Userfriendly Wearable Networking for Interactive Fashion

Bachelor's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Timm Meiwes

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Ulrik Schroeder

Registration date: 18.01.2019
Submission date: 18.05.2019

Eidesstattliche Versicherung

______________________________	______________________________
Name, Vorname	Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

____

____

____

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

______________________________	______________________________
Ort, Datum	Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

______________________________	______________________________
Ort, Datum	Unterschrift

Contents

List of Figures

List of Tables

Abstract

This thesis introduces a system for a modular, user-friendly, wearable development system. It uses a modified UART Serial communication protocol to facilitate the communication between the system and a number of modules. These modules follow a strict design code but allow for complex sensors and actuators. The system is capable of localising connected modules, which allows it to react to changes on the fly. It can be used for educational purposes by simulating a breadboard, where the modules represent the components.

This thesis describes the changes that allow a UART communication between more than two participants as well as the method of localising the individual modules. It also defines the standards with several demonstrator modules, which all the other must follow. This allows the creation of further modules in the future to create a more complex toolkit.

Überblick

Diese Arbeit stellt ein System für ein modulares, benutzerfreundliches, tragbares Entwicklungssystem vor. Es verwendet ein modifiziertes UART Serial Kommunikationsprotokoll, um die Kommunikation zwischen dem System und einer Reihe von Modulen zu ermöglichen. Diese Module folgen einem strengen Designcode, ermöglich aber komplexe Sensoren und Aktoren. Das System ist in der Lage, angeschlossene Module zu lokalisieren, so dass es jederzeit auf Änderungen reagieren kann. Es kann für Bildungszwecke verwendet werden, indem es ein Breadboard simuliert, wobei die Module verwendete Komponenten repräsentieren.
Diese Arbeit beschreibt die Änderungen, die eine UART-Kommunikation zwischen mehr als zwei Teilnehmern ermöglichen, sowie die Methode zur Lokalisierung der einzelnen Module. Es definiert auch den Standard mit mehreren Demonstratormodulen, denen alle Module folgen müssen. Dies ermöglicht die Erstellung weiterer Module in der Zukunft, um ein komplexeres Toolkit zu erstellen.

Acknowledgements

I thank Prof. Dr. Jan Borchers and Prof. Dr. Ulrik Schroeder for being my auditors as well as granting me access to the FabLab.

I also thank my supervisor Jan Thar to whom I could always talk when I ran into problems.

Thank you!

Conventions

Throughout this thesis we use the following conventions.

Text conventions

Definitions of technical terms or short excursus are set off in coloured boxes.

EXCURSUS:
Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
Excursus

Source code and implementation symbols are written in typewriter-style text.

`myClass`

The whole thesis is written in Canadian English.

Download links are set off in coloured boxes.

File: myFile[a]

[a]http://hci.rwth-aachen.de/public/folder/file_number.file

Chapter 1

Introduction

While there are many systems that have some kind of networking functions, non of them focusses on the networking aspect. This paper introduces a system with the main goal to be a light weight but efficient for networking a number of microcontrollers. While focussing on networking the aspects of user-friendliness and wearability are considered as well.

The system is divided into two parts. The board handles the communication, which has two modi, in which incoming data is handled, as well as the localisation of chips, which are connected to the board. The board section describes the early ideas and why they were abandoned. It explains the decision for the hardware used and the software challenges that arose because of it.
The section on the chips discusses the standard chip, which is adapted to form all designed chips so far. The section is subdivided into two based on the complexity of the chips introduced. It is concluded by explaining the process to overcome the challenges that were encountered in the design of the chips.

To conclude the paper, the system is evaluated based on how well it performs in regards to user-friendliness, wearability and networking. It also includes thoughts on what might be implemented in a future review.

Chapter 2

Related work

There are already a number of systems that are capable of performing partly similar functions to the discussed system. This section will introduce them in a very brief form.

2.1 Commercially available systems

The first system is called TinkerBot, it is produced by Kinematics. It is a system made up of a number of blocks, both active and passive, that can be directly connected to one another.

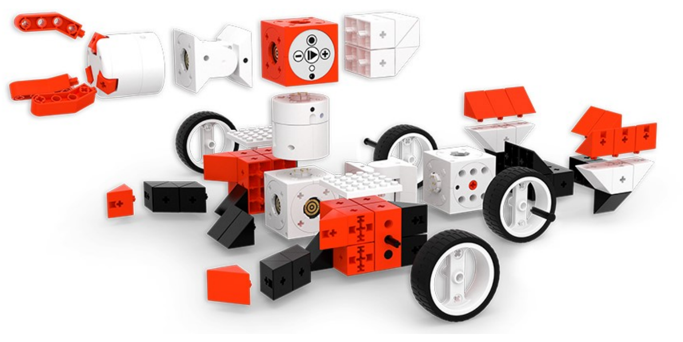

Figure 2.1: Example of one TinkerBot product

The square red-orange cube, in the upper middle of the figure, is called the Powerbrain. It is similar to the board, see Section 3.3. It is responsible for controlling the individual sensor and actuator blocks while at the same time it is providing power to the individual modules. The only information about how the communication is handled is found on their indiegogo page. It is said there, that the Powerbrain uses a Serial UART bus-system for communication. Further more, the Powerbrain is also capable to communicate with mobile devices via Bluetooth.
There are only motion focused actuators such as a motor, a tilting joint, one that is akin to a servo and a gripper. There is a distance and a light sensor for input. The Powerbrain also contains a three axis accelerometer and gyroscope.
The whole system runs on a built-in lithium polymer battery.

The second commercially available system is Qwiic by SparkFun. It is less of a whole system but a connection and communication scheme. There is no direct code or master module readily available. It relies on the user to create such a module but provides the user with an easily to use set of components. All these components are based on the same design code. They use I2C for communication and have the same pinout on all boards. Using standardised and polarised connectors the user does not need to worry about connecting the modules in the correct way. It is possible to daisy chain modules because of the nature of I2C. However, SparkFun does not provide a finished system in a sense, that once the user has connected the modules the system knows what to do. A positive aspect is that users are not limited to SparkFun products. Anyone can create modules that work in tandem with Qwiic modules because the pinout, the connector used, and the operating voltage of 3.3V is known.

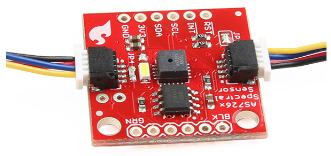

Figure 2.2: A Qwiic component

2.2 Academic systems

The first academic system is the Tangible Video Editor. It was designed by Zigelbaum et al. [2007] at MIT and Tufts University.
It enables the user to create video clips by connecting different modules together. There is a play-controller which is connected to the user's computer, see Figure 2.3 lower right. The user can connect individual clip-holders to it which contain a video ID. Before the system is used, the IDs are matched to video clips by the software. After the first clip-holder is attached the user can connect further clip-holders. There is a slot between two clip-holders, in which the user can place one of three transition connectors. These determine what kind of transition the software uses. To transmit data the furthest clip-holder on the right side sends its ID to the previous one via a frequency modulated audio signal encoding ASCII data. This clip-holder appends the data stream by its own ID, the transition used and sends it to its left neighbour. Once the data stream reaches the play-controller it is sent to the PC where the video is assembled according to the video IDs and transitions.

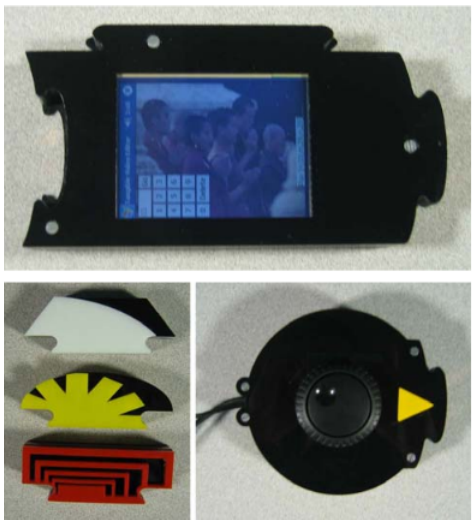
Figure 2.3: The Tangible Video Editor system

The second system is the ActiveQube by Watanabe et al. [2004] from the University of Osaka. It is designed to allow the user to create virtual objects by connecting a number of ActiveQubes to each other. The whole system is connected to a PC via a base cube. By connecting or disconnecting other cubes the user can create virtual objects in real time. Each cube has its own ID as well as one ID per face to allow identification of which faces are connected between two cubes. The communication is done via a RS-485 port.

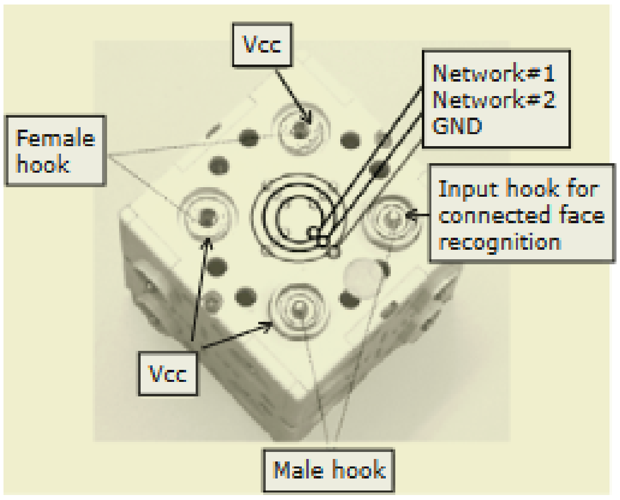

Figure 2.4: An ActiveQube with labelled connectors

FlowBlocks is another system, designed by Zuckerman [2007] from the MIT. Its aim is, to help children develop an intuitive understanding of dynamic relationships. It is a set of modules that try to visualise dynamic processes by using a number of lights. Starting from an Inflow module other modules can be connected via magnets. It is important to include a Straight module as it contains the batteries that power the whole system. Communication is done by a 16 bit serial connection which uses the magnets as connectors.

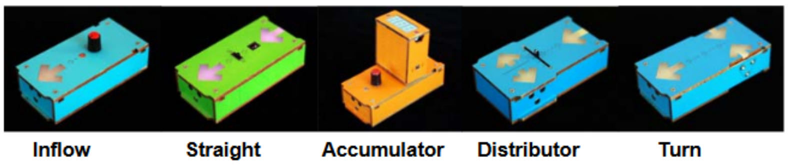

Figure 2.5: The FlowBlocks modules

Last but not least there is the Flowboard by Brocker et al. [2019] from the i10 chair of the RWTH Aachen. It uses an iPad as a main controller, on which the user can design programs using Flow-Based Programming. The iPad interfaces with an Arduino Uno which controls the hardware input/output pins via Bluetooth. Any change in the flow graph takes immediate effect. This is possible because of the slightly altered Firmata protocol used. The pins of the Arduino are connected to a custom switching PCB which allows the iPad to connect them either to the input (to the left of the iPad) or output (to the right) side of the board.

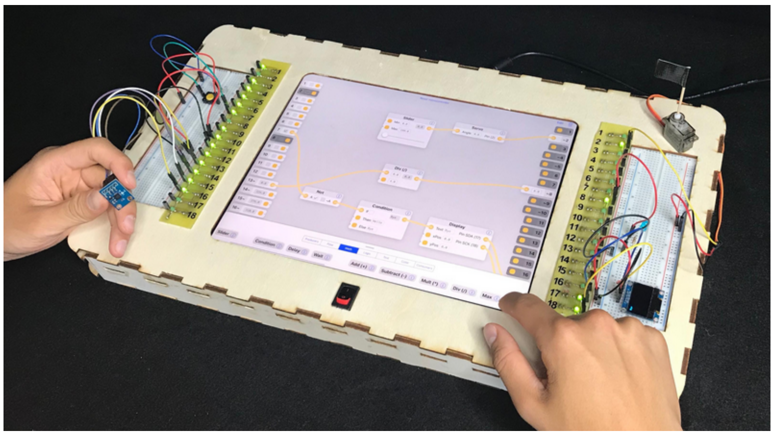

Figure 2.6: The Flowboard in use

Chapter 3

Own work

The goal of this thesis was to create a user-friendly wearable networking system. It consists of a main board, which handles the networking and a number of chips which can be placed at any connection point at any time. This section explains the software requirements and the reasons why a UART Serial communication was chosen. Afterwards it introduces the board, explains why an Arduino MKR1000 was used as the main microcontroller and explains what is needed for the communication. The section closes by discussing the chips designed so far and what challenges arose in the design process.

3.1 Requirements

There were certain requirements for the system that had to be met: First the system had to be able to communicate with multiple chips. An appropriate communication protocol had to be chosen for that. A weak point in each wearable system are the connectors and connections, as they endure the most stress, either from frequent connecting and disconnecting, or from bending while wearing it. Because of this the goal was to minimise the number of connections needed. For that reason SPI was excluded from the beginning. The two remaining protocols which are supported by

From here on the term Serial refers to the UART Serial communication protocol

the AtTiny are I2C and Serial. The decision process is explained in the section below.

For the first mode the system does not need to know where the chips are located, however, it is imperative for the second mode. For that the system needs a way to find and store the location of any connected chip.

The chip requirements were very simple. Aside from the pinout described in Section 3.4 and the wish to use an AtTiny85 there were no other requirements.

3.1.1 I2C vs. Serial

I2C and Serial are both protocols that are usually directly supported by the hardware of most microprocessors. This and the fact, that they are fairly simple and only use two pins, make them the best choices for infrequent and, compared to SPI, slow data communication. This section will explain the reasons why Serial was chosen instead of I2C.

The I2C communication protocol is a master/slave system. It uses a two-wire-bus: the first wire carries a clock while the second carries the data signals. It uses a 7 bit address for each module, thus there are 127 different possible addresses. Each module is connected to the same bus.
The reasons why I2C is not a suitable candidate for the system lies within the master/slave setup of the protocol. In this configuration only the master is able to initiate the communication. This disqualified I2C because the chips were supposed to be able to start sending data as soon as they get them. Another draw back of I2C is that each module, that is connected to the bus, needs to have a unique address. It should be possible to connect chips with the same address to the board.

The Serial communication protocol also uses two wires, but instead of using one wire for a clock signal Serial uses both for data transmission. Each is carrying the communication in one direction. Thus the sender of one module (called tx, short for transmitter) is connected to the receiver (called rx)

of the other and vice versa. In contrast to I2C both modules can start to send data at any time. The data is buffered until the receiver reads it. As Serial communication is only between two modules there is no need for an address.
That is a disadvantage of Serial compared to I2C. Serial communication is only between exactly two participants.
In theory, it is possible to connect each tx pin to each rx pin of all the other modules. In praxis, this would not scale to more than a couple of modules.

Serial was chosen as the chips should be able to initialise data transmission as well as the board. My task was to find a way around the limitation.

3.2 Early Work

A first attempt was to use the Arduino SoftwareSerial library and simply combine both the sender and receiver function at the same pin. This would, in theory, produce a bus, like I2C uses, and cut down the numbers of pins needed, which would in turn increase the number of usable pins at the AtTiny.

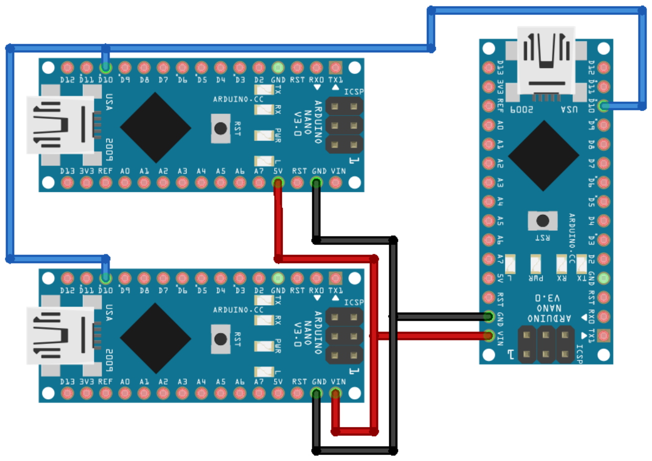

Figure 3.1: Attempt for one wire communication

This did not work. In theory, all that had to be done was to change the pin from receiver to sender each time the chip had to send data. In praxis, the write buffer needed

to be cleared before this could happen. There is a function, called flush(), within the SoftwareSerial library. However, this function only clears the input buffer. As SoftwareSerial is not intended for duplex use, it does not need to be able to clear the write buffer.
There exists a library called SoftwareSerialWithHalfDuplex by the user nickstedman on GitHub. This library had implemented all that was needed. It offered Serial communication with half duplex over one wire.
After a positive test with two Arduino Nanos communication over one wire was also successful with a combination of four AtTiny and one Arduino Nano.

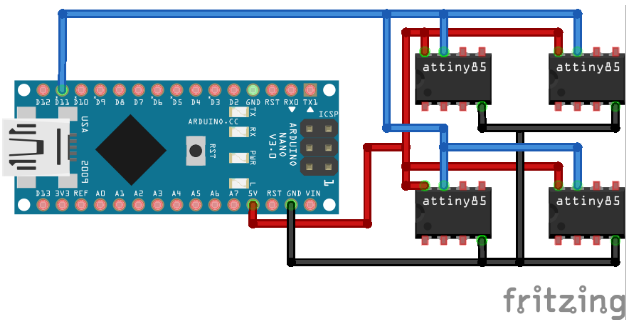

Figure 3.2: Successful one wire communication

Addresses similar to I2C were used to identify the microprocessors. Each microcontroller got a character as an address assigned. In front of each data byte the sender had to place one byte containing the address character. Each other microcontroller read the address byte as soon as it had arrived, compared it to its own address and decided whether the data, that it had read directly after the address, was intended for it or not.

3.3 Board

The main microcontroller will be called the board from now on. It handles all the communication by checking where incoming data is from and readdressing it to the correct chip, according to the selected mode. Beside communication it is also responsible for localising chips, which are connected to it, as well as handling the button, which changes the mode.

3.3.1 Upgrade

Before progressing any further from the state, described at the end of the last section, and keeping the user-friendly aspect in mind, the decision was made against using an Arduino Nano as the main microcontroller. To allow for more complex programs and the addition of further hardware, which need more RAM space, the decision was made to upgrade. Another deciding factor was, that while it is possible to have multiple SoftwareSerial ports in the system, it is not possible to have them all listen at the same time. So if in the future the need arises to have multiple buses, a system that uses SoftwareSerial will not work.

The choice was made to use the Arduino MKR1000 (for better readability called MKR). It has 32kb of RAM compared to 2kb of the Nano and the possibility to configure multiple hardware serial ports. It is, however, no longer an AVR chip.
It was no longer possible to use the SoftwareSerialWithHalfDuplex library, on which the whole setup was relying, because the MKR uses an ARM chip.

The first test was simply to connect the tx pin of the MKR to all the rx pins of the Arduino Nanos and to connect all the Nanos tx pins together to the rx pin of the MKR. As the chips do not need to send data directly to each other, they do not need connections between them. This was only partly successful. The MKR was able to send data to all the Nanos, but the other way round did not work.
This behaviour is caused by electric impedance. A single Nano is not able to drive the communication line low enough when there is more than one receiver. However, this challenge can be overcome by using a simple diode (in this case a 1N4148 signal diode), creating the impression for an individual Nano, that there is no other communication partner than the MKR.

3.3.2 Localisation

The initial idea was to localise chips, which are connected to the board, by using a similar technique as used to drive a LED dot matrix. The connections were organised in a matrix form with rows and columns. By applying 5V and ground accordingly, it is possible to select a specific location in the matrix. If a chip is connected to this location, the chip boots up and sends its address, from now on called ID. If the MKR does not receive an ID after 50ms, it knows, that there is no chip at this location and will check the next one. After this localisation step there were challenges expected, when all chips are supplied with power again. The chips are not able to differentiate whether they are powered, because the board tries to localise them, or if they should return to normal operation. Thus all chips always send their IDs when supplied with power. The intended solution was, that the MKR ignores any data coming in for a short period after switching on all chips.

However, this solution was never implemented. The AtTinys were not able to successfully send their ID after powering up.
At first the chips were directly powered by a pin of a Arduino Nano. It was uncertain whether the power supplied was powerful or stable enough. To stabilise it, a capacitor was placed parallel to the chip. Another try was, to power the chip via a MOSFET directly from the power supply.
Another possible cause for that behaviour might have been that there were some terminal characters in the incoming data. All available methods for sending data were tried, as well as clearing the buffer after reading the first byte.

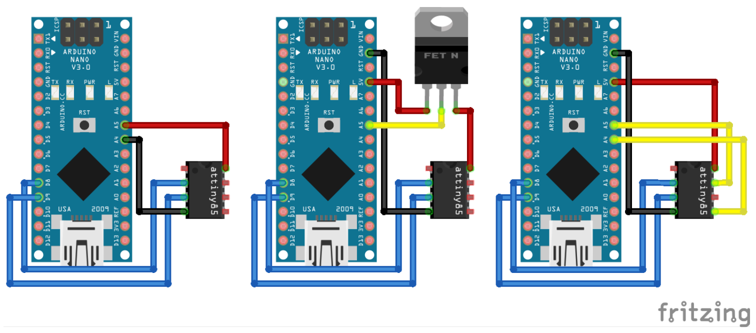

Figure 3.3: Evolution of the location test setup

As the booting process does not allow for precisely timed intervals and might cause power fluctuations this approach was discarded.
As a new approach two wires were directly connected to pins on the chip. If these wires are pulled low, the chip will send its ID without the need to boot again.
To reduce the number of pins needed, a 4011 NAND IC was included, which leaves two pins on the AtTiny free to handle its tasks.

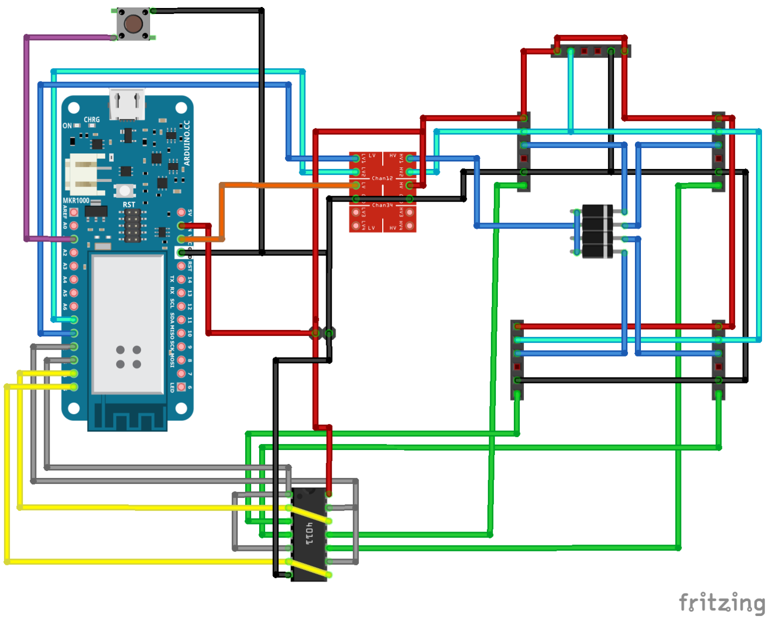

Figure 3.4: Schematic of the finished board

3.3.3 Hardware

Besides the above explained functions the board needs some additional components. Because the MKR is a 3.3V microcontroller and can not handle 5V, which the AtTinys use, the communication between both controllers is not possible without level shifting.

In addition to the level shifter (the red rectangular in Figure 3.4) there are five sockets on the board: four of these are for connecting regular chips. Above these four there is an extra socket. This one is special, as it is not connected to any localisation line. Therefore, any chip connected with it will not be found by the localisation routine of the MKR. This socket is intended for a display chip, see Section 3.4.2.

There is also a tactile switch above the MKR. It is connected to the MKR via an internal pullup resistor. By pressing this button the communication mode is changed. For more information about these modi see the next section.

JST = Japanese Solderless Terminal (0.1in pitch)

Last there is a JST for the power connection. A micro USB port is the intended way to supply power, but if the user wants to use a different power source, he simply needs to connect it via this JST. The JST ensures, that it is not possible to connect the power source the wrong way. The boards operational voltage is 5V.

3.3.4 Software

In this section the software which is operating the board will be discussed. There are three main components. The first thing the software checks is, whether the mode change button is pressed and acts accordingly. After that the software handles any incoming data. The last part checks if there is a change in the connected chips.
While booting, the board initialises the second hardware Serial port and reserves memory for the location array, which will contain the location of connected chips. It also checks if there is a Serial connection to a PC. If there is none, it tries to find a connection for five seconds.

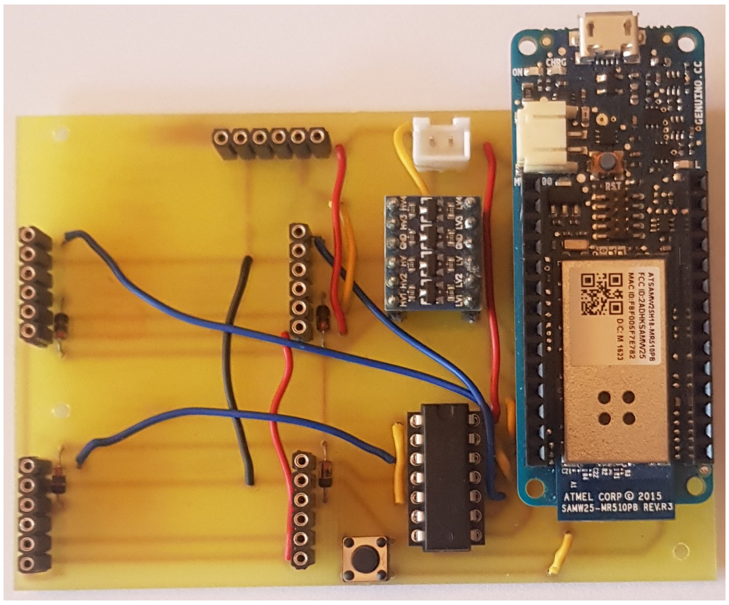
Figure 3.5: The finished board PCB

Mode Change

The first thing the MKR does is to check whether the mode change button, which is connected to pin A1, was pressed. In this case the board checks if a confidence level is reached: that means the button is pressed for 50 successive loop iterations. If the button is released within the 50 cycles, the MKR will not recognise a successful button press. This is done to eliminate any bouncing.
Once the MKR recognises a successful button press, it changes the mode variable. The mode variable is of the type character. Up to now there are only two possible modi. The character 0 represents the hard coded communication system. The other possible mode is called pass-on and is associated with the character 1.
After a successful mode change the MKR activates the location subroutine.

Communication

After handling a possible button press the MKR checks both Serial ports.

The native Serial port might be connected to a PC. If it is not connected, there will never be any data and the MKR will never enter this subroutine. If there is data, it will either be the single character t, which will prompt the MKR to jump into the localisation subroutine, or if it is not, the MKR will try to read a second character and send this to the chip with an address matching the first read byte.

After checking the native Serial port the MKR will check the second one. This is the one connected to the sockets and the chips. If there is data, the MKR expects two bytes. The first byte contains the address of the chip, which has sent the data, while the second byte is the data itself. If there is no second data byte, the MKR will simply put in the character x as data.
Having received data, the MKR checks the mode variable to determine which communication mode to use.

The simpler mode is the hardcoded one. In this mode the MKR sends the data, based on the senders ID, according to a hardcoded lookup table to the receiver.

The pass-on mode is a little more complex. After receiving data, the MKR checks in the location array, whether the position of the sender is known. If it is unknown, it will jump into the localisation subroutine to find the position. If it is still not able to find the position, it will discard the data package.
If it finds the sender's location, the board will look up the ID of the chip at the next higher position. After that, it simply sends the data to that chip. If there is no chip in the next position, it will discard the package as well.

Localisation Routine

The localisation subroutine will only be called every five seconds if it is not called by one of the two others. It has two parts. The first is responsible for finding the chips, the second sends data to a display chip, see Section 3.4.2.

The positions are arranged in a matrix form. By using this configuration the board is able to use only the number of $rows + columns$ pins, to check $rows * columns$ positions.

The first step is to pull the lines going to the corresponding position high. Which in turn will pull the location line going from the 4011 NAND to the chip low. The board waits for 50ms, before it checks, whether there is an ID message at the second Serial port. If there is a message, the MKR will store the ID in the locations array, otherwise it will place the character x, to indicate that the position is empty. Afterwards it repeats this procedure for all possible locations.

After checking all locations the subroutine will try to send the new locations array via the native Serial port to a connected computer as well as to a display chip. The data for the display chip is extended by the byte representing the mode.

3.4 Chips

The different chips are introduced in this section. All chips have the same pinout to connect to the board, therefore, the user can connect each chip to each location.
The chip, the right side up, the pins on the top, starting from the left pin: the first is the localisation pin, if that one is pulled low, the chip sends its ID to the board. The next pin is ground. After that there is an empty pin, which makes the connector not symmetrical and prevents the user from accidentally placing the chip in the wrong direction. The next two pins are the communication pins. The last pin is the voltage supply.

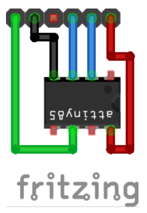

Figure 3.6: Pinout for a chip without anything connected to it

Most chips use the AtTiny85 microcontroller. The pin configuration described above leaves only three pins unused on the AtTiny. One of these three is the reset pin, which is only usable if the fuses are set by a high voltage programmer. This means that there are only two pins left for input and output. Luckily most applications do not need more.

The software on these chips is mostly the same. Each chip has its own address which is stored in a variable of type character called ID. The software contains up to three parts. The first handles the localisation request, the second checks the Serial port and the third is optional. If the chip has some kind of input, like a button for example, the third part handles that.
The localisation subroutine checks each cycle if the localisation line is pulled low. In this case, the chip sends its ID. Up to now there are chips with IDs in the range from zero to four.
After sending the ID the chip goes to sleep for 100ms.

ID	Chip
0	Either Display or LEDs
1	Gyroscope
2	Toggle or momentary switch
3	LED
4	Seven-segment-display

Table 3.1: List of all assigned IDs so far

The communication subroutine checks if there is new data. If there is data, the chip reads the first byte containing the address and immediately after that it tries to read the data byte. If there is no data to be read, the data byte is set to the character x. Afterwards the chip compares the just read address with its own ID. If they match, it knows, that the data was intended for itself and can act accordingly. Otherwise it ignores the data byte.

As the third part is optional, it is individual to the specific chip and will be discussed in the section corresponding to the chip.

3.4.1 Simple Chips

Simple chips are chips, that do not rely on a communication protocol to get their data from their sensor or to get it to the actuator. At the moment there are two chips of this category.

Button

This chip has a button connected to pin 7 of the AtTiny, see Figure 3.7.
The software of the AtTiny checks each cycle if the button is pressed. The same method to debounce the button is used as the one on the board previously.
When a button is successfully pressed, the chip sends its ID followed by the information from the button press. There are two different versions of this software. The first sends the present state of the button: if the button is pressed, the chip sends 255, and if the button is released, the chip sends a 0. The second software variant implements a toggle switch. The button press changes the state of a boolean variable which is afterwards send to the board.
Incoming data is ignored in both versions of the software.

Possible other versions of this kind of chip include one with two buttons or a potentiometer.

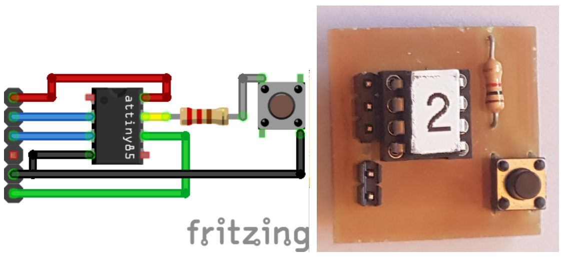

Figure 3.7: Schematic of the button chip

LED

This chip has a simple LED which is connected via a current limiting resistor to pin 5.
When data arrives at the chip, it is read and used to set the duty cycle for a PWM signal, which drives the LED.

PWM = Pulse Width Modulated

Further possible versions include the addition of another LED or the switch to an addressable LED like a WS2812b.

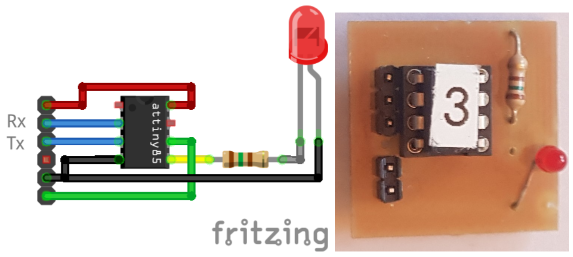

Figure 3.8: Schematic of the LED chip

Other

It is possible to design even more chip, that fall into this category. For example one could use a potentiometer or a thermistor as an input.
Other outputs might include a piezo buzzer or a small motor. With a dedicated driver it would even be possible to drive a stepper motor.
As long as it is possible to connect any input or output device with only two pins to the AtTiny, it is possible to design a chip.

3.4.2 I2C Chips

These chips are more advanced compared to those in the last category. Nearly all of these chips use the same base chip. It has two $4.7k\Omega$ pullup resistors for I2C communication. Additionally to the resistors the chip has a socket to connect to a shield, see Figure 3.9. A shield is a board, that can be plugged into the socket on the chip. It communicates with the chip via I2C.
The specific pinout: the top pin is the serial data line, below that lies the clock line. After that there is ground and the last one is 5V.
To prevent inserting a shield the wrong way, the user only has to keep in mind, that the shields are supposed to be directly above the chip itself.

The software uses the TinyWireM library by Adafruit [c] to enable the AtTiny to use the I2C interface as a master device.

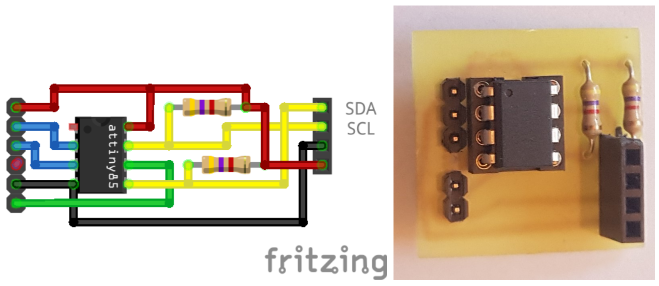

Figure 3.9: Schematic of the basic I2C chip

Seven-Segment-Display

This shield is an example, that even though there are only two pins still available at the AtTiny it is possible to use more. By using an MCP23017 IO expander IC it is possible to drive two seven-segment-displays.
While powering up the MCP is set up by directly accessing its registers. It is configured that all its 16 pins, divided into two ports, are outputs.
When receiving data the AtTiny parses it. Afterwards the byte, which is necessary to display the correct number cor-

responding to the data, is looked up. These bytes are then sent to the corresponding registers of the MCP.

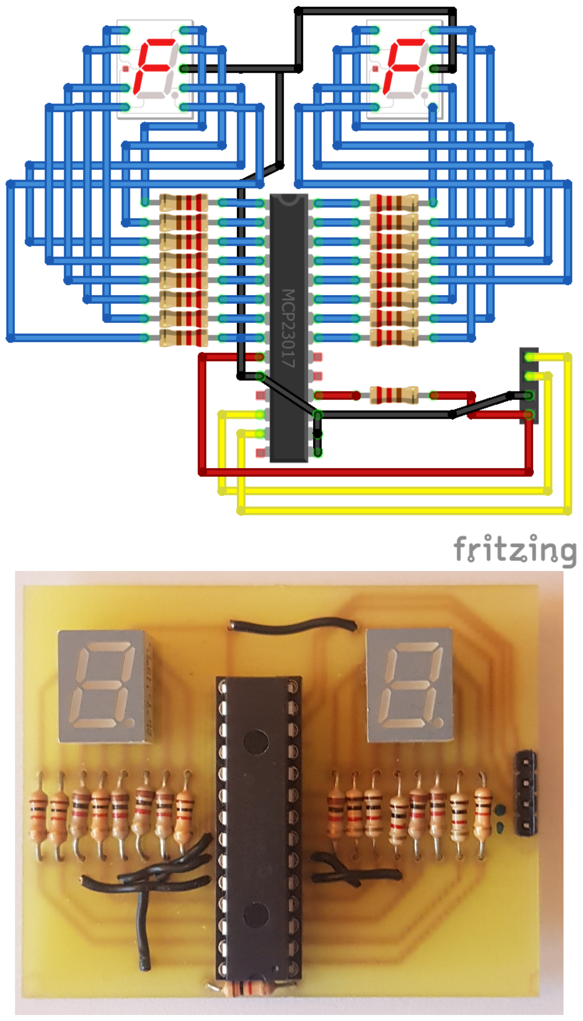

Figure 3.10: Schematic of the seven-segment-display shield

Gyroscope

A GY-521 breakout board is used for the gyroscope. It has a MPU-6050 chip, which contains among other things a 3-axis gyroscope, a 3-axis accelerometer and a digital motion processor (DMP). The GY-521 can be plugged directly into the chip as its pinout is the same as the one on the chip.

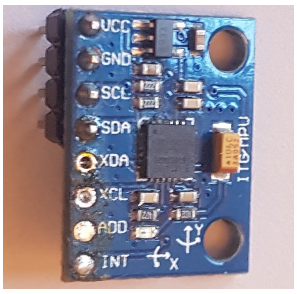

Figure 3.11: The GY-521 breakout board

After operating the gyroscope with an Arduino Nano successfully, a transfer to the AtTiny was attempted. It did not work, at least not while simultaneously accessing the AtTiny with a SoftwareSerial connection. The most plausible explanation for this is, that both, the SoftwareSerial and the TinyWire library, utilise timer1 of the AtTiny. Another possible explanation is that the DMP has not been set up correctly by the AtTiny. Determining, which is causing that behaviour, is beyond the scope of this thesis. Because of that, only the gyroscope chip uses an Arduino Nano instead of the AtTiny, which all the other chips use.

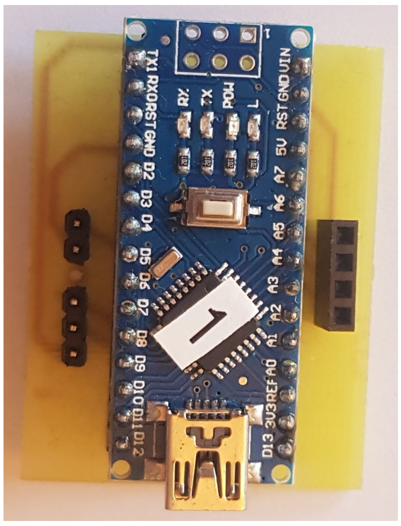

Figure 3.12: The Arduino Nano I2C chip

The software fetches data from the MPU in each cycle. To get the angle at which the chip is currently held, the software applies a gyroscope scaler, which can be found in the

datasheet. Then it calculates the accelerometer angle, integrates it to get the gyroscope angle and applies a filter to it. The resulting angle is then sent about two times per second to the board. To be exact, it is sent every 437 microseconds, so repetition of collisions can be avoided.
It is a very simple implementation of an angle calculation. The software does not compensate for anything like gyrodrift or offset errors.

Display

This shield, as well as the next one, were intended to be used in the extra socket to display which chips are currently connected to the board. However, they work in any socket and with minor alteration to the software can display other information as well.

Similar to the gyroscope the hardware for the display shield consists only of a single 0.96″ OLED display. As the pinout of this particular display is in the exact opposite order, an adapter PCB was made to keep it consistent with the other shields.

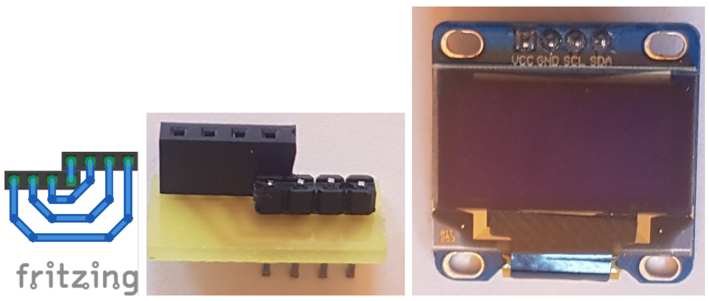

Figure 3.13: Schematic of the OLED Display adapter

The software uses the TinyWireM and the TinyOzOLED library from user SensorsIot on Github to drive the display. As the display used has a different address than most of the others the address had to be changed in the library.
After initialising the display the AtTiny shows the static information like the locations and texts. At the end of the location subroutine the board sends the IDs of all chips found and the information of which mode the board is currently

in to the display. These IDs are converted from characters to integers and sent to the display together with the text which mode is currently active.

Individually addressable LEDs

This shield is designed, like the one above, to display the location of chips connected to the board.

It uses five WS2812b individually addressable LEDs. The four at the top (see Figure 3.14) are for indicating which chip is connected to which socket, the one below is to show which mode the board is currently in. The hardware consists of two electrolytic capacitors to reduce spikes in current draw and one resistor to protect the first LED, according to the best practice described on the Adafruit [b] website. To address the LEDs the Neopixel library by Adafruit [a] is used.

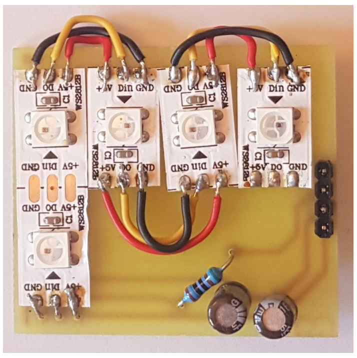

Figure 3.14: The individually addressable LED shield

Similar to the display above the incoming data is converted to an integer, which is then used to get the colour code corresponding to the ID out of a predefined array. This colour code is sent to the LED corresponding to the location.

Colour	ID	Mode
off	0	
red	1	hard coded
green	2	pass-on
blue	3	
yellow	4	

Table 3.2: Colours, the corresponding ID and mode

3.4.3 Challenges

While writing the code for the I2C chips, there were some major obstacles with the display as well as the individually addressable LEDs.

Although the display does not require a very complex program it did not work as expected.
While trying to display the IDs the first position always showed a -72. An Arduino Nano was set up to listen to the Serial communication with the chip, see Figure 3.15. However, that did not provide any useful information, as it only confirmed that the chip received the right data.

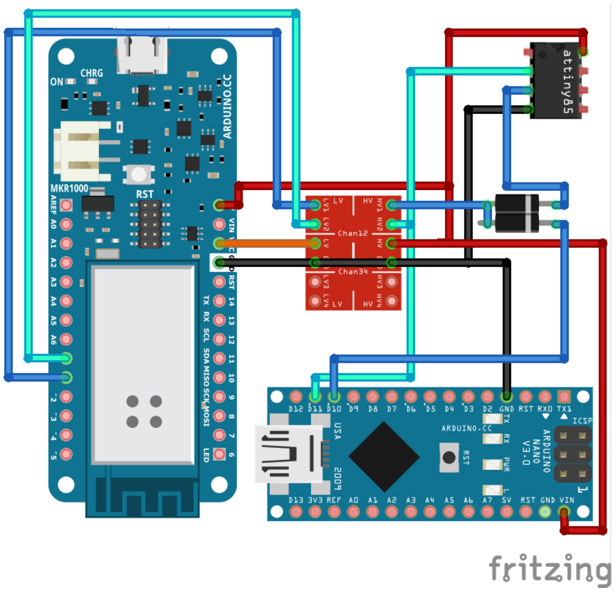

Figure 3.15: Setup to test whether communication works

After programming the chip to send back all data before displaying it, it became clear that the information received by the chip was corrupted somewhere between the socket on the board and the part of the software that displayed it. By not connecting the chip in the extra socket but in one of the four standard ones it was possible to check if the wiring was correct. As the chip functioned flawlessly at this position, it was clear that there was a wiring problem. The only difference from the standard to the extra socket was that at the extra socket the localisation line was not connected.
After connecting the localisation line of the extra socket to 5V the chip worked at that location as well.

Similar to the display above the individually addressable LEDs did not initially work as intended. After receiving data the shield displayed the right colours at the right places, but after less than one second the colours changed. The colours which were then displayed were colours not among the preconfigured ones. These wrong colours stayed until the chip got the next message from the board. After swapping the chips at the board the first displayed colours were also changed correctly. But again after less than a second the colours changed to the previously wrong ones. Altering the program the same way as the software of the display shield the chip now sent the data, which it tried to display on the LEDs, to the board.
It turned out that the chip received another data set after the correct one. The monitoring of the data transmission proved that there was only the correct data on the bus.
The solution was to check, before sending data to the LEDs, whether the data is a known ID or not.

Chapter 4

Evaluation

The goal of this thesis was to create a user-friendly wearable networking system. This chapter is going to evaluate which of these three aspects were sufficiently met and which need to be improved.

4.1 User-friendliness

There are many aspects to consider whether a system is user-friendly or not.

Does the system require any additional parts that are not included?
The only thing, the user has to provide, is a power source. He is free to choose what kind of power supply he wants to use. He can use the provided micro USB port to power it from any USB power supply, or he can disconnect this port form the JST and connect any other power supply there, as long as it is capable of providing a stable 5V.

If there are connectors, is it possible to connect them the wrong way and destroy any electrical components?
Most connectors in the system are designed in a way that it is physically impossible to connect them the wrong way. The connectors between board and chips are asymmetrical,

so there is no way to force them into the socket the wrong way. For the power connector the board uses a JST with registration teeth which prohibit wrong insertion.
The only parts that can be connected in a way not intended are the shields for the I2C chips. However, they are designed in a way, that the user only has to remember, that the shield has to be directly above the chip. If he forgets this, the power pins are located in such a way that it is still impossible to connect ground to 5V and vice versa.

Does it matter which chip is connected to which location? And can the location be changed at any moment?
This depends on the mode the user chooses. After the board has powered up it is in the hardcoded mode. In this mode it does not matter where a chip is connected and the system checks, only based on the senders ID, to which chip the message is supposed to go. That means, that it is possible to connect and disconnect any chip at any given time, because even if the addressed chip is not there, the message is sent anyway.
In the second mode the user has to keep in mind that the chips, which he wants to communicate, have to be directly next to one another. Although this is more complex it offers more options. New setups can be created without the need to change the software. In this mode chips can also be connected and disconnected at any given moment. Before sending the data the system always checks the sender's location and if there is a chip in the next position to which the data will be addressed.
Although the above introduced system works there are some aspects to be improved, see Section 5.2 for ideas.

Does the system provide information about which chip is connected to which location?
The board has an extra socket to connect a display chip. Up to now there are two kinds of displays: a 0.96" OLED display which can show the positions, the connected chips and which mode the board is in. The other option is the WS2812b individually addressable LED shield. It displays, which chip is connected to which position by assigning each ID its own colour and by displaying that colour at the LED corresponding to its location. The mode is displayed in the same way.

If the user wants, he can create other methods for displaying the information.

Is it possible to create new chips?
The user can create as many chips as he needs. He has to use the same pinout as described in the chip section. He has to keep the communication format, first sending the chips ID followed by a data byte. The chip has to be able to handle the information sent by the board, this includes the capability to handle the longer data stream containing the location information for the displays. But otherwise the user can create any chip he wants or needs. He is not even restricted to the use of the AtTiny85.

4.2 Wearability

The system needs to be small enough to be comfortable to wear, it needs to be self contained and to have its own power source. All these aspects are met successfully, however, there is still room for improvements, see Section 5.2. At the moment the board has a maximum length of 9.5cm at a width of 6.5cm, which makes it smaller than most modern smart-phones. Even if it is in a case, the user is able to carry it comfortably attached to a belt or in a pocket. The chips use the AtTiny85 to keep them as small as possible.

The system does not require any external hardware to operate except for the power supply. While booting, the board checks whether there is a Serial connection to a computer, however, that is only to provide the user with additional information: such as which chip sent what kind of data. If there is no connection, the system will still operate perfectly without one.

The system uses a micro USB socket as a power connector to provide it with power. Using a powerbank is the simplest way. These are available in many different sizes and shapes, so the user can choose the one that he prefers.

4.3 Networking

The most interesting part of this work was the networking. As described in Section 3.2 the initial idea was to combine both communication lines into a single one. This is possible as long as the system uses an AVR chip. By utilising the SoftwareSerialWithHalfDuplex library it is feasible to have multiple chips capable of communicating with one another over the same wire. The highest tested number of successfully communicating microcontrollers was five. This is sufficient to create the desired networking.

A change was made to an ARM based processor because of the reasons described in Section 3.3.1. As there is no equivalent library for this processor architecture the whole system needed to be reworked. By using signal diodes it is possible to create virtual communication channels which enable one chip to talk to the board without noticing that there are multiple other chips connected. After adjusting the address the board simply broadcasts all messages it received to all connected chips. Thus it appears to the chips that they communicate with each other, although they do not know for which chip their message is or where that chip is located.

If the redirecting is hardcoded, the board will not have to do much. The board simply checks the senders ID and looks up the correct receiver before broadcasting the data. Only the right receiver gets the data as all chips discard messages that do not contain their ID. The board does this regardless whether or not the correct chip is connected.

The more interesting part is redirecting based on the location of the sender. For this the board needs to know where a sender is located. The board checks whether the location of the sender is known. If it is not, the board tries to find that chip. After successfully locating the chip the board checks whether there is a chip at the next position. If there is one, the data is sent to it. Otherwise the data is discarded.

Both modi implement valid networking strategies. The last task is to determine if there are not too many collisions for the system to operate effectively. As collisions can only

happen if two chips try to send data at the same time, it is necessary to look at the rate at which the chips send data. The chip that sends most frequently data is the gyroscope chip. It sends data about two times per second. The line is occupied for only 4.17 milliseconds if a boud rate of 9600 is used and the chip sends two times 20 bit $(2x(1startbit + 8databits + 1stopbit))$ per second. This means a collision can hardly happen. Even if one occurs, the damage is most surely corrected by the next data package sent by the gyroscope.
However, this might be improved in the future by implementing a system that either is able to detect if a collision happened or that is able to prevent them from occurring.

Chapter 5

Summary and future work

5.1 Summary and contributions

This thesis dealt with two main goals. Primary to create a networking system and secondly to provide a user-friendly experience. The networking is done via a Serial port. By combining the transmitter lines of the connected chips with the use of 1N4148 signal diodes it is possible to have communication between two components even if there are more. The board offers two modi in which the networking is done. In the first mode the address corresponding to an incoming message is directly written into the software of the board. The second mode offers more flexibility. It redirects messages based on the location of the sender. The system is able to localise where a certain chip is connected.
Each aspect is designed with user-friendliness in mind. Besides the possibility to have message-passing, either hard-coded or dependant on the position, the user is able to connect any chip at any location. He can connect or disconnect chips at any time and the system will adapt to it. The only thing the user has to think about is, if he uses pass-on mode, that chips which should communicate have to be located next to each other. The sockets on the board are designed

in a way that it is impossible to connect a chip the wrong way. The whole board runs on 5V, it has a micro USB connector to power it. But if the user wants, he can connect any power supply to the board via a JST connector, as long as it provides a stable 5V. The user can create any chip he needs because there is a blank chip template, see Figure 3.6. But he can also adapt more complex hardware to utilize the standard I2C chip. The user has the option to connect a display or individually addressable LEDs at the extra socket to show which chips are connected at which location.

Ideas to take away from this work: it is possible to use Serial communication which is designed to be a communication protocol between two participants. But it can also be used to communicate with more than two if the communication is done via a communication hub. Using diodes it is possible to simulate direct communication channels between two members. However, depending on the rate at which data needs to be transmitted it is possible that collisions might occur.
Another important point is that all input pins must be connected either to the supply voltage or to ground. If one is left unconnected, this pin might cause havoc, even if the software does not access it. If all pins are connected the right way, there might still occur some effects which need to be handled so that the software can run reliably, see Section 3.4.3.

5.2 Future work

For the future there are still some aspects that can be improved.
The most important one would be to shrink the footprint of all parts. Except from the MKR and the one Arduino Nano, which are already SMD and cannot easily be shrunk any further, all parts are through-hole components. Replacing them with SMD ones the size can be reduced significantly. This will increase the comfort for the user while wearing the system.

SMD = surface mounted device

Afterwards the next two points are of equal importance.

The addition of a third mode to the board: this one would be a mix of the already existing two. This means that the user has the possibility to hardcode the communication between certain chips, while leaving some communication to be determined by the location on the board. This would be useful in the case that one chip is intended to work with another specific one, like the seven-segment-display is intended to work with the gyroscope. The activation of the first chip (e.g. the gyroscope) can be determined by any chip (e.g. the button) which is placed in front of it. As shown above it would be possible to use the button chip with either the toggle or the momentary switch program. Another possibility: a chip with a potentiometer can be used to start the communication of the gyroscope if a certain threshold value is reached.

Additionally to the third mode, the hardcoded mode can be made a bit more user-friendly by changing the way the hardcoding is done. Up to now the user has to change the hardcoded list within the code. The user should be able to create a simple text document which is stored on a SD card. While booting, the board checks the text file and adjusts the definitions how incoming data is redirected to the chips. This could be done in a way that any connection, that is not specified by the user, is treated as if the board is in pass-one mode.

The system can even be more expanded if more chips are designed. As mentioned in Section 3.4 there is nearly no limit to the number of possible chips. Especially the creation of a potentiometer chip would be advantageous. It would allow the user, for example, to control the rate at which the gyroscope chip is sending data or to define the brightness of a LED.
Besides the potentiometer, a SD card chip would be extremely useful: either to log certain data or to read data from a file and use it to influence the behaviour of the system.
For this purpose a totally new type of chip needs to be created as the AtTiny85 does not have a sufficient number of pins for the SPI communication on which the SD card relies. It might be possible to create a SPI chip using the physically bigger brother of the AtTiny85, the AtTiny84. If this proves

to be impossible, one could still use an Arduino Nano to create such a chip.

Last the communication protocol should be changed significantly. Up to now it is only possible to send a single byte of information, aside from the case where the board sends the localisation data. The protocol should be changed in a way that, after the chip sent its ID, it has to declare the number of bytes that it intends to send. Thus it would be possible to have chips that are capable to send and receive more complex data. This would be advantageous for a chip like the gyroscope, as it is limited to just one axis of rotation at the moment.

When the above mentioned points are changed or implemented one might start to think about a method to prevent collisions or at least to detect whether a collision has occurred.

The above described system offers an easy to use and quite capable entry point into wearable electronics. Its modularity offers the user the possibility to experiment with different setups and configurations without previous knowledge in electronics. As the system is easily expandable, by creating new modules, it is excellently suited for educational purposes on nearly every skill level.

Appendix A

Git

All software files can be found here: Software[1]

The hardware files are at this address: Hardware[2]

[1]https://github.com/TMei90/BachelerSoftware
[2]https://github.com/TMei90/BachelorHardware

Appendix B

Version History

Software versions after the switch to the Arduino MKR1000 in chronological order.

- MKR1000
  - MKR1000ExtraHardwareSerial: Implementation of a second hardware Serial port — Successful
  - MKR1000Substring: Implementing Serial communication with multiple participants — Successful
  - MKR1000LocationTest: Implementation of the localisation subroutine— Successful
  - MKR1000LocationTestV2: Expanding V1 by adding hardcoded message passing — Successful
  - MKR1000LocationTestV3: Implementation of pass on mode — Successful
  - MKR1000LocationTestV4: Code cleaning, implementation of mode change — Successful
- Nano
  - NanoSoftwareSerial: Implementation of a software Serial port — Successful

  - NanoSoftwareSerialEcho: Echoing Serial data back to the sender — Successful
  - NanoSSIDResponse: Assigning IDs, responding only if own ID is recognised — Successful
  - NanoLocationTest: Implementing the localisation subroutine — Successful
  - NanoIOExpander: Implementation of an IO expander using a library — Successful
  - NanoIOExpanderV2: Implementing the IO expander without a library — Successful
  - NanoGyro: Implementing an I2C gyroscope using a library — Successful
  - NanoGyroV2: Implementing an I2C gyroscope without a library — Successful
  - NanoGyroV3: Adding communication to the GyroV2 — Successful

- AtTiny85 (most versions are direct adaptations of the Nanos implementation)
  - AtTinySSIDResponse: Adapting NanoSSIDResponse for the AtTiny — Successful
  - AtTinyLocationTest: Adapting NanoLocationTest for the AtTiny — Successful
  - AtTinyIOExpander: Adapting NanoIOExpanderV2 for the AtTiny — Successful
  - AtTinySerialDisplay: Expanding AtTinyIOExpander with Serial communication — Successful
  - AtTinyGyro: Adapting NanoGyroV2 for the AtTiny — not Successful
  - AtTinyButton: Implementing a button, after pressing a 255 is sent, after releasing the button a 0 is sent — Successful
  - AtTinyLED: Implementing a LED, reading a 255 turns the LED on, reading a 0 off — Successful
  - AtTinyToggle: Changing the button to a toggle switch — Successful
  - AtTinyNeoPixelTest: Implementing WS2812b LEDs to display chip positions — Successful

- AtTinyOLEDTest: Implementing a 0,96″ OLED display, displaying chip positions — Successful
- AtTinyNeoPixelV2: Adding network functions to the AtTinyNeoPixel — Successful
- AtTinyOLEDV2: Adding network functions to the AtTinyOLEDTest — Successful
- AtTinyOLEDV3: Adding mode change — Successful
- AtTinyNeoPixelV3: Adding mode change — Successful

Bibliography

Adafruit. Neopixel, a. URL `https://github.com/adafruit/Adafruit_NeoPixel`.

Adafruit. Basic connections, b. URL `https://learn.adafruit.com/adafruit-neopixel-uberguide/basic-connections`.

Adafruit. Tinywirem, c. URL `https://github.com/adafruit/TinyWireM`.

Anke Brocker, Simon Voelker, Tony Zelun Zhang, Mathis Müller, and Jan Borchers. Flowboard: A visual flow-based programming environment for embedded coding. In *to appear: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, New York, NY, USA, May 2019. ACM. ISBN 978-1-4503-5971-9/19/05. doi: 10.1145/3290607.3313247. URL `https://doi.org/10.1145/3290607.3313247`.

Kinematics. Tinkerbots robotic building kit. URL `https://www.indiegogo.com/projects/tinkerbots-robotic-building-kit-toys#`.

nickstedman. Softwareserialwithhalfduplex. URL `https://github.com/nickstedman/SoftwareSerialWithHalfDuplex`.

SensorsIot. Tinyozled. URL `https://github.com/SensorsIot/TinyOzOled`.

SparkFun. qwiic. URL `https://www.sparkfun.com/qwiic#connections`.

Ryoichi Watanabe, Yuichi Itoh, Michihiro Kawai, Yoshifumi Kitamura, Fumio Kishino, and Hideo Kikuchi.

Implementation of activecube as an intuitive 3d computer interface. In *International Symposium on Smart Graphics*, pages 43–53. Springer, 2004. URL https://link.springer.com/chapter/10.1007/978-3-540-24678-7_5#enumeration.

Jamie Zigelbaum, Michael S Horn, Orit Shaer, and Robert JK Jacob. The tangible video editor: collaborative video editing with active tokens. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 43–46. ACM, 2007. URL https://static1.squarespace.com/static/4f57841c24ac1bb6d947d820/t/50f828a6e4b02a75e89d8c95/1358440614743/TangibleVideoEditor_TEI2007.pdf.

Oren Zuckerman. *Flowness+ FlowBlocks: uncovering the dynamics of everyday life through playful modeling*. PhD thesis, Massachusetts Institute of Technology, 2007. URL https://www.media.mit.edu/publications/flowness-flowblocks-uncovering-the-dynamics-of-everyda

Typeset May 14, 2019