Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

A Protocol Gateway for the Internet of Things

Semester Thesis

Jonas Bächli
baechlij@student.ethz.ch

Computer Engineering and Networks Laboratory
Department of Information Technology and Electrical Engineering
ETH Zürich

Supervisors:
Felix Sutton
Romain Jacob
Prof. Dr. Lothar Thiele

June 10, 2016

Acknowledgements

I would like to thank my supervisors Felix Sutton and Romain Jacob for the excellent support, constructive criticism and late night emails.

In addition I would like to thank the Computer Engineering and Networks Laboratory for the granted access to a personal workplace and the measurement equipment.

And last but not least I want to thank Prof. Dr. Lothar Thiele and the Department of Electrical Engineering and Information Technology at ETH Zürich for giving me the opportunity to work on this thesis.

Abstract

The Internet of Things offers unique opportunities through the establishment of connections between before unconnected devices. This thesis investigates the feasibility of a protocol gateway which is able to connect the existing Low-power Wireless Bus with a smartphone using Bluetooth Low Energy. Its implementation had to satisfy in terms of low power requirements as well as in terms of bandwidth available. In order to develop this protocol gateway, suitable hardware had to be selected. To implement its functionality in software, Bluetooth Low Energy had to be explored while writing software on two platforms, on an embedded System-on-Chip as well as on an Android smartphone, providing a reliable connection for data transfers. In addition it involved a thorough evaluation, using the data obtained through measurements to derive a power state model, making it possible to estimate the power consumption of a Bluetooth Low Energy device by the means of its specifications. The developed system has a deep sleep power consumption of only 4.29 μW, enabling it to be available for over 600 days using a single coin cell.

Contents

Chapter 1

Introduction

1.1 Motivation

The Internet of Things (IoT) is not only a buzzword used by classical media and social media alike, it is also challenge to overcome by means of electrical engineering. The Internet of Things proclaims the idea of connecting everything, be it a watch, a coffee machine, a home or a car, by establishing connections in-between them, often by providing access to the internet, hence the name Internet of Things. The promise made by the Internet of Things is the simplification and optimisation of our lives; for example the electric car would warm up for its owner and then drive itself in front of its owners home, because the coffee machine sent information that the owner just had their second espresso and given the owners usual pattern, they should leave the house in 3.4 minutes. The price to pay for this luxury is the data provided by us, which, given the right algorithms may reveal more than what we are comfortable with, but this matter shall not be the subject of this thesis.

In order to enable devices to access each other, they first have to be connected to a suitable network which may have limitations depending on their environment as well as their requirements. While this network might be a common network, such as an Ethernet network or a WLAN of the IEEE 802.11 range, which could provide direct access to the internet, it may very well also be a network which is using a different protocol. In order to connect such a network to other networks and/or devices, one would use so called protocol gateways, which basically translate data from one protocol to another.

An example for a network using an uncommon protocol would be devices connected together with the Low-power Wireless Bus [2] (LWB), a communication protocol which makes use of constructive interference, turning a wireless multi-hop network into a wireless bus.

The goal of this thesis was to develop and evaluate a protocol gateway for the Low-Power Wireless Bus, establishing a connection to a smartphone, as shown in fig. 1.1. It needs to have fairly low power requirements, suitable for use to-

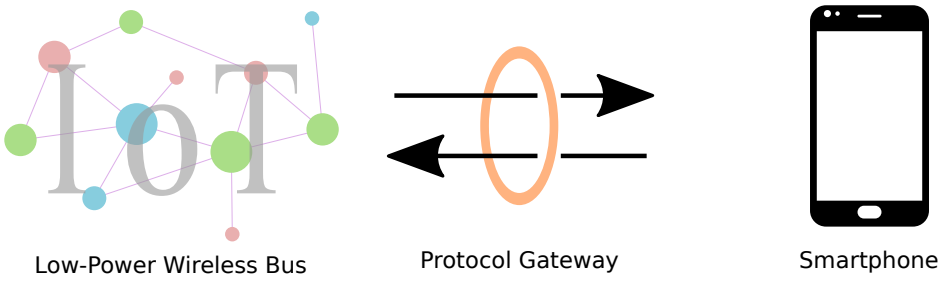

Figure 1.1: Protocol gateways enable new data transfer opportunities.

gether with the LWB and IoT in general. Bandwidth wise it should provide reliable moderate to low data rate. Intended applications include the use as a debugging interface in the field, for example in a LWB sniffer unit. The terms Low-power Wireless Bus (LWB) and Internet of Things (IoT) are used interchangeably throughout this thesis.

Due to the low power requirements of both the Low-power Wireless Bus as well as the Internet of Things in general it was necessary to find a suitable communication interface. Since all recent smartphones support Bluetooth Low Energy, this was the technology of choice. It was therefore necessary to find a platform which was BLE capable and could connect to the LWB.

1.2 Contributions

The contributions provided in this thesis can be summarised as follows:

- Establishing a suitable hardware architecture to form the proposed protocol gateway
- Selection of a BLE development platform
- The design and implementation of the system, spanning two platforms, a SoC on one side and an Android smartphone on the other
- An investigation of Bluetooth Low Energy, explaining relevant aspects for this thesis
- Evaluation, consisting of the following contributions:
  - The power analysis, the capture and interpretation of power traces
  - An estimation of the achievable bandwidth
  - The derivation of a power state model based on the obtained data, allowing the estimation of power consumption of a BLE device for given specifications

Chapter 2

Selection of Hardware Components

This chapter illustrates how the hardware components have been evaluated and selected, forming the system architecture of the protocol gateway.

2.1 System on Chip

In order to select a suitable commercially available BLE development board, we narrowed the search space by considering the following criteria:

- Software support
- Available peripherals
- Availability of examples for both the embedded as well as the Android platform
- Availability at distributors
- Price

After investigating the commercial products, specifically the CC2650 Launchpad by Texas Instruments, the BLE Pioneer Kit CY8CKIT-042-BLE by Cypress as well as the Multi-Sensor Development kit A20737A-MSDK1 by Anaren, we selected the Cypress BLE Pioneer Kit because of the capable and free IDE and the additionally included components.

The kit features two devices: a Cypress PSoC (Programmable System-on-Chip) 4 BLE and a Cypress PRoC (Programmable Radio-on-Chip) BLE device. The difference between the PSoC 4 BLE and the PRoC BLE lies in the additional analog front end included in the PSoC 4 BLE device, containing additional low-power opamps, low-power comparators and DACs. Since there was no use for

those components in this project, the decision was made to use the PRoC BLE module for this project, shown in fig. 2.1.

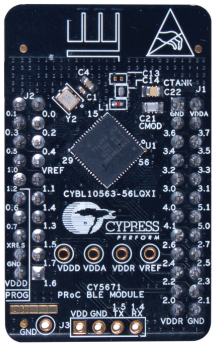

Figure 2.1: Photo of the PRoC BLE module by Cypress.

The PRoC BLE device is the CYBL10563 chip which provides the following features, as shown in fig. 2.2

- A 32-bit, 48 MHz ARM Cortex-M0 CPU Core
- A BLE Subsystem (BLESS) consisting of a hardware block and a proprietary software protocol stack
- 128KB flash
- 16KB SRAM
- Ultra-Low-Power support, including a 1.3-μA Deep-Sleep mode
- Two serial communication blocks (SCBs) which may be configured as I^2C, SPI or UART
- CapSense
- and more features

The commercial kit further included the baseboard, which serves as a programmer and a pin breakout as well as a BLE capable USB dongle with an additional programmable PRoC device on it. In combination with the Cypress software CySmart this USB dongle can be utilised to debug BLE devices.

Something notable about the PRoC and PSoC devices is that they offer extensive hardware customisation through programming it with a bitstream file, similar to what is known from CPLDs and FPGAs. With the PRoC BLE we used a lower end device, which is not as capable as the higher tier devices (even though

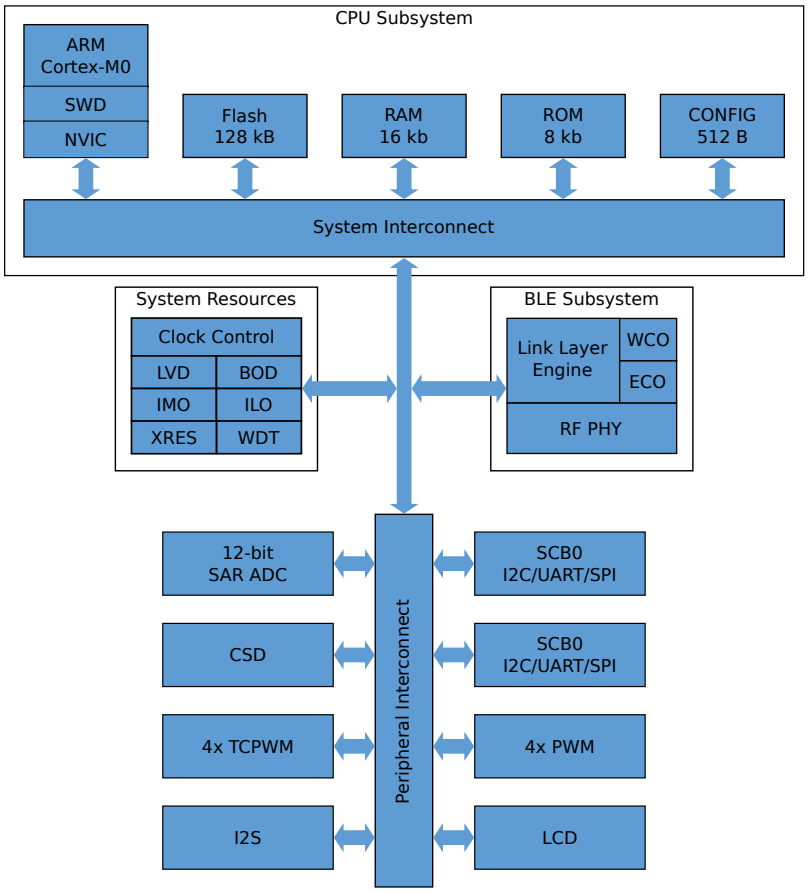

Figure 2.2: System architecture of the CYBL10563 by Cypress.[7]

one is, for example, still able to directly connect an input pin to an output pin in hardware without having to write a single line of code). The higher tier devices allow further configuration with predefined logic gates and extended components, even the possibility of writing custom components in Verilog.

2.2 Processor Interconnect

To connect the PRoC BLE to the Low-power Wireless Bus, we decided to use an existing platform tailored specifically for this application: BOLT.

The BOLT platform is designed explicitly for low power applications such as the Low-power Wireless Bus. As depicted in fig. 2.3, it enables the exchange of messages between two processors while decoupling them in terms of time, power and clock domains. This makes it possible to design the two systems without having to setup synchronisation between them. At the same time, the

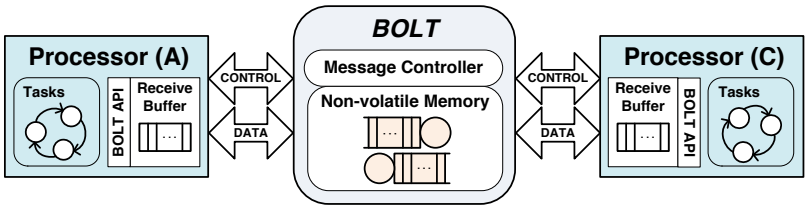

Figure 2.3: Overview of the BOLT architecture.

power overhead of BOLT is neglectable compared to the power used by the main processor, as shown in [9].

The communication interface consists of a three wire SPI bus (clock, MISO (master in, slave out) and MOSI line (master out, slave in)), a very common peripheral available in most MCUs, as well as four control lines. Figure 2.4 shows the sequence of the two operations provided by BOLT. The connecting processors (referred to as 'Processor (A)' or 'Processor (C)' in fig. 2.3) have to drive the lines marked by the dashed circles, while BOLT drives the lines marked with the solid circles. $R/\overline{W}$, *REQ*, *ACK* and *IND* hereby are the control lines, while *DATA* represents the SPI bus.

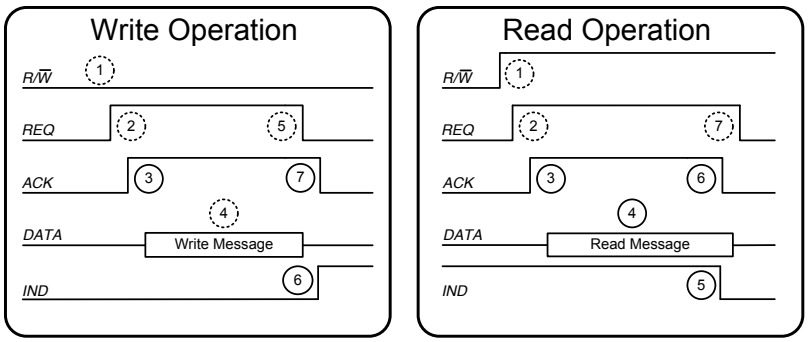

Figure 2.4: Figure showing the signal sequences for the BOLT operations Write and Read.

2.3 System Architecture

The selection of those components lead to the development of the following system architecture, used throughout this thesis and depicted in fig. 2.5.

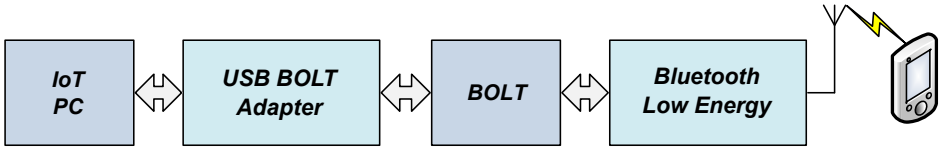

Figure 2.5: System architecture of the protocol gateway.

The use of the BOLT platforms makes it easy to use a PC to emulate the Low-power Wireless Bus. The device used in this project is the existing USB BOLT adapter. In detail, the system architecture consists of the following components:

- A PC running a serial console, emulating the Low-power Wireless Bus.
- The USB to BOLT adapter board. This component features an FTDI FT232R for a virtual COM port as well as a Texas Instruments MSP430 which provides a simple console to a connected PC and an UART to BOLT converter. It is powered over USB. This board enables the PC mentioned above to serve as a LWB replacement by providing a way data can be written to and read from the system.
- The BOLT platform. It is powered by the 3.3V rail of the Cypress development board.
- The Bluetooth Low Energy platform, consisting of two components:
  - The PRoC BLE module which hosts the MCU and the BLE subsystem.
  - The Cypress Pioneer Board which provides the programmer/debugger, power rails and breaks out the pins of the PRoC BLE module.
- And finally a Nexus 6 running Android 6.0.1 and a custom BLE app.

Chapter 3

Design and Implementation

This chapter explains the components involved to form the final product as pictured in fig. 3.1. It goes over the most relevant features of Bluetooth Low Energy, shows the design of the software and finally defines the resulting functional behaviour of the protocol gateway.

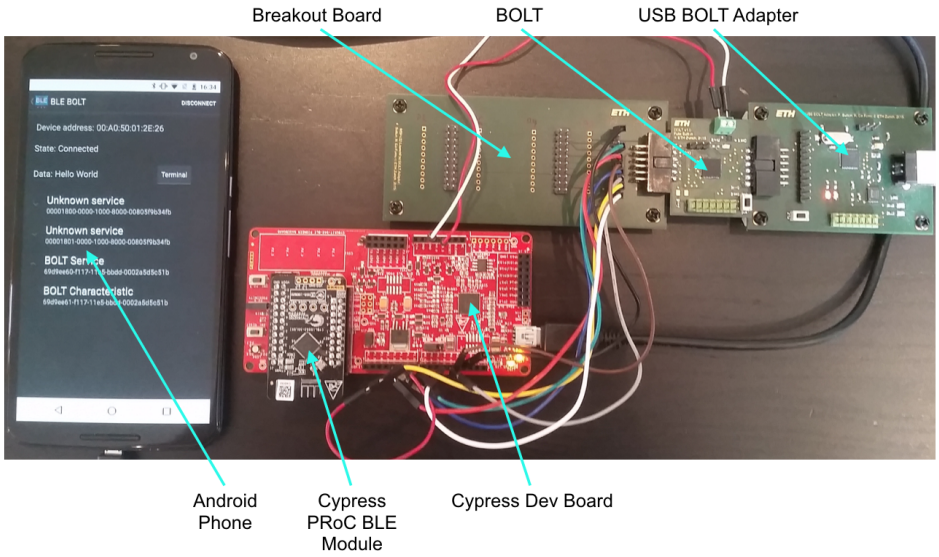

Figure 3.1: Photo of the final implementation.

3.1 Bluetooth Low Energy

3.1.1 Overview

Bluetooth Low Energy (BLE), also known as Bluetooth Smart, is a low-power, low-range, low-data-rate wireless personal area network defined by the Bluetooth

Special Interest Group. It was first developed at Nokia under the name Wibree in 2006 before being integrated into version 4.0 of the Bluetooth Core Specification in 2010 [1].

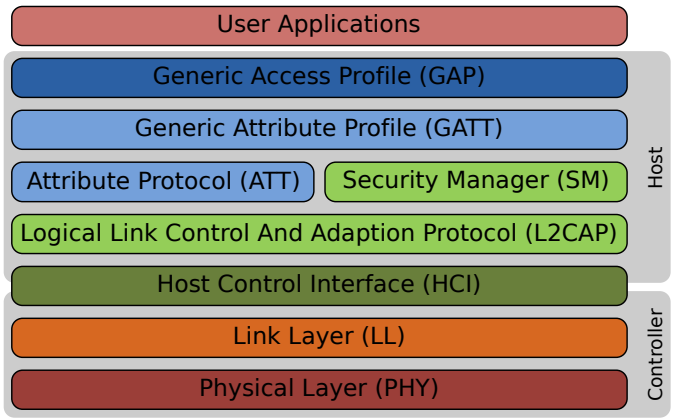

Figure 3.2: The BLE Stack [6]

As shown in fig. 3.2, the BLE stack consists of the following layers:

- The 2.4 GHz RF Physical Layer (PHY) with a 1-Mbps data rate.
- The Link Layer (LL) that defines the timing and the packet format for the PHY.
- The Host Control Interface (HCI) which provides an interface for the hardware control layer and the firmware host layer.
- The Logical Link Control and Adaptation Protocol (L2CAP) assembles and disassembles incoming and outgoing packets while providing data multiplexing and QoS for higher layers.
- The Attribute Protocol Layer (ATT) which defines how application data can be accessed with the use of a client-server protocol.
- The Security Manager (SM) provides encryption options for BLE data.
- The Generic Attribute Profile (GATT) provides the actual application data based on the ATT layer.
- The Generic Access Profile (GAP) controls connections and visibility of the device.

In the following three subsections we provide a more detailed explanation of the GAP, the ATT, and GATT layer, since those layers define the overall behaviour of a BLE device and introduce the functionality used by the application.

3.1.2 Generic Access Profile Layer

The configuration of the GAP layer defines the properties of a BLE device in terms of discoverability, link management and connectivity to other devices. It can be configured to one of four defined roles:

Peripheral

This is a passive role where the device sends periodic advertisement packets. It can be discovered by a central device which is then able to establish a connection from the central to the peripheral device, forming a physical link. On the link layer the peripheral device is then a slave of the central device which acts as the master. The peripheral device is not capable of more than one connection.

Central

This is the active counterpart to the peripheral role. The BLE device scans for advertisement packets of peripheral devices. It can then establish a connection to the peripheral device and operates as the master of the link. Contrary to a peripheral device, a central device is capable of having multiple connections with different peripheral devices at once.

Broadcaster

There is also a connection-less data transmission model in BLE. Contrary to the the connection-bound model, it is the active BLE device sending advertisement packets containing a small (up to 31 bytes) payload to be received by observer devices.

Observer

This is the counterpart of the broadcaster role. The BLE device is listening for advertisement packets which contain the data payloads.

3.1.3 GAP Configuration of the Protocol Gateway

The peripheral/central configuration forms a reliable link between two connected BLE devices, while the broadcaster/observer configuration provides a simple way of sending data from one BLE device to many others without the need for a connection between the involved devices. However, those data transfers are limited in size (max. 31 bytes) and there is no guarantee of reception. The fact that the potential size of the incoming messages (BOLT supports messages with a size up

to 128 bytes) massively excels the capabilities of the Broadcaster/Observer configuration, as well as the missing guarantees of delivery mark this configuration as unsuitable for the given application.

Hence we decided in favour of the peripheral/central scheme, since supported message size and the reliability heavily outweigh the drawbacks of having to establish a connection between two BLE devices as well as the limitation of the 1-to-1 connection.

In the design of the protocol gateway the PRoC BLE is the peripheral device. It sends periodic advertisement packet to be received by the smartphone, which acts as the central device, scanning for the advertisement packets of the PRoC and establishing the user-initiated connection.

3.1.4 Attribute Protocol Layer

In BLE, data is organised in so called attributes. They consist of the following:

- The Attribute Type, a 16-bit, 32-bit or 128bit UUID which defines the type of data the attribute contains. The Bluetooth SIG provides a list of preassigned UUIDs, but it is also possible to use 128-bit custom UUIDs.
- The Attribute Handle, a 16-bit address which provides access to the attribute.
- The Attribute Value, the actual payload of an attribute with a (optionally variable) size between 0 and 512 bytes.
- The Attribute Permission defines access, authentication, and authorisation requirements. Those are managed by higher layers.

This layer implements a peer-to-peer protocol between an attribute server and an attribute client. The ATT client can send commands, request and confirmation to an ATT server which sends responses, notifications and indications back to the client.

The maximum size of the attribute values a device is able to support is called the Maximum Transmission Size or MTU. The ATT layer itself uses up to 3 bytes for additional information like the operation code. If an attribute is too large to fit into one lower level package, it is automatically split into several packets, transparent to the upper layers.

3.1.5 Generic Attribute Profile

The GATT layer is closely linked to the ATT layer and provides the actual functionality of the server using attributes. They have the following organisation:

- Characteristics are a collection of attributes providing access to data. They consist of the following attributes:
  - Characteristic Declaration Attribute marks the beginning of a characteristic.
  - Characteristic Value Attribute contains the data. Those first two attributes are mandatory.
  - Characteristic Descriptor Attributes: There might be one or multiple attributes with additional information about the characteristic, such as the unit, a value range or an option to turn on notifications or indications.
- Services are a collection of characteristics and/or other services. They provide a way of ensuring hierarchy and grouping. There exist two types of services, primary and secondary services. The primary services provide access to the main functionality of a device while secondary services provide additional data, such as the battery level.
- And finally the so-called profiles represent a collection of services required for a certain application.

To access the data, Bluetooth Low Energy defines a client/server relationship based upon the ATT layer. A device either operates as a GATT server (providing the data) or as a GATT client (requesting data). After connecting to a server, the client first issues a service discovery, to which in response it receives the provided profiles, services, characteristics, and attributes. From this point on, the client is able to exchange data by sending requests to the previously discovered attributes on the server. Some of the more common request types are the read and the write request. Furthermore, BLE supports so called notifications and indications. Those message types enable the GATT server to push data to the GATT client. The difference between notifications and indications is that notifications do not require a response, making them faster and cheaper but less reliable.

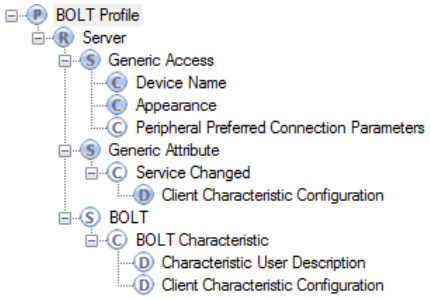

Figure 3.3: A screenshot of the configuration used for the PRoC BLE device.

3.1.6 Recap and Actual Configuration

A visual representation of the two involved devices is shown in fig. 3.4. As discussed earlier, the Cypress PRoC BLE is configured as the peripheral device while the smartphone is configured as the central device. The PRoC hosts the GATT server, publishing the custom BOLT profile [4], containing some generic services which contain data such as the name of the device or its appearance as well as the BOLT service (please refer to fig. 3.3 for the exact configuration). The smartphone, which acts as the GATT client, can then connect to the server on the PRoC, accessing and manipulating its data.

Below is a sequence diagram showing typical message sequences.

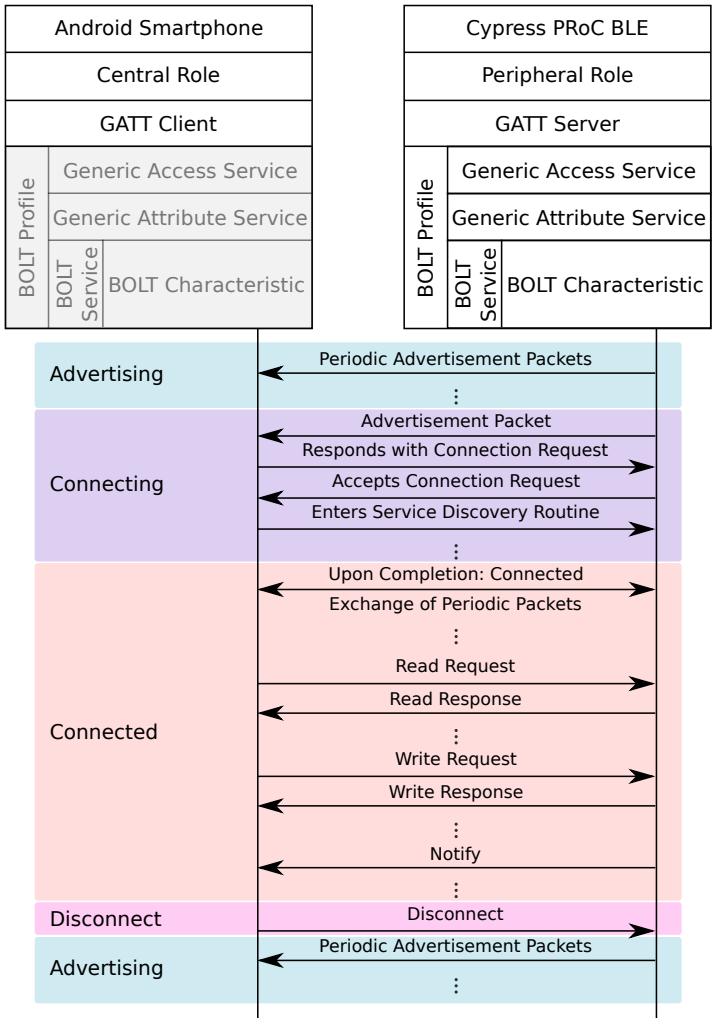

Figure 3.4: This figure shows the configuration of both the PRoC BLE device and the smartphone and shows a sequence of typical messages.

3.2 Embedded Software Design

This section explains the design of the embedded software und highlights some of the decisions made during development. To develop the software, the PSoC Creator was used, a free IDE provided by Cypress, providing the ability to configure, program, and debug the firmware.

The embedded software had to provide the following features:

- Ensure that the BLE device is discoverable by central devices while not connected.
- Accept incoming BLE connections and issue a connection parameter update request in order to make sure the connection parameters are suitable for low power operation.
- Provide the structure to transmit, receive and handle BLE data.
- Provide the structure to transmit, receive and handle BOLT data.
- Enable transfer in between the BLE and the BOLT component.

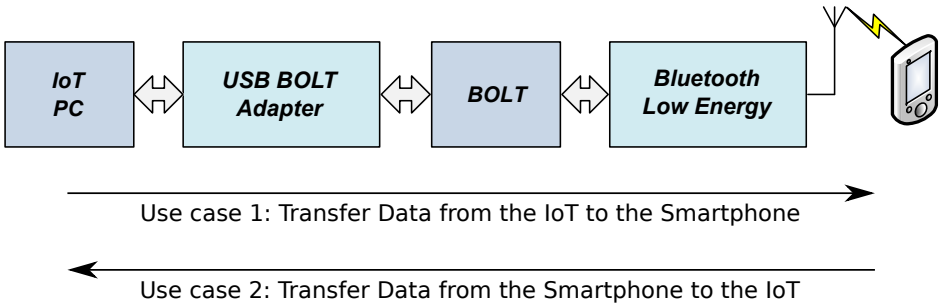

Figure 3.5: Use cases describing the functionality of the protocol gateway.

Furthermore, there exist two obvious use cases, illustrated in fig. 3.5, which are referenced throughout this thesis:

Use case 1 enables the transfer of data from the Internet of Things, in this case the Low-power Wireless Bus to the smartphone, involving the BOLT platform and the BLE component.

Use case 2 is the exact opposite of *Use case 1*, defining the transfer of data from the smartphone to the Low-power Wireless Bus.

Those requirements lead to the development of the two major components, the BLE application and the BOLT application, being glued together in the main loop, as explained below.

Figure 3.6 shows the flow of the designed embedded system. The following paragraphs will describe each point in further detail.

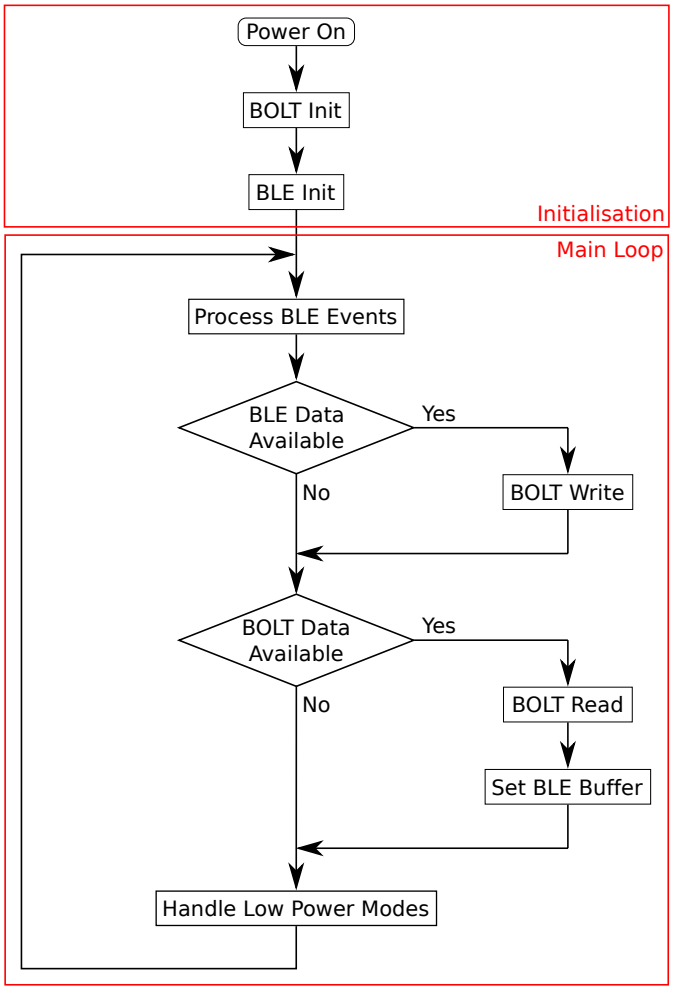

Figure 3.6: Diagram showing the embedded system control flow.

The *Initialisation* consists of the following blocks:

On *Power On*, the PRoC first runs through the proprietary setup code provided by Cypress which initialises the stack, the heap, the pin routing and similar components as configured in the IDE PSoC Creator by Cypress. This is where the software jumps to the user defined main function, containing user initialisation code as well as the main loop, running the actual application.

BOLT Init initialises the BOLT component. While this includes initialising the SPI component, the GPIO components have already been initialised during the first code segment provided by Cypress. Therefore, we only have to put

the SPI and GPIO components into their low power state as well as to set the internal state of the BOLT application. It is notable that the BOLT component manages its sleep state on its own, since it is heavily dependent on its current inner state[3].

BLE Init initialises the proprietary BLE software protocol stack and registers the callback *BoltBleApp_eventHandler* through which the BLE stack connects to the application, informing it of certain events and allowing the firmware to take action if required.

After *Initialisation* the application enters the *Main Loop*:

Process BLE Events calls the *CyBle_ProcessEvents* function to enter the proprietary software protocol stack. This function manages the BLE subsystem hardware block and the BLE software stack depending on its internal states. The BLE stack informs the firmware of certain events by calling the *BoltBleApp_eventHandler*. Those events include events such as the connection / disconnection of a central device, the reception of GATT requests and error states. If data is received by the BLE component, it is copied to a buffer and a flag is set.

BLE Data Available checks for the flag set by the *Process BLE Events*. If it is set, it copies the buffer to the BOLT component and initiates a *BOLT Write* sequence (see section 2.2). This corresponds to use case 2, as defined in fig. 3.5.

BOLT Data Available checks for pending BOLT data by reading the IND line (refer to section 2.2 for details). If data is available, a *BOLT Read* sequence is performed and upon success the data is stored in a buffer and forwarded to the BLE component with *Set BLE Buffer*. The BLE component then takes care of making this data available for read request of an ATT client and if notifications are enabled, it also prepares the notification for the next connection interval. This corresponds to use case 1, as defined in fig. 3.5.

Handle Low Power Mode finally puts the system to the lowest power mode currently available as discussed in the subsection below.

3.2.1 Low Power Management

This subsection introduces the power modes supported by the PRoC BLE device. Please note that the MCU and the BLE subsystem each have their own power modes.

MCU Low Power Modes

Figure 3.7 represents the 5 power modes supported by the MCU. Due to the long wakeup time, the power modes *Hibernate* and *Stop* are irrelevant for this project. Therefore, the remaining power modes are *Active*, *Sleep* and *Deep Sleep*.

Power Mode	Current Consumption	Code Execution	Digital Peripherals Available	Clock Sources Available	Wake Up Sources	Wake Up Time
Active	850 μA + 260 μA per MHz	Yes	All	All	-	-
Sleep	1.1 mA at 3 MHz	No	All	All	Any interrupt source	0
Deep Sleep	1.3 μA	No	WDT, LCD, I^2C/SPI, Link-Layer	WCO, ILO	GPIO, WDT, I^2C/SPI, Link Layer	25 μs
Hibernate	150 nA	No	No	No	GPIO	2 ms
Stop	60 nA	No	No	No	Wake-Up pin, XRES	2 ms

Figure 3.7: MCU low power modes as described in the user manual [5] by Cypress.

If it were not for the BLE subsystem, the *Sleep* mode would not be required. However the BLESS does not allow the MCU to go to deep sleep for every internal state. This means that the MCU has to choose the lowest possible power state depending on the current internal state of the BLESS.

BLE Subsystem Low Power Modes

As shown in [5] the BLE subsystem comes with its own low power modes. Even though the table is quite overwhelming at the beginning, one can reduce the table to the following:

1. The BLESS supports 3 different low power modes: *Active*, *Sleep* and *Deep Sleep*. Each of those low power modes is linked to some of the internal states of the BLESS.

2. There is only one internal state which requires the MCU to be in the *Active* power mode.

3. There are two internal states where the MCU is allowed to go to *Deep Sleep*, one of those is the *Deep Sleep* state itself.

4. Every other internal state allows the MCU to go to *Sleep*, although not into *Deep Sleep*.

This means, that the active / deep sleep interval of the MCU is disturbed by some intervals where it is only allowed to go to sleep. It is also notable, that the MCU is allowed to go to sleep during active intervals of the BLESS where either the transmitter or the receiver is active.

Implementation of the Low Power Modes

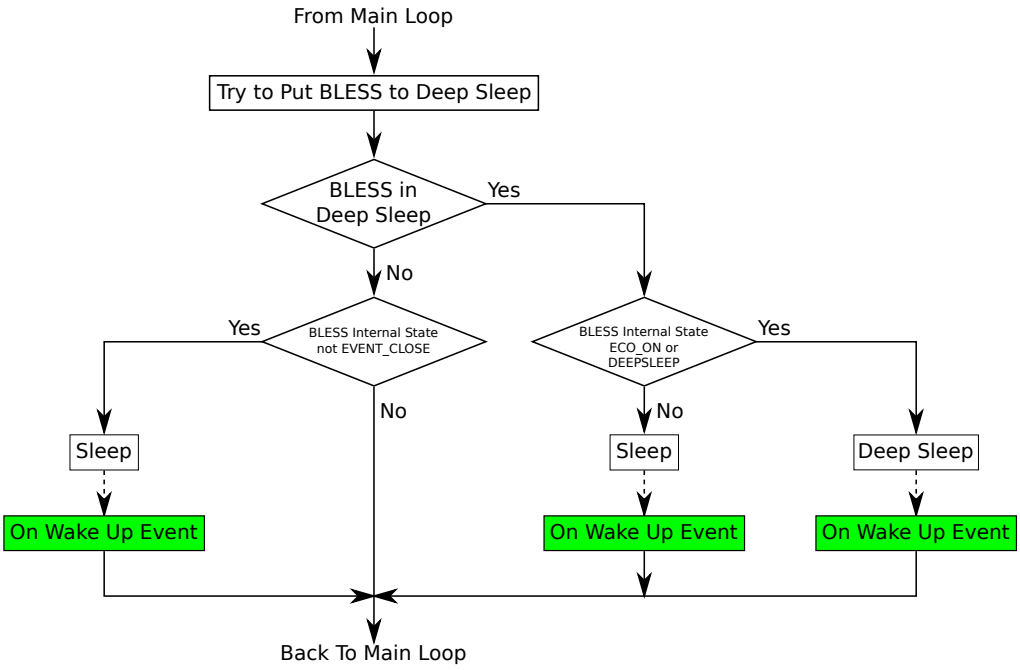

Figure 3.8: Diagram showing the low power mode control flow.

Given the facts presented in the last two sub-subsections, it is now clear that the low power mode of the MCU depends on the low power mode of the BLESS. As shown in fig. 3.8, the firmware first tries to put the BLESS into deep sleep. If that succeeds (depending on the current internal state of the BLESS), the MCU then has to decide whether it is allowed to go to deep sleep or if sleep is the lowest allowed low power mode. In case the MCU was unable to put the BLESS into deep sleep, it checks if the internal state of the BLESS is the one internal BLESS state prohibiting the sleep mode of the MCU. If so, the function returns back to the main loop, otherwise the MCU goes to sleep. If the MCU is able to

go to sleep or even deep sleep, with the current setup the only wake up event is the BLE interrupt.

3.2.2 Message Handling

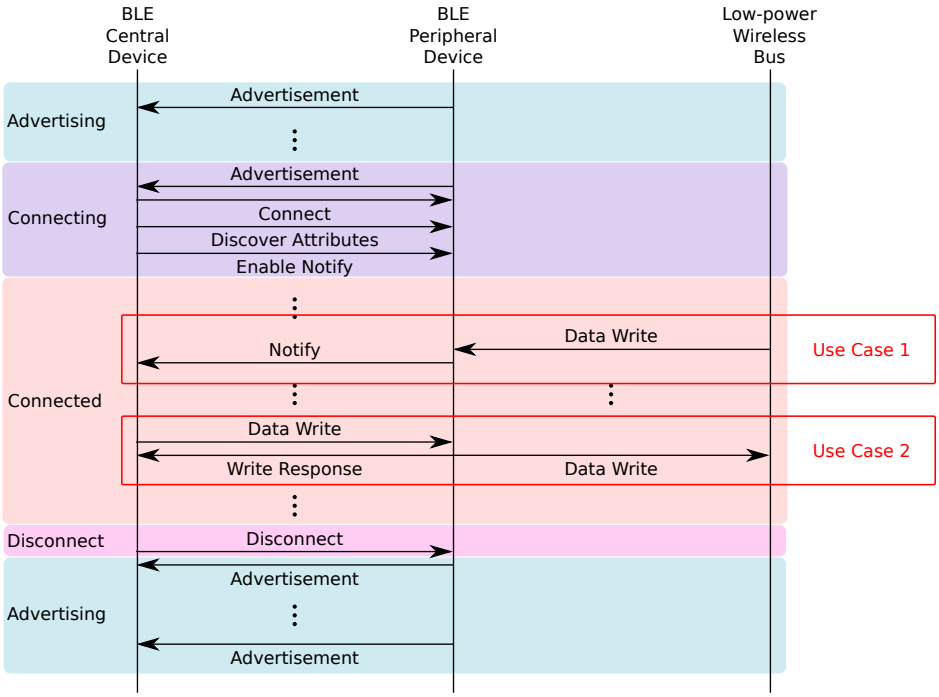

Figure 3.9: Sequence diagram of messages between the central device, peripheral device and the Low-power Wireless Bus.

The sequence diagram depicted in fig. 3.9 shows the most important messages in the system during a typical user interaction. We differentiate between 4 states and transitions: *Advertising*, *Connecting*, *Connected*, and *Disconnect*. The following paragraphs will discuss further details.

During *Advertising*, the peripheral device periodically transmits advertisement packets, unaware whether or not another device is receiving them.

When a central device receives an advertisement packet sent by the peripheral device, it has the opportunity to respond immediately after receiving the advertisement packet with a connection request. This has to be done 150 ns (the Interframe-Space (IFS) defined by the BLE specifications) after the reception of an advertisement packet, since this is the only period the peripheral device is listening for incoming transmissions. The connection request aims at the establishment of a connection, starting the *Connecting* sequence. The peripheral device may choose to accept or decline this request. If the connection attempt

is successful on the link layer, the upper layers also start their respective connection routines. On the ATT/GATT layer, this means that the client on the central device initiates the service discovery routine to discover the GATT profile published by the server on the peripheral device. As a final step, the client sends a request to the server to set the notify enable flag for the data of interest, in this case the BOLT characteristic.

While two BLE devices are *Connected*, they exchange periodic packets to keep up the link, even if no data needs to be transferred. During this state, the two use cases come into play. Use case 1 starts with data arriving at the Low-power Wireless Bus, which is then written to the BOLT platform. Then the Main Loop (as defined in fig. 3.6) checks for and reads the BOLT data and transfers it as a BLE notification to the smartphone, where it is received an displayed. Use case 2 starts with the user entering data on the smartphone, which then transmits this data to the connected peripheral device. The peripheral device receives said data, confirms the reception with a write response to the phone and writes the data to BOLT. The LWB component may now access this data from BOLT, whenever convenient.

3.3 Android Software Architecture

In order for the user to be able to access the data captured on the Low-power Wireless Bus and transferred to the smartphone by means of the protocol gateway, a user interface is required. This user interface should enable the user to discover BLE devices, connect and disconnect from them, as well as the ability to display and enter data exchanged with the PRoC BLE device.

To develop and debug the Android app, the software suite Android Studio was used. The Android app is based on the BLE Example App provided within Android Studio, expanded with some custom code and constants. Out of the box, the example is able to discover BLE devices and display them in a list. The connection can then be established by clicking the appropriate item in the list. This initiates the service discovery routine, so the client on the smartphone fetches the services published on the server on the PRoC BLE. The discovered services are then displayed on screen as shown in fig. 3.10. Clicking on a characteristic reads said value and sets the notify flag on the PRoC BLE server, so updates are pushed from the PRoC BLE to the smartphone immediately. Data can be entered by pressing the 'Terminal' button, which opens a dialog through which text can be entered and then be sent to the PRoC BLE.

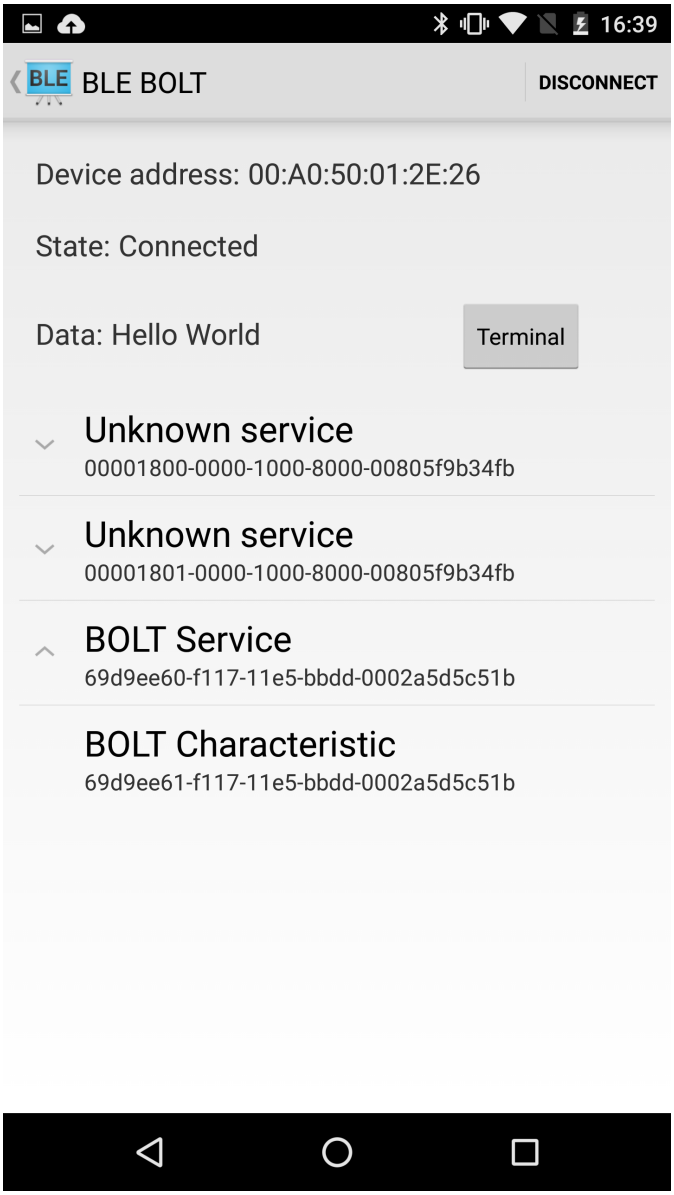

Figure 3.10: A screenshot of the Android app. One can see the received message as well as the discovered services.

3.4 Functional Behaviour

This subsection shows how the current implementation works. It covers the two use cases but leaves out the IoT, starting direct at the BOLT platform.

3.4.1 Use Case 1 - IoT To Phone

After reading data from BOLT, the peripheral device forwards this data to the BLE-Tx-Buffer and prepares the data to be transferred over BLE. Upon reception by the smartphone app, the data is then displayed to the user. While the BOLT platform is currently configured to a packet size of 48 bytes, it is designed for a maximal packet size of 128 bytes [9]. The tests which included BOLT used its maximum supported message size of 48 bytes, while there were separate tests conducted, using only the BLE connection with message sizes up to 128 bytes.

3.4.2 Use Case 2 - Phone To IoT

Sending data from the phone to the peripheral device is also a straightforward data transfer. However, the SPI component used in the PRoC BLE has a tendency to cut off/scramble the last one or two bits of a transmission. Therefore we decided to add 2 additional bytes: One at the beginning of the message containing the amount of bytes of the message (without the additional bytes) and one at the end to guarantee the correct transmission of the message. That means the receiver is always able to read the full message, is aware of its size and can get rid of additional, possibly scrambled bytes.

Chapter 4

Experimental Evaluation

This chapter illustrates the experimental setup used for the evaluation, discusses and interprets the results obtained in order to finally conclude whether or not the requirements in terms of low power consumption and sufficient bandwidth are met.

4.1 Power Analysis

4.1.1 Experimental Setup

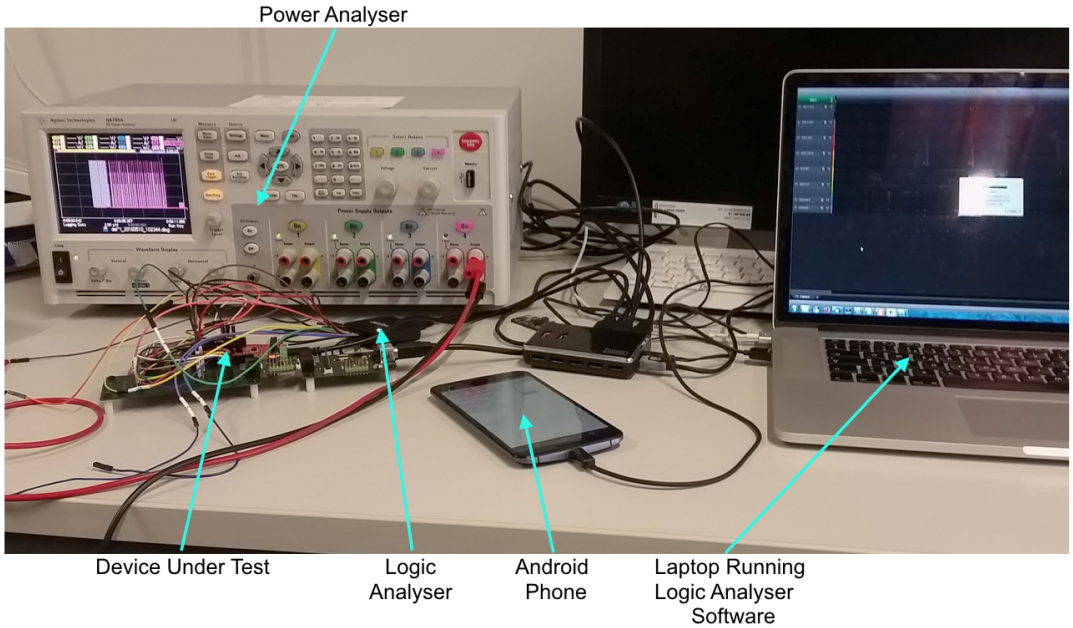

Figure 4.1: The experimental setup for the measurements.

To log the current the Agilent N6705A DC Power Analyser was used. Additionally the Saleae Logic 8 logic analyser was used to log further data:

- BOLT REQ line
- BOLT ACK line
- BOLT IND line
- BOLT SCLK line
- BLE INT (goes high when entering the BLESS interrupt service routine, low when leaving it)
- Sleep (High when the MCU is in sleep)
- Deep Sleep (High when the MCU is in deep sleep)

For convenience there are also 2 additional signals generated from the logged data as explained below:

- BOLT Active is defined as '(BOLT REQ or BOLT ACK or BOLT SCLK)'
- MCU Active is defined as 'not (Sleep or Deep Sleep)'

The use of the logic analyser made it possible to later on match the traces produced by the power analyser and the logic analyser, giving insight to the inner workings of the system.

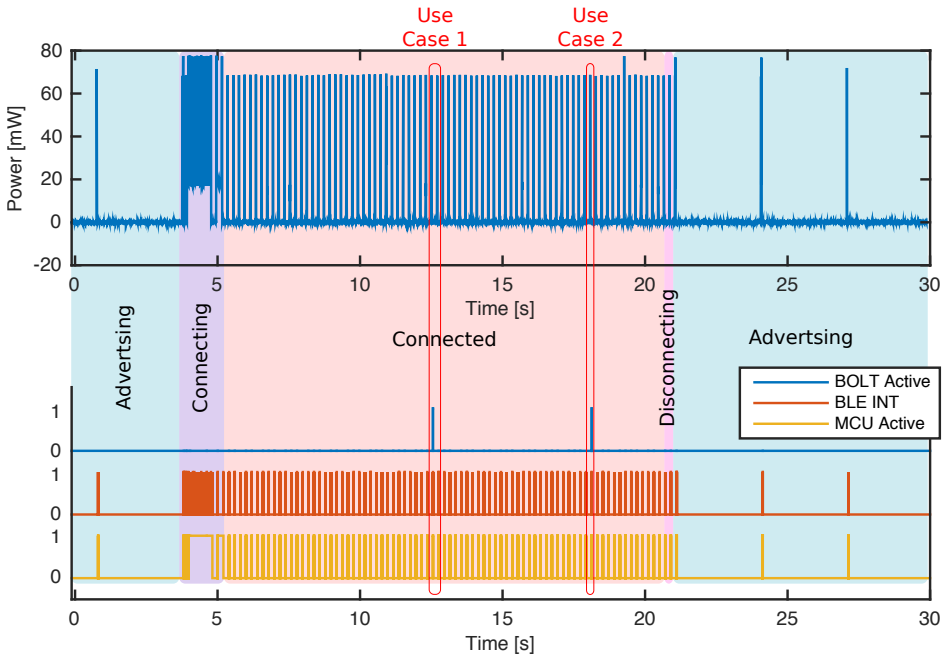

Figure 4.2: A full power trace: From advertising, connecting to a device, exchanging messages, disconnecting and back to advertising.

Figure 4.2 shows a trace of every possible state of the system. You can see the power profile on top as well as the logic lines below. The first spike is an advertisement event. Then the central device establishes the connection and discovers the services of the GATT server on the peripheral device. This triggers the peripheral device to issue a connection parameter update request to the link master, which increases the connection interval to save power. There are 2 BOLT transmissions representing the two use cases, each containing 8 bytes. Afterwards, the smartphone disconnects and the peripheral device goes back to advertising.

During normal operation the deep sleep current was measured at 1.3μA. However, due to the use of the additional GPIOs for the logic analyser, the deep sleep current went up to 4.5μA during the measurements. However, the 1.3μA measured before was used during calculations, as it was a well established number at that point.

4.1.2 BLE Advertising

When the peripheral device is in the advertisement state, it periodically transmits one to three advertisement packets. Each BLE activity including one or more advertising packets is called an *Advertising Event*. There are 3 advertisement channels used in BLE (channels 37 through 39). During the advertisement event one packet is sent to each channel, however, it is possible to disable the transmission for one or two channels. Optionally, the device listens for a scan request after each transmission, issued by a device which received the advertisement packet and would like to receive more data, not included within the original advertisement packet. If the peripheral device receives a scan request, it would respond directly afterwards with the scan response. The peripheral device is configured to have an advertisement interval of 3 seconds. During deep sleep the BLESS clock is driven by the WCO (watch crystal oscillator) which is used to generate timer interrupts. The intervals depend on the current state the device operates in.

In Figure 4.3 we can see the power profile during one advertisement event. We can observe the following sequence:

1. At t=0ms the system is in deep sleep.
2. A BLESS interrupt wakes up the system and turns on the external crystal oscillator (ECO), then goes back to deep sleep.
3. When the ECO reaches a stable amplitude, it fires a second interrupt. The device has to wait for the ECO frequency to stabilise. The system goes to sleep.
4. When the ECO has a stable frequency, it changes the BLESS clock from the WCO to the now stable ECO, wakes the BLESS link layer from deep sleep, and generates another interrupt.
5. The CPU wakes up and the BLESS enters active mode. If there is no processing required, the CPU can go back to sleep until the BLESS operation completes.
6. The actual transmission consisting of
   (a) The transmission of the advertisement packet itself, on channels 37, 38 and 39.
   (b) A waiting period: the Inter-Frame Space, a time interval of 150 μs to allow the RF component to reconfigure between transmission and receiving.
   (c) The peripheral device listens for a packet of a peer device. If it times out it stops listening.

(d) The interval between advertisement packets is called Inter channel delay which is defined as 1.25ms.

7. After the last transmission sequence the BLESS generates an *end-of-event* interrupt to wake the CPU which can then process event-specific or app-specific requests. After this processing period, the CPU changes the BLESS clock source back to WCO and puts the BLESS back to deep sleep.

8. The last interrupt signalises that the BLESS is now in deep sleep and the CPU can now also safely go to deep sleep.

The duration of this advertising event was 7.414 ms and it used 23.921 mW of power.

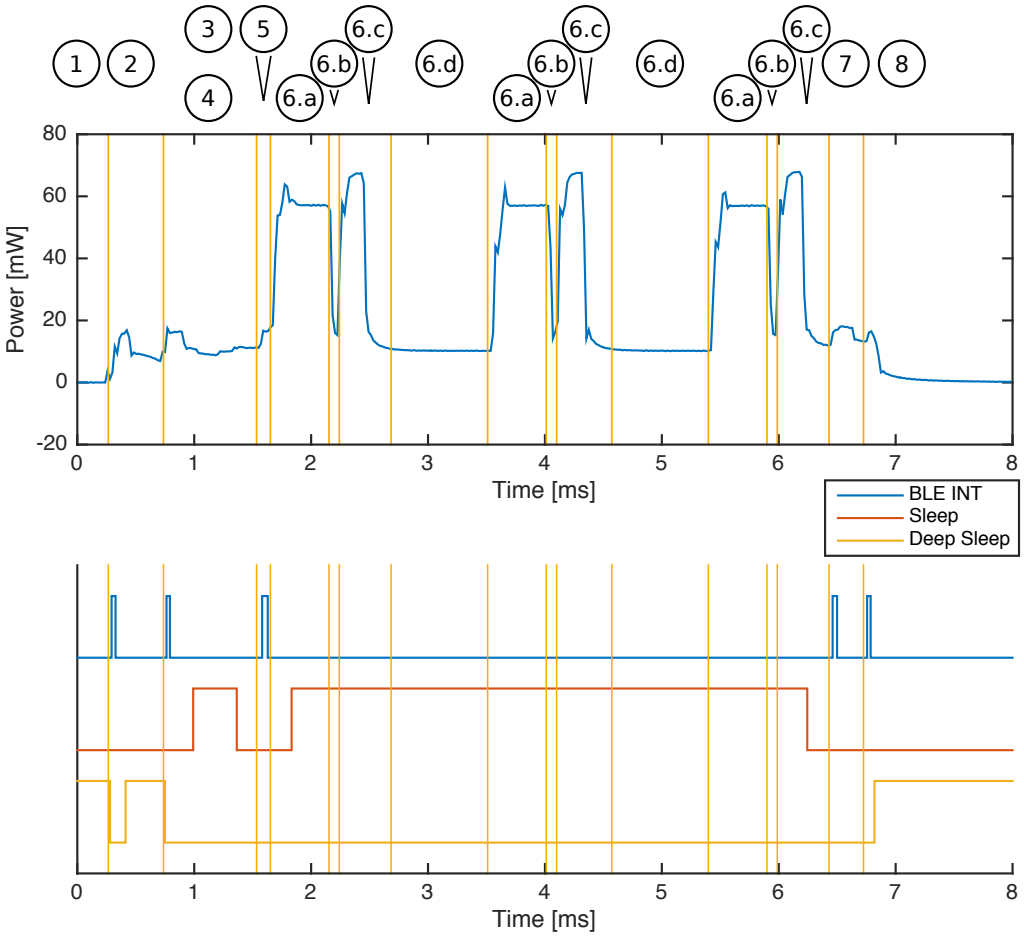

Figure 4.3: Power profile of one advertising event.

4.1.3 BLE Connecting

Figure 4.4 shows the transition from the advertising state to the connected state. First the central device responds to an advertisement packet with a connection request which contains all the required data to establish a connection (1). Afterwards, one can see the GATT service discovery routine (2) which fetches the available profiles, services and characteristics. When the GATT service discovery routine has finished, the peripheral device requests a connection parameter update (3) from the central device to increase the connection interval to 200ms (from 50ms as propagated by the central device). Then the connection is kept until either the central or the peripheral device initiate a disconnect (4).

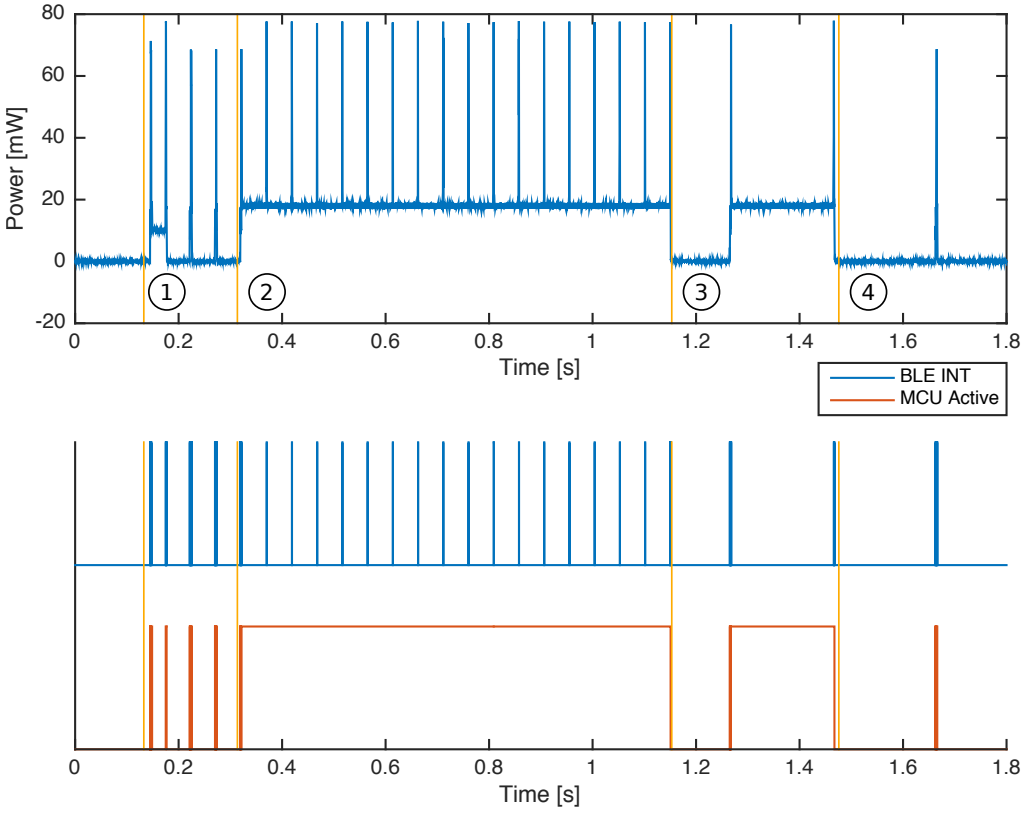

Figure 4.4: The power trace of a connection establishment

4.1.4 BLE Connected

A connection for data transfer is established between a central device (master at the link layer) and a peripheral device (slave at the link layer). Each of those packet exchanges is called a *Connection Event*. The central device initiates each periodic connection event by a transmission. Afterwards the devices may alternately transmit packets until they stop the exchange or the limit of allowed transmissions (for example due to either the capabilities of the device or the beginning of the next connection event) is reached and then close the connection event.

Figure 4.5 shows the power profile during one connection event. We can observe the following sequence (similar to the advertisement sequence):

1. At t=0ms the system is in deep sleep.
2. A BLESS interrupt wakes up the system and turns on the ECO, then goes back to deep sleep.
3. When the ECO reaches a stable amplitude, it fires a second interrupt. The device has to wait for the ECO frequency to stabilise. The system goes to sleep.
4. When the ECO has a stable frequency, it changes the BLESS clock from the WCO to the now stable ECO, wakes the BLESS link layer from deep sleep, and generates another interrupt.
5. The CPU wakes up and the BLESS enters active mode. If there is no processing required, the CPU can go back to sleep until the BLESS operation completes.
6. The actual packet exchange consists of the following:
   (a) The peripheral device listens for a packet sent by the central device.
   (b) After receiving a packet, it waits for 150μs, the so called inter-frame space.
   (c) The peripheral device can then send its own response packet.
7. After the last transmission sequence the BLESS generates an *end-of-event* interrupt to wake the CPU which can then process event-specific or app-specific requests. After this processing period, the CPU changes the BLESS clock source back to WCO and puts the BLESS back to deep sleep.
8. The last interrupt signalises that the BLESS is now in deep sleep and the CPU can now also safely go to deep sleep.

The duration of this connection event was 4.301 ms and it used 18.41 mW of power.

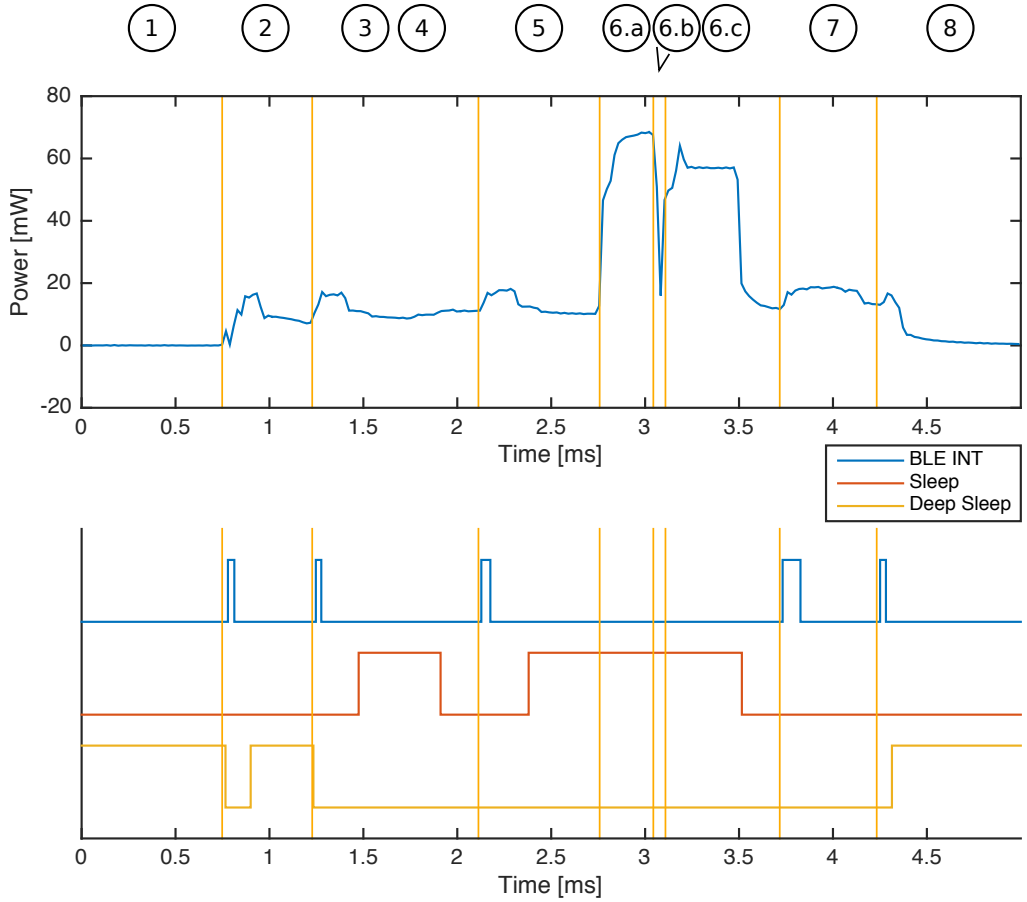

Figure 4.5: Power profile displaying one connection event.

4.1.5 Comparison of Power Consumption During Advertisement and Connection

As shown in fig. 4.6, the average power during advertisement is higher than the power during the connection event. This is because there are three packets transmitted during an advertising event, whereas it is only one packet during a connection event. This is compensated by the fact that the time between two advertisement events is usually much longer that the time between two connection events.

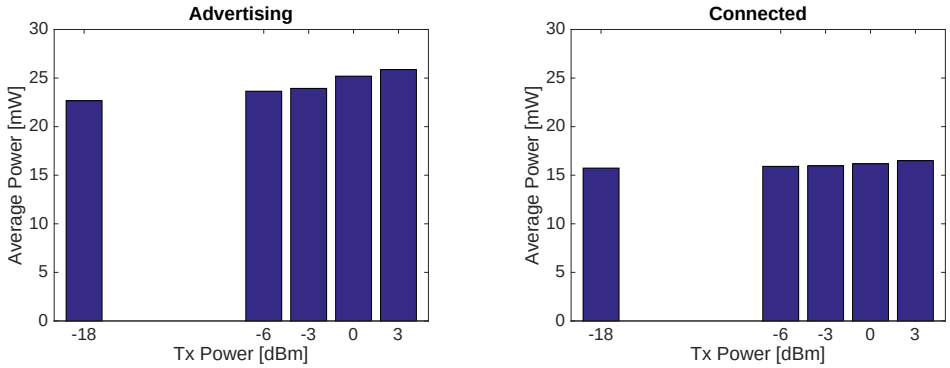

Figure 4.6: Average power for one advertisement respectively one connection event versus the set Tx Power.

4.1.6 Use Case 1 - IoT to Phone

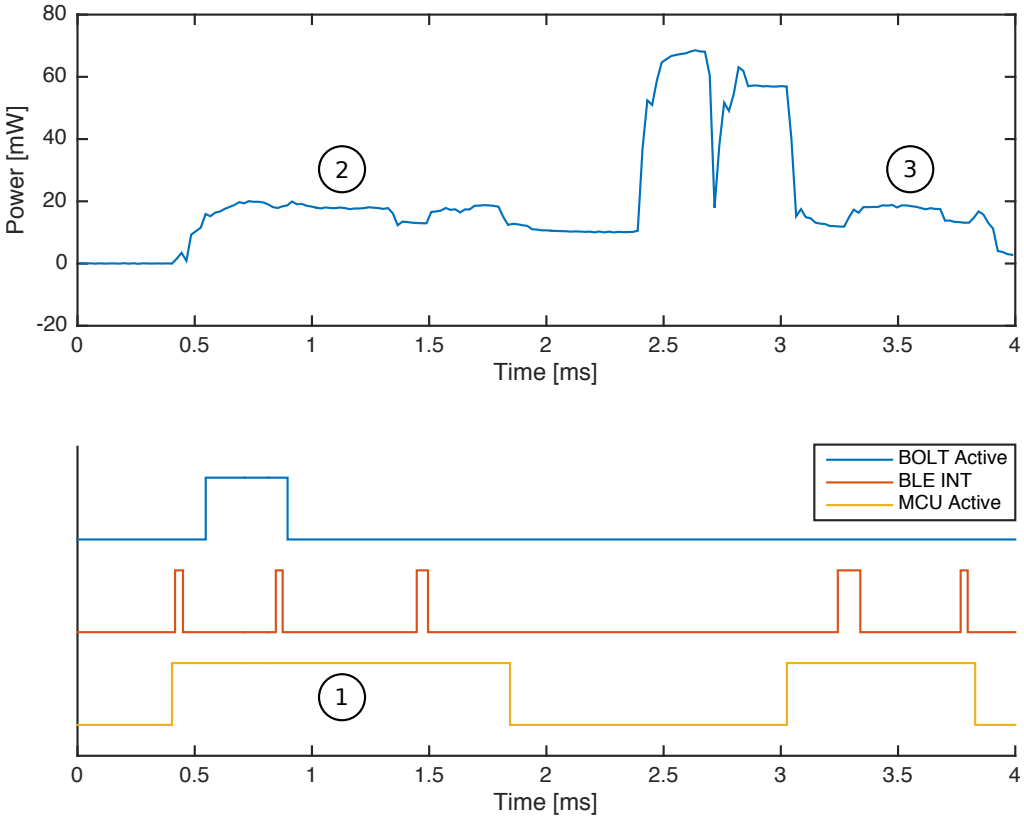

Figure 4.7: Close up of a connection event depicting use case 1.

Figure 4.7 shows the power trace of a connection event where data is being read from BOLT and then sent to the smartphone. When this trace is compared to the trace depicted in fig. 4.5, the similarities are obvious, even though there are a few small differences. In a little more detail:

1. After the first BLE interrupt, the MCU is not able to go to sleep for ∼1.5ms. This is because it has to read the data from BOLT.
2. This is also visible in the power trace.
3. The processing after the transmission is nearly identical.

The duration of this connection event was 4.301 ms and it used 18.767 mW of power. Compared to the connection event without BOLT activity, this shows no difference in time but a small difference in power: +0.357 mW.

4.1.7 Use Case 2 - Phone to IoT

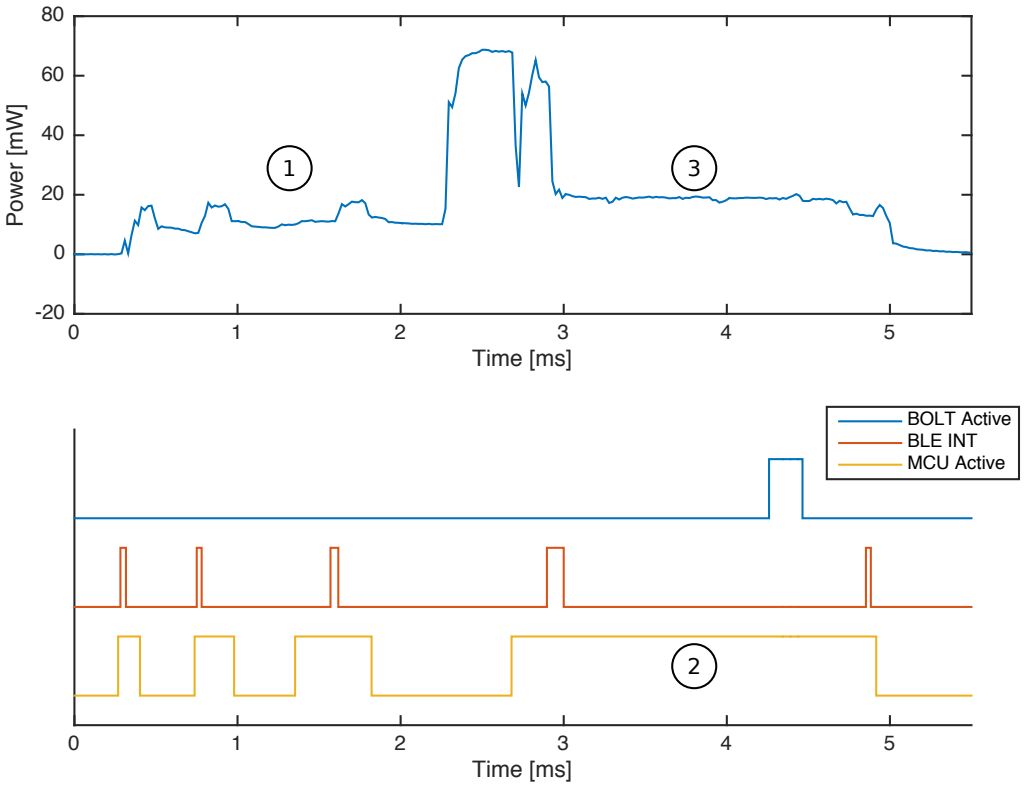

Figure 4.8: Close up of a connection event depicting use case 2.

Figure 4.8 shows the power trace of a connection event where data is being sent by the smartphone and then written to BOLT. When this trace is compared to the trace depicted in fig. 4.5, the similarities are obvious, even though there are a few small differences.. In a little more detail:

1. The processing before the transmission is nearly identical
2. There is ∼2.3ms of MCU activity after the reception of the packet compared to ∼ 0.8ms for a normal a connection event without BOLT activity. This is because the data received over BLE has to be written to BOLT.
3. This is also visible in the power trace.

The duration of this connection event was 5.366 ms and it used 18.536 mW of power. Compared to the connection event without BOLT activity, this shows a time difference of 1.065 ms and a small difference in power: +0.126 mW.

4.2 Bandwidth Estimation

This section shows how an upper bound for the bandwidth of a BLE connection can be calculated. This is an important part of the evaluation, since it shows whether or not this implementation of the protocol gateway is actually feasible.

The measurements were taken with a connection interval of 200ms but this parameter is configurable from 7.5ms up to 4000ms. During each connection event one or more packets can be sent. But they do not necessarily need to contain data, they are also used to maintain the connection. For this application, there are two relevant cases:

1. **Empty packet** - Consists of 10 bytes: preamble (1B), synchronisation word (4B), link layer header (2B), and CRC (3B)
2. **ATT data packet** - Consists of at least 14 bytes: preamble (1B), synchronisation word (4B), link layer header (2B), L2Cap length(2B), channel Id (2B), and CRC (3B). May include up to 27 bytes of payload.

Therefore, two devices could exchange nothing but empty packets, resulting in a minimum bandwidth of 0 bits per second.

To calculate an upper limit, it is necessary to investigate further. It should be pointed out that the calculation of the BLE bandwidth is a complex process, especially when the data transfers in both directions should be considered. The following calculation will assume transfer in only one direction. BLE uses a 1 Mbit/s radio, so the transmission of an ATT packet with the maximum payload of 27 bytes requires:

$$(27 + 14) \cdot 8\,\text{bits} = 41 \cdot 8\,\text{bits} = 328\,\text{bits} \Rightarrow 328\,\mu\text{s} \tag{4.1}$$

To configure the RF component from Tx to Rx and back takes 150 μs.

An empty packet, which is needed to continue the transfer between the devices, is 10 bytes long, requires 80 μs.

All of this combined means that for sending 27 bytes of payload we need:

$$328\,\mu\text{s} + 150\,\mu\text{s} + 80\,\mu\text{s} + 150\,\mu\text{s} = 708\,\mu\text{s} \tag{4.2}$$

Furthermore, the PRoC device is able to schedule data upon 1.25 ms before the start of the next connection interval. Assuming there are devices that are capable of receiving or transmitting almost continuously, we can use the longest connection interval available, leading to a theoretical maximum bandwidth of:

$$B_{max} = \frac{\frac{4000\,\text{ms} - 1.25\,\text{ms}}{708\,\mu\text{s}} \cdot 27 \cdot 8}{4\,\text{s}} \approx 316\,\text{kbps} \tag{4.3}$$

But due to the limitations of real life products the maximum bandwidth is limited. There are several limitations:

- Most smartphones support only ∼4-6 packets per connection interval
- The fastest connection interval supported is often only around 30-40ms, but there are some Android smartphones that support 7.5ms connection intervals.

Typical values for an Android smartphone would be a connection interval of 7.5 ms and 4 packets per connection event. This leads to a more realistic estimation of the maximum bandwidth (please note that it is possible to get more than double this throughput when not using a smartphone [8]):

$$B_{max} = \frac{4 \cdot 27 \cdot 8\,\text{bits}}{7.5\,\text{ms}} = 115.2\,\text{kbps} \tag{4.4}$$

Figure 4.9 depicts a broad range of possible configurations.

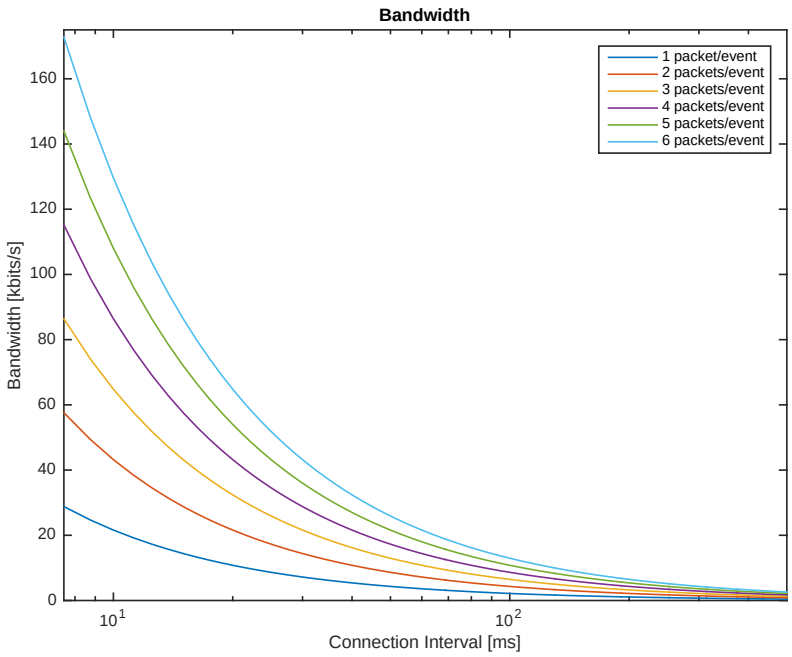

Figure 4.9: Comparison between the connection interval in log scale (from 7.5ms up to 500ms) and the achievable throughput.

Given the intended application of the protocol gateway, this bandwidth is absolutely sufficient. The expected bandwidth the system should be able to handle lies in the lower 10's of kbps.

4.3 Power State Model

To estimate the power consumption of the device for a longer amount of time, for example to be able to calculate how long a device would last given a battery with a certain capacity, a model was required. This section explains how this model was developed.

Due to the consistency of the data obtained during the evaluation it was possible to derive a model. The states are defined by the activity over time while the power over time defines the parameters of the model.

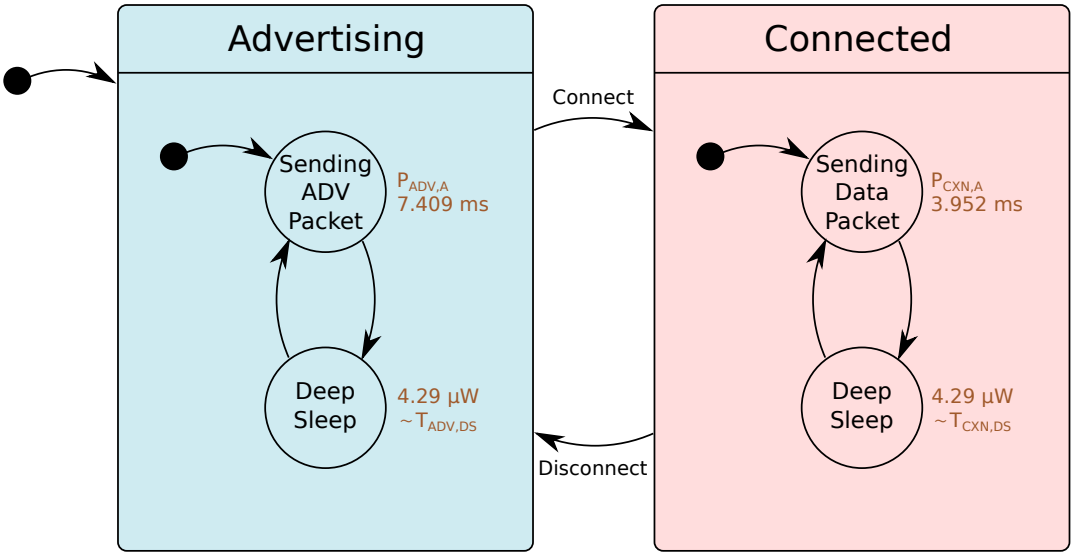

Figure 4.10: Graph describing the power state model.

Parameters

$P_{ADV,A}$	Advertising Tx Power
$P_{CXN,A}$	Connection Tx Power
T_{ADV}	Interval for advertising events
T_{CXN}	Interval for connection events

Here, P and T denote the power respectively the interval parameter. The subscripts $_{ADV}$ and $_{CXN}$ explain whether the symbol corresponds to the *Advertising* ($_{ADV}$) or *Connected* ($_{CXN}$) state. Furthermore, the subscripts $_{,A}$ and $_{,DS}$ (not yet used) denote the substate, either *Active* ($_{,A}$) or *Deep Sleep* ($_{,DS}$). The missing of both subscripts, $_{,A}$ and $_{,DS}$, means that the symbol describes the superstate.

Figure 4.10 shows the derived state chart. There are two superstates with two substates each. The two superstates represent the overall state the device is in, either *Advertising*, sending periodic advertisement packets, or *Connected*,

maintaining a connection to a central device, through a periodic exchange of packets. The upper substate in both superstates represents the active state of the system, where the BLE device sends and receives packets. The lower substate represents the deep sleep state. The numbers besides the states describe how much power is consumed for how long. The power during deep sleep is constant at 4.29 μW for both superstates. The time the device stays in deep sleep is dominated by the interval parameter for the respective superstate as shown further down. For the active state, the interval is given by the measurements, the power by the chosen Tx Power parameter.

The cost for the state transitions from *Advertising* to *Connected* and back are assumed to be zero, since those events should only appear sporadically and should not notably change the outcome of the model.

Tx Power	$P_{ADV,A}$	$P_{CXN,A}$	P_{DS}
3 dBm	25.873 ± 0.103 mW	16.499 ± 0.299 mW	4.29 μW
0 dBm	25.196 ± 2.173 mW	16.181 ± 0.464 mW	4.29 μW
-3 dBm	23.936 ± 1.566 mW	15.962 ± 0.304 mW	4.29 μW
-6 dBm	23.647 ± 1.421 mW	15.911 ± 0.268 mW	4.29 μW
-18 dBm	22.661 ± 1.469 mW	15.733 ± 0.282 mW	4.29 μW

Figure 4.11: Average power for active substates and the deep sleep substate in different configurations.

Advertising	7.409 ± 0.144 ms
Connected	3.952 ± 0.071 ms

Figure 4.12: Average duration of advertising and connection events.

	Advertising	**Connected**
Active-substate	$T_{ADV,A} = 7.409\,\text{ms}$	$T_{CXN,A} = 3.952\,\text{ms}$
Deep Sleep-substate	$T_{ADV,DS}$	$T_{CXN,DS}$
Superstate	$T_{ADV} = T_{ADV,A} + T_{ADV,DS}$	$T_{CXN} = T_{CXN,A} + T_{CXN,DS}$

Figure 4.13: Interval parameters.

4.3.1 Derivation of the Model

We will first derive the formula for the advertisement state. As a first step we use the data from fig. 4.13 to calculate how long the device stays in the deep sleep state.

$$T_{ADV,DS} = T_{ADV} - T_{ADV,A} \tag{4.5}$$

Now we can calculate the average power of the super state with this simple formula:

$$P_{ADV} = \frac{P_{ADV,A} \cdot T_{ADV,A} + P_{DS} \cdot T_{ADV,DS}}{T_{ADV}} \tag{4.6}$$

Using eq. (4.5) in eq. (4.6) we can get rid of $T_{ADV,DS}$, resulting in the final formula:

$$P_{ADV} = \frac{T_{ADV,A}(P_{ADV,A} - P_{DS}) + T_{ADV} \cdot P_{DS}}{T_{ADV}} \tag{4.7}$$

In the same manner we can derive

$$P_{CXN} = \frac{T_{CXN,A}(P_{CXN,A} - P_{DS}) + T_{CXN} \cdot P_{DS}}{T_{CXN}} \tag{4.8}$$

The plotted values are shown in fig. 4.14 and fig. 4.15. One can clearly see that both plots are heavily dominated by the the interval parameters T_{ADV} and T_{CXN}.

4.3.2 Trade-offs

Considering the almost neglectable impact of the Tx Power parameter, we can safely assume that the additional range and reliability gained through the use of a higher Tx Power parameter like 0 dBm or even +3 dBm would always outweigh the additional power used.

The advertisement interval parameter T_{ADV} mainly defines how convenient it is to discover and connect to a peripheral device. Experiments conducted during the development of the protocol gateway showed that while longer intervals reduced power, they also made discovery and connection establishment inconvenient and unreliable. It was found that a value of 3 s for T_{ADV} is a good trade-off between power consumption and user experience.

The connection interval parameter T_{CXN} on the other hand is one of the limiting factors of the maximum bandwidth as shown in section 4.2 and especially in fig. 4.9. To find an optimal value for this parameter the designer of the system has to evaluate the expected bandwidth and then choose the parameter accordingly.

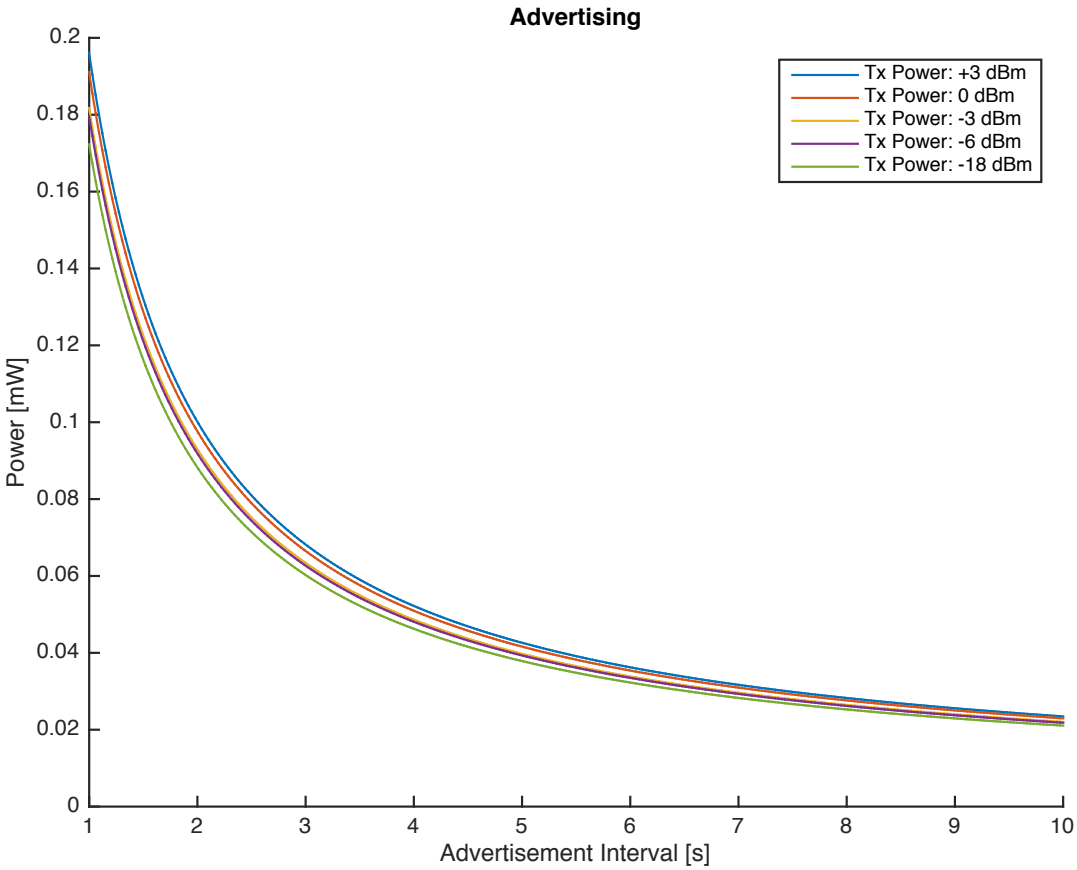

Figure 4.14: A plot of the power state model for the Advertising state, displaying the result of the different parameters. Please note that while the advertisement interval is only plotted from 1 s to 10 s, it could go as low as 20ms.

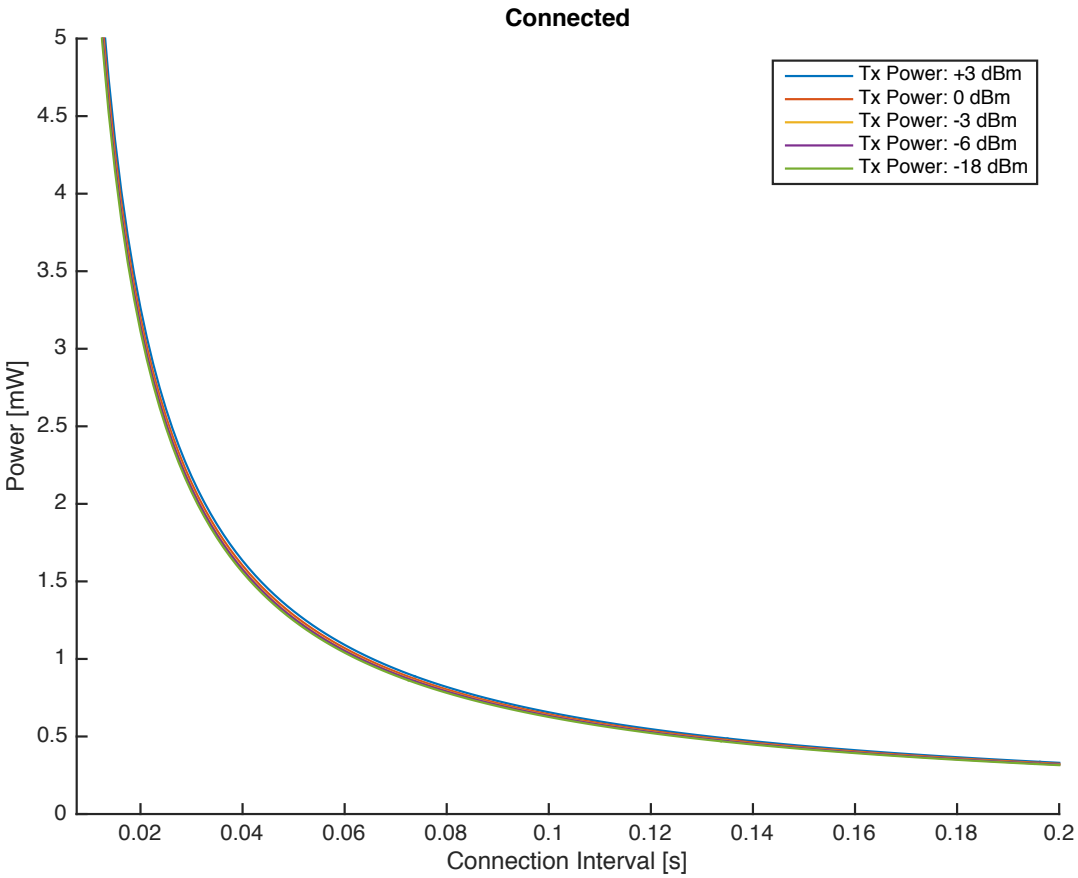

Figure 4.15: A plot of the power state model for the Connected state, displaying the result of the different parameters. Please note that while the connection interval is only plotted from 7.5 ms to 200 ms, it could go up to 4000ms.

4.3.3 Example

In order to establish whether or not this implementation of the protocol gateway suffices in terms of its low power requirements, we have to use the developed model to estimate the power consumption and compare the result to the capacity of a common power source.

For this example lets assume that we have a BOLT device which is connected to a Low-power Wireless Bus, sending one packet of 24 bytes every 500ms. The available power source is a common coin cell battery.

As we saw in section 4.2 we could therefore choose $T_{CXN} = 0.5s$ and let be $T_{ADV} = 4s$ for this purpose. This would mean that during one connection event the PRoC BLE would send one packet with 27 bytes of payload which should be feasible for all smartphones.

Furthermore we would assume a Tx power setting of 0 dBm for both the advertising as well as the connection phase. As a power supply we use a common coin cell, the CR2032. Those have a typical capacity of around 225 mAh.

Those are the parameters required to feed the model:

$$\begin{aligned} T_{ADV} &= 4000\,\text{ms} \\ T_{CXN} &= 500\,\text{ms} \\ T_{ADV,A} &= 7.409\,\text{ms} \\ T_{CXN,A} &= 3.952\,\text{ms} \\ P_{ADV,A} &= 25.196\,\text{mW} \\ P_{CXN,A} &= 16.181\,\text{mW} \\ P_{DS} &= 4.29\,\mu\text{W} \end{aligned}$$

Using eq. (4.7) and eq. (4.8) we can then calculate:

$$\begin{aligned} P_{ADV} &= \frac{T_{ADV,A}(P_{ADV,A} - P_{DS}) + T_{ADV} \cdot P_{DS}}{T_{ADV}} \\ &= \frac{7.409\,\text{ms}(25.196\,\text{mW} - 4.29\,\mu\text{W}) + 4000\,\text{ms} \cdot 4.29\,\mu\text{W}}{4000\,\text{ms}} \\ &= 0.051\,\text{mW} \\ P_{CXN} &= \frac{T_{CXN,A}(P_{CXN,A} - P_{DS}) + T_{CXN} \cdot P_{DS}}{T_{CXN}} \\ &= \frac{3.952\,\text{ms}(16.181\,\text{mW} - 4.29\,\mu\text{W}) + 500\,\text{ms} \cdot 4.29\,\mu\text{W}}{500\,\text{ms}} \\ &= 0.132\,\text{mW} \end{aligned}$$

Using the capacity of the coin cell we can calculate the energy stored:

$$E = 225\,\text{mAh} \cdot 3.3\,\text{V} = 742.5\,\text{mWh} \tag{4.9}$$

Only advertising, the device would last

$$\frac{742.5\,\mathrm{mWh}}{0.051\,\mathrm{mW}} = 14\,559\,\mathrm{h} = 607\,\mathrm{d} \tag{4.10}$$

and could stay connected for

$$\frac{742.5\,\mathrm{mWh}}{0.132\,\mathrm{mW}} = 5625\,\mathrm{h} = 234\,\mathrm{d} \tag{4.11}$$

For the given application, both of those values absolutely fulfil the requirements of the protocol gateway.

Chapter 5

Conclusions and Future Work

In this semester thesis the feasibility of the proposed protocol gateway has been successfully demonstrated. It involved the selection of suitable hardware components such as the Cypress Semiconductor PRoC BLE platform, selected from multiple commercially available BLE development platforms after a thorough investigation as well as the existing BOLT platform, serving as an asynchronous processor interconnect. Furthermore, software was developed and tested on two platforms, the PRoC BLE and an Android smartphone. The implemented protocol gateway was then subject to an evaluation, including the exact analysis of power traces as well as the derivation of a power state model based on the measurements taken. This power state model was then used to successfully show that the protocol gateway would indeed last for a sufficient amount of time while providing appropriate bandwidth.

This thesis sets the foundation for future work based on the developed protocol gateway, such as

- Implementation of a Low-power Wireless Bus sniffer. This would require the development of the LWB node sending data to BOLT as well as an extension of the Android app to display the data in an user-friendly way.
- Further improvement of the developed power state model. Even though the current model is sufficiently exact for the purpose, it could be improved, for example by considering the number of advertisement packets sent per advertisement event (currently fixed to 3) or the number of packets per connection event (currently fixed to 1)

Bibliography

[1] SIG Bluetooth. Specification of the Bluetooth system version 4.0. *Bluetooth SIG*, 2010.

[2] Federico Ferrari, Marco Zimmerling, Luca Mottola, and Lothar Thiele. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 1–14. ACM, 2012.

[3] Cypress Semiconductor. AN86233 - PSoC® 4 Low-Power Modes and Power Reduction Techniques, 2016. [Online; accessed 8-June-2016].

[4] Cypress Semiconductor. AN91162 - Creating a BLE Custom Profile, 2016. [Online; accessed 8-June-2016].

[5] Cypress Semiconductor. AN92584 - Designing for Low Power and Estimating Battery Life for BLE Applications, 2016. [Online; accessed 8-June-2016].

[6] Cypress Semiconductor. AN94020 - Getting Started with PRoC BLE, 2016. [Online; accessed 8-June-2016].

[7] Cypress Semiconductor. CYBL10X6X Family Datasheet: Programmable Radio-on-Chip With Bluetooth Low Energy (PRoC BLE) , 2016. [Online; accessed 8-June-2016].

[8] Cypress Semiconductor. Project #024: BLE Throughput - Pushing the Limits, 2016. [Online; accessed 9-June-2016].

[9] Felix Sutton, Marco Zimmerling, Reto Da Forno, Roman Lim, Tonio Gsell, Georgia Giannopoulou, Federico Ferrari, Jan Beutel, and Lothar Thiele. Bolt: A stateful processor interconnect. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 267–280. ACM, 2015.

Appendix A

A.1 Hardware Wiring

The pin header P5 on the BOLT breakout board provides the following connections, starting with pin 1:

- BOLT_SPI_SIMO
- BOLT_IND
- BOLT_SPI_SOMI
- BOLT_MODE
- BOLT_SPI_CLK
- BOLT_REQ
- BOLT_TIME_REQ (not used)
- BOLT_ACK
- BOLT_C_IND (not used)
- BOLT_FUTURE_USE (not used)
- GND

On the Cypress BLE Pioneer base board the following pins were used:

- P1.0 for BOLT_ACK
- P1.1 for BOLT_IND
- P1.2 for BOLT_REQ

- P1.3 for BOLT_MODE
- P0.0 for BOLT_SPI_MOSI / BOLT_SPI_SIMO
- P0.1 for BOLT_SPI_MISO / BOLT_SPI_MISO
- P0.3 for BOLT_SPI_CLK

Furthermore the BOLT platform was connected to the 3.3V rail provided by the Cypress BLE Pioneer base board.

Appendix B

B.1 Embedded Source Code Organisation

The embedded source code is written in C and managed with the PSoC Creator IDE. The main directory is 'PSoCProject/ST1'. The ST1.cywrk is the project file, referring to source code files stored in 'PSoCProject/ST1/ST1.cydsn'.

- bolt_ble_app.c/.h contain the BLE component
- bolt_if.c/.h contain the BOLT component
- Debug.c/.h contain helper functions to control GPIO pins which were used during the evaluation
- HandleLowPower.c/.h contain the low power component.
- main.c contains the main function

B.2 Android Source Code Organisation

The main directory for the Android app is 'AndroidBluetoothLeGatt'. This is a project generated and managed with the IDE Android Studio. The source code for the Android app is in the subdirectory 'Application/src/main/java/com/example/android/bluetoothlegatt'.

Appendix C

C.1 Abbreviations

BLE Bluetooth Low Energy
ECO External Crystal Oscillator
HCI Host Control Interface
IDE Integrated Development Environment
IFS Bluetooth Inter Frame Space, an interval of 150μs
IO Input /Output, referring to pins of CPUs
IoT Internet of Things
LWB Low-power Wireless Bus
MTU Maximum Transmission Unit
PRoC Programmable Radio on Chip, a SoC by Cypress Semiconductor
PSoC Programmable System on Chip, a SoC series by Cypress Semiconductor
QoS Quality of Service
SoC System on Chip
SPI Serial Peripheral Bus
SRAM Static Random-Access Memory,
UUID Universally Unique Identifier
WCO Watch Crystal Oscillator