Bluetooth Communication System for Smart Textiles

Charlotte Feryn, Carlos Canas Carrillo

Promotoren: prof. dr. ir. Jan Vanfleteren, prof. dr. ir. Ingrid Moerman
Begeleider: Thomas Loeher, TUBerlin

Masterproef ingediend tot het behalen van de academische graad van
Master in de ingenieurswetenschappen: elektrotechniek

Vakgroep Elektronica en informatiesystemen
Voorzitter: prof. dr. ir. Jan Van Campenhout

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. Daniel De Zutter

Faculteit Ingenieurswetenschappen
Academiejaar 2009-2010
Bluetooth Communication System for Smart Textiles

Charlotte Feryn, Carlos Canas Carrillo

Promotoren: prof. dr. ir. Jan Vanfleteren, prof. dr. ir. Ingrid Moerman
Begeleider: Thomas Loeher, TUBerlin

Masterproef ingediend tot het behalen van de academische graad van
Master in de ingenieurswetenschappen: elektrotechniek

Vakgroep Elektronica en informatiesystemen
Voorzitter: prof. dr. ir. Jan Van Campenhout

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. Daniel De Zutter

Faculteit Ingenieurswetenschappen
Academiejaar 2009-2010
Permission for use of content

“The authors give permission to make this master dissertation available for consultation
and to copy parts of this master dissertation for personal use.
In the case of any other use, the limitations of the copyright have to be respected, in
particular with regard to the obligation to state expressly the source when quoting
results from this master dissertation.”

Berlin, August 2010

Carlos Cañas

Charlotte Feryn
This thesis could not have taken place before the Erasmus era. Therefore, we would like to seize the opportunity to thank the University of Ghent and the Technical University of Berlin for giving us the opportunity to experience a year abroad. In particular we thank Prof. Dr. Ir. Roel Baets and Prof. Dr. Ir. Jean-Pierre Martens for their assistance during this year.

We say thanks to Prof. Dr. Ir. Jan Vanfleteren for contacting us with the people from the Fraunhofer Institute for Reliability and Microintegration, and to Prof. Dr. Ir. Jan Vanfleteren and Prof. Dr. Ir. Ingrid Moerman for acting as our Belgian promotors.

Next, we thank the researchers at the Fraunhofer Institute, for giving us a warm welcome, putting our names on the office door, and offering us support when we encountered a problem.

However, we are especially grateful for the help René Vieroth has given us, he always made time to sit down whenever we had a question.

Last but not least, we would like to thank our family and friends. Even though we were miles away, they supported and encouraged us to bring this thesis to a good end. So thank you Arnaud, Jens, Alejandro, Elise, Liesbeth and our parents Carine, Hans, Linet and Jose Carlos.

We could not have done this without you.

August, 2010

Carlos Cañas
Charlotte Feryn
Overview

In 2008, an interactive dress was designed at the IZM and TUB in Berlin. This dress, called K-Light, is equipped with 32 LEDs that shine in an animation mode depending on the movements of the dress. This thesis describes how the initial static system can be made dynamic. The original system is extended with an extra microcontroller and a Bluetooth module. A mobile phone can be used to make a connection to this module. To enable the communication between the two devices, a special protocol is developed that uses framed data as communication packages.

The extension has a minimal size, which does not only implies a maximal comfort, it also allows the system to be easily integrated into other systems.

The developed system enables the user to adapt certain system parameters such as the LEDs’ brightness, the sensor’s sensitivity, or the animation mode. The user can ask for K-Light system information, which is then displayed on the mobile’s screen.

In the first chapter a short introduction on smart textiles and the original K-Light system is given. Then follows a more detailed description of the SCB and Bluetooth technology. In the third chapter, a description of the developed hardware system is given, while in the forth chapter the mobile application is discussed. The conclusion lists the capabilities of the developed system and the encountered problems during the design process.

Key Words

Bluetooth, Java ME, Stretchable Circuit Board,
wearable electronics, wireless communication
Bluetooth Communication System for Smart Textiles

Carlos Cañas, Charlotte Feryn
Supervisors: Jan Vanfleteren, Ingrid Moerman

Abstract — In 2008, an interactive dress was designed at the IZM and TUB in Berlin. This dress, called K-Light is equipped with 32 LEDs that shine in an animation mode depending on the movements of the dress. This article describes how the initial static system can be made dynamic. The original system is extended with an extra microcontroller and a Bluetooth module. The extension has a minimal size, which does not only imply a maximal comfort, it also allows the system to be easily integrated into other systems. A mobile phone can be used to make a connection to the developed module. To enable the communication between the two devices, a special protocol is developed that uses framed data as communication packages.

Keywords — Bluetooth, Java ME, Stretchable Circuit Board, wearable electronics, wireless communication

I. INTRODUCTION

Smart Textiles are hotter than ever. In the sport, medical and even fashion industry, engineers working integrating electronics into textiles are beloved employees. From 2006 until 2010 the European Commission funded the Stretchable Electronics for Large Area applications (STELLA) [1] project. In the frame of this project Belgian, French, Dutch and German researchers pulled together in their quest for a special technology that does not only implies a maximal comfort, it also allows the system to be easily integrated into other systems. A mobile phone can be used to make a connection to the developed module. To enable the communication between the two devices, a special protocol is developed that uses framed data as communication packages.

II. HARDWARE DESIGN

At first instance, the wireless technology for the remote control has to be chosen. Due to its wide spread presence in portable devices, the Bluetooth technology is the best choice [4]. In July 2010 the Bluetooth SIG also announced a Bluetooth v.4, which is a new low power implementation [5]. The chosen Bluetooth module is the LMX9838 from National Semiconductor [6]. This module has an integrated antenna. Moreover, it is the smallest and cheapest one offering the RFCOMM protocol and supporting the SPP profile. Size is the most important restriction, because the smaller the SCB, the more comfortable to wear it. Price is also important as the developed system is intended to compete with commercial products.

When the Bluetooth module is selected, a microcontroller is chosen to regulate the dataflow between the LMX9838 and the K-Light system. An ATMega644P from Atmel [7] is selected, a microcontroller that implements LED animation. As the K-Light system uses the same microcontroller, the development of the Bluetooth communication firmware is achieved faster.

Philips’ I²C [8] is selected to implement the communication bus between the already existing system and its expansion. The original system already implemented the bus, and using the same bus implies an easy routing of the expansion board without making major adjustments to the existing K-Light design and code.

A communication protocol that defines what data has to be exchanged in which manner has to be developed. Moreover, the protocol implements a way of software flow control as it ensures that no data gets lost when one of the components is too busy. The protocol uses framed data as communication packages. The frame header contains information about which action is to be performed and the total length of the package. The frame is shown in figure 2.

![Fig. 1: Scheme of the end design](image-url)
Next, a kind of error management is implemented in the protocol. When specific errors occur in the system, they are written in the system info, and are displayed on the mobile phone when the user asks for it.

For the physical design, first a prototype board is built. This board allows easy debugging and reliable testing. It offers headers to monitor the communication between the microcontroller and the Bluetooth module, as well as an UART and an I²C interface. The board can be programmed with ISP and JTAG interfaces.

The final board is built once the total system is designed. This board has a minimal size. It has pads for ISP programming, for I²C communication and for one I/O pin. All the pads, except those for ISP programming are found at the bottom of the board. For reliability reasons, every signal covenants with more than one pad. These pads are also used to attach the board on the SCB in order to connect it to the existing design and are placed symmetrically to enhance the mechanical stability. Figure 3 shows the result.

![Figure 3: Picture of the final board.](image)

Some problems were encountered when placing the components on the PCB. Tests pointed out that the Bluetooth module is sensitive for temperatures above 200°C. To solve this, other soldering pastes besides SnAgCu were tested. Due to granularity issues, SnPb was the optimal choice available.

III. SOFTWARE DESIGN

The software application for the remote control is written in Java ME [9]. This way the software is made as portable as possible. In order to run the application, the end user needs a mobile phone that supports the SPP Bluetooth profile and the Java ME libraries [10].

The main function of the program is to change the LEDs’ brightness, the sensor’s sensibility, the lightning pattern and to display the system information. The ability of the program to change the light mode comes in handy for demonstration purposes. The application can set all LEDs on or off, or the LEDs can be lighted in the animation mode. The system information contains the system version, communication status and potential errors.

The program also remembers the address of the remote device over multiple sessions, which makes future connection establishments to the device much faster. Another feature of the application is that it tries to reconnect immediately after detection of connection loss. If this is not possible, the user gets an error message and is asked whether he wants to perform another device search.

IV. CONCLUSION

The K-Light system was designed to illustrate how the SCB technology can be used in everyday applications. This work describes an extension for the original system that makes the system reachable via Bluetooth technology. Therefore, an application for mobile phones is created. The application gives the end user access to some of the parameters of the LED animation. It is written in Java ME, which implies that it is available for the majority of contemporary mobile devices.

A lot of attention was paid to make the extension module as small as possible. This does not only improve the wearer’s comfort, it also allows an easy integration in other, already existing smart textile systems.

As recent Bluetooth modules combine a programmable microcontroller with the Bluetooth technology, even a smaller size for the communication module can be achieved, and the Bluetooth SIG recently announced a new, low power version, the future for Bluetooth enabled smart textiles looks promising.

REFERENCES

Bluetooth Communicatie Systeem voor Smart Textiles

Nederlandstalige samenvatting

Carlos Cañas Carrillo, Charlotte Feryn

1. Inleiding

Deze thesis is voortgevloeid uit een samenwerking tussen onderzoekers aan de Universiteit Gent en de Technische Universiteit in Berlijn. In het kader van het Europees project STELLA, zochten ze tussen 2006 en 2010, samen met verscheidene andere onderzoekers uit o.a. Nederland, Duitsland en België naar een oplossing voor zogeheten Large Area Applications.[1,2,3] De onderzoekers aan de TUB werkten in deze zoektocht nauw samen met de mensen van het Fraunhofer Institut für Zuverlässigkeit und Mikrointegration (IZM), eveneens in Berlijn. Samen ontwikkelden ze een nieuwe technologie, Stretchable Circuit Board (SCB) genaamd.[3]

Om deze nieuw ontwikkelde technologie te demonstreren, ontwierpen de onderzoekers van het IZM en TUB, in samenwerking met een studente mode aan de Hochschule für Technik und Wirtschaft in Berlin (HTW), Mareike Michel het K-Light systeem.[4] Dit is een kleed uitgerust met een versnellingsmeter, 32 LEDs en een kleine batterij, geïntegreerd in de riem van het kleed. De lichtjes zijn her en der aan de linkerkant van het kleed geplaatst zoals figuur 1 illustreert. Elk van de lichtjes kan afzonderlijk aangestuurd worden en ze lichten op volgens een geïmplementeerde animatiemode. Afhankelijk van hoe snel de vrouw die het kleed draagt zich voortbeweegt, zullen de lichtjes zwakker of feller fonkelen.

Een nadeel van dit ontwerp is, dat alles hard geprogrammeerd is, met andere woorden, het K-Light systeem is een statisch systeem. Dit betekent dat de maximale helderheid van de lichtjes en de gevoeligheid van de sensor niet meer kunnen aangepast worden na encapsulatie van het systeem. Vandaar de bedoeling van dit project om een communicatie systeem te ontwikkelen, dat met een GSM als afstandsbediening de parameters van het K-Light systeem toch kan aanpassen. Het uiteindelijke systeem kan gezien worden in figuur 2.

vii
Figure 1: K-Light, met de lichtjes duidelijk zichtbaar.
2. Hardware Configuratie

Het doel van dit project is het reeds bestaande systeem uit te rusten met een extra module zodat communicatie mogelijk is. Het reeds bestaande K-Light systeem wordt uitgebreid met een Bluetooth module, die bestuurd kan worden m.b.v. een mobiele telefoon. Deze doet dan dienst als afstandsbediening.

Bij het ontwerp van de extra module aan de K-Light zijde, wordt met twee zaken rekening gehouden. Allereerst ligt de focus op de draagbaarheid van het ontwerp, wat specifiek wil zeggen dat het gehele ontwerp zo klein mogelijk moet zijn, dat het uiteindelijke ontwerp geen scherpe randen heeft, etc. Enkel zo kan het draagcomfort van de applicatie gegarandeerd worden.

Vervolgens moet ook in het achterhoofd gehouden worden dat het om een uitbreiding van een bestaand systeem gaat. Aan het originele systeem mag dus in principe niets veranderd worden. Dit levert spijtig genoeg problemen en beperkingen op in verband met interne communicatieprotocollen en bedrading van het eigen ontwerp.

In wat volgt wordt achtereenvolgens dieper ingegaan op de selectieprocedure van de verschillende componenten, op de geïmplementeerde code, om te eindigen met het fysische ontwerp van de verschillende borden.

2.1 Selectieprocedure

Een belangrijke stap in het proces om een hardware systeem te bouwen, is de selectie van de componenten. Bij het ontwerp van dit systeem werd steeds voor ogen gehouden dat het resultaat draagbaar moet zijn, met een laag verbruik. Dit wil zeggen dat de componenten zowel compact als zuinig moeten zijn. Spijtig genoeg zijn deze perfecte oplossingen niet
altijd verkrijgbaar, en moeten verscheidene compromissen gesloten worden om een goede selectie te maken.

De belangrijkste bouwsteen van het hardware ontwerp is de Bluetoothmodule. De beschikbare Bluetoothmodules verschillen doorgaans in de geïmplementeerde Bluetoothprofielen en -protocollen, de aanwezigheid van een antenne, de communicatie interface en de module grootte.

Er werd voor de LMX9838 gekozen, wegens de geïntegreerde antenne, zijn kleine volume en relatief lage prijs [5]. Deze module was ook reeds aanwezig in het IZM, Berlijn, wat de ‘levertijd’ vanzelfsprekend gevoelig verlaagt.

Om data te verwerken en de communicatie met het K-Light systeem op punt te stellen, wordt een microcontroller gebruikt. Aangezien het IZM over een verzameling AVR microcontrollers beschikt, wordt een AVR controller ATMega644P gekozen. Deze microcontroller heeft een flashgeheugen van 64kByte, en een SRAM van 2kByte, twee USART en één I2C interface [6]. Een betere flow control wordt bereikt met behulp van grotere buffers, die bijgevolg ook meer geheugen vereisen. Daarom werd de ATMega644P verkozen boven de andere beschikbare AVR microcontrollers, wegens zijn groot beschikbare geheugen zonder een verhoogd verbruik.

Bovendien is in het oorspronkelijke K-Light system een controller van hetzelfde type gebruikt. Als ook voor het nieuwe communicatiesysteem voor deze microcontroller gekozen wordt, vergemakkelijkt de ontwikkeling van het algoritme aanzienlijk.

Voor de overige componenten is het belangrijkste selectie criterium de grootte. Aangezien niet alle componenten in de juiste package beschikbaar zijn in het IZM, moet echter ook rekening gehouden worden met de levertijd. Componenten die uit de Verenigde Staten van Amerika komen, lopen aan de douane doorgaans een grote vertraging op, en komen dus niet in aanmerking.

Voor de weerstanden, capaciteiten en LEDs werden standaardmaten gekozen, voor de protoversie 0603 en voor de finale versie 0402. Bij de selectie van de kristallen is ook de werkingsspanning van het systeem van belang. Is deze te laag, dan kan het zijn dat het kristal zal vibreren volgens een andere frequentie dan aangegeven in de datasheet.

x
2.2 Programma

Hoewel aan de hardware van het originele systeem niets veranderd wordt, zijn een aantal kleine veranderingen aan de software toch noodzakelijk. De code van de K-Light μCU wordt aangepast zodat hij met de BT μCU kan communiceren. Tussen de twee controllers worden frames, die eruitzien als in figuur 3 uitgewisseld. Hierbij vertelt de actionbyte of het een schrijf-, lees- of speciale opdracht betreft, terwijl de commandbyte aanduidt over welke toepassing het gaat: de parameters, de animatie mode of de systeem informatie. Er zijn twee speciale opdrachten, het pollen van de communicatiebus, en het verzenden van een acknowledgment of error.

![Figure 3: Het frame gebruikt voor de interne communicatie](image)

Als communicatiebus wordt de I²C van Philips gebruikt [7]. Deze bus is reeds aanwezig in het originele systeem, waardoor de keuze voor dezelfde bus tussen het originele systeem en het uitbreidingssysteem zorgt voor een eenvoudigere routering van het nieuwe systeem. Bovendien zullen slechts weinig extra lijnen nodig zijn, aangezien in het origineel systeem de langste datalijnen aan deze bus toe behoren, en de bus dus makkelijk toegankelijk is voor het nieuw ontwerp.

2.3 Fysiek ontwerp

Eens de code voor de microcontrollers ontwikkeld is, en alle componenten geselecteerd zijn, kan met het fysieke ontwerp begonnen worden. Dit wordt gedaan met behulp van Eagle v5.10, een software pakket van CadSoft. Om de ontwikkelde code te testen en debuggen, wordt eerst een protobord ontwikkeld. Dan pas volgt de ontwikkeling van het finale bord, waarbij speciale aandacht wordt gegeven aan de compactheid ervan.

Bij het ontwerp van het protobord is het belangrijk te onthouden welke lijnen moeten openblijven voor communicatiebussen en programma interfaces. Op deze manier kan het protobord ten volle benut worden om elke activiteit te monitoren, wat gebeurt met con-
trolelampjes, en om het debuggen te vergemakkelijken. Vooraleer aan de eigenlijke lay-out te kunnen beginnen, moeten de footprints van alle componenten aanwezig zijn. Eagle beschikt over een verzameling footprints standaardcomponenten, maar de prints van de LMX9838, de ATMega644P en de kristallen moeten nog zelf getekend worden. Dan pas kan overgegaan worden tot de plaatsing van de componenten. Hierbij worden de connecties tussen de BT$_\mu$CU en de LMX9838 zo kort mogelijk gehouden, en worden de componenten reeds zo geplaatst dat een routing mogelijk is.

Als de code succesvol is aangepast met behulp van het protobord, kan begonnen worden aan het finaal ontwerp. Zoals reeds vermeld ligt de focus van dit ontwerp praktisch uitsluitend bij de omvang van het bord. Het wordt zo klein mogelijk gemaakt, en beschikt enkel over afgeronde randen teneinde het draagcomfort te maximaliseren. Dit bord is uitgerust met vijftien pads, die symmetrisch op de onderkant van het bord zijn aangebracht. Deze pads worden gebruikt om het bord aan het kleed te bevestigen. Ze dienen echter ook als connectielijnen voor de communicatiebus en de input/output. Het uiteindelijke resultaat is te zien in figuur 4.

![Figure 4: Het finale ontwerp. Links is de bovenkant weergegeven, rechts de onderzijde met pads.](image.png)

**Figure 5:** Het bestukte protobord. De onderbreking van de voedingslijn is omcirkeld.

Om de componenten op het finale bord te kunnen solderen met een lagere temperatuur van ongeveer 150 °C, wordt ook een andere soldeerpasta, nl. een SnBi pasta, gebruikt. Deze
pasta bleek echter te korreliën, waardoor deze de fijne pads niet nauwkeurig kon bestrijken. Bijgevolg werd geopteerd voor een SbPb pasta, die fijner is van structuur, maar een temperatuur van 200°C vereist voor een goede vasthechting van de componenten. Deze pasta is echter niet toegelaten voor commerciële doeleinden, zodat in een later stadium naar een andere oplossing gezocht moet worden. In figuur 6 wordt een bestukt exemplaar getoond.

Figure 6: Het bestukte eindontwerp.

3. De GSM toepassing

Om met het uitgebreide K-Light systeem te kunnen communiceren, wordt een mobiele telefoon gebruikt. Met behulp van een zelfgeschreven toepassing kan deze telefoon als een afstandsbediening gebruikt worden. Zo kunnen bepaalde parameters van het kleed ook na encapsulatie van het systeem nog aangepast worden.

De applicatie is geschreven in Java ME, en werd ontwikkeld in de NetbeansIDE. Java ME is een hoog-niveau taal, en een applicatie geschreven in Java ME zou moeten werken onafhankelijk van het gebruikte bedrijfssysteem. Bij het testen van het programma op een Palm Pre smartphone bleek dit echter niet het geval. Terwijl de Palm Pre het Java platform wel aanbiedt, kunnen Java ME toepassingen niet uitgevoerd worden.

Aangezien telefoons die lopen op een Symbian bedrijfssysteem wereldwijd het meest verspreid zijn, wordt de applicatie voor een Symbian telefoon ontwikkeld. Nokia biedt talrijke GSM emulators aan, wat de keuze voor een Nokia GSM mee bepaalde. Meer specifiek werd een Nokia X6 gekozen, wegens zijn touchscreen en Bluetooth Serial Port Profile implementatie (SPP). Dit is vereist omdat de communicatie met de LMX9838 tot stand gebracht wordt via een SPP link.
3.1 Werking van het programma

In wat volgt wordt kort de werking van het programma, op voorwaarde dat geen problemen met de verbinding optreden, besproken.

Bij het opstarten van het programma, wordt een connectie gemaakt met hetzelfde toestel als de laatste sessie. Als de connectie is opgezet, verschijnt er op het scherm van de telefoon een bericht met welke module of welk apparaat de GSM verbonden is.

Indien het programma echter voor de eerste keer wordt opgestart, verschijnt er eerst een wachtscherm met de boodschap dat de telefoon een device discovery uitvoert. Tijdens zo’n device discovery wordt de omgeving afgezocht naar beschikbare toestellen. Deze worden opgeslagen in een lijst en later aan de gebruiker gepresenteerd. Deze kan dan het juiste apparaat selecteren, en de telefoon zet een SPP verbinding op met het gekozen apparaat.

Als de telefoon is verbonden met het K-Light systeem, wordt op de GSM het hoofdmenu weergegeven. Dit menu biedt de gebruiker drie keuzemogelijkheden, een zoek-en een exitfunctie. De zoekfunctie kan gebruikt worden om een nieuwe device discovery uit te voeren, als er met een ander systeem geconnected wil worden.


\hspace{1cm}

\begin{center}
\begin{tikzpicture}
\node (deviceDiscovery) {device discovery};
\node (availableDevices) [below of=deviceDiscovery] {available devices};
\node (establishConnection) [below of=availableDevices] {establish connection};
\node (startMidlet) [right of=establishConnection] {start MIDlet};
\node (mainMenu) [right of=startMidlet] {main menu};
\node (parameterSettings) [right of=mainMenu] {parameter settings};
\node (lightModes) [below of=parameterSettings] {light modes};
\node (systemInformation) [below of=lightModes] {system information};
\path (deviceDiscovery) edge (availableDevices);
\path (availableDevices) edge (establishConnection);
\path (establishConnection) edge (mainMenu);
\path (mainMenu) edge (startMidlet);
\path (startMidlet) edge (parameterSettings);
\path (parameterSettings) edge (lightModes);
\path (lightModes) edge (systemInformation);
\end{tikzpicture}
\end{center}

\textbf{Figure 7:} Eenvoudig overzicht van de schermen, als de applicatie voor de eerste maal wordt uitgevoerd.
3.2 Communicatie met het K-Light systeem

Alle zaken omtrent de Bluetooth connectie worden geregeld door de java klasse `BTManager`. Deze klasse implementeert de device discovery, de service discovery, en de manier waarop met de LMX9838 wordt gecommuniceerd. De volledige code is weergegeven in bijlage nr. B.4.2.

Terwijl het eerste wachtscherm wordt getoond aan de gebruiker, wordt de device discovery geïnitieerd door middel van de functie `startInquiry()`. Elke keer een apparaat ontdekt wordt, wordt de methode `deviceDiscovered()` opgeroepen. Het apparaat en zijn gebruiksvriendelijke naam worden opgeslagen, zodat na terminatie van de device discovery alle gevonden apparaten kunnen weergegeven worden.

Als de gebruiker een apparaat heeft geselecteerd om connectie mee te maken, wordt het correcte poortadres opgevraagd. Hiervoor worden de beschikbare services gescand die het apparaat aanbiedt. Als een SPP service gevonden wordt, wordt deze opgeslagen in de servicerecord. Later wordt deze record gebruikt om het adres op te halen. Tenslotte worden met behulp van dit adres achtereenvolgens de connectie, een inputstream en een outputstream opgezet.

Nu de connectie met het K-Light systeem een feit is, kunnen de streams gebruikt worden om de eigenlijke communicatie op punt te stellen. De telefoon zendt LMX9838 frames naar de LMX9838, zoals reeds weergegeven in figuur 3 in paragraaf 2.2. De eerste byte wordt gebruikt om aan te geven of de gebruiker informatie wenst op te vragen, of te schrijven. Er zijn nu dus slechts twee mogelijke acties. Over welke informatie het gaat, wordt gespecificeerd met behulp van de tweede byte. Enkel de parameters, animatie mode en systeeminformatie kunnen aangeschreven worden. Vervolgens wordt ook de lengte van het volledige pakket meegegeven. Ten slotte volgt de eigenlijke data. Het spreekt voor zich dat dit laatste veld leeg is in geval de gebruiker informatie opvraagt. Aangezien de LMX9838 nooit informatie verzendt op eigen initiatief, zien de inkomende frames er iets anders uit. De data worden nu slechts voorgegaan door één byte, die de totale lengte van het frame definieert.
3.3 Problemen met de connectie

In het voorgaande werd geen rekening gehouden met eventuele connectieproblemen. Er mag echter niet uit het oog verloren worden dat Bluetooth een draadloze connectie is, en dus een zekere 'error handler' vereist. In figuur 8 wordt weergegeven welke acties ondernomen worden in geval van verbindingverlies.

![Diagram](image)

**Figure 8:** Ondernomen acties in geval van verbindingverlies.

Als de applicatie minstens éénmaal is uitgevoerd, wordt de volgende keer automatisch verbinding gemaakt met het vorige geconnecteerde apparaat. Als dit apparaat echter niet beschikbaar is, omdat zijn Bluetooth module bijv. nog is uitgeschakeld, of als het apparaat zich buiten bereik bevindt dan wordt een verbinding opgesteld met een onbereikbaar adres en zal de gebruiker een errorscherm te zien krijgen. Vervolgens wordt een device discovery uitgevoerd, en krijgt de gebruiker de beschikbare apparaten te zien.

Indien de mobiele telefoon en het K-Light systeem reeds verbonden zijn als de verbinding wegvalt, zal een fout optreden als de in- of uitputstreams worden aangesproken. Om de gebruiker over het verlies te informeren, krijgt de gebruiker ook nu het errorscherm te zien. Hij kan kiezen om de connectie weer op te zetten, of om een device search uit te voeren, om zo na te gaan welke apparaten binnen het bereik van de telefoon liggen.
4. Conclusie

Het in 2008 ontworpen K-Light systeem is een statisch systeem, wat betekent dat na encapsulatie niets meer kan veranderd worden aan de programmatie. Het project besproken in dit werk heeft als doel dit systeem dynamisch te maken. M.b.v. een GSM als afstandsbediening moet het mogelijk zijn de animatie van dit kleed aan te passen aan de wensen van de gebruiker.

De ontwikkelde applicatie kan de intensiteit van de LEDs en de gevoeligheid van de accelerometer aanpassen. Ook kunnen alle lichtjes aan of uit gezet worden. Ten slotte kan informatie opgevraagd worden over het K-Light systeem zelf.

Als de verbinding tussen telefoon en kleed wegvalt, wordt de gebruiker hiervan op de hoogte gesteld.

De bedoeling van dit geheel is een mogelijke alledaagse SCB toepassing te illustreren.

De compatibiliteit van de module is echter niet beperkt tot het K-Light systeem. Zijn minimale grootte garandeert immers niet enkel het comfort van de gebruiker, de integratie in een reeds bestaand systeem wordt er ook aanzienlijk door vereenvoudigd.

De dag van vandaag is de meerderheid van de moderne GSMs uitgerust met de Bluetooth technologie. Recente Bluetooth modules bieden de Bluetooth technologie aan, in combinatie met een programmeerbare microcontroller. Bij gebruik van zo’n module kan aanzienlijk plaats bespaard worden. Bovendien lanceerde de Bluetooth Special Interest Group deze zomer Bluetooth v.4. Deze versie combineert een zeer laag gebruik met een, voor deze toepassing hoog genoegge datarate.

Deze recente ontwikkelingen beloven een rooskleurige toekomst voor smart textiles met Bluetooth functionaliteit.
Contents

Contents xix
Nomenclature xxiii

1 Introduction 1
  1.1 Smart textiles . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
  1.2 The K-Light dress . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

2 Literature Study 4
  2.1 The Stretchable Circuit Board Technology . . . . . . . . . . . . . . . . . . . 4
    2.1.1 The STELLA project . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
    2.1.2 The developed SCB technology . . . . . . . . . . . . . . . . . . . . . . 5
    Base Material . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
    Conductive Material . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
    Process Flow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
    Rigid components on a stretchable substrate . . . . . . . . . . . . . . 8
    2.1.3 The original K-Light System . . . . . . . . . . . . . . . . . . . . . . 9
  2.2 The Bluetooth Wireless Technology . . . . . . . . . . . . . . . . . . . . . . . 10
    2.2.1 Protocol stack . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
    Bluetooth Radio Layer . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
    Baseband and Link Transport Layer . . . . . . . . . . . . . . . . . . . . 13
    Link Manager Protocol . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
    Host Controller Interface . . . . . . . . . . . . . . . . . . . . . . . . . . 15
    Logical Link Control and Adaptation Protocol . . . . . . . . . . . . . . 15
    Radio Frequency Communication . . . . . . . . . . . . . . . . . . . . . . 16
    2.2.2 Profile Specification . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
    Generic Access Profile . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
    Service Discovery Profile . . . . . . . . . . . . . . . . . . . . . . . . . . 18
    Serial Port Profile . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
Contents

4.2 Communication with the remote system .............................................. 48
  4.2.1 Java class BTManager .......................................................... 48
    Device discovery ........................................................................... 49
    Service discovery and connection establishment ......................... 50
    Communication protocol ............................................................ 51
  4.2.2 Connection issues .................................................................... 53
    The remote device cannot be reached ........................................... 53
    A connection loss occurred ......................................................... 54

5 Conclusion ......................................................................................... 56

A Manual Remote Control ................................................................. 57
  A.1 Installing the application on your phone ...................................... 57
  A.2 Connecting to K-Light ............................................................... 60
  A.3 Changing the parameters .......................................................... 60
  A.4 Changing the animation modes ................................................... 60
  A.5 Requesting system information .................................................. 60

B Source Code Remote Control .......................................................... 61
  B.1 MIDlet: RemoteControl.java ..................................................... 61
    B.1.1 Functionality ................................................................. 61
    B.1.2 Code .............................................................................. 61
  B.2 Visual Design: Graphics.java .................................................... 72
    B.2.1 Functionality ................................................................. 72
    B.2.2 Code .............................................................................. 72
  B.3 Interface: Handler.java .............................................................. 79
    B.3.1 Functionality ................................................................. 79
    B.3.2 Code .............................................................................. 79
  B.4 Class: BTManager.java .............................................................. 80
    B.4.1 Functionality ................................................................. 80
    B.4.2 Code .............................................................................. 80
  B.5 Class: Parameters.java ............................................................... 86
    B.5.1 Functionality ................................................................. 86
    B.5.2 Code .............................................................................. 86
  B.6 Class: Connectable.java .............................................................. 89
    B.6.1 Functionality ................................................................. 89
    B.6.2 Code .............................................................................. 89
  B.7 Class: ConnStore.java ............................................................... 93
Contents

B.7.1 Functionality ................................................. 93
B.7.2 Code .......................................................... 93

Bibliography ......................................................... 96

List of Figures ....................................................... 99
Nomenclature

Abbreviations
ACL Asynchronous connectionless
CDC Connected Device Configuration
CID Channel Identifier
CLDC Connected Limited Device Configuration
CTS Clear to Send
EEPROM Electrically Erasable Programmable Read-Only Memory
FH/TDD Frequency Hopping / Time Division Multiplexing
FINA Fédération Internationale de Natation
GAP General Access Profile
GFSK Gaussian Frequency Shift Keying
GOEP General Object Exchange Profile
GPIO General Purpose Input/Output
HCI Host Controller Interface
I^2C Inter Integrated Circuit bus
I/O Input/Output
IP Internet Protocol
ISM Industrial, Scientific and Medical
ISO International Organization for Standardization
ISP In-System Programming
IZM Institut für Zuverlässigkeit und Mikrointegration
JTAG Joint Test Action Group
L2CAP Link Logic Control and Adaptation Protocol
LC Link Control
LED Light Emitting Diode
LMP Link Manager Protocol
MAC Media Access Control
ME Micro Edition
Contents

OS  Operation System
OSI  Open Systems Interconnection
PAN  Personal Area Network
PCB  Printed Circuit Board
RFCOMM  Radio Frequency Communication
RTS  Ready to Send
SCB  Stretchable electronic Circuit Board
SCL  Serial Clock Line
SCO  Synchronous connection-oriented
SDA  Serial Data Line
SDP  Service Discovery Profile
SIG  Special Interest Group
SMI  Stretchable Molded Interconnect technology
SPI  Serial Peripheral Interface bus
SPP  Serial Port Profile
SRAM  Static Random Access Memory
STELLA  Stretchable Electronics for Large Area applications
TPU  Thermoplastic Polyurethane
TUB  Technische Universität Berlin
UART  Universal Asynchronous Receiver/Transmitter
USART  Universal Synchronous Asynchronous Receiver/Transmitter

Symbols
BT  μCU  Microcontroller from the Bluetooth module
K-Light  μCU  Microcontroller from the original K-Light system

Units
kbs  kilo bit per second
Mbs  Mega bit per second
Chapter 1

Introduction

1.1 Smart textiles

About a decade ago, a standard engineer had very little to do with textiles. Nowadays a growing interest in the textile, and even fashion industry is shown.

Think about sportswear. By wearing the LZR Racer Speedo swimsuit Michael Phelps amongst others broke 84 world records since February 2008. It is not for nothing Speedo advertises the suit as the World’s fastest swimsuit. It was developed in cooperation with the universities of Otago in New Zealand and Nottingham in the United Kingdom, and tested in NASA’s wind tunnels. It is made of woven elastane-nylon and polyurethan, and proved so effective that the original version got banned by the FINA starting January 2010. \[14\] \[15\]

Also in the medical sector smart textiles are applied in several areas going from wound care materials, to drug-based release systems and electronic sensors for health care. \[16\]

But in the fashion industry the interest for technology is growing as well. Not only is it considered hip to dress up like a nerd, the newest hype in fashion design is to integrate electronics into clothes. The American Diana Eng, a contestant in the 2005 Project Runway with Heidi Klum, is probably one of the best known fashion designers integrating electronics into her designs. In February, 2010, she organized the Fairytale Fashion Show, a show that made it onto American national television. With the help of LEDs, microcontrollers and conductive wires, she made a whole collection of illuminating dresses, tops and skirts. \[17\]
Chapter 1. Introduction

1.2 The K-Light dress

A prove of the growing interest even on a European level, is the funding of the Stretchable Electronics for Large Area applications, or simply the STELLA project, for an amount of 7 million Euro from 2006 to 2009.

During this period, STELLA served as a technology platform for Dutch, French, Belgian and German researchers looking for innovative ways to bring wearable electronics based on woven technologies closer to the citizen. [ref. newsletter 1]. Within the project different technologies for Stretchable Circuit Boards (SCB) were developed. The people from IMEC created the Stretchable Molded Interconnect (SMI) technology in association with the University of Ghent[7]. Using this technology conductors and interposers are moulded in silicone.[?] At the Technische Universität Berlin (TUB) and the Fraunhofer Institut für Zuverlässigkeit und Mikrointegration (IZM), both located in Berlin, Germany, a new technology was developed. This technology is extensively described in section 2.1.2.

In order to demonstrate their newly developed technique, the researchers from IZM and TUB created a dress called K-Light, which is pronounced like kleid, the German word for dress. 32 LEDs are randomly distributed over the left side of the lower part of the dress. Each of these LEDs can be switched on and off or just dimmed individually. When the woman wearing the dress stands still, the dress will be in idle mode, and the light intensity is billowing. This effect is comparable with stars on a bright night. When the woman starts moving, the LEDs will start twinkling more intensively, without ever giving the impression of being a Christmas tree though.

The dress was manufactured by Mareike Michel, a fashion design student at the Hochschule für Technik und Wirtschaft in Berlin. It is made out of a dark blue fabric, as pictured in figure 1.1. On her website, a demonstration movie is available, showing the effects for different velocities.[4]

In June, 2009 the developers of the IZM and TUB and Mareike Michel received the Avantex Innovation Award for their design. Judged by an international jury, this price is given at the international exhibition for innovative apparel textiles held in Frankfurt.[18]

However, K-Light is a static system, it is thus not possible to change any program parameter after the system is encapsulated.

This is why a communication system is designed that makes it possible to program the dress even after encapsulation. It is opted to use a mobile phone as remote control.
Chapter 1. Introduction

Figure 1.1: The K-Light, with the shining LEDs visible. [3]
Chapter 2

Literature Study

2.1 The Stretchable Circuit Board Technology

2.1.1 The STELLA project

As already mentioned before, the STELLA project was liberally funded by the European Commission. The project ran from June 2006 until February 2010. To demonstrate the project’s achievements, several wearable large area electronic systems were developed. These systems ought to be stretchable, bendable and soft-touch in order to guarantee a high level of comfort while wearing them. Some examples of the developed smart textile systems are a fitness monitoring system for Philips, an infant respiratory monitoring system and a pressure sensing system for smart shoes. [2].

These systems are called Physically Ambient Electronics and in general they consist of a small sensor, an algorithm to interpret the measurements and a preferably wireless connection to transmit the data for possible further processing.

As these applications are often integrated into clothes, the main focus of STELLA the developers was the ease and comfort by which these applications could be worn near or attached to the body.

Traditional electronic applications are stiff, and therefore quite uncomfortable to wear. Flexible boards usually bend in only one direction, which is not sufficient to follow the movements of the human body. Hence the cooperation of the STELLA developers to accomplish a new fabrication method for a complete stretchable board. [19].

With those stretchable boards, wearable electrical systems that follow the shape and movements of the body no longer are an illusion. Electronic applications evolve from being completely rigid and hard to e.g. a soft piece of clothing.
2.1.2 The developed SCB technology

The TUB and IZM partnered together to develop a new method to manufacture a SCB. They premised that their technology should be mostly stretchable. Also, the process steps should guarantee a mass production at a low or medium cost. At last, the whole process should be compatible with standard electronic components and packages in order to make a market entrance possible [3].

The result looks like a single-sided Printed Circuit Board (PCB), but is stretchable in at least two dimensions. The individual electrical components or the system as a whole can be encapsulated if necessary and the system can be laminated onto fabric.

Base Material

For the base material, thermoplastic polyurethane (TPU) is chosen. This is an elastic material, and moreover it is biocompatible. For instance, the rubber bands in underwear are made from TPU, as well as the rubber covers for your iPhone or iPod [20]. With a melting temperature of 170°C, TPU can be used as a glue to laminate the system on a substrate, or textile in particular in a later stadium. Furthermore, it is cost effective and available in high quantities [3].

Conductive Material

The SCB uses the standard PCB conducting metal copper foil (with a thickness of 35µm) as conducting material. One of copper’s great assets, is its high and not fluctuating conductivity, also during stretching. Here it outperforms other tested alternatives, like conductive inks or pastes.

That copper is chosen to conduct, may be surprising, as copper is in fact only elastic for deformations smaller than 0,1%. Although, when shaped in the proper manner, elongation of more than 100% can be achieved. The best results were realized with a horse-shoe like pattern, with a lengthening of more than 300%. [3]. Figure 2.1 shows the copper lines on TPU first in relaxed and then in elongated position. In the stretched track, the special copper pattern is clearly recognizable. The tracks are seperated from each other due to testing purposes, but in an actual system this is not needed.

Up to 1600 stretching cycles with a lengthening of 10% are required until the tracks break down, which implies a high reliability. If they are laminated onto an unelastic fabric however, this number increases even more.
Chapter 2. Literature Study

Figure 2.1: The special structure of the copper lines. The left side displays the relaxed tracks, the right the elongated ones. [3]
Chapter 2. Literature Study

Process Flow

In order to keep costs low and to assure an easy implementation of the new technology, the SCB production process is based on the ordinary PCB process as much as possible. This way the same equipment can be used and as a result not too big investments are required.

First, the copper wires are laminated to the TPU substrate. Next the TPU can be structured, which happens in a similar manner than for a normal PCB. Pitches down to 100µm line/space can be achieved with photolithography. This structured substrate could already be used for simple electronic circuits, but for more sophisticated ones, some extra process steps were developed.

At first, solder resist is applied and structured using a standard flex solder mask. In figure 2.2 a copper surface finish is applied next, but in fact, this can be done before the first step as well. Subsequently, the conductor lines outside the solder resist are isolated. This is done by laminating another, already structured TPU foil onto the substrate. This foil should overlap with the substrate for stability reasons. Also is the TPU much harder to structure in an accurate way than the solder resist. Now the components can be assembled onto the substrate. An overview of how this more sophisticated substrate is developed, is given in figure 2.2.

![Figure 2.2: The process flow for sophisticated SCB manufacturing.](image-url)


Rigid components on a stretchable substrate

Since standard electrical components tend to be rigid, the SCB will consist of both rigid and stretchable areas. Because it is important to safeguard the components and the connections with the conductor lines from mechanical stress, the distinction between the two areas has to be introduced in the SCB. If the strain can be distributed over a surface, the local strains on the connection will decrease considerably. This distribution can either be done via encapsulation of the single electrical components with stiffer materials, or with the addition of a big copper area, as shown in figure 2.3. The first will also shield the components against water for instance, while the latter will decrease the local strain at connections between the rigid components, the flexible base material and conductor lines. As a consequence, this will reduce the chances of failure. Using such a copper area also allows the integration of a whole module on an SCB. Next, also solder resists can be applied on those areas that require a certain stiffness, as most of the contemporary resists are not very stretchable.

\[\text{Figure 2.3: An extra copper area can be used to introduce an extra stiffness to a component area.} \]

As the substrate material has a melting temperature of 170°C, a solder paste that melts at a lower temperature is required. SnBi showed the best results, but also other, low melting point adhesives could be used. \[\text{3}\].

8
Chapter 2. Literature Study

2.1.3 The original K-Light System

K-Light was designed to demonstrate the special SCB features. To prove that the technology can handle large areas, the conducting copper wires on which the 32 LEDs are placed, are up to 50 cm long.
To show the airiness of SCBs, the system is laminated onto the fabric. The fabric however never looses the natural and free motions while the dress is moving.
At the end of the conductor lines conducting press buttons are punched through the fabric. These buttons attach the SCB to a belt equipped with a tiny integrated battery.
The heart of the whole system is the microcontroller, which generates the light effects.
Two LED drivers control the brightness of the LEDs.
A not yet encapsulated version of the system manufactured on an SCB is shown in figure 2.4.

\[\text{Figure 2.4: The K-Light system on an SCB before encapsulation.}\]
2.2 The Bluetooth Wireless Technology

The Bluetooth wireless technology is a relatively new technology developed by Ericsson in 1994. Four years later, in 1998, IBM, Intel, Toshiba and Nokia joined Ericsson and formed the Bluetooth SIG, short for Bluetooth Special Interest Group.[21] Later that year, several other companies were invited to join the SIG, in return they contributed to the development of the Bluetooth technology. In 1999 the first open standard version was released. The Bluetooth SIG does not make, manufacture, or sell Bluetooth enabled products itself but guarantees the standardization of the Bluetooth technology. Its main task is to publish Bluetooth specifications, administer the qualification program, protect the Bluetooth trademarks and evangelize Bluetooth wireless technology. The group now has over 13000 members.

The technology got its working title inspired on a viking king of Denmark. This king named Harald Bløtand christianized Scandinavia and united Denmark, a part of Sweden and Norway. An analogy was made as the new technology was expected to unify the mobile phone, automotive and computing industries. SIG marketeers could not come up with a better name and the name stuck.[?, bookJavaAPI]

The original intention of the Bluetooth technology was to get rid of the salad of cables between a computer, mouse, printer and so on. But soon it was realized that Bluetooth could offer so much more. By connecting all kinds of fixed and mobile devices within a reach of 10 meters, it appeared possible to set up a personal area network (PAN). Distances up to 100 meters can be reached by increasing the sending power from 1mW to 100mW, which however has a bad influence on the battery life.[9, 10, 11, 12]

2.2.1 Protocol stack

The technology is built like a protocol stack corresponding to the ISO OSI model for communication systems developed in the late seventies[22]. The Bluetooth technology’s protocol stack is therefore quite analog to the more familiar IP-protocol stack, as shown in figures 2.5 and 2.6(a). On the right side of these figures, the accordancy with the OSI model is clarified. By taking horizontal slices through the Bluetooth protocol stack, each layer is described, and it is defined how the technology works. The whole stack can be divided in two parts. The lower part is called the bluetooth controller, the upper part is the bluetooth host, as shown in figure 2.6(b). They communicate with each other through the Host Controller Interface (HCI).
Figure 2.5: IP Protocol stack with on the right the OSI Model
(a) Bluetooth protocols with their correspondence to the OSI Model.

(b) Bluetooth protocol stack

Figure 2.6: The Bluetooth protocols
Chapter 2. Literature Study

Bluetooth Radio Layer

The bottom layer is called the blue-tooth radio layer, and is used for the wireless transmission of data packets. This is done in the unlicensed 2.45 GHz frequency band, properly known as the ISM frequency band which is reserved for industrial, scientific and medical use. For optimal use of this frequency band, the GFSK modulation principle is adapted, so that a symbol rate of 1 megasymbol per second can be achieved. This corresponds with a bit rate from 1 Mbps for the blue-tooth low energy technology, to 24 Mbps for version 3.0 HS (High Speed) which was released in April 2009. This means that a trade off between battery life and data rate has to be made. The latest version of the low energy technology, Bluetooth version 4.0 was announced in December 2009, and released in July 2010 [23, 24].

Data is sent over the channel by modeling changes in frequency, at a rate of 1600 per second. Because of the intense use of the ISM band, adaptive hopping among 79 frequencies displaced with 1 MHz is applied to avoid interference. This means that the actual use of the blue-tooth frequency band goes from 2.402 GHz to 2.480 GHz. [25] In order to reduce the spectral width of the pulses, the spectral width is reduced with a Gaussian filter. [26]

Baseband and Link Transport Layer

The second layer is called the baseband or link transport layer. Over this layer two blue-tooth devices make a connection. Devices connected to each other are grouped in a piconet. Each piconet consists of one master and up to seven active slave devices. However, more slaves can be connected to the piconet when they are in an inactive state. It is possible to connect several piconets into one scatternet, and a slave in one piconet may be a master in another. A device can only be master in one piconet. Figure 2.7 shows four piconets grouped together in one scatternet.

A physical channel between two devices implements a FH/TDD algorithm in order to minimize interference in the open ISM band. The frequency hopping sequence is determined by the unique 48-bit Blue-tooth Device Address, comparable to the MAC-address in the link-layer of the IP-protocol stack, and by the selected sequence. The frequency hopping phase on the other hand is determined by the blue-tooth master’s internal clock. The channel is chopped into time slots with a length depending on the type of channel. For the basic piconet physical channel time slots have a length of 625 $\mu$s. Because of the TDD implementation, up- and downlink traffic happens in the same frequency band, during different time slots. A slave device will use the odd numbered time slots, while the master will use the even numbered slots.
Only one physical link can exist between a master and each of his slaves. This physical link may however carry several logical links over which the communication happens. Five types are defined, the three main types are link control (LC), synchronous connection-oriented (SCO) and asynchronous connectionless (ACL) logical links. The LC is used at the link control level present in the baseband layer and is carried in the packet header while the other logical links are carried in the payload header. The data transmission in SCO links happens at a rate of 64 kByte per second, and packets are never retransmitted. As a consequence this type is mainly used to send voice data, as is the case with UDP in the IP transport layer. This link type can only occur in a symmetric point-to-point piconet, in other words, in a master-single slave configuration. A distinction can be made between user synchronous (SCO-S) and user extended synchronous (eSCO-S) connection oriented links. Any master can have up to 3 SCO links.

On the other hand, the link transport layer also offers asynchronous connectionless (ACL) communication. An ACL logical link between two devices will retransmit packets if one gets lost. A master only supports one ACL link at a time, but can be connected to multiple slaves. An ACL logical link can thus be seen as a point-to-multipoint link. There are two subtypes of ACL logical links. The first is ACL Control (ACL-C), the second is User Asynchronous/Isochronous (ACL-U). [27, 28, 29]
Chapter 2. Literature Study

Link Manager Protocol

The upper layer of the Bluetooth controller is the Link Manager Protocol (LMP). This protocol operates on devices connected via an ACL link, and is used to communicate between their Link Managers. By controlling and negotiating all aspects of the connection between two devices, the protocol provides a certain level of security. This protocol decides e.g. over authentication, pairing, power control, etc. This is done in terms of transactions between two devices, where each transaction has a certain goal and consists of a set of messages. These messages are also known as Protocol Data Units (PDUs) and are combined in a single slot package with a payload header of 1 byte. This header indicates whether the ACL link is of the ACL-C type or the ACL-U type. The first one has a higher priority, is used for LMP messages and is carried either by SCO or ACL logical transport, while the latter is usually carried by ACL logical transport and contains L2CAP and user data. [30].

Host Controller Interface

The three layers mentioned above make up the Bluetooth controller and are considered the hardware implemented part of the Bluetooth protocol stack. The part that usually is implemented in software is called the Bluetooth host. These two major parts of the Bluetooth protocol stack communicate with each other through the host controller interface (HCI). In some devices like a headset e.g., the Bluetooth host and controller are integrated, and the HCI is not implemented though. The HCI exists across three subsections: Bluetooth host, physical bus and Bluetooth controller. At the host side, the HCI is located at the HCI driver where it notifies the host when an event occurs. Such an event reflects on e.g. device discovery, authentication and encryption or host flow control. The HCI firmware on the other hand, implements HCI commands for the Bluetooth hardware. The physical bus with the host control transport layer acts like a translator between the HCI implementation on the lower software layers and the HCI implementation on the firmware on the Bluetooth hardware. The transport layer is able to transfer data without any knowledge of it, so both the driver and the controller can be updated without affecting the transport layer. [31]

Logical Link Control and Adaptation Protocol

The lowest host layer is located right on top of the HCI and is called the Logical Link Control and Adaptation Protocol (L2CAP). Because of the limited size of packets in the baseband, L2CAP supports packets up to 64 kbytes, the maximum transmission unit is also indicated as MTU. Longer packets will be segmented and reassembled in the upper layers.
L2CAP is based on the principle of channels and provides a per-channel flow control. Each end point of such a channel has a channel identifier (CID). Every device can assign CIDs independent from the other devices, hence a CID is not unique. There are some reserved CIDs for specific purposes though. With the use of the CIDs L2CAP is able to multiplex between different services offered by the upper layer protocols. These services can be both connection-oriented and connectionless services, but all L2CAP channels are supported by an ACL logical transport. This implies that the protocol implements the retransmission of lost packets and thus offers the required quality of service. The connectionless channels provide only one-way traffic and are used when a device wants to map a group of addresses into a piconet. The CID on the source then represents one or more remote devices. An overview of how the communication between two devices happens is shown in figure 2.8.

![Diagram of communication between devices](image)

**Figure 2.8:** Different devices are connected with each other through different kinds of channels

**Radio Frequency Communication**

On top of L2CAP, the radio frequency communication protocol, or simply the RFCOMM protocol operates. It provides an emulation of the RS-232 serial port connection over a wireless link. It is a simple transport protocol that doesn’t make a distinction between the following two types of devices. The first type consists of communication end point devices,
while the second type of devices consist of devices that are just a part of a communication segment. RFCOMM is the standard protocol used in an application that runs on JavaME. Data is then sent in streams, similar to a socket connection.  

### 2.2.2 Profile Specification

The SIG has defined standard ways of interaction between the different protocols of the Bluetooth protocol stack, which are called the Bluetooth profiles. Depending on the purpose of a Bluetooth device, one or more profiles are implemented in the device. The goal of these profiles is to ensure that devices from different manufacturers can communicate with each other. There are several types of Bluetooth profiles, but the basic four are the Generic Access Profile (GAP), the Serial Port Profile (SPP), the Service Discovery Profile (SDP) and the Generic Object Exchange Profile (GOEP). These profiles are known as the transport profiles, while other profiles that usually depend on one of these basic four are called application profiles. An overview of the basic profiles and some of their depending profiles is shown in figure 2.9.

![Figure 2.9: The most important profiles of the Bluetooth Profile hierarchy](image)
Chapter 2. Literature Study

Generic Access Profile

GAP is the mother of all profiles, all other profiles are dependent on the GAP. This profile defines the generic procedure of how a connection between devices should be established. This includes amongst others the device discovery and link management.\[36\]

Service Discovery Profile

SDP defines the operations necessary to perform the service discovery. The profile dictates how to locate and identify services registered in remote Bluetooth devices. SDP is the only user-initiated profile. More specifically this means that, in contrast to the other Bluetooth profiles initiated via application-level actions, SDP is initiated via user-level actions. During service discovery a device scans the available services or searches for a desired collection of services on a specific device, or on any discoverable device in the area. Both of these approaches require first a link with the discovered devices upon which can be inquired. The service inquiry is done over a connection-oriented transport service in L2CAP (2.2.1), which is built upon a baseband ACL-link, as explained in 2.2.1.

A distinction between two kinds of devices is made, the first is the local device, the second the remote device. A local device initiates the service discovery procedure, and must therefore implement the client side of the specification for SDP. A local device must contain a user interface to enter the user input and return the service search results. A device without such an interface cannot be considered a local device.

A remote device on the other hand must have the server side of the SDP specification implemented. It responses on the service inquiries generated by the local device. Either several remote devices are accessed during one service inquiry, or a service inquiry is executed on one specific remote device.

A complete and detailed specification of this profile is given in \[35\].

Serial Port Profile

SPP is the profile that describes how a RS232 serial cable emulation between two devices can be established with the use of the RFCOMM protocol. SPP ensures that any application can run on either two devices, as if there were a real cable between the two peers. Data rates up to 128kByte per second are guaranteed, higher rates are optional. As only one connection at a time is possible with this profile, only point to point connections are considered. This doesn’t necessarily mean that only one connection can exist between two devices though. Most devices can run multiple sessions of the profile at the same time, resulting in multiple cable emulations. This is not very surprising, as two devices could be connected by multiple cables as well.
Chapter 2. Literature Study

The emulated cable connection will not be released when devices are put in low power modes. In the next paragraph two possible SPP adaptations are described, a detailed overview of how the profile cooperates with the Bluetooth protocols is given in [34].

Whenever a device desires to emulate a cable connection with another device, the first device should obtain the RFCOMM channel number of the desired application in the remote device. This is done using the SDP profile described above. Next, if authentication and encryption are desired, these can be turned on. When there isn’t already an RFCOMM session running between the two devices, an L2CAP channel should be established upon which a new RFCOMM session can be started. Then a new data link connection can be initiated on this RFCOMM session with the known service channel number and the virtual serial cable between the two applications is ready to use. In the case there was already an RFCOMM session running, the new data link can be initiated upon this one. In order to accept an incoming request for serial cable emulation, a device should first take part in the authentication procedure and turn on encryption if necessary. To establish the serial connection, the device should then successively accept the new L2CAP channel request, the RFCOMM session on this channel and at last, the new data link connection.

Generic Object Exchange Profile

GOEP is built upon the SPP profile, and contains several concrete-use profiles. The profile defines the requirements to send an object from one device to another.
Chapter 3

Hardware design

The main goal of this master thesis is to add a communication interface to the existing K-Light system. This chapter covers the hardware part of the project. First, a small introduction to the main problems is presented. Next, the selection process follows, in which it is described which components were chosen and why. This chapter also explains the written code into detail, and explains which algorithms were developed to reach the communication goal. Another important part of the hardware development is the physical design. The fabricated boards and the problems encountered in the process will also be discussed. It is clear that all these parts of the design are not independent. The last section tries to give an overview of the whole system, explaining the connections between the explained sections.

3.1 System Design

3.1.1 Existing System

The system developed by IZM and TUB consists of a main microcontroller [6], an accelerometer, two light sensors, two LED drivers and 32 LEDs. The microcontroller communicates with the sensors via the I²C bus and with the drivers via the SPI interface. From now on, this microcontroller will be referred to as the K-Light \( \mu \text{CU} \).

The main task of the microcontroller is to recollect and process the sensor data. The processed data is then sent to the LED drivers. This way each single LED can be addressed.

This is a static system, which means that once the K-Light \( \mu \text{CU} \) is programmed, and the whole system packaged, the animation for the LEDs and the calibrated sensibility for the sensors cannot be changed anymore.
3.1.2 Developed System

For this project, the goal is to add flexibility so the program of the system can be changed. This means, to make it possible to control the animation and the sensor parameters, even after laminating the electronics on textile. Since the main system needs to react in real time, the K-Light \( \mu \text{CU} \) will overload when handling the communication. To overcome this issue, an extra microcontroller is added. From now on, this microcontroller will be called BT \( \mu \text{CU} \)\(^6\). Its task is to manage the Bluetooth communication and to send only the useful data to the main microcontroller.

3.1.3 Communication Buses

The most used communication buses used in the design are the I\(^2\)C and UART interfaces. A short description will follow in order to explain the basic functioning of these buses.
Chapter 3. Hardware design

I²C Bus

I²C stands for Inter-Integrated Circuit. This is a communication bus developed by Philips for low-speed peripherals. It is a synchronous multi-master serial single-ended bus. This communication bus uses only two bidirectional open-drain lines, the serial clock line (SCL) and the serial data line (SDA). These are pulled up with resistors.

In theory, speed rates of 100 kbs are reached in standard I²C mode, 400 kbs in Fast mode, 1 Mbs in Fast mode plus or Fm+, and 3.4 Mbs in High Speed mode. In practice, these speed rates are lowered due to unwanted capacitances and due to the framing of the data. A total of 112 nodes can communicate on the same bus. This is because I²C uses a 7-bit address and had 16 reserved addresses.

The communication protocol is always started by the master. It gives a start signal consisting of lowering the data line while keeping the clock line high. It then transmits its clock on the SCL line. The data is sent over the SDA line, being sampled every time the clock rises. Finally, the master transfers a stop condition by pulling the data line up while the clock stays high.

There are four possible transmission modes when using one master in a bus, Master Receiver Mode, Master Transmitter Mode, Slave Receiver Mode or Slave Transmitter Mode. Every frame consists of 8 data bits and one (non) acknowledgment bit. The slave address and the data is transferred over the same line. In order to know when the address is sent and when the data, the protocol determines that the first byte after a start condition should be the address. Since the address space has only 7 bits, the last bit is left to say whether it will be a read or write operation. If a slave recognizes its address in this first byte, it pulls the SDA line low on the 9th clock tick. This means the address has been acknowledged and the transfer can continue.

What will happen next depends on whether the data will be transmitted by the slave and received by the master or the other way round. Nevertheless, the principle stays the same, the data is transferred by the sender on the first 8 clock ticks, and the receiver acknowledges on the 9th tick.

UART Interface

UART stands for Universal Asynchronous Receiver/Transmitter. This is an interface that converts data between its serial and parallel form. Since this is an asynchronous communication, some parameters settings have to be agreed
Chapter 3. Hardware design

by both sides. These settings include baud rate, data bits, little endian or big endian, parity bit, stop bits, flow control, among others.

The UART communication used in this design uses 4 data lines. These are the receive line Rx, transmit line Tx, clear to send CTS, ready to send RTS. When connecting two UART interfaces, the Rx/Tx and RTS/CTS lines are crossed, as figure 3.3 illustrates.

Like the names suggest, the Rx line is the data input line and the Tx line is the data output line. This means that receiving and transmitting can happen simultaneously.

The CTS and RTS lines are used for hardware flow control, CTS being the input line and RTS the output line. In a normal situation, the RTS line is low. If a device is not able to receive more data, it will pull up its RTS line. The other device will sense that its CTS line is pulled up and should stop transmitting after finishing the last byte of the current transmission.

![UART communication](image)

Figure 3.3: UART communication

3.2 Selection Process

In order to integrate the new design on a single SCB with the already existing one without losing wearability, some restrictions regarding size and power consumption have to be met. First, if the designed system is too large, it is uncomfortable to wear. Second, if the design consumes too much energy, the lifetime of the battery shortens. To maintain the same battery lifetime a stronger, and thus in general a bigger battery is needed. The battery pack is integrated in the dress’s belt however, so a bigger battery can not be allowed. A problem encountered during the selecting procedure is, that the market does not offer a big variety of both small and power saving modules. A lot of meditated choices have to be made in order to develop a good design out of non-ideal components.
Chapter 3. Hardware design

3.2.1 Bluetooth module

The first and most important component to select is the Bluetooth module. There are many types of modules available in the market, differing in offered profiles, protocols, antenna, communication interface and module size.

The major part of modules only offer the Bluetooth Controller, which then can be accessed with the help of the HCI protocol. Other modules offer just the General Access Profile, which means that the implementation of the upper protocol stack is left up to the designer. Some modules offer the protocol stack up to the RFCOMM protocol, enabling the SPP profile or even more sophisticated profiles like Object Push and Dial-up Networking Profile to operate. To be able to focus on the real design instead of on the implementation of Bluetooth protocols, the choice is made for a module that offers the Bluetooth protocol stack up to at least the RFCOMM protocol.

Another crucial selection criterion between the available modules, is the size of the module. Since the design does not offer any other wireless technologies, a module with an integrated antenna is the best choice. This spares an extra balun and eliminates the need for an external antenna, which would be size consuming.

The last main difference between the Bluetooth modules, is the communication interface they offer towards the other components. The most common buses are UART, USART, SPI and I²C. For this design, the communication interface is not the main focus, so not much attention is paid to it.

All these differences are reflected on price, size and usability. For instance, a Bluetooth Controller module without an integrated antenna will be the cheapest and smallest (6.5 mm x 9 mm) one, while on the other hand a module with three available communication buses, an integrated antenna and the most used high-level profiles will be about 14 mm x 23 mm and will cost about €50.

In other words, at some point a trade-off between size and price has to be made. Price is important when thinking that this product could be commercialized. On the other hand, size is crucial for the wearability of the system.

At the time the module was selected, the new Low Energy Technology for Bluetooth (Bluetooth v.4.0) was not available yet, so focus was to minimize the size of the module and to maximize the ease of integration with the existing system.

In order to achieve a very compact design, the possibility to get a Bluetooth module with a programmable internal microcontroller was explored. This would mean that no extra microcontroller is needed, sparing size, power and extra passive components.

Panasonic offered a perfect solution, but after two months of contact and negotiations,
the deal blew off. Given the lack of time, size became the most important restriction. The module is then chosen out of a selection of other Panasonic modules: PAN1315, PAN1555, the Blue Radio’s BR-C46AR and the National Semiconductor’s LMX9838 and LMX9830.

The PAN1315 and LMX9830 do not have an integrated antenna, so these were the first modules to be rejected. The choice had then to be made between the PAN1555, BR-C46AR and the LMX9838. A table comparing size and price can be seen in figure 3.4.

<table>
<thead>
<tr>
<th>Module</th>
<th>Size [mm]</th>
<th>Price [€]</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMX9838</td>
<td>10.0 x 17.0 x 2.0</td>
<td>19.67</td>
</tr>
<tr>
<td>BR-C46AR</td>
<td>11.8 x 17.6 x 1.9</td>
<td>51.30</td>
</tr>
<tr>
<td>PAN1555</td>
<td>13.5 x 22.7 x 2.9</td>
<td>28.98</td>
</tr>
</tbody>
</table>

Figure 3.4: Comparison table between the three possible Bluetooth modules

Basing the decision on the price-size ratio and on the availability, this Bluetooth module was selected.

The Bluetooth module LMX9838

In addition to the short description of the LMX9838 above, in the next section the module will be described in more detail. The National Semiconductor LMX9838 Bluetooth Serial Port module consists of integrated 2.4 GHz radio, baseband controller, crystal, antenna, low-dropout regulator and glue electronics. It offers communication via its four wire UART interfaces, baud rate selection pins, 2 GPIOs and external codec connections. It also offers the SDAP, SPP and GAP Bluetooth profiles. Besides, it can be put in 4 different low power mode types.

Its core is based on the National Semiconductor’s CompactRISC 16-bit processor architecture and Digital Smart Radio Technology. It has ROM and RAM, but also EEPROM, and Patch RAM. EEPROM ensures that the system can save certain configurations, even when the system is powered off. Patch RAM is used to update the firmware.
Chapter 3. Hardware design

At start-up, the module initializes its EEPROM and UART interface. During this time period the module should neither be reset, nor powered off, as this will cause the EEPROM to go in a lockup situation.

In order to get the right baud rate on the UART interface, the OP3, OP4 and OP5 pins should be set like shown in figure 3.5.

![UART frequency selection](image)

Figure 3.5: UART frequency selection

The LMX938 communicates with simple commands contained within a special package format. This format is shown in figure 3.6. The start and end delimiter indicate respectively the start and the end of a package. The packet type contains the kind of packet that is contained in the frame. The opcode is a command specifier. More information about which package types and commands the LMX9838 has implemented, is available in[5]. The checksum is a simple Block Check Character checksum. This is obtained by taking the least significant byte of the sum of the bytes from ”Packet type” to ”data length”.

![LMX9838 communication package](image)

Figure 3.6: LMX9838 communication package

3.2.2 Microcontroller

The next component is the microcontroller. A lot of equipment, going from development kits to large amounts of stocked components, is available at IZM to develop systems with
AVR microcontrollers. Hence, it is obvious the microcontroller selection pool is limited to AVR controllers, so the system can be developed in a shorter amount of time.

AVR offers three kinds of 8-bit low power µCUs. These are the Tiny AVR, the Mega AVR and the XMEGA. The Tiny devices are used for very simple tasks, as they offer a maximum of 512 Bytes RAM, 16 Kbytes Flash and 2 communication buses. They are used in e.g. toys, wireless sensors, remote controls, lighting control, etc. The Mega family consists of a wide range of devices, differing in memory capacity, pins and peripherals. They offer a bigger memory size and more powerful CPUs than the tiny devices. The XMEGA devices focus on performance. They offer a predictable timing, a wide selection of peripherals, crypto engines, DMA Controller to boost the performance, Event System for CPU-independent inter-peripheral communication, among others. The Tiny family is too small for the desired function, while the XMEGA family would be a complete overkill. Hence a Mega device is selected.

At first instance an ATMega168 is chosen. This 8-bit microcontroller offers 16 Kbytes of Flash, 1 Kbyte of SRAM, one USART/SPI interface and one I²C interface. The component has a small footprint and the power consumption working at 4 MHz and with a supply voltage of 3V, has a maximum of 3.5 mA. Although, when starting the algorithm design, it became clear that this µCU would be too small for the application. The next used microcontroller is a ATMega324P. The 8-bit microcontroller offers 32 Kbytes of Flash, 2 Kbytes of SRAM, two USART interfaces (one can be used as SPI), and one I²C interface. The component has an In-System Self-programmable Flash, which makes it possible to write a Boot Loader to change the firmware through e.g. Bluetooth. The power consumption working at 4 MHz and with a supply voltage of 3V, has a maximum of 2.7 mA.

However, for the prototype board and the final board an ATMega644P is used. This microcontroller has the same specifications as the ATMega324P, but has 64 Kbytes of Flash and 4 Kbytes of SRAM. The reason why this is chosen over the ATMega324P, is due to the way flow control is achieved. This is done using buffers to store data. Because of the bigger memory size and the same power consumption, this µCU is a better choice when using large buffers. Moreover, the already existing K-Light system also uses an ATMega644P. Choosing the
same microcontroller facilitates the algorithm development process of the new module because of the recollected experience.

### 3.2.3 Passive components

The most important selection criterion for the passive components is their size, but it also must be guaranteed that the selected components can be delivered in a reasonable amount of time.

For the resistors, capacitors and LEDs, it was decided to use a standard size. For the protoboard the package 0603 was used, and for the final board the package 0402. These numbers represent the dimensions of the rectangular shape in hundredths of inches. The 0402 package is 0.04” x 0.02” (1016 μx 508 μ) big, and the 0603 measures 0.06” x 0.03” (1524 μx 762 μ).

For the crystals some extra precautions have to be taken. For the specification of the crystal it is important to know which voltage is the working voltage of the system. A too low voltage can imply that the crystal vibrates at a different frequency than expected. The two crystals selected for this design were chosen because of their minimal size.

The buttons and pin headers used for the protoboard are standard pieces found at IZM. The only pin headers selected based on size, are the ISP programming pads placed on the final board.

### 3.3 Firmware Implementation

In order to integrate the existing system with its newly developed extension, the code from the K-Light μCU has to be slightly changed. These changes, as well as the communication with the BT μCU are explained in this section. The communication flow and the used protocols are handled first, so the written code will be easier to understand. Next, the different steps of the system development are illustrated.

#### 3.3.1 Internal Protocols

To be able to make a working communication between the K-Light module and the Bluetooth module, some kind of communication protocol has to be developed. A protocol that uses framed packages as communication units is put forward. This allows the system to
be expanded with more features.

The first byte of the frame defines whether it is a reading, writing or special action. The implemented special actions are polling, acknowledgment or error. The second byte tells which parameter will be addressed. The third byte is the total size of the package. To keep the memory management in the microcontrollers simple, one communication unit is 256 bytes long. This is an amount of bytes that can be represented within one byte. This is important for the polling protocol of the I²C bus. This means that 253 bytes of data can be carried per package, an amount that suffices the system’s needs. A scheme of the package can be seen in figure 3.7.

![Scheme of the developed frame](image)

**Figure 3.7:** Scheme of the developed frame

The mentioned polling action is used in the I²C communication between the two μCUs. This is a solution for the next problem. In the existing system the K-Light μCU was already the I²C master. The K-Light μCU asks the BT μCU if there is already some data available for transfer. If this is the case, it then asks the BT μCU to send this data. The polling protocol consists of sending 0x04 to the BT μCU and receiving the amount of bytes of data are ready for transmission. This communication frame is shown in figure 3.8.

![Scheme of the polling frame](image)

**Figure 3.8:** Scheme of the polling frame
Because this polling frame is the most used in the whole system, both the second and the third byte of the normal frame, illustrated in figure 3.7, are removed to make the I\textsuperscript{2}C communication as short as possible.

The last implemented actions are the acknowledgments and the errors. These are used to make the system robust and reliable. Every time the connected remote Bluetooth device sends a command, it is either acknowledged or reported as erroneous. The error can contain information about where the communication went wrong.

If a communication error happens, the K-Light µCU will notice it and make an error frame containing data about what went wrong. As soon as the communication runs again, it will send it to the user.

### 3.3.2 K-Light µCU

The code of the K-Light µCU is not really changed, but only adapted to be able to integrate the BT µCU. The changes can be organized in two groups, the main flow of the program and the handler for the I\textsuperscript{2}C communication.

#### Main Program

The main program processes the received data from the peripherals and updates the LEDs. First of all, the µCU must set its clock right, and initiate its interfaces in the correct manner. Next, the main code checks if there is new data from the BT µCU in the receive buffer. This is done with the already explained polling protocol. If data was sent, it is parsed and the µCU does what it is asked to do. Finally, the controller reads the information coming from the sensors and updates the display. The task of the parser is to decode the incoming data, read the commands and the actions. Once this is known, the main program can proceed by processing the task that the sent command specifies.

If the user sends a read command, the K-Light µCU sends the data it is asked to send. If the user wants to change some data, he K-Light µCU then updates the asked parameter in its EEPROM memory. Data will be lost if the module is turned off during this process. By storing it there, it is assured that the data will still be available even after powering off the K-Light system. After it has stored this data, the main code sends an acknowledgment package back to the remote Bluetooth device. Suppose something went wrong in the process, an error package telling the end user what went wrong in particular is sent. Figure 3.9 illustrates the code operation.
Chapter 3. Hardware design

Figure 3.9: Flow chart of the main code for the K-Light microcontroller

I²C Handler

The handler sends and receives data via the I²C bus. This must happen fast and as transparent as possible. If the handler is assigned too much processing time, the main program will not be able to give a real time experience to the end user. This experience is achieved with the use of a circular receive buffer, written by the handler and read by the main code. This handler implements the driver for the master side of the I²C bus. It manages the receive buffer and communicates with the peripherals. It also monitors the bus, so that if something goes wrong it can act immediately. The handler will then try to reestablish the communication or report a communication error to the main code. After the communication is reestablished, the error frame is sent to the end user.

As this is the master side of the handler, it implements the start and stop conditions, as well as writing data to an addressed slave. Besides these general functions, some specific functions are also offered. The polling protocol and writing functions for registers of some slaves are implemented.
3.3.3 BT $\mu$CU

The main function of the BT $\mu$CU is to act as a bridge between the Bluetooth module and the main system. This is needed because the LMX9838 sends a lot of information and confirmation packages that are not relevant for the proper working of the K-Light system. Besides, this Bluetooth module sends its data inside the frame explained in [...]. This means that a special parser for this kind of packages has to be implemented. Because of the big amount of memory the BT $\mu$CU has available, it is also in charge of buffering the received data. This buffering action and flow control makes it able to assure that no data will get lost due to the slower working of the K-Light $\mu$CU.

Normally, it is expected that the amount of time this controller is used, is very small in comparison with the other controller. That is why it is possible to implement a low power mode that minimizes the power consumption whenever there is no Bluetooth communication with a remote device. To achieve this, all communications are interrupt driven.

Main Program

The main task of this program is to read and decode the data received from the Bluetooth module. First of all, the $\mu$CU must set its clock right, and initiate its interfaces in the correct manner. Next, the $\mu$CU initiates the LMX9838. This is done by resetting the module and setting the right voltage in the module’s input lines. These determine the baud rate for the UART communication between the $\mu$CU and the module.

The $\mu$CU then waits until a frame is received in the UART receive buffer. After this, the package is parsed and the place of the useful information in the buffer is stored. Once this is done, an interpreter checks which kind of package it was, and what to do with it. If the $\mu$CU receives a data package, it just frames it in the correct way and sends it to the I$^2$C transmit buffer.

Every other package contains information that is only relevant for the communication. This kind of data is handled inside de BT $\mu$CU, sparing the K-Light $\mu$CU from this kind of work. Examples of this kind of packages are ACL connection confirmations, device discoveries, SPP connection confirmations, name requests, etc. A flow chart of the main code is shown in figure 3.10.

Receive Function

This function forms the heart of the whole working of the main code. This function reads the packages sent from the Bluetooth module. If this function is too slow, the receive
buffer will become full and the whole system will have to wait. That is why a lot of time was invested to make this function as robust and fast as possible. As explained in section 3.2.1 a LMX9838’s package looks like in figure 3.6. To be able to know where the right information can be found, the begin of the frame must first be found. This is done with a combination of the UART interrupt vector and the receive function. Once the frame is located inside the receive buffer, the function reads all the header fields and stores them. The most important field for the transmission is the Data Length field. Once it is known how many data bytes were sent, the receive function can assign a pointer to the beginning of this bare data and look if the end of the package has been received. If the last byte of the package, the end delimiter, is found, the receive function returns successfully.

**UART Handler**

This handler uses interrupts to drive the UART interface of the microcontroller. The two main functions are the receive and transmit interrupt vectors. They use two circular
buffers, one for receiving and another for transmitting. Besides, the handler offers functions to initiate the interface and to store and read data from the circular buffers.

The receive interrupt vector is activated when a byte is received in the UART interface. It then checks whether the receive buffer is full yes or no. If it is full, it asserts its flow control signal. But if the buffer still has enough space it then checks if a start of a LMX9838 package can already be found. If no package is found, the vector ignores the incoming bytes until a start delimiting byte is found. After this, or if a package was already found, the head of the buffer is updated.

The transmit interrupt vector gets activated when the transmit data register becomes empty. This interrupt is only enabled when data gets written in the transmit buffer, and is disabled after the last byte is sent. The vector first checks whether there is in fact data available in the buffer. If there is, the vector updates the tail of the buffer and the data gets sent.

I²C Handler

In contrast with the I²C handler of the K-Light µCU, this handler has limited options and is interrupt driven. Because it is the slave side of the communication, it only offers an initiation function and a read and write function.

The most important part of this code is the interrupt vector. This is the real bus handler. When the hardware part of the interface is finished with the transmission or reception, the software part must make the right decision. The decision making policy is based on the data on both the receive and status registers. To gain more insight on the workings of the interaction between hardware and software, figures 3.11 and 3.12 explain which codes are read on the status register. This is shown in case the master reads or writes information.

The tricky part of making the decision is to know whether a polling frame is sent or just data. By assuring that polling frames do not start with 0x04, the interrupt vector can make the difference between a normal and a polling frame. This explains the first byte of the self developed frame.
Chapter 3. Hardware design

3.3.4 Programmer code for the LMX9838

The LMX9838 module is shipped with factory settings. One of these settings is the Operation Mode. This parameter allows to use the module as a simple cable replacement module, enabling it to transmit raw data without framing. Although, since framing is implemented for the internal protocol, this parameter has to be turned off. Therefore, a special initialisation code was developed. The programmer code makes the LMX938 ready to use after running once. It also allows it to change the friendly name of the module and to turn off all other RFCOMM ports. The latter ensures that no other connection can be made if the module already has a SPP connection. Figure 3.13 illustrates this code.

3.3.5 Development

Descartes once said: „Divide each difficulty into as many parts as is feasible and necessary to resolve it.”

It is by this principle that the firmware implementation is developed. Working on each problem separately, gives the advantage of using smaller microcontrollers. It also makes it possible to develop the algorithms without a lot of helping materials. In this section it
Figure 3.13: Flow chart of the programming code
is explained which steps were followed to develop the entire firmware.

**Bluetooth Communication**

This subsystem was created to get to know the commands of the LMX9838 and how to control it. National Semiconductor offers different freeware programs to control the Bluetooth module. These are run on a computer connected to the LMX9838 via UART. As a remote device a cellular phone or a laptop equipped with the Bluetooth technology, and more specifically the SPP profile can be used.

It can be stated that this system is only used as learning material. In later developing stages, this system is often used to look at or simulate a Bluetooth response.

To obtain the communication, a routing circuit board, a Atmel AVR STK500 and a standard RS-232 serial cable are used. The routing circuit board is a PCB with the Bluetooth module soldered on it and is used to make the pins from the module easy to reach. The STK500 is a development board that includes power regulation and easy access to the data lines coming from the serial cable.

![Figure 3.14: Scheme of the Bluetooth communication testing system](image)

**UART Communication**

This testing subsystem is used to control the LMX9838, although its main focus is the UART communication.

The UART communication is developed using only a microcontroller and a computer.
Later, a Bluetooth module was added to implement flow control. This system is built using the routing circuit board explained in last section, an Atmel AVR STK500 and a standard RS-232 serial cable and a bread board. The system is sketched in figure 3.15.

![Figure 3.15: Scheme of the system used to develop the UART communication and LMX9838 controller](image)

The most challenging problem to be solved with this setup, is the correct initialization of the LMX9838. Many people encountered problems when booting the module. The module has trouble initializing its UART interface, and this apparently happens at a random basis.

However, thanks to this setup, enough tests could be run to understand the problem. The LMX9838 needs a certain amount of time between the power up and the start of the communication. If the module is powered off or reset during this time interval, the EEPROM will lock up and the module will malfunction.

Normally, the way to program the GPIOs of the microcontroller is to define which are inputs and which are outputs, to then set the output pins to the desired level. But, when the programming is done this way, with the reset line of the bluetooth module, a glitch is created in the small amount of time that elapses between the two commands. This will power on the module and power it off almost immediately. Hence, the EEPROM will end up in a lock up situation and the module will never become ready for communication.

This is solved by changing the order of the commands. First, the values of the pins are determined, and only after this, which pins act as input and which as output. This is allowed because of the way the ports are built. When the values are assigned, the pins are either high-Z or monitoring the line.
Chapter 3. Hardware design

I²C Communication

Implementing the I²C communication has been the most time consuming part of the whole development. Not because of the I²C itself, but because it cannot be done without a good internal protocol. A representation of the system can be seen in figure 3.16.

To acquire this setup, two Atmel AVR STK600, a self made board, two serial cables and two computers are used. The self made board is a simple solder board with pull-up resistances and a cable, which makes the I²C communication possible. The STK 600 is a development board similar to the STK500, but it supports the ATMega644P/324P too, and offers an on-board debug system via JTAG.

![Figure 3.16: Scheme of the system used to develop the I²C communication and the internal protocol](image)

When the protoboard is ready, one STK600 is replaced by the prototype. This makes the test more trustworthy and the development more comfortable.

System Integration

The scheme of the final design, as can be seen in figure 3.17 is created with different setups. At first instance, the protoboard is connected to a STK600. The development board has a microcontroller programmed to emulate the whole existing K-Light system, which makes it possible to monitor the data between the two microcontrollers thanks to the extra UART interfaces. This setup is well suited for debugging.

A second setup is achieved by replacing the STK600 by the prototype of the K-Light system. Now the only debugging possibility is the UART interface that is left open in the protoboard. This setup is suitable for testing the code with the real K-Light design, which can bring up problems that could not be seen on the emulation before, made by the single microcontroller on the previous setup.

The last step is to replace the protoboard by the final board. This makes debugging impossible, but makes the interference and speed settings tests more realistic.
Chapter 3. Hardware design

3.4 Physical Design

In order not to make the woman wearing the dress feel uncomfortable, the developed module has to be small. Before designing this final small module however, a prototype is built. This protoboard is made in the early phase of the system design, facilitating the debug and testing process. In this section the complete layout and components of the protoboard will be explained. Then the final board will be presented. To conclude, the encountered problems will be covered. This will help future designers to improve the existing design with a shorter development time.

The schematics and the PCB design of the system are made with EAGLE v5.10, the newest layout editor of CadSoft. CadSoft offers a freeware version of EAGLE that suffices the needs of the system.

3.4.1 Protoboard

Before the prototype board is manufactured, the debugging and testing is done using development boards, self made circuit boards and breadboards. These work fine for an early algorithm development, but they do not suffice when it comes to speed testing and interference measurements. This is due to the fact that the boards and interconnections introduce extra impedances. Hence, the main focus of the protoboard is not to build the system, but to give access to all the communication interfaces.

The first step is to look for passive components like capacitors, resistors, crystals, pin headers, etc. Ordering the components at the beginning is crucial for the PCB design because it is mandatory to know the exact dimensions of the components when making the board layout.

It is also important to have the components available within a reasonable amount of time.
Chapter 3. Hardware design

Sometimes, very cheap options can be found on internet shops, but it takes too long to ship them to Germany, Berlin. Because of the necessary paperwork that precedes the shipment, this is mostly the case with components coming from the United States.

Unfortunately, the design process is not straight forward. At first instance, by drawing the schematics of the circuit, new possibilities and problems are encountered. This means that more components are to be selected. Next, sometimes the exact desired component can not be found. This implicates that the circuit has to be slightly changed in order to keep the functionality with alternative components. Either how, looking for the glue electronics and drawing the schematics happens at the same time.

When drawing the schematics, it must be kept in mind which lines to leave open. These lines are e.g. the programming interfaces, the communication busses and some GPIOs from the controller. It is important to keep them accessible because of the function of this protoboard: to make it possible to monitor everything and to make the debugging process easier.

At the end, the lines with an access to the outside are the two UART interfaces with their flow control lines, the ISP and JTAG programming interfaces, the I²C interface, the reset line and one GPIO of the LMX9838. The rest of the passive components are the decouple capacitors, the crystals with their respective capacitors, pull up resistances, LEDs for debugging and two switches for input.

After the schematics are free from errors, and it is sure that nothing is forgotten, the board design can begin. The EAGLE package contains a lot of libraries with footprints of standard components. Anyway, not every component can be found there. This means that before even starting the layout, the missing footprints have to be drawn. In this case, the packages of the crystals, the LMX9838 and the ATMega644P have to be created.

At this point, all the footprints are drawn and the schematics proven. This means that the lay out can begin. The most important part of the board is the connection between the BT µCU and the Bluetooth module. Hence, the components are arranged in such a manner that these connections are as short as possible. A good placement of the components before starting to draw the connection lines, is the key to make the design routable. It was opted to let all the external pins at the side of the board, and to place the µCU in a central position. In this process, the autorouter offered in EAGLE is used. The most difficult connections to route are found by analyzing where the computer has troubles.
Chapter 3. Hardware design

autorouting. By iterating moving components and autorouting, an optimal position for the components can be found.

Once a good placement is achieved, the actual connection lines are drawn by hand. Using the autorouter at this point is not recommended. The computer does not know which lines are sensible to interference, which lines carry more or less current nor which lines are to be kept short.

Since a digital and an analog component are used, it is preferred to use two different ground planes, connected at a single point. The power lines are made thicker to reduce resistance. Figure 3.18 shows the the protoboard with components.

![Figure 3.18: The protoboard with the components placed. The red circle indicates an interruption of the powerline for testing purposes.](image)

3.4.2 Final Board

It takes approximately one month from the start of the design of a board before it is functional. Given the lack of time, it is not possible to wait until the testing phase is
Chapter 3. Hardware design

finished and the algorithm is fully functional to start designing the final board. Hence, some choices have to be made regarding the functionality of this board. These choices are reflected in the programming interfaces and passive components. It is chosen to use only the ISP programmer to contact the microcontroller. All LEDs and buttons are left out of the design and an open I/O pad is placed instead.

Thanks to the experience recollected on the design of the protoboard, some designing steps could be made faster and better than before.
To start, the schematics have to be changed just slightly. More specifically, all the unused components are left out. Moreover, only the footprint of the programming interface has to be drawn.
Next, the search for passive components was less intensive. Also, this time it is chosen to use smaller packages to make the overall design more compact.
At last, the placement of the components is already at a given optimum. To achieve an optimum for the smaller design, some of the I/O lines from the microcontroller are changed in order to reduce the amount of vias and hence making the connections shorter.

The design is not without restrictions however. It is obvious that the design has to be as small as possible. Also, no sharp edges are allowed, as this can damage the textile. But the most important restriction regards the mechanical stability. In order to attach the board solidly to the dress, some pads have to be inserted at the bottom layer of the PCB. These also carry the four connection lines of I²C and the open I/O. In order to make the design reliable, a certain redundancy has to be added. This is why these connection lines are duplicated, resulting in at least ten pads. However, in order to achieve the highest stability possible, fifteen pads were used, placed symmetrically, they are clearly visible on figure 3.19.
This is the biggest challenge of all, because the two ground planes ought to be connected at only one single point. By inserting the pads and by routing them on the bottom layer, the ground planes could get pierced, which might result in some components with a floating reference. To avoid this, the upper layer has to be used as much as possible. Unfortunately this results in a problematic routing, as almost all the connections needed for this system, are situated on the same side of the microcontroller.
3.4.3 Placement of components

Most of the hindrances found in the whole physical designing process, are encountered in the actual placing of the components on the PCB.

The placing of the components on the protoboard is used as an experiment for the end design. The first step is to make a template out of a copper sheet with a thickness of $17\mu m$. Holes are burned in this copper sheet, wherever there is no solder stop layer on the PCB and a component should be glued on the board. With the help of this template, SbAgCu solder paste is applied on the protoboard.

A pick and place machine is used to put the components on the protoboard. Afterwards, everything goes in the reflow oven.

Five boards were mounted, but only one worked. The problem was a short circuit between ground and power. At first instance, it was thought that because of the thickness of the solder paste the short circuit was introduced. So next, a copper sheet with $35\mu m$ was used.

As a result the soldering looks better under the microscope, but the board still gives a short circuit. Under the X-ray machine no explicit failures could be seen, not in the PCB, nor on the solder pads.
The design was double checked to make sure that no design error was made, but after making one functioning module, the reason could be found. It appeared that the LMX9838 module is very sensitive for the temperatures used in the profile of the reflow oven. Since there were no boards left to mount, the next test would be on the final board.

For the final board, the solder paste was changed to allow soldering with lower temperatures. A SnBi paste was used at first. The paste is made by grinding the alloy and mixing it with flux, which makes it spreadable. The problem here was that the metal grains were too big for the small size of the BT $\mu$CU pads. This makes possible short circuits even before the board is sent into the oven. This problem was solved using a SbPb soldering paste. However, in the future other options should be explored, as for commercial applications only lead-free soldering pastes are allowed. Figure 3.20 shows the result.

Figure 3.20: Het bestukte eindontwerp.
Chapter 4

Mobile phone application

The goal of the developed communication system is to add interactivity to the already existing K-Light system. By allowing the user to connect with the dress via Bluetooth, parameters like the LED brightness or the animation mode can be changed. This is done with the help of a mobile phone, for which a unique application was developed.

The communication with the dress happens over Bluetooth via a serial port emulation. This sets a first requirement for the mobile phone, which, as a consequence, has to support the SPP profile. A touchscreen device was desired to add a fancy flare to the whole system. For developer’s purposes it is helpful to have an emulator to test an application without the need for an actual mobile phone. Because of the available emulators, a Nokia X6 has been chosen. This device runs on Symbian OS, which is world wide the most popular mobile operating system, running on hundreds of millions of devices, which means it is even more popular then Google’s Android and Apple’s iOS.

The program has been developed with the help of Netbeans IDE, which integrates the Nokia emulators and eases the design of the graphic interface. To facilitate the development and test the functionality of the written application, this interface is initially kept very simple. When the application works like it should, the simple user interface can be replaced by a more fancy one. With the help of SVG files, the application can be enriched with self-designed backgrounds and buttons can be used in the application. The current version of this IDE is Netbeans 6.9.

The application is written in Java Micro Edition (Java ME), a compact version of Java developed for constrained devices. There are two possible configurations of Java ME. The connected limited device configuration, or CLDC is intened to run on devices with a limited processing power and battery life. The current version, CLDC 1.1 was released in
Chapter 4. Mobile phone application

2002. For devices with a higher memory capacity, such as smartphones, the connected device configuration, or CDC, was developed. This configuration has however never reached its full potential, and is not very relevant anymore.

An application written in JavaME for CLDC devices is called a MIDlet, and has a prescribed lifecycle. At any point of time, a MIDlet can be in an active, paused or destroyed state. A MIDlet can e.g. enter paused state when the application is running or active and a call comes in or a message is received.[10] [11] [12]

At first instance, the choice for the Java ME platform was made because of its high-level character. This implies that a program written in the Java language should work independent from the used cell phone, its hardware, and even its operating system. More specifically this would mean that the MIDlet written for the chosen Nokia phone, should also work properly on e.g. any Samsung or Sony Ericsson device. Unfortunately, experience has taught otherwise. When trying the developed Java ME application on a Palm Pre telephone, the application wouldn’t run. Research showed that Java ME is not supported on several types of high-end smartphones, but still is the perfect solution for the so called in-between cell phones. Listed under this group are the mobile phones which can do more than just call or send messages, but which aren’t real smart phones either. Anyhow, Java ME is still the widest spread application platform in the world.

4.1 Screen Flow

The actual mobile phone program is used to set up the connection with the dress, and to send and receive data to and coming from the LMX9838. In this section it is explained how the application goes through the connection set up, and how the user can communicate with the LMX. It is assumed no errors whatsoever occur. The error situations are covered in the next section, which handles the communication protocol.

At first, the device makes a connection to the device it was connected to the last session. If the application is run for the first time however, an inquiry is started to look for available Bluetooth devices. This is not the regular start up mode, as the device discovery tends to be very time consuming.

The inquiry runs in the background, while a wait screen with the information ”..discovering devices in reach...” is displayed. More details on how the establishment of the Bluetooth connection happens, are given in paragraph 4.2.1. When the device discovery succeeds, a list with the user friendly names of the discovered devices is shown on the screen. Next, the user can choose which device he or she wants to connect to. During the establishment of the connection another wait screen displaying ”..Establishing Connection...” appears.
Chapter 4. Mobile phone application

If the device is able to connect to the Bluetooth module of the dress, the display of the mobile phone is changed into the main menu. This menu offers three choices and an exit button. For instance, the user can choose to change the parameter settings. With the help of two sliders, the sensor sensibility and the light intensity can be adapted. For the intensity this gives the effect of a dimmer. The user can also choose to change the animation mode. This option was implemented to satisfy demonstration needs. The user can either put all lights on or off, or let the LEDs shine in an animation mode. At last, the user can request the system info of the K-Light system. An overview of the above described screen flow is given in figure 4.1.

4.2 Communication with the remote system

The communication between the mobile phone and the LMX9838 happens over a Bluetooth RFCOMM channel. In this section it is clarified how this communication channel is established, how information is exchanged over this channel, and how possible connection losses are handled.

4.2.1 Java class BTManager

With the help of the Java API JSR-82 the Java class BTManager handles the realization of an RFCOMM channel between the mobile phone and the selected device. The BTManager also defines how the communication over this channel will happen. In the following sections only abstracts of the code are given and explained, while the complete BTManager class can be found in the appendix, B.4.2.

![Figure 4.1: Simplified flow for the first execution of the MIDlet.](image-url)
Chapter 4. Mobile phone application

Device discovery

The first thing to do is to initiate the device search. This is done by accessing the local Bluetooth module, the mobile phone with the method LocalDevice.getLocalDevice(). If the Bluetooth controller of the mobile phone is not turned on, this method will throw a BluetoothStateException and the user will be asked to activate the phone’s Bluetooth feature.

To start the search for all devices in general or limited discoverable mode, Discovery.GIAC should be passed to the startInquiry() method, along with an object that has the interface BluetoothListener implemented. The interface BluetoothListener makes sure the method deviceDiscovered() is implemented. A device can only support one device search at a time.

LocalDevice local = LocalDevice.getLocalDevice();
this.agent = local.getDiscoveryAgent();
this.agent.startInquiry(DiscoveryAgent.GIAC, this);

Whenever a device is found during the inquiry, the method deviceDiscovered() is called. This method is initially empty, and has to be implemented by the software developer. Every discovered device is added to a vector called availableDevices. The user friendly name is saved in another vector, used to visualize the discovered devices. Unfortunately, this can be a timeconsuming action, as in order to obtain this user friendly name, the local device already has to make a link with the remote device. Suppose the discovery agent is not able to find out this name, then the device’s unique Bluetooth address will be saved in deviceNames.

public void deviceDiscovered(RemoteDevice device, DeviceClass cod) {
    availableDevices.addElement(device);
    try {
        deviceNames.addElement(device.getFriendlyName(false));
    } catch (IOException e) {
        te.errorHandler("User friendly name not found: "+e.getMessage());
        deviceNames.addElement(device.getBluetoothAddress().toString());
    }
    return;
}

When the scanning of the area for discoverable devices is finished, the method inquiryCompleted() is called. From this moment, a list with the reachable devices can be displayed on the mobile’s screen.
Chapter 4. Mobile phone application

Service discovery and connection establishment

When the end user wishes to establish a connection with a selected device, a wait screen is displayed with the message “establishing connection”. On the background the application calls the BTManager method \texttt{makeConnection()}. This method tries to find out the right connection URL and uses this URL to connect to the remote device. But first, the remote device is for desired services past through a UUID array, with \texttt{searchServices()}. The remote device will only be scanned for services supported with SPP, which have the UUID equal to 0x1101, an SIG reserved value.

Analogue to the device discovery inquiry, the service discovery has a callback method \texttt{serviceDiscovered()}, which is called each time a service is discovered. After performing the service search, the right connection URL can be retrieved with \texttt{getConnectionURL()}. This method requires 2 arguments. The first specifies the security measurements for the connection. The second argument is a boolean and defines whether the connection is established with the local device acting as a master or a slave. Here the boolean is false, which indicates that the mobile phone will serve as the Bluetooth slave. This can be confusing, as the phone is the one that initiates the connection.

Once the phone has obtained the correct connection URL, it opens the connection. At the same time, also the input and output stream are established upon this connection. All communication with the LMX9383 will happen over these streams.

```java
public void makeConnection (int index) throws ConnectionNotFoundException{
    try {
        if (connTo == null) {
            RemoteDevice toConnect =
                (RemoteDevice) this.availableDevices.elementAt(index);
            UUID[] uuidList = {new UUID(0x1101)};
            agent.searchServices(null, uuidList, toConnect, this);
            while (!serviceComplete) {
                synchronized (this) {
                    this.wait(5);
                }
            }
        }
        if (this.record != null) {
            connTo = record.getConnectionURL(0, false);
        }
    }
```
Chapter 4. Mobile phone application

```java
} else {
    te.errorHandler("This device does not support SPP");
}

establishConnection(connTo);
return;
} catch (IOException e) {
    te.connBroken("Exception occured:" + e.getMessage());
} catch (InterruptedException e) {
    te.connBroken("Exception occured:" + e.getMessage());
}
```

```java
public void establishConnection(String URL){
    try {
        this.connection = (StreamConnection) Connector.open(URL);
        this.in = connection.openInputStream();
        this.out = connection.openOutputStream();
        conn = true;
    } catch (IOException e) {
        te.connBroken("The connection cannot be (re)established. \
                      Exception occured:" + e.getMessage());
    }
}
```

**Communication protocol**

Once the connection between the mobile phone and the dress is established, an input and an output stream are opened. The outputstream can be used to set the parameter values, or light modes, while the input stream can for instance be used to obtain the initial values of the parameters.

In figure 4.2(a) it is shown what an outputstream should look like before it is flushed, or sent to the LMX. The first byte defines whether the user wants to write info to, or obtain info from the K-Light system. The next byte is used to specify what the user specifically wants to read or write. There are three defined values, one for parameters, one for light modes, and one for system information. The third byte represents the length of the stream.

51
Chapter 4. Mobile phone application

to be flushed. At last there is the data part, or the payload. It is clear that when \texttt{write()} is called by the \texttt{read()} method, so when the user wants to obtain data from the LMX9838, this field will be empty. Hence, the output stream will then contain three bytes.

A scheme of an inputstream is given in figure 4.2(b) Until this acknowledgement is received, the application blocks the phone.

![Scheme of an outputstream](image)

(a) Scheme of an outputstream

![Scheme of an inputstream](image)

(b) Scheme of an inputstream

Figure 4.2: Prescribed forms for in- and output streams

```java
public boolean write(byte[] data) {
    try {
        out.write(new byte[]{this.WRITE, data[0], toByte((data.length + 2))});
        for (int i = 1; i < data.length; i++) {
            out.write(data[i]);
        }
        out.flush();
        byte[] check = new byte[2];
        in.read(check);
        if (check[1] == 0x05) {
            return true;
        }
    } catch (IOException e) {
        te.connBroken("IOException caught: " + e.getMessage());
    } catch (Exception e) {
        te.errorHandler("Exception caught: " + e.getMessage());
    }
    return false;
}
```

If the user wishes to obtain information coming from K-Light, the method \texttt{read()} is called. This method uses the already described function \texttt{write()} to notify the remote device of its request, e.g. the initial values of the parameters. After receiving the desired information,
Chapter 4. Mobile phone application

this method returns the read information through a byte array. If an error would have occurred, `read()` will return an empty byte array.

```java
public byte[] read(byte request){
    try {
        out.write(new byte[]{this.READ, request, 3});
        out.flush();
        int length = in.read();
        if (length > 0) {
            byte[] data = new byte[length - 1];
            in.read(data);
            return data;
        } else {
            te.errorHandler("Not able to read the request");
        }
    } catch (IOException e) {
        te.connBroken("IOException caught: " + e.getMessage());
    } catch (Exception e) {
        te.errorHandler("Exception caught: " + e.getMessage());
    }
    return new byte[0];
}
```

4.2.2 Connection issues

Although the SIG guarantees its Bluetooth technology to be robust, and to offer a quality of service, Bluetooth remains a wireless technology. This means that the connection between two devices can be lost at any time, due to intense traffic on the wireless channel for instance.

**The remote device cannot be reached**

There are two situations in which a particular remote device could be unreachable. The first is when the application is just started, the second occurs when a connection loss was experienced. This situation is handled in the next section.

When the MIDlet is started, the mobile phone tries to establish a connection with the device it was connected to the last time. That is, if the application has run before, as
mentioned above. If however, this device is out of reach, or simply has not turned on its Bluetooth module, there is no chance of succeeding. A simple error screen with the information that the attempt to connect to this remote device failed, will be shown to the user. He or she is then offered two choices. He can either try again, or perform a device search. The option to try again comes in handy when the user e.g. forgot to enable the bluetooth module of the remote device. If the user wants to connect to another device, a new device search is the preferable option.

A connection loss occurred

It is clear an error will occur whenever the application will read or write something after the connection was lost. To inform the user about this loss, a special error form will be displayed where the choice is offered to either re-establish the connection to the same device, or to perform a device search. When the user goes for the first choice, the device will try to reconnect with the Bluetooth module of the dress with the connection URL obtained before. This approach will only succeed if the SPP port of the LMX3938 is still available.

However, when the K-Light is out of reach, or because the Bluetooth module is out of batteries, the SPP port will not be reachable anymore. So, the mobile phone will then perform a new device search, comparable to the device search at the first execution of the MIDlet, as shown in figure 4.1. Figure 4.3 shows the actions that are taken when a connection loss is detected.
Figure 4.3: Flow chart for connection loss.
Chapter 5

Conclusion

In 2008, the K-Light dress was created in order to demonstrate the newly developed SCB technology. However, the dress has a major drawback. Once the animation is programmed and the system is encapsulated, the system parameters cannot be changed anymore. A module is added to the original K-Light system in order to circumvent this issue.

The extension for the original K-Light system, can remotely adapt the animation mode to the user’s wishes. With the use of a mobile phone as a remote control, the developed application is able to change the LEDs’ intensity, and the sensor’s sensitivity. Also, all the LEDs can be turned on or off, or they light in the animation mode. At last, the user can request system information about the K-Light system. The user gets informed whenever the connection between the dress and the cellular phone fails.

The complete system is intended for demonstration purposes. The dress illustrates how the SCB technology can be used in everyday applications.

The compatibility of the developed system should not be restricted to the K-Light system though. Because of its minimal size, not only the wearer’s comfort is assured, but it also makes the module easy to place on already existing systems. Moreover, the extra module was made as modular as possible, in order to allow the design to be integrated to other systems without having to make major adjustments.

The Bluetooth technology was chosen because its wide spread, and nowadays the majority of cellular phones is equipped with it. As recent Bluetooth modules combine a programmable microcontroller with the Bluetooth technology, even a smaller size for the communication module can be achieved. Moreover, with the release of Bluetooth v.4, the Bluetooth technology is even more qualified to serve as a wireless control channel, as it combines a very low power consumption with a high enough datarate.

These recent developments imply a promising future for Bluetooth enabled smart textiles.
Appendix A

Manual Remote Control

A.1 Installing the application on your phone

The first step to use your mobile phone as a remote control for K-Light is of course to install the application on the mobile phone of your choice. As mentioned in chapter 1, the application is developed for Symbian phones, that run JavaME applications, and have the SPP Bluetooth profile implemented. For Nokia phones, it can be found whether or not a mobile phone supports these features on [13]

During the development process for this application, only Nokia tools were used. Therefore, only the required steps to install the application on a Nokia phone will be described in this manual. For mobile phones from other brands, it is recommended to check the brand’s homepage.

When the selected phone satisfies the needs for this application, the application can be installed on the mobile phone. In order to do so, the program Nokia PC Suite should be installed on your computer. This program is available at the Nokia Website, and is free to download.[13] With the help of Nokia PC Suite it is a piece of cake to download and install the application on your phone. On the welcome screen of this program, a mobile phone, and several options, amongst which install applications, are displayed, as can be seen in figure A.1 Upon clicking on the icon for install applications, a new program called application installer will start.

First, it is up to the user to choose how to bring the application through the phone. This can either be done with the help of a USB cable, delivered with the phone, or with Bluetooth, and so on.

The easiest way to perform this transfer, is via a USB cable.

Once the phone and the computer or connected with each other, the user has to select
Figure A.1: Screen shot of NokiaPCSuite
Appendix A. Manual Remote Control

PCSuite on the mobile phone’s screen.
Now, the phone is accessible by PC Suite, and the already installed applications are then displayed on the *My Phone* part in application installer, as illustrated in figure A.2.

To install the application to the phone, the user browses for the .jar file on the *My Computer* side of the application installer, as shown in figure A.2. After selecting this file, one click on the arrow between the two sides will start the application process on the phone.

![Figure A.2: Screen shot of Nokia Application Installer](image)

On the mobile’s screen an alert is displayed to question the user whether or not this application can be installed on the mobile phone. Press OK.

Because this application does not come with a certificate of an authorized application seller, an alert will be shown to the user that this application might be harmful to your phone. Simply ignore this message, and press CONTINUE.

Next, it is asked where the application should be installed to. It is up to the user to decide where. When the installation succeeds, the user can choose to run the application. In order to start the application, press START.
Appendix A. Manual Remote Control

A.2 Connecting to K-Light

When the application is started for the first time, the mobile phone automatically starts searching for other devices. When this device discovery is completed, all detectable devices in reach of the phone’s Bluetooth module, should be detected, and then displayed on the mobile’s screen. If the mobile phone was able to obtain the user friendly name of these remote devices, they are listed with this name.

The user can then choose with which device he wishes the mobile phone to be connected to. This means, that this phone could be used as a remote control for different K-Light, or other smart textile systems. When the desired device is selected, press connect to establish the connection. When the connection is realized, the main menu is displayed, along with an alert saying which device the phone is connected to.

If this device, for any reason might not be the right device, the user can choose to perform a second device search, by pressing NEW SEARCH at the main menu options.

A.3 Changing the parameters

To change the parameters of the K-Light system, choose PARAMETER SETTINGS in the main menu. Then, two sliders are presented. One for the brightness of the LEDs, and one for the sensibility of the accelerometer. Simply slide to change the values for the desired parameter.

A.4 Changing the animation modes

In order to change the animation mode, press K-LIGHT LIGHTNING PROGRAMS in the main menu. A choice list will be displayed, enabling the user to select a desired mode. Upon selecting one mode, the dress will automatically update its lightning mode. There are three available modes, all LEDs off, all LEDs on, or animation mode.

A.5 Requesting system information

To get information about the version number of the K-Light system, and possible errors, press SYSTEM INFORMATION. A textfield containing the desired info will be displayed.
Appendix B

Source Code Remote Control

B.1 MIDlet: RemoteControl.java

B.1.1 Functionality

The class RemoteControl is the actual ‘brain’ of the application. This class extends the MIDlet class, which means that it has to implement the prescribed MIDlet lifecycle. This requires the functions startMIDlet(), pauseMIDlet() and resetMIDlet() to be present.

Furthermore, this class organizes the whole application flow. Therefore the class implements the two interfaces CommandListener and ItemCommandListener, and the according method commandAction().

B.1.2 Code

```java
public class RemoteControl extends MIDlet implements CommandListener,
       ItemCommandListener, Handler, ItemStateListener {

    private Graphic graphic = new Graphic(this);
    private BTManager link = new BTManager(this);
    public Parameters pars = new Parameters() ;
    public ConnStore store = new ConnStore(this);
    private boolean midletPaused = false;
    private int waitType;
    public static int DEVICE_DISCOVERY = 1;
    public static int MAKE_CONNECTION = 2;
    public static int RETRIEVE_CONNECTION = 3;
```
private SimpleCancellableTask connectionTask;
private SimpleCancellableTask discoverTask;
public Command exitMIDlet, backMIDlet, okgo;
public Command newSearch;
public Command makeConn;

public RemoteControl() {} {

public BTManager getBT() {
    return link;
}

public Parameters getPars() {
    return pars;
}

public void errorHandler(String tekst) {
    graphic.makeAlert(tekst);
    // graphic.getErrorForm2(tekst + "\n");
}

public void parrError(String what) {
    graphic.getErrorForm2("Something went wrong and the " + what +
    "could not be changed. \
", new Command[]{this.backMIDlet});
}

public void connBroken(String tekst) {
    graphic.switchDisplayable(null, graphic.getErrorForm(tekst + "\n"));
    link.conn = false;
}

public void itemStateChanged(Item item) {
    if (item == graphic.modes)
        writeProgram();
    else
        writeParameters();
}
Appendix B. Source Code Remote Control

}.

public void removeList(){
    if(graphic.deviceList!=null){
        graphic.deviceList.deleteAll();
        graphic.deviceList = null;
    }
    link.availableDevices.removeAllElements();
    link.deviceNames.removeAllElements();
}

public SimpleCancellableTask getDiscoverTask() {
    SimpleCancellableTask task = new SimpleCancellableTask();
    task.setExecutable(new org.netbeans.microedition.util.Executable(){

        public void execute() throws Exception {
            link.deviceSearch();
            // write task-execution user code here
        }
    });
    return task;
}

public SimpleCancellableTask makeConnectionTask(int index) {
    final int chosen = index;
    SimpleCancellableTask task1 = new SimpleCancellableTask();
    task1.setExecutable(new org.netbeans.microedition.util.Executable(){

        public void execute() throws Exception {
            link.makeConnection(chosen);
        }
    });
    return task1;
}
Appendix B. Source Code Remote Control

```java
public void commandAction(Command command, Item item) {
    switch (command.getCommandType()) {
    case Command.SCREEN:
        // Go to the selected menu
        if (item.getLabel().equals(graphic.mainMenuStrings[0])) {
            // Go to the parameter settings menu:
            graphic.switchDisplayable(null, graphic.getParamMenu());
        } else if (item.getLabel().equals(graphic.mainMenuStrings[1])) {
            // Go to the Lightning programs menu:
            graphic.switchDisplayable(null, graphic.getProgramMenu());
        } else {
            graphic.switchDisplayable(null, graphic.makeInfo(
                link.read(link.SYSTEMINFO).toString()));
        }
        break;
    case Command.OK:
        // Apply the chosen programm-mode
        if (item != graphic.modes) {
            updateFirmware();
            graphic.switchDisplayable(null, graphic.getProgramMenu());
        }
        break;
    }
}

public void commandAction(Command command, Displayable display) {
    // if (command == Alert.DISMISS_COMMAND) {
    //     graphic.switchDisplayable(null, graphic.getMainMenu());
    //     return;
    // }
    //
    if (command == SplashScreen.DISMISS_COMMAND) {
        runApplication();
    }
}
```
Appendix B. Source Code Remote Control

if (command == WaitScreen.FAILURE_COMMAND) {
    switch (waitType) {
    case 1:
        // coming from deviceDiscovery
        commandAction(exitMIDlet, display);
        break;
    case 2:
        // coming from makeConnection
        errorHandler("Failure to connect to an SPP port");
        deviceDiscSuccessful();
        break;
    case 3:
        removeList();
        runApplication();
        break;
    }
    return;
} else if (command == WaitScreen.SUCCESS_COMMAND) {
    switch (waitType) {
    case 1:
        // coming from deviceDiscovery
        deviceDiscSuccessful();
        break;
    case 2:
        // coming from makeConnection
        this.store.updateStore(new Connectable(link.friendly,
                link.connTo));
        graphic.switchDisplayable(null, graphic.getMainMenu());
        break;
    case 3:
        if (link.conn) {
            graphic.switchDisplayable(null, graphic.getMainMenu());
        } else {
            removeList();
            waitType = DEVICE_DISCOVERY;
            graphic.switchDisplayable(null, graphic.makeWait(
        }
Appendix B. Source Code Remote Control

"..Device Discovery..", getDiscoverTask());

} break;
}
return;

if(command == backMIDlet) {
if (display == graphic.errorForm) {
    waitType = RETRIEVE_CONNECTION;
    graphic.switchDisplayable(null, graphic.makeWait(
        "..Reopening Connection..", connectionTask));
    return;
} else if (display == graphic.parMenu) {
    Parameters current = new Parameters();
    byte[] bytearray = getBT().read(getBT().PARAMETERS);
    current.setIntValue(bytearray[1]);
    current.setSensValue(bytearray[0]);
    if (!(current.getIntValue()== pars.getIntValue()) ||
        !(current.getSensValue() == pars.getSensValue())) {
        this.parrError(" parameter settings ");
    }else {
        graphic.switchDisplayable(null,graphic.getMainMenu());
    }
} else if (display == graphic.progMenu) {
    Parameters current = new Parameters();
    byte[] bytearray = getBT().read(getBT().PROGRAMMODE);
    current.setMode(bytearray[3]);
    if (!(current.getMode()== pars.getMode())){
        this.parrError(" animation mode ");
    }else {
        graphic.switchDisplayable(null,graphic.getMainMenu());
    }
}

else {
    switch (command.getCommandType()) {

}
Appendix B. Source Code Remote Control

// Go back to the previous displayable (backMIDlet)
case Command.BACK:
    graphic.switchDisplayable(null, graphic.getMainMenu());
    break;

// Exit the MIDlet (exitMIDlet)
case Command.EXIT:
    exitMIDlet();
    break;

// Search again for available devices (newSearch)
case Command.SCREEN:
    removeList();
    waitType = DEVICE_DISCOVERY;
    graphic.switchDisplayable(null, graphic.makeWait(''
        ..Device Discovery..", getDiscoverTask()));
    break;

// Establish connection to spp port of the selected device
case Command.ITEM:
    waitType = MAKE_CONNECTION;
    connectionTask = makeConnectionTask(((List)
        graphic.getDeviceList()).getSelectedIndex());
    graphic.switchDisplayable(null, graphic.makeWait(
        "..Establishing Connection..", connectionTask));
    break;
}
}

public void runApplication(){
    Connectable lastConnected= this.store.getLastConnectable();
    if (lastConnected == null) {
        waitType = DEVICE_DISCOVERY;
        graphic.switchDisplayable(null, graphic.makeWait(".. Discovering devices ..", discoverTask));
    } else {

67
Appendix B. Source Code Remote Control

```java
link.establishConnection(lastConnected.getConnURL());
if (link.conn)
    graphic.switchDisplayable(null, graphic.getMainMenu());
else
    graphic.getErrorForm("The device to which the phone was previously connected to, is not available now\n");
}
}
public void writeParameters() {
    // to make sure the 'apply-operation' cannot be cancelled
    byte[] toWrite = new byte[] {link.PARAMETERS, this.pars.toByte(graphic.parMenuItems[0].getValue()),
        this.pars.toByte(graphic.parMenuItems[1].getValue())};
    boolean okay = link.write(toWrite);
    if (okay) {
        this.pars.setSensValue(toWrite[1]);
        this.pars.setIntValue(toWrite[2]);
    } else {
        connBroken("An error occurred while changing the settings");
    }
}

// The chosen lightning program will be installed
public void writeProgram() {
    /* where :
        * 0 == all LEDs off
        * 1 == all LEDs on
        * 2 == animation mode
     */
    byte mode = 0x00;
    switch (graphic.modes.getSelectedIndex()) {
        case 0:
            mode = pars.LEDSOFF;
            break;
        case 1:
            mode = pars.LEDSON;
```

68
Appendix B. Source Code Remote Control

```java
break;
case 2:
    mode = pars.ANIMATION;
    break;
}
if (link.write(new byte[]{link.PROGRAMMODE, mode})) {
    this.pars.setMode(mode);
} else {
    connBroken("An error occured while adapting the program");
}

public void update(){
    if(link.connTo!=null && link.friendly!=null){
        store.updateStore(new Connectable(link.friendly,link.connTo));
    }
}

public void deviceDiscSuccessful() {
    if (link.deviceNames.isEmpty()) {
        Command[] c = {newSearch, exitMIDlet};
        if (link.isOn) {
            graphic.getErrorForm2("No devices were discovered \n",c);
        } else {
            graphic.getErrorForm2("Please turn on your BT module \n",c);
        }
    } else {
        graphic.switchDisplayable(null, graphic.getDeviceList());
    }
}

public void exitMIDlet() {
    link.breakConnection();
    graphic.switchDisplayable(null, null);
    destroyApp(true);
    notifyDestroyed();
```
public void startApp() {
    if (midletPaused) {
        resumeMIDlet();
    } else {
        initialize();
        startMIDlet();
    }
    midletPaused = false;
}

public void pauseApp() {
    midletPaused = true;
}

public void destroyApp(boolean unconditional) {
}

private void initialize() {

    /*************** EXIT COMMANDS ***************
    exitMIDlet = new Command("EXIT", "EXIT PROGRAM", Command.EXIT, 0);

    /*************** BACK COMMANDS ***************/
    backMIDlet = new Command("BACK", null, Command.BACK, 0);

    /*************** OK COMMANDS ***************

    /*************** ITEM COMMANDS ***************
    makeConn=new Command("CONNECT", "CONNECT TO DEVICE", Command.ITEM, 0);

    /*************** SCREEN COMMANDS ***************
    // this one will be called by CommandListener(parameters)
    newSearch = new Command("NEW SEARCH", "SEARCH", Command.SCREEN, 0);
    // this one will be called by ItemCommandListener

Appendix B. Source Code Remote Control

    okgo = new Command("OK", "Go to selected menu", Command.SCREEN, 0);

    discoverTask = getDiscoverTask();
}

public void startMIDlet() {
    graphic.switchDisplayable(null, graphic.getSplashScreen());
}

public void resumeMIDlet() {
}
}
Appendix B. Source Code Remote Control

B.2 Visual Design: Graphics.java

B.2.1 Functionality

This class implements the user interface. IF, in possible future versions of the application a functionality is added to the system. It should be remembered to update the menus as well. This can easily be done by adding the desired feature to the String array corresponding to the right menu.

The main menu displays each element as a button. When a parameter is added, automatically a slider is added as well.

The different error forms are used to inform the user whenever the application fails.

B.2.2 Code

```java
public class Graphic extends GraphicUserInterface{

    private RemoteControl midlet;

    public List deviceList;
    private Form mainMenu;
    public Form parMenu;
    private TextBox information;
    public Form progMenu;
    public Form errorForm;
    private SplashScreen splashScreen;

    public String[] mainMenuStrings= {"Parameter Settings",
                                        "K-Light lightning Programs","System Information");
    public Item[] mainMenuItems;

    public String[] parMenuStrings= {"Sensibility acceleration sensor",
                                        "Global light intensity"};
    public Gauge[] parMenuItems;

    public String[] progMenuStrings= {"All LEDs off", "All LEDs on",
                                        "Animation"};
    public ChoiceGroup modes;
```
public WaitScreen makeWait(String message, SimpleCancellableTask todo) {
    WaitScreen wait = new WaitScreen(Display.getDisplay(midlet));
    wait.setTask(todo);
    wait.setText(message);
    wait.setCommandListener(midlet);
    return wait;
}

public Alert makeAlert(String message) {
    Alert alert = new Alert(message);
    alert.setTimeout(5000);
    alert.addCommand(midlet.newSearch);
    alert.setCommandListener(midlet);
    Display.getDisplay(midlet).setCurrent(alert);
    return alert;
}

public Displayable getDeviceList() {
    if (deviceList == null) {
        Command[] commands = {midlet.makeConn, midlet.exitMIDlet, midlet.newSearch};
        deviceList = new List("Devices discovered", Choice.EXCLUSIVE);
        for (int i = 0; i < commands.length; i++) {
            deviceList.addCommand(commands[i]);
        }
        deviceList.setCommandListener(midlet);
        for (int i = 0; i < midlet.getBT().deviceNames.size(); i++) {
            deviceList.append(midlet.getBT().deviceNames.elementAt(i).toString(), null);
        }
    }
    return deviceList;
}

public Displayable getMainMenu() {
    if (mainMenu == null) {

73
Appendix B. Source Code Remote Control

```java
Command[] commands = {midlet.okgo, midlet.exitMIDlet,
                     midlet.newSearch};

mainMenu = new Form("Main Menu");
for (int i = 1; i < commands.length; i++) {
    mainMenu.addCommand(commands[i]);
}
if (mainMenuItems == null) {
    mainMenuItems = new Item[mainMenuStrings.length];
    for (int i = 0; i < mainMenuItems.length; i++) {
        mainMenuItems[i] = new StringItem(null, "\n",
                                           Item.BUTTON);
        mainMenuItems[i].setLabel(mainMenuStrings[i]);
        mainMenuItems[i].setDefaultCommand(commands[0]);
        mainMenuItems[i].setItemCommandListener(midlet);
        mainMenu.append(mainMenuItems[i]);
    }
}
mainMenu.setCommandListener(midlet);
return mainMenu;
}

public Displayable getParamMenu() {
    Command[] commands = {midlet.backMIDlet};
    if (parMenuItems == null) {
        parMenu = new Form("Parameter Settings");
        for (int i = 0; i < commands.length; i++) {
            parMenu.addCommand(commands[i]);
        }
    byte[] bytearray = midlet.getBT().read(midlet.getBT().PARAMETERS);
    midlet.pars.setIntValue(bytearray[1]);
    midlet.pars.setSensValue(bytearray[0]);
    }
    int[] values = midlet.getPars().getValues();
    parMenuItems = new Gauge[parMenuStrings.length];
    for (int i = 0; i < parMenuItems.length; i++) {
        74
```
Appendix B. Source Code Remote Control

```java
parMenuItems[i] = new Gauge("\n", true, values[2 * i],
values[2 * i + 1]);
parMenuItems[i].setLabel(parMenuStrings[i]);
parMenu.append(parMenuItems[i]);
}
parMenu.setItemStateListener(midlet);
parMenu.setCommandListener(midlet);
else {
    int[] values = midlet.getPars().getValues();
    for (int i = 0; i < parMenuItems.length; i++) {
        // gauge: first maximum value, than current value
        ((Gauge) parMenu.get(i)).setValue(values[2 * i + 1]);
    }
}
return parMenu;
}

public Displayable getProgramMenu() {
    Command[] commands = {midlet.okgo, midlet.backMIDlet};
    if (modes==null) {
        modes = new ChoiceGroup("Available lightmodes", Choice.EXCLUSIVE);
        for (int i = 0; i < progMenuStrings.length; i++) {
            modes.append(progMenuStrings[i], null);
        }
        midlet.getPars().setMode(midlet.getBT().read(midlet.getBT().
PROGRAMMODE)[0]);
    }
    progMenu = new Form("Light Program Menu");
    progMenu.append(modes);
    progMenu.addCommand(commands[1]);
    progMenu.setItemStateListener(midlet);
    progMenu.setCommandListener(midlet);
    }
    modes.setSelectedIndex(midlet.getPars().getMode(), true);
    return progMenu;
}
```
Appendix B. Source Code Remote Control

```java
public Displayable makeInfo(String info) {
    if (information == null) {
        Command[] commands = {midlet.backMIDlet};
        information = new TextBox("System Information", null, 100,
                TextField.ANY | TextField.UNEDITABLE);
        information.setString(info);
        for (int i = 0; i < commands.length; i++) {
            information.addCommand(commands[i]);
        }
        information.setCommandListener(midlet);
    }
    return information;
}

public Form getErrorForm(String s) {
    if (errorForm == null)
        errorForm = new Form("Errors");
    errorForm.deleteAll();
    errorForm.append(s);
    errorForm.addCommand(midlet.backMIDlet);
    errorForm.addCommand(midlet.exitMIDlet);
    errorForm.setCommandListener(midlet);
    return errorForm;
    //    switchDisplayable(null, errorForm);
}

public void getErrorForm2(String s, Command[] commands) {
    Form orForm = new Form("Errors");
    orForm.append(s);
    for (int i = 0; i < commands.length; i++) {
        orForm.addCommand(commands[i]);
    }
    orForm.setCommandListener(midlet);
    switchDisplayable(null, orForm);
    orForm = null;
}
```
public void switchDisplayable(Alert alert, Displayable nextDisplayable) {
    Display display = getDisplay();
    if (alert == null) {
        display.setCurrent(nextDisplayable);
    } else {
        display.setCurrent(alert, nextDisplayable);
    }
}

public SplashScreen getSplashScreen() {
    if (this.splashScreen == null) {
        splashScreen = new SplashScreen(getDisplay());
        splashScreen.setTitle("Welcome");
        splashScreen.setCommandListener(this.midlet);
        splashScreen.setFullScreenMode(true);
        splashScreen.setText("K-Light Remote Control");
        splashScreen.setTimeout(2000);
        try{
            splashScreen.setImage(Image.createImage("/image.jpg"));
        }catch(java.io.IOException e){
        }
    }
    return splashScreen;
}

public Display getDisplay () {
    return Display.getDisplay(midlet);
}

public void exitMIDlet() {
    switchDisplayable (null, null);
    // midlet.destroyApp(true);
    midlet.notifyDestroyed();
}
Appendix B. Source Code Remote Control

B.3 Interface: Handler.java

B.3.1 Functionality

This interface is initially developed to make sure that communication between the class BTManager and RemoteControl is possible. In a later stadium also the ConnStore uses the Handler. The function are implemented in the class RemoteControl.

The class RemoteControl implements the interface Handler. By using this interface, both BTManager and ConnStore can be constructed with a Handler as argument, which enables both classes to inform RemoteControl in case an error occurs.

If the BTManager wants to use the in-and outputstreams when the connection has failed, an exception will be thrown and via connBroken() the reconnection mechanism is triggered. Whenever the LMX was not able to adapt the new parameter settings or light mode for some reason, parrError() is called, and the user is referred to an error screen. The function errorHandler() is used for other errors.

B.3.2 Code

```java
public interface Handler {

    public void connBroken(String tekst);

    public void parrError(String what);

    public void errorHandler(String tekst);

}
```

79
Appendix B. Source Code Remote Control

B.4 Class: BTManager.java

B.4.1 Functionality

The class BTManager is used to handle all the actions regarding the communication. Its most important functions are already discussed in the main part of this work.

When the application is exited, the method closeConnection() is called, and all streams and connections are closed. Because a simple casting to a byte does not always suffice to send the right number, a special method, toByte() has been implemented. This way, also streams with a big length can be sent, although for this application the longest stream to be flushed consists of only three bytes.

B.4.2 Code

```java
public class BTManager implements DiscoveryListener {

    public static final byte WRITE = 0x00; // 00
    public static final byte READ = 0x01; // 01
    public static final byte PARAMETERS = 0x12; // 12
    public static final byte PROGRAMMODE = 0x13; // 13
    public static final byte SYSTEMINFO = 0x14; // 14
    public static final byte UPDATE = 0x15; // 15
    public static final int BT_TIMEOUT = 15000; // in ms
    public Vector deviceNames = new Vector();
    public Vector availableDevices = new Vector();
    private DiscoveryAgent agent;
    private ServiceRecord record;
    private StreamConnection connection;
    private InputStream in;
    private OutputStream out;
    public String connTo;
    public String friendly;
    public boolean isOn;
    public boolean deviceComplete;
    public boolean serviceComplete;
    public boolean conn = false;
```
private Handler te;

public BTManager(Handler te) {
    this.te = te;
}

public void deviceSearch() {
    deviceComplete = false;
    try {
        if (LocalDevice.isPowerOn()) {
            isOn = true;
            LocalDevice local = LocalDevice.getLocalDevice();
            this.agent = local.getDiscoveryAgent();
            this.agent.startInquiry(DiscoveryAgent.GIAC, this);
            while (!this.deviceComplete) {
                synchronized (this) {
                    this.wait(this.BT_TIMEOUT);
                }
                if (!deviceComplete) {
                    agent.cancelInquiry(this);
                }
            }
        } else {
            inquiryCompleted(this.INQUIRY_ERROR);
        }
    } catch (Exception e) {
        te.errorHandler(e.getMessage());
        return;
    }
}

public void deviceDiscovered(RemoteDevice device, DeviceClass cod) {
    availableDevices.addElement(device);
    try {
        deviceNames.addElement(device.getFriendlyName(false));
    } catch (IOException e) {
    }
    } catch (IOException e) {

te.errorHandler("User Friendly name of discovered device could not be found: "+ e.getMessage());
    deviceNames.addElement(device.getBluetoothAddress().toString());
}

//if (cod.getMajorDeviceClass() == choice) {
   //}else deviceNames[discovered] = device.getBluetoothAddress();
   return;
}

public void inquiryCompleted(int discType) {
   deviceComplete = true;
   synchronized (this) {
      this.notify();
   }
}

public void servicesDiscovered(int transID, ServiceRecord[] servRecord){
   this.record = servRecord[0];
}

public void serviceSearchCompleted(int transID, int respCode) {
   serviceComplete = true;
   return;
}

public void makeConnection(int index) {
   friendly = (String) deviceNames.elementAt(index);
   try {
      if (connTo == null) {
         RemoteDevice toConnect = (RemoteDevice) availableDevices.
            elementAt(index);
         UUID[] uuidList = {new UUID(0x1101)};
         agent.searchServices(null, uuidList, toConnect, this);
         while (!serviceComplete) {
            synchronized (this) {
}}
Appendix B. Source Code Remote Control

```java
this.wait(5);
}

if (this.record != null) {
    connTo = record.getConnectionURL(0, false);
} else {
    te.errorHandler("This device does not support SPP");
}

establishConnection(connTo);
return;
}

public void establishConnection(String url) {
    try {
        this.connection = (StreamConnection) Connector.open(url);
        this.in = connection.openInputStream();
        this.out = connection.openOutputStream();
        conn = true;
    } catch (Exception e) {
        te.connBroken("Error with connection");
    }
}

public byte[] read(byte request) {
    try {
        out.write(new byte[]{this.READ, request, 3});
        out.flush();
        int length = in.read();
        if (length > 0) {
            byte[] data = new byte[length - 1];
```
Appendix B. Source Code Remote Control

```java
in.read(data);
return data;
} else {
    te.errorHandler("Not able to read the request");
}
} catch (IOException e) {
    te.connBroken("IOException caught: " + e.getMessage());
} catch (Exception e) {
    te.errorHandler("Exception caught: " + e.getMessage());
}
return new byte[0];

public boolean write(byte[] data) {
    try {
        out.write(new byte[]{this.WRITE, data[0],
                            toByte((data.length + 2))});
        for (int i = 1; i < data.length; i++) {
            out.write(data[i]);
        }
        out.flush();
    } catch (IOException e) {
        te.connBroken("IOException caught: " + e.getMessage());
        return false;
    } catch (Exception e) {
        te.errorHandler("Exception caught: " + e.getMessage());
        return false;
    }
    return true;
} return (byte) (i - 256);

public void breakConnection() {
    try {
        if (in != null) {
            this.in.close();
        }
    }
```
Appendix B. Source Code Remote Control

```java
if (out != null) {
    this.out.close();
}
if (connection != null) {
    this.connection.close();
}
} catch (Exception e) {
    te.errorHandler("Unable to close connection:"+e.getMessage());
}

public byte toByte(int i) {
    if (i < 128) {
        return (byte) i;
    }
    return i;
}
```
Appendix B. Source Code Remote Control

B.5 Class: Parameters.java

B.5.1 Functionality

With the help of this class, it is only once required to get the parameter information from
the remote device. After this, the parameter information is saved in a Parameter object,
where it is internally updated, with the methods setSensValue(), setIntValue() and
setMode(). The toInt() and toByte() methods are used to transform the integers to
bytes and vice versa, as a normal cast won’t suffice in many cases.

B.5.2 Code

```java
public class Parameters {

    public static final byte LEDSOFF = 0x00; //10
    public static final byte LEDSON = 0x01; //11
    public static final byte ANIMATION = 0x02; //12

    /*************** Fields ***************/
    private int sensMax = 255;
    private int sensValue = -1;
    private int intMax = 255;
    private int intValue = -1;
    private int lightMode = -1;

    /************* Constructor **************/
    public void Parameters() {
    }

    /*************** Getters ***************/
    public int getSensMax() {
        return sensMax;
    }

    public int getIntMax() {
        return intMax;
    }

}```
Appendix B. Source Code Remote Control

```java
public int getSensValue() {
    return sensValue;
}

public int getIntValue() {
    return intValue;
}

public int[] getValues() {
    int[] values = {sensMax, sensValue, intMax, intValue};
    return values;
}

public int getMode() {
    return lightMode;
}

/**************** Setters ****************/
public void setSensValue(byte sensor) {
    this.sensValue = toInt(sensor);
}

public void setIntValue(byte intensity) {
    this.intValue = toInt(intensity);
}

public int setMode(byte lightMode) {
    if (lightMode == this.LEDSOFF) {
        this.lightMode = 0;
        return 1;
    } else if (lightMode == this.LEDSON) {
        this.lightMode = 1;
        return 1;
    } else if (lightMode == this.ANIMATION) {
        this.lightMode = 2;
```

87
Appendix B. Source Code Remote Control

```java
    return 1;
}

    return -1;
}

class RemoteControl {

    public int toInt(byte b) {
        if (b < 0) {
            return (int) b + 256;
        }
        return b;
    }

    public byte toByte(int i) {
        if (i < 128) {
            return (byte) i;
        }
        return (byte) (i - 256);
    }
}
```
Appendix B. Source Code Remote Control

B.6 Class: Connectable.java

B.6.1 Functionality

This class is developed together with the ConnStore to make it possible to save the device the mobile phone was connected to the previous time. Different objects of the type Connectable, can be saved into a store, which makes it possible to store these Connectables over different MIDlet sessions. This implies that the time consuming device discovery is not always performed at start up of the MIDlet.

A Connectable represents a remote device and has three fields, namely the user friendly name, the obtained connection URL to the SPP port, a boolean, and an integer. The boolean is used to indicate whether or not this was the last device the mobile phone was connected to. The integer is used in the ConnStore, to identify each record in the store.

As a RecordStore can only store its objects like a bag of bytes, this class provides methods to convert a Connectable to a bytearray, or the other way around.

B.6.2 Code

public class Connectable {
    private final static int FIELD_FRIENDLY = 1;
    private final static int FIELD_URL = 2;
    private final static int FIELD_USED = 3;

    public final static int NO_ID = -1;

    /*************** Fields ***************/
    private String friendlyName;
    private String connectionURL;
    private boolean lastUsed = true;
    private int recordID;

    /************* Constructors *************/
    public Connectable(String friendlyName, String connectionURL){
        this.friendlyName = friendlyName;
        this.connectionURL = connectionURL;
        this.recordID = this.NO_ID;
    }
Appendix B. Source Code Remote Control

    }

public Connectable(byte[] bytes){
    fromBytes(bytes);
    recordID = NO_ID;
}

public Connectable(byte[] bytes, int ID){
    fromBytes(bytes);
    this.recordID = ID;
}

/*************** Getters *****************/
public String getFriendly(){
    return friendlyName;
}

public String getConnURL(){
    return connectionURL;
}

public int getID(){
    return recordID;
}

public boolean getLastUsed(){
    return lastUsed;
}

/*************** Setters *****************/
public void setUsed(boolean lastUsed){
    this.lastUsed = lastUsed;
}

public void setFriendly(String friendlyName){
    this.friendlyName = friendlyName;
}
Appendix B. Source Code Remote Control

public void setConnURL(String connectionURL){
    this.connectionURL = connectionURL;
}

public void setID(int recordID){
    this.recordID = recordID;
}

public byte[] toBytes(){
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    DataOutputStream dat = new DataOutputStream(out);
    try{
        dat.writeInt(this.FIELD_USED);
        dat.writeBoolean(lastUsed);
        dat.writeInt(this.FIELD_FRIENDLY);
        dat.writeUTF(getFriendly());
        dat.writeInt(this.FIELD_URL);
        dat.writeUTF(getConnURL());
    }catch(Exception e){
        return new byte[0];
    }byte[]toReturn = out.toByteArray();
    dat = null;
    out = null;
    return toReturn;
}

public void fromBytes(byte[] b){
    ByteArrayInputStream in = new ByteArrayInputStream(b);
    DataInputStream dat = new DataInputStream(in);
    try {
        while (true) {
            int field = dat.readInt();
            switch (field) {
            }catch(Exception e){
                return new byte[0];
            }
            byte[]toReturn = out.toByteArray();
            dat = null;
            out = null;
            return toReturn;
        }
    }

public void fromBytes(byte[] b){
    ByteArrayInputStream in = new ByteArrayInputStream(b);
    DataInputStream dat = new DataInputStream(in);
    try {
        while (true) {
            int field = dat.readInt();
            switch (field) {
            }catch(Exception e){
                return new byte[0];
            }
            byte[]toReturn = out.toByteArray();
            dat = null;
            out = null;
            return toReturn;
        }
    }
case FIELD_FRIENDLY:
    setFriendly(dat.readUTF());
    break;

case FIELD_URL:
    setConnURL(dat.readUTF());
    break;

case FIELD_USED:
    setUsed(dat.readBoolean());
    break;

}
Appendix B. Source Code Remote Control

B.7 Class: ConnStore.java

B.7.1 Functionality

The class ConnStore is used to save Connectables over several MIDlet instances. This implies that the sometimes very time consuming device discovery is not necessary anymore, which is a main advantage in demonstrations of the system. The storage is done by saving the store in a nonvolatile region of the device, e.g. flash memory, RAM memory backed up by a battery and so on. This implies that the MIDlet does not have immediate access to the store, as the store is located elsewhere on the phone’s memory.

A general RecordStore can store

B.7.2 Code

```java
public class ConnStore {

    public static final String storeName = "connectionURLs";
    private RecordStore connectablesStore= null;
    private Handler err = null;

    public ConnStore(Handler err){
        this.err = err;
    }

    private void openStore() throws RecordStoreException{
        if(connectablesStore == null){
            connectablesStore = RecordStore.openRecordStore(storeName,true);
        }
    }

    private void closeStore(){
        if(connectablesStore !=null){
            try{
                connectablesStore.closeRecordStore();
            }catch(Exception e){

            }
        }
        connectablesStore=null;
}
```

93
public void updateStore(Connectable conn) {
    try {
        this.openStore();
        Connectable available = getConnectable(conn.getFriendly());
        this.openStore();
        if (connectablesStore.getNumRecords() == 0) {
            conn.setID(this.connectablesStore.getNextRecordID());
            addConnectable(conn);
        } else if (available == null) {
            this.closeStore();
            this.connectablesStore.deleteRecordStore(storeName);
            this.openStore();
            conn.setID(this.connectablesStore.getNextRecordID());
            addConnectable(conn);
        } else {
            available.setConnURL(conn.getConnURL());
            byte[] b = available.toBytes();
            openStore();
            connectablesStore.setRecord(available.getID(), b, 0, b.length);
            closeStore();
        }
    } catch (RecordStoreException e) {
        err.errorHandler("Connectable is not added to the store");
        closeStore();
    }
}

private void addConnectable(Connectable conn) throws RecordStoreException {
    byte bytes[] = conn.toBytes();
    connectablesStore.addRecord(bytes, 0, bytes.length);
    err.errorHandler(" now added 
");
}
Appendix B. Source Code Remote Control

// to get a connectable by search for its friendlyName
private Connectable getConnectable(final String friendlyName)
    throws RecordStoreException {
    Connectable toReturn = null;
    openStore();
    if (this.connectablesStore.getNumRecords() > 0) {
        Connectable inList =
            new Connectable(this.connectablesStore.getRecord(1), 1);
        if (inList.getFriendly().equalsIgnoreCase(friendlyName)) {
            toReturn = inList;
        }
        inList = null;
    }
    closeStore();
    return toReturn;
}

// to get the connectable according the remote device
// that was last connected to the phone
public Connectable getLastConnectable() {
    Connectable toReturn = null;
    try {
        openStore();
        if (this.connectablesStore.getNumRecords() > 0) {
            toReturn =
                new Connectable(this.connectablesStore.getRecord(1), 1);
        }
        closeStore();
    } catch (RecordStoreException e) {
        err.errorHandler("Record could not be found");
    }
    return toReturn;
}
Bibliography


Bibliography


# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>K-Light, met de lichtjes duidelijk zichtbaar.</td>
<td>viii</td>
</tr>
<tr>
<td>2</td>
<td>Overzicht van het uitgebreide systeem.</td>
<td>ix</td>
</tr>
<tr>
<td>3</td>
<td>Het frame gebruikt voor de interne communicatie.</td>
<td>xi</td>
</tr>
<tr>
<td>4</td>
<td>Het finale ontwerp. Links is de bovenkant weergegeven, rechts de onderzijde met pads.</td>
<td>xii</td>
</tr>
<tr>
<td>5</td>
<td>Het bestukte protobord. De onderbreking van de voedingslijn is omcirkeld.</td>
<td>xiii</td>
</tr>
<tr>
<td>6</td>
<td>Het bestukte eindontwerp.</td>
<td>xiv</td>
</tr>
<tr>
<td>7</td>
<td>Eenvoudig overzicht van de schermen, als de applicatie voor de eerste maal wordt uitgevoerd.</td>
<td>xv</td>
</tr>
<tr>
<td>8</td>
<td>Ondermomen acties in geval van verbindingsoverlies.</td>
<td>xvii</td>
</tr>
<tr>
<td>1.1</td>
<td>The K-Light, with the shining LEDs visible.</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>The special structure of the copper lines. The left side displays the relaxed tracks, the right the elongated ones.</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>The process flow for sophisticated SCB manufacturing.</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>An extra copper area can be used to introduce an extra stiffness to a component area.</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>The K-Light system on an SCB before encapsulation.</td>
<td>9</td>
</tr>
<tr>
<td>2.5</td>
<td>IP Protocol stack with on the right the OSI Model.</td>
<td>11</td>
</tr>
<tr>
<td>2.6</td>
<td>Protocol stacks</td>
<td>12</td>
</tr>
<tr>
<td>2.7</td>
<td>Several piconets make up one scatternet.</td>
<td>14</td>
</tr>
<tr>
<td>2.8</td>
<td>Different devices are connected with each other through different kinds of channels.</td>
<td>16</td>
</tr>
<tr>
<td>2.9</td>
<td>The most important profiles of the Bluetooth Profile hierarchy</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>Scheme of the existing design.</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Scheme of the end design.</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>UART communication</td>
<td>23</td>
</tr>
</tbody>
</table>
List of Figures

3.4 Comparison table between the three possible Bluetooth modules .................. 25
3.5 UART frequency selection ................................................................. 26
3.6 LMX9838 communication package ....................................................... 26
3.7 Scheme of the developed frame ............................................................ 29
3.8 Scheme of the polling frame ................................................................. 29
3.9 Flow chart of the main code for the K-Light microcontroller .................. 31
3.10 Flow chart of the main code for the Bluetooth microcontroller ............. 33
3.11 Scheme of data transmission, and values read in the status register ........ 35
3.12 Scheme of data reception, and values read in the status register .......... 35
3.13 Flow chart of the programming code .................................................... 36
3.14 Scheme of the Bluetooth communication testing system .................... 37
3.15 Scheme of the system used to develop the UART communication and LMX9838 controller .......................................................... 38
3.16 Scheme of the system used to develop the PC communication and the internal protocol ................................................................. 39
3.17 Scheme of the end design for the extended K-light system ................. 40
3.18 The protoboard with the components placed. The red circle indicates an interruption of the powerline for testing purposes ..................... 42
3.19 The routing of the final design. On the right the pads are clearly visible .. 44
3.20 Het bestukte eindontwerp ................................................................. 45
4.1 Simplified flow for the first execution of the MIDlet ............................... 48
4.3 Flow chart for connection loss .............................................................. 55
A.1 Screen shot of Nokia PCSuite .............................................................. 58
A.2 Screen shot of Nokia Application Installer ........................................... 59