Reading Dexcom CGM blood glucose levels using ESP32 SoC

Bachelor thesis by Max Kaiser
Date of submission: May 31, 2020

1. Review: Prof. Dr. Max Mühlhäuser
2. Review: M.Sc. Julien Gedeon
Under the supervision of M.Sc. Jens Heuschkel
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department
Telecooperation Lab

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Max Kaiser, die vorliegende Bachelor-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

English translation for information purposes only:

Thesis Statement pursuant to § 22 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Max Kaiser, have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

For a thesis of the Department of Architecture, the submitted electronic version corresponds to the presented model and the submitted architectural plans.

Darmstadt, May 31, 2020

M. Kaiser

Abstract

This thesis addresses the topic of reading blood glucose data from the Dexcom continuous glucose monitoring (CGM) system. The used hardware to communicate with the CGM system is a system on chip (SoC) microcontroller. The chosen device is from the Espressif Systems ESP32 microcontroller family. This gives the opportunity to read glucose levels portable. Also, the device is freely programmable to implement future algorithms or share glucose data with other devices. The goal is to implement an open-source project to provide real-time glucose levels for future projects. All communication with the CGM system is using the Bluetooth Low Energy (BLE) technology. The communication protocol between the ESP and the CGM hardware is implemented from the open-source xDrip project. Also, the optimization of the algorithm for power-efficient usage of the microcontroller is achieved in this thesis. This allows running the microcontroller portable with a battery.

Der Schwerpunkt dieser Arbeit befasst sich mit dem Auslesen von Blutzuckerwerten des continuous glucose monitoring (CGM) Systems der Firma Dexcom. Zur Kommunikation mit dem CGM System ist ein System on Chip (SoC) Mikrocontroller gewählt worden, spezieller gesagt die ESP32 Mikrocontroller Familie der Firma Espressif Systems. Das gibt die Möglichkeit die Blutzuckerwerte mobil auszulesen. Der Mikrocontroller gibt außerdem die Möglichkeit für spätere Anwendungen bezüglich Software und Hardware erweitert zu werden und die Blutzuckerwerte mit anderen Geräten teilen zu können. Das Ziel ist ein Open Source Projekt zur Bereitstellungen der Blutzuckerwerte in Echtzeit. Die Kommunikation funktioniert über die Bluetooth Low Energy (BLE) Technologie. Das Kommunikationsprotokoll zwischen dem ESP und der CGM Hardware ist anhand des Open Source Projekts xDrip implementiert worden. Ein weiteres Ziel dieser Arbeit ist die Optimierung des Mikrocontroller Algorithmus für eine energieeffizientere Nutzung und weniger Verbrauch. Das eröffnet die Möglichkeit den Mikrocontroller mobil mit einer Batterie zu nutzen.

Contents

1 Introduction **1**
1.1 Structure . . . 2

2 Diabetes Mellitus **4**
2.1 Blood Glucose Level Measuring . . . 4
2.2 CGM compared to FGM . . . 5
2.3 Insulin Injection . . . 6

3 Background **7**
3.1 ESP . . . 7
3.2 Bluetooth Low Energy . . . 8
3.3 Pairing and Bonding . . . 9
3.4 Power Optimization . . . 9
3.5 Sensor, Transmitter and Reader . . . 10
3.6 xDrip-js and xDrip+ . . . 12

4 Dexcom Protocol **14**
4.1 Request Response Message . . . 15
4.2 BLE Name and Service Advertisement . . . 15
4.3 Authentication . . . 15
4.4 Bonding . . . 17
4.5 Data Requests . . . 17
4.5.1 Time . . . 18
4.5.2 Glucose . . . 18
4.5.3 Sensor . . . 19
4.5.4 Battery and Temperature . . . 19
4.5.5 Last Calibration . . . 20
4.5.6 Glucose Backfill . . . 20

5 Implementation **22**
5.1 Bluetooth Low Energy . . . 22
5.2 Power Optimization . . . 23
5.3 Memory Management . . . 25
5.4 Connection Problem Handling . . . 25

6 Working Environment and Evaluation **26**
6.1 ESP32 and Arduino IDE . . . 26
6.2 Sensor Transmitter Emulation . . . 26
6.3 Software . . . 28

6.4 Testing 29

7 Conclusion **30**

8 Future Work **31**

List of Figures

2.1 Glucose Meter, Test Strip and Lancet 5
2.2 Insulin Pump & Insulin Pen 6

3.1 Espressif Systems ESP32 System on Chip 7
3.2 Bluetooth Low Energy - Central Peripheral 9
3.3 Dexcom G6 Sensor (top) and Transmitter (bottom) 11
3.4 Dexcom Reader & Dexcom G6 Android Application 12
3.5 xDrip+ Android Application 13

4.1 UUID Version 4.0 16

5.1 Indication & Notification 22

6.1 Laboratory Setup for Sensor Emulation 27
6.2 Eagle Circuit Diagram Sensor Emulation 27

List of Tables

3.1 ESP Component States in Power Mode . . . 10
3.2 Power Consumption for ESP Power Modes . . . 10

4.1 Dexcom G6 Bluetooth Low Energy Services . . . 14
4.2 Characteristics of the Protocol Service UUID . . . 14
4.3 Communication Message Structure . . . 15
4.4 Authentication Response at Auth Channel . . . 17
4.5 Keep Alive & Start Bonding Requests at Auth Channel . . . 17
4.6 Transmitter Status Codes . . . 17
4.7 Time Message at Control Channel . . . 18
4.8 Glucose Message at Control Channel . . . 19
4.9 Sensor Message at Control Channel . . . 19
4.10 Battery Message at Control Channel . . . 20
4.11 Last Calibration Message at Control Channel . . . 20
4.12 Backfill Message at Control Channel . . . 21
4.13 Backfill Transmission at Backfill Channel . . . 21

6.1 Transmitter Versions . . . 29

1 Introduction

Diabetes mellitus is one of the most common metabolic disorders and is characterized by a higher concentration of blood glucose in the patient's body over a long time. In 2017 about 425 million people worldwide were diagnosed with one type of diabetes mellitus [31]. More than 90 percent are type 2 diabetes patients [30]. Compared to type 1 diabetes, these patients can produce insulin by themselves to regulate the glucose level but have an insulin resistance which affects the regulation. Type 1 patients cannot produce insulin by themselves and rely on multiple insulin injections every day. For the therapy, it is important to measure the blood glucose level. A high blood glucose level over a long time will result in consequential damages in the cardiovascular system. Also, kidney, nerve, and eye damage are common. But the opposite can be much more dangerous for the patient in short term. A low glucose level, even over a short period of time, can lead to dizziness, coma, or in the worst case, to death. To raise the glucose level, carbohydrates are administered by food. To regulate a high blood glucose concentration, type 1 diabetics inject insulin throughout the day. The doses which are injected depend on the consumed food and the blood glucose concentration.
For this, it is important to measure the current glucose level. The manual way of doing this is using lancet & test strips. Using the lancet the skin on one side of a fingertip is penetrated and a drop of blood is applied to the test strip. A portable reader device for the test strip calculates and displays the glucose level. A much more practical way of measuring glucose concentration is by using a sensor that is applied onto the patient's body. These sensors have a short needle that reach under the skin into the interstitial fluid. There it is also possible to measure a glucose level which will reflect the blood glucose concentration. The benefit of this stationary sensor is that no blood sample has to be taken every time the glucose level is measured. This also allows more frequent measurements which help to detect extreme glucose concentrations faster. Measurements on a regular interval are statistically better evaluable for therapy and research purposes. The higher amount of glucose data available through these sensors opens up new possibilities for scientific work. For example, predicting future glucose trends using deep neural networks and developing closed-loop systems.

Continuous glucose monitoring (CGM) systems that consist of this type of sensor do normally not have an open communication interface to access the glucose data. The glucose data can only be displayed by the portable reader device which is part of the CGM system. Most of the CGM and FGM systems allow reading glucose data also with the included smartphone application. But the main purpose of the reader device and the smartphone application is to show the current glucose level and displaying a graph of historical values for therapy support. Excessive data export other than for therapeutic purposes is rarely supported. This makes it complicated for scientific projects to use this data.

Multiple solutions are provided by open source applications that can communicate with these CGM systems. However, these are neither created nor backed by the manufacturer. Most of them use the original communication channel and protocols which were reverse-engineered from the smartphone applications. They do not sniff or monitor the communication from the system components to the reader device to get glucose data. This means the original reader device is not needed when using these open-source solutions. The currently available software for this purpose is either for PC (Windows or Linux) or smartphones. This

results in that either it is not portable or that it is not power efficient enough to run multiple weeks to collect data. Apart from this, the device has to be in close proximity to the glucose system to read data.

In this thesis, the Dexcom G6 CGM system is used. This system communicates over Bluetooth Low Energy which gives a theoretical range of a few meters. It is desirable to have a device that is small enough to keep it all the time at the patient but which is also energy efficient enough to run multiple weeks on the same battery. It should also be easily programmable and expandable for own algorithms and future research projects.

Best suited to these requirements is a small system on chip microcontroller like the ESP32 chip from Espressif Systems. It has a built-in module for Bluetooth Low Energy and can run with batteries. It can be programmed in C/C++ and has many libraries and support. It is used in this thesis as a reader device for the Dexcom G6 CGM system. The ESP manages all the communication with the G6 transmitter. It can read the glucose values, but also data for monitoring the CGM systems health state, like battery status or runtime. The ESP saves the glucose values and has the potential to share these data over open protocols like Bluetooth or Wi-Fi. Also saving the glucose values on removable storage could be done with the ESP32.

1.1 Structure

First of all, the metabolic disorder diabetes mellitus is explained, i.e. what is characteristic of this disease and which health related problems can be caused by a low or high blood glucose concentration. The common treatment method which includes measuring the glucose concentration and injecting insulin is shown.

Next, some basic knowledge about the ESP and CGM system is described. It starts with the ESP32 and its features to give an overview of what is all possible with this chip. Then some Bluetooth Low Energy basics are described, because this transmission concept is different compared to normal Bluetooth or serial communication. These basics are necessary to understand how the Dexcom protocol works. Also, the paring and bonding concept in Bluetooth Low Energy is explained a little bit further, because some problems can happen with this. After this, ESP power optimization techniques are explained and later it is described which mode is chosen for this project. The next section explains the components of the Dexcom G6 CGM system. Which device is involved in the communication and what the sensor looks like. The last section is about the already available open-source software which can communicate with the G6 system.

Next there is one chapter about the details of the Dexcom protocol and how it is used with the Bluetooth Low Energy characteristics. From finding the right transmitter up to decoding messages byte by byte to parse the data. Also, the security features and authentication process is explained. Selected tables show what the protocol messages consist of and how the data is transmitted. The backfill mechanism to receive older glucose values is also explained in detail.

Following some software concepts are explained which do not necessarily depend on the protocol, like memory or problem handling. Some Bluetooth Low Energy and transmitter specific difficulties are explained which occurred during the development of the protocol implementation.

The next chapter is about the working environment and the developing cycle. The software environment for the ESP and an instruction for running xDrip-js [9] under Linux is outlined. Furthermore, the emulation of the G6 sensor for setting up a testing environment is shown. This is useful when the transmitter should be

used without a sensor.

In conclusion, the accomplished functionality and future use are discussed. Additionally, possible goals to implement are shown, which should be taken into account when this project is used for future research.

2 Diabetes Mellitus

Diabetes mellitus, or better known short as diabetes, is one of the most common groups of metabolic disorders. For 2017 the International Diabetes Federation (IDF) reported a worldwide total of 425 million patients with diabetes. In Germany, about 10 percent of the total population have one diabetes disorder which is in more than 90 percent of the cases type 2 diabetes [30]. Every year there are about 500.000 new diabetes diagnoses only in Germany [31]. Diabetes can be grouped into three main types of this disease. Type 1, type 2, and gestational diabetes. The latter is quite rare and is therefore not mentioned further in this thesis. All types are characterized by a high blood glucose concentration over an extended period of time. The higher glucose level is due to a lack of insulin produced in the body or an insulin resistance. Insulin is a peptide hormone that is produced in the pancreas. It controls the metabolism of carbohydrates, protein, and fats by allowing the muscles to absorb glucose when needed. In my thesis, I refer mainly to type 1 diabetes mellitus.

Type 1 diabetes is caused by an autoimmune response. The pancreas is failing to produce enough insulin for regulating the body's metabolism. This is due to a loss of beta cells which are located in pancreatic islets of the pancreas. This type of diabetes can only be treated by multiple insulin injections every day.

Type 2 diabetes is the more common case with about 90 percent of all diabetes diagnoses, because of lifestyle factors. At the beginning of the disease, the body develops an insulin resistance which can be caused by the combination of excessive body weight and a lack of physical activity. In more advanced cases also a lack of produced insulin can occur. Due to the different severity levels of the disease, multiple treatment methods are used. From oral medication in lighter cases to insulin injections equal to type 1 diabetes therapy when not enough insulin can be produced.

2.1 Blood Glucose Level Measuring

One of the most important steps in diabetes therapy is measuring the glucose level frequently. Only when the current glucose concentration is known, further actions can follow. Measuring the glucose concentration can be done in two ways. Directly by measuring the blood glucose level using a blood sample or by measuring the concentration in the interstitial fluid. The method using the interstitial fluid has the benefit of allowing the use of a sensor that is applied onto the patient's body which measure multiple times. The disadvantage is that the glucose concentration in the interstitial fluid is delayed by a few minutes compared to the blood concentration [26]. But this is in most of the therapy cases, not of concern.

For a healthy person, the normal consistent range of the glucose level in the blood is from 4.4 to 6.1 mmol/L which is about 79 to 110 mg/dL [27]. Values outside this range can be dangerous. A high blood glucose level over a long time will result in consequential damages in the cardiovascular system. Also, kidney, nerve, and eye damage are common. This can be life-threatening when diabetes mellitus is not detected or not treated

right. A too low glucose level has short-term effects. Even over a short period of time, a low glucose level can lead to dizziness, coma, or in the worst case, to death.

Therefore it is necessary to measure the glucose concentration. The most common way for the blood measurement is using a lancet and test strips. First of the patient has to clean one fingertip for the measurement. Then often a lancet device is used which helps to penetrate the skin layer of a fingertip to reach the blood. The test strip is inserted into the glucose meter and then the patient has to apply a drop of blood onto the test strip to get a measurement. The first picture shows the lancet and lancet device, glucose meter, and the test strip. The alcohol swap is for cleaning the fingertip before using the lancet. The second picture demonstrates applying the blood sample to the test strip.

The advanced method is using an interstitial fluid sensor used in CGM and FGM systems. This sensor is placed onto the skin and has a short needle that will reach into the interstitial fluid. These sensors can measure multiple times and have the advantage that no blood sample has to be taken. The sensor itself does not have a display or any other components for direct patient interactions. The sensor communicates with a special reader device or a smartphone which then displays the current glucose level. More details are presented in the next section about the two types of this system.

Figure 2.1: Glucose Meter, Test Strip and Lancet

Graphic by HealthHub [13]

Graphic by Ken Research [22]

2.2 CGM compared to FGM

Flash glucose monitoring (FGM) or also known as intermittent scanning continuous glucose monitoring (iscCGM) is one method of continuous glucose monitoring. The main difference compared to the classic method using lancet & test strips is that the patient does not have to take a blood sample every time he wants to measure the glucose level. A small sensor is placed onto the body which measures the glucose level through a needle that reaches a few mm deep into the skin. To read the glucose level the patient has to “scan” the sensor with a smartphone or reader device. This works by placing the reader device on top of the sensor to create a wireless communication.

Continuous glucose monitoring (CGM) is similar to FGM. It also works with a sensor that is applied onto the body. The main difference is that the glucose level is transmitted automatically to the receiver devices. With the Dexcom G6 system, this happens every five minutes [33]. The disadvantage compared to the manual or FGM method is that the glucose level is delayed by a few minutes [26].

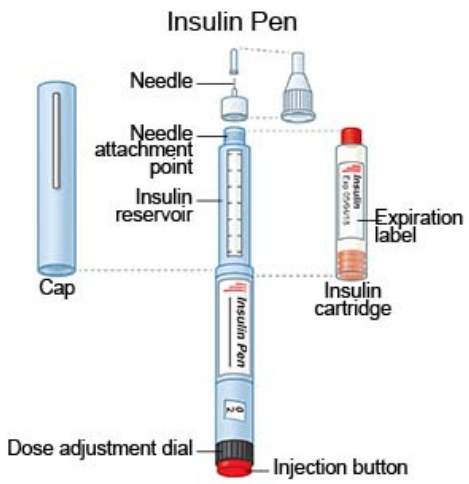
The sensors of both types have a limited lifespan and must be replaced frequently. For the Dexcom G6 system, the sensor has a lifetime of a maximum of 10 days [12].

2.3 Insulin Injection

After the glucose level is measured and not within in the valid range, actions have to be taken to avoid long term damages. When the glucose level is too low, carbohydrates can be administered via food or beverage. In most of the cases it is enough when the patient consumes some products with a high carbohydrate concentration. But in more serious situations when the level decrease faster, pure dextrose can be used.
In the other case, when glucose levels are too high or the patient is planning to eat some meals which will increase the glucose level, insulin is injected to prevent or correct high glucose concentrations. In the past insulin injections were done using a syringe. But this method is not very convenient when multiple insulin injections are needed throughout the day. Today the tool of choice is an insulin pen that combines an insulin reservoir and an injector as shown in the graphic below 2.2. This facilitated the process for every day insulin injections. Even more advanced is an insulin pump. It is a small electronic device that will inject insulin from a reservoir into the body through a catheter. The advantage is that the catheter can be used multiple days and replaces the intraday injections with the pen. Also, the insulin pump can communicate with a glucose meter and CGM / FGM systems. This is useful for simplifying the process of correcting the glucose level.

Figure 2.2: Insulin Pump & Insulin Pen

Graphic by FAULHABER [23]



Graphic by drugs.com [14]

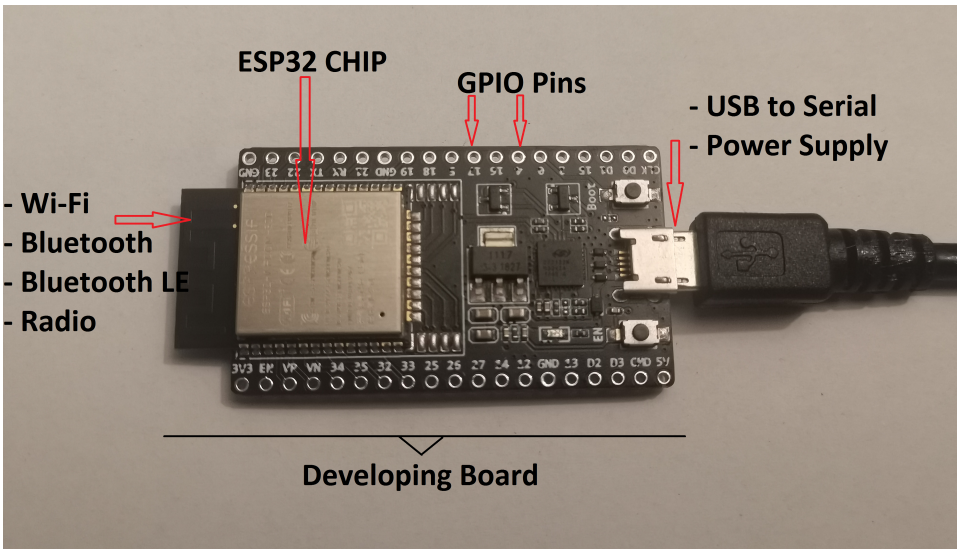
3 Background

This chapter explains some basic knowledge and terms of the ESP and continuous glucose monitoring hardware which is used in diabetes therapy. These pieces of background information are useful to understand further concepts of this thesis.

3.1 ESP

The objective of this bachelor thesis is how to communicate with the Dexcom G6 system using a small portable device. For PC and smartphones such software is already available. But these devices are not portable or have a short battery life of a few days in maximum. The goal was to implement it on an ESP32 to have a small device that can run for month on the same battery. The size of the ESP is about the size of a 2-euro coin. The developing boards are bigger because of the additional hardware like USB to serial converter and modules which helps in the development cycle. But this optional hardware is not necessary for the core ESP and can be omitted when the ESP is used in the real environment. The ESP is a System on Chip (SoC) with a dual or single-core Xtensa 32-bit microprocessor. It has built-in modules for Wi-Fi, Bluetooth and Bluetooth Low Energy, Ethernet, GPIO pins, PWM, CAN bus, and many more [7].

Figure 3.1: Espressif Systems ESP32 System on Chip



Graphic by Max Kaiser

The code for the ESP can be written either in *C/C++* with the Arduino [1] or the ESP-IDF (Espressif IoT Development Framework) [5] Platform. But also in *Lua* using the NodeMCU framework or *Python 3* with MicroPython [24]. The compiled code is flashed onto the ESP's persistent memory and can then be used independently from any other software and hardware which are needed for the development.

3.2 Bluetooth Low Energy

When called BLE, Bluetooth LE, or simply Bluetooth in this thesis, this stands for the Bluetooth Low Energy wireless PAN (personal area network) technology. It is mostly used for Internet of Things (IoT) devices like health and fitness trackers or smart home products. It was developed to have a communication protocol that is power and cost-efficient and therefore suitable for small devices that will run on batteries. Bluetooth Low Energy is quite different from the normal Bluetooth connection. The structure can be compared to a web server with web pages. Multiple users can connect and read information from different web pages.

Central can be seen as a client who wants to connect to the peripheral to read or write some data. This is most of the time a smartphone or other device to control the peripheral.

Peripheral is like a server that has data or receives commands from the clients. Peripherals are for example heart rate monitors, fitness trackers, IoT light bulbs, or temperature controls.

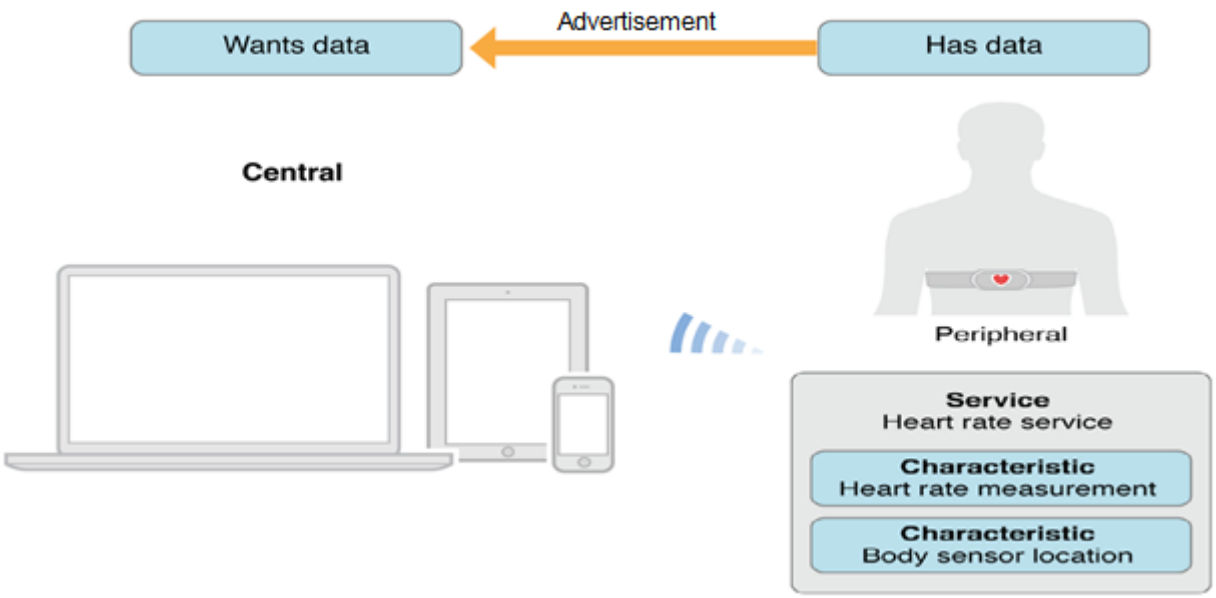
A **Service** is a collection of characteristics and has an ID instead of a name. This ID is a 16-bit or 128-bit universally unique identifier (UUID). A fitness tracker could be split into two services. One service for the heart measurements and one for the distance/step measurements.

Characteristics are like variables of the peripheral. They hold data. Also, the characteristics have a 16 or 128 bit UUID to be distinguished.

The **Properties** of a characteristic describes what the central devices can do with the characteristic. For example, READ, WRITE, NOTIFY, or INDICATE are possible. The **Descriptor** is a special property and contains more metadata regarding the characteristic. Notification and Indication are some mechanisms that will let the central device know when the value of the characteristic has changed, so the central does not have to check it by themselves.

The peripheral device sends advertisement packages over Bluetooth Low Energy to indicate that he is active and capable of handling client devices. In these packages, he advertises one service called an **Advertisement Service**. This helps the central devices to find the correct peripheral they are interested in.

Figure 3.2: Bluetooth Low Energy - Central Peripheral



Base Graphic by iWave Systems [32]

3.3 Pairing and Bonding

Paring in the context of a BLE connection is establishing an encrypted connection and exchanging information. This means authenticating the identity of each device and distributing temporary keys. Also, information about the input and output capabilities of the devices are transmitted. This information is used to decide the paring mode. For example devices without any input or output capabilities cannot use paring mode with a pin because of the lack of displaying or entering a pin. That also applies to the ESP because it is used in this thesis without a display or keys [19].

Bonding can be done after successful paring with the other device and means to save the previously exchanged keys. These keys can then be used for the next connection of these devices and act as a pre-shared secret. The paring process doesn't have to be done the next time and every party uses the stored keys right from the beginning [19].

3.4 Power Optimization

When the ESP is used in the real environment the power supply is a battery. In contrast to using it on a power adapter, it is then important to run the ESP efficiently when running on batteries. This saves energy and allows a longer runtime before the battery must be replaced. To achieve lower power consumption the ESP has some build-in power optimization techniques, like different sleep modes which can be compared to the standby function on TVs and computers. The device is not turned off but is also not running completely. It can

run some small tasks or wait for external interaction. Below the table 3.1 shows which components of the ESP are shut down in which deep sleep stage. The other table 3.2 shows the sleep modes and power consumption compared with fully active and shutoff chip.

Table 3.1: ESP Component States in Power Mode

Power mode	Active	Modem-sleep	Light-sleep	Deep-sleep	Hibernation
Sleep pattern	Association sleep pattern			ULP sensor-monitored pattern	-
CPU	ON	ON	PAUSE	OFF	OFF
Wi-Fi/BT baseband and radio	ON	OFF	OFF	OFF	OFF
RTC memory and RTC peripherals	ON	ON	ON	ON	OFF
ULP co-processor	ON	ON	ON	ON/OFF	OFF

Original table by the ESP32 datasheet [7] and graphic from here [6].

Table 3.2: Power Consumption for ESP Power Modes

Power mode	Description	Power consumption
Active (RF working)	Wi-Fi Tx packet 14 dBm ~ 19.5 dBm	Please refer to Table 10 for details.
	Wi-Fi / BT Tx packet 0 dBm	
	Wi-Fi / BT Rx and listening	
Modem-sleep	The CPU is powered on.	Max speed 240 MHz: 30 mA ~ 50 mA
		Normal speed 80 MHz: 20 mA ~ 25 mA
		Slow speed 2 MHz: 2 mA ~ 4 mA
Light-sleep	-	0.8 mA
Deep-sleep	The ULP co-processor is powered on.	150 μA
	ULP sensor-monitored pattern	100 μA @1% duty
	RTC timer + RTC memory	10 μA
Hibernation	RTC timer only	5 μA
Power off	CHIP_PU is set to low level, the chip is powered off	0.1 μA

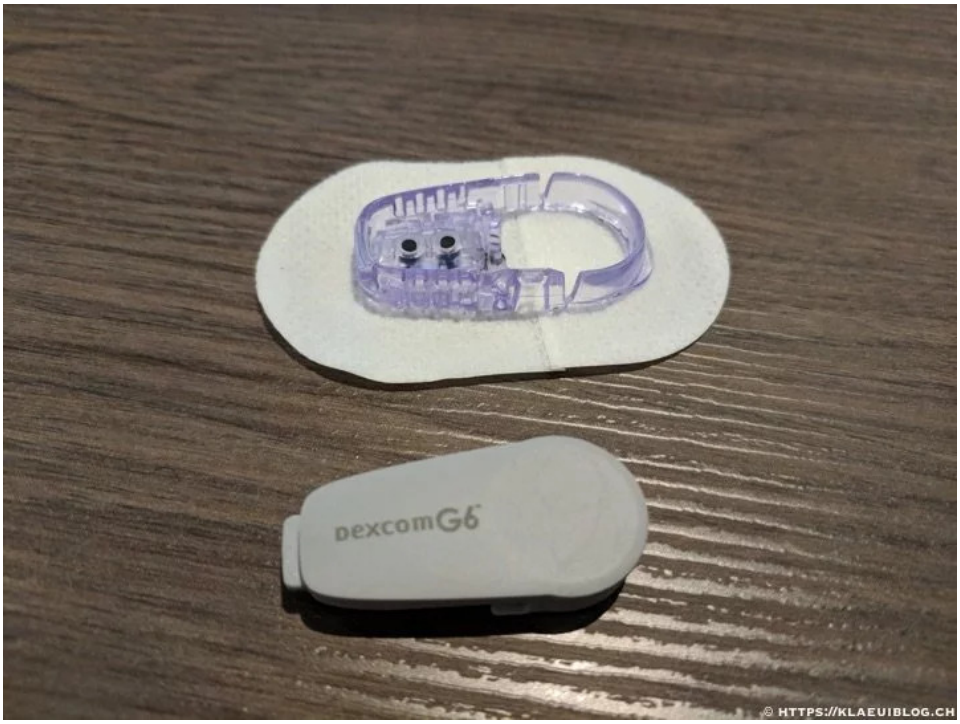
Original table by the ESP32 datasheet [7] and graphic from here [6].

3.5 Sensor, Transmitter and Reader

The Dexcom G6 Real-Time Continuous Glucose Monitoring (rtCGM or short CGM) system consists of three main components. One is the glucose sensor (shown in the upper side of the picture 3.3) which is inserted into the patient's body using an applicator. It has a short needle to reach the interstitial fluid a few millimeters under the skin. This sensor can be used up to 10 days before it must be replaced [12]. The sensor itself has a two pin connector and does not have any logic components.

The transmitter (shown in the bottom side of the picture 3.3) is the heart of the system. It interprets the electric characteristics of the sensor and calculates the corresponding glucose level. It also manages the communication with reader devices. The transmitter also has different features like calibration, backfill, predict glucose trends, and many more.
The third component is the patient's reader device which connects to the transmitter and displays the data. This device is most likely the Decom G6 receiver (see graphic 3.4) or a smartphone using the Dexcom G6 application [4]. But it could also be a compatible insulin pump like the t:slim X2™ Insulin Pump [28]. An unofficial aftermarket alternative is software like xDrip-js [9] for PC or xDrip-plus [20] for Android. The client device allows the user to read the glucose levels from the transmitter but is also used to warn when predefined dangerous glucose levels are reached.

Figure 3.3: Dexcom G6 Sensor (top) and Transmitter (bottom)



Graphic by Andreas Klaui [17]

Figure 3.4: Dexcom Reader & Dexcom G6 Android Application

Graphic by Scott Jung [15]

Graphic by Dexcom [3]

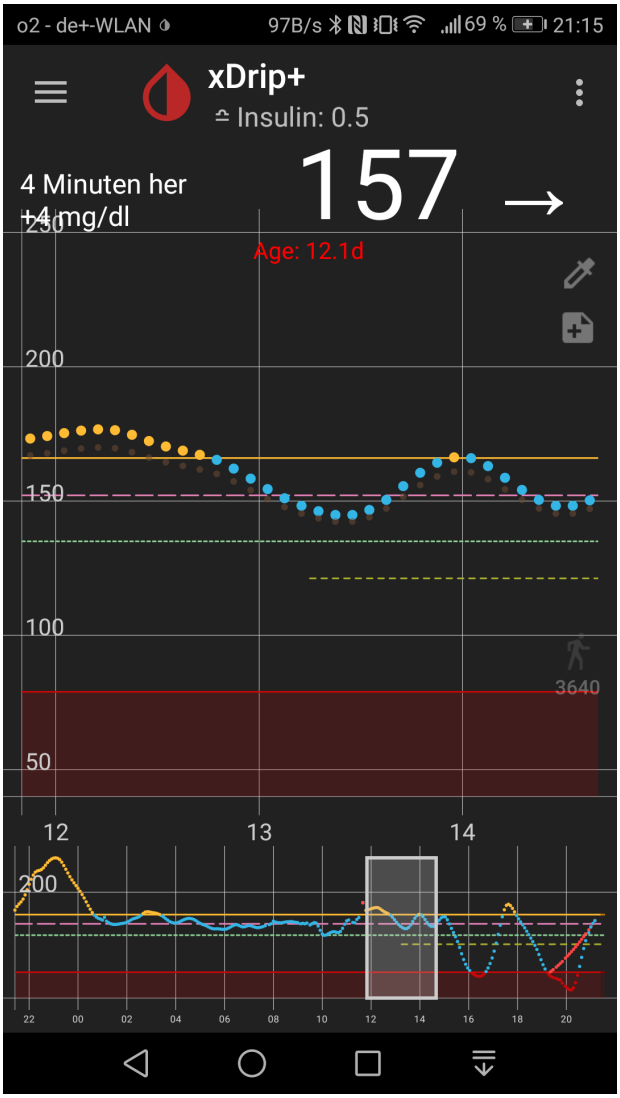
3.6 xDrip-js and xDrip+

xDrip [2] is open-source software that can communicate with the Dexcom CGM product family. The communication protocol was reverse engineering by the community and is neither created nor supported by Dexcom. The software uses Bluetooth Low Energy to send and receive data from the transmitter. xDrip-js [9] is the JavaScript version of this software. It can run under Windows and Linux and supports the main features like receiving glucose levels, battery and session data, calibration, backfill, transmitter restart, and some more.

xDrip+ or xDrip-plus [20] is an Android application implementing amongst other CGM systems the xDrip algorithms. It runs on most smartphones supporting BLE. It is mostly used as a replacement for the official Dexcom CGM app [4] or the Dexcom receiver. It supports all xDrip features and logs glucose values which are shown in a graph. But xDrip+ does also support the glucose systems of other companies. Below is a picture of the xDrip+ app 3.5.

The data transmission between the ESP and the G6 transmitter in this bachelor thesis is based on the Dexcom communication protocol from the xDrip-js code [9].

Figure 3.5: xDrip+ Android Application



Graphic by Martin Schiftan [25]

4 Dexcom Protocol

The Dexcom protocol uses different BLE characteristics which can be seen as separate communication channels. These are used for different tasks. Below is a table 4.2 of the characteristics and their purpose. These characteristics are categorized under the main protocol service universally unique identifier (UUID). All service UUIDs are shown in the first table 4.1. The Dexcom protocol and all communication with the transmitter in this thesis are based on the open-source project xDrip-js [9].

All UUIDs in the second table 4.2 are 128 bits long and end with "-849E-531C-C594-30F1F86A4EA5".
"**G6**" stands for all transmitters with serial numbers start with 80 or 81.
"**G6 Plus**" is the newer transmitter model with 8G, 8H, or 8J serial number prefixes.
The column shows the available properties for that characteristic.

Table 4.1: Dexcom G6 Bluetooth Low Energy Services

UUID	Description
f8083532-849e-531c-c594-30f1f86a4ea5	The service UUID for the protocol characteristics.
0000febc-0000-1000-8000-00805f9b34fb	This is the advertised service to find the transmitter.
180A	The general BLE service with device information.

Table 4.2: Characteristics of the Protocol Service UUID

UUID	Name	G6	G6 Plus	Description
F8083533	communication	NOTIFY, READ	NOTIFY, READ	special communication.
F8083534	control	INDICATE, WRITE	INDICATE, WRITE	command & response.
F8083535	authentication	INDICATE, READ, WRITE	INDICATE, WRITE	authentication.
F8083536	backfill	NOTIFY, READ, WRITE	NOTIFY	receive backfill data.
F8083537	unknown	READ	READ	
F8083538	Unknown	- not available -	NOTIFY, READ	

For the newer transmitters, the characteristics have slightly changed. This could be because some of them are implicit and therefore not used in the protocol. For example, READ is implicit in NOTIFY. It is not needed to read the value manually when getting notified for each value modification. Furthermore, the WRITE attribute for the backfill characteristic is not used. All commands are send and received over the control characteristic. The backfill channel is only used to receive the backfill glucose date.

4.1 Request Response Message

The main communication is based on a request-response method. The client sends a request or receives a response which is structured as followed:

Table 4.3: Communication Message Structure

Byte 0	Byte 1 to N-2	Byte N-1 and N
Opcode	Optional Data Payload	Error Detecting Code

Important to know is that all values in the request and response are in little-endian byte order. Thus, to convert the data on the ESP directly into a datatype, the order of the bytes is flipped first.
The length of the message is different for each command but has a predefined size. The size depends on the payload data which represents the parameters for the command.
The operation code (opcode) is one byte which represents the action that is requested. For most of the commands, the opcode value also represents the sequence they are used in. For example, the opcode for the authentication, keep alive, and bonding range from 0x01 to 0x07. These are also the first things to do, in this order, when starting a communication.
Following the opcode is the data payload. This data is optional and only in use by few requests. Much often used in the responses for transmitting the requested data. An example is the backfill request. There the request payload contains the start and end date from which the glucose values are returned.
The last two bytes of each message is an error detecting code. For this protocol, a 16-byte cyclic redundancy check (CRC) is used. The 32 byte CRC function is for example used in the Ethernet protocol and some more [18]. But in this case, the more specific **CRC-16-XMODEM** mode is used. But the CRC result bytes are in reverse order. This is because of the little-endian byte order. The CRC calculation is done over the opcode and the payload data.

4.2 BLE Name and Service Advertisement

To identify the transmitter there are two main aspects. First, the Bluetooth name starts with “Dexcom” followed by the last two digits of the serial number. The second aspect is that the transmitter also advertises an empty service. This service is the same for all transmitters and only for the BLE advertisement. This service does not have any characteristics to interact with. All communication is through the protocol service.

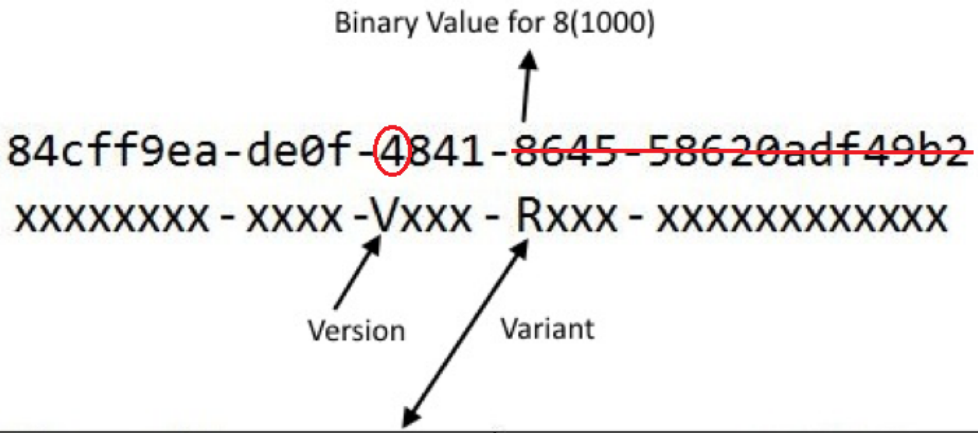
The transmitter wakes up every five minutes from a sleep/power-saving state to allow a BLE connection. During this connection, multiple commands can be sent. Two devices can be used concurrently (every wakeup cycle) to request and send data to the transmitter.

4.3 Authentication

The first step after the Bluetooth communication is established, is the authentication process. This process is a challenge-response authentication between the transmitter and every client that connects. This authentication

process is done after every wakeup period of the transmitter individually by all clients when they connect. This authentication is also necessary when the client and transmitter are bonded on the Bluetooth LE layer. The pre-shared secret is the serial number. The transmitter already knowns this number because it is set in the factory. For the patient's client devices the serial number is printed on the backside of the transmitter or onto the package. It is printed on the backside for security reasons so it cannot be read when the transmitter is used and therefor inserted into a sensor. Both client and transmitter create a challenge for each other by taking the first 8 bytes of a UUID version 4. This version is (pseudo) randomly generated and has in the first 8 bytes only 4 bits fixed which represents version 4 of the unique identifier. These graphic 4.1 shows the used bits from the UUID.

Figure 4.1: UUID Version 4.0



Msb0	Msb1	Msb2	Description
0	x	x	Reserved, NCS backward compatibility.
1	0	x	Leach-Salz
1	1	0	Reserved, Microsoft Corporation Backward Compatibility.
1	1	1	Reserved for future definition.

Original graphic by Pankaj [21]

This challenge is then encrypted by the other party using the **Advanced Encryption Standard** (AES) with 182 bit in **Electronic Code Book** (ECB) mode. The key used for encryption is the pre-shared secret. The ciphertext is then sent back and the client/transmitter can check if the other party also knows the secret. This is done by also encrypting his own generated challenge with his secret key and check if the ciphertext matches the response from the other party. This challenge-response authentication is done in both directions to ensure that the transmitter only accepts clients which knows the secret but also that the client knows he is connected with the right transmitter which has his desired serial number. This authentication process uses the first opcodes ranging from 0x01 to 0x05. The challenge and ciphertext are located in the data payload. Important to know is that these messages do not have a CRC error code attached. This is because when a bit flips in the transmission the authentication automatically gets declined because then the ciphertexts do not match anymore. When the authentication was finished the transmitter sends a 3-byte message with the opcode 0x05 which indicates if the authentication was successful and if a bonding should be initiated.

Table 4.4: Authentication Response at Auth Channel

1. Byte (opcode)	2. Byte (authenticated)	3. Byte (bonding)
0x05	0x01 OK 0x02 FAILED	0x01 YES 0x02 NO

4.4 Bonding

When the authentication is successful the transmitter indicates if the client should initiate a Bluetooth LE bonding process with the transmitter using the opcode 0x05 message (see above table 4.4). When bonding not exists the transmitter normally asks for a new bonding. In this case, the client first sends a keep-alive message (0x06) followed by the bonding command (0x07). These messages don't generate a response from the transmitter in the form of a message and are sent right after each other.

Table 4.5: Keep Alive & Start Bonding Requests at Auth Channel

Message	1. Byte (opcode)	2. Byte (payload)
Keep Alive	0x06	0x19
Start Bonding	0x07	

After that, the transmitter and the client are bonded on the BLE level. On some Bluetooth devices, there are problems when trying to use the bonding information for the next connection. With xDrip-js [9] on some Linux distributions, this problem occurs and can be solved by unpairing the transmitter before every new connection cycle. This also deletes the bonding information to force a clean new connection for the next time. But when using xDrip-plus on an Android device this problem does not occur.

4.5 Data Requests

All of the following commands do not have any parameters and are read-only. This means it is not possible to change the transmitter state. The messages are starting with one byte for the opcode and followed by two bytes for the CRC error code. The time request is the only mandatory command which is needed for future requests. The current transmitter time is needed for the backfill command. And also to convert the timestamps from glucose, sensor, and last calibration messages into real-time timestamps. Most of the responses have a status code which represents the state the transmitter is in.

Table 4.6: Transmitter Status Codes

Code	Status
0x00	OK
0x81	Low Battery
0x83	Expired

4.5.1 Time

The first command to request is the time. The time command gives the current timestamp and also the time when this sensor session was started. This means the time when the current sensor was applied into the patient's body. Because from then on a new session is starting. The time format is the time since the transmitter was started in seconds. In the ESP algorithm, the time response is the first command which receives the transmitter's status code. The status code is used to inform the user when the battery is low or the transmitter is expired. This means that the end of the official warranty period for the transmitter (not the sensor) is reached and the transmitter must be replaced.

Table 4.7: Time Message at Control Channel

Opcode	CRC
0x24	0xE6 0x64

Request (3 byte)

Opcode	Status	Current Time	Session Start Time	Unknown	CRC
0x25	1 byte	4 byte	4 byte	4 byte	2 byte

Response (16 byte)

4.5.2 Glucose

This is the most important command and returns the current live glucose level for the patient. This command is a little bit different for the G6 and the older G5 models. The difference is the request and response opcode. From this follows also a different CRC error code. The request and associated response are shown below for the two transmitter models.

Status represents the transmitter status as in the other commands. Sequence is a consecutive number which counts how often the glucose level was requested. The timestamp contains the time when the glucose level was measured by the transmitter. This is most of the time a view seconds back compared with the current time received by the time command. I suspect that the transmitter will measure the glucose level right before a possible Bluetooth connection to have it ready when a client connects. Next is 2 bytes for the glucose level. As a quick reminder, all values are in little-endian byte order and the order must be flipped when used on a big-endian architecture like the ESP32 SoC.
State is the sensor state. It represents the condition of the sensor. The byte 0x06 stands for everything working correctly. It is recommended to check this state before trusting the glucose level. For example some other states can be: **stopped** (0x01), **warmup** (0x02), **calibration needed** (0x07) or sensor/session/calibration errors [34]. The last field before the CRC code is one byte which indicates the trend in which the glucose level could change. This is calculated by the internal algorithm and the history of values.

Table 4.8: Glucose Message at Control Channel

Model	Opcode	CRC
G5	0x30	0x53 0x36
G6 (G6 Plus)	0x4E	0x0A 0xA9

Request (3 byte)

Model	Opcode	Status	Sequence	Timestamp	Glucose	State	Trend	CRC
G5	0x31	1 byte	4 byte	4 byte	2 byte	1 byte	1 byte	2 byte
G6	0x4F	1 byte	4 byte	4 byte	2 byte	1 byte	1 byte	2 byte

Response (16 byte)

4.5.3 Sensor

This request is only useful with older transmitters because only on these the response contains the raw values of the sensor measured by the transmitter. The newer G6 models do not support this feature anymore. The glucose level gets calculated by the internal algorithm, therefore there is no need for receiving the raw sensor values. This could be one reason why this feature was suspended.

Table 4.9: Sensor Message at Control Channel

Opcode	CRC
0x2E	0xAC 0xC5

Request (3 byte)

Model	Opcode	Status	Timestamp	Unfiltered Raw	Filtered Raw	CRC
G6 old firmware	0x2F	1 byte	4 byte	4 byte	4 byte	2 byte
G6 new firmware	0x2F	1 byte	4 byte			2 byte

Response (16 byte / 8 byte)

4.5.4 Battery and Temperature

The battery command is similar to the sensor command. The response also depends on the firmware version of the transmitter and the model. The older firmware also returns the battery resistance R. The status is the same as on the other commands and represents the transmitter state. Voltage A and B are two values which indicate the remaining battery runtime. The temperature is in degrees Celsius and can be used to check if the transmitter is in a working temperature range. Also, it indicates if the transmitter is placed on the body (nearly body temperature) or stored somewhere. For example, when the temperature drops but the transmitter is still placed into the sensor this can mean that the patient is in a cold environment or in water which cools down the transmitter. The other case is more problematic when the transmitter heats up. This could potentially cause damage to the transmitter and or sensor and therefore it is useful to monitor the temperature changes.

Table 4.10: Battery Message at Control Channel

Opcode	CRC
0x22	0x20 0x04

Request (3 byte)

Model	Opcode	Status	Voltage A	Voltage B	R	Runtime	Temp.	CRC
G5 / G6 old firmware	0x23	1 byte	2 byte	2 byte	2 byte	1 byte	1 byte	2 byte
G6 Plus new firmware	0x23	1 byte	2 byte	2 byte		1 byte	1 byte	2 byte

Response (12 byte / 10 byte)

4.5.5 Last Calibration

It is possible to receive the last glucose value which was used to calibrate the sensor. This command's response contains the value and the time when the calibration was accepted. This request can be useful to check if the last calibration was too long ago and the user should enter a new calibration.

Table 4.11: Last Calibration Message at Control Channel

Opcode	CRC
0x32	0x11 0x16

Request (3 byte)

Model	Opcode	Unknown	Glucose	Timestamp	Unknown	CRC
G5	0x33	10 byte	2 byte	4 byte		2 byte
G6	0x33	10 byte	2 byte	4 byte	1 byte	2 byte

Response (19 byte / 20 byte)

4.5.6 Glucose Backfill

The glucose backfill command is for receiving the last glucose data. Sometimes it happens that the client did not connect to read the current glucose level. This can happen when the client device is out of range, turned off, or has a problem connecting to the transmitter. Also when the client is not time-synchronized so it misses when the transmitter wakes up. To receive these missed glucose levels the backfill command is used. It can return with a resolution of 5 minutes (every transmitter wakeup cycle) up to the last ~12 hours of data depending on the transmitter firmware version. This command is a little bit different from the other Dexcom commands. It uses two BLE characteristics at the same time, the control and backfill channel. The control characteristic is used the same as on the other commands. The request is sent here. In the payload, the start and end time is transmitted from which range the glucose levels are then returned. The glucose levels are then transmitted over the backfill characteristic. This is because potentially a lot of glucose levels could be requested at once. The response is spitted into multiple blocks with a size of 18 bytes. Then for each block, a 1-byte sequence and 1-byte identifier number are added. The last block is not filled up so it could be shorter than 20 bytes. They are then transmitted using the notify BLE mechanism. The client then combines

all blocks in the right order and can parse the glucose data. The order is determined by the sequence number. The glucose values are transmitted from the oldest to the newest. When all backfill blocks are transmitted a second response is sent through the control channel with information about the start and end time and a CRC code. The actual start and end time can be different from the requested ones. This is when the requested time does not have a glucose value because it is in between two wakeup cycles. Then the nearest timestamp is used which is still in the requested time range.

Table 4.12: Backfill Message at Control Channel

Opcode	Fixed	Start Time	End Time	Empty	CRC
0x50	0x05 0x02 0x00	4 byte	4 byte	0x00 0x00 0x00 0x00 0x00 0x00	2 byte

Request with Parameter (20 byte)

Opcode	Status	Backfill Status	Identifier	Start Time	End Time	Unknown	CRC
0x51	1 byte	1 byte	1 byte	4 byte	4 byte	6 byte	2 byte

Response after all Backfill Bata arrived (20 byte)

Table 4.13: Backfill Transmission at Backfill Channel

Request Counter	Unknown	Glucose Data
2 byte	2 byte	n × (Time: 4 byte; Glucose: 2 byte; Type: 1 byte; Trend: 1 byte)

Complete Backfill Data Response Message

Sequence Number	Itentifier	Data Payload
1	1 byte	18 byte (from the complete backfill message)
2	1 byte	18 byte (from the complete backfill message)
3	1 byte	18 byte (from the complete backfill message)
...		
n	1 byte	Rest of the backfill message. Could be less that 18 byte.

Packets which are received over the backfill channel and contains the response data (20 byte).

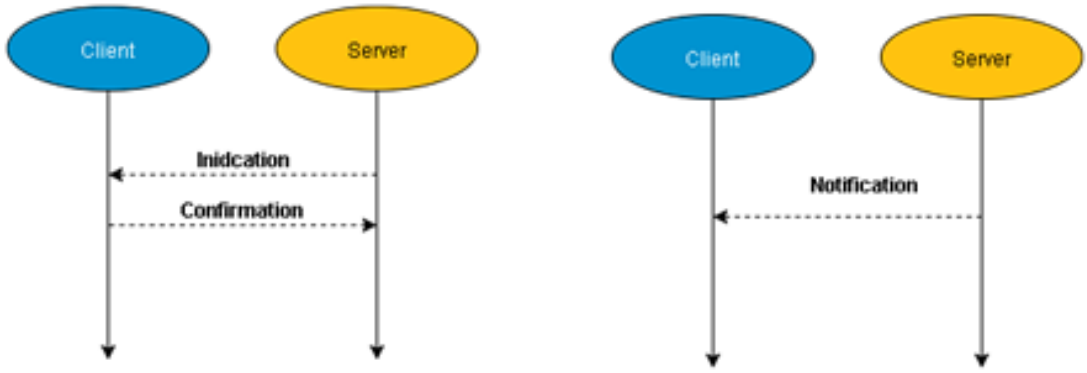
5 Implementation

This chapter is about the mechanics and software concepts used in the ESP code. For this project the chosen programming language is C/C++. But also other programming languages would have been possible for the ESP (see 3.1). The full source code can be found in this Github repository [16] and can help understanding the software concepts. Also the source code is fully documented using inline comments on the right side in the source code.

5.1 Bluetooth Low Energy

The basics of Bluetooth Low Energy are explained in chapter 3.2. The Dexcom transmitter communication uses the write and notify/indicate properties of characteristics. The write property is used to send data from the client to the transmitter. These are the protocol request messages. The client writes the data into the specific characteristic. The transmitter can read it from there. The other way, from the transmitter to the clients, is by using the indicate mechanism. This means that an interrupt is called client-side when new data arrive from the transmitter. To work with these interrupts in a sequential program some flow control mechanism are needed. The best way is to provide a method which sends data to the transmitter and return the response when it arrives. This method loops until the response arrived through the callback function which is called by the interrupt. The request and response are transmitted over the same characteristic.
The difference between notification and indication is that with indication the client sends back a confirmation to the BLE server (transmitter) that he has received the data.

Figure 5.1: Indication & Notification



Graphic by Iliana Berenice Tejeda Hernandez [11]

Using transmitters with newer firmware, there are some problems with indication. On the G6 transmitter with the serial number starting with 81.... the indication is working fine. But on the G6 Plus models with serial number prefix 8G some unexpected behavior is happening. The newer transmitter has some changed characteristic properties as shown in this table 4.2. Some read and write properties are removed from the characteristics. The indication is still supported but does not trigger an interrupt on the client-side. The solution was to let the client register for indication and notification on the characteristics. Even when the notification feature is not advertised by the transmitter. An explanation could be that the official Dexcom reader device does not distinguish between indication and notification when registering for the characteristic. It listens to both and only sends a confirmation when it was an indication package. The same is happening in the xDrip-js code. There is only the notification callback used which is technically wrong because the transmitter is advertising that he only supports indication for the control and authentication channel. But this is still working because the JavaScript libraries are smart enough to handle the correct one. On the ESP this is done manually. The code for the solution to register for indication and notification at the same time with the same callback is shown below. A method to register for both is not supported by the Arduino library.

```
/**
 * Register for indication AND notification.
 */
// Byte values for the notification / indication.
const uint8_t none[]         = {0x0, 0x0};
const uint8_t notification[] = {0x1, 0x0};
const uint8_t indication[]   = {0x2, 0x0};
const uint8_t both[]         = {0x3, 0x0};

// First register for indication
pBLERemoteCharacteristic>registerForNotify(_callbackMethod, false);
// True to wait for acknowledge, set to both, manually set the bytes because
// there is no such funktion to set both.
pBLERemoteCharacteristic->getDescriptor(BLEUUID((uint16_t)0x2902))
                        ->writeValue((uint8_t *)both, 2, true);
```

5.2 Power Optimization

As shown in table 3.2 putting the ESP into deep sleep mode is a great way to save energy. The problem, however, is to decide when and for how long to go into the deep sleep mode. Also which sleep mode saves the most energy without shutting down essential hardware components. The first problem is quite straightforward when looking at the transmitter's wakeup cycle. About every 5 minutes, within an accuracy of a few seconds, the transmitter wakes up. So for the first time, the ESP has to wait until it finds the transmitter. Then after the connection, the ESP and transmitter go both into deep sleep at the same time. From then on the ESP's and transmitter's time cycles are synchronized. The ESP wakes up after 4.5 minutes and starts waiting for the transmitter. These additional 30 seconds earlier will compensate for the transmitter's wakeup deviation. This gives the insurance of not missing the transmitter. Then after the connection, both will go into deep sleep at the same time again. This will prevents being asynchronous over time.

The ESP uses the "deep sleep" mode from the table above 3.1. This mode will shut down the CPU & main memory, peripherals, and also all modules for Bluetooth, BLE, Wi-Fi, and Radio. The ULB co-processor

is also not needed because the ESP has in deep sleep mode no work to do. The wakeup is triggered by a hardware clock. Important to know, is that the main memory is shut down. This means that all variables and the program states are lost. To save values beyond the deep sleep the RTC memory is used. This is also the only ESP component that is not turned off. Below is the source code for entering the deep sleep. Even though the slow RTC memory is used for saving data, the fast RTC memory is kept on. This is due to a hardware bug in Rev 0 ESP-32 modules. The flash memory is not starting fast enough which will trigger a "**TG0WDT_SYS_RESET**" event. This event will force the ESP to restart and thus deletes all data from the RTC memory. To prevent this, a delay in the wakeup period is needed [29]. An easier solution is to keep the faster RTC memory on. Even when not used. One of my development boards had this problem. So I could figure out this easier workaround when using this board: *ESP32-ST*. No problem occurs with turned off the faster RTM memory when using the "*ESP32_DevKitc_V4*" board with the newer *ESP32-WROOM-32D* chip.

In deep sleep the power consumption goes down to about 10μA. When active, the consumption can rise up to 130mA with transmitting and receiving data over BLE [6]. This will give a worst case power consumption of about $130mA * \frac{1}{10} + 0,01mA * \frac{9}{10} = 13mA$. The normal case is lower because most of the active time the ESP is just scanning passively for devices. This consumes much less power than transmitting data over BLE, which is only a few seconds when the connection is established.

```
/**
 * This function sets the ESP to deep sleep mode
 * with a power consumption of ~10uA.
 */
void sleepHibernation()
{
  esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_SLOW_MEM, ESP_PD_OPTION_ON);
  // Keep the RTC slow memory on to save the last glucose values
  // and last error variable.
  esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_FAST_MEM, ESP_PD_OPTION_ON);
  // Could be turned off in later esp chips but is on to not trigger this bug
  //https://forum.mongoose-os.com/discussion/1628/tg0wdt-sys-reset
  //-immediately-after-waking-from-deep-sleep
  esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
  esp_sleep_enable_timer_wakeup(270 * 1000000); // Sleep for 4.5 minutes.
  . . .
  esp_deep_sleep_start(); // Go to sleep.
}
```

Additionally to save energy a passive BLE scan is performed to find other BLE devices. This means that the ESP is not actively sending packages to find devices. It only listens to find advertisement packages. In my environment, this works fast enough to find the transmitter before he goes back into deep sleep. When this is not the case in the future, an active scan can be helpful. This consume much more energy but will potentially find devices faster.

5.3 Memory Management

The ESP has the capability of saving the last glucose values received from the transmitter. The amount is variable and can be set in the source code. The values are stored in special RTC memory which survives the deep sleep mode and does not get reset when the processor wakes up. The memory itself is volatile. This means that the stored information is lost when the ESP shuts down or the power is turned off. After each wakeup, all stored glucose values are displayed through the serial connection when a serial monitor is connected.

On every connection cycle, the newly received glucose level is stored. This happens by pushing all older values back in the array to make space for the current value at position 0. The last (oldest) value gets dropped from the memory. The array stores the glucose levels from newest to oldest in the resolution of 5 minutes.

When the ESP missed some values in the history array a backfill is executed. Some missing values can arise when the transmitter is out of range or some problems in the Bluetooth connection occurred. A Backfill is also done right at the beginning when the ESP connects for the first time after being powered on. This means only one time after the ESP started from being powered off, not from waking up out of the deep sleep. This behavior has the benefit of filling the history array right from the start when the data is available on the transmitter. Otherwise, it could take up to several hours until enough connection cycles are done. When the backfill data is requested from the transmitter, it is in reverse order. This means the first message contains the oldest requested value. The backfill values from the response messages 4.13 get all inserted at position 0 in the array. Before each value is inserted, all other values in the array are shifted to a higher position. This results in a sorted array from the newest to the oldest value. All previous data from the array are overwritten by this action. This is not of concern because the backfill data is complete and consistent with also in 5-minute resolution.

5.4 Connection Problem Handling

One point is to detect problems and errors during the connection. It is important to know when it was not possible to request the current glucose value or other data. Because then the history array which saves the last glucose levels can get corrupted by missing or false data. To detect missing or false values a simple plausibility check is used which scans the history array for invalid glucose values and if one invalid value occurs a backfill is performed automatically.

To detect and prevent connection problems a “set and check” mechanism using a state variable is implemented. At each wakeup of the transmitter, this variable is set. After all successful Bluetooth communication, the state of the variable is flipped. Before the ESP goes back into the deep sleep, the state of the variable is checked. When not all planed protocol requests can be performed, the state of the variable does not change. Also when the transmitter timed out or other Bluetooth problems occur the state does not change. Before the ESP goes back into the deep sleep, it saves if a problem occurred in this session. It is saved in RTC memory which is also available after the next wakeup. So the ESP always knows if the last connection was successful or not. When a problem happened in the last Bluetooth connection, it forces a BLE re-bonding with the transmitter to avoid bonding problems (see 4.4). Also, a backfill is performed to pull the missing glucose data which were not received in the last connection from the transmitter’s memory.

6 Working Environment and Evaluation

For developing an ESP32 application and decoding the Dexcom communication protocol special hardware and software are used on the ESP and transmitter, respectively. For developing the Dexcom communication xDrip-js [9] and xDrip-plus [20] are used to crosscheck the protocol. xDrip-js is capable to log and show the transmitted data which helps to understand the request and response messages.

6.1 ESP32 and Arduino IDE

For developing and testing an ESP32 development board is used. The benefit of these boards is that they have all additional hardware already build in. Most important a USB-to-serial module for flashing the compiled binary files into the ESP's flash memory. This serial module can also be used to transmit data between the computer and the ESP. Additionally, the power supply through the USB connection and some buttons and LEDs for direct user interaction are built in. A picture of the ESP developing board is shown in graphic 3.1.

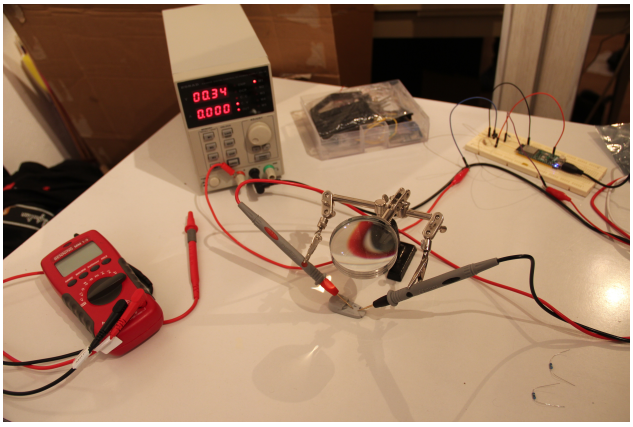
Compiling C source code for the ESP can be done by the official Espressif IoT Development Framework [5] or using the Arduino IDE [1]. The latter uses the Arduino platform libraries which have a lot of functionality also developed for the ESP. Before developing ESP32 software in Arduino IDE the Espressif board extension must be installed by adding this URL (https://dl.espressif.com/dl/package_esp32_index.json) to "Additional Boards Manager URLs" under Preferences. Then using the Board Manager under *Tools -> Board -> Board Manager* the "esp32" extension can be installed. In this thesis, the currently newest version 1.0.4 is used. This gives the capability to select one of many development boards on the market with the ESP32 chip and set board-specific settings. The Arduino IDE also comes with a serial monitor/plotter for communicating with the ESP. This is used for showing debug information and also received values from the transmitter. For stack tracing in the Arduino IDE the *EspExceptionDecoder* [8] plugin is recommended.

6.2 Sensor Transmitter Emulation

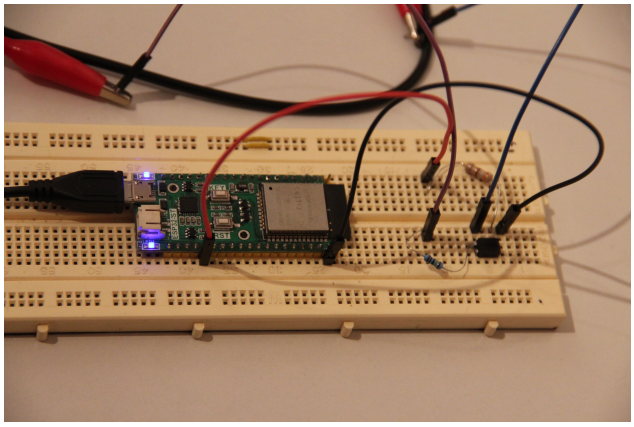
Normally all features of the G6 and G5 transmitter only work when placed into a glucose sensor which is inserted into the patient's body. Then a so-called session can be started and glucose levels can be received. When this is not possible a workaround is necessary to emulate a glucose sensor and allow the transmitter to measure predefines values. The sensor itself acts as a low direct current (DC) source in the range from 100nA to 2000nA. This was found out by the nightscout community for the older G4 models [10].
The presumption has been confirmed by testing that this is also suitable for the G6 model. With a laboratory power supply, the current is limited to 0.001A and the voltage is set to a level below 1V. An additional 1MΩ will bring the current level in the correct range to be valid for the transmitter. The transmitter does also

run some plausibility checks to detect faulty or expired sensors. To pass this check the current value has to fluctuate a little in the valid range so that every five minutes the raw value that is measured is different. To change the value a second ESP is interposed which regulates the current using pulse wide modulation. To physically detach the ESP from the transmitter circuit an optocoupler is inserted between them.

Figure 6.1: Laboratory Setup for Sensor Emulation

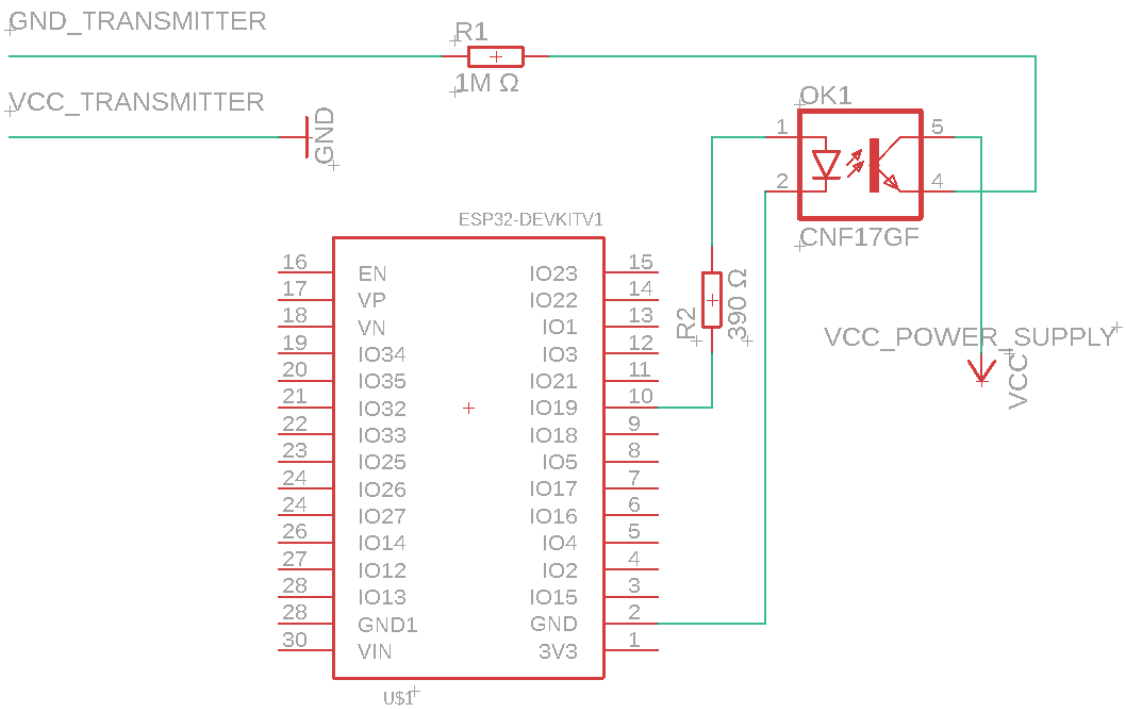


Graphic by Max Kaiser



Graphic by Max Kaiser

Figure 6.2: Eagle Circuit Diagram Sensor Emulation



Graphic by Max Kaiser

6.3 Software

For the development of the Dexcom communication protocol, it is useful to have xDrip-js [9] working. xDrip-js can output hex values of the transmitted data from the request and response messages which where send over the different Bluetooth Low Energy characteristics. This helps in understanding the protocol code to implement it on the ESP. At first, a PC with the capability of running JavaScript code and a Bluetooth Low Energy module are needed. When no built-in module is available, a compatible Bluetooth dongle can be used. However not all Bluetooth chips are supported by the JavaScript libraries. For example, the Broadcom BCM20702 chip is compatible. Also, some bonding problems occurred in xDrip-js when using Windows 8.1 or different Ubuntu versions, resulting in a rejected bonding request. A working environment can be set up using Kali Linux version 2020.1 and this software packages:

```
# Update Linux:
sudo apt-get update
sudo apt-get upgrade

# Install Bluetooth Support:
sudo apt-get install bluetooth bluez bluez-tools rfkill
# this will list all found Bluetooth modules and shows if they are blocked
sudo rfkill list
sudo rfkill unblock bluetooth
sudo service bluetooth start
sudo apt-get install blueman

# Install Node.js:
sudo apt install nodejs
sudo apt install npm

# Install libudev-dev:
# otherwise npm install failes on xDrip-js
sudo apt-get install libudev-dev

# Install xDrip-js:
# clone the Github repository from https://github.com/xdrip-js/xdrip-js
cd xdrip-js
sudo npm install
npm test
```

For running xDrip-js this commands are used:

```
# Start Bluetooth:
sudo service bluetooth start

# UNPAIR Transmitter befor every new connetion:
bt-device -r Dexcom{XX}
# OR
bt-device -r {MAC ADDRESS}
```

```
# Start xDrip-js:
# Important to run xDrip-js as root.
sudo DEBUG=smp,transmitter,bluetooth-manager node example {ID}
```

On Kali Linux, the bonding process with the transmitter is working. When the transmitter was found a message box (from the operating system, not xDrip-js) appears and asks to accept the bonding. After that, the xDrip-js code can continue and communicate with the transmitter. Important to know is that the bonding cannot be reused. This will result in problems. Therefore the transmitter must be unpaired before every run of xDrip-js.

6.4 Testing

To ensure that the communication protocol on the ESP is working correctly, all functionality (if possible) is tested on two different G6 transmitter models:

Table 6.1: Transmitter Versions

Model	Firmware	Serial Number Prefix	State
G6 Plus	> 1.6.5.25	8Gxxxx	Started, Session, OK
G6 (SW11163)	1.6.5.25 /1.6.13.139	81xxxx	Stopped, Low Battery

The first and newer transmitter model is used in the real environment. This means that the sensor which is used by the transmitter is inserted into the human body. Moreover, the transmitter has a running session and is used simultaneously for therapy reasons. All implemented functionality and backfill are tested successfully. The second transmitter is the one used during development. This one is running using an emulated sensor. Therefore it is not possible to start a sensor session. When no session is started the current glucose value does not get calculated by the transmitter and could not be requested. More specifically: the glucose value is zero. But it is possible to receive the raw sensor values. Furthermore no backfill data is available. Besides this, the other functions like battery, time, etc are tested successfully. Additionally, a long time test was done with this transmitter to check if connection problems occur. In a timespan of 10h and 22 minutes all 102 connections where successful and none failed. The surprising thing about this is, that this gives an average frequency of one connection every ~6 minutes. This means that the real wake-up cycle is about every 6 minutes, not every 5 minutes as described by Dexcom here [33]. It could be that this deviation is caused by the low battery state of the transmitter. For future tests, more transmitter models and different firmware versions should be used. This is important because the Bluetooth Low Energy characteristics and the transmitter behavior can change. As seen in table 4.2 the properties have already slightly changed for the newer transmitter. Also some functionality like reading the raw sensor values where removed.

7 Conclusion

To conclude, in this thesis an algorithm was developed to allow a communication between the continuous glucose monitoring (CGM) system and an ESP32 system on chip. The basic knowledge about diabetes mellitus and characteristics like high blood glucose are shown. Also, the importance of insulin injections which depends on blood glucose concentration, to prevent long term damage is explained. These glucose level measurements are mandatory to adjust the daily insulin injections. The CGM system consists of a glucose sensor, transmitter, and a reader device. The communication between the reader device (which also could be a smartphone) is over the Bluetooth Low Energy protocol. Up to two devices can simultaneously be connected with the transmitter. The Dexcom protocol is based on a request-response message structure. There are different messages for authentication, getting glucose level data, maintenance, and glucose backfill.
The ESP32 used in this thesis is a small microcontroller with built-in modules for Wi-Fi, Bluetooth and Bluetooth Low Energy, and more. The ESP connects every 5 minutes to the transmitter of the CGM system. This allows to read the glucose data and save it on the ESP.
Also, some energy-saving methods were developed to allow the ESP to run energy efficiently. This allows running the ESP multiple weeks on a battery.
For getting continuous data, also connection problem handling routines where implemented. An algorithm keeps the glucose data saved on the ESP consistent even when a connection cycle failed. Therefore, the backfill mechanism of the transmitter is used. It allows receiving older glucose values form the last hours.
To ensure the correctness of the protocol and algorithms on the ESP, some testing using different transmitter models where done. Additionally, one transmitter was used in the real environment, which means that the transmitter was inserted into a sensor that was applied onto the skin to measure the glucose concentration.
This was done to ensure that the code on the ESP will work with most G6 transmitter versions. The ESP can then be used for future research projects which will need to read the live glucose measurements portable and energy efficient.

8 Future Work

For future investigations, one goal could be to implement additional functions to run the ESP completely independently. This means without the need of any Dexcom devices for maintenance. To achieve this some mandatory functions for calibration and starting/stopping sessions are needed. This allows the user to replace the sensor every 10 days with the use of the microcontroller. Then the ESP can replace all functionality of the original Dexcom reader. Additionally, some graphical user interface for displaying the current glucose level or showing a graph of the data for the last hours is interesting for future work. This interface could also be used to show warnings or errors which may occur; i.e. low battery or when the transmitter is out of range. Further, a function to warn the user when dangerous glucose levels are reached is a very useful extension. Even when the ESP is only used to collect data and not used as a complete replacement for a smartphone or a reader device.

Bibliography

[1] *Arduino IDE*. Arduino. URL: https://www.arduino.cc/en/Main/Software (visited on 05/24/2020).

[2] Emma Black. *Github xDrip*. URL: https://github.com/StephenBlackWasAlreadyTaken/xDrip (visited on 05/24/2020).

[3] *Das neue Dexcom G6*. DexCom, Inc. URL: https://www.dexcom.com/de-DE/de-dexcom-g6-cgm-system (visited on 05/16/2020).

[4] *Dexcom G6 Android Application*. DexCom, Inc. URL: https://play.google.com/store/apps/details?id=com.dexcom.g6 (visited on 05/24/2020).

[5] *ESP-IDF*. Espressif Systems. URL: https://github.com/espressif/esp-idf (visited on 05/24/2020).

[6] *ESP32 Deep Sleep with Arduino IDE and Wake Up Sources*. URL: https://randomnerdtutorials.com/esp32-deep-sleep-arduino-ide-wake-up-sources/ (visited on 05/12/2020).

[7] *ESP32 Series Datasheet*. Espressif Systems. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (visited on 05/12/2020).

[8] Me No Dev @ Github. *Exception Stack Trace Decoder for ESP8266 and ESP32*. URL: https://github.com/me-no-dev/EspExceptionDecoder (visited on 05/28/2020).

[9] *Github xDrip-js*. URL: https://github.com/xdrip-js/xdrip-js (visited on 05/24/2020).

[10] *Gitter.im nightscout/decoding-dexcom*. URL: https://gitter.im/nightscout/decoding-dexcom?at=5af8d239d245fe2eb7a9efff (visited on 04/16/2020).

[11] Iliana Berenice Tejeda Hernandez. *Indication and Notification*. URL: https://community.nxp.com/docs/DOC-328525 (visited on 05/17/2020).

[12] *How long can I wear the sensor?* DexCom, Inc. URL: https://www.dexcom.com/faq/how-long-can-i-wear-sensor?category=2641 (visited on 05/24/2020).

[13] *How to Use a Glucose Meter*. HealthHub. URL: https://www.healthhub.sg/live-healthy/1431/how-to-use-a-glucose-meter (visited on 05/27/2020).

[14] *Insulin Pens*. URL: https://www.drugs.com/cg/insulin-pens.html (visited on 05/27/2020).

[15] Scott Jung. *A Brief Look at Dexcom G6 Continuous Glucose Monitoring System*. May 3, 2018. URL: https://www.medgadget.com/2018/05/a-brief-look-at-dexcom-g6-continuous-glucose-monitoring-system.html (visited on 05/16/2020).

[16] Max Kaiser. *Dexcom-ESP32-Reader Github Repository*. URL: https://github.com/TheEpicBigBoss/Dexcom-ESP32-Reader.

[17] Andreas Kläui. *Die neue Generation Dexcom G6*. Sept. 23, 2018. URL: https://klaeuiblog.ch/2018/09/die-neue-generation-dexcom-g6/ (visited on 05/11/2020).

[18] Philip Koopman. *32-Bit Cyclic Redundancy Codes for Internet Applications*. 2002. URL: https://users.ece.cmu.edu/~koopman/networks/dsn02/dsn02_koopman.pdf (visited on 05/20/2020).

[19] Laura Nao. *BLE Pairing and Bonding*. URL: https://www.kynetics.com/docs/2018/BLE_Pairing_and_bonding/ (visited on 05/15/2020).

[20] *Nightscout xDrip+*. Nightscout Foundation. URL: https://jamorham.github.io/#xdrip-plus (visited on 05/24/2020).

[21] Pankaj. *Java UUID Generator – Java GUID*. URL: https://www.journaldev.com/17182/java-uuid-generator-java-guid (visited on 05/08/2020).

[22] *picture glucose meter*. Ken Research. URL: https://www.kenresearch.com/blog/wp-content/uploads/2017/07/Europe-Cervical-Cancer-Test-Market-1024x883.jpg (visited on 05/27/2020).

[23] *picture insulin pump*. FAULHABER. URL: https://www.faulhaber.com/fileadmin/_processed_/c/b/csm_wo_Insulinpump_d69a922314.jpg (visited on 05/27/2020).

[24] *Programming Languages for ESP32 and ESP8266*. Electronic DIYs. URL: https://www.electronicdiys.com/2018/11/programming-languages-for-esp32-esp8266.html (visited on 05/12/2020).

[25] Martin Schiftan. *Artikelserie: Closed Loop; 2. Was braucht man (Hardware/Software/Faktoren)?* Nov. 17, 2017. URL: https://zehn.be/2017/11/17/artikelserie-closed-loop-2-was-braucht-man-hardware-software-faktoren/ (visited on 05/18/2020).

[26] Günther Schmelzeisen-Redeker et al. *Time Delay of CGM Sensors*. Aug. 4, 2015. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4667340/ (visited on 05/22/2020).

[27] *Screening for Type 2 Diabetes*. AMERICAN DIABETES ASSOCIATION. Apr. 1989. URL: http://journal.diabetes.org/clinicaldiabetes/v18n22000/pg69.htm (visited on 05/27/2020).

[28] *t:slim X2 Insulin Pump*. TANDEM. URL: https://www.tandemdiabetes.com/tslimx2 (visited on 05/24/2020).

[29] *TG0WDT-SYS-RESET immediately after waking from deep sleep*. Mongoose OS. URL: https://forum.mongoose-os.com/discussion/1628/tg0wdt-sys-reset-immediately-after-waking-from-deep-sleep (visited on 05/24/2020).

[30] *Typ-2-Diabetes: Grundlagen*. Diabetesinformationsdienst München. URL: https://www.diabetesinformationsdienst-muenchen.de/erkrankungsformen/typ-2-diabetes/grundlagen/index.html (visited on 03/11/2020).

[31] *Typ-2-Diabetes: Verbreitung*. Diabetesinformationsdienst München. URL: https://www.diabetesinformationsdienst-muenchen.de/erkrankungsformen/typ-2-diabetes/verbreitung/index.html (visited on 03/11/2020).

[32] *USB Bluetooth Low Energy on i.MX6 Android platform*. iWave Systems. URL: https://www.iwavesystems.com/usb-bluetooth-low-energy (visited on 05/28/2020).

[33] *What is the Dexcom G6?* DexCom, Inc. URL: https://www.dexcom.com/faq/what-dexcom-g6 (visited on 05/24/2020).

[34] *xDrip-js Sensor Session Status Codes*. URL: https://github.com/xdrip-js/xdrip-js/blob/master/README.md#session-status-valid-values (visited on 05/24/2020).