Implementation of a Prototype for Body-coupled Communication using Embedded Electronics

Implementation of a distributed system of sensors and actuators using BodyCom development board

Andrey Maleev
Bachelor of Science Thesis in Electrical Engineering
Implementation of a Prototype for Body-coupled Communication using Embedded Electronics
Andrey Maleev
LiTH-ISY-EX-ET—18/0473--SE

Supervisor:
Yonatan Kifle
Ph.D. student, ISY, Linköping University

Examiner:
J Jacob Wikner
Associate Professor, ISY, Linköping University

Division of Integrated Circuits and Systems
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden
Abstract

A wireless body network with sensors and actuators is a topical subject in current situation, because the healthcare services cannot meet peoples requirements for personal health-care. Such a network can be used to monitor the health status of e.g. elderly people and provide a drug delivery without external human interaction. In this project we will implement a prototype of a distributed system of sensors and actuator using the human body as a transmission line for communication purposes (Capacitive Body-coupled Communication), as a solution for the problem. Similar systems have been implemented earlier, using radio-based wireless communication which consumes more power and have critical security issues, compared to capacitive body-coupled communication. This document describes how the system is implemented with focus on robust gathering of sensor data from several sensors from a single node using capacitive body-coupled communication and an actuator control with user interaction.
Acknowledgments

I would like to thank the Prof. J Jacob Wikner for giving an instructive project for the thesis with day and night access for guidelines and support.

I would also like to thank the Ph.D. student Yonatan Kifle for valuable discussions and support through the project.

Lastly, I would like to thank Patrik Arven from Acero RI.SE for providing a modified version of BodyCom mobile unit with an Inter-Integrated Circuit connection to Arduino UNO which saved me a lot of time during the project.
# Table of Contents

1. **System specification** ......................................................................................... 1
2. **Introduction** ....................................................................................................... 2
   2.1 Problem statements ......................................................................................... 4
   2.2 Limitations ..................................................................................................... 5
3. **Theory** ............................................................................................................... 6
   3.1 Body-coupled communication ....................................................................... 6
      3.1.1 Capacitive-coupling ................................................................................. 7
      3.1.2 Galvanic-coupling ................................................................................ 7
   3.2 BodyCom development board ...................................................................... 8
      3.2.1 BodyCom coupling/touch – pad .............................................................. 9
      3.2.2 BodyCom bidirectional communication process .................................. 10
      3.2.3 BodyCom data packet structure ............................................................ 11
   3.3 Multiple Access Protocol ............................................................................. 12
      3.3.1 Time Derived Multiple Access ............................................................... 13
      3.3.2 Taking Turn Protocol ............................................................................ 14
   3.4 Other communication interfaces and technologies ..................................... 15
      3.4.1 Inter-Integrated Circuit (I\textsubscript{2}C) ................................................ 15
      3.4.2 Universal Synchronous/Asynchronous Receiver/Transmitter (USART) ... 16
      3.4.3 Bluetooth .............................................................................................. 16
4. **Method** ........................................................................................................... 17
   4.1 Additional hardware ....................................................................................... 17
      4.1.1 Arduino UNO ......................................................................................... 17
      4.1.2 Bluetooth module .................................................................................. 18
   4.2 Investigation of the BodyCom development board ................................... 19
      4.2.1 BodyCom data packet structure ............................................................ 19
      4.2.2 Send data packet using BodyCom ......................................................... 20
      4.2.3 Receive data packet using BodyCom .................................................... 20
      4.2.4 BodyCom base unit .............................................................................. 21
      4.2.5 BodyCom mobile unit .......................................................................... 23
   4.3 System overview ............................................................................................. 24
      4.3.1 System behavior .................................................................................... 25
   4.4 Proposed master node implementation ..................................................... 26
      4.4.1 Bluetooth connection .......................................................................... 26
      4.4.2 Proposed workflow of master node ....................................................... 26
      4.4.3 Request sensor data using Turn-taking polling protocol ..................... 28
   4.5 Sensor node implementation ......................................................................... 30
      4.5.1 Sensor node connections ..................................................................... 30
      4.5.2 Proposed workflow of sensor node ......................................................... 31
      4.5.3 Proposed workflow of Arduino UNO ....................................................... 32
   4.6 Actuator node implementation ..................................................................... 33
   4.7 Interpretation of requests .............................................................................. 34
      4.7.1 Interpretation of requests between the master node and sensor node .... 34
      4.7.2 Interpretation of requests between the master node and actuator node ... 36
      4.7.3 Interpretation of requests between BodyCom mobile unit and Arduino UNO . 37
   4.8 Sensor data conversations ............................................................................. 38
List of Figures

Figure 1 - Capacitive body coupling model with electric fields, transmitter, receiver, signal electrodes, ground electrodes and the ground................................................................. 6
Figure 2- BodyCom mobile unit (MU)...........................................................................................................8
Figure 3 - BodyCom centralized controller/ base unit (BU) ........................................................................... 8
Figure 5 - BodyCom base unit with highlighted capacitive/touch-pad ......................................................... 9
Figure 4 – Capacitive/touch-pad with illustrated partitions of Coupling PAD and Touch PAD ........ 9
Figure 6 – Overview of the signal path between the base unit and the mobile unit in BodyCom system, with illustrated modules of each BodyCom unit ........................................................................... 10
Figure 7 - BodyCom capacitive body-coupled communication data packet structure .................... 11
Figure 8 - Communication system with multiple users (n1, n2 and n3) that share a common communication channel (gateway) for bidirectional data transmission................................. 12
Figure 9 - (A): Multiplexer approach for TDMA with nx users, (B): Simple time diagram for TDMA for each nx user ........................................................................................................... 13
Figure 10 - (A): Polling protocol: illustration of nodes (n1, n2 and n3) communicating with the master node, (B): Token passing protocol: illustration of nodes (n1, n2 and n3) communicating with the sink ............................................................................. 14
Figure 11 - (A): Simple illustration of I2C-bus connection (M = master device, Sx = slave devices), (B): Illustration of the I2C data frame ........................................................................................................... 15
Figure 12 - (A): USART pin connection synchronous/asynchronous modes, (B): USART 8-bit waveform of the data sequence ........................................................................................................... 16
Figure 13- Arduino UNO board, view from above ...................................................................................... 17
Figure 14 - SparkFun Bluetooth module, both sides of the module .............................................................. 18
Figure 15- BodyCom base unit extension header with pin enumeration ................................................. 21
Figure 16 - Modified copy of BodyCom mobile unit ................................................................................. 23
Figure 17 - Block diagram of distributed system of sensors and actuators with highlighted nodes. Dashed lines represent the capacitive body-coupled communication, straight lines represent wired communication and waves represent radio-based communication ...................... 24
Figure 18 - (BU): BodyCom base unit and (BT-MODULE): Bluetooth module, connections. ......... 26
Figure 19 - BodyCom base unit (master node) workflow with parts: (A): Sequential request of sensor data, (B): Receive sensor data and request for action ................................................. 27
Figure 20 - Flowchart of Turn-taking polling protocol, approach 1......................................................... 28
Figure 21 - Flowchart of Turn-taking polling protocol, approach 2 ......................................................... 29
Figure 22 – Schematic of the sensor node. (S1): Photoresistor, (S2): Potentiometer. ......................... 30
Figure 23 – Sensor node workflow illustrated in a flowchart ..................................................................... 31
Figure 24 - Flowchart of proposed sensor reading method for Arduino UNO on sensor node... 32
Figure 25- Schematic of the actuator node. ................................................................................................. 33
Figure 26 - DataBuffer of type MDLL_PacketData_t, with enumeration of elements .................... 34
Figure 27- Custom frame design illustration based on the DataBuffer. The frame is used for communication between the master node and sensor node ......................................................... 35
Figure 28 - Custom frame design illustration based on the DataBuffer. The frame is used for communication between the master node and actuator node ......................................................... 36
Figure 29 - Custom frame design illustration based on the I2C data frame. The frame is only valid for Direction 2......................................................................................................................... 37
Figure 30 - (A): Illustration of 8-bit array to unit32_t conversion, (B): Illustration of uint32_t to 8-bit array conversation ......................................................................................................... 38
Figure 31 - The path of measured time, (1): BU of master node send a request for a sensor value, to MU of sensor node, (2): MU of sensor node send a request for a sensor value to Arduino UNO of sensor node, (3): Arduino UNO deliver the requested sensor value to MU of sensor node, (4): MU of sensor node deliver the requested sensor value to BU of master node, (5): BU of master node deliver the indication of a sensor value.

Figure 32 - Picture of the entire system with highlighted coupling-pads of BU (Pad 1) and MU (Pad 2).

Figure 33 - Flowchart of master nodes workflow, (A): Display sensor values, (B): Actuator node request, (C): Sensor node request and (D): Receiving part.

Figure 34 – Principle of the USART friendly conversation of a 32-bit integer into a char array.

Figure 35 - Custom frame design illustration based on the DataBuffer. The frame is used for communication in master node to actuator node direction.

Figure 36 - Custom frame design illustration based on the DataBuffer. The frame is used for communication in master node to sensor node direction.

Figure 37 - Custom frame design illustration based on the DataBuffer. The frame is used for communication in sensor node to master node direction.

Figure 38 - Flowchart of sensor node workflow, (A): The BodyCom mobile unit, (B): The Arduino UNO.

Figure 39 - Custom frame design illustration based on J2C data frame. The frame is used for communication in Arduino UNO to BodyCom mobile unit, direction.

Figure 40 - Flowchart of actuator node workflow.

Figure 41 - The resulting prototype of the distributed system of sensors and actuators using capacitive body-coupled communication.

Figure 42 - Screenshot of the terminal in a laptop with sensor values.

Figure 43 - Screenshot of the terminal in a laptop with user interface for actuator control.

Figure 44 - Future implementation of (A): OEIP-pump and (B): Strain sensor.
List of Tables

Table 1 - Description of the hardware and data sequence - parts of the capacitive body-coupled communication data packet ................................................................. 11
Table 2 - MDLL_PacketData_t structure .................................................................................. 19
Table 3 - Expansion header pin numeration, definition and description .......................... 21
Table 4 - Built data packet of type MDLL_PacketData_t ............................................................. 44
Table 5 - Association of system nodes with MU_ID/Sensor ID. .................................................. 46
Table 6 - Actuator ID value for corresponding pin settings and LED colors, that act as actuators ............................................................................................................ 51
Table 7 - Measurement result on implemented Taking-turn protocol performance ............... 52
## Notation

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
<th>Explanation</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>BU</td>
<td>Base unit</td>
<td>Base unit of the BodyCom development kit.</td>
<td>Described in Section 3.2.</td>
</tr>
<tr>
<td>MU</td>
<td>Mobile unit</td>
<td>Mobile unit of the BodyCom development kit.</td>
<td>Described in Section 3.2.</td>
</tr>
<tr>
<td>I²C</td>
<td>Inter-Integrated Circuit</td>
<td>A serial communication bus interface.</td>
<td>Described in Section 3.4.1.</td>
</tr>
<tr>
<td>SDA</td>
<td>Serial Data Line</td>
<td>Line for data transmission in I2C-communication.</td>
<td>Described in Section 3.4.1.</td>
</tr>
<tr>
<td>SCL</td>
<td>Serial Clock Line</td>
<td>Line for a common clock between two devices that uses I2C-communication.</td>
<td>Described in Section 3.4.1.</td>
</tr>
<tr>
<td>ACK/NACK</td>
<td>Acknowledgement and not acknowledgement</td>
<td>A bit in I2C-frame used to control the I2C-communication.</td>
<td>Described in Section 3.4.1.</td>
</tr>
<tr>
<td>RX</td>
<td>Receiver</td>
<td>Receiving pin for serial communication.</td>
<td>Receiver pin in USART-communication.</td>
</tr>
<tr>
<td>TX</td>
<td>Transmitter</td>
<td>Transmitting pin for serial communication.</td>
<td>Transmitting pin in USART-communication.</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
<td>A standard for a fast serial bus in computers.</td>
<td>As in a smartphone charger adapter connection.</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/output</td>
<td>Defining directions of a pin.</td>
<td>As in pins of a microcontroller.</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse-width modulation</td>
<td>A modulation technique of a signal.</td>
<td>As in steering PWM signal for an electrical motor.</td>
</tr>
<tr>
<td>bps</td>
<td>Bits per second</td>
<td>A data-transfer rate for serial communication, also known as baud.</td>
<td>As in USART data-transfer rate.</td>
</tr>
<tr>
<td>RTS-O</td>
<td>Request to send-output</td>
<td>Indication of a state in serial communication.</td>
<td>Described in Section 4.1.2.</td>
</tr>
<tr>
<td>CTS-I</td>
<td>Clear to send-input</td>
<td>Indicate the state in serial communication.</td>
<td>Described in Section 4.1.2.</td>
</tr>
<tr>
<td>VCC</td>
<td>Voltage common collector</td>
<td>An IC-power supply pin.</td>
<td>Power supply pin for a microcontroller.</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
<td>Details</td>
<td>Related Section</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>---------</td>
<td>-----------------</td>
</tr>
<tr>
<td>GND</td>
<td>Ground</td>
<td>Reference point in an electric circuit.</td>
<td>As in a minus-connection for a power supply.</td>
</tr>
<tr>
<td>MCU</td>
<td>Microcontroller unit</td>
<td>Programmable computer for embedded systems.</td>
<td>As in Arduino UNO.</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
<td>An interface bus for communication between two microcontrollers.</td>
<td>As in communication between two Arduinos using SPI.</td>
</tr>
<tr>
<td>USART</td>
<td>Universal synchronous and asynchronous receiver-transmitter</td>
<td>A serial interface for communication between two microcontrollers or other devices.</td>
<td>Described in Section 3.4.2.</td>
</tr>
<tr>
<td>sensor node</td>
<td>Sensor node</td>
<td>A node in the system that is implemented in this project that takes care of sensor readings and requests.</td>
<td>Described in Section 4.5.</td>
</tr>
<tr>
<td>master node</td>
<td>Master node</td>
<td>A node in the system that is implemented in this project that is controlling entire system.</td>
<td>Described in Section 4.4.</td>
</tr>
<tr>
<td>actuator node</td>
<td>Actuator node</td>
<td>A node in the system that is implemented in this project that takes care of action requests and performs the actions.</td>
<td>Described in Section 4.6</td>
</tr>
<tr>
<td>PC</td>
<td>Personal computer</td>
<td>A computer with a user interface.</td>
<td>A computer with Windows as an operative system.</td>
</tr>
<tr>
<td>BT</td>
<td>Bluetooth</td>
<td>A wireless radio-based communication standard for short distances.</td>
<td>Described in Section 3.4.3.</td>
</tr>
<tr>
<td>OK/NOK</td>
<td>Okay/not okay</td>
<td>Give permission and not giving permission</td>
<td>Described in Section 4.4.2.</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic redundancy check</td>
<td>Code that is used for error detection in raw data in digital networks.</td>
<td>As in a check for corrupted data in mobile networks.</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated development environment</td>
<td>A tool used by software developers to test and write code.</td>
<td>Described in Section 4.10.</td>
</tr>
<tr>
<td>OS</td>
<td>Operative system</td>
<td>A software that takes care of a computer hardware and software resources.</td>
<td>As in Windows XP.</td>
</tr>
<tr>
<td>GTK</td>
<td>Gimp Toolkit</td>
<td>A software programmer uses to make a graphical interface for their program.</td>
<td>As in Qt for C++</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>MU_ID</td>
<td>Mobile unit identifier</td>
<td>A number that identifies a specific BodyCom mobile unit.</td>
<td>Described in Section 5.1.4.</td>
</tr>
<tr>
<td>Sensor ID</td>
<td>Sensor identifier</td>
<td>A number that identifies a specific sensor.</td>
<td>Described in Section 4.7.1.</td>
</tr>
<tr>
<td>S1</td>
<td>Sensor 1</td>
<td>A photoresistor that is used as a sensor in the implemented system in this project.</td>
<td>Described in Section 4.5.1.</td>
</tr>
<tr>
<td>S2</td>
<td>Sensor 2</td>
<td>A potentiometer that is used as a sensor in the implemented system in this project.</td>
<td>Described in Section 4.5.1.</td>
</tr>
<tr>
<td>Actuator ID</td>
<td>Actuator identifier</td>
<td>A number that identifies a specific actuator.</td>
<td>Described in Section 5.3.</td>
</tr>
<tr>
<td>Acknowledgement ID</td>
<td>Acknowledgement identifier</td>
<td>A number that identifies that the actuator node has performed the requested task.</td>
<td>Described in Section 5.3.</td>
</tr>
<tr>
<td>OEIP</td>
<td>Organic Electronic Ion Pump</td>
<td>A drug delivery pump.</td>
<td>See Abstract.</td>
</tr>
<tr>
<td>DispSensor</td>
<td>Display sensors</td>
<td>A variable in the implemented system that is set to one when it is time to display sensor values.</td>
<td>Described in Section 5.1.1.</td>
</tr>
<tr>
<td>A/D-converter</td>
<td>Analog-to-Digital Converter</td>
<td>System that converts an electric analog signal to a digital.</td>
<td>Often implemented in digital circuits.</td>
</tr>
<tr>
<td>sensor_type</td>
<td>Sensor type</td>
<td>A variable in the implemented system that handles the turn-taking of sensor requests.</td>
<td>Described in Section 5.1.5.</td>
</tr>
<tr>
<td>Act</td>
<td>Action request</td>
<td>A variable in the implemented system that is set to one when it is time to send an action request.</td>
<td>Described in Section 5.1.3.</td>
</tr>
<tr>
<td>BAN</td>
<td>Body Area Network</td>
<td>A network within body area.</td>
<td>Described in Section 2.</td>
</tr>
<tr>
<td>MBAN</td>
<td>Medical Body Area Network</td>
<td>A medical network within body area.</td>
<td>Described in Section 2.</td>
</tr>
<tr>
<td>BCC</td>
<td>Body-coupled Communication</td>
<td>Wireless communication, that uses human body as a transmission line</td>
<td>Described in Section 2.</td>
</tr>
<tr>
<td>kHz, MHz, GHz</td>
<td>(kilo, mega, giga) Hertz</td>
<td>Frequency unit.</td>
<td>As in radio.</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
<td>Definition</td>
<td>Example</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
<td>A frequency for radio communication.</td>
<td>As in a car radio.</td>
</tr>
<tr>
<td>LED</td>
<td>Light Emitting Diode</td>
<td>Semiconducting light source.</td>
<td>As in diode-matrix.</td>
</tr>
<tr>
<td>MSB</td>
<td>Most Significant Bit</td>
<td>Highest order bit position in a binary number.</td>
<td>A bit in binary number with highest value.</td>
</tr>
<tr>
<td>LSB</td>
<td>Least Significant Bit</td>
<td>Lowest order bit position in a binary number.</td>
<td>A bit in binary number with lowest value.</td>
</tr>
<tr>
<td>LCD</td>
<td>Liquid-crystal Display</td>
<td>A type of display.</td>
<td>As in displays for smartphones.</td>
</tr>
</tbody>
</table>
1. System specification

Desired specification for a distributed system of sensors and actuators:

1. The sensor system consists of a
   a. BodyCom main board that will act as a gateway with a Bluetooth connector and
   b. several BodyCom mobile units. Each BodyCom mobile unit, in turn, interacts with another slave board, namely an Arduino.

2. The system shall connect to a Personal Computer (PC) or a smartphone (terminal) through a Bluetooth device, this implies that
   a. we will be able to visualize sensor data and
   b. send actions to the actuators.

3. The system should be able to interconnect at least 2 sensors and 3 actuators.
   a. This implies that we should have a method to interact with many sensors in “parallel”.
      i. The sensors will be of three various kinds:
         1. XYZ (gyroscope), taking some absolute coordinates.
         2. Temperature sensor, taking a value.
         3. XYZ (strain sensor), taking an action.
      ii. The actuators will be:
         1. The Organic Electronic Ion Pump (OEIP) driven by the slave board namely an Arduino, mentioned above.
         2. A diode that is turned on for a certain period. Where the time/blink rate, or similar can be controlled by the system.

There is no performance specification for the system, such as communication speed and system responsiveness. The system shall be robust and modular constructed.
2. Introduction

In the current situation the traditional health-care services in Sweden, experiences pressure because of lack of staff with expertise in healthcare and limited economic resources [2]. This leads to a backlash against people daily health-care, even when it is important to support the personal health. It also includes home care which concerns the elderly people, handicapped people and people with chronic diseases etc. The duties as a healthcare staff is to monitor patients well-being, make sure they take their medicine at the right time and much more. The mentioned tasks and many other are performed in an inefficient way and there is a need for a better solution for current health-care systems.

A Body Area Network (BAN) also referred to Medical Body Area Network (MBAN) with integrated sensors is a candidate for resolve problems with current health-care systems. MBAN consist of wireless wearable/implanted devices that are mounted within the human body and wirelessly connected to a base unit, typically used for health monitoring and remote control of medical devices [3]. The health monitoring is achieved by embedded sensors in the wearable/implanted devices that collects vital data from the human body such as temperature, blood pressure, glucose levels and so on. Embedded actuators in wearable/implanted devices such as insulin pump, makes it possible to e.g. remotely control glucose levels in a human body. It is life critical for the MBAN to meet requirements for safety which in this case implies low power consumption and avoiding communication interference. Current MBANs uses radio-based wireless communication technologies to communicate between wearable/implanted devices and the base unit, this solution makes the requirements for safety not as good as desired [4]. A better substitute for radio-based wireless communication technologies in context of MBANs and safety requirement is body-coupled communication. Body-coupled communication (BCC) uses the human body as a transmission line for data transferring, this solution makes it harder to interference the communication between wearable/implanted devices and the base unit, to no one radio-based wireless communication is occurred. The BCC have also an advantage over radio-based wireless communication technologies in terms of energy consumption, which benefits the MBAN even more [5].

In this document we will aim to implement a prototype of a distributed system of sensors and actuators using capacitive body-coupled communication as a replacement for radio-based wireless communication technologies, with focus on robustness. We will use the BodyCom development kit, provided by Microchip [1], that consists of a base board and several mobile units that uses the capacitive body-coupled communication to communicate with each other. The BodyCom development kit makes it easy to develop prototypes or applications with capacitive body-coupled communication. The BodyCom boards have integrated reprogrammable processors and gives desirable freedom for modifications and custom applications.
The distributed system of sensors and actuators will continuously collect data from several sensors connected to a BodyCom mobile unit using methods from existing “parallel” communication methods and similar techniques to establish a failsafe (robust) data collection, using a single communication channel. The actuators will be connected to a separate mobile unit and take unsynchronized actions using BCC. In Section 1 a detailed specification of prototype of the distributed system of sensors and actuators using (BCC) is described.

The proposed prototype is desired by BioCom Lab which is a Swedish cooperation between Acero Swedish Information and Communication Technologies AB and Linköping University for research and development for e-health solutions. BioCom Lab vision is to create an intra-body network that will help to gather vital information from a human body that can be big-data analyzed to perform a drug-delivery into human body using implanted actuators supported by mobile networks and the clouds. Their goal is to improve life quality and reduce healthcare costs using the intra-body network, when it has more opportunities than those mentioned [6].
2.1 Problem statements

To implement the prototype of a distributed system of sensors and actuators we need to investigate the BodyCom development kit from Microchip and find a suitable method for “parallel” communication. To accomplish that we need to answer several questions.

1. What restrictions does the BodyCom development kit have? More particular:
   a. Is there enough space for modifications?
      i. Such as, free and accessible pins on the Microcontroller Unit (MCU)
      ii. and the BodyCom development board?
      iii. Available peripherals on the MCU, such as serial communication?
   b. How big is the data packages that can be sent using (BCC)?

2. What factors are important in context of robustness that is applicable to our distributed system of sensors and actuators using BCC?

3. To implement the prototype of a robust distributed system of sensors and actuators, which is the most suitable method for “parallel” communication? Given the answers from questions above.

As a summary of the problem statements, we need to investigate the use of BodyCom development kit from Microchip [1] to create a distributed system of sensors and actuators, and find a suitable method for “parallel” communication.
2.2 Limitations

The system does not have any power consumption restrictions, but it would be beneficial to operate on a 3.3-V coin-cell battery.

The system does not have any size limits, but the form factor of BodyCom mobile units is in order of bracelets. This is not valid for the BodyCom main board, see Fig. 3 in Section 3.2.

The distributed system of sensors and actuators will consist of two sensors and three actuators connected to each BodyCom mobile unit, see Fig. 2 in Section 3.2.
3. Theory

This chapter covers all the theoretical knowledge about body-coupled communication, BodyCom development board and communication/access protocols needed to make a prototype of a robust distributed system of sensors and actuators with multiple access using BCC.
This chapter provides a basic knowledge required for the project and to get a better picture of what the project is about in details. By using relevant information from earlier experience, we will be able to find a suitable method of implementation the distributed system of sensors and actuators.

3.1 Body-coupled communication

Body-coupled communication is a way to communicate wirelessly, using the human body as a transmission medium for the signal [5]. The operating frequency range is somewhere between a couple of hundreds of kHz to tens MHz. Body-coupled communication was first introduced by Zimmerman in 1996, where a prototype using capacitive-coupled communication was developed. Zimmerman demonstrated that it is possible to bidirectionally send digital information by shaking hands [7].

Figure 1 - Capacitive body coupling model with electric fields, transmitter, receiver, signal electrodes, ground electrodes and the ground.
There are two main types of body-coupled communication, galvanic- and capacitive-coupled communication [5]. Capacitive-coupled communication is the most advantageous type because of its low power consumption, high gain and higher frequency operation range compared to galvanic-coupled communication. Both types of body-coupled communication use a differential pair of electrodes, one pair for the transmitter and one pair for the receiver, see Fig. 1 where differential pair of electrodes named signal electrode and ground electrode. With capacitive-coupling, the electrodes do not need to touch the skin to be able to communicate through the body because of its high gain and therefore the transmitting/receiving device can be held in a pocket. While for the galvanic-coupling, the electrodes needs to be directly attached to the skin to be able to communicate through the body [8].

3.1.1 Capacitive-coupling

In Fig. 1, an electric signal is generated by the transmitter between the signal – and ground electrodes where those electrodes have a dissimilar capacitive coupling to the human body and forces an electric field through the body tissues. In Fig. 1 the receiver can sense a differential signal (electric field forced by transmitter) between the signal electrode and the ground electrode. The signal the receiver senses is very weak, but it is enough to detect it. Signal weakness is occurred because of losses of electric field to the ground, sent by the transmitter [9] which also occurs the propagation loss of the signal in the human body, where the longer distance between the transmitter and receiver affects the transmitted signal negative in terms of signal strength.

With this approach the human body acts as a conductor between the transmitter and receiver, the human body can be used as a communication bridge between two nodes (transmitter/receiver) [10].

3.1.2 Galvanic-coupling

Like the capacitive-coupling an electric signal is generated by the transmitter between the signal – and ground electrodes, see Fig. 1. Because of the electric signal that is generated by the transmitter, a differential current occurs between the electrodes (signal electrode and ground electrode) and induces a secondary current into the human body. The receiver senses the induced current through the human body and as a result a transmission line through the human body is created [10].

With respect to physics, the main difference between the galvanic-coupling and capacitive-couplings transmitter and receiver are capacitively coupled to the human body and as a result forms a bridge, using the human body. Galvanic-coupling uses a different approach by instead take an advantage of dielectric characteristics in the human body, where the flowing ions is carrying the information [9].
3.2 BodyCom development board

BodyCom development board is a communication system, made by Microchip [1], that uses capacitive-coupled communication to be able to use the human body as a transmission line with short-range and low-data-rate communication that communicates within the human body area at 125 kHz by default. The system consists of one centralized controller, the Base Unit (BU), Fig. 3 and one or several tags, the Mobile Unit (MU), see Fig. 2.

BodyCom communication system benefits with respect to MBANs are very useful in terms of power consumption and security. Because of BCC, BodyCom communication system does not need any wireless transceivers, which in turn saves energy and the communication is harder to interference [9].
3.2.1 BodyCom coupling/touch – pad

BodyCom capacitive/touch-pad in Fig. 5 consists of a touchpad and a coupling element, see Fig. 4.

The Touch PAD partition in Fig. 4 is used as a touch-sensor, where the user of the BodyCom communication system wants to touch the pad to start the communication. The Coupling PAD in Fig. 4 is a coupling element that acts as an electrode for capacitive-coupled communication, see Section 2.1. If desired, the Touch PAD can be turned off, so the user doesn’t need to touch the pad to start the communication, the BodyCom system have reliable performance even without the touch [8].

The design of the capacitive/touch-pad makes the finger touch position on the pad almost irrelevant. In Fig. 5 we can see the capacitive/touch-pad of the BodyCom base unit and compare it with Fig. 4 to have a better picture of how the capacitive/touch-pad in Fig. 5 is designed. With fact that the capacitive/touch partitions cover the whole area of the actual pad, it is almost impossible to have a bad capacitive-coupling and touch detection [8].
### 3.2.2 BodyCom bidirectional communication process

BodyCom communication system always initiates the communication from the BU side, to communicate with the MUs. The BU continuously scan the touchpad in Fig. 5, the scanning stops when a touch is detected, and the BU starts to search for detectable MUs using the coupling element. The coupling element forces a signal through the human body and turning it into a Radio Frequency (RF) emitter with low-frequency. The human body acts as an extension of the coupling element and permits the signal transportation if there is a connection to the MU, see Fig. 6 [8]. The MU receives the transmitted signal from the BU and decodes the transmitted data. The MU will send a response if it is programmed to do that. The respond works the same way as for the BU transmitting process and the MU is also using a 125 kHz transmitting channel, see Fig. 6 [1].

The BU acts as a fixed part of the BodyCom communication system and it consumes more power than a MU. Higher power consumption for the BU makes it possible to have a low-powered MU. The BU applies more power on the transmitting channel and operates at low-frequency of 125 kHz [8].

![Diagram of BodyCom system](image)

**Figure 6 – Overview of the signal path between the base unit and the mobile unit in BodyCom system, with illustrated modules of each BodyCom unit**
3.2.3 BodyCom data packet structure

The data packet used in BodyCom communication system Fig. 7 is designed for robust communication. The data packet consists of two parts, first part that is needed by the hardware and second part is a data sequence. In Table. 1, both parts of the data packet are described [8].

<table>
<thead>
<tr>
<th>Hardware part</th>
<th>Data sequence part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preamble time – Used to configure and activate the Automatic Gain Control (AGC)</td>
<td>Command byte – Bits that can be configured by user. Controls how the system should act.</td>
</tr>
<tr>
<td>Wake-up Filter – Bits that can be configured by user to tell the device to wake up and transmit data to the PIC Microcontroller</td>
<td>Address – 32-bit address of the mobile unit. Used to decide which (MU) to communicate with.</td>
</tr>
<tr>
<td>-</td>
<td>Length byte – The length of Data in bytes</td>
</tr>
<tr>
<td>-</td>
<td>Data [0..n-1] – The actual data sent between the (BU) and (MU)</td>
</tr>
</tbody>
</table>

The data sequence is Manchester encoded [8] because of suitability for single link data transmission, low-rate data communication, that is applicable for coupled-communication purpose [11]. The interesting feature of the Manchester code is that the Direct Current (DC) component of the Manchester encoded signal carries no data and the signal connections may be e.g. capacitive with galvanic isolation. It fits communication standards that does not carrying power for data transmission, such as capacitive body-coupled communication.
3.3 Multiple Access Protocol

Multiple Access Protocol (MAP) is mainly used in communication systems, everything from wireless radio-based networks to wired networks. MAPs are used to avoid collision when multiple users use a common communication channel [12]. In Fig. 8, nodes $n\times$ represents the users and the gateway represents the single node of common communication channel. In this document we are interested in bidirectional collision-free multiple access communication without setting ambitious standards on communication speed.

Figure 8 - Communication system with multiple users (n1, n2 and n3) that share a common communication channel (gateway) for bidirectional data transmission.
3.3.1 Time Derived Multiple Access

Time Derived Multiple Access or TDMA is a MAP that is used in single channel networks and known for being good at avoiding collisions in such networks [13]. TDMA can be compared with a multiplexer see Fig. 9A, where each selection changes in a sequential loop and each selection is active for a predefined constant amount of time $t_n$ in Fig. 9B, loaded in the timer $T$ in Fig. 9A [12]. In the time diagram in Fig. 9B, we have a small period $t_s$ that is the nonconstant time it takes to switch between the nodes $n_x$ [14]. The total time it will take $x$ nodes to be selected:

$$k = \text{total number of nodes}, k \neq 0$$

$$t_{\text{total}} = \text{total time to select all } k \text{ nodes}$$

$$t_n = \text{constant time given each node}$$

$$t_s = \text{nonconstant time it takes to switch between the nodes}$$

$$t_{\text{total}} = t_n \times \sum_{x=1}^{k} t_s$$

(1)

Figure 9 - (A): Multiplexer approach for TDMA with $n_x$ users, (B): Simple time diagram for TDMA for each $n_x$ user
3.3.2 Taking Turn Protocol

Taking Turn Protocol (TTP) is a MAP that can be used in single channel networks and it works by the principle that each node taking its turn for transmitting data. After data have been transmitted for a node, the turn changes to another node and so on. There are two main types of TPPs that we are about to describe.

The first type of TPP, called polling protocol, requires a master node that will be in charge for turns taking between the slave nodes $n_x$. In Fig. 10A, the master node sends a message to node $n_1$ and tells the node $n_1$ that it can transmit some data. After node $n_1$ have transmitted the requested data, the master node sends a message to node $n_2$ that gives permission for transmit its data. After node $n_2$ have transmitted the required data, the master node sends a message to next node and so on. The master node is always aware of when each node is done transmitting data and have responsibility to deliver the transmitted data from the slave nodes to the sink. One main drawback with polling protocol is that if the master node fails, the whole network becomes inoperative [15].

The second type of TPP is called token passing protocol. Token passing protocol does not require any master node, instead all the nodes passing a sort of token between each other in a predefined order. When a node is holding a token, it has permission to transmit data to the sink. In Fig. 10B, node $n_1$ has transmitted data to the sink and node $n_1$ always pass the token to the node $n_2$. Node $n_2$ is now allowed to transmit data to the sink. After node $n_2$ has transmitted the required data it always passes the token to the node $n_3$ and so on. All the nodes are using single channel to communicate with each other and transmits data to the sink. The main drawback with this approach is that if one of the nodes fails or crashes, whole network will fail/crash with it [16].

Figure 10 - (A): Polling protocol: illustration of nodes ($n_1$, $n_2$ and $n_3$) communicating with the master node, (B): Token passing protocol: illustration of nodes ($n_1$, $n_2$ and $n_3$) communicating with the sink
3.4 Other communication interfaces and technologies

This chapter mentions communication protocols/interfaces that can be used to meet project specification in Section 1.

3.4.1 Inter-Integrated Circuit (I²C)

Inter-Integrated Circuit (I²C) is a serial communication bus, see Fig 11A, that is mainly used in embedded systems, but it can be used in various electronic devices. I²C is known for its simplicity and cost reduction for Integrated Circuit (IC)-pins and therefore became a known communication standard [17]. It consists of two wires Serial Data Line (SDA) and Serial Clock Line (SCL) that handles the data and the synchronization clock respectively, see Fig. 11A. The devices that is connected to the I²C-bus plays different rolls in the communication bus. In Fig. 11A, M stands for master, the master device is the one who request the information from the slave devices, S1 and S2, and at the same time the master device provides the clock signal for the bus on SCL wire. The communication starts by the master device with a start bit on the SDA wire, see Fig. 11B, by that all slave devices are listening to the master device. The master device sends a slave address on the bus to indicate which slave device the master device wants to communicate with and ends it with a read/write bit (R/W) to indicate if the master device wants to write something to the slave device or read some information provided by the slave device, see Fig. 11B. When one of the slave devices gets its address and the action it need to perform, it answers with an Acknowledge bit (ACK) or Not Acknowledge (NACK). The acknowledge bit sent by the slave device tells the master device if there are coming more data or if it is the end of the data stream, therefore ACK/NACK. When the master device gets all the data from the slave, the master device sends a stop bit, to end the communication [16] [18].

Slave address is a 7-bit address with an additional R/W-bit, with total of 8-bits. Each data sequence sent on the SDA-wire is always 8-bit data, inclusive the slave address and the R/W-bit. The actual data sent/requested by the master device, see Fig. 11B, can be divided the 8-bit data chunks, to able to transmit/receive more data while communicating [16].

![Figure 11 - (A): Simple illustration of I2C-bus connection (M = master device, Sx = slave devices), (B): Illustration of the I2C data frame](image-url)
3.4.2 Universal Synchronous/Asynchronous Receiver/Transmitter (USART)

Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is a serial communication interface that is widely used in embedded systems and PC serial ports. USART works in two different modes, synchronous and asynchronous. Compared to the synchronous mode, the asynchronous mode does not need a separate transmission line for the clock, to be able to send and receive data between two units, see Fig 12A. The signal on the Transmitter (TX) and Receiver (RX) pins are high, as a logic level signal, when there is no active communication. The communication starts when the TX pin goes low, as an indicator for the RX pin, followed by an 8-bit data stream and an additional high logic level signal stop to indicate the end of the communication, see Fig. 12B [19].

![USART pin connection and 8-bit waveform](image)

Figure 12 - (A): USART pin connection synchronous/asynchronous modes, (B): USART 8-bit waveform of the data sequence

3.4.3 Bluetooth

Bluetooth is a radio-based communication technology that operates over short ranges in the 2.4GHz band. It is widely used in consumer electronics such as smartphones, notebooks, headphones and so on. It can be used to form a network or just to be able to communicate wirelessly between two devices [20]. There are five versions of the Bluetooth technologies, each version is backward compatible which makes it easy to use different versions of Bluetooth in electronic devices [21]. Apart from the mentioned versions, Bluetooth technology is divided in two main radio versions Bluetooth Low Energy (LE) and Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR). (LE) have the advantage in lower power consumption compared to the (BR/EDR), by 50%-99% and have more network topologies than (BR/EDR) such as Point-to-Point, Broadcast and Mesh. (BR/EDR) is more optimized for continuous data streaming and have more communication channels than (LE) [22].

(BR/EDR) is more compatible to external platforms such as iOS and Android, (LE) is only compatible with the newer versions of such platforms. As an example, for iOS and Android, (LE) is compatible with iOS versions 4.0 and up, and Android versions 4.3 and up while (BR/EDR) is compatible with all versions [23].
4. Method

In this chapter we will explain the proposed methods of implementation of the distributed system of sensors and actuators using BodyCom development board. This chapter also includes the schematics of each part of the system and investigation of the BodyCom development kit in terms of restrictions.

4.1 Additional hardware

This section describes the additional hardware devices that is used in this project. The description includes the technical specification of each device to give the reader a better picture of the system.

4.1.1 Arduino UNO

Arduino UNO is a development board, based on the reprogrammable microcontroller ATmega328P that operates on a frequency of 16MHz, see Fig. 13. Input voltage limits, for powering the board is 6-20V, but it can also be powered by an USB type B cable. Arduino UNO have 6 analog pins for the analog-to-digital conversation, 14 digital Input/Output (I/O)-pins where 6 of them can provide a Pulse-width modulation (PWM) signal and two of them can provide serial communication at (TX) and (RX) pins. The digital I/O-pins can interpret a logic level signal between 3.3-5V [24].

![Arduino UNO board, view from above.](image)

In this thesis project, the Arduino UNO is used to read the sensors values and provide those values to the BodyCom MU of the sensor node. The Arduino UNO is a good fit for such purposes because of its development simplicity.
4.1.2 Bluetooth module

The Bluetooth module that is used in this project is a *SparkFun Bluetooth Mate Silver* with operating voltage between 3.3-6V, see Fig. 14. This Bluetooth module have an encrypted connection and can communicate with serial communication protocols at 2400-115200 bits per second (bps) [25]. The module provides six different pins:

- *(RX-I)* – Receiver pin for serial communication (input)
- *(TX-O)* – Transmitter pin for serial communication (output)
- *(RTS-O)* – Request to send pin for serial communication (output)
- *(CTS-I)* – Clear to send pin for serial communication (input)
- *(VCC)* – Supply voltage pin
- *(GND)* – Ground pin

The Bluetooth module will be used as a communication link between the BodyCom base board and the internal Bluetooth module of personal computer or a smartphone. The Bluetooth module will be connected to the BodyCom base board and create a communication link between and the internal Bluetooth module of personal computer or a smartphone.

![SparkFun Bluetooth module, both sides of the module.](image)

Figure 14 - SparkFun Bluetooth module, both sides of the module.
4.2 Investigation of the BodyCom development board

Location of available I/O-pins, MCUs peripherals pins and functionality of the BodyCom system will be investigated in this section. Being aware of BodyCom system restrictions will help us to design the distributed system of sensors and actuators.

4.2.1 BodyCom data packet structure

Both BodyCom (BU) and (MU) have same data packet structure defined in the MDLL_DataLinkLayer.h header-file that can be found in BodyCom template codes. The data packet has its own type structure called MDLL_PacketData_t containing a Command, Address, DataLenght and DataBuffer, see Table. 2.

<table>
<thead>
<tr>
<th>MDLL_PacketData_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type:</td>
</tr>
<tr>
<td>Command</td>
</tr>
<tr>
<td>Address</td>
</tr>
<tr>
<td>DataLenght</td>
</tr>
<tr>
<td>DataBuffer</td>
</tr>
<tr>
<td>Array (Yes/No):</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
</tr>
<tr>
<td>Number of elements:</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1-16</td>
</tr>
</tbody>
</table>

Table 2 - MDLL_PacketData_t structure

The Command is an 8-bit unsigned integer variable that controls the action of the system, see Table. 2. There are four predefined commands PING_PAIRED, PAIR_DEVICE, ECHO_REQUEST and ECHO_RESPONSE. Each command has a predefined hexadecimal value that is recognized by the BodyCom system to take an action. The commands have following functions:

- **PING_PAIRED** – Pings the MUs in BU range
- **PAIR_DEVICE** – Pairs the MUs with the BU
- **ECHO_REQUEST** – BU requesting data/address of the MU
- **ECHO_RESPONSE** – MU responses with data/address to the BU

The Address is an 8-bit unsigned integer array of 4 elements that represents the address of the MU which BU want to communicate with, see Table. 1.

The DataLenght is an 8-bit unsigned integer variable that define the length of the DataBuffer array. The number of elements in DataBuffer array is the length unit that is defined in DataLenght.

The DataBuffer is an 8-bit unsigned integer array that is used to store the data to be sent between the BU and MU, or the other way around. The DataBuffer array have a maximum size of 16 elements and a minimum size of one element.
4.2.2 Send data packet using BodyCom
A default function in the BU template code that is used to send data packets between the BU and MU called *BC_SendDataCommand*, that can be found in *BC_Application.c* source-file. The *BC_SendDataCommand* takes four following inputs:

- *cmd* – Command
- *Address* – MU address
- *data* – Data buffer
- *length* - Length of the data buffer
- *timeout* – Timeout value for the packet transmission

*BC_SendDataCommand* creates a temporary packet *tempPacket* of type *MDLL_PacketData_t* and inserts functions inputs to the respective structure location in the *tempPacket*, see Section 4.2.1. By returning with function *MDLL_sendPacket*, with the *tempPacket* as an input the BU will send the packet to addressed MU.

4.2.3 Receive data packet using BodyCom
Both BU and MU are provided with a command handler function from the template code, that takes care of receiving data packets from each other. Basically, the command handler function checks if the *MDLL_receiveDataPacket* function returns a value that is larger than zero and by that indicates MU or BU have received a data packet. The received data packet is a global variable called *MDLL_PacketDataBuffer* of type *MDLL_PacketData_t*, that can be extracted by its structure for further desired use.

After receiving a data packet, MU uses similar functionality to *BC_SendDataCommand*, see Section 4.2.2, to respond back to the BU. BU default command handler outputs MU address after receiving a packet.

The *MDLL_receiveDataPacket* functions returning value acts as an indicator that the received data packet has arrived without any errors.
4.2.4 BodyCom base unit

BodyCom BU board have an extension header that is connected to BU MCU I/O-pins, see Fig. 15. The BU board are foreseen with the extension header to give the user an easy access to MCU I/O-pins for creating custom prototype applications [26]. MCU I/O-pins have contrasting functions that is useful for embedded systems. We are interested in I/O-pins for serial communication such as asynchronous USART and $I^2C$.

![Figure 15- BodyCom base unit extension header with pin enumeration](image)

As a reference to Fig. 15, see Table. 3 were the I/O-pin enumeration matches the actual I/O-pins on the BodyCom BU MCU with functionality description. Note that the I/O-pins on the MCU have more functionality than presented in Table. 3. The only interest is in the I/O-pin functionality that is intended for this project [25] [27].

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+3.3V</td>
<td>Supply voltage</td>
<td>2</td>
<td>RA5</td>
</tr>
<tr>
<td>3</td>
<td>N/C</td>
<td>-</td>
<td>4</td>
<td>RA4</td>
</tr>
<tr>
<td>5</td>
<td>N/C</td>
<td>-</td>
<td>6</td>
<td>RA3</td>
</tr>
<tr>
<td>7</td>
<td>N/C</td>
<td>-</td>
<td>8</td>
<td>RC5</td>
</tr>
<tr>
<td>9</td>
<td>RC7</td>
<td>I/O-pin</td>
<td>10</td>
<td>RC4</td>
</tr>
<tr>
<td>11</td>
<td>RC6</td>
<td>I/O-pin</td>
<td>12</td>
<td>RC3</td>
</tr>
<tr>
<td>13</td>
<td>RB7</td>
<td>I/O-pin/TX1/SCL2</td>
<td>14</td>
<td>RA0</td>
</tr>
<tr>
<td>15</td>
<td>RB6</td>
<td>I/O-pin/SCL1</td>
<td>16</td>
<td>RA1</td>
</tr>
<tr>
<td>17</td>
<td>RB5</td>
<td>I/O-pin/RX1/SDA2</td>
<td>18</td>
<td>RA2</td>
</tr>
<tr>
<td>19</td>
<td>RB4</td>
<td>I/O-pin/SDA1</td>
<td>20</td>
<td>RC0</td>
</tr>
<tr>
<td>21</td>
<td>N/C</td>
<td>-</td>
<td>22</td>
<td>RC1</td>
</tr>
<tr>
<td>23</td>
<td>N/C</td>
<td>-</td>
<td>24</td>
<td>RC2</td>
</tr>
<tr>
<td>25</td>
<td>N/C</td>
<td>-</td>
<td>26</td>
<td>+5V</td>
</tr>
<tr>
<td>27</td>
<td>N/C</td>
<td>-</td>
<td>28</td>
<td>GND</td>
</tr>
</tbody>
</table>

Table 3 - Expansion header pin numeration, definition and description
Some I/O-pins with possibility for serial communication mentioned above, presented in Table. 3, are taken by the BodyCom BU system functionality [25]. Following I/O-pins have been taken by the BodyCom BU system:

- **RB7** – spare
- **RB6** – taken for $I^2C$ – communication (clock) with the Liquid-crystal Display (LCD) unit
- **RB5** – spare
- **RB4** - taken for $I^2C$ – communication (data) with the LCD unit
- **RC5** – taken for UART-communication (TX) with an UART to USB2 converter unit
- **RC4** – connected with pin **RC3** and **RC6**
  - **RC3** – taken by the Low Frequency (LF) (125kHz) driver
  - **RC6** - taken for UART-communication RX with an UART to Universal Serial Bus (USB2) converter unit

Pins **RB7** and **RB5** is free and can be used for UART-communication, which is connected with the Bluetooth module in **Section 3.1.2.**
4.2.5 BodyCom mobile unit

BodyCom MU board does not have any extension header for custom prototype applications as BU does. The only accessible I/O-pins on the MU are those which control two Light-Emitting Diodes (LEDs) that acts as visual indicators on the physical MU board. Following I/O-pins are accessible on the MU:

- **RB6** – red LED
- **RB7** – green LED

Those I/O-pins does not provide any functionality in form of desired serial communication USART or \(I^2C\) [28].

A modified copy of BodyCom MU was provided by Patrik Arven from the company Acero RI.SE, see Fig. 16. The modification includes previously inaccessible asset to pins **RB2** (SDA2) and **RB5** (SCL2) which provides MCU inbuilt serial communication functionality, such as USART, SPI and \(I^2C\) [27].

![Figure 16 - Modified copy of BodyCom mobile unit.](image.png)
4.3 System overview

The proposed block diagram of the distributed system of sensors and actuators is presented in Fig. 17. **BU** is the BodyCom base unit that acts as a master node and mobile units (**MU1, MU2**) acts as slave nodes in the distributed system of sensors and actuators. Each slave unit play separate roles in the system. **MU1** is an intermediator that takes requests from the **BU** about a specific sensor value and requests the sensor value from the **Arduino UNO**, were the **Arduino UNO** takes care of actual sensor readings. After receiving the requested sensor value from **Arduino UNO**, **MU1** responds with the same value to the **BU**. Like the **MU1**, **MU2** is an intermediator that takes requests from the **BU**, but in this case, requests are in form of different actions. **MU2** is an actuator that performs the requested actions and responds to the **BU**, when the action is performed. **MU1** is considered as a *sensor node*, **MU2** as an *actuator node*, **BU** as a *master node* and **PC** as a *user node*. **A1-3** are the actuators and **S1-2** are the sensors.

**BU** have a Bluetooth module **BT** connected, that communicates with a **PC**. **PC** is a computer device such as smartphone or a laptop than have an inbuilt Bluetooth module. **PC** and **BT** allow us to interact with the distributed system of sensors and actuators wirelessly and gives us an opportunity to create a user-interface.

![Figure 17 - Block diagram of distributed system of sensors and actuators with highlighted nodes. Dashed lines represent the capacitive body-coupled communication, straight lines represent wired communication and waves represent radio-based communication.](image-url)
4.3.1 System behavior

The proposed behavior of the distributed system of sensors and actuators can be divided in two parts.

In the first part, the master node continuously collects sensor data from the sensor node and compare sensor values to corresponding predefined threshold values as they arrive. The predefined threshold values act as triggers for the system and the comparison result between sensor data and threshold values can be considered as satisfied or not satisfied. Respective sensor values that are received by the master node are visualized on the PC using the Bluetooth (BT), see Section 4.3.

The second part getting started when the comparison result between the sensor value with predefined threshold value are not satisfied. As an outcome of negative comparison result, the master node will be able to send one of several action requests to the actuator node. The action requests will be suggested by the system where the user is able to choose between suggested action requests. The suggested action requests will be visualized on the PC using the BT, see Section 4.3.
4.4 Proposed master node implementation

In this section we will go through the proposed master node implementation. It includes the hardware implementation and the proposed workflow of the system with associated MAP.

4.4.1 Bluetooth connection

BU is connected to a Bluetooth module (SparkFun Bluetooth Mate Silver, see Section 5.1.2) to provide USART serial communication protocol connection. The BU transmitting pin (RC5) connects with the receiving pin on the Bluetooth module (RX-I) and BU receiving pin (RC4) connects to the transmitting pin (TX-O) on the Bluetooth module, see Fig. 18. The voltage supply of 3.3V for the Bluetooth module is provided by the extension header pins of the BU together with the ground, see Section 4.2.4 for extension header layout and Fig.18 for voltage supply connections.

4.4.2 Proposed workflow of master node

The proposed master node workflow is illustrated in the flowchart in Fig. 19. The workflow is divided in two main parts, (A) Request sensor data and (B) Receive sensor data. Part (A) illustrates the continuous requests for sensor values from the sensor node, as described in Section 3.7.2, see Fig. 19A. The request for each sensor must change sequentially to be able to get sensor data from all sensors in the system. Part (B) illustrates the receiving part of the master node, which starts when we have received some sensor data. Received sensor data must be interpreted from which sensor (S1, S2) the sensor data was received, see Fig. 19B. When we know from which sensor the data is
coming from, we can transmit the sensor data through USART to the Bluetooth module. By comparing the sensor data with the threshold value there is two options (OK, NOK) that is respective reflected to Section 4.3.1 as satisfied and not satisfied. Option NOK leads to the request for action to the actuator node controlled by a user input, while option OK returns the system to request the sensor data.

(A) Request sensor data

(B) Receive sensor data

Figure 19 - BodyCom base unit (master node) workflow with parts: (A): Sequential request of sensor data, (B): Receive sensor data and request for action
4.4.3 Request sensor data using Turn-taking polling protocol

Two approaches of requesting sensor data using Turn-taking polling protocol is introduced in this section. The approaches aim to how the master node handles the sequential turn-taking of requests to the sensor node.

**Approach 1:** First approach is based on the receiving part of the master node, see Fig. 19B. Initially, we give one sensor request (Send request S1) the turn to start request the sensor data from the sensor node, see Fig. 20. When the request is transmitted, we are checking if we have received the requested sensor data from sensor S1. If not, we will request the sensor data from sensor S1 until we receive the requested data back. When we have received the requested sensor data S1, we can process the received sensor data and pass the turn to request for the sensor data from sensor S2 and so on.

The main problem with this approach (considered S1 and S2 as separate nodes) is, if the first node (S1) fails or is destroyed, the second node (S2) will not be able to send any requests. As a result, the entire system will get caught in an infinite loop and lose its functionality.

The advantage of this approach (considered sensor S1 and sensor S2 never fails) is that the turn-taking is performed without any data losses within each turn, if a temporary communication error occurs, such as Cyclic Redundancy Check (CRC)-error.

![Flowchart of Turn-taking polling protocol, approach 1.](image-url)
**Approach 2:** Second approach is based on a more irresponsible turn-taking, were the turn passes neither the data have been received or not. Like **Approach 1**, we are giving the first turn to request for $S_1$, see Fig. 21. Unlike the **Approach 1**, we will not transmit repeatable requests for sensor $S_1$ while we not received any sensor data after the first request for sensor $S_1$. Instead we pass the turn to transmit the request for sensor $S_2$. If we receive sensor data from sensor $S_1$ within the first request, the requested sensor data will be processed, and the turn passed to the sensor $S_2$.

![Flowchart of Turn-taking polling protocol, approach 2.](image)

The main problem with this approach (considered sensor $S_1$ and sensor $S_2$ as separate nodes) occurs when $S_1$ or $S_2$ get a communication error, such as CRC-error. The exposed node will not have a second attempt to request the sensor data and must wait until the next node is done transmitting the request. This problem will result a data loss from the exposed node during the round of turn-taking.

The advantage of this approach is that the system will never get caught in an infinite loop, unlike the first approach.
4.5 Sensor node implementation

In this section we will go through the implementation of the sensor node with peripherals functionality description and connection.

4.5.1 Sensor node connections

The sensor node consists of two main parts, the modified copy BodyCom MU, see Section 3.6.2 and Arduino UNO board. The MU pins RB5 and RB2 are connected to Arduino UNO pins A4 and A5, respectively and these two units shares a common ground (GND), see Fig. 22. The connection is used to establish an \( I^2C \) – communication between the modified copy of the BodyCom mobile unit and Arduino UNO board. The Arduino UNO is wired to a photoresistor and a potentiometer, which acts as sensors in our system, see Fig. 22 (S1 and S2). The sensors output an analog value that can be read by analog input pins A1 and A0 of Arduino UNO.

![Figure 22 – Schematic of the sensor node. (S1): Photoresistor, (S2): Potentiometer.](image-url)
4.5.2 Proposed workflow of sensor node

The proposed workflow of the sensor node is illustrated in the flowchart in Fig. 23. The MU is always waiting for the sensor request from the master node. As soon a request has arrived, the MU interpret the request to know which sensor (S1 or S2) the master node requested for, see Fig. 23. The interpreted request is fast-forwarded to Arduino to get the sensor value for requested sensor. The Arduino respond to MU with the requested sensor value and the MU transmit the sensor value to the master node. When MU has respond to the master node with requested sensor value, the MU goes back to the Wait for request state, see Fig. 22.

Figure 23 – Sensor node workflow illustrated in a flowchart.
4.5.3 Proposed workflow of Arduino UNO

The proposed sensor reading method for Arduino UNO is illustrated in the flowchart in Fig. 24. The Arduino UNO start with sequential reading of each sensor and saves the sensor values, following by a check if any request from the MU has arrived. Described states executes one by one until a request arrives. An arrived request is interpreting to know which sensor is requested. The Arduino UNO send the requested sensor value to the MU and returns to Read and save sensor values, see Fig. 24.

Figure 24 - Flowchart of proposed sensor reading method for Arduino UNO on sensor node.
4.6 Actuator node implementation

The actuator node connects with two led diodes to simulate an action, see Fig.25. Used pins on the MU are described in Section 3.6.2. The led diodes can simulate four states of action:

1) RED = OFF and GREEN = OFF
2) RED = ON and GREEN = OFF
3) RED = OFF and GREEN = ON
4) RED = ON and GREEN = ON

The proposed workflow of the actuator node is alike the MU of the sensor node in Section 4.5.2. Where a request for a sensor, performed by the master node is exchanged against an action request. The action request is interpreted by which state of action is requested, following by the execution of action. The actuator node sends back an acknowledgement, to inform the master node that the requested action is executed and waits for forthcoming requests.

Figure 25- Schematic of the actuator node.
4.7 Interpretation of requests

In this section we will go through the interpretation of the requests in terms of e.g. indication of which sensor value is requested, which action state the actuator node should perform and so on. We will also go through sensor node internal interpretation of the requests, such as MU to Arduino UNO.

4.7.1 Interpretation of requests between the master node and sensor node

The interpretation of requests between the master node and sensor node are based on the DataBuffer, which is a part of the BodyCom data packet, structure of type MDLL_PacketData_t, see Section 4.2.1. By using the DataBuffer, see Fig. 26, we can design our own frame with segments that will have desired indication. Each segment can only consist of a multiple of a byte, which is a sort of limitation where the smallest segment is one byte and the largest is 16 bytes. Each byte can have 256 states that can be defined as an indicator, so even the smallest segment (one byte) have a lot of opportunities.

DataBuffer of type MDLL_PacketData_t

![DataBuffer Diagram](image)

Figure 26 - DataBuffer of type MDLL_PacketData_t, with enumeration of elements

The most important segment in the custom frame to request from the BU (master node) to MU (sensor node) is which sensor value we want the sensor node to deliver, called Sensor Identification (Sensor ID). Depending on how many sensors the sensor node has we can decide how many bytes we need to use for definition of each sensor for the requests, in Sensor ID segment.
The custom data packet frame sent by the MU (sensor node) to BU (master node) can be designed like the BU (master node) custom frame request. But in this case, we need to include the actual sensor data in the custom frame and therefore we need an additional segment *Sensor Data*, see Fig. 27. Depending on the resolution of sensor data, the *Sensor Data* segment can be adjusted in terms of bytes. The *Sensor ID* segment can be implemented in the same way as for the BU (master node) custom frame request, to inform the master node which is sent by the MU (sensor node).

**Custom frame using DataBuffer**

<table>
<thead>
<tr>
<th>Sensor ID</th>
<th>Sensor Data</th>
<th>Free elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1 bytes</td>
<td>n2 bytes</td>
<td>16 - (n1 + n2) = n3 bytes</td>
</tr>
</tbody>
</table>

Figure 27- Custom frame design illustration based on the DataBuffer. The frame is used for communication between the master node and sensor node.
4.7.2 Interpretation of requests between the master node and actuator node

The interpretation of requests between the master node and the actuator are based on the DataBuffer, like the Section 4.7.1, but with a distinctive design of the custom frame. In this case we are interested to give the actuator node an Actuator Identifier (Actuator ID), to indicate which actuator we are about to manipulate and some parameters that will manipulate each actuator in a desired way, see Fig. 28.

![Custom frame using DataBuffer](image)

Figure 28 - Custom frame design illustration based on the DataBuffer. The frame is used for communication between the master node and actuator node.

Having different actuators can result assorted designs of the custom frame for each actuator, where the parameter segments have unique manipulation definitions for each actuator. As an example, we can take a water pump. We may want to manipulate the water pressure, the amount of time we want the water pump to pump and set a period for the duty-cycling of the water pump.

The actuator node response can act as an acknowledgement, to indicate that required manipulation has been accepted and performed. The custom frame can be the same as sent by the master node. In this case the Actuator ID segment must have a constant placing in the custom frame for all actuators, so the master node can check the right segment to detect which actuator acknowledgement have been received.
4.7.3 Interpretation of requests between BodyCom mobile unit and Arduino UNO

Interpretation of requests between the MU and Arduino UNO in the sensor node is divided in two directions, MU to Arduino UNO and vice versa.

**Direction 1 (MU to Arduino UNO):** In this case the MU only needs to send the Sensor ID of the sensor that is requested by the master node. The amount of data frames using $I^2C$ – communication depends on the number of sensors that are used in the system.

**Direction 2 (Arduino UNO to MU):** In this case we have a different scenario compared to the Direction 1. The response frame to the MU needs to be customized to be able to provide the Sensor Data and the Sensor ID, see Fig. 29.

Depending on the number of sensors and the bit resolution of the Sensor Data the custom frame in Fig. 29 can be designed in several ways where (2) need to hold.

\[
I^2C\_DATA\_LIMIT = \text{Maximum number of bits defined as a data frame}
\]

\[
n_1 + n_2 \leq I^2C\_DATA\_LIMIT
\]  \hspace{1cm} (2)

![Custom frame using I2C data frame](image_url)

Figure 29 - Custom frame design illustration based on the $I^2C$ data frame. The frame is only valid for **Direction 2**.
4.8 Sensor data conversations

The Sensor Data that was provided by the Arduino UNO need to be converted into the DataBuffer array, so the MU of the sensor node will be able to send it to the master node. As an example, in Fig. 30, we consider the length of the Sensor Data of 32-bits of type `uint32_t`. The 32-bit Sensor Data provided by the Arduino UNO is divided in four parts of 8-bit. Each part corresponds to one element of the DataBuffer and by using logical bitwise operators can be stored in the array, see Fig. 30B.

When the master node receives a data packet with the Sensor Data in the DataBuffer array, we need to perform a conversation from 8-bit array to a `uint32_t` to be able to interpret the sensor value, see Fig. 30A. The conversation is performed using the logical bitwise operators.

It is important to keep track of the Most Significant Bit (MSB) or Least Significant Bit (LSB) for each element in the DataBuffer array to avoid conversation errors.

![Figure 30](image_url)  
**Figure 30** - (A): Illustration of 8-bit array to `uint32_t` conversation, (B): Illustration of `uint32_t` to 8-bit array conversation.
4.9 Measurement on Taking-turn protocol

To test the Taking-turn protocol we made a measurement on system responsiveness in terms of how long time it takes for the master node to transmit a request for a sensor (S1 or S2) from sensor node and receive it back. The measurement is performed by manually manipulate the sensors on the sensor node, so the sensor values is below the predefined threshold of 500. As a result, the User Interface (UI) for actuator control will appear in the PC serial terminal as an indicator. The time is measured between the manually manipulated sensor value drop and the appearance of the UI for actuator control on PC serial terminal, see Fig. 31.

The time measurement starts when we can see on the Arduino UNO serial terminal that the desired sensor value is below 500 and stops when we can see the appearance of the UI for actuator control on the PC serial terminal for Bluetooth communication. The time measurement is manually performed using a timer application on a smartphone device.

The distance between the BU (master node) coupling pad (Pad 1) and the MU (sensor node) coupling pad (Pad 2) can appreciated in Fig. 32 with a folding rule in the background as a reference in centimeter.
Figure 32 - Picture of the entire system with highlighted coupling-pads of BU (Pad 1) and MU (Pad 2).

The measurement was performed by holding a human hand 10-30cm above the coupling-pads in Fig. 32. Ten sequential time measurements have been made for each sensor.
4.10 Development environment

Software modifications on the BodyCom development board and mobile units was performed in *MPLAB X Integrated Development Environment (IDE)* version 4.15, provided by Microchip. *MPLAB X (IDE)* is a development environment used to create applications for microcontrollers provided by Microchip. It can run on most popular operative systems such as, Windows, Mac OS and Linux [29]. This project was performed on a Linux machine (*Linux Mint 18.3 ‘Sylvia’*) with an additional compiler for the *MPLAB X (IDE)*, namely *MPLAB XC8* version 1.45, that supports all 8-bit MCUs provided by Microchip. *MPLAB XC8* is a C compiler with no performance optimizations because of missing possession of a pro version of the compiler [30].

To develop with *Arduino Uno*, we used Arduino’s own development software *Arduino Software (IDE) 1.8.5* that have an inbuilt C/C++ compiler and runs on the most popular operative systems for personal computers (Windows, Mac OS X and Linux) [31].

4.10.1 Additional software

To be able to communicate with the Bluetooth module we used a software *GTK+ Bluetooth Manager*, that allows us to connect the Bluetooth communication to a serial port on the Linux machine [32]. The serial port interaction was made by *Putty*, which is a terminal emulator with different communication protocols [33]. In our case we used the serial connection, to be able to interact with the Bluetooth module.
5. Result

In this chapter we will show how the distributed system of sensors and actuators is implemented and motivate the selected method of implementation.

5.1 Master node

The master node final workflow is presented in the flowchart in Fig. 33. Fig. 33 is divided in parts A-D where we will present a detailed implementation result of each part with motivation why a certain method is selected, if needed. Parts C and D can be compared with the Fig. 19A-B in Section 4.4.2, respectively. Presented workflow in Fig. 33 represent the while loop of the main program of the master node.

Figure 33- Flowchart of master nodes workflow, (A): Display sensor values, (B): Actuator node request, (C): Sensor node request and (D): Receiving part.
5.1.1 Part A: Display sensor values using USART

Variable *DispSensor* acts as an indicator that mediates if it is time to display current sensor values stored in the *master node*, where *DispSensor* can be set to one or zero. *DispSensor* = 1 indicates that it is time to display current sensor values and by setting the variable to zero we will not display anything. The *DispSensor* variable is triggered (set to one) in the part D in Fig. 33, after we received sensor data from any sensor and the program *DisplaySensors* is executed.

5.1.2 Part A: Program *DisplaySensors*

To display the sensor values we are running the program *DisplaySensors* that handles the USART communication with the Bluetooth module. The Bluetooth module communicates with the internal Bluetooth module of the laptop used in the project and the received data from the BU is outputted in laptops serial terminal with 8-bit American Standard Code for Information Interchange (ASCII)-table encoding. The 32-bit sensor value is divided in parts number by number and stored in an array of type *char*, see Fig. 34.

![Figure 34 – Principle of the USART friendly conversation of a 32-bit integer into a char array.](image)

The conversation from 32-bit integer to a *char* array is achieved by following sequential loop of equations:

\[
\begin{align*}
\text{sensor value} \mod 10 &= a \quad \text{(3)} \\
(s\text{ensor value} - a) \div 10 &= b \quad \text{(4)} \\
\text{sensor value} - a &= 0 \quad \text{(5)}
\end{align*}
\]

Equations (3) and (4) is executed in a loop in respectively order and the variable *b* will be the new *sensor value* for every loop execution. In (3) the variable *a* is referred to third element in the *char* array in first loop execution, see Fig.34. In second loop execution the variable *a* is referred to the second element in the *char* array and so on, see Fig. 34. The execution of the loop stops when (5) is true.

By adding each number of the element in the *char* array with 48, we achieve the ASCII encoded representation of each number that is sequentially transmitted using USART.
5.1.3 Part B: Action request from actuator node

Variable Act indicates if any sensor value from sensors S1 or S2 are below the predefined threshold value of 500, see Section 4.3.1. The variable Act set to one indicates that one of sensors values is below the threshold and variable is set to zero otherwise, see Fig 33. If Act = 1 the program DisplayAct executes. The program DisplayAct will transmit a string to the serial terminal on the laptop, see Section 5.1.1, that asks the user which action the user want the actuator node to perform. The user has three input alternatives from the keyboard (r, g and y) that defines the actions simulated by a LED with three colors red (r), green (g) and yellow (y) implemented in the actuator node. Chosen input alternative is treated by the function ActionRequest.

5.1.4 Part B: Function ActionRequest

Function ActionRequest is a modified version of function BC_SendDataCommand, that performs the transmission of a data packet of type MDLL_PacketData_t to a BodyCom MU, see Section 4.2.2. In this case, the function ActionRequest is designed to transmit data packets only to the MU of the actuator node.

The function ActionRequest takes three inputs:
1) **Actuator ID** – Which actuator to manipulate
2) **Data length** – Length of DataBuffer in bytes
3) **Timeout** – Given time to perform the transmission of a BodyCom data packet and receive a response from the MU before counted as a data packet loss

Input Timeout is based on the length of the DataBuffer. If the Timeout value is not large enough for a specific data length, then all data packets will be lost. The predefined Timeout value (PING_TIMEOUT) by the BodyCom system is 24. We are using a Timeout value that is three times larger than the predefined value of (PING_TIMEOUT), to ensure that no losses of data packets occur.

Function ActionRequest create a temporary data packet of type MDLL_PacketData_t and build the temporary data packet by load the needed information, see Table. 4.

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Loaded information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command</strong></td>
<td>ECHO_REQUEST</td>
</tr>
<tr>
<td><strong>Address</strong></td>
<td>TAG_2_ADDRESS = {0x03, 0x03, 0x03, 0x03}</td>
</tr>
<tr>
<td><strong>DataLength</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>DataBuffer</strong></td>
<td>See custom frame design in Fig. 33</td>
</tr>
</tbody>
</table>

Table 4 - Built data packet of type MDLL_PacketData_t
In Table 4 the ECHO_REQUEST is the only Command that accepts a data packet of type MDLL_PacketData_t with a DataBuffer with DataLength that is larger than zero.

The DataBuffer is considered as a custom frame, see Section 4.7.2. Each segment of the custom frame corresponds to one element in the DataBuffer, see Fig. 35. The ActionRequest function input Actuator ID is placed in the first (zeroth) element of the DataBuffer followed by Parameter 1-3 which is unused elements and a MU-ID that act as an additional security control ID of the MU in actuator node, see Fig. 35. Lastly, the function ActuatorRequest return the function MDLL_sendPacket with pre-built temporary packet as an input, see Section 4.2.2.

5.1.5 Part C: Sensor data request from the sensor node
The turn-taking of requests for sensor data is based on the variable sensor_type. The variable sensor_type can be set to zero or one that indicates request for sensor data from S1 or S2, respectively, see Fig. 31. The request for sensor data is performed by execution of the function SensorRequest.

```
<table>
<thead>
<tr>
<th>Actuator ID</th>
<th>Parameter 1</th>
<th>Parameter 2</th>
<th>Parameter 3</th>
<th>MU-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>1 byte</td>
<td>1 byte</td>
<td>1 byte</td>
<td>1 byte</td>
</tr>
</tbody>
</table>
```

Figure 35 - Custom frame design illustration based on the DataBuffer. The frame is used for communication in master node to actuator node direction

5.1.6 Part C: Function SensorRequest
The function SensorRequest works in an equivalent way as the function ActionRequest in Section 5.1.4. The difference is that we are using another Address for MU on the sensor node and the custom frame segment loaded with Sensor ID instead of Actuator ID, see Fig. 36. The Address of the sensor node is stored in variable TAG_2_ADDRESS, where TAG_2_ADDRESS = {0x04, 0x04, 0x04, 0x04}.

```
<table>
<thead>
<tr>
<th>Sensor ID</th>
<th>Parameter 1</th>
<th>Parameter 2</th>
<th>Parameter 3</th>
<th>MU-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>1 byte</td>
<td>1 byte</td>
<td>1 byte</td>
<td>1 byte</td>
</tr>
</tbody>
</table>
```

Figure 36 - Custom frame design illustration based on the DataBuffer. The frame is used for communication in master node to sensor node direction.
5.1.7 Part C: Taking-turn protocol

The turn-taking method of sensor data requests is based on approaches in Section 4.4.3. Both approaches have their advantages and disadvantages in terms of data loss in certain situations. As a result, we used both approaches in our implementation to provide better robustness in terms of data loss. **Approach 1** have advantages that covers **Approach 2** disadvantages and vice versa, see Section 4.4.3.

Implementation of **Approach 1** can be found in part D in Fig. 33, where the pass of the turn is performed after the master node has received a data packet. **Approach 2** is implemented in part D in Fig. 33, where the pass of the turn occur in the same block as the function **SensorRequest** is executed.

5.1.8 Part D: Receive and process data

Program **BC_Handler** monitors if a data packet of type **MDLL_PacketData_t** is received. The indicator of a received data packet, which is used by **BC_Handler**, is when the function **MDLL_receiveDataPacket** returns a value that is bigger than zero, see Section 4.2.3. Received data packet is interpreted by the program **BC_Handler** to know which node (sensor node or actuator node) the data packet was transmitted from. Interpretation is done when both conditions (6) and (7) are fulfilled.

\[
\begin{align*}
\text{Command} &= \text{ECHO\_RESPONSE} \quad (6) \\
\text{DataBuffer}[4] &= \text{MU\_ID or Sensor ID} \quad (7)
\end{align*}
\]

The command **ECHO\_RESPONSE** in (6), is the same for all nodes. The interpretation takes place in (7) when the read Mobile Unit Identification (MU\_ID) or Sensor ID is associated with one of MUs of existing nodes, see Table 5.

<table>
<thead>
<tr>
<th>Node</th>
<th>MU_ID/Sensor ID</th>
<th>ID [value]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor node</td>
<td>Sensor ID (S1)</td>
<td>195</td>
</tr>
<tr>
<td>Sensor node</td>
<td>Sensor ID (S2)</td>
<td>60</td>
</tr>
<tr>
<td>Actuator node</td>
<td>MU_ID (ACK)</td>
<td>70</td>
</tr>
</tbody>
</table>

Table 5 - Association of system nodes with MU\_ID/Sensor ID.

The actuator node transmits data packets containing an acknowledgement byte, were the acknowledgement byte is not associated with the Actuator ID. In this case the segment MU\_ID is used to interpret an acknowledgement transmitted by the actuator node.
The sensor node transmits data packets containing sensor data from different sensors. In this case the MU of sensor node transmits data packets with Sensor ID instead of just a MU_ID, to interpret which sensor the sensor data is coming from.

### 5.1.9 Part D: Acknowledgement handling

The acknowledgement (ACK) sent by the actuator node informs the master node that the requested manipulation of an actuator is performed, see Fig. 33 and Table. 5. When the BC_Handler receive a data packet with an ACK it simply output a string on the BodyCom BU LCD that informs the user that the requested manipulation of an actuator is performed.

### 5.1.10 Part D: Processing sensor data

Received sensor data are also processed in the program BC_Handler. The received packet containing sensor data is copied to a global packet of type MDLL_PacketData_t associated with the Sensor ID see Section 4.1.3 and the turn is for SensorRequest is passed to next sensor, by changing the sensor_type variable. Global packet DataBuffer array elements holding the sensor data, see Fig. 37. Where the array is converted to a 32-bit global variable of type uint32_t, by using the method in Section 4.8.

Sensor data stored in the 32-bit global variable, where the variable is converted to an

<table>
<thead>
<tr>
<th>MSB</th>
<th>Sensor Data</th>
<th>Sensor ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Figure 37 - Custom frame design illustration based on the DataBuffer. The frame is used for communication in sensor node to master node direction.

USART friendly global array using the method in Section 5.1.2 and the DispSensor variable is set to one, to indicate that the new sensor data is ready for USART transmission, see Fig. 33.

The comparison of received sensor data and the predefined threshold values is performed the program ActAction called by the BC_Handler. Program ActAction controls if condition (8) is fulfilled.

\[
\text{sensor value of } (S1) \text{ or } (S2) < 500
\]

(8)

If condition (8) is fulfilled, the variable Act will be set to one, to indicate that an ActionRequest is needed, see Section 5.1.4. Unfulfilled condition (8) results a return to part A of the workflow, see Fig. 33.
5.2 Sensor node

The sensor node workflow is presented in flowchart in Fig. 38. The sensor node is divided in two parts A-B, where part A illustrates the workflow of the MU and part B of the Arduino UNO. Part A-B can be compared with the proposed workflow in Section 4.5.2 and part B only in Section 4.5.3.

Presented workflows in Fig. 38 represents the while loop in the main program for respectively part.

*Figure 38 - Flowchart of sensor node workflow, (A): The BodyCom mobile unit, (B): The Arduino UNO.*
5.2.1 Part A: BodyCom mobile unit

To monitor incoming data packets of type MDLL_PacketData_t, send by the master node, we use a modified version of original function BCM_CommandHandler, provided by Microchip. The indication of a received data packet, interpretation of requested Command and MU-ID is equivalent to the function BC_Handler in Section 5.1.8.

The Sensor ID is sent to Arduino UNO using 8-bit \(I^2C\)-communication protocol, to inform which sensor value is required from the master node. While saving the received data packet, we give the Arduino UNO time to handle the request for the Sensor ID, before requesting the sensor value with associated Sensor ID, see Fig. 38A. The request for sensor value is performed using 32-bit \(I^2C\)-communication protocol where the sensor data is stored in a variable with equivalent bit-length.

As an additional control of sensor value (received from Arduino UNO) association with the Sensor ID, we are using the method presented in Section 4.7.3. The custom 32-bit \(I^2C\) data frame is illustrated in Fig. 39. The Sensor ID is represented with one bit where the Sensor ID bit have two states (one and zero). Sensor ID bit state one corresponds to S1 and Sensor ID bit state zero to S2.

### 32-bit custom \(I^2C\) data frame sent by Arduino UNO

<table>
<thead>
<tr>
<th>Sensor ID</th>
<th>Free Space</th>
<th>Sensor Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>21 bit</td>
<td>10 bit</td>
</tr>
</tbody>
</table>

Figure 39 - Custom frame design illustration based on \(I^2C\) data frame. The frame is used for communication in Arduino UNO to BodyCom mobile unit, direction.

Received sensor value from Arduino UNO is converted into DataBuffer array, illustrated in Fig. 35. The Sensor ID is loaded in the DataBuffer array segment for Sensor ID, as an 8-bit value which is known for the master node, see Fig. 37.
5.2.2 Part B: Arduino UNO

Arduino UNO workflow is presented in Fig. 38B. To interpret which sensor value (S1 or S2) need to be delivered to the MU, the Arduino UNO is performing a request for the Sensor ID using 8-bit I2C-communication protocol and saves the Sensor ID.

The sensor values are read by Arduino UNO internal 10-bit A/D-converter for both sensors S1 and S2 [24], see Section 4.5.1 in Fig. 22 for schematic over the connected sensors. Sensor values are stored in 32-bit unsigned integer variables.

Sensor value from sensor with associated Sensor ID is transmitted back to the MU using the custom 32-bit I2C data frame in Fig. 39.
5.3 Actuator node

The actuator node workflow is illustrated in flowchart in Fig. 40. The workflow represents the while loop in the main program of MU in the actuator node.

![Flowchart of actuator node workflow](image)

Indication of received data packets of type `MDLL_PacketData_t` and interpretation of the received `Command` with the Actuator ID is handled by an equivalent `sensor node` function `BCM_CommandHandler`.

The LEDs are set as described in Section 4.6 associated to the Actuator ID, see Table. 6. Note that the yellow color in Table. 6 is achieved by turning on the green and red LED.

<table>
<thead>
<tr>
<th>Actuator ID</th>
<th>Pin settings</th>
<th>LED color</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>LATB6 = 1, LATB7 = 0</td>
<td>RED</td>
</tr>
<tr>
<td>20</td>
<td>LATB6 = 0, LATB7 = 1</td>
<td>GREEN</td>
</tr>
<tr>
<td>30</td>
<td>LATB6 = 1, LATB7 = 1</td>
<td>YELLOW</td>
</tr>
</tbody>
</table>

Table 6 - Actuator ID value for corresponding pin settings and LED colors, that act as actuators

The response data packet is loaded with an acknowledgement in form of an 8-bit Acknowledgement ID = 195 and sent to the master node. The Acknowledgement ID is loaded into the `DataBuffer` array in fourth element.
5.4 Results of measurement on Taking-turn protocol

The responsiveness measurements on implemented Taking-turn protocol performance were performed for both sensors S1 and S2, see Section 4.9. Measured results are presented with seconds as a time unit in Table 7.

<table>
<thead>
<tr>
<th>Measurement nr</th>
<th>Photoresistor (S1) [s]</th>
<th>Potentiometer (S2) [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.58</td>
<td>2.28</td>
</tr>
<tr>
<td>2</td>
<td>2.93</td>
<td>4.52</td>
</tr>
<tr>
<td>3</td>
<td>2.52</td>
<td>3.62</td>
</tr>
<tr>
<td>4</td>
<td>6.64</td>
<td>3.31</td>
</tr>
<tr>
<td>5</td>
<td>2.27</td>
<td>1.39</td>
</tr>
<tr>
<td>6</td>
<td>8.96</td>
<td>11.98</td>
</tr>
<tr>
<td>7</td>
<td>4.45</td>
<td>0.95</td>
</tr>
<tr>
<td>8</td>
<td>8.70</td>
<td>4.76</td>
</tr>
<tr>
<td>9</td>
<td>0.82</td>
<td>8.13</td>
</tr>
<tr>
<td>10</td>
<td>3.39</td>
<td>6.41</td>
</tr>
</tbody>
</table>

Table 7- Measurement result on implemented Taking-turn protocol performance

Where the longest time for S1 is 10.58 s and the longest time for S2 is 11.98 s. Shortest time for S1 is 0.82 s and the shortest time for S2 is 0.95 s. The average time for S1 is 5.126 s and for S2 is 4.735 s.
5.5 Resulting prototype

The resulting prototype of the distributed system of sensors and actuators using capacitive body-coupled communication is presented in Fig. 41.

Figure 41 - The resulting prototype of the distributed system of sensors and actuators using capacitive body-coupled communication.

System components are highlighted and marked with a letter in Fig. 41 with following description:

A. Bluetooth module connected to the BodyCom base unit
B. The BodyCom base unit (*master node*)
C. Coupling pad of the base unit (*master node*)
D. Coupling pad of the original BodyCom mobile unit (*actuator node*)
E. Coupling pad of the modified BodyCom mobile unit (*sensor node*)
F. The entire *sensor node*
G. The entire actuator node
H. The LED that simulates an actuator in the *actuator node*
I. The potentiometer (*S2*) used as a sensor in the *sensor node*
J. The photoresistor (*S1*) used as a sensor in the *sensor node*
K. The Arduino UNO of the *sensor node*
L. The modified BodyCom mobile unit of the *sensor node*
In Fig. 42, the sensor values are visualized in a terminal of a laptop, see Section 4.10.1. Where PhotoSensor stands for the photoresistor with respective sensor value, see Fig. 42. Min_value1 and Min_value2 are representing the threshold value mentioned in Section 5.1.10.

![Figure 42 - Screenshot of the terminal in a laptop with sensor values.](image1.png)

In Fig. 43, the user interface for actuator control is shown. Where the user chooses which actuator to manipulate after one of sensor values is below the predefined threshold value.

![Figure 43 - Screenshot of the terminal in a laptop with user interface for actuator control.](image2.png)
6. Discussion and conclusion

In this chapter we will discuss the work that have been done with the achieved results.

6.1 Measurement on Taking-turn protocol

The measured path time can be considered as a longest path, when the system need to react when the one of the sensor values bypasses the defined threshold, see Section 4.9 in Fig. 31. The measured time is not precise because we used a timer application on a smartphone where the timer was triggered manually by hand at visually indicated measurement points. The time measurement of the longest path gives us an idea of how long it takes for the system to travel the measured path and as a result it showed us that no sensor request is absent.

The performed measurement is not valid for verification in a replicated/existing system in terms of measured time due to unprecise measurement method with weak definition of measured path and measurement circumstances. This measurement can be used in replicated/existing system to validate that no requests is absent, as a measure for systems robustness.

The time variation in Section 5.4, in Table. 7 can depend on many factors such as propagation loss for capacitive-coupled communication [5], poor capacitive-coupling to the human body and the given turn for measured sensor using Turn-taking protocol. Propagation loss for capacitive-coupled communication and poor capacitive-coupling to the human body result data losses in terms of request sent between the BU and MU. As a result, the data losses will affect the measured time as the time became longer. Initial turn for a measured sensor request matters in terms of measured time. If the turn is given to S1 and we are measuring the time for S1 then it will result a faster response than if the turn is given to S2 while measuring time for S1.
6.2 Ethical analysis

Having a distributed system of sensor and actuators using capacitive-coupled communication for healthcare purposes will benefit a lot of people in terms of better life quality and reduced healthcare costs [6]. Both benefit factors are highly desired and highly dependent on each other, where the main factor is the healthcare costs. As an example, such concept of the system can be able to handle many diseases without a face to face interaction with the doctor and as a result the hospitals and people can save the economic resources and maybe save lives wirelessly. There are several areas in healthcare that is negative affected by limited economic resources such as salaries for the hospital staff [2].

There is also a backside of the system in terms of personal vital data that can be accessed over the internet. We think that people may be concerned over the security of the entire healthcare solution, included our system. Theoretically such a system can be manipulated wirelessly for bad purposes, which can be life critical, as an argument why people might be concerned. This whole healthcare solution need to be integrated with society to be able to pull out the benefits. There are technologies that is life critical and have been integrated with society, namely a defibrillator and the vital personal data is already stored in hospitals computers without people concerning the data access.
6.3 Conclusion

The resulting distributed system of sensor and actuators using capacitive-coupled communication prototype means that it is realistically to make such a system using the BodyCom development kit and gives us a better technical picture of future opportunities of the system. The implementation result is not perfect in terms of system architecture, optimization and design, but can be drastically improved. The reason for system imperfection is lack of time, for one would like to spend more time on several parts of the system to find better solutions. This also acts as motivation behind the choice of the Taking-turn protocol over TDMA where the chosen multiple access protocol was less complicated to implement and provided desired robustness for the system. The custom frames designs can be optimized in terms of minimizing the length of frames which can result faster transmissions. Systems architecture can be improved with better definition of interaction between objects in the system and more modularly structured.

Systems specification in Section 1 is not fully fulfilled when it comes to desired types of sensors and actuators. The resulting prototype excludes the OIEP, gyroscope and the strain sensor due that the components did not arrived in time, excluding the gyroscope. The gyroscope was not implemented because it was no need to have it to proof that the system works, and it saved us a lot of time during the implementation. However, the system can manage to transmit several types of numbers (positive/negative numbers and floating-point numbers) due to the 16 bytes bit-length of DataBuffer in MDLL_PacketData_t with additional software-drivers for conversation.

The robustness of the system is limited to the requests sent between the nodes. There the focus is on that all sensor values must be delivered to master node and no sensor is absent. This also applies to requests for actuator control.

With performed investigation of the BodyCom development kit we found some restrictions on the board but also available I/O-pins with needed peripherals and communication, see Section 4.2.4. The BodyCom development board is not fully investigated, but just enough for our purposes.
6.7 Future work

The definition of robustness of the system can be known by performing an analyze to find the critical parts in the system and find suitable solutions. The robustness of the system is directly referred to security of the system. The ensure more secure capacitive-coupled communication, the data transmitted through the human body can be encrypted. The architecture of the system can be more modular structured and with more respect to robustness of the system, so failed modules would not affect other modules as a chain reaction.

Custom frames designs can be optimized to the set of sensors and actuators the future system will use. The OIEP-pump and the strain sensor are a known future implementation where the custom frame can form a more realistic design and there can be added software-drivers for sensor value conversations into/from array, see Fig. 44.

A custom frame may be designed for USART communication using the Bluetooth module for a better structure for future integration with smartphone devices. The size of the system nodes can be reduced to a bracelet size. That implies a further investigation of BodyCom development kit to be able to control the signal strength for capacitive-coupling to the human body where the signal strength is highly dependent on the power supply, which will probably be a battery due to the desired mobility of the system. The entire system need to be optimized for lower power consumption to become more portable.

There may be a better alternative to Turn-taking protocol among multiple access protocols that will be more robust and a better fit for the system.
7. References


8. Appendices

This chapter includes needed appendices for this thesis, including the code for the program for the system, list of mentioned programs with description and location.

8.1 Appendix A

Program code of the system can be found on the ISY server of Linköping University. 
/site/edu/es/EXJOBB/bodyLink/BodyComSensorSystem_andma590

The program code can also be requested from J Jacob Wikner or Andrey Maleev.
8.2 Appendix B

List of the programs mentioned in the thesis with description and location. The location is described as the name of the source-file, where the program is located.

<table>
<thead>
<tr>
<th>Program/function</th>
<th>Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC_SendDataCommand</td>
<td>An original function that is used to send data packets in the BodyCom system</td>
<td>BC_Application.c</td>
</tr>
<tr>
<td>DLL_receiveDataPacket</td>
<td>An original program that indicates a received and available packet in the receive buffer in the BodyCom system</td>
<td>MDLL_DataLinkLayer.c</td>
</tr>
<tr>
<td>DisplaySensors</td>
<td>A custom program that displays received sensor values using the USART communication and Bluetooth module</td>
<td>Main.c in the BodyCom base unit</td>
</tr>
<tr>
<td>DisplayAct</td>
<td>A custom program that displays user interface for action requests using the USART communication and Bluetooth module</td>
<td>Main.c in the BodyCom base unit</td>
</tr>
<tr>
<td>ActionRequest</td>
<td>A custom function that is used to send action requests</td>
<td>Main.c in the BodyCom base unit</td>
</tr>
<tr>
<td>SensorRequest</td>
<td>A custom function that is used to send sensor value requests</td>
<td>Main.c in the BodyCom base unit</td>
</tr>
<tr>
<td>BC_Handler</td>
<td>A custom program that process the received data</td>
<td>Main.c in the BodyCom base unit</td>
</tr>
<tr>
<td>ActAction</td>
<td>A custom program that compares sensor values to the threshold value</td>
<td>Main.c in the BodyCom base unit</td>
</tr>
<tr>
<td>BCM_CommandHandler</td>
<td>An original program that process the received data</td>
<td>BCM_Application.c</td>
</tr>
</tbody>
</table>