Gefördert durch:

Bundesministerium
für Wirtschaft
und Klimaschutz

aufgrund eines Beschlusses
des Deutschen Bundestages

Technische
Hochschule
Rosenheim

MonSec

XBee – Setup (API-Mode)

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 mit zwei verschiedenen Sensoren (Bosch BME280 sowie DHT22) realisiert. Als Empfänger 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.

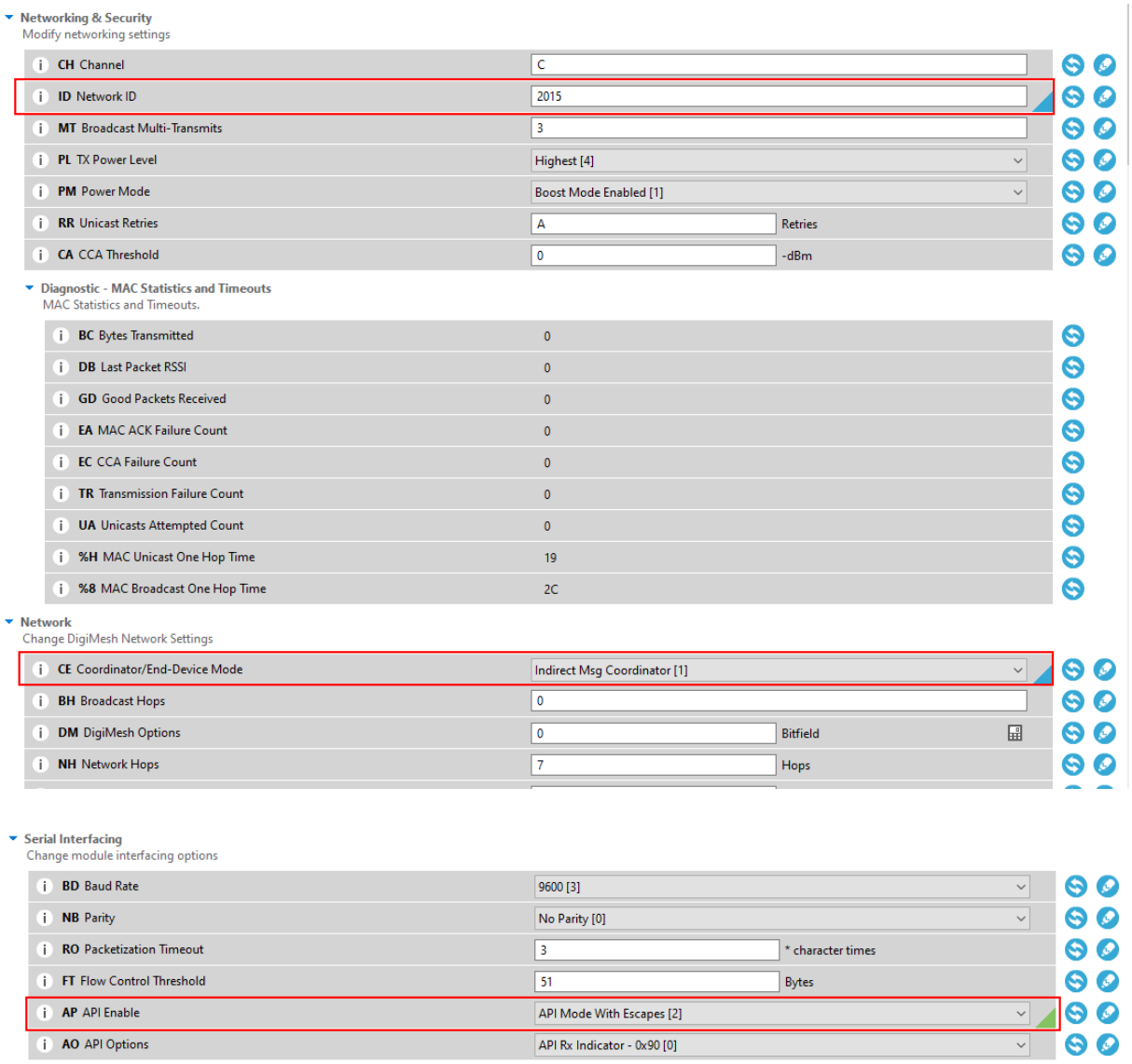

Abbildung 1: Parameter des Coordinators

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.

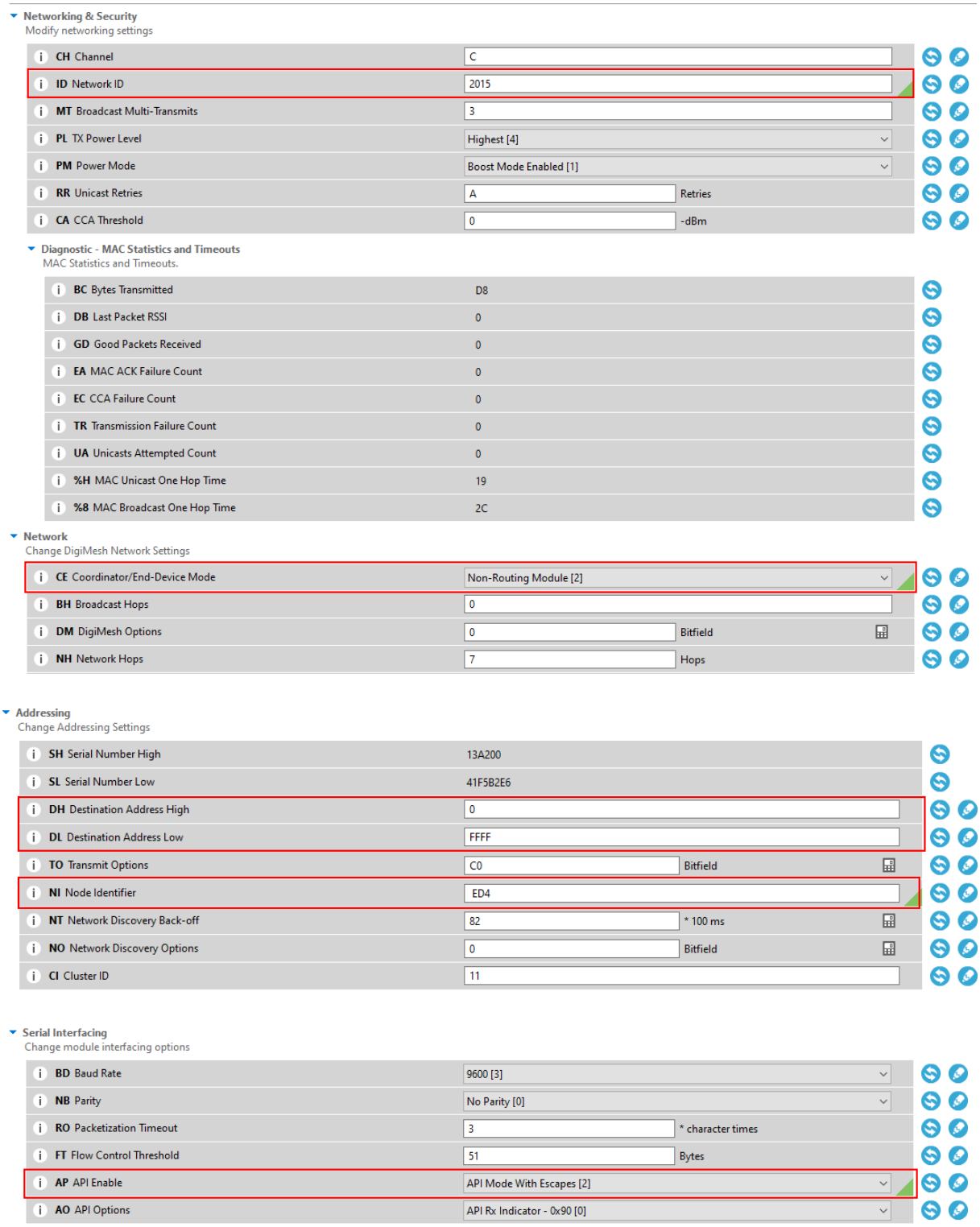

Abbildung 2: Konfiguration der Endknoten

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

2.2 Arduino

Für dieses Projekt wird ein Arduino Uno verwendet.

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

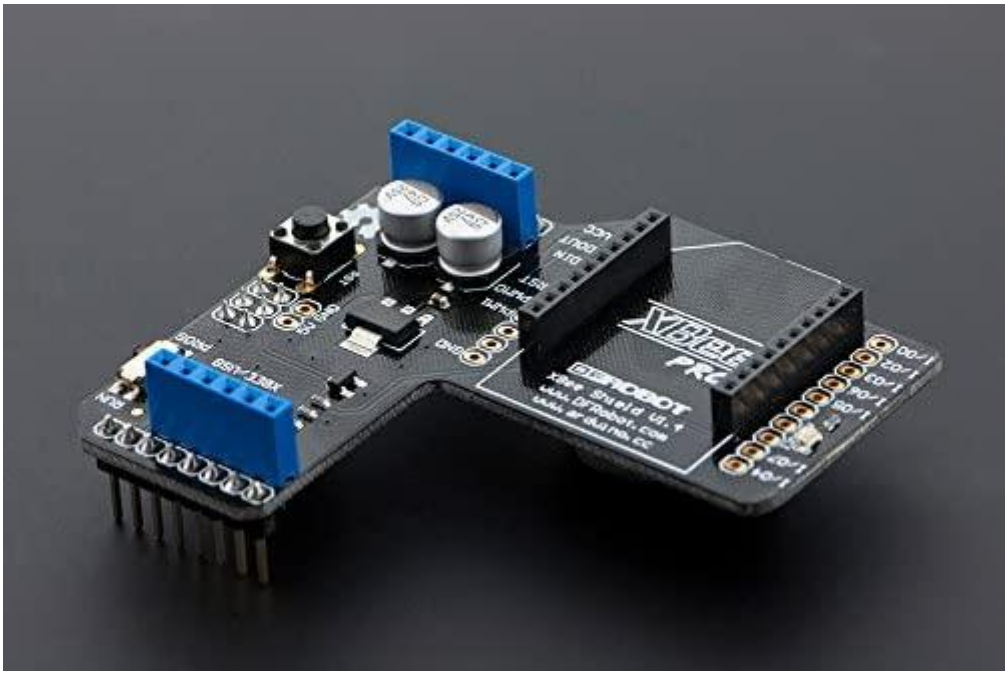

Abbildung 3: XBee Shield für Arduino [DFRobot]

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

2.2.1 Sensor – BME280

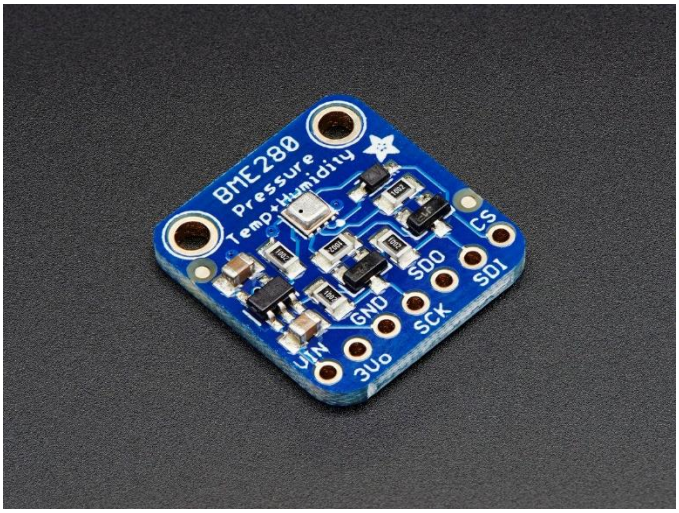

Abbildung 4: BME280 [Adafruit]

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:

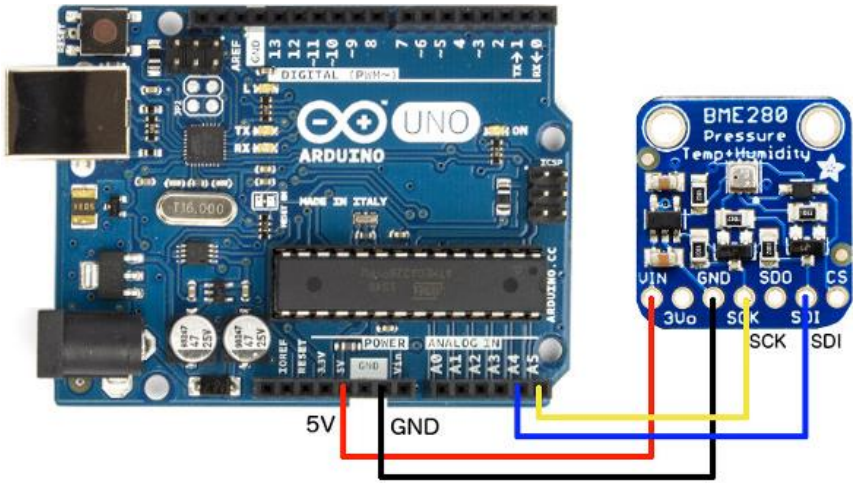

Abbildung 5: Anschlussschema BME280 an Arduino [http://cactus.io/]

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 – DHT22

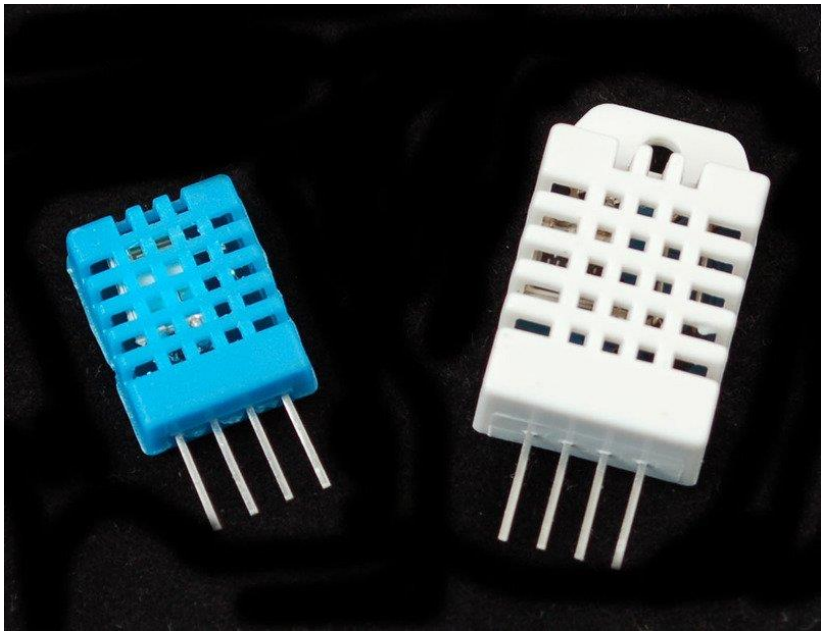

Abbildung 6: DHT11 und DHT22 [Adafruit]

Sowohl der DHT11 als auch der DHT22 sind Sensoren für Temperatur und Luftfeuchte. Die Genauigkeit des DHT22 ist jedoch höher. Die Sensoren sind vom Anschluss her gleich und verwenden den OneWire Bus. Sie werden nach folgendem Schema an den Arduino angeschlossen.

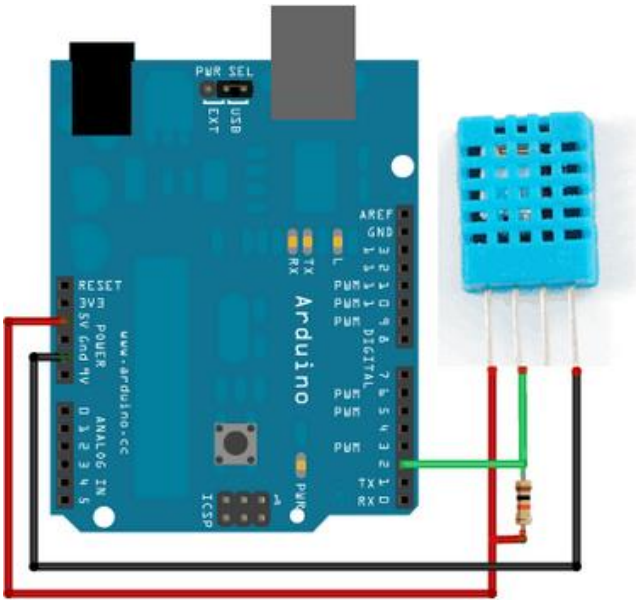

Abbildung 7: Anschlussschema DHT22 [Adafruit]

Wie in Abbildung 7 zu erkennen muss ein Widerstand (Pullup, 10 KΩ) zwischen die Datenleitung und der Spannungsversorgung eingebaut werden (Sofern noch nicht vorhanden).

Die Bibliothek ist direkt in der Arduino IDE unter dem Stichwort DHT zu finden. Bei neuen Versionen muss noch die Adafruit Unified Sensor Bibliothek zusätzlich eingebunden werden.

Der Code auf dem Arduino ist im Anhang (Kapitel 3.1.2) zu finden.

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

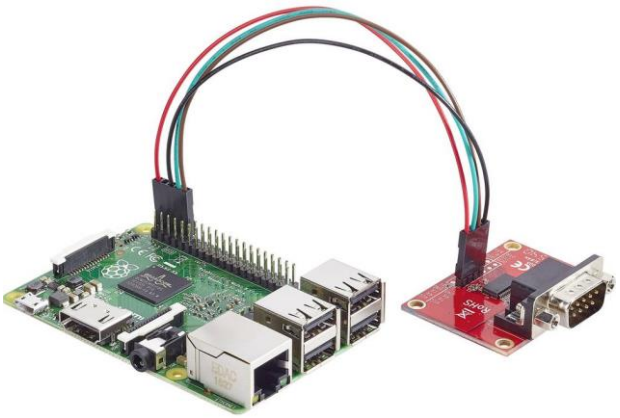

Abbildung 8: Raspberry Pi mit RS232 Erweiterung

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.

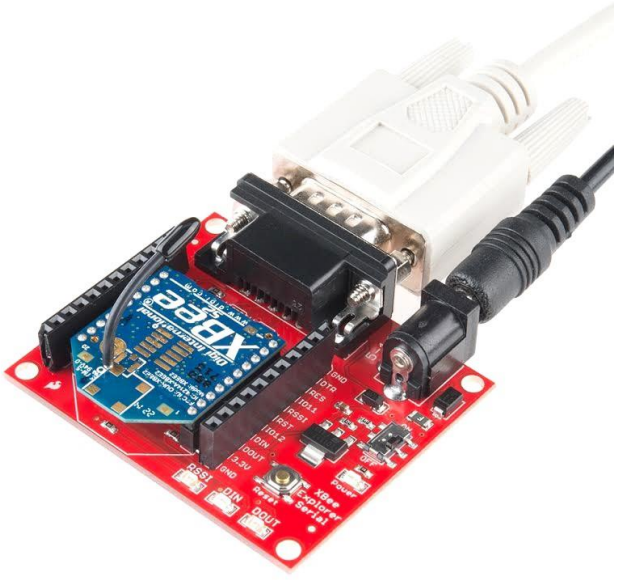

Abbildung 9: XBee Explorer Serial Entwicklungsplatine

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.

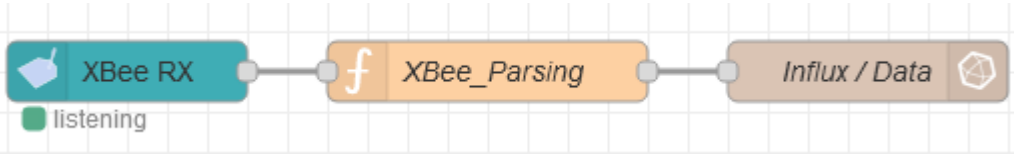

Abbildung 10: Darstellung XBee Parsing zur Datenbank

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.

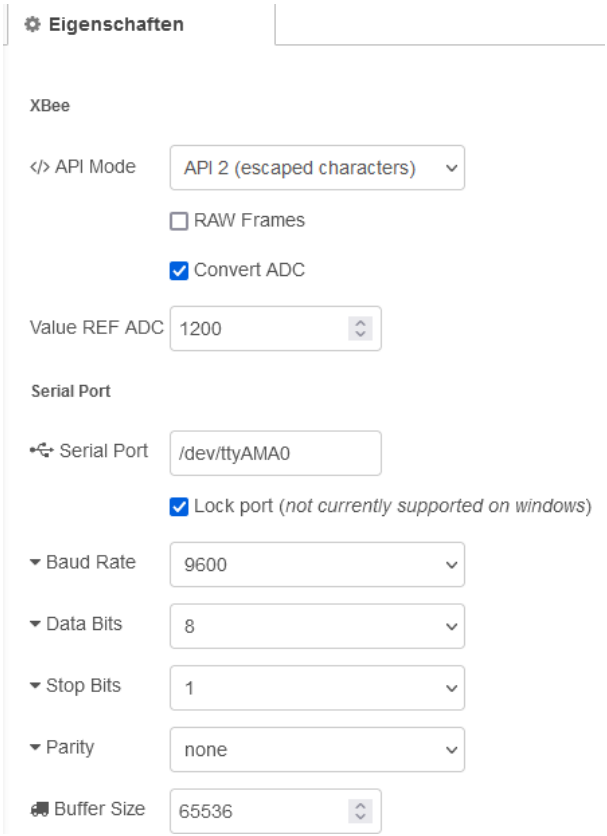

Abbildung 11: Einstellungen des XBee RX Knoten

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

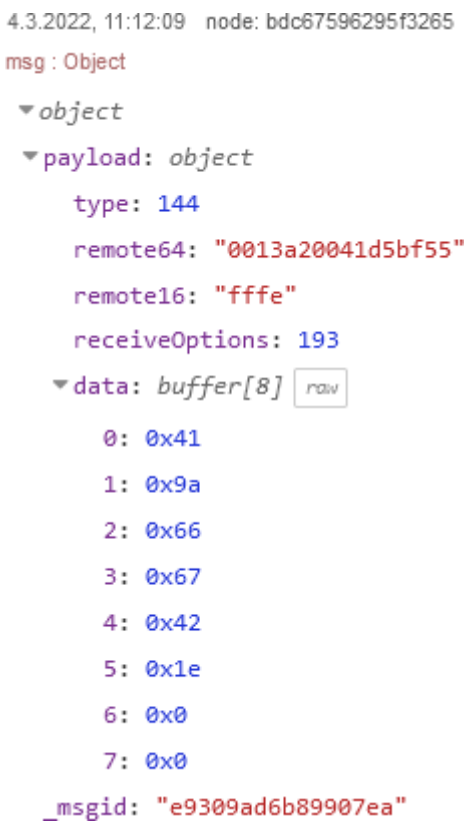

Abbildung 12: Beispielnachricht (JSON) aus dem XBee RX Knoten

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

MacAddress XBee:	0013a20041d5c40a				0013a20041d5bf55				
Sensor:	BME280				DHT22				
length payload	12				8				
Payload	Datatyp	Measurement	Unit	Influx		Datatyp	Measurement	Unit	Influx
0					0				
1					1				
2					2				
3	float	Temperature	[°C]	Temperature01	3	float	Temperature	[°C]	Temperature02
4					4				
5					5				
6					6				
7	float	Humidity	[%]	Humidity01	7	float	Humidity	[%]	Humidity02
8									
9									
10									
11	float	Pressure	[hPa]	Pressure01					

Abbildung 13: 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 DHT22
if (Source == "0013a20041d5bf55"){
    msg.payload = {
    Temperature02: msg.payload.data.readFloatBE(0),
    Humidity02:  msg.payload.data.readFloatBE(4),
    };
}
return msg;
```

Abbildung 14: Konfiguration der Parsingfunktion

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

Der Ausgangsknoten für die Datenbank bedarf ausschließlich Standardeinstellungen.

3 Anhang

3.1 Arduino Code

3.1.1 Code für BME280

```
/**************************************************************************
*
 Programm zum Auslesen eines BME280 Sensors und senden der Daten über XBEE

 Markus Hartmann
 24.02.2022

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

**************************************************************************
/
#include <Wire.h>
#include <SPI.h>
#include <XBee.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#define SEALEVELPRESSURE_HPA (1013.25)
// initialize BME280
Adafruit_BME280 bme; // I2C
// Create XBee object
XBee xbee = XBee();
// Initialize payload (für BME 12 bytes)
uint8_t ui8Payload[12] =
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
//setup xbee request
// Adresse Coordinator
XBeeAddress64 addr64 = XBeeAddress64(0x0013a200, 0x41d5c29a);
ZBTxRequest zbTx = ZBTxRequest(addr64, ui8Payload, sizeof(ui8Payload));
// Intervall für die Messungen
int delayTime = 6000;

void setup() {
    Serial.begin(9600);
    xbee.setSerial(Serial);
    delay(2000);
    bme.begin();
    delay(2000);
}

void loop() {
  //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 );
  xbee.send(zbTx);
  delay(delayTime);
}
```

3.1.2 Code für DHT22

```
/*************************************************************************
*
 Programm zum Auslesen eines DHT22 Sensors und senden der Daten über XBEE

 Markus Hartmann
 24.02.2022
*************************************************************************
/
#include <XBee.h>
#include "DHT.h"

#define DHTPIN 2       // Digital pin connected to the DHT sensor
#define DHTTYPE DHT22  // DHT 22  (AM2302), AM2321

// Initialize DHT sensor.
DHT dht(DHTPIN, DHTTYPE);
// Create an XBee object at the top of your sketch
XBee xbee = XBee();
// Initialize payload (für DHT22 8 bytes)
uint8_t ui8Payload[8] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
//setup xbee request
XBeeAddress64 addr64 = XBeeAddress64(0x0013a200, 0x41d5c29a);
ZBTxRequest zbTx = ZBTxRequest(addr64, ui8Payload, sizeof(ui8Payload));

// Intervall für die Messungen
int delayTime = 6000;

void setup() {
```

```
  Serial.begin(9600);
  xbee.setSerial(Serial);
  delay(1000);
 dht.begin();
  delay(1000);
}

void loop() {
  //read Temperature [°C]
  uint32_t ui32Temperature = 0;
  float fTemperature = dht.readTemperature();
  ui32Temperature = (reinterpret_cast<uint32_t&>(fTemperature));
  //read Humidity [%]
  uint32_t ui32Humidity = 0;
  float fHumidity = dht.readHumidity();
  ui32Humidity = (reinterpret_cast<uint32_t&>(fHumidity));

  // 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 );

  xbee.send(zbTx);
  delay(delayTime);
}
```