Weekly Reports

Luftqualität in Innenräumen - Gruppe 1

10. Januar 2022

Name	Matrikel Nr.	Arbeitsaufwand (h)
Friedrich Just	1326699	20,00
Stipe Knez	1269206	23,00
Lucas Merkert	1326709	20,00
Achim Glaesmann	1309221	23,50
Max-Rene Konieczka	1211092	23,00
Can Cihan Nazlier	1179244	20,00

Tabelle 1: Arbeitsaufwand dieser Woche

1 Überblick

1.1 Friedrich Just

Es wurde sich entschieden, alle Daten im JSON-Format an den Koordinator zu schicken. Der Koordinator übernimmt das Array und gibt es über die UART-Bridge an die COM-Schnittstelle weiter. Dies hat die Vorteile, dass der Koordinator keine Formatierungen am Array durchführen muss und es am Endgerät liegt, die Daten im richtigem Format zu schicken. Dies hat den Nachteil, dass viele Bytes für das Format mitgeschickt werden. Dies ist aber kein Problem, da das Zigbee-Netzwerk eine Datenrate von 250 kbps besitzt. Das JSON-Format ist so aufgebaut, dass als erstes die ID des Endgerätes gesendet wird. Danach folgen die verschiedenen Sensornamen und danach folgen die verschiedenen Messwerte. Nach der Festlegung des JSON-Formats wurde dieses auch korrekt übertragen. Allerdings gibt es Probleme mit der Verbindung beider Programme zum Messen der Sensoren und zum Verschicken an den Koordinator. Es kommen keine Daten an bzw. es werden keine versendet.

1.2 Stipe Knez

Nachdem zu Beginn der Weihnachtsferien die Struktur der Mockdaten an die der richtigen Daten nochmals angepasst wurde, ging es daran, die gesendeten

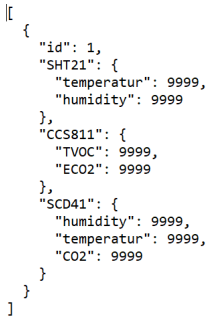

Abbildung 1: Beispiel der JSON Datei

Daten im Backend zu empfangen, zu parsen und schließlich in die Datenbank abzuspeichern.

Das Erstellen und Senden der Mockdaten geschah mit einer Python Anwendung und einem Programm zur Erstellung virtueller Ports. Das Empfangen und Auslesen der ankommenden Mockdaten wurde im Backend mit Hilfe des Serial-Port packages von Node.js realisiert. Das Einspeichern der eingehenden Daten ist uns auch erfolgreich gelungen.

Für den weiteren Verlauf sind Optimierungen in Hinblick auf den Verarbeitungs- und Sendeprozess der ankommenden Daten geplant.

1.3 Lucas Merkert

Über die Weihnachtsferien: Einarbeitung in die Funktionalität des Sensors CCS811:

- Schreiben auf den Sensor über die Adresse 0x5A
  1. Schreiben des Befehls 0x40 in das Register 0x01
  2. Bisher return 0 von HALWriteI2CPacket()
- Lesen des Sensor über die Adresse 0X5B
  1. Lesen von 4 Bytes zum Lesen der CO2 und TVOC werte im Register 0x02
  2. Bisher return 0 von HALReadI2CPacket()

Der Wake-Pin ist zurzeit an GND angeschlossen. Allerdings scheint noch etwas nicht zu funktionieren, diesen soll in der nächsten Woche geklärt werden.

1.4 Achim Glaesmann

Über die Vorlesungsfreie Zeit wurde entsprechend der Aufgabenverteilung das Auslesen des SCD41 realisiert, so wie eine Firmware entwickelt, welche in der

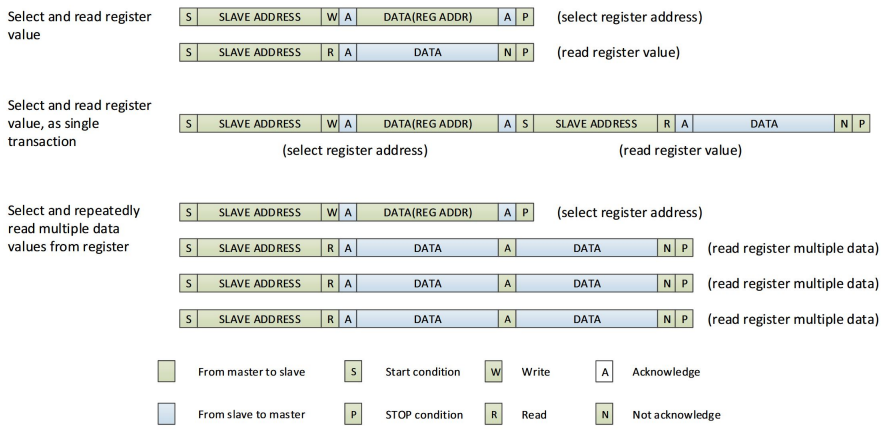

Abbildung 2: Funktionsweise der I2C-Packet Übertragung[1]

Lage ist sowohl den SHT21 wie auch den SCD41 parallel auszulesen. Das auslesen des SCD41 bereitete zunächst enorme Schwierigkeiten. Verschiedene in der Firmware des Atmel Mikroprozessors eingebaute Funktionen verursachten schwer nachzuvollziehende Bugs. Den Versuch die I2C Schnittstelle über den Befehl HAL_CloseI2cPacket(&i2cdescriptor) an einer Stelle im Code außerhalb der Callback Funktion des Descriptors, führte zu einem Bug bei welchem in ständiger Wiederholung der Mikroprozessor neu startete. Der Neustart trat allerdings nicht an der Stelle auf an dem die Funktion aufgerufen wurde. Das Problem konnte behoben werden, nachdem die $\mathrm{HAL}_{C}loseI2cPacket()Funktionnurnochinderdem Descriptorzugeh$

```
1  static void delaytimer(uint16_t time, uint8_t _next_appstate){
2    delayTimer.interval = time;
3    next_appstate = _next_appstate;
4    HAL_StartAppTimer(&delayTimer) //startet timer delayTimer welcher nach der angegebenen
         Zeit in ms die Funktion delayTimerComplete(); aufruft
5  }
6
7  static void delayTimerComplete(){
8  appstate = next_appstate;
9  SYS_PostTask(APL_TASK_ID);
10 }
```

Listing 1: Delay Timer

Es war daraufhin nötig Zustände, die eine Unterbrechung des Programmablaufs benötigten um die Rechenaufgaben der Sensoren zu ermöglichen, zu trennen. Es war möglich die Applikation in 10 Zustände zu unterteilen.

```
1  #define APP_NOTHING_STATE    0
2  #define APP_STARTUP_STATE     1
3  #define APP_RESET_SENSOR_STATE    2
4  #define APP_INIT_STATE     3
5  #define APP_CALL_FOR_READ_SCD_STATE   4
6  #define APP_CALL_FOR_READ_TEMP_SHT_STATE  5
```

```
7  #define APP_CALL_FOR_READ_RH_SHT_STATE  6
8  #define APP_READ_SCD_STATE    7
9  #define APP_READ_TEMP_SHT_STATE   8
10 #define APP_READ_RH_SHT_STATE    9
11 #define APP_AUSGABE_STATE     10
```

Listing 2: APPSTATE definitions

Die I2C Verbindung zu den einzelnen Sensoren konnte über den eingeführten Delay stabilisiert werden. Die Kommunikation zum SCD41 warf jedoch weiterhin Fehler auf, die zunächst nicht nachvollzogen werden konnten. Nach testen des Sensors auf dem Arduino und dem recherchieren der Funktionen der von Sensirion bereitgestellten Arduino-Library wurde erkannt, das zum erfolgreichen Starten der periodischen Messung zunächst ein Reset des Sensors durchgeführt werden muss. Das unterbrechen der Stromversorgung zum Sensor scheint hierbei nicht zwingend auszureichen. Ein erneutes übersenden des Befehls zur Periodischen Messung führt ohne den vorherigen Reset dann nicht zu der erwünschten Messung. Weiterhin ist es nötig den Sensor vor der ersten I2C-Kommunikation für eine Zeit mit Strom zu versorgen damit dieser die zur Funktion nötigen Anweisungen ausführen kann. Das der Befehl nach Neustart des Sensors dennoch zurückgesetzt werden muss, war leider nicht eindeutig im Datenblatt vermerkt, weswegen auch das finden des Fehlers in dieser Kommunikation beachtliche Zeit in Anspruch nahm. Zuletzt war es jedoch möglich beide Sensoren auszulesen und in einer Firmware zu verbinden. Nachdem meine Teamkollegen die 2.4Ghz Verbindung so wie den CCS881 erfolgreich implementiert haben ist es möglich die Firmware abschließend zu entwickeln, so dass die Sensoren in den jeweiligen Räumen platziert und mit der Applikation verbunden werden können.

1.5 Max-Rene Konieczka

Während zurzeit noch daran gearbeitet wird die Daten der Sensoren auszulesen, wurde zunächst versucht die zuvor mit Python erstellten Mockdaten, in unsere Datenbank einzulesen. Dafür haben wir das Serialport Package von Node sowie ein Programm zur Erstellung von virtuellen Ports verwendet. Es hat sich die Frage gestellt, ob und wie die Mockdaten in der Datenbank angezeigt werden könnten. Das Abspeichern hat funktioniert, doch ist der Gedanke aufgekommen, dass wir die Daten direkt auf einem Modul, mit Atmel Studio erstellen sollten. Da nämlich zusätzliche Software benutzt wird, müsste diese von zukünftigen Benutzern ebenfalls heruntergeladen und installiert werden. Dies soll allerdings verhindert werden.

1.6 Can Cihan Nazlier

Das Abspeichern von Sensoren in den Räumen in Echtzeit, wurde nun in der Datenbank realisiert. Das heißt, sobald ein Sensor aus einem Raum entfernt wird und ein anderer hinzugefügt wird, dann bekommt das Backend dies mit und gibt diese Information an die Datenbank weiter. Somit besteht nun eine Verbindung

vom Raum und Sensor über deren ID's. Zudem wurde das Abspeichern und Abfangen der Mockdaten realisiert. Die Mockdaten werden momentan mit Python erstellt, dies gab einigen zu Bedenken dass es ohne zusätzliche Software evtl. nicht klappen könnte in den Entwicklungsumgebungen aller Entwickler, sodass der nächste Schritt ist, die Mockdaten via eines Zigbee Moduls zu senden. Im gleichen Schritt wird als Nächstes das Dashboard erstellt.

Literatur

[1] *Datasheet CCS811*. URL: `https://learn.adafruit.com/adafruit-ccs811-air-quality-sensor?view=all#documents`.