Gefördert durch:

Bundesministerium für Wirtschaft und Klimaschutz

aufgrund eines Beschlusses des Deutschen Bundestages

Technische Hochschule Rosenheim

MonSec

XBee – Setup (API-Mode)

Akkubetriebene Version mit INA219

Im Projekt MonSEC- Monitoring Secure

TH Rosenheim

Bearbeiter: Markus Hartmann

Stand: 06.12.2022

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Diese Dokumentation beschreibt den Aufbau von mobilen Sensorknoten zum Messen von Raumluftdaten (Temperatur, Luftfeuchte sowie Luftdruck) und Übertragung der Daten über XBee Funkmodule (Firma Digi). Die Sender werden auf Basis eines Arduino Uno (bzw. Arduino Zero) mit dem Luftdaten Sensor Bosch BME280 realisiert. Als Empfänger und Datenhub wird ein Raspberry Pi verwendet.

1.1 Quellen

1.1.1 Offizielle Quellen

XBee API Mode

https://www.digi.com/resources/documentation/Digidocs/90001942-13/containers/cont_api_mode.htm?tocpath=XBee%20API%20mode%7C_____0

XBee Reichweitentest

https://de.digi.com/blog/post/wireless-communication-range-testing-with-digi-xct

1.1.2 Blog Artikel und sonstiges für Entwicklung

https://www.instructables.com/How-to-Use-XBee-Modules-As-Transmitter-Receiver-Ar/

https://www.ardumotive.com/how-to-use-xbee-modules-as-transmitter--receiver-en.html

2 Aufbau und Setup

2.1 XBee Module

Die XBee Module werden über die von Digi bereitgestellte Software XTCU konfiguriert.

2.1.1 Coordinator

Der Connector stellt den Sammelknoten dar, der als Koordinator des Netzwerkes und der Sammlung der Daten entspricht. Die beiden wichtigen Einstellungen sind in der folgenden Darstellung rot markiert.

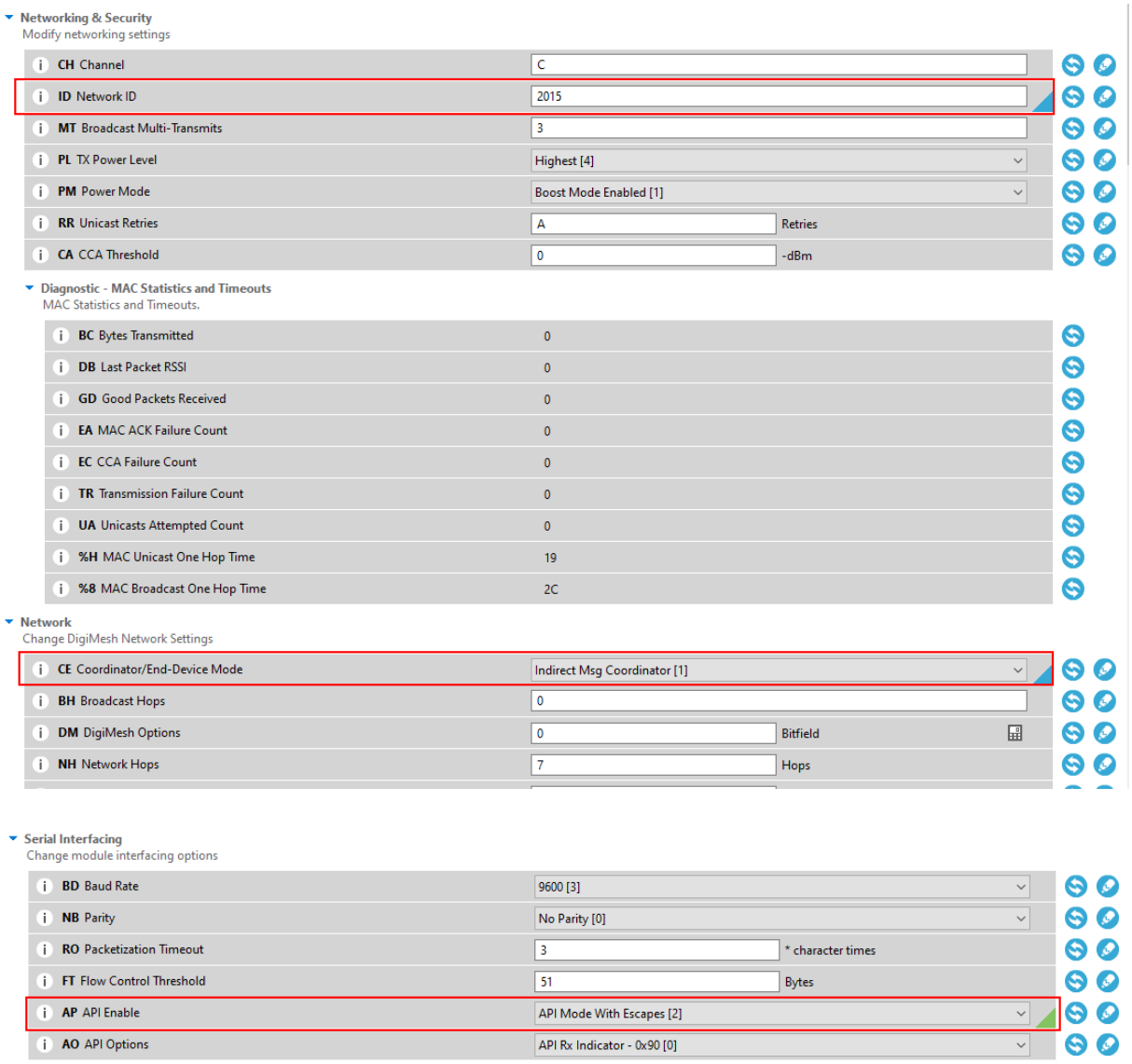

Abbildung 1: Parameter des Coordinators

Es sollte auch die Adresse des Coordinators ausgelesen werden, damit die Endknoten diese als Empfänger der Nachricht erkennen.

2.1.2 Endknoten

Die Endknoten sind die Sensorknoten. Diese stellen die Daten dem Coordinator zu Verfügung.

Die Endknoten müssen im gleichen Netzwerk – ID sein, damit diese Nachrichten an den Coordinator verschicken können. Die Parametrierung ist in der folgenden Darstellung abgebildet.

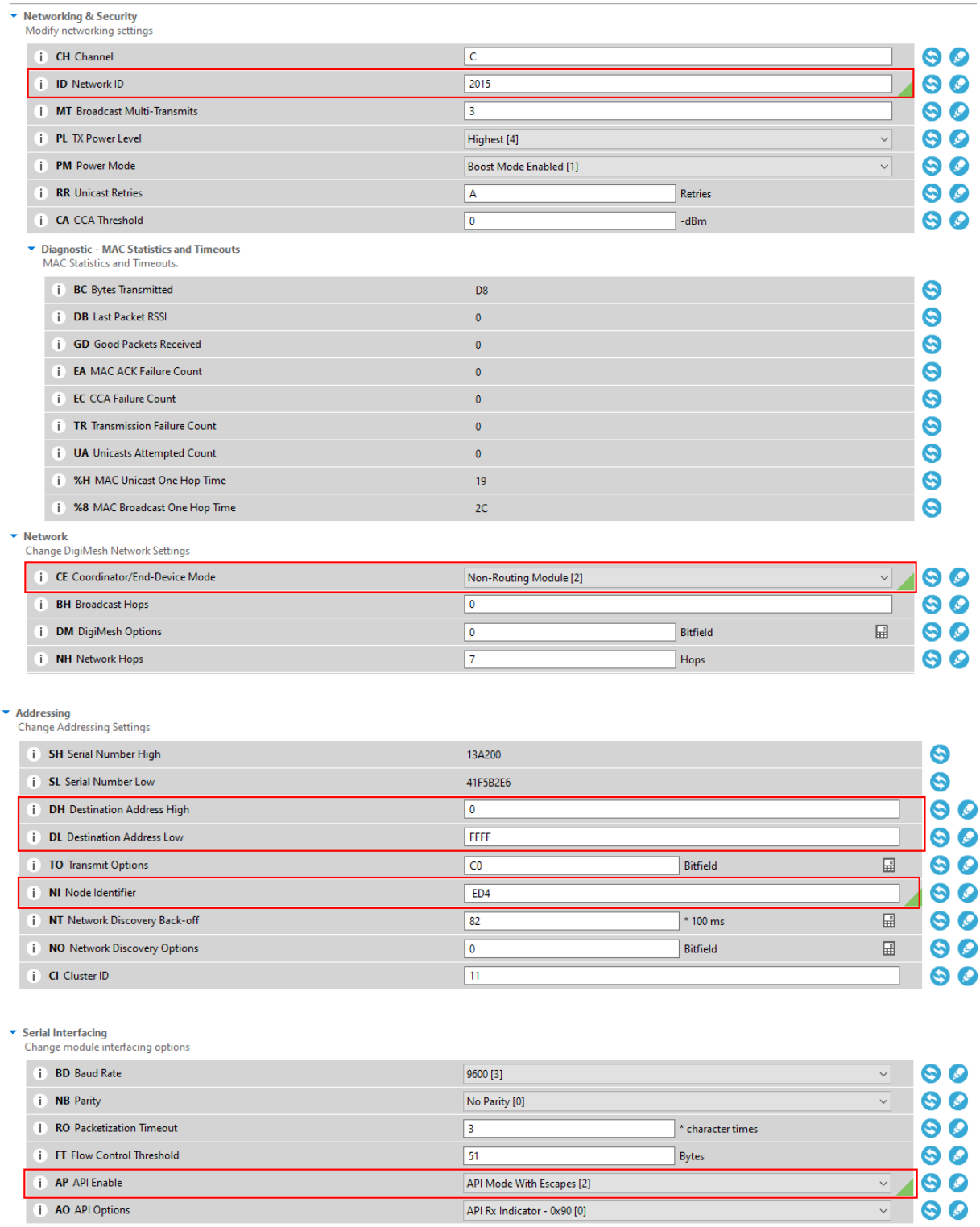

Abbildung 2: Konfiguration der Endknoten

Der Parameter NI Node Identifier ist für jeden Knoten verschieden zu vergeben

2.2 Arduino

Für dieses Projekt wird ein Arduino Uno (bzw. Arduino Zero) verwendet.

Für die Verbindung zwischen Arduino und Xbee wird ein Shild der Firma DF Robot verwendet.

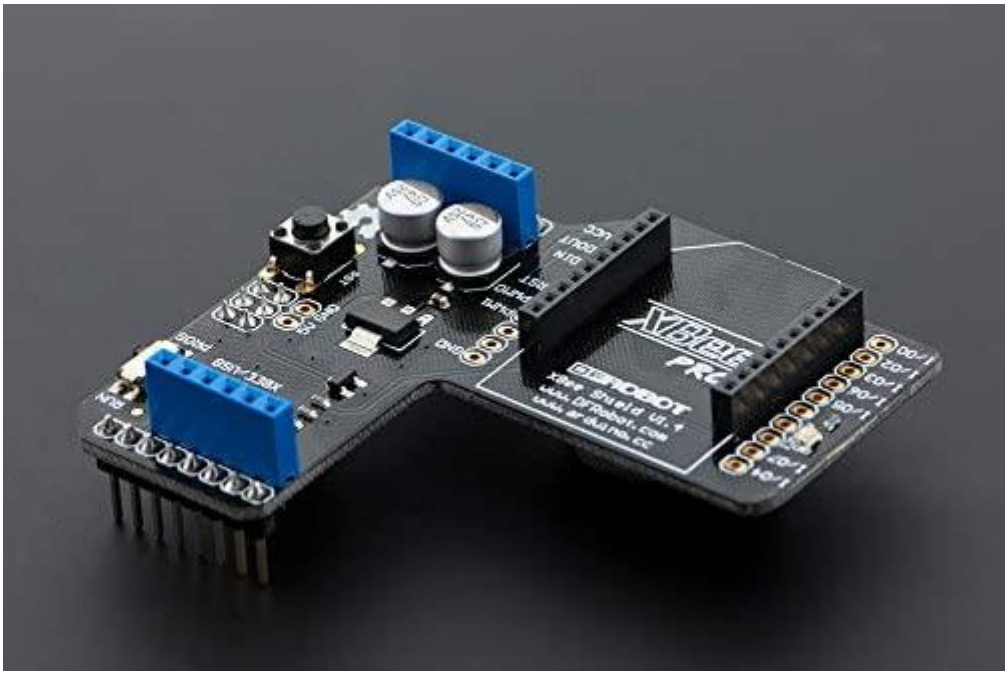

Abbildung 3: XBee Shield für Arduino [DFRobot]

Die Anleitung für das Shild ist auf der folgenden Seite einzusehen:

https://wiki.dfrobot.com/Xbee_Shield_For_Arduino__no_Xbee___SKU_DFR0015_

Als Bibliothek für die XBee Kommunikation wird auf dem Arduino die folgende Bibliothek verwendet:

https://github.com/andrewrapp/xbee-arduino

Anmerkung:
Beim Verwenden eines Arduino Zero erfolgt die Kommunikation des XBee mit dem Arduino über die Funktion Serial1 (vgl. Code im Anhang).

2.2.1 Sensor – BME280

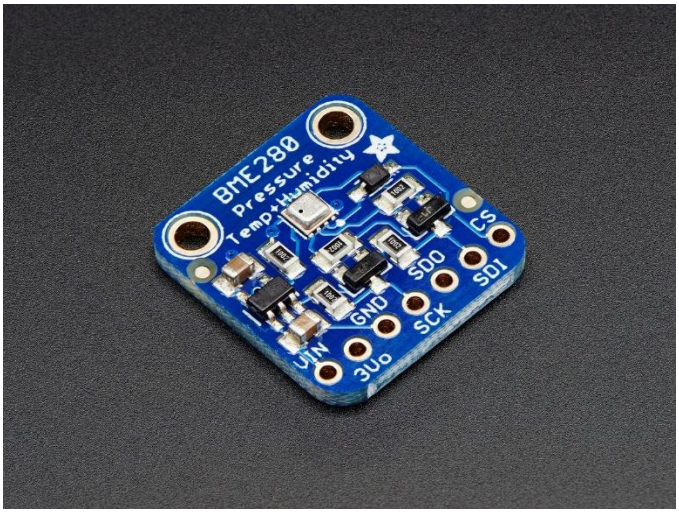

Abbildung 4: BME280 [Adafruit]

Als Sensor wird ein BME280 von der Firma Bosch verwendet. Es handelt sich um einen Kombinierten Sensor für Temperatur, Luftfeuchte und Luftdruck.

Für den leichteren Einsatz wird eine fertig Bestückte Platine von der Firma Adfruit verwendet. Der Sensor wird über den I2C Bus an den Arduino verbunden. Er wird nach folgendem Schema an den Arduino angeschlossen:

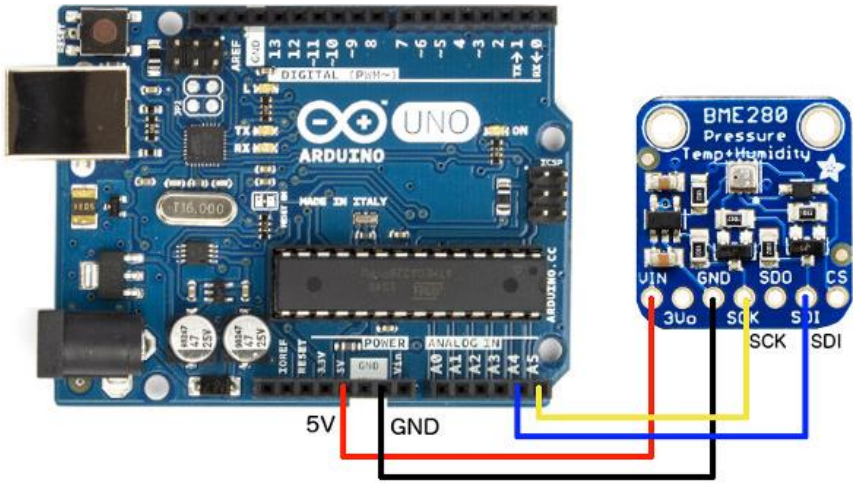

Abbildung 5: Anschlussschema BME280 an Arduino [http://cactus.io/]

Die Bibliothek für den Sensor ist auf Github zu finden.

https://github.com/adafruit/Adafruit_BME280_Library

Der Sensor wird durch das Programm abgefragt und die Daten werden in Binärer Form an das XBee Empfängermodul gesendet. Der Code auf dem Arduino ist im Anhang (Kapitel 3.1.1) zu finden.

2.2.2 Sensor Ina219

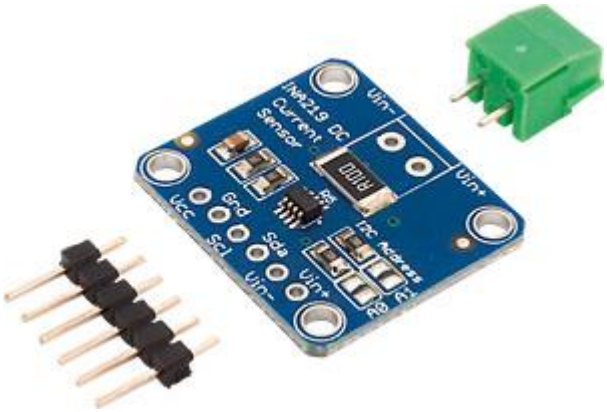

Abbildung 6: Breakoutboard Ina219 [Adafruit]

Als Überwachung der Akkubetriebenen Spannungsversorgung wird ein der Ina219 verwendet. Dieser ist als Breakoutboard von der Firma Adafruit erhältlich. Der Anschluss der Messtechnik an den Arduino erfolgt über den I2C-Bus. Zur Überwachung des Stromkreises, in welchem gemessen werden soll, wird der Ina219 vor dem Vebraucher geschaltet. Eine typische Schaltung des Ina219 sieht wie folgt aus:

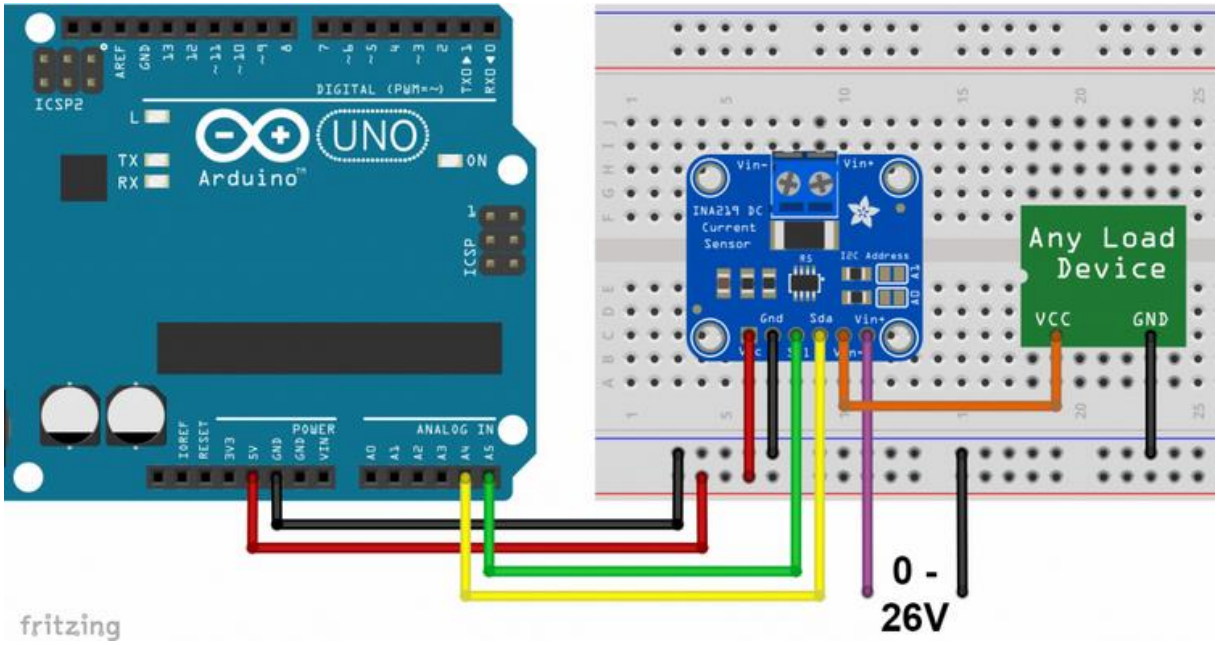

Abbildung 7: Anschlussschema INA219 [Adafruit]

Die Messung erfolgt über den Spannungsabfall des integrierten Shunt-Widerstandes.

Die Arduino Bibliothek für die konfiguration des Ina219 im Arduino Programm ist direkt von Adafruit erhältlich.

https://github.com/adafruit/Adafruit_INA260

Die gesamte Schaltung des Senderknotens sieht wie folgt aus.

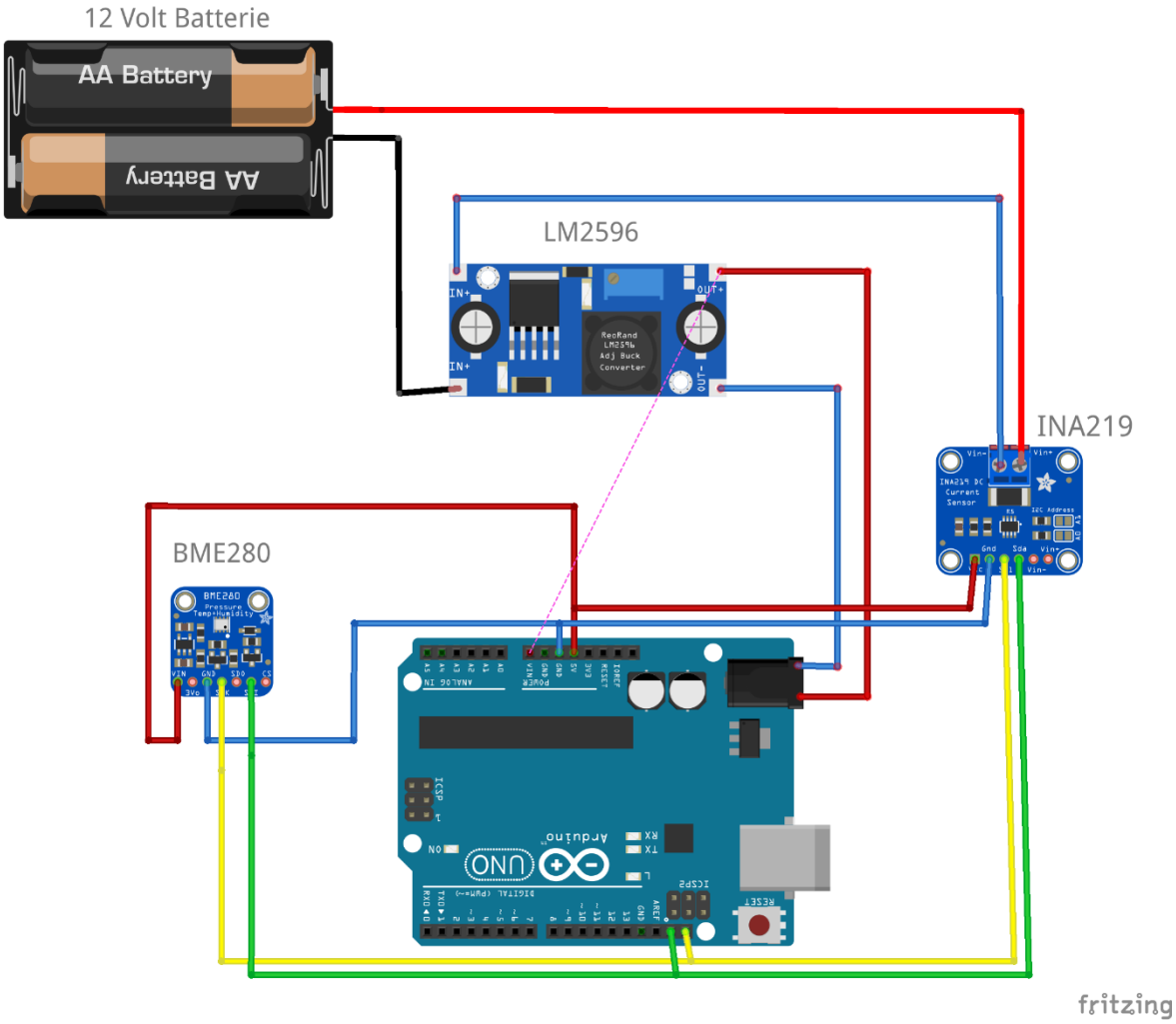

Abbildung 8: Gesamtschaltbild des Sensorknotens

Zur Sicherheit wurde noch ein Buck-Down DC/DC Konverter (LM2596) zur Stabilisierung der

```
var Source = msg.payload.remote64;
msg.measurement ="home";
// Mapping für BME280
if (Source == "0013a20041d5c40a"){
    msg.payload = {
    Temperature01: msg.payload.data.readFloatBE(0),
    Humidity01:  msg.payload.data.readFloatBE(4),
    Pressure01: msg.payload.data.readFloatBE(8),
    };
}
// Mapping für BME280+INA219
if (Source == "0013a20041d5bf55"){
    msg.payload = {
    Temperature02: msg.payload.data.readFloatBE(0),
    Humidity02:  msg.payload.data.readFloatBE(4),
    Pressure02: msg.payload.data.readFloatBE(8),
    Shuntvoltage: msg.payload.data.readFloatBE(12),
    Busvoltage: msg.payload.data.readFloatBE(16),
    Current: msg.payload.data.readFloatBE(20),
    Loadvoltage: msg.payload.data.readFloatBE(24),
    Akku_Power: msg.payload.data.readFloatBE(28),
    };
}
return msg;
```

Spannungsversorgung eingefügt, der über den Jack Eingang mit dem Arduino verbunden ist. Dieser ist auf die Versorgungsspannung 9V eingestellt. (Anmerkung: Der Arduino kann theoretisch mit den 12V des Akkus auch ohne LM2596 versorgt werden)

2.3 Raspberry Pi - Empfängerknoten

Als Empfänger wird ein Raspberry Pi (3B+) verwendet. Das Betriebssystem ist dabei Raspian Buster lite in der der neuesten Version. An dem RaspberryPi ist der XBee Coordinator über eine Serielle Schnittstelle (RS232) angeschlossen ist. Die Schnittstelle wird über eine Erweiterungsplatine realisiert.

2.3.1 Einrichten der Seriellen Schnittstelle

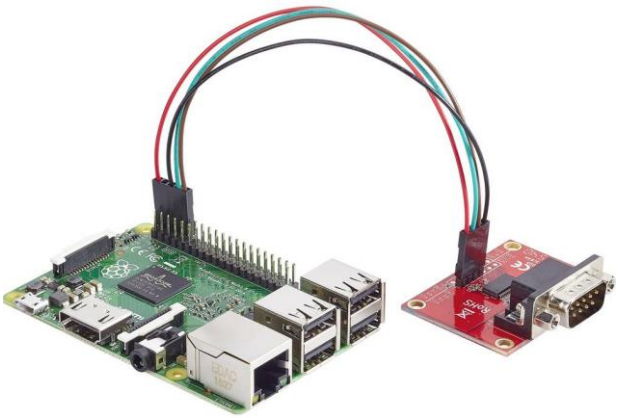

Abbildung 9: Raspberry Pi mit RS232 Erweiterung

Auf dem Raspberry muss die Serielle Schnittstelle erst eingerichtet und konfiguriert werden. Hierzu über den Befehl „sudo raspi-config“ das Konfigurationsmenü aufrufen und in den Interfacing Options → Serial zuerst bei der Frage nach dem „loginShell = Nein“ und bei der Frage nach Verwendung der „serial port hardware = Yes“ einstellen. Anschließend muss noch das vorhandene Bluetooth Modul deaktiviert und die UART Schnittstelle eingeschalten werden. Hierzu über den Befehl „sudo nano /boot/config.txt“ die Konfigurationsdatei bearbeiten. Am Ende der Datei muss die folgende Zeile eingefügt werden.

```
dtoverlay=pi3-miniuart-bt
```

An die RS232 Schnittstelle wird das XBee Explorer Serial mit XBee Modul per D-Sub Kabel angeschlossen.

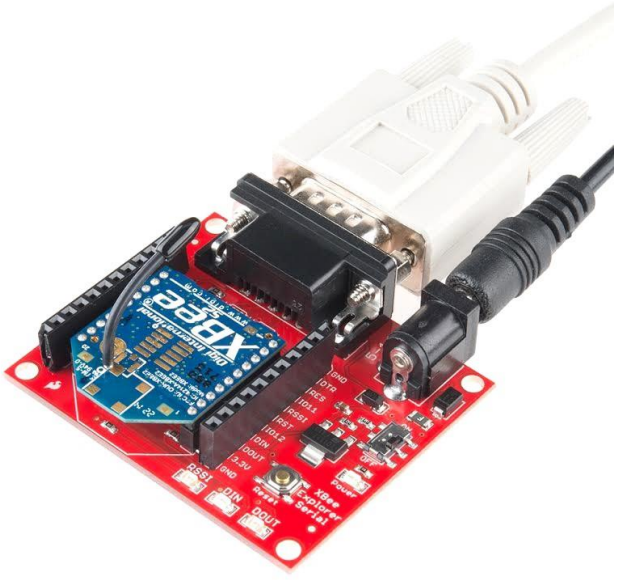

Abbildung 10: XBee Explorer Serial Entwicklungsplatine

2.3.2 Konfiguration der Node-RED Knoten

Die Abfrage der Seriellen Schnittstelle und das Einschreiben der Messdaten in die Datenbank ist über Node-RED realisiert. Als Datenbank dient eine InfluxDB die ebenfalls auf dem RaPi läuft.

In folgenden wird die Konfiguration der NodeRed Knoten erklärt.

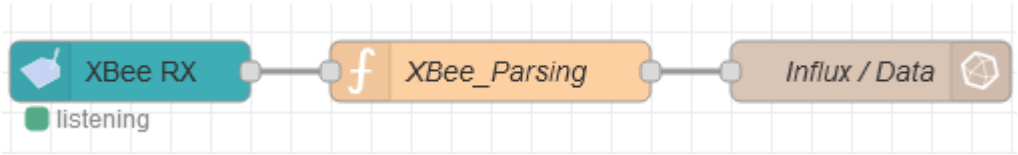

Abbildung 11: Darstellung XBee Parsing zur Datenbank

Es werden 3 Knoten benötigt:

- XBee RX:
  Dieser Knoten beobachtet die Serielle Schnittstelle auf eingehende XBee Nachrichten.
  Dieser Knoten entstammt aus der Palette „node-red-contrib-xbee“
- XBee_Parsing:
  Diese Funktion entschlüsselt die Payload Daten und verknüpft diese mit den Variablen für das Einspeichern in die Datenbank
- Influx / Data:
  Anbindung an die Datenbank

Die Konfiguration des XBee Eingangknoten erfolgt nach dem folgenden Bild.

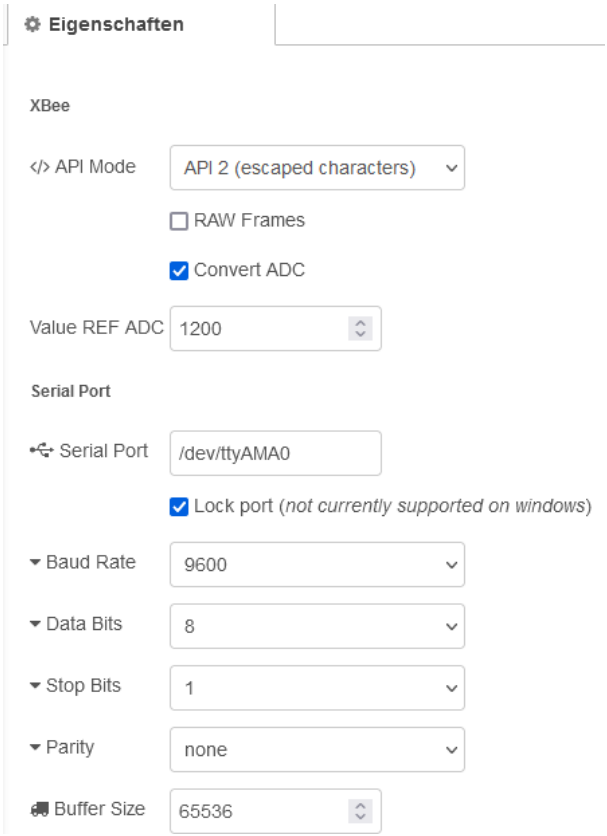

Abbildung 12: Einstellungen des XBee RX Knoten

Der Knoten erzeugt bei eingehenden XBee Frame (Erkennung des Startwertes 0x7E) und gibt folgende Nachricht weiter.

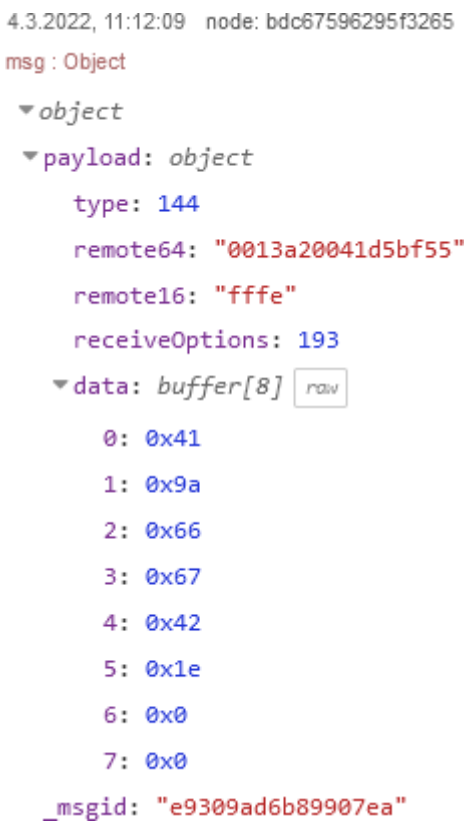

Abbildung 13: Beispielnachricht (JSON) aus dem XBee RX Knoten

In dieser Anwendung werden die Daten auf dem Arduino in ein UInt8-Array nach folgendem Schema übergeben.

0013a20041d5bf55				
BME280 + INA219				
32				
	Datatyp	Measurement	Unit	Influx
0				
1				
2				
3	float	Temperature	[°C]	Temperature02
4				
5				
6				
7	float	Humidity	[%]	Humidity02
8				
9				
10				
11	float	Pressure	[hPa]	Pressure02
12				
13				
14				
15	float	Shuntvoltage	[mV]	Shuntvoltage
16				
17				
18				
19	float	Busvoltage	[V]	Busvoltage
20				
21				
22				
23	float	Current	[mA]	Current
24				
25				
26				
27	float	Loadvoltage	[V]	Loadvoltage
28				
29				
30				
31	float	Power	[mW]	Akkuleistung

Abbildung 14: Datenstruktur der Payload

Die Konfiguration des Parsing Knoten erfolgt nach der folgenden Abbildung.

```
var Source = msg.payload.remote64;
msg.measurement ="home";
// Mapping für BME280
if (Source == "0013a20041d5c40a"){
    msg.payload = {
    Temperature01: msg.payload.data.readFloatBE(0),
    Humidity01:  msg.payload.data.readFloatBE(4),
    Pressure01: msg.payload.data.readFloatBE(8),
    };
}
// Mapping für BME280+INA219
if (Source == "0013a20041d5bf55"){
    msg.payload = {
    Temperature02: msg.payload.data.readFloatBE(0),
    Humidity02:  msg.payload.data.readFloatBE(4),
    Pressure02: msg.payload.data.readFloatBE(8),
    Shuntvoltage: msg.payload.data.readFloatBE(12),
    Busvoltage: msg.payload.data.readFloatBE(16),
    Current: msg.payload.data.readFloatBE(20),
    Loadvoltage: msg.payload.data.readFloatBE(24),
    Akku_Power: msg.payload.data.readFloatBE(28),
    };
}
return msg;
```

Abbildung 15: Konfiguration der Parsingfunktion

Durch die IF Bedingungen werden die Nachrichten nach der MAC Adresse der Sender XBee-Module sortiert und entsprechend der Register übersetzt.

Der Ausgangsknoten für die Datenbank bedarf ausschließlich Standardeinstellungen.

3 Anhang

3.1 Arduino Code

3.1.1 Code für BME280+INA219

```
/**************************************************************************
*
 Programm zum Auslesen eines BME280 Sensors und senden der Daten über XBEE
 Zusätzlich Auslesen des INA219 zur Überwachung der Batteriespannung
 Markus Hartmann
 27.06.2022

 BME wird über I2C ausgelesen --> Wenn andere Pins verwendet werden, muss
das Programm angepasst werden

**************************************************************************
/

#include <Wire.h>
#include <Adafruit_INA219.h>
#include <SPI.h>
#include <XBee.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
// Instantierung ina219
Adafruit_INA219 ina219;
// Instantierung BME280
#define SEALEVELPRESSURE_HPA (1013.25)
Adafruit_BME280 bme; // I2C
// Create XBee object
XBee xbee = XBee();
// Initialize payload (für BME 12 bytes. für INA219 alle Werte 20)
uint8_t ui8Payload[32] = {0x00,0x00,0x00,0x00,
                          0x00,0x00,0x00,0x00,
                          0x00,0x00,0x00,0x00,
                          0x00,0x00,0x00,0x00,
                          0x00,0x00,0x00,0x00,
                          0x00,0x00,0x00,0x00,
                          0x00,0x00,0x00,0x00,
                          0x00,0x00,0x00,0x00};

// Adresse Coordinator
XBeeAddress64 addr64 = XBeeAddress64(0x0013a200, 0x41d5c29a);
//setup xbee request
ZBTxRequest zbTx = ZBTxRequest(addr64, ui8Payload, sizeof(ui8Payload));
// define Zykluszeit [ms]
int delayTime = 60000; // Messzyklus 1 Minute

void setup(void)
{
Serial1.begin(9600); // Arduino Zero → Serial1 !
xbee.setSerial(Serial1);
delay(1000);  //delay for safety reasons
```

```
bme.begin();
delay(1000);  //delay for safety reasons
ina219.begin();
delay(1000);  //delay for safety reasons
ina219.setCalibration_16V_400mA(); // calibration of INA219
delay(1000);  //delay for safety reasons
}

void loop(void)
{ //reading INA219
  //read Shuntvoltage [mV]
  float fShuntvoltage = 0;
  uint32_t ui32Shuntvoltage =0;
  fShuntvoltage = ina219.getShuntVoltage_mV();
  ui32Shuntvoltage = (reinterpret_cast<uint32_t&>(fShuntvoltage));
  //read Busvoltage [V]
  float fBusvoltage = 0;
  uint32_t ui32Busvoltage =0;
  fBusvoltage = ina219.getBusVoltage_V();
  ui32Busvoltage = (reinterpret_cast<uint32_t&>(fBusvoltage));
  //read Current [mA]
  float fCurrent_mA = 0;
  uint32_t ui32Current =0;
  fCurrent_mA = ina219.getCurrent_mA();
  ui32Current = (reinterpret_cast<uint32_t&>(fCurrent_mA));
  //Calculate Loadvoltage [V]
  float fLoadvoltage = 0;
  uint32_t ui32Loadvoltage = 0;
  fLoadvoltage = fBusvoltage + (fShuntvoltage / 1000);
  ui32Loadvoltage = (reinterpret_cast<uint32_t&>(fLoadvoltage));
  // read power [mW]
  float fPower_mW = 0;
  uint8_t ui32Power = 0;
  fPower_mW = ina219.getPower_mW();
  ui32Power = (reinterpret_cast<uint32_t&>(fPower_mW));
  //read BME280
  //read Temperature [°C]
  uint32_t ui32Temperature = 0;
  float fTemperature = bme.readTemperature();
  ui32Temperature = (reinterpret_cast<uint32_t&>(fTemperature));
  //read Humidity [%]
  uint32_t ui32Humidity = 0;
  float fHumidity = bme.readHumidity();
  ui32Humidity = (reinterpret_cast<uint32_t&>(fHumidity));
  //read Pressure
  uint32_t ui32Pressure = 0;
  float fPressure = bme.readPressure()/100.0F;
  ui32Pressure = (reinterpret_cast<uint32_t&>(fPressure));
  // Mapping to Payload
  //Temperature
  ui8Payload[0] = (byte)( (ui32Temperature >>24) &0x000000FF );
  ui8Payload[1] = (byte)( (ui32Temperature >>16) &0x000000FF );
  ui8Payload[2] = (byte)( (ui32Temperature >>8)  &0x000000FF );
  ui8Payload[3] = (byte)( ui32Temperature        &0x000000FF );
  //Humidity
  ui8Payload[4] = (byte)( (ui32Humidity >>24) &0x000000FF );
  ui8Payload[5] = (byte)( (ui32Humidity >>16) &0x000000FF );
  ui8Payload[6] = (byte)( (ui32Humidity >>8)  &0x000000FF );
  ui8Payload[7] = (byte)( ui32Humidity        &0x000000FF );
```

```
  //Pressure
  ui8Payload[8] = (byte)( (ui32Pressure >>24) &0x000000FF );
  ui8Payload[9] = (byte)( (ui32Pressure >>16) &0x000000FF );
  ui8Payload[10] = (byte)( (ui32Pressure >>8)  &0x000000FF );
  ui8Payload[11] = (byte)( ui32Pressure        &0x000000FF );
  //Shuntvoltage
  ui8Payload[12] = (byte)( (ui32Shuntvoltage >>24) &0x000000FF );
  ui8Payload[13] = (byte)( (ui32Shuntvoltage >>16) &0x000000FF );
  ui8Payload[14] = (byte)( (ui32Shuntvoltage >>8)  &0x000000FF );
  ui8Payload[15] = (byte)( ui32Shuntvoltage        &0x000000FF );
  //Busvoltage
  ui8Payload[16] = (byte)( (ui32Busvoltage >>24) &0x000000FF );
  ui8Payload[17] = (byte)( (ui32Busvoltage >>16) &0x000000FF );
  ui8Payload[18] = (byte)( (ui32Busvoltage >>8)  &0x000000FF );
  ui8Payload[19] = (byte)( ui32Busvoltage        &0x000000FF );
  // Current
  ui8Payload[20] = (byte)( (ui32Current >>24) &0x000000FF );
  ui8Payload[21] = (byte)( (ui32Current >>16) &0x000000FF );
  ui8Payload[22] = (byte)( (ui32Current >>8)  &0x000000FF );
  ui8Payload[23] = (byte)( ui32Current        &0x000000FF );
  //Loadvoltage
  ui8Payload[24] = (byte)( (ui32Loadvoltage >>24) &0x000000FF );
  ui8Payload[25] = (byte)( (ui32Loadvoltage >>16) &0x000000FF );
  ui8Payload[26] = (byte)( (ui32Loadvoltage >>8)  &0x000000FF );
  ui8Payload[27] = (byte)( ui32Loadvoltage        &0x000000FF );
  //Power
  ui8Payload[28] = (byte)( (ui32Power >>24) &0x000000FF );
  ui8Payload[29] = (byte)( (ui32Power >>16) &0x000000FF );
  ui8Payload[30] = (byte)( (ui32Power >>8)  &0x000000FF );
  ui8Payload[31] = (byte)( ui32Power        &0x000000FF );
  // Sending Payload XBee
  xbee.send(zbTx);
  delay(delayTime);
}
```