

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Prof. Dr. Ulrich Schäfer

Projektarbeit Mobile & Ubiquitous Computing SoSe 2024

Gruppe 42: Johannes Grosch & Daniel Hartl

Studiengang Industrie-4.0-Informatik

9. Juli 2024

Inhaltsverzeichnis

1	Einleitung	2
2	Projektplanung und Vorgehen	2
2.1	Planungsphase	2
2.2	Vorgehen	2
2.3	Kommunikation zwischen Komponenten	2
3	Implementierung	3
3.1	Änderungen an der MAX30100 Bibliothek	3
3.2	ESP32 Code	4
3.3	Node-RED Datenfluss	5
3.4	Android Applikation	7
3.4.1	Frontend	7
3.4.2	MQTT Schnittstelle	7
3.5	L ^A T _E X Template Erweiterungen	9
4	Probleme und Diskussion	9
5	Zusammenfassung und Ausblick	10
	Literatur	11

1 Einleitung

Das Ziel des Projekts ist es, mittels des optische Herzfrequenzmess- und Pulsoximetriesensormoduls MAX30100[3] ein Puls- und Pulsoximetriemessgerät zu realisieren, welches über eine MQTT Broker Anbindung fähig ist Daten an eine Android Applikation zu übertragen.

2 Projektplanung und Vorgehen

2.1 Planungsphase

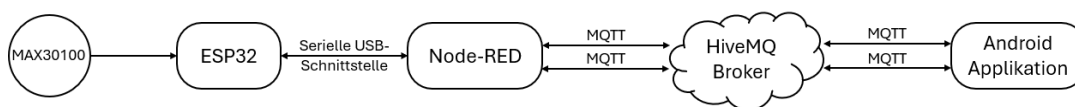


Abbildung 1: Datenflussdiagramm innerhalb des Projekts

Die Planung ging nach dem Prinzip „Divide et impera“ Teile und herrsche von Stat-ten. Da in dem System so viele separate Komponenten, die bis auf die ein- und aus-gehenden Werte autonom voneinander sind. Diesen, bzw. dem Format, in dem diese zwischen den Komponenten übertragen werden ist das Unterkapitel 2.3 gewidmet.

Die Arbeitsaufteilung wurde derartig vorgenommen, dass Herr Grosch die Android Applikation, was wir als das umfangreichste Subsystem realisiert haben und Herr Hartl die zwei kleineren Subsysteme und die \LaTeX Dokumentation erstellt, da er bereits versiert mit \LaTeX ist.

2.2 Vorgehen

Zur Realisierung haben wir uns an einen inkrementellen Ansatz gehalten. So haben wir zuerst die Datenquelle, die angepasste MAX30100 Bibliothek entwickelt und uns bildhaft in Abbildung 1 nach rechts vorgearbeitet. So konnte das, in der Entwicklung befindliche System jederzeit mit realen Daten getestet werden und es bestand kein Bedarf an Testdatensätzen o.ä.

Dieser Entwicklungsansatz wird ebenfalls in der Struktur des Kapitels 3 wiedergegeben.

2.3 Kommunikation zwischen Komponenten

Die Serielle Kommunikation zwischen dem ESP_32 und dem Laptop ist mit 10Hz auf 115200 Baud realisiert. Um unnötigen Overhead zu vermeiden findet sie in folgender, minimalistischer Synthax statt:

<Pulsschlag>;<Sauerstoffsättigung>

Diese Telegramme werden dann in Node-RED[8] eine Sekunde bzw. zehn Nachrichten lang gesammelt, nach Parametern aufgedröselt, je ein Durchschnitt gebildet um den Jitter zu verringern und die Graphen zu glätten und dann an den HiveMQ Broker[7] zu senden, wo es unter dem Topic **afkdcdjkcns/sensor/data/raw/sensor** für die Sensordaten der App zugänglich gemacht werden.

Andersherum published die Android Applikation unter dem Topic **afkdcdjkcns/sensor/ctrl/raw/sensor** die zuletzt ausgewählte View, nach der die Node-RED Logik die zu publishenden Daten auswählt.

3 Implementierung

Der ESP_32[2] ist wie in folgender Abbildung gezeigt mit dem MAX30100[3] verbunden. Diese Konfiguration ist von der Website Electronicwings.com übernommen.

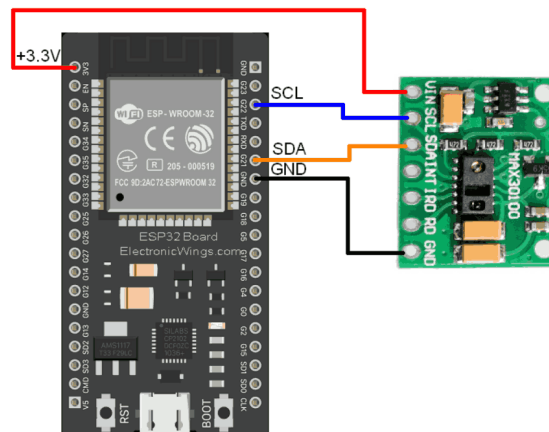


Abbildung 2: Verkabelung des ESP_32 mit dem MAX30100[6]

- 3,3V \longleftrightarrow VIN
- G22 \longleftrightarrow SCL
- G21 \longleftrightarrow SDA
- GND \longleftrightarrow GND

3.1 Änderungen an der MAX30100 Bibliothek

„[Der] MAX30100 ist ein optisches Herzfrequenzmess- und Pulsoximetriesensormodul. Es integriert zwei LEDs (IR und Rot), einen Photodetektor (Rot), eine optimierte Optik und eine rauscharme analoge Signalverarbeitung zur Erkennung von Pulsoximetrie- und Herzfrequenzsignalen. Das Signal wird von einer rauscharmen analogen Signalverarbeitungseinheit verarbeitet und über die I2C-Schnittstelle an die Ziel-MCU weitergeleitet.“[3]

Die Basis zum Betreiben des MAX30100 bildet die gleichnamige Bibliothek[4] von Connor Huffine. Diese wurde zur besseren Erfüllung der speziellen Anforderungen wie folgt angepasst.

Zur effizienteren Bedienung des Puls Oximeters wurde das Enumerate „BoardType“ hinzugefügt, das für den Wert 0 die Konstante ESP_32 und für den Wert 1 LOLIN_32 zuordnet. Hierbei wird auf den LOLIN_32[1] speziell Rücksicht genommen, da dieser als kleinere Version eines ESP_32 weniger Pins aufweist, die dadurch über andere IDs angesteuert werden müssen.

In der Klasse PulseOximeter selbst wurde die Signatur der Methode begin() um den Parameter „board“ modifiziert:

```
bool begin(BoardType board, PulseOximeterDebuggingMode debuggingMode_);
```

Listing 1: Signatur der begin() Methode

Hierbei sind beide optional und board hat als Standardparameter ESP_32, was den bisherigen Ablauf anstößt. Sollte die Methode mit dem Parameterwert 1 oder LOLIN_32 aufgerufen werden, so werden die Konstanten für den SDA Pin auf 0 und die für den SCL Pin auf 4 gesetzt.

3.2 ESP32 Code

Der ESP32 wurde so programmiert, dass er nach CALLBACK_INTERVAL Millisekunden über die Serielle Schnittstelle mit 115200 Baud, wie in 2.3 beschrieben, die Herzfrequenz und den Sauerstoffgehalt des Blutes zur weiteren Verarbeitung an den Laptop überträgt, wo diese dann weiterverarbeitet werden können.

```
#include <Wire.h>
#include "MAX30100_PulseOximeter.h"

// reporting interval 100ms
#define CALLBACK_INTERVAL 100

// driver entity
PulseOximeter oximeter;

// timestep of the last output
int lastOutputTimestamp;

void setup() {
    // init the serial interface for 115200 Baud
    Serial.begin(115200);
    while(!Serial);

    // try to init the oximeter and count the failed tries and delay 500ms.
    Serial.println("Begin initializing the oximeter");

    int initFailCounter = 0;
```

```

while (!oximeter.begin(ESP_32)) {
    Serial.print("Init failed ");
    Serial.print(++initFailCounter);
    Serial.println(" times!");
    delay(500);
}

Serial.println("Oximeter initialized");
}

void loop() {
    // get values from oximeter.
    oximeter.update();

    // print heartrate and o2 ratio
    Serial.print(oximeter.getHeartRate());
    Serial.print(";");
    Serial.println(oximeter.getSpO2());

    delay(CALLBACK_INTERVAL);
}

```

Listing 2: Quellcode des ESP32-Programms

3.3 Node-RED Datenfluss

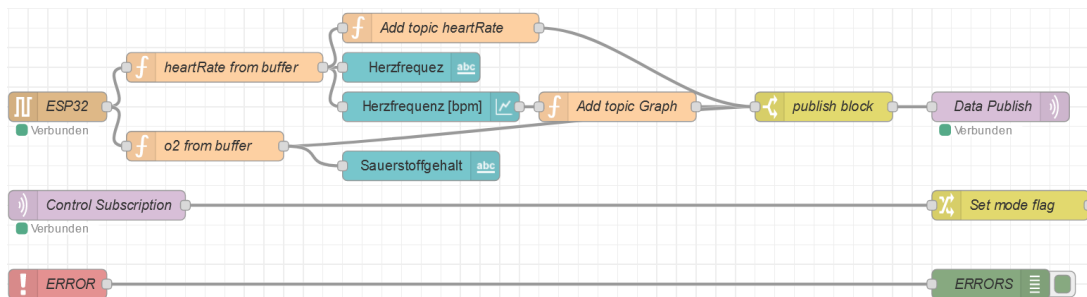


Abbildung 3: Node-RED[8] Datenfluss Darstellung

Serielle Schnittstelle

Die serielle Schnittstelle ist so konfiguriert, dass sie mit einer Baudrate 115200 für 1000ms die Übertragung buffert und diese dann weitergibt. Dies geschieht im Knoten „ESP32“.

Verarbeitung des seriellen Inputs

In den Knoten „heartRate from buffer“ und „o2 from buffer“ wird der serielle Mitschnitt jeweils aufgebrochen, die gesuchten Werte separiert und der Durchschnitt ge-

bildet, um den Graph etwas zu glätten.

```
var o2 = 0.0;
var i = 0;

// split the serial protocol of 1 second into seperate messages.
var words = msg.payload.split("\r\n");

// seperate each message in single values for o2 and heartrate.
words.forEach(function (item) {
    if(item.includes(";"))
    {
        var values = item.split(";");
        if(!isNaN(values[1]))
        {
            // add all valid o2 values and count them.
            o2 = parseFloat(o2) + parseFloat(values[1]);
            i = i + 1;
        }
    }
});

// return the average o2 value in payload.
return [ {payload: o2 / i, topic: 2}];
```

Listing 3: Quellcode der Methode „o2 from buffer“ zum Lesen des Sauerstoffgehalts aus dem Buffer

Die Ausgabe dieser Methoden wird dann im Node-RED internen Dashboard als Absolutwerte und in Form eines Graphen für die Herzfrequenz visualisiert (Siehe Abbildung 4). Das Graphen-Objekt wird im Datenfluss weitergegeben, um in der Android App weiterverwendet werden zu können.

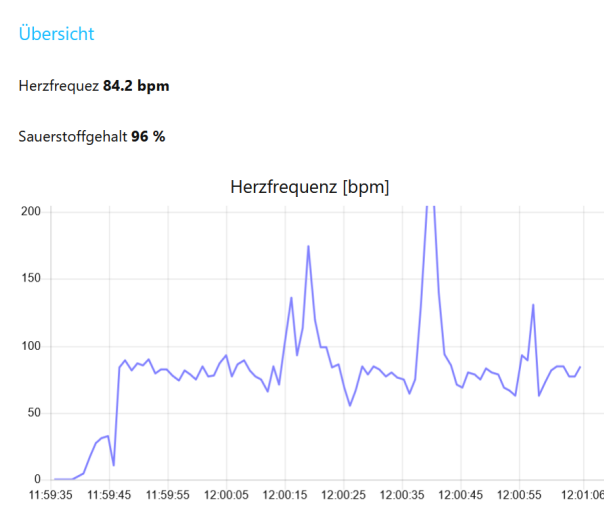


Abbildung 4: Node-RED Dashboard

MQTT Publishing

Im Anschluss daran werden die sogenannten Messages mit einem Topic, was dem in der Android App definierten Enumerate „selectedView“ entspricht.

```
enum SelectedView
{
    heartRate,
    graph,
    o2
}
```

Listing 4: Quellcode des Enumerates „selectedView“ zur Auswahl der zu übertragenden Werte

In dem switch-Knoten „publishBlock“ werden dann nur die Messages, deren Topic Attribut der ID der ausgewählten Ansicht entsprechen, an den MQTT Publish Knoten weitergegeben. Dieser published dann die Werte an das Topic „afkdcdjcnks/sensor/data/raw/sensor“.

Moduseinstellung

Der Knoten „Control Subscription“ abonniert das Topic „afkdcdjcnks/sensor/ctrl/raw/sensor“. Auf dieses published die Android Applikation die ID der ausgewählten Ansicht (Siehe Listing 4). Mit einem Abtastintervall von einer Sekunde wird der aktuelle Wert überprüft und im flow Context „mode“ gespeichert, auf welches dann der „publish block Knoten“ zugreift.

3.4 Android Applikation

Die Android Applikation unterstützt Android Oreo oder höher. lässt sich im wesentlichen in drei Teile unterteilen. Das Frontend, das MQTT Modul und das Konfigurations-Lademodul.

3.4.1 Frontend

3.4.2 MQTT Schnittstelle

Die MQTT Schnittstelle besteht aus zwei Bereichen. Zuerst dem eigentlichen Client, der in Kapitel 3.4.2.2 näher beschrieben wird und dann dem Interface, welches zur Weitergabe der Nachrichten in der Applikation dient und in Kapitel 3.4.2.1 erläutert ist.

3.4.2.1 ISubscribe

Das MQTT Modul wurde mit der Paho Bibliothek der Eclipse Foundation[5] realisiert. Es ist eine statische Klasse, die nach Aufbau einer Verbindung zu einem MQTT-Broker bis zu drei Abonnenten vom Typ „ISubscribe“.

```
public interface ISubscribe {
    void onMessageReceived(String message);
    void unsubscribe();
}
```

Listing 5: Quellcode des Interfaces ISubscribe

Von diesen wird dann, sobald ein neuer Wert am Broker eingetroffen ist die Methode „onMessageReceived“ aufgerufen, mit dem Wert als Parameter. So werden die Daten automatisch in die Frontend Fragments geladen, sofern diese korrekt initialisiert sind.

Die Methode „unsubscribe“ ist dazu da, explizit das Abonnement zu beenden und den Platz für ein anderes Fragment freizumachen. Sie wird in der Klasse **MainActivity** beim ändern des Tabs automatisch aufgerufen.

3.4.2.2 MqttModule

Die eigentliche MQTT Modulklassse hat folgende Methoden:

Name	Parameter	Beschreibung
init	String, String, String	Füllt die Topic Attribute und den genutzten Klientnamen.
isInit		Prüft ob alle in init gesetzten Parameter nicht leer sind.
connect	String, int	Baut eine TCP-Verbindung zum Broker auf mit URL und Port als Parameter.
connect	String, int, String, String, String	Ruft init und connect auf.
disconnect		Baut die TCP-Verbindung zum Broker ab.
publishCtrlMessage	SelectedView (Listing 4)	Published den Parameter als Integer an den Broker.
subscribeData		Abonniert das angegebene DataTopic und setzt die Callbackmethode, dass in allen registrierten ISubscribe Objekten die onMessageReceived Methode aufgerufen wird.
addSubscription	ISubscribe	Fügt Parameter zu Abonnenten hinzu, sofern ein Platz frei ist.
removeSubscription	ISubscribe	Entfernt Parameter aus Abonnenten, sofern er einer ist.

Abbildung 5: Methoden der Klasse MqttModule

Sie wird genutzt, indem beim Start der Anwendung der Client mit dem Broker verbunden wird und das Topic abonniert. Wenn sich das angezeigte Fragment ändert, wird dieses als Abonnent hinzugefügt, während das letzte entfernt wird. Zudem wird bei einem Wechsel ein Publish mit der neuen, ausgewählten View vollzogen.

3.5 L^AT_EX Template Erweiterungen

Erweitern der Inhaltsverzeichnis-tiefe

Zur übersichtlicheren Aufschlüsselung wurde ein weiteres Level an Überschriften benötigt. Hierzu erweiterten wir unsere Dokumentenkonfiguration um folgenden Absatz, den wir von Stackexchange[9] übernahmen.

```
\setcounter{secnumdepth}{5}
\setcounter{tocdepth}{5}
\newcommand\simpleparagraph[1]{%
    \stepcounter{paragraph}\paragraph*{\theparagraph\quad{}}{#1}}
```

Listing 6: L^AT_EX Quellcode zum Anzeigen einer vierten Inhaltsverzeichnisebene

Sprachsupport für JavaScript im listings-Paket

Um den Quellcode von Listing 3 ansprechend darstellen zu können wurde das von Prof. Dr. Schäfer gestellte L^AT_EX Template erweitert um Quellcodesupport für JavaScript[10]. Der dafür benötigte Quellcode wurde von Stackexchange übernommen und geringfügig verändert, um dem bestehenden Konzept zu entsprechen.

```
\lstdefinlanguage{JS}{
    keywords={typeof, new, true, false, catch, function, return, null, catch, switch,
        var, if, in, while, do, else, case, break},
    keywordstyle=\color{blue}\bfseries,
    ndkeywords={class, export, boolean, throw, implements, import, this},
    ndkeywordstyle=\color{blue}\bfseries,
    identifierstyle=\color{black},
    sensitive=false,
    comment=[l]{//},
    morecomment=[s]{/*}{*/},
    commentstyle=\color{battleshipgrey}\ttfamily,
    stringstyle=\color{cardinal}\ttfamily,
    morestring=[b]',
    morestring=[b]"
}
```

Listing 7: L^AT_EX Quellcode für JavaScript Support im Paket listings

4 Probleme und Diskussion

Die größten Probleme bei der Implementation traten beim Umsetzen der MQTT Verbindungen auf. Der Zugriff auf ein persönliches HiveMQ Cluster von Node-RED ist selbst nach mehrstündiger Recherche nicht gelungen, weshalb wir probeweise auf den HiveMQ Public MQTT Broker gewechselt sind. Dies funktionierte auf Anhieb ohne Probleme.

5 Zusammenfassung und Ausblick

Das Projekt ist als Konzept durchaus brauchbar für medizinische Anwendung. Man müsste nur die Alle Daten kontinuierlich zur Verfügung stellen. Sollte ein zuverlässiger Puls und Sauerstoffgehalt Sensor eingebunden werden, der ESP_32 eigenständig mit dem Broker kommunizieren und mit einem Token oder einer ID ausgestattet werden, könnte man das System kommerziell einsetzen.

So wäre es möglich, dass man über ein affiliate System Arztpraxen die Möglichkeit gibt, ihnen zugeordnete Geräte auszugeben, die der Patient nutzt. Durch diese Zuordnung könnte der Arzt vorbereitend auf eine Sprechstunde oder zur Ferndiagnose die gemessenen Werte online einsehen.

Diese Idee ist zugegeben Datenschutztechnisch nicht auf Realisierbarkeit geprüft, könnte jedoch die Behandlung von chronischen Krankheiten, deren Symptomatik mit Blutwerten zusammenhängt modernisieren und da diese oft bei älteren, weniger mobilen Menschen auftreten vereinfachen.

Abbildungsverzeichnis

1	Datenflussdiagramm innerhalb des Projekts	2
2	Verkabelung des ESP_32 mit dem MAX30100[6]	3
3	Node-RED[8] Datenfluss Darstellung	5
4	Node-RED Dashboard	6
5	Methoden der Klasse MqttModule	8

Quellcodeverzeichnis

1	Signatur der begin() Methode	4
2	Quellcode des ESP32-Programms	4
3	Quellcode der Methode „o2 from buffer“ zum Lesen des Sauerstoffgehalts aus dem Buffer	6
4	Quellcode des Enumerates „selectedView“ zur Auswahl der zu übertragenden Werte	7
5	Quellcode des Interfaces ISubscribe	8
6	L ^A T _E X Quellcode zum Anzeigen einer vierten Inhaltsverzeichnisebene .	9
7	L ^A T _E X Quellcode für JavaScript Support im Paket listings	9

Literatur

- [1] AZ-DELIVERY GMBH: *ESP32 Lolin32 microcontroller (Datenblatt zum ESP32 Lolin32)*. <https://megma.ma/wp-content/uploads/2021/08/Wemos-ESP32-Lolin32-Board-BOOK-ENGLISH.pdf>, Abruf: 06.07.2024
- [2] AZ-DELIVERY GMBH: *ESP32 NodeMCU Module (Datenblatt zum ESP32 Lolin32)*. https://cdn.shopify.com/s/files/1/1509/1638/files/ESP_-_32_NodeMCU_Developmentboard_Datenblatt_AZ-Delivery_Vertriebs_GmbH_10f68f6c-a9bb-49c6-a825-07979441739f.pdf, Abruf: 06.07.2024
- [3] AZ-DELIVERY GMBH: *MAX30100 Modul (Beschreibung des Breakout-Moduls)*. <https://moodle.oth-aw.de/pluginfile.php/304161/course/section/43767/MAX30100%20de.pdf>, Abruf: 06.07.2024. – PDF im Moodlekurs als "Kurzbeschreibung" verfügbar
- [4] CONNOR HUFFINE: *MAX30100 Bibliothek (Dokumentation zur max30100 Bibliothek)*. <https://www.arduino.cc/reference/en/libraries/max30100/>, Abruf: 06.07.2024
- [5] ECLIPSE FOUNDATION: *Paho Bibiliothek - Github Repository*. <https://github.com/eclipse/paho.mqtt.android>, Abruf: 09.07.2024
- [6] ELECTRONICWINGS DIGITAL TECHNOLOGIES PVT. LTD: *MAX30100 Pulse Oximeter Interfacing with ESP32*. <https://www.electronicwings.com/esp32/max30100-pulse-oximeter-interfacing-with-esp32>, Abruf: 06.07.2024

- [7] HivEMQ GmbH: *HiveMQ Public MQTT Broker*. <https://www.hivemq.com/mqtt/public-mqtt-broker/>, Abruf: 06.07.2024
- [8] OPENJS FOUNDATION: *Node-RED – Low-code programming for event-driven applications*. <https://nodered.org>, Abruf: 19.05.2024
- [9] USER COMMONHARE FROM STACKEXCHANGE: *Paragraph numbers and table of contents*. <https://tex.stackexchange.com/questions/204170/paragraph-numbers-and-table-of-contents>, Abruf: 09.07.2024
- [10] USER LUCAS AND JMLARSON FROM STACKEXCHANGE: *Create language JavaScript for usage in lstlisting*. <https://tex.stackexchange.com/questions/89574/language-option-supported-in-listings>, Abruf: 06.07.2024