ätter152020Bl

Frankfurt University of Applied Science, Informatik
Zigbee, WS20/21

Drahtlose Sensornetzwerke - Luftqualität in Innenräumen

February 23, 2021

Sebastian Monok, Barbar Ahmad ..... Applikation
Aylin Erdem, Omar Osman, Pascal Schieferstein, Emin Gaplan ..... Mikrokontroller
Dipl. Inf. Markus Krauße ..... Professor

Contents

1 Einführung (Barbar Ahmad)

Das Thema dieser Arbeit lautet: Luftqualität in Innenräumen. Dieses Thema ist besonders zutreffend auf die derzeitige Pandemie, da wir mittlerweile wissen, dass das Virus nicht nur durch Tröpchen und Schmierinfektionen, sondern sich auch an Aerosole anhängt und somit länger in der Luft verweilen kann. Da dieses Problem insbesondere in Dichtbesiedelten Innenräumen auftritt (Vorlesungssaal, Klassenraum), passt dieses Thema perfekt in die Universität. Mit Hilfe von Sensoren und eines Dashboards zur Kontrolle, kann schnell auf Veränderungen der Luftgüte reagiert werden und somit potentielle Ausbrüche einer Krankheit entgegengewirtkt werden.

2 Aufgabenverteilung

Anforderung	Bereich	Mitglied
Zeichnen eines Raumplans in der Applikation	Applikation	Barbar Ahmad
Platzieren von Sensormodulen in den Raumplan	Applikation	Barbar Ahmad
Benachrichtigung beim Erreichen einer bestimmten Luftqualität	Applikation	Barbar Ahmad
Benachrichtigung beim Erreichen einer bestimmten Temperatur	Applikation	Barbar Ahmad
Melden eines Ausfalls des Sensors	Applikation	Barbar Ahmad
Anzeigen der täglichen Statistik der Luftqualität	Applikation	Barbar Ahmad
Anzeigen der wöchentlichen Statistik der Luftqualität	Applikation	Sebastian Monok
Anzeigen der Gesamtstatistik der Luftqualität	Applikation	Sebastian Monok
Anzeigen der Anzahl der Übertretungen des Grenzwertes	Applikation	Barbar Ahmad
Manuelles Setzten der Grenzwerte ermöglichen	Applikation	Barbar Ahmad
Microservice Architektur verwenden	Applikation	Barbar Ahmad
Übertragung der Werte in eine MYSQL Datenbank	Applikation	Barbar Ahmad
Microservice zum auslesen der ZigBee Module und schreiben in die Datenbank	Applikation	Sebastian Monok
Websocket Verbindung zum Webserver für live Updates der Statistiken	Applikation	Barbar Ahmad
Übertragung der Sensordaten über JSON API	Applikation	Sebastian Monok, Pascal Schieferstein
Webapplikation für Multi-Betriebsystem Support aufbauen	Applikation	Barbar Ahmad
Containisierung der einzelnen Microservices	Applikation	Barbar Ahmad
Einbindung des CCS811 Sensors	Mikrokontroller	Aylin Erdem, Pascal Schieferstein
Einbindung des SHT21 Sensors	Mikrokontroller	Omar Osman
Einbindung des MG881 Sensors	Mikrokontroller	Emin Gaplan
Überwachung der Batteriespannung der Funkmodule	Mikrokontroller	Pascal Schieferstein

3 Herangehensweise und Lösung der Teilaufgaben

3.1 Barbar Ahmad

3.1.1 Zeichnen eines Raumplans in der Applikation

Das Zeichnen eines Raumplanes hat sich am Anfang als schwierig herausgestellt. Der erste Gedanke war es, eine Open Source Library zu nutzen, die mir das Zeichnen auf einem Canvas erleichtert. Ich habe daraufhin auf https://github.com/cvdlab/react-planner zurückgegriffen. Dies ist ein Open Source Github Projekt, dass auf das Zeichnen von Raumplänen spezialisiert ist. Dies stellte sich aber auf langer sicht als ein Fehler heraus, da dieses Tool auf einer alten React Version basiert und nicht kompatibel mit der der aktuellen React Version ist. Dies hat dazu geführt, dass ich praktisch den gesamten Source-Code dieser Library Updaten musste. Im Endeffekt wäre es schlauer gewesen, diese Funktionalität selbst zu schreiben, da dies Wahrscheinlich schneller und sauberer gewesen wäre.

3.1.2 Platzieren von Sensormodulen in den Raumplan

Für das platziernen der Sensordaten in den Raumplan haben wir uns entschieden, alle Module, die noch nicht in einem Raum gesetzt worden sind in einem Dropdown Menu anzuzeigen. Somit kann mann zwischen den einzelnen Sensoren in einem Raum wechseln und sich die genaueren Statistiken dazu anzeigen lassen.

3.1.3 Benachrichtigung beim Erreichen einer bestimmten Luftqualität

Für die Benachrichtigung beim erreichnen einer bestimmten Luftqualität nehmen wir den derzeitigen Durchschnitt der Luftqualität, und falls der aktuelle Wert um mehr als 10 Prozent um den Durchschnitt abweicht (geringer ist), bekommen alle Benutzer eine E-Mail mit einer Warnung.

3.1.4 Benachrichtigung beim Erreichen einer bestimmten Temperatur

Für die Benachrichtigung beim erreichnen einer bestimmten Luftqualität nehmen wir den derzeitigen Durchschnitt der Luftqualität, und falls der aktuelle Wert um mehr als 10 Prozent um den Durchschnitt abweicht, bekommen alle Benutzer eine E-Mail mit einer Warnung.

3.1.5 Melden eines Ausfalls des Sensors

Falls ein Sensor für mehr als 3 Stunden keine Daten mehr liefert, wird in der Datenbank ein Eintrag hinterlegt, dass der Sensor überprüft werden sollte. Dies wird auch in der Navigation der Applikation angezeigt.

3.1.6 Anzeige der täglichen Statistik der Luftqualität

Wenn man auf einen Sensor klickt, kommt man auf eine Kalender übersicht, auf der man sich für verschiedene Zeiträume die Statistik dieses Sensors ansehen kann. Dort werden die Min, Max und Durchschnittswerte des Sensors über einen gewählten Zeitraum angezeigt. Falls kein Enddatum gewählt wird, wird eine persistente Websocket Verbindung erstellt und ein Livefeed des Sensors gezeigt.

3.1.7 Anzeigen der Anzahl der Übertretungen des Grenzwertes

Sobald einer der Grenzwerte übertreten wird, wird der Sensor und der Grenzwert in die Datenbank gespeichert. Diese Inhalte werden auf der Übersicht der Räume angezeigt und es wird angezeigt, wie oft dieser Raum einen der Grenzwerte übertreten hat.

3.1.8 Microservice Architektur verwenden

3.1.9 Übertragen der Werte in eine MYSQL Datenbank

Wir haben uns für eine MYSQL Datenbank entschieden, da wir Erfahrungen in der Universität mit MYSQL gesammelt haben. Unser Datenbankschema ist wie folgt aufgebaut:

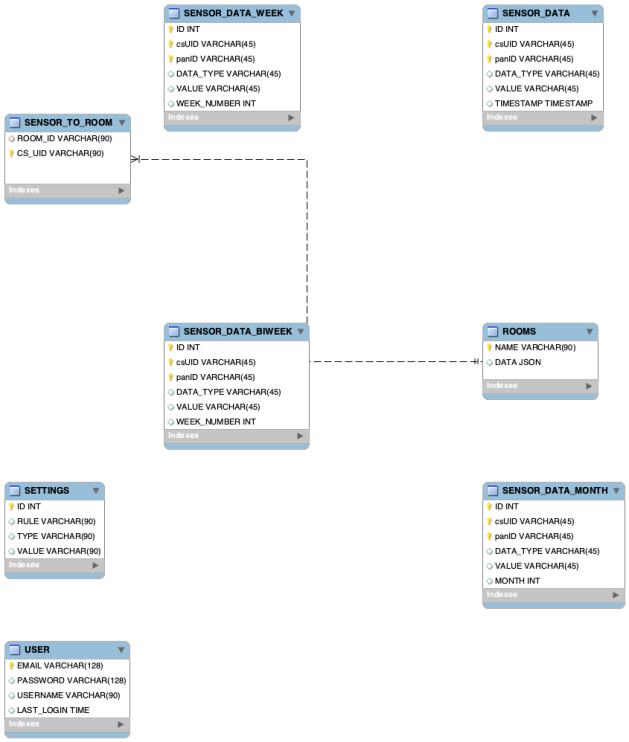

Figure 1: Database Schema

USER Die Usertabelle enthält alle Registrierten Benutzer der Applikation. Dort wird die E-Mail, das Passwort (als sha256 Encrypt), der Benutzername und der zuletzt eingeloggte Zeitpunkt angezeigt.

SETTINGS In der Settingstabelle werden verschiedene Einstellungen im Bezug auf die Applikation getätigt. Es ersetzt praktisch eine Konfigurationsdatei. Dort können Grenzwerte für Sensoren eingetragen werden.

ROOMS In der Roomstabelle werden die Einzelnen Räume angelegt. Der Raumname ist als Primary-Key genutzt und die DATA wird als JSON Representation gespeichert. Diese wird dann wieder in der Applikation geladen.

SENSOR_TO_ROOM In der Roomstabelle werden die Einzelnen Räume angelegt. Der Raumname ist als Primary-Key genutzt und die DATA wird als JSON Representation gespeichert. Diese wird dann wieder in der Applikation geladen.

SENSOR_DATA In dieser Tabelle werden die einzelenen Werte der Sensoren gespeichert. Wir nutzen eine ID, die csUID und die panID als Shared Primaray Key. Desweiteeren werden der Datentyp (Temperatur, Volatility etc.), der eigentliche Wert und der Zeitstempel gespeichert.

SENSOR_DATA_BIWEEK/MONTH/WEEK Diese Tabellen sind genauso aufgebaut wie die Sensor Data Tabelle, und wird für die Statistiken verwendet.

3.1.10 Webapplikation für Multi Betriebsystem Support aufbauen / Containerisierung

Für die Anzeige der Webaplikation haben wir uns für ein React - Gatsby Frontend entschienden bzw. für eine Progressive Web Applikation. Ein Webserver läuft und stellt über das Netzwerk den Zugriff auf die Applikation zur Verfügung. Diese ist über den Browser erreichbar. Der Vorteil bei dieser Lösung ist, dass die Applikation Betriebssystemunabhängig laufen kann. Auf dem HOST System wird nur Python benötigt. Die Applikation ist außerdem Containerisiert. Über Docker wird die Einrichtung somit noch leichter.

3.1.11 Websocket Verbindung zum Webserver für Live Updates der Statistiken

Für die Kommunikation zwischen den Frondend und dem Backend, haben wir uns für Websockets entschieden, da die persistente Verbindung eine Live Reload der Daten ermöglicht.

3.2 Sebastian Monok

3.2.1 Anzeigen der wöchentlichen, Bi-Wöchentlichen und Gesamtstatistik der Luftqualität

In zeitlichen Abständen wird jeweils der Median der Sensor-Daten berechnet und in eine seperate Tabelle für die tägliche, wöchentliche, zweiwöchentliche, monatliche oder jährliche Auswertung gespeichert.

3.2.2 Übertragung der Sensordaten über JSON API Und Mikroservice zum auslesen der ZigBee Module und schreiben in die Datenbank

Um die Daten der ZigBees an dem Computer auszulesen wurde die Python Library serial verwendet, welche über das SerialPort Daten auslesen kann. Der ZigBee Koordinator schickt die Daten an die UART bridge in einem Json Format:
{"csUID":"%s","panID":"%s","DATA_TYPE":"%s","VALUE":"%s"} Die Daten werden anschließend mit Hilfe der Python Library mysql in einer mySQL Datenbank gespeichert.

Datum und Uhrzeit werden vom Python Script hinzugefügt. Bei ersten Tests wurde festgestellt, dass der Serial Port, über den die Daten der ZigBees empfangen werden, nicht bei allen Teammitgliedern gleich ist. Der Port wird dynamisch gesetzt indem eine Schleife, 1 – 10, durchlaufen wird und dabei wird überprüft ob über den Port die entsprechenden Daten empfangen werden. Da ich nicht alle Sensoren bei mir habe musste für weitere tests ein Datenstrom über den Serial Port simuliert werden. Dies wurde ebenfalls mit Hilfe der Python Library serial umgesetzt.

3.3 Aylin Erdem, Pascal Schieferstein

3.3.1 Einbindung des CCS811 Sensors

Um den CCS881 Sensor erfolgreich zum Laufen zu bringen und im Anschluss nutzen zu können, haben wir folgendes Equipment und Standards zum Programmieren verwendet. Abgesehen von dem Equipment rund um den Sensor, wurde sehr oft zusammen am Sensor gearbeitet und auch dafür einige Standards zur Kommunikation in Anspruch genommen.

- CCS811 - Sensor
- ATmega256RFR2
- Zwei 1,5 Volt Batterien (Funkmodul Anschluss)
- 2x USB auf Mini USB-Kabel
- USART Bridge
- Atmel-ICE Entwicklungstool
- Atmel Studio
- Putty

Der CCS811 Sensor ist ein Gas Sensor, welches die Werte VOC (flüchtige organischen Verbindungen) und CO2 (eine chemische Verbindung aus Kohlenstoff und Sauerstoff) ausmessen kann, wobei die VOC Messung durch den Menschen veränderbar ist. Der CO2 Wert wird zwischen 400ppm und 8192ppm gelesen und der VOC Wert zwischen 0ppb und 1187ppb. Außerdem unterstützt der Sensor mehrere Modes zum Messen, die optimiert wurden, um den Stromverbrauch während einer aktiven Messung zu verringern und die Batterielebensdauer zu verlängern.

Man kann den CCS811 Sensor so verstehen, dass ein Mikrokontroller innerhalb des Sensors existiert, welches über den i2c Bus programmiert werden muss. Dieses Verhalten könnte man ebenso mit dem Server - Client Prinzip vergleichen, die hin und her kommunizieren. Der Sensor hat 5 verschiedene Modes wie folgt:

Modes of Operation

The CCS811 has 5 modes of operation as follows

- Mode 0: Idle, low current mode
- Mode 1: Constant power mode, IAQ measurement every second
- Mode 2: Pulse heating mode IAQ measurement every 10 seconds
- Mode 3: Low power pulse heating mode IAQ measurement every 60 seconds
- Mode 4: Constant power mode, sensor measurement every 250ms

Figure 2: Measure modes of operation

Um den gewollten Modus einzustellen, müssen wir das Register 0x01 über den i2c Bus auswählen, unseren Mode schreiben und anschließend in das Ergebnis Register wechseln. Um den Mode 1 zu nutzen, müssen wir das Register mit 0x10 beschreiben, für Mode 2 mit 0x20. Nach der Messung wird in den Intervall geschrieben in dem er eingestellt wurde und wird dort je nach dem Mode, gelesen.

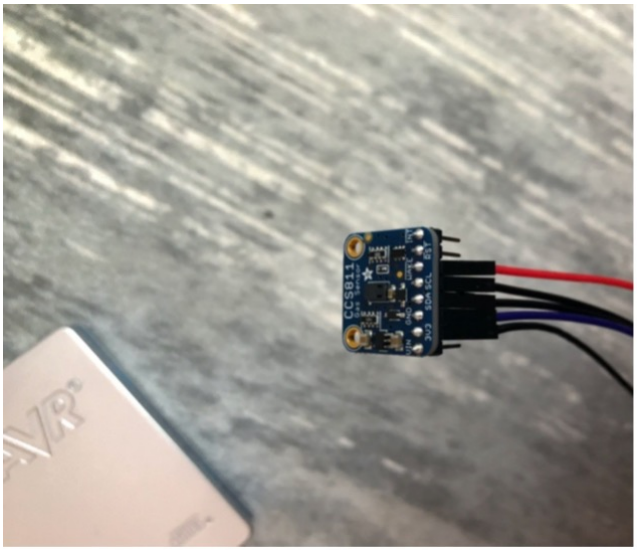

Figure 3: Measure modes of operation

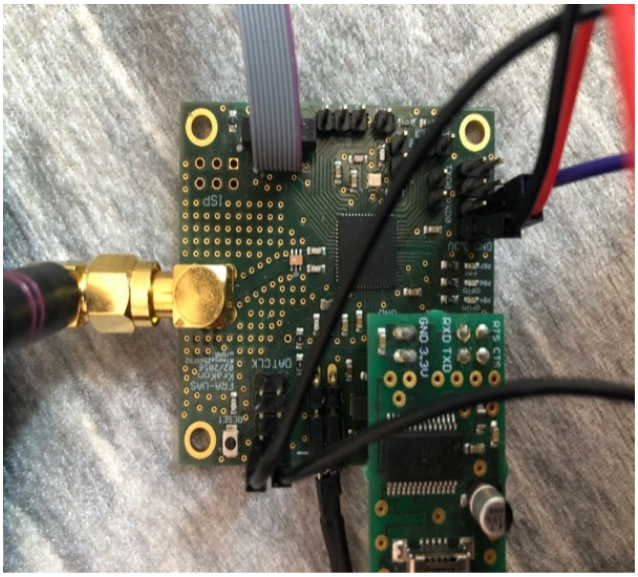

Figure 4: Measure modes of operation

Beim Anschluss ist ganz wichtig, „Wake“ und „GND“ bloß nicht zu vergessen, denn ohne diesen Anschluss kann man Tagelang überlegen, wo der Fehler liegt und findet ihn nicht, weil der Sensor dann nicht funktioniert.

Die ZigBee Module, sind in der Abgabe nicht dazu konzipiert Koordinator oder Router zu sein. Es ist noch etwas alter Code vorhanden der einst diese Option während der Testphase unterstützt hat, aber dieser ist in der Abgabe unvollständig und wird zu Fehlern oder gar den Softlock der ZigBee Module führen. Als Koordinator dient der Basic-Coordinator, der nur Koordinator sein kann.

Wichtig, der Sensor wird sich nach dem Start nur einmal bei der Applikation melden und darauffolgend für 20 Minuten keine Daten ausgeben. Der Sensor braucht nämlich 20 Minuten zum Aufwärmen, das heißt jene Datenwerte sind bis dahin nicht nutzbar und werden daher zwar intern ausgelesen, aber nicht übermittelt.

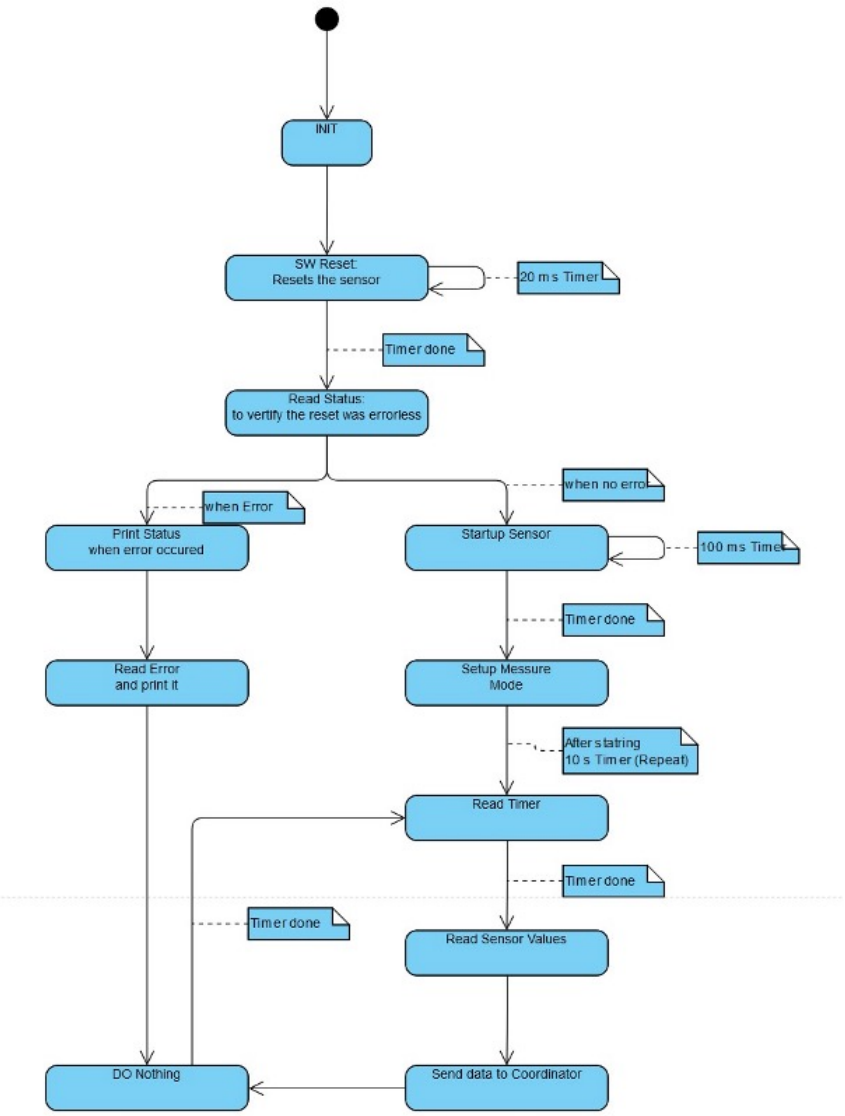

Figure 5: Zustandsautomat

Zustandsautomat Der Microcontroller geht beim Programmstart in den Initialisierungs Zustand INIT. In diesem Zustand wird der BitCloud Stack mit allen notwendigen Daten für das Auslesen des Sensors und das Übertragen der Daten an den Koordinator initialisiert. Nach der Initialisierung wechselt der Sensor bedingungslos in den Zustand SW Reset. In diesem Zustand setzen wir den Sensor aus Sicherheitsgründen zurück, um die optimale Operation des Sensors sicher zu stellen. Das Zurücksetzen nimmt 20 Millisekunden in Anspruch, dessen Einhaltung wir durch einen Timer sicherstellen. Erst nach Ablauf des 20 ms Timers wechseln wir in den Zustand Read Status. In diesem Zustand lesen wir das Status-Register des CCS-881 Sensors aus, um sicherzustellen, dass das Zurücksetzen erfolgreich war. Wenn

dieser Vorgang nicht erfolgreich war, also wenn das Status-Register ungleich 0x10 ist, dann geben wir den Status Code aus, und Springen in den Zustand Read Error. Hier im Read Error, wird das Error-Register gelesen und der Error Code über die USART Brücke (falls) angeschlossen ausgegeben. Der Error wird aber nicht über das Netzwerk gesendet.

Wenn wie zu erwarten der Wert des Status-Registers gleich 0x10 ist, dann wird der Sensor gestartet. An diesem Punkt haben wir, da im Datenblatt keine genaue Zeit definiert ist wann der Sensor startet, die Wartezeit von 100 ms aus der Offiziellen Arduino Library von CCS-881 Sensor übernommen. Nach dem die 100 ms abgelaufen sind, wird der Sensor mit den richtigen Parametern eingerichtet, damit er alle 10 Sekunden, Sensordaten ausließt. Hier Starten wir auch einen weiteren 20 Minuten Timer (nicht dargestellt), der das Versenden von Sensordaten für 20 Minuten blockiert. Wie schon beschrieben braucht der Sensor nämlich 20 Minuten zum Aufwärmen. Wenn die 20 Minuten abgelaufen sind, startet der Sensor, um die Sensordaten alle 10 Sekunden zu übermitteln

Herangehensweise und Lösungen Zu Beginn und auch während dem gesamten Projekt waren die Datenblätter, die Übungsblätter aus dem Modul „Informatik Projekt “ und das Buch „Drahtlose Zigbee- Netzwerke“1, Größtenteils die Bestandteile der Vorgehensweise. Dem Datenblatt, den Übungsblättern und auch dem Buch konnte man sehr viele nützliche Informationen entnehmen und nach mehrfachem lesen auch selbst anwenden. Da zuvor noch nicht mit dem BitCould Stack gearbeitet wurde, haben wir damit angefangen die gestellte Aufgabe mit dem Arduino zu testen. Dazu benötigt man einen Arduino, Steckplatine, ein USB-Kabel, mehrere Kabel, um die Verbindung zwischen dem Board und dem Sensor herzustellen und ganz wichtig, den CCS811 – Sensor.

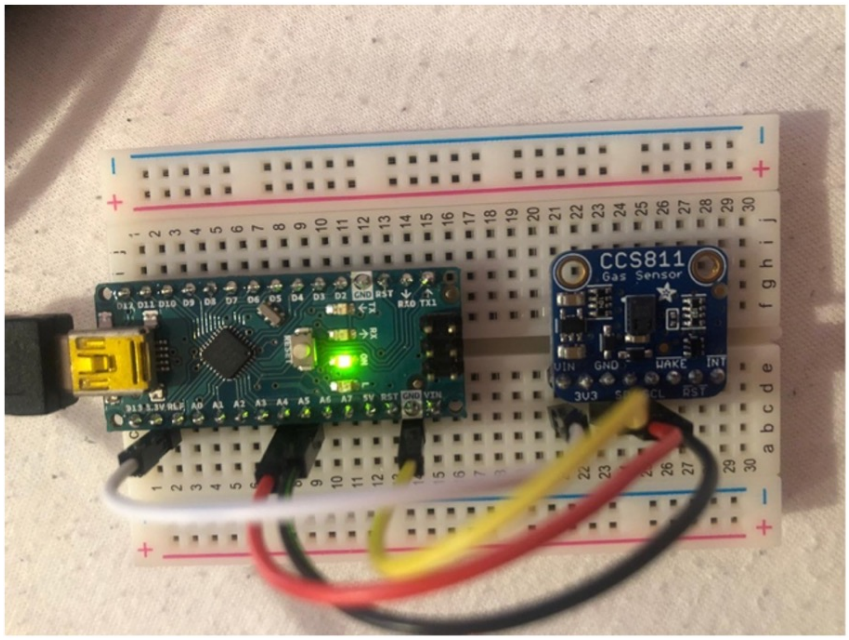

Figure 6: Zustandsautomat

Nachdem Der Sensor mit dem Arduino erfolgreich getestet wurde und auch die gewünschten

Ergebnisse mit „CO2“ und „VOC“ angezeigt wurden, haben wir uns die Arduino Library für den CCS-881 näher angeschaut, um den gesamten Hintergrund besser nachvollziehen zu können. Parallel dazu wurde das Datenblatt durchgelesen und die wichtigsten Informationen markiert und rausgeschrieben. Mithilfe der Übungsblätter gab es einen Ansatz, um mit dem ZigBee Code zu starten. Nachdem ein Teil des Codes entworfen wurde, sind wir die Library vom Arduino Code Zeile für Zeile durchgegangen, um genau zu verstehen was genau wo gemacht wird, um den Sensor richtig zu programmieren. Es wurde in Betracht gezogen die C++ Library in den ZigBee Code einzubinden. Am Ende haben wir aber davon abgesehen, da C++ Objekte mit Structs, sowie Arduino Methoden durch BitCloud Methoden zu ersetzen schwierig ist umzubauen und mehr Arbeit wäre als einfach den Code direkt in einen C Code zu Programmieren. Da wir aber die Gesamte Library durchgegangen sind, hatten wir ein gutes Verständnis über die Programmierung des CCS881 Sensor erlangt. Ähnlich wie die Arduino Library, haben wir erst einmal damit begonnen, die Hardware ID des CCS811 Sensors auszulesen. Das Auslesen der Hardware ID ist aber nicht mehr in der Abgabe erhalten. Der Sensor beginnt damit sich nach einer Verbindung mit einer Stromversorgung, zurück zu setzen, indem wir die Bytefolge 0x11, 0xE5, 0x72, 0x8A in das Register 0xFF schreiben. Die ersten Sensor Daten werden als hexadezimale Werte ausgegeben, um sicher zu stellen und zu prüfen, ob sich diese bei externem Einfluss (den Sensor anpusten) verändern. Die Werte der CO2 und VOC Rechnung werden mithilfe vom Bit-Shifting in die richtige Darstellung gebracht. Die I2C Steuerung und Callback Funktion sind in separate Dateien ausgelagert, um den Code auszuräumen. Da viele Callback Funktionen sehr identisch waren, wurden diese in eine Universelle Callback Funktion verbunden. Die PAN_ID_ID und CS_NWK_ADDR_ID werden mit der Funktion CS_ReadParameter ausgelesen und dann in einen HexString umgewandelt, Zu guter Letzt werden dann die Informationen (PAN_ID_ID, CS_NTW_ADDR_ID, _eCO2, _eTVOC) in einen JSON String gepackt, davon haben wir zwei, jeweils einen für den CO2 und einen für den TVOC. Nach der CS_NWK_ADDR_ID und der PAN_ID_ID folgen einmal "DATA_TYPE:eco2,value:valueOf_eco2" für unseren CO2 und einmal "DATA_TYPE:etvoc,value:valueOf_etvoc" für den TVOC. Anschließenden wird das Endgerät als Koordinator als separate Applikation gesendet und von diesem über die USART Bridge, welches die Kommunikationsschnittstelle ist, ausgegeben. Außerdem wurde das End Gerät als Sender der Daten mit dem SHT21 Sensor verbunden und an dem Netzwerk und der Kommunikation gearbeitet. Für die Übertragung der Daten über das Funknetz, haben wir uns dazu entschlossen, die Bandbreite der Nutzerdaten vollkommen auszunutzen, aber auch zu vermeiden, dass eine Änderung der Sicherheitseinstellungen, die Funktion der Sensoren beeinträchtigt.

Probleme Das Erste Problem, welches aufgetreten war, war die Allgemeine Nutzung vom Zigbee Gerät und dem Programmieren auf Atmel Studio. Die bisherigen Erfahrungen waren auf den Arduino begrenzt und das Übertragen vom Arduino auf das Atmel Studio war doch nicht so einfach wie gedacht. Nach durcharbeiten der Datenblätter und den Übungsblättern war es verständlicher und es ging besser voran als am Anfang. Im Laufe des Projekts traten immer mal wieder einige Schwierigkeiten auf, wie zum Beispiel das oben angestoßte Problem, welches man auf keinen Fall vergessen sollte, und zwar die Verbindung zwischen dem Sensor und dem Funkmodul („Wake“ und „GND“) richtig anzuschließen oder wie die Erstellung der Rechnungen, um Die Werte CO2 und VOC ausgeben zu lassen. Nachdem wir nun die Richtige Rechnung gefunden hatten, war das Nächste Problem, dass Die Werte nicht ausgegeben wurden beziehungsweise es zu keiner Ausgabe kam. Um diesen Fehler ausfindig zu machen, sind wir lange den Code durchgegangen und haben Debug Ausgaben

hinzugefügt, um zu schauen, bis wo der Code läuft und haben einiges von Grund auf neu geschrieben um den Fehler so zu fixen.

Ein weiters Problem war es, das die sizeOf() Funktion in C nicht mit Bytefolgen wie beim des Zurücksetzen des Sensor funktioniert, die Größe des 5 Byte wird von der sizeOf() Funktion zufällig als 2 oder 3 Bytes zurückgegeben. Indem wir die Anzahl der auf den i2c Bus zu schreibenden Bytes statisch setzen, wurde dieses Problem gelöst.

3.4 Omar Osman

3.4.1 Einbindung des SHT21 Sensors

Um den SHT21 Sensor einzubinden, mussten folgende Teilaufgaben abgeschlossen werden:

- Endgerät als Koordinator (separate Applikation)
- End Gerät als Sender der Daten verbunden mit dem SHT21 Sensor
- Netzwerkaufbau und Kommunikation
- UART war die Kommunikationsschnittstelle

Um den HAL_I2cDescriptor verschieden callback‘s je nach Zustand zu geben, haben wir zuerst eine „configure“ Funktion geschrieben. Dies hat den callback und die Länge für diesen Zustand konfiguriert. Im Laufe des Projektes wurde die Funktion mit dem Funktionen HAL_Open_i2c und dem Write_i2c Funktion erweitert und in die Datei i2contoller.c als die Funktion setRegister ausgelagert. Für das lesen von Registern hast die neue Methode readRegister welche ebenfalls im i2contoller.c ist. Nachdem wir den Callback Funktion und die Länge für den lese oder schreibe-Zustand gesetzt haben, wird dieser ausgeführt und nach Abschluss unsere gesetzte Callback Funktion ausgeführt.

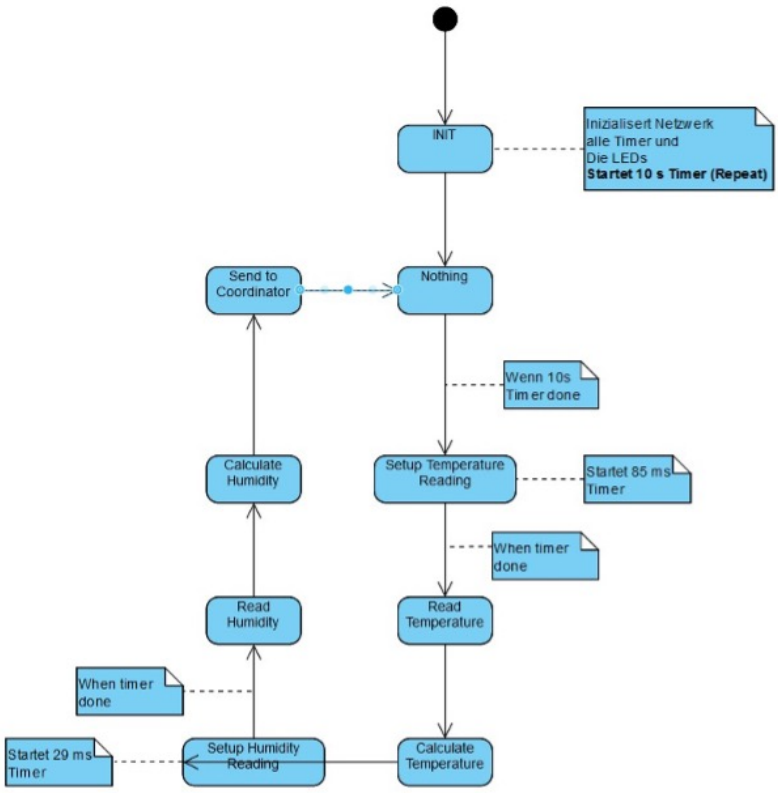

Figure 7: Zustandsautomat

Zustandsautomaten ermöglichen uns schneller und genauer den Fehler im Programm zu finden. Bei APL_TaskHandler geht es um den Taskhandler, der den Ablauf der Anwendung steuert. Dies geschieht in Form eines Zustandsautomaten, deswegen ist am einfachsten, das Laufen des Programmes durch einen Zustandsautomaten zu beschreiben. Der Microcontroller geht beim Programmstart in den Initialisierung Zustand INIT. In diesem Zustand wird der BitCloud Stack mit allen notwendigen Daten für das Auslesen des Sensors (HAL Stack) und die Übertragen der Daten (Network Stack) an den Koordinator initialisiert sowie ein 10 Sekunden Timer gestartet. Vom Initialisierung Zustand springen wir dann erst einmal in den Nothing Zustand, und warten bis der Timer abgelaufen ist. Wenn der Timer abgelaufen ist, wechseln wir in den Zustand Setup Temp. Dieser Konfiguriert den Sensor so dass dieser die Temperatur misst, und wir sie dann 85 Millisekunden später vom Sensor auslesen können. Diese Verzögerung wird benötigt, damit der Sensor genügend Zeit hat, die Daten zu messen, vorzubereiten und bereitzustellen. Der Sensor liefert einen Wert von 16 Bits zurück, die letzten zwei Bits von den 16 Bits sollen auf null gesetzt werden. Deswegen shiften wir die ganzen 16 Bits 2 Stellen nach rechts und dann wieder zurück um 2 Stellen nach Links. Darauffolgend verarbeiten wir die gelesenen Werte mit der aus dem Datenblatt entnommenen Formel, welche erstmal die Werte durch 65536 (216) dividiert, bevor das Ergebniss dann mit 175,72 multipliziert und am Ende noch -46,85 abgezogen wird. Jetzt

bestimmen wir noch ob die Zahl positiv oder negativ ist und fügen dann in den JSON String ein + oder – Symbol ein und als aller letztes, wird dann der Temperaturwert in einen JSON String umgewandelt und hinter das + oder Minus Symbol in zwei Schritten geschrieben. Einmal die Stellen vor dem Komma, und das andere mal die Stellen nach dem Komma. Nach dem Auslesen der Temperatur, wird eine ähnliche Prozedur noch einmal für die Luftfeuchtigkeit wiederholt, nur das wir in diesem Fall nicht 85 ms sonder nur 29 ms auf das Ergebnis warten müssen und kein + oder – Symbol haben, da eine negative Luftfeuchtigkeit nicht möglich ist.

3.5 Emin Gaplan

3.5.1 Einbindung des MG881 Sensors

Mein Sensor den ich zu bearbeiten habe, war nach absprache, der SEN0159 von DFRobot. Um mit meinen Sensor zu arbeiten habe ich folgende Hardware benutzt:

- ZigBee-Module
- Sensorboard SEN0159 von DFRobot (MG-811)
- Joy-it 3,3V/5V Spannungsversorgung für Breadboards inkl. Netzteil
- FTDI Uart-Bridge
- Arduino Nano, Breadboard und Kabel zum Testen der Sensoren

Zu allerserst musste ich um den Sensor zu verstehen was der genau macht und wie der funktioniert nachdem Datasheet suchen und es einmal durchlesen. Da habe ich erfahren das der Co2 Sensor eine Eingangsleistung zwischen 3.3V-6V(wobei 5V empfohlen wird) und die Ausgänge vom Sensor, welches wo bei einen Arduino dran kommt.

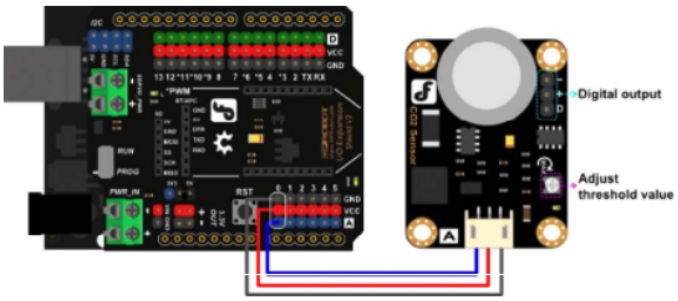

Figure 8: Zigbee

Also habe ich es erst einmal mit dem Arduino verbunden, ich habe den blauen kabel auf den Bild bei A0 angesteckt, den roten bei 3.3V und den grauen bei 3.3V angesteckt. Daraufhin wollte ich mit den programmieren anfangen, bevor ich mit dem ZigBee loslegen wollte, hatte ich mir erst einmal vorgenommen es mit den Arduino zu verbinden. Zu allererst habe ich mir das Programm Arduino heruntergeladen und installiert.

In meinem nächsten Schritt habe ich erstmal recherchiert wie genau ich den Arduino programmieren sollte. Aus dem Datasheet gab es einen Code mit den ich es einmal für den Arduino probiert habe. Doch gab es mit dem Code eine fehlerhafte Ausgabe, also habe ich mir den Code einmal genauer angeguckt und festgestellt das der Code eher für den Sensor MG-811, der im Sensor SEN0159 ist, gemacht ist.

Also habe ich genauer nachdem SEN0159 recherchiert mit ein paar hilfen aus Stackoverflow und noch ein paar anderen Seiten habe ich einen neuen Code geschrieben. Mit C++ eine cpp datei, habe dazu einen header noch definiert und noch eine ino datei für die Arduino. Mit den Programm Putty habe ich folgende Ausgaben bekommen:

```
17:23:34.058 -> CO2 value: 404
17:23:36.072 -> CO2 value: 404
17:23:38.108 -> CO2 value: 396
17:23:40.137 -> CO2 value: 408
17:23:42.161 -> CO2 value: 384
17:23:44.189 -> CO2 value: 384
17:23:46.205 -> CO2 value: 392
17:23:48.236 -> CO2 value: 399
17:23:50.249 -> CO2 value: 388
17:23:52.296 -> CO2 value: 399
17:23:54.300 -> CO2 value: 388
17:23:56.323 -> CO2 value: 396
17:23:58.349 -> CO2 value: 392
17:24:00.375 -> CO2 value: 399
17:24:02.412 -> CO2 value: 396
```

Figure 9: Putty

Wie man im ertsen Blick sehen kann sind die Werte eigentlich für einen Innenraum ganz durchnittlich, doch hat sich beim reinpusten ins Sensor leider nur wenig verändert, mal auch nur leicht und ziemlich langsam angestiegen. Daran habe ich auch gemerkt das was an diesen Sensor nicht stimmt.

Auch nachdem ich vieles am Code verändert habe, konnte ich es nicht hinbekommen mit den Arduino aud ein richtiges Ergebniss zu kommen, also habe ich mir vorgenommen es direkt auf dem ZigBee zu programmieren. Nach absprache mit dem Professor hat er mir geraten erstmal einen Zustandautomaten für den Programm, den ich auf ZigBee programmieren will zu machen. Also habe ich erst einmal recherchiert wie ich bei einem Zustandautomaten am besten anfangen kann. Nach einigen versuchen kam ich auf das folgende Ergebniss: siehe nächste Seite.

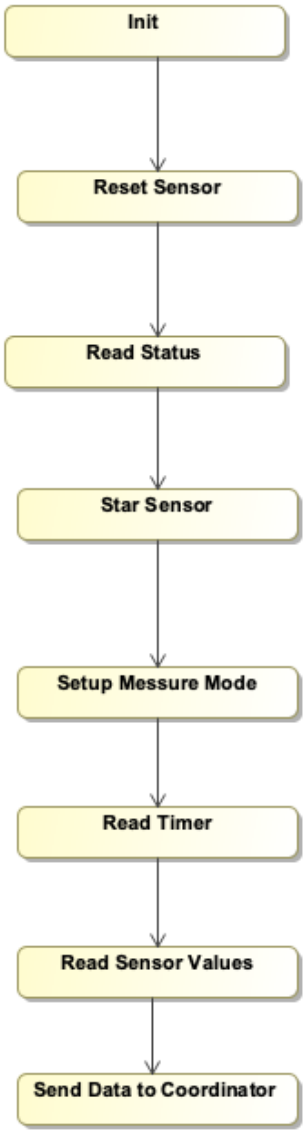

Figure 10: Zustandsautomat

Nachdem ich den Zustandsautomaten gemacht habe, habe ich mich an das programmieren mit dem Zigbee gemacht. Zu allererst habe ich mich bevor ich angefangen habe die Aufgabe vom Blatt 2 noch einmal angeguckt in der wir mit dem ADC- Sensor gearbeitet haben um noch einmal genauer zu wissen wie ich voran gehe am besten. Als erstes habe ich die UART initialisiert, als static void.

Daraufhin habe ich den Timer auch als static void initialisiert. Nach ein paar weiteren zeilen code, habe ich es den ADC port geöffnet, um die Werte des Sensors auszulesen. Die

Daten des Sensor wollte ich versuchen zum Koordinator zu schicken, doch hatte ich dort noch ein paar errors.

Also dachte ich mir ich versuche das ganze zu allererst einmal ohne den Koordinator, habe ein paar Zeilen vom Code raus genommen, die dazu gehören. Ich habe es daraufhin noch einmal versucht auszugeben, doch hatte ich keinerlei Erfolg, auch nach mehreren Recherchen und umschreibungen des Codes kam ich nie zum richtigen Ergebnis.

3.5.2 Überwachung der Batteriespannung der Funkmodule

4 Benutzte Standards und Werkzeuge, Eingesetztes Equipment

Es wurde mit Speziellem Equipment und Standards gearbeitet, um den CCS811 - Sensor erfolgreich zum Laufen zu bringen und im Anschluss benutzen zu können. Abgesehen von dem Equipment rund um den Sensor, wurde sehr oft zusammen am Sensor gearbeitet und auch dafür einige Standards zur Kommunikation in Anspruch genommen.

- Zwei Mikrokontroller (Type ATMega256RFR2)
- Sensor SHT21, CCS 881
- Programmiergerät (AVR Atmel)
- Atmel Studio
- Batterie (Funkmodul Anschluss)
- USB- Kabel
- USART Bridge
- Whatsapp
- Zoom
- GitHub Desktop und Web
- Notpad ++
- Etherpad
- Visual Paradimen Online
- Putty, Atmel Wireshark Sniffer Interface Tool
- Visual Studio Code
- Docker for Windows
- Docker Desktop
- npm
- Visual Studio Code

5 Teamarbeit

Die Teamarbeit war sehr gut in Takt. Wir haben viele Dinge über Atmel gelernt. Es gab Meinungsverschiedenheiten, wenn einer von uns mit einer Idee nicht einverstanden war, hat er die seine Meinung dazu geäußert und hat erklärt warum ebendiese Idee oder solcher Vorschlag nicht akzeptabel ist und wir kamen zu jedem beliebigen Zeitpunkt zu einer Antwort. Wir haben uns zu jedem beliebigen Zeitpunkt respektvoll unterhalten, egal in welcher Situation wir waren.

6 Ausblick und Erweiterungsmöglichkeiten der Anwendung

Das Programm könnte noch in vielen Richtungen erweitert werden, zum Beispiel man könnte noch einen Melder an das Projekt anbringen, der blinken kann, sobald die Luftqualität schlecht wird.

Potenzielle Erweiterungsmöglichkeiten währen, Die Sensordaten des SHT-21 (Temperatur und Luftfeuchtigkeit) oder LM-73 (nur Temperatur) and den CCS-881 als Environmental Data (ENV-Data) zu übermitteln, um noch genauer Messergebnisse zu bekommen. Eine Version mit den einbinden des LM-73 ist sogar bereits in 90Des Weiteren wäre es möglich die Sicherheitseinstellungen bei der Übertragung zu erhöhen. Der Code ist sogar für ein solche Option bereits ausgelegt, es wurde aber nie getestet. Anstelle eines Timer könnte man auch ein Externen Interrupt verwenden um dem Mikrocontroller mitzuteilen, das die Daten bereit sind.

References

[1] Adafruit cs811 library. https://adafruit.github.io/Adafruit$_C CS811/html/class_adafruit___c_c_s811.html$.

[2] Arduino css811 library. https://stackoverflow.com/questions/141525/what-are-bitwise-shift-bit-shift-operators-and-how-do-they-work, .

[3] Arduino css811 library. https://stackoverflow.com/questions/141525/what-are-bitwise-shift-bit-shift-operators-and-how-do-they-work, .

[4] C++ klassen in structs umbauen januar 2017 von nutzer freestyle4:. https://stackoverflow.com/questions/41707545/converting-class-to-struct, .

[5] Datenblatt zum atmega256fr2. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8393-MCU_Wireless-ATmega256RFR2-ATmega128RFR2-ATmega64RFR2_Datasheet.pdf%5B%7B%22num%22%3A

[6] Eckstein komponente zugriffsdatum (03.02.2021):. https://eckstein-shop.de/GY-CCS811-Gas-Air-Quality-Sensor-Module.

[7] Bitwise shift. https://docs.microsoft.com/de-de/cpp/c-language/bitwise-shift-operators?view=msvc-160.

[8] Youtube. https://stackoverflow.com/questions/141525/what-are-bitwise-shift-bit-shift-operators-and-how-do-they-work.

[9] Youtube. https://www.youtube.com/watch?v=eNhc5yxZ2z4ab_channel=WSN%26IoT, .

[10] Youtube. https://www.youtube.com/watch?v=eNhc5yxZ2z4feature=youtu.beab_channel=WSN%26IoT, .

[11] Youtube. https://www.youtube.com/watch?v=zmSdeJ4ASrgab_channel=WSN%26IoT, .

[12] Bitcoud porting guide von atmel corporation 2011:. http://ww1.microchip.com/downloads/en/AppNotes/doc8430.pdf.

[13] Zigbee applications sending and receiving data. https://www.eetimes.com/zigbee-applications-part-1-sending-and-receiving-data/.

[14] Bitcloud api reference. http://www2.ee.ic.ac.uk/t.clarke/projects/Resources/BitCloud/BitCloud_ZDK_1_4_1/Docum jan 2015.
ätterätter

[ä] Aufgabenblätter, 10 2020.