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

7. 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 8
3.5 Komplexere LaTeX Erweiterungskommandos 8
3.5.1 Sprachsupport für JavaScript im listings-Paket 8

4 Probleme und Diskussion **8**

5 Zusammenfassung und Ausblick **8**

Literatur **10**

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

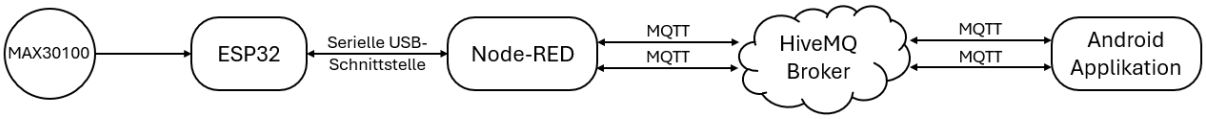

Abbildung 1: Datenflussdiagramm innerhalb des Projekts

Die Planung ging nach dem Prinzip „Divide et impera“ Teile und herrsche von Statten. Da in dem System so viele separate Komponenten, die bis auf die ein- und ausgehenden 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[7] 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[6] zu senden, wo es unter dem Topic **afkdcdjkcnks/sensor/data/raw/sensor** für die Sensordaten der App zugänglich gemacht werden.

Andersherum published die Android Applikation unter dem Topic **afkdcdjkcnks/-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.

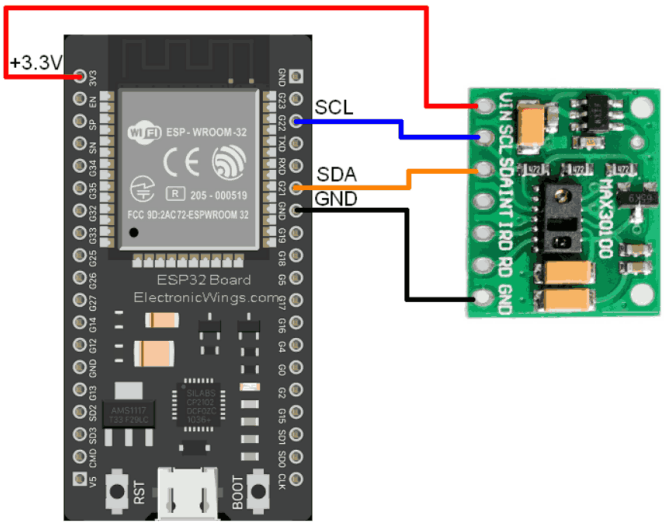

Abbildung 2: Verkabelung des ESP_32 mit dem MAX30100[5]

- 3,3V ⟷ VIN
- G22 ⟷ SCL
- G21 ⟷ SDA
- GND ⟷ 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;

// timestap of the last output
int lastOutputTimestamp;

void setup() {
       // intit 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

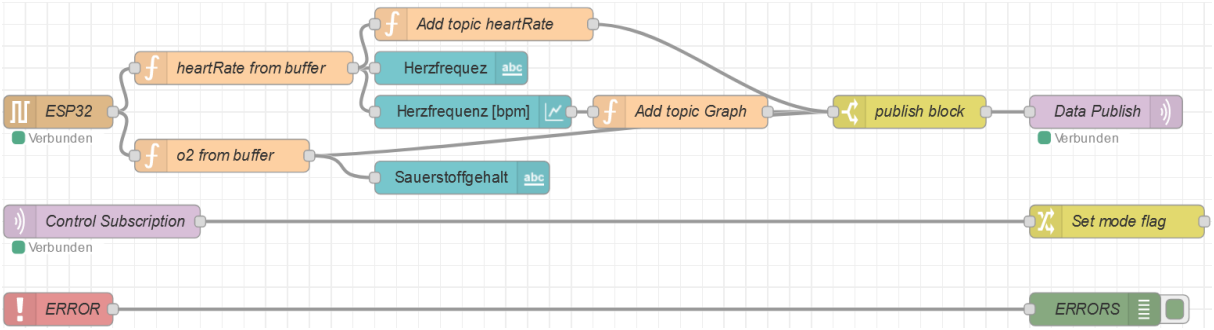

Abbildung 3: Node-RED[7] Datenfluss Darstellung

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 gebildet, 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.

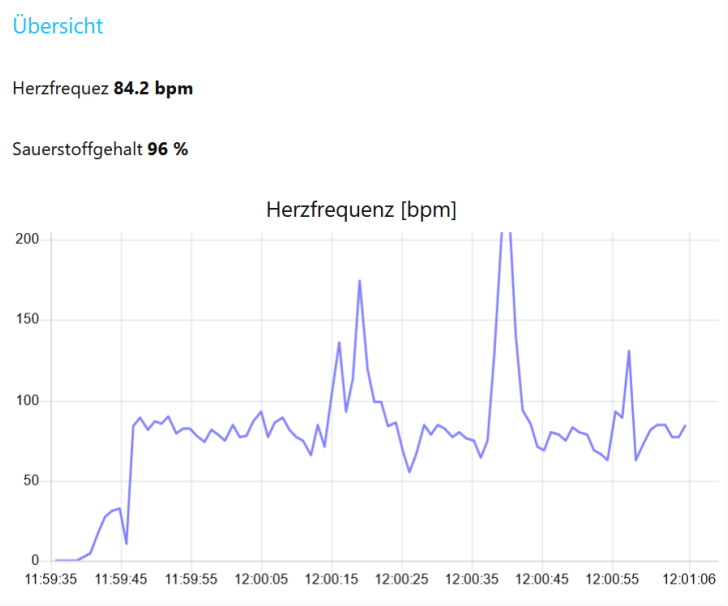

Abbildung 4: Node-RED Dashboard

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 = 0,
        graph = 1,
        o2 = 2
}
```

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 „afkdcdjkcnks/sensor/data/raw/sensor“.

Moduseinstellung

Der Knoten „Control Subscription“ abonniert das Topic „afkdcdjkcnks/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

...

3.5 Komplexere LaTeX Erweiterungskommandos

3.5.1 Sprachsupport für JavaScript im listings-Paket

Um den Quellcode von Listing 3 ansprechend darstellen zu können wurde das von Prof. Dr. Schäfer gestellte LaTeX Template erweitert um Quellcodesupport für JavaScript[8]. Der dafür benötigte Quellcode wurde von Stackexchange übernommen und geringfügig verändert, um dem bestehenden Konzept zu entsprechen.

```
\lstdefinelanguage{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 5: LaTeX 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 zuverlässigerer Puls und Sauserstoffgehalt Sensor eingebunden werden, der ESP_32 eigenständig mit dem Broker kommunizieren und mit einem Token oder einer ID ausgestattet werden, könnte man das System komerziell 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[5] 3
3 Node-RED[7] Datenfluss Darstellung 5
4 Node-RED Dashboard 7

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 LaTeX Quellcode für JavaScript Support im Paket listings 8

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] 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

[6] HiveMQ GmbH: *HiveMQ Public MQTT Broker*. https://www.hivemq.com/mqtt/public-mqtt-broker/, Abruf: 06.07.2024

[7] OpenJS Foundation: *Node-RED – Low-code programming for event-driven applications*. https://nodered.org, Abruf: 19.05.2024

[8] User commonhare from stackexchange: *Create language Javascript for usage in lstlisting*. https://tex.stackexchange.com/questions/89574/language-option-supported-in-listings, Abruf: 06.07.2024