures to direct the research work. DSRM defines six steps for DS research; problem identification, objectives for the solution, development including the design, showcasing, evaluation and communication. Process model of the DSRM can be seen in Figure 6.

![Process of the DSRM methodology](Peffers et al., 2008).

In the DSRM method there is no need to execute activities in sequential order. Researcher can start at any of the first four activities and continue outwards. The four starting activities can be chosen depending on the nature of the research entry.

If the observation of the problem or suggested future research is based on the earlier research papers, the problem centered entry is chosen. An artifact based on “business needs” can be addressed to an object-centered entry. For example an existing artifact is derived from another domain which can be a base for the design- and development-
Problem identification addresses and defines the research problem and sees the value of the solution which is to be built. This provides the advantages; meaning that the researcher and the audience are motivated to find out a suitable solution. Knowledge and resources are needed to execute this action. Objectives of a solution can be either quantitative, or qualitative. The quantitative research method tries to prove that the solution which was built is better than the current ones. When the objectives of a solution are qualitative based the artifact is expected provide solutions for the problem. The inferred goals from the problem definition phase are accomplished in this step. In the Design and development step the artifact is developed. The functionality and architecture are defined for the solution and the artifact is created. The Demonstration step shows how efficiently the artifact solves the problem identified in the problem identification step. The method to demonstrate the efficiency of artifact of can be experimentation, case study, proof of concept, or other suitable practices. The Evaluation step measures how the solution designed and constructed by the artifact supports the problem. In this part an activity comparison of the objectives and an actual solution is done. Performance measurements or surveys, client feedback or simulations can be used as an evaluation method. At the end of this activity the researchers can decide to iterate back to step 3 and try to improve the solution. Communication is the final step in DSRM model. The aim of this step is to communicate the problem, the artifact, its utility and novelty, rigor design and effectiveness to the researchers and other relevant audience. (Peffers et al., 2006; Peffers et al., 2008)

3.4 Process implementation

The DSRM methodology was used to conduct a research, described in Figure 7. The problem-centered approach was chosen as a starting point entry for the research. DSRM is particularly well suited to situations where the researcher is part of the problem solving or product development team. This chapter introduces the steps and activities used in the study.

Figure 7. Research process for developing sensor communication protocol.
**Problem centered approach**, there are plenty of papers related to sensor systems and a sensor protocol exists. Sensor node papers describe the whole system used to interact with sensor data. For example an embedded IoT world provides its own protocols for accessing sensor data. There are “send/response” and “publish/subscribe” types of protocols patterns introduced.

**Problem identification and motivation**, “Off-the-shelf” types of adapters for sensor evaluation kits already existed but they did not fully meet the criteria for evaluation, demonstration and testing the newly manufactured sensors. The idea of the microcontroller base sensor communication protocol was born and the requirements for the prototype were written down. The objective was to develop a protocol to read and write sensor data over different kinds of communication busses for microcontroller base devices.

**Design and development**, the protocol design and development was based on a formal software developing process. The designing, implementing and testing was done using iterative process. UML modelling language was used to describe the protocol structured view and the dynamic behavior in reference platform. Protocol messages and values were defined in the design phase and also supplemented when new requirements were occurred in the development phase.

**Demonstration**, Command line and GUI evaluation kit clients were used to demonstrate the protocol running in the microcontroller based device. See chapter demonstration.

**Evaluation**, The evaluation of the artifact was done by gathering the feedback of how the protocol was working an embedded device. Also the protocol was ported to another embedded platform to identify problems as well as pros and cons of the design.

**Communication**, this study follows the design science research principle. Process includes narrow literature review, the research process, a demonstration, the evaluation and the conclusion.
4. Development of artefact

This chapter describes the designing and implementation of the artefact; the sensor communication protocol. Also the sensor evaluation kit concept is introduced because the protocol is an important part of it.

4.1 Sensor evaluation kit concept

The evaluation kit client software runs on PC implementing the sensor driver functionalities for a specific sensor. Typical scenarios of sensor functionality using the evaluation kit are data streaming; polling, interrupting and buffering mode. Advanced scenarios usually refer to integrated ASIC features such as orientation, tap/double tap, activity detecting and free fall algorithms which are based on inertial acceleration. These features can be fine-tuned using the evaluation kit and afterwards ported to for example some specific embedded device platform. The evaluation kit software including the sensor communication protocol itself runs on the target the sensor evaluation kit concept is illustrated in Figure 8.

![Evaluation Kit HW](Image)

**Figure 8. Sensor evaluation kit concept.**

Two versions of the evaluation kit client software are present: the graphical user interface and the command line interface software. The GUI version provides an easy way to use and visualize the system, including real time data view, data logging and accessing sensor registers as well as demo applications.

The command line interface offers an engineering tool for technical people and software developers for a quick development cycle. With this interface it is quick and easy to test low level sensor features, for example double tap and wake up detections.

Sensor evaluation kit hardware and software can be divided into two different types: the “off-the-shelf” or the “microcontroller based” approach. The “off-the-shelf” approach means that the device already has the firmware and needed software APIs available to communicate with the sensor. The hardware where the sensor is connected and software are provided by the vendor. The second one, a more complicated approach, is “microcontroller based”, where all the firmware and software APIs must be implemented so the sensor communication protocol is applied this solution.
4.1.1 Off the shelf approach

Aardvark is one of the examples about the “off-the-shelf” adapters and can been seen in the following Figure 9. It provides serial messaging using the I2C and SPI protocols with a sensor and provides USB connection to PC. (Total Phase, 2016)

![Figure 9. “Off the shelf” sensor evaluation kit concept.](image)

In the “off-the-self” approach the hardware and the firmware is provided by the vendor. Also PC side driver software and the APIs are delivered for the user. In this example the connectivity bus1 connects the sensor to the adapter and the connectivity bus2 connects the adapter to the client. The sensor is connected using I2C or SPI bus interfaces to host the adapter; connectivity bus1 in the Figure 9. The connectivity bus2 in this example is a USB bus and the implementation is provided by the vendor. In this approach only the software API provided by the vendor is taken into use. Usually this kind of a solution is expensive and sets some unwanted limitations, such as voltage levels and bus operation speeds.

4.1.2 Microcontroller approach

In the “microcontroller based” solution the firmware and the software APIs must be implemented to provide communication between the evaluation kit client and the microcontroller based device. A high level concept about the sensor evaluation kit using the microcontroller base approach can be seen in the Figure 10. Examples of these kinds of environments are Kionix IoT Board based on Aistin, Embedded Linux, Beagle bone black, Arduino Uno, Cypress PSoC® 4 S-Series and FRDM-K64F.

![Figure 10. Microcontroller based sensor evaluation kit concept.](image)

Firmware software consists of a main application, connectivity buses, drivers and the protocol engine to manage sensor communication protocol messages.

The Figure 10 above shows the protocol engine which is needed in the firmware. The connectivity bus2 is needed to commutate with the surroundings; i.e. the evaluation kit client software in client hardware. An example of the connectivity bus2 implementation might be the USB UART and BLE communication. The intent of the protocol is to
provide operational readiness for reading and writing the sensor registers. For example a read message can be sent from the evaluation kit client to access certain sensor data in the device. When the protocol engine in the target device is read a certain sensor device registers it and then sends the response message with data back to the client. The idea of the protocol is simple but it should be built carefully to ensure easy porting for other embedded platforms. In this prototype phase the wearable device platform was used for testing and to see how well the evaluation kit concept works.

4.2 Protocol development environment and system architecture

The protocol was built using the wireless and wearable solution which set its own challenges for testing and debugging the application in the device. The following subchapters describe the system architecture and the development environment used in the construction phase of the protocol.

4.2.1 Development platform and sensors

Kionix IoT board platform is based on the Aistin blue wireless and wearable solution. It is used for developing a sensor communication protocol to interact with multiple sensors. The Aistin blue was developed around the Nordics semiconductor low energy nrf51822 SoC, which includes a 32-bit ARM® Cortex™ M0 CPU with 256kB flash + 32kBkB RAM. The embedded 2.4GHz transceiver supports both the Bluetooth smart and the Nordic Gazell protocol stacks. A flexible 31-gpio pin system allows the I/O like serial interfaces. The Aistin IoT device integrates the Bluetooth low energy, the USB connector (FTDI), and the sensors to establish the powerful sensor based IoT device platform. (iProtoxi, 2016)

The board includes assembled sensors provided by the inertial sensor manufacturer Kionix for detecting the 3D-acceleration, 3D-magnetism and 3D-rotation. The following sensor products are included in the Kionix IoT board:

- KXG03 advanced 6-axis accelerometer/gyroscope combined device. The Low power architecture, I2C and SPI digital communications and auxiliary bus for external sensors. Gyroscope has a full-scale range of ±256, ±512, ±1024, and ±2048 °/sec. Accelerometer has a full-scale of user-programmable ±2g/±4g/±8g/±16g. (Kionix, 2016c)

- KMX62 includes tri-axis magnetometer and tri-axis accelerometer, I2C digital communication and ultra-low power. User-programmable ±2g/±4g/±8g/±16g full scale range for the accelerometer and a +/-1200 uT range for the magnetometer. User-selectable ODR from 0.781Hz to 1.6kHz. (Kionix, 2016a)

- KX122 includes tri axis accelerometer, I2C and SPI digital communication and the low power or high resolution modes. Orientation, tap/double tap, activity detection and free fall algorithms integrated. User-programmable +/-2g, +/-4g or +/-8g range and 2048 byte large FIFO buffer. (Kionix, 2016b)

- BM1383GLV is a piezo-resistive air pressure sensor which offers high accurate measurements its entire operational temperature range -40°C to 85°C. Pressure measurement ranges 300hPa to 1,100hPa. Includes integrated temperature sensor. (Rohm Semiconductor, 2016)
4.2.2 Software tools and compiler

Nordic semiconductor provides instructions to set-up the software development environment for nRF51822 SoC. (Nordic Semiconductor, 2016b) Keil V5.14.0.0 is used as IDE and the ARM compiler version 5.05u1. (Arm Group, 2016) CMSIS package installer is used to install the nRF51_SDK_v9.x.x SDK (Nordic Semiconductor, 2016a)

The nRF5 SDK provides a developing environment for the nRF5 Series based devices including a wide range of drivers, libraries, examples and APIs for using the pre-compiled and linked soft device protocol stack. The soft device s110 stack is used because the device behaves as an end node from the Bluetooth point of view. See high level Figure 11 of the nRF51822 software architecture.

![Figure 11. nRF51 based IoT device software modules (Nordic Semiconductor, nRF51822, 2016).](image_url)

Figure 11 shows the most important SDK libraries used by the application which runs the sensor communication protocol implementation. TWI provides the I2C digital communication with a sensor and the GPIOTE for interrupt detection. UART is used for serial port communication and the s110 soft device for the Bluetooth communication. (Nordic Semiconductor, 2016b)

4.2.3 System architecture

The prototype of the artefact; the sensor evaluation kit control and data protocol is running a part of the sensor evaluation system. The protocol development work was done on the Kionix IoT board and it is used as a reference development platform for the research. The Figure 12 shows in high level the software layers and the communication protocols from/to nRF51822 based the IoT device.

The sensor(s) are connected through the communication layer via connectivity bus adapters to the main application running in the device. The protocol engine is running in the application context for serving the messages received from the communication connectivity busses. When messages arise to the UART or BLE connection, the handler detects the messages and either processes them internally or interacts with the sensor by reading or writing the sensor register values. When a message is served the response is sent to the client via a communication adapter instance. In the BLE case the Android phone acts as a router to deliver messages to the evaluation kit client, which is running on a PC. In the USB UART case the message is delivered to the client over the serial communication bus.
Figure 12. Sensor evaluation kit software architecture.

The evaluation kit client implements the functionalities of a specific sensor and is acting like a sensor driver. The driver can implement functionalities such as data streaming or advanced scenarios as detections. The sensor communication protocol itself is running on target device for processing protocol message to reading and writing sensor data.

4.3 What was designed and developed

Following items where the author’s responsibility when designing, implementing and evaluating the artefact:

- Identifying technical and functional requirements.
- Protocol design.
- Reference implementation was figured out.
- Connectivity adapter interfaces were defined.
- Protocol’s dynamic behavior was planned and described in UML.
- Message formats were defined.
- The needed implementation was done.
- A demonstration was built using Python scripts.
- The results were peer reviewed.
4.4 Technical and functional requirements

Functional requirements:

- Protocol version query.
- Read sensor registers.
- Write sensor register.
- Request GPIOs state.
- Reading and writing the multiple simultaneous connected sensors.

Technical requirements:

- Generic implementation for supporting the porting to embedded devices.
- C99 C language standard.
- Protocol description and message definitions.
- A connectivity bus adapter interfaces.
- Example of connectivity bus1 adapter for the sensor communication.
- Example of connectivity bus2 adapters for communication to outside world of embedded device.
- Synchronous messaging.

Next release(s) features:

Some restrictions are now set for the protocol implementation. New features are listed but not yet implemented in this prototyping phase. The next iteration of the protocol will add more functionality and also performance improvements for the sensor data reading.

- Message length is limited due to BLE to 20 octets.
- Parity or CRC checks are not implemented for protocol engine.
- Error tolerance for handling protocol messages
- At least 100Hz ODRs are required for sensor data streaming.
- Message segmentation and reassembly.
- 9D-sensor data stream are required for the sensor fusion.
- Debug message improvements, clear define prototype functions for debugging.
- Asynchronous messaging.
- Bus configuration messages, speed, pin setup e.g. SDA, SCL.
- Friendly name set/get query.
- FIFO reading (reading data from sensor internal buffer).
4.5 Protocol design

The sensor communication protocol consists of several header and .c files. The files with an explanation are listed below.

- evkit_bus1_if.h
  - Includes the connectivity bus1 interface to connect sensor to protocol engine.
- evkit_bus2_if.h
  - Includes the connectivity bus2 interface to connect protocol engine to communication protocols to/from the embedded device.
- evkit_debug.h
  - Includes empty debug function which can be overwritten to this file.
- evkit_msg.h
  - Includes protocol message definitions and values.
- evkit_msg_handler.h
  - Protocol engine interface.
- evkit_msg_handler.c
  - Protocol engine implementation.
- evkit_types.h
  - Data types can be type defined in this file, if platform specific types are need.
- evkit_incoming_message_buffer.h
  - Incoming message buffer interface for bus2
- evkit_incoming_message_buffer.c
  - Implementation of the bus2 input message buffer.

4.5.1 Structured view (Reference implementation)

The sensor communication protocol consists of a protocol engine and it provides interfaces an opportunity to be implemented when the protocol is ported to the embedded devices. In Figure 13 there are two bus2 adapters and one bus1 adapter implemented. The bus2 adapters are realizing the bus2 interface and so on, providing the entries an opportunity to deliver and send messages to the embedded device. nRF51822 provides software libraries for the Bluetooth and USB connections. Both connections can be used for/in the message transferring.

The bus1 adapter realizes the connectivity bus1 interface to establish a connection via TWI interface to the sensor. GPIO interface is used for detecting the GPIO line state. A prototype implementation assumes that the firmware configures the GPIOs initialize values; pin as input, polarity (low/high) and pull up resistor (down, up). Sensor interrupt lines can be connected the SoC GPIO system for detecting the interruptions.
Figure 13. Structured view of sensor communication protocol and connectivity adapters with UML notation.

The protocol engine itself is responsible for acting as a router between the connectivity bus adapters. The engine translates the connectivity bus2 messages to function the calls for the connectivity bus1 interface. The connectivity bus1 adapter maps a function call to the TWI or other sensor communication module, e.g. SPI. In other words the bus1 adapter is implementing the actual interaction with the sensor. The protocol design is based on polling of the messages through the connectivity bus2 interface and handling the messages synchronously, i.e. communication via the connectivity bus1 interface to the sensor. When the sensor slave address is sent by the bus1 adapter module to the TWI connection module which in this case means I2C communication with a sensor, each device on the bus compares it with the internally stored slave address. If the address is a match, the device considers itself addressed by the master.

Another design aspect of the protocol engine is the context switching between the threads or the processor states. These don’t need to be taken into account inside the implementation. If necessary, context switching can be implemented for the bus adapters provided by the interfaces. The heap memory allocation is avoided in the protocol engine and all the structures and buffers are allocated in the stack memory. Otherwise, the memory allocation interface would have been needed.

4.5.2 Connectivity bus adapter interfaces

The two connectivity bus adapter interfaces were designed. The functions were implemented and exported in such a way that they were available via static instance at the compiler time. The function pointer address of the connectivity bus1 interfaces was set through the protocol engine interface during the initialization phase and the address could not change the runtime. The bus2 adapters address was passed to the engine during the execute function call and could be changed during the runtime. For example in a USB connection it was possible to change to the BLE during the execution call.
The connectivity bus2 interface was designed to offer API for the adapter modules implementing protocol communication to and from a device using platform specific libraries. The function pointer interface contained the rx and tx functions for sending and receiving data bytes. Evkit_bus2_t defines the function pointer interface and the user set the functions to the interface. See bus2 example initialize below.

```c

    evkit_bus2_t m_bus2_ble;

    evkit_bus2_t* evkit_bus2_ble_init(void) {
        memset(&m_bus2_ble, 0, sizeof(m_bus2_ble));
        m_bus2_ble.rx = ble_rx;
        m_bus2_ble.tx = ble_tx;
        return &m_bus2_ble;
    }
```

The BLE initialize function was implemented to the adapter module and the pointer to interface was returned and assigned to the protocol engine. The interface functions; ble_rx and ble_tx functions were also implemented in the adapter module.

The connectivity bus1 interfaces were designed to offer API for the adapter module which was providing digital communication with sensors. The function pointer interface contained read and write functions for accessing the specific sensor data and the state function to request certain a GPIO line state. The evkit_bus1_t defines the function pointer interface and the user set the functions to the interface. See the bus1 example initialize below.

```c

    evkit_bus1_t m_bus1_twi;

    evkit_bus1_t* evkit_bus1_twi_init(void) {
        memset(&m_bus1_twi, 0, sizeof(m_bus1_twi));
        m_bus1_twi.read = read_i2c;
        m_bus1_twi.write = write_i2c;
        m_bus1_twi.state = gpio_state;
        return &m_bus1_twi;
    }
```

The TWI initialize function was implemented to the adapter module and the pointer to the interface was returned and assigned to the protocol engine during the protocol engine initialize time. The interface functions; read, write and GPIO state functions were implemented in the adapter module. The read and write methods were using the TWI connection module provided by the nRF51822 software. The state function was using the GPIO module for detecting the line state changes. See interfaces in next subchapters.
Bus1 adapter interface

/**
 * @file evkit_bus1_if.h
 * @brief bus1 adapter interface
 * @copyright Kionix Inc, 2016
 */
#include "evkit_types.h"
#include "evkit_msg.h"

/**
 * @brief Function to read sensor data through bus1 adapter instance.
 * @param[in] sad Sensor device address (Slave address).
 * @param[in] reg Sensor register to be read
 * @param[out] data Buffer for sensor data
 * @param[in] length Read length
 * @return uint8_t Returns EVKIT_SUCCESS when operation successfully,
 * Otherwise EVKIT_BUS1_ERROR
 */
typedef uint8_t (*evkit_bus1_read) (
    uint8_t sad,
    uint8_t reg,
    uint8_t *data,
    uint16_t data_length);

/**
 * @brief Function to write sensor data through bus1 adapter instance.
 * @param[in] sad Sensor device address (Slave address).
 * @param[in] reg Sensor register to be write
 * @param[in] data Buffer for sensor data
 * @param[in] length Write length
 * @return uint8_t Returns EVKIT_SUCCESS when operation successfully,
 * Otherwise EVKIT_BUS1_ERROR
 */
typedef uint8_t (*evkit_bus1_write) (
    uint8_t sad,
    uint8_t reg,
    uint8_t *data,
    uint16_t data_length);

/**
 * @brief Function to request GPIO state.
 * @param[in] gpio Gpio number.
 * @param[out] gpio_state State of the GPIO line.
 * @return uint8_t Returns EVKIT_SUCCESS when operation successfully,
 * Otherwise EVKIT_GPIO_INVALID
 */
typedef uint8_t (*evkit_bus1_gpio_state) (
    uint8_t gpio,
    uint8_t *gpio_state);

/**
 * @brief Bus1 interface for reading and writing the sensor register through the bus1 adapter.
 * The interface by which the protocol engine accesses the bus1 adapter
 * for reading and writing sensor data. All functions need to implemented
 * and export them such that they are available to the static instance at compile time.
 * Bus1 interface pointer is given to the protocol engine through the engine interface
 * init method. The bus1 pointer cannot change during the execution time.
 */
typedef struct
{
    evkit_bus1_read read; /**< function to read bus1 adapter data */
    evkit_bus1_write write; /**< function to write bus1 adapter data */
    evkit_bus1_gpio_state state; /**< function to request GPIO state through
                                   the bus1 adapter*/
} evkit_bus1_t;
Bus2 adapter interface

/**
 * @file evkit_bus2_if.h
 * @brief bus2 adapter interface
 * @copyright Kionix Inc. 2016
 */

#include "evkit_types.h"

/**@brief Function to receive data from bus2 adapter instance.
 * @param[out] data pointer to the data buffer, where the bus2 Adapter instance stores an incoming message data.
 * @param[in] length max buffer length, limited to 20 bytes.
 * @return uint16_t, bytes received.
 */
typedef uint16_t (*evkit_bus2_rx) (uint8_t * data,
                                 uint16_t length);

/**@brief Function to send data to bus2 adapter instance.
 * @param[in] data pointer to the data buffer, where the bus2 Adapter instance can read outgoing message.
 * @param[in] length amount bytes to be sent.
 */
typedef void (*evkit_bus2_tx) (uint8_t * data,
                             uint16_t length);

/** The interface by which the Evkit Protocol Engine accesses the bus2 adapters for reading and writing message data. All functions need to be implemented and export them such that they are available to the static instance at compile time.
 * Bus2 interface pointer is given to the protocol engine through the engine interface execute method. The bus2 pointer can be change during the execution.
 */
typedef struct
{
    evkit_bus2_rx rx; /**< function to poll bus2 data */
    evkit_bus2_tx tx; /**< function send data to bus2 */
} evkit_bus2_t;
4.5.3 Dynamic behaviour

*Initialize protocol engine and bus adapters*

The application initialized the bus1 and bus2 adapters and passed the function pointer interface to the protocol engine. See Figure 14. There can be several bus2 adapters which are realizing the interface, but only one bus1 adapter.

![Interaction diagram](image)

Figure 14. Behavior of the initialize message.

After the initialization the application starts polling loop for receiving the message from the bus2 adapter. The protocol is set to initialized state.
Protocol version querying

To set the protocol to a ready state the version query must be executed by the client. See Figure 15. When the valid version query message arises the protocol engine sends the response message to the client with the status code EVKIT_SUCCESS to indicate a successful state transition.

Figure 15. Behavior of the version query message.

If the application changes the bus2 adapter instance by detecting a new connection, the new instance is delivered to the protocol engine via an execute function. The protocol engine stores the new bus2 interface pointer and sets the protocol to the initial state. The transactions and buffers are then cleared in the protocol engine. Version request query is required for the new bus2 connection to set the protocol to a ready state to server its client.
Read sensor data

The Read message is received via the rx function pointer call and the message is extracted by the protocol engine. If the message is valid it is mapped to the bus1 interface call to request sensor data. See Figure 16.

The bus parameters, the sensor data buffer and length are passed to the bus1 adapter. The bus1 adapter uses the platform specific modules to read sensor data. If the sensor data reading is successful, the data is written to the buffer and the EVKIT_SUCCESS status code is returned, otherwise an EVKIT_BUS1_ERROR is received. The adapter module is responsible for implementing the retry counters if reading the sensor data fails.
Write sensor data

A write message is received via the rx function pointer call and the message is extracted by the protocol engine. If message is valid it is mapped to the bus1 interface call for request to write data to sensor. See Figure 17.

Figure 17. Behavior of write message.

The bus parameters, the sensor data buffer and length are passed to the bus1 adapter. The bus1 adapter uses the platform specific modules to write sensor data. The data is passed from the sensor buffer via connectivity bus1 interface and written to the sensor register defined by the bus parameters. If the sensor data writing is successful the EVKIT_SUCCESS status code is returned, otherwise EVKIT_BUS1_ERROR is received. The adapter module is responsible for implementing the retry counters if writing the sensor data fails.
Request GPIO state

A GPIO state request message is received via the state function pointer call and the message is extracted by the protocol engine. If the message is valid it is mapped to the bus1 interface call to request the GPIO line state. See Figure 18.

![Diagram](image)

Figure 18. Behavior GPIO state message request.

A specific GPIO line state is returned indicating either a low or a high state. If the operation is successful the EVKIT_SUCCESS status code is returned, otherwise EVKIT_GPIO_INVALID is received.

4.6 Message formats

This chapter describes the sensor communication protocol message format, definitions and individual messages.

4.6.1 General message format

The general message format of a sensor communication protocol is seen in Figure 19.

![Diagram](image)

Figure 19. Generic message format.

The protocol message consists of two parts: the message header of a 2 octets’ length and the variable part, which depends on the type of the message.

The whole length of the message is restricted to 20-octets due to the message transfer capability of the Bluetooth low energy. The segmentation and assembling of the messages are avoided.


4.6.2 Message header

The message header format is seen in Figure 20.

<table>
<thead>
<tr>
<th>Message Length</th>
<th>Message Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 octet</td>
<td>1 octet</td>
</tr>
</tbody>
</table>

Figure 20. Message header.

The message length is 1-octet long and it determines the whole length of the message including the length octet.

The message type is 1-octet long and it determines the type of the message. The message types are described in the following Table 1.

Table 1. Sensor communication protocol message types.

<table>
<thead>
<tr>
<th>Message type</th>
<th>Message type value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVKIT_MSG_READ_REQ</td>
<td>0x01</td>
</tr>
<tr>
<td>EVKIT_MSG_READ_RESP</td>
<td>0x02</td>
</tr>
<tr>
<td>EVKIT_MSG_WRITE_REQ</td>
<td>0x03</td>
</tr>
<tr>
<td>EVKIT_MSG_WRITE_RESP</td>
<td>0x04</td>
</tr>
<tr>
<td>EVKIT_MSG_VERSION_REQ</td>
<td>0x05</td>
</tr>
<tr>
<td>EVKIT_MSG_VERSION_RESP</td>
<td>0x06</td>
</tr>
<tr>
<td>EVKIT_MSG_GPIO_STATE_REQ</td>
<td>0x0E</td>
</tr>
<tr>
<td>EVKIT_MSG_GPIO_STATE_RESP</td>
<td>0x0F</td>
</tr>
<tr>
<td>EVKIT_MSG_ERROR_IND</td>
<td>0x11</td>
</tr>
</tbody>
</table>

4.6.3 Message variable part

**Bus parameters**

The bus parameter defines the sensor address (I2C or SPI) and the sensor register to be either ‘read’ or ‘write’. See Figure 21.

<table>
<thead>
<tr>
<th>Sensor address 1 Octet</th>
<th>Sensor register 1 Octet</th>
</tr>
</thead>
</table>

Figure 21. Sensor bus parameters.

**Message status codes**

The status code field is 1-octet long and the status codes are described in following Table 2.
Table 2. Sensor communication protocol status codes.

<table>
<thead>
<tr>
<th>Status code name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVKIT_ERROR_BASE_NUM</td>
<td>0x00</td>
</tr>
<tr>
<td>EVKIT_SUCCESS</td>
<td>EVKIT_ERROR_BASE_NUM + 0</td>
</tr>
<tr>
<td>EVKIT_BUS1_ERROR</td>
<td>EVKIT_ERROR_BASE_NUM + 1</td>
</tr>
<tr>
<td>EVKIT_BUS2_ERROR</td>
<td>EVKIT_ERROR_BASE_NUM + 2</td>
</tr>
<tr>
<td>EVKIT_GPIO_INVALID</td>
<td>EVKIT_ERROR_BASE_NUM + 3</td>
</tr>
<tr>
<td>EVKIT_MSG_LENGT_ERROR</td>
<td>EVKIT_ERROR_BASE_NUM + 4</td>
</tr>
<tr>
<td>EVKIT_BUS2_BUFFER_FULL</td>
<td>EVKIT_ERROR_BASE_NUM + 5</td>
</tr>
</tbody>
</table>

**Payload**

The payload field is used in the EVKIT WRITE REQ and EVKIT READ RESP of type’s messages to convey the sensor data. The field is a variable length and contains the actual data. Because of the Bluetooth low energy the EVKIT WRITE REQ and EVKIT READ RESP message payload is limited to a maximum of 12-octets.

**Version number**

The version number consists of a 2 octets’ major and minor part, See Figure 22.

![Figure 22. Protocol version number.](image)

**GPIO number**

The GPIO number field is 1-octed long and it identifies the GPIO line which is to be observed.

**GPIO sense**

The GPIO sense interprets the GPIO line state and the following values are defined in Table 3.

Table 3. GPIO line states.

<table>
<thead>
<tr>
<th>Define name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVKIT_MSG_GPIO_PIN_NOSENSE</td>
<td>0x00</td>
</tr>
<tr>
<td>EVKIT_MSG_GPIO_PIN_SENSE_LOW</td>
<td>0x01</td>
</tr>
<tr>
<td>EVKIT_MSG_GPIO_PIN_SENSE_HIGH</td>
<td>0x02</td>
</tr>
</tbody>
</table>
4.6.4 Individual messages

The variable parts depend on the type of the message and they are described in each individual message. The message header is included for the messages.

**Version request and response**

The version request message is used to query the version of the protocol and to change the protocol ready state. The client can identify the version of the protocol and act accordingly. See version request message in Figure 23.

<table>
<thead>
<tr>
<th>Message Length</th>
<th>Message Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octet 0</td>
<td>Octet 1</td>
</tr>
</tbody>
</table>

Figure 23. Version request message.

Version response includes the version number, which is divided into a major and minor number. See Figure 24.

<table>
<thead>
<tr>
<th>Message Length</th>
<th>Message Type</th>
<th>Status</th>
<th>Version Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octet 0</td>
<td>Octet 1</td>
<td>Octet 2</td>
<td>Octet 3,4</td>
</tr>
</tbody>
</table>

Figure 24. Version query response message.

**Write request and response**

The write message is used to write data octet(s) to the sensor register. The bus parameter identifies the sensor and defines the sensor register to be written. The payload includes the actual data to be written in the sensor specific register. See Figure 25.

<table>
<thead>
<tr>
<th>Message Length</th>
<th>Message Type</th>
<th>Bus parameters</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octet 0</td>
<td>Octet 1</td>
<td>Octet 2,3</td>
<td>Octet 4:n</td>
</tr>
</tbody>
</table>

Figure 25. Write request message.

The write response message includes bus parameters and the status code field to indicate the success or failure of the operation. See Figure 26.

<table>
<thead>
<tr>
<th>Message Length</th>
<th>Message Type</th>
<th>Bus parameters</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octet 0</td>
<td>Octet 1</td>
<td>Octet 2,3</td>
<td>Octet 4</td>
</tr>
</tbody>
</table>

Figure 26. Write response message.

**Read request and response**

The read message is used to read a specific sensor register. The Sensor is identified with the sensor I2C or SPI addresses. The bus parameter defines the sensor address and the sensor register to be read. Length indicates amount of octet’ to be read. See Figure 27.
Figure 27. Read request message.

The read response message includes the bus parameters and the status code field to indicate the success or failure of the operation. The payload contains the actual sensor register data. See Figure 28.

<table>
<thead>
<tr>
<th>Message Length</th>
<th>Message Type</th>
<th>Bus parameters</th>
<th>Read length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octet 0</td>
<td>Octet 1</td>
<td>Octet 2,3</td>
<td>Octet 4</td>
</tr>
</tbody>
</table>

Figure 28. Read response message.

**GPIO state request & response**

The sensor interrupt lines are connected to the SoC or other microcontroller based system GPIO handlers. Using the GPIO state request the specific GPIO line state can be requested. See Figure 29.

<table>
<thead>
<tr>
<th>Message Length</th>
<th>Message Type</th>
<th>Gpio nrb</th>
<th>Gpio state</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octet 0</td>
<td>Octet 1</td>
<td>Octet 2</td>
<td>Octet 3</td>
<td>Octet 4</td>
</tr>
</tbody>
</table>

Figure 29. GPIO state request message.

The GPIO state response message includes the GPIO line number and the state of the line. See Figure 30.

<table>
<thead>
<tr>
<th>Message Length</th>
<th>Message Type</th>
<th>Gpio nrb</th>
<th>Gpio state</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octet 0</td>
<td>Octet 1</td>
<td>Octet 2</td>
<td>Octet 3</td>
<td>Octet 4</td>
</tr>
</tbody>
</table>

Figure 30. GPIO state request response.

**Error indication**

If an unexpected error occurs an error indication is sent to client. See Figure 31.

<table>
<thead>
<tr>
<th>Message Length</th>
<th>Message Type</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octet 0</td>
<td>Octet 1</td>
<td>Octet 2</td>
</tr>
</tbody>
</table>

Figure 31. Error indication message.

4.7 Functional description

A **version request** is needed to set the protocol engine to a ready state to serve its clients. If the bus2 instance is changed during the execute call, new version query needs to be sent to initialize the protocol to a ready state. All message buffers and transactions are cleared when a version request message occurs.

**Write/Read** request messages are processed by the protocol engine according to the requests parameters. The status code is set to EVKIT_SUCCESS if the sensor
write/read operation over the TWI connection was successful. Otherwise error status code is returned. In Figure 32, the client is performing the version request to set the protocol state to ready. Sensor specific measurement settings are written to the sensor using the EVKIT WRITE REQ message.

Figure 32. Client reading sensor data.

The EVKIT READ REQ message is used to read the sensor data from the device. The read speed depends on the communication bus of the device (e.g. USB/BLE).

The GPIO status request assumes that the pin number state which is requested is set to input mode to detect the line low or the high status. The application which is running the protocol engine is responsible for setting the pin input mode. The EVKIT WRITE REQ is used to set sensor interrupt routings and interrupt mode. The recommended mode is latched because the client is responsible for latching the interrupt by using the EVKIT READ REQ message.
The EVKIT GPIO STATE RESP message is sent to inform the pins state when the status code EVKIT_SUCCESS is returned. In Figure 33 the interrupt low or high is detected and the client can read for the example data register associated with the interrupt configuration. When the data register is read the interrupt is latched.

The **error indication** is sent when the protocol detects an invalid message or if some unspecified event occurs.

### 4.8 Demonstration

The protocol was demonstrated by using the reference platform and also the set of the command line python scripts which were implemented to the “off-the-shelf” solution like the aardvark and NRT tiger board (Nano River Technologies, 2016; Total Phase, 2016). The purpose of the scripts was to work with the micro controller base solution without any changes. Both connections were demonstrated; the USB serial and the Bluetooth low energy with Android BLE router application. USB UART and BLE bus2 adapters were built for the reference platform verifying the connections to and from the device and enabling the protocol messages delivery. The Android BLE router implemented to pass the BLE message to the PC side via socket connection. The Android router was implemented and used the ADB socket connection. The Python socket connection class was responsible for setting the port forwarding using the ADB command for data routing from the device to the socket and vice versa.
A simple sensor communication protocol handler was implemented for the command line evaluation kit to provide common message handling for the test scripts. Also the serial port and socket connection channels were built to send and receive protocol messages. See Figure 34.

Following cases were demonstrated:

- Data writing and reading (KX122, KXG03 and KMX62 sensors)
- Double tap (KX122)
- Tilt position (KX122)
- Wake-up functionality (KX122)

Figure 35 presents Kionix IoT board USB serial port setup. The device is connected to the PC via USB cable and the evaluation kit clients are configured to use the USB serial port connection.

Windows automatically installed the FTDI driver USB serial communication. The port number can be seen from the PC Device Manager which then can be configured to the scripts. The baud rate is needed to set-up a right bus connection speed.
The Bluetooth low energy connection setup is illustrated in Figure 36.

Figure 36. Bluetooth low energy set-up.

The Android router was built to establish a BLE connection between the Kionix IoT boards and the PC. The router was scanning the advertisement reports which were published by the Kionix IoT board. The Android router was responsible for making a connection when the device was found. The Bus2 BLE adapter was implemented to use the nRF51 platform Bluetooth low energy GATT protocol. The green led colour was added to indicate the Bluetooth connection.

The output of data read python script for KXG03 gyroscope and accelerometer data can be seen in following Figure 37.

Figure 37. KXG03 data output.
The 25Hz ODR was selected for the gyroscope and accelerometer sampling. A 2G range setting was used for the accelerometer data and a 1024 deg/sec angular velocity setting for gyroscope. The wake mode and 16 bit resolution mode were selected. Gyroscope output data is usually converted to radians/second. The KXG03 sensor raw data output range and format can be found in KXG03 specification. (Kionix, KXG03, 2016.)

Output of data read python script for KXM62 magnetometer and accelerometer can be seen in following Figure 38.

![Figure 38. KMX62 data output.](image)

The 25Hz ODR was selected for the accelerometer and magnetometer sampling. A 2G range setting was used for the accelerometer data and the sensor default setting for the magnetometer data. A high resolution mode was selected to produce 16 bit data from the sensor. Usually the accelerometer raw data is converted to m/s\(^2\) and the magnetometer output is converted to micro tesla (\(\mu\)T). The KMX62 sensor raw data output range and format can be found in KMX62 specification.

The output of the double tap Python script (KX122 accelerometer) can be seen in the following Figure 39. The KX122 built in double tap algorithm setting was written to the KX122 sensor register via the sensor communication protocol.
Figure 39. Double tap detection.

The user double taps the device in/from different positions and the results indicates; which axis causes the double tap. This result can be seen from the script. Using this kind of a solution the algorithm can be fine-tuned before it is really implemented into specific embedded device. There is set of parameters in the sensor ASIC which can affect the double detection such as sensitive timers’ and g-range threshold.

The output of tilt position Python script (KX122) can be seen in following Figure 40. The KX122 built in tilt position algorithm settings were written to the KX122 sensor register via the sensor communication protocol.

Figure 40. Tilt position.
The user turns the device from different positions and sees when the axis is pointing towards the ground. There is a set of parameters in the sensor ASIC which can affect the tilt position detection, timers’, angels, and the hysteresis threshold values between the rotation states.

For example the wake-up functionality output of Python script (KX122) can be seen in the following Figure 41. The KX122 built in wake-up detection algorithm setting was written to the KX122 sensor register via the protocol.

![Figure 41. Wake-up detection.](image)

The user starts shaking the device. The algorithm detects the axis and direction of the motion. There is a set of parameters in the sensor ASIC which can affect the wake-up detection, timers’, g-range threshold.

The sensor communication protocol was also demonstrated using the evaluation kit GUI. See following Figure 42. KXG03 sensor was selected for the demonstration
Figure 42 shows that the IoT board is connected via a USB to the GUI application. By using the ‘read’ and ‘write’ buttons the user can view the current setting of a specific register or set the new setting for the sensor. When the user executes a command action the GUI sends the protocol message through the USB or BLE to the IoT board device. After the device responds the values are updated to GUI.

4.9 Evaluation

The protocol prototype was constantly evaluated during the development phase and also tested with the sensor evaluation kit GUI and command line applications. There were already implemented sensor driver scripts for the KX122, KXG03 and KMX62 sensor types. The scripts were implemented using the python language and were aimed at the “off-the-shelf” type of a solution. The verification was done by executing the same script and verifying that the result is the same in a microcontroller based system. In this prototype phase the main focuses were concentrating the read and write operations with values checks rather than the speed of the operations. It was also observed and verified that the data received from the sensor through the protocol was in the right range of what was configured to the sensor. The KX122 sensor has built in gesture detection algorithms and some of those features were tested such as double tap and orientation.

The evaluation was carried out using the following methods during the prototype development phase, see Table 4.
Table 4, Evaluation methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements gathering</td>
<td>Requirements were gathered whole prototype design and implementation phase and are introduced in chapter 4.</td>
</tr>
<tr>
<td>Weekly meetings</td>
<td>Current status was checked and new ideas for design, implementation and requirements were gathered in weekly meetings.</td>
</tr>
<tr>
<td>Design meetings with other evaluation kit developers</td>
<td>Protocol was developed collaboration with other developers who participated in implementing the evaluation kit concept. Bus interfaces evaluated from simplicity and ease of use point of view.</td>
</tr>
<tr>
<td>GUI Evaluation Kit software, CLI evaluation Kit software</td>
<td>Evaluation kit client software’s was used to read and write sensor register over USB and BLE connections. It was also used for testing the protocol.</td>
</tr>
</tbody>
</table>

One of the engineers was porting the protocol to the Arduino environment without any major changes or problems. The Arduino development environment included the BM1383 barometer and KX122 accelerometer sensor. The Arduino 1.6.8 open-source software version was used (Arduino, 2016). The KX122 and BM1383 registers were read/write successfully and the data was in the right format and range.

An important part of the evaluation was a peer review focusing on the usefulness and portability of the developed protocol. The selected reviewers were the best experts in the area and they utilized the results and also further developed the results. The comments of the reviews are summarized below:

**Reviewer 1**

"Protocol is very important and gives good possibilities to take into use a new embedded device platform for sensor evaluation.”

**Reviewer 2**

"In the porting phase the protocol engine and connectivity bus interfaces were not so well understandable. After some refactoring during the porting phase the engine code and interfaces became clearer and easily understandable. No major issues for porting the protocol to the Arduino environment.”

**Reviewer 3**

“The protocol is useful and easy to use for testing sensors. The sensor can be tested in the embedded device without writing the C code. It is easy to port to different embedded device platforms because of its simplicity. The problems with this can be for example how the customer takes the protocol into use in a new embedded device platform. How the code which is written to the sensor evaluation kit client can be ported to the end device? ”
5. Discussion and conclusions

The purpose of this study was to produce a protocol for accessing sensor data over wired and wireless connections in embedded device development platforms which would be utilized when developing and constructing new technological solutions for testing a sensor and developing the driver and algorithms software. Sensor communication protocol was proven useful and was utilized in the prototype phase for building sensor demos, data collection and algorithm development. The concept of porting the protocol was verified and proven using another embedded device development platform showing the minimized software effort for introducing new sensor development and testing environment. In this study the sensor communication protocol was developed for a special purpose to access sensor registers in the AISCs. Because the many sensors can be connected via the same I2C, the protocol provided multi-read support for register reading. So far there has not been such a protocol which could be adopted in the current development work by providing the features needed in the sensor evaluation kit concept. In this study the connectivity bus interfaces were created for having an easy access to use protocol engine through the adapter interfaces and this was also verified and stated in the porting phase of the protocol. This study provided a huge amount of information for developing the firmware software to the nRF51822 based wearable device solution. Knowledge of the development, flashing, debugging and tracing tools were acquired and will be utilized in the future development projects. NRF51822 base application scheduling, interrupt handling, UART and Bluetooth was learned to accomplish the goal of the study.

There are many data centric protocols developed for the wireless sensor networks and Internet of Things related systems (Schneider, 2013; Zia, 2015). The protocols are mainly using the request-response or publish -subscribe types of messaging paradigms and they are running in the application layer; mostly over TCP/IP and UDP data transport protocols (Postscapes, 2014; Schneider, 2013; Zia, 2015). Many of WSN and IoT protocols were complex and hard to understand because they were developed for a special purpose. Some of the protocols were designed to the energy save, low bandwidth and high link failures sense. A part of the protocols was offered an aspect of data collecting method in wireless sensor networks (sensor nodes, clusters). The most suitable protocols which could be addressed to the sensor evaluation kit would be the MQTT-SN and CoAP because of their simple specification. If the sensor evaluation kit would need several devices connected at the same time it could consider the MQTT-SN protocol. The protocol is easier to understand and it also takes into account the restriction to the message length. The restricted length is an important issue in for example Bluetooth low energy communication. The sensor communication protocol development work continues and some advantages can come from exploiting the other protocols.

In the future the sensor communication protocol could be a synthesis of the RR or PS types of protocol messaging patterns because of the new requirements for example data stream of multiple sensors with the high ODR rates. In this study the sensor evaluation kit concept was to introduce and provide an environment description where the sensor communication protocol operated.
References


