Arduino-basierte Auswertung von Messdaten und Data-Logging für den Feinstaubsensor SDS011 von Nova Fitness

Bernd Laquai, Update 5.01.2017

Unter den günstigen Feinstaub Sensoren, die derzeit auf dem chinesischen Markt angeboten werden taucht auch ein besonders günstiger auf, der SDS011 der Firma Nova Fitness. Er ist offensichtlich für Klimaanlagen entwickelt worden, das könnte man aus dem Schlauchstutzen schließen, der als Luftansaug-Öffnung dient. Er kostet um die 20 Dollar und ist bei etlichen Händlern der chinesischen Internet Market Plattform Ali-Express zu haben. Oft wir er mit einem USB nach Seriell Konverter geliefert. Meist fehlt jedoch ein Hinweis auf einen passenden Windows Treiber. Da bei den chinesischen USB2TTL-Konvertern (oder auch COM-Port Konverter genannt) meist nicht der Marktführer FTDI, den auch die Arduino Boards verwenden, sondern der chinesische Counterpart CH340 oder CH341 des chinesischen Herstellers Winchiphead / Nanjing QinHeng Electronics Co., Ltd (http://www.wch.cn/), wo man auch die passenden Treiber findet. Um die Daten des Sensors zu ordentlich anzuzeigen, nützt das unter Windows leider zunächst wenig, da sie ja in binärer Form angeliefert werden und nicht als einfach lesbarer ASCII Text.

Der SDS011 Sensor arbeitet ebenfalls mit einem Halbleiterlaser nach dem Streulichtverfahren. Er liefert aber nur die Werte für die PM10 und PM2.5 Massenkonzentrationen in ug/m^3 und nicht wie der HK-A5 Sensor von DfRobot oder der PMS1003 von Plantower die Partikelzählergebnisse. Dafür aber hat der SDS011 aber auch analoge PWM-Ausgänge, mit denen das Messergebnis für die PM10 und PM2.5 Massenkonzentration über das Tastverhältnis eines Rechtecksignals angezeigt wird. Dieses Signal lässt sich nach entsprechender Tiefpassfilterung auch mit einem analogen Eingang eines Mikrocontrollers auslesen, oder aber hochohmig mit einem Messverstärker oder Multimeter messen. Die Auswertung der digitalen Daten ist jedoch deutlich sicherer, wenn man sie mal softwaretechnisch im Griff hat als die Auswertung der analogen Signale.

Unter dem Linux Betriebssystem ist die schnelle Auswertung der Binärdaten, die an einem USB-Port angeliefert einfacher zu bewerkstelligen als unter Windows. Es gibt sehr kompetente Skript-Befehle, mit denen man ruck zuck eine Umformung zusammen-hacken kann, vorausgesetzt man kennt sich mit diesen Tools gut aus. Zudem braucht man auch keine Treiber installieren, denn Linux (z.B. Ubuntu) dockt so einen Konverter automatisch unter „/dev/ttyUSB0“ an. Für die Linux-Fans sei hier im Anhang mal so ein „Quick-Hack“ (modifiziert aus /1/) angefügt. Unter manchen Linux Varianten unterscheiden sich die Befehlsoptionen der Skriptbefehle etwas, vor allem beim Befehl od (octal dump) gibt es leider nicht immer die bequeme option „—endian-big“ um die Reihenfolge von Low Byte und High Byte je nach Prozessor umzudrehen, deswegen ist es sinnvoller hierzu auf den in den core utilities des GNU-Projekts in der Regel immer vorhandenen Befehl dd (disk dump) zurückzugreifen.

Um den Sensor nun in Betrieb zu nehmen reicht es, den USB2TTL Konverter in den USB-Port zu stecken und das Skript mit dem superuser Befehl: sudo ./sds011_loop zu starten, wenn sds011_loop der Name des Skriptes ist. Diese Version des Skripts läuft dann auch unter anderen Linux Varianten wie z.B. Redhat RHEL 7 problemlos und gibt die Messwerte des Sensors in einer Endlos-Schleife auf dem Bildschirm aus.

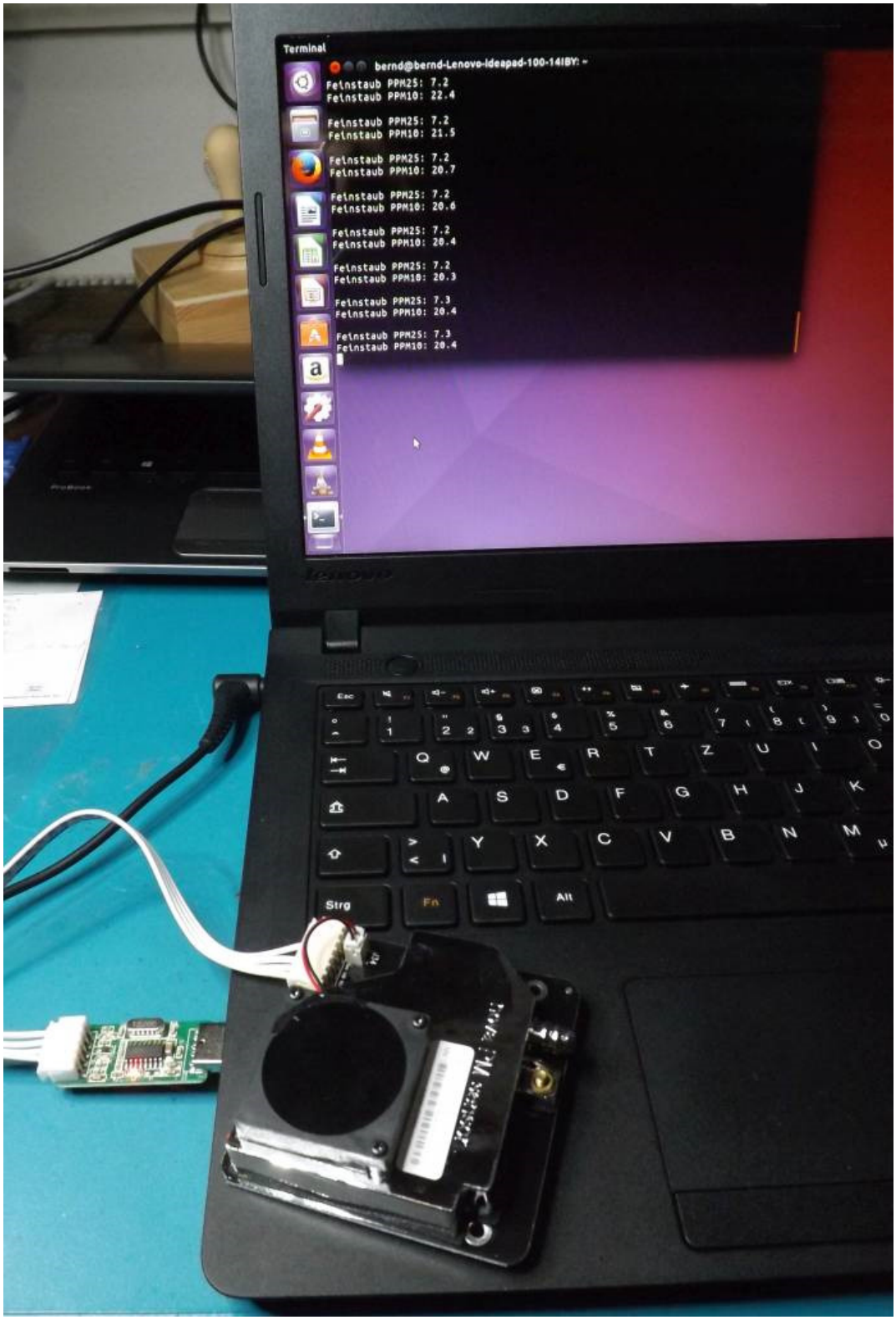

Abb. 1: Auslesen des Feinstaubsensors mit USB2TTL Konverter unter Linux (Ubuntu)

Sehr viel mehr Sinn macht es aber, gleich einen Mikrocontroller über die IDE mit der Aufgabe der Dateninterpretation zu betrauen. Hierzu braucht man auch nicht gleich einen Raspi, der Arduino Uno oder ein entsprechender Clone tut es auch. Diese Auswertung lässt sich später dann auch leicht mit einem Data-Logging auf einem SD-Card Shield zu einem autonomen System kombinieren, wo man dann für die eigentliche Messung keinen PC-basierten Rechner mehr benötigt.

Wie beim HK-A5 Sensor von DfRobot oder dem PMS1003 von Plantower ist der wesentliche Aspekt für das Auswerteprogramm die Synchronisation zum Datenstrom des Sensors. Die Daten werden auch hier nach einem bestimmten Protokoll gesendet, das eine Framestruktur erzeugt. Ein Frame beginnt immer mit dem Byte 0xAA gefolgt von 0xC0. Der SDS011 hat aber im Gegensatz zum HK-A5 oder PMS1003 eine Kennzeichnung des Frame-Endes mit dem Byte 0xAB. Im vorletzten Byte wird eine Quersumme aus den 6 Daten-Bytes, gebildet, allerdings nur Modulo 256, d.h. der Überlauf aus der Summenbildung wird einfach ignoriert. Da nur 6 Datenbytes gesendet werden, sind die Frames allerdings deutlich kürzer. Man so aber auch die korrekte Übertragung und Auswertung der Framestruktur überprüfen kann. Auf diese Framestruktur kann daher auch synchronisiert werden, wenn man auf die fehlerfreie Auswertung prüft.

Die Synchronisation wird wieder so gemacht, dass zunächst mit incomingByte = Serial.read() byteweise gelesen wird, was gerade auf der seriellen Schnittstelle des Arduino eintrifft. Wenn das Byte 0xAA im Datenstrom auftaucht, geht man davon aus, dass dies einen Frame-Beginn kennzeichnet, obwohl es ja auch ein Datenbyte sein könnte. Dann liest man die nächsten 9 Bytes in den Framepuffer der Länge 9 ein und testet an der Stelle Null des Pufferspeichers ob hier jetzt das zweite Framestart-Byte 0xC0 steht und ob an der letzten Stelle das Frame-Ende mit 0xAB auftaucht. Ist das der Fall, geht man davon aus, das man den Frame richtig erwischt hat.

Von zwei Bytes, die einen Wert darstellen, wird immer das höherwertige Byte zuerst gesendet und dann das niederwertige. Um das höherwertige und das niederwertige Byte, das man aus einem Pufferspeicher liest, zu einer Zahl zusammen zu setzen, kann man sehr vorteilhaft vom Links-Schiebe-Befehl „<<“ Gebrauch machen. So schiebt z.B. der Befehl x = (buf[i]<<8) die Bits des Byte an der Stelle i in dem unsigned char Pufferspeicher buf um 8 Bits nach links (was einer Multiplikation mit 256 entspricht). Um nun die passende Integer Zahl x für einen zwei-Byte Wert zu erhalten, muss man das niederwertige Byte, das beim SDS011 zuerst gesendet wird, ohne es zu verschieben, zum verschobenen höherwertigen Byte dazu zählen, also x = ((buf[i+1]<<8) + buf(i)). Die Klammern sind zwingend nötig, sonst hat der Compiler mit der Interpretation der Verschieboperation Probleme. Diese Methode der Auswertung wird nur bei den Daten die durch 2 Bytes dargestellt sind, verwendet. Danach prüft man den Pufferspeicher-Inhalt auf die Quersumme, und wenn diese stimmt, dann werden die Daten als korrekt angesehen, ansonsten wird der Pufferspeicher-Inhalt verworfen und eine entsprechende Fehlermeldung ausgegeben, bevor der nächste Synchronisationsversuch beginnt.

Gelingt die Synchronisation, werden die Werte der beiden PM-Konzentrationen in ug/m^3 als Zahlen zusammengesetzt und die 2 Anzeigestrings gebildet und auf dem Bildschirm ausgegeben. Beachtet werden muss noch, dass die Werte vom Sensor mit einem Faktor 10 größer ausgegeben werden, also noch durch 10 geteilt werden müssen. Wenn keine Übertragungsfehler auftreten, dann bleibt der Lesevorgang beim nächsten Schleifen-Durchlauf in Sync, d.h. die Vergleiche liefern sofort logisch wahr und die Anzeige wird so im Sekunden-Takt aktualisiert.

Dieses in Listing 1 gezeigte Arduino-Programm lässt sich nun ganz einfach so erweitern, dass die Daten auch auf einem Data-Logging Shield (z.B. von Adafruit) mit dem Zeitstempel einer Real Time Clock

abgespeichert werden. Zudem kann noch eine leichte Filterung vorgesehen werden, wenn beispielsweise nur die atmosphärischen Feinstaubdaten aufgezeichnet werden sollen, denn im Sekundentakt macht eine Aufzeichnung kaum Sinn.

Für die Mittelung bietet sich die Berechnung eines gleitenden Mittelwerts über ein Fenster von N Messwerten an, wobei aber nur jeder N-te Mittelwert auf SD-Karte abgespeichert wird. Da die PM10 und P2_5 Werte stark korreliert sind, und der PM2.5 wegen dem stärkeren Einfluss auf die Gesundheit der Wichtigere ist, kann man auch Speicherplatz dadurch sparen, dass man nur den PM2.5 Wert auswertet und speichert.

Für die Verwendung der Real Time Clock steht eine Bibliothek des Herstellers zur Verfügung und die Bibliothek für die SD-Karten-Funktionen ist bereits in der Arduino IDE enthalten. Der Vorteil des Adafruit Data-Logging Shields ist, dass beides, der Speicherkartenhalter und die Ansteuerung, wie die Real-Time-Clock auf dem Shield enthalten sind.

Da der Einfachheit halber das Hardware-Serial Interface an Tx und Rx des Arduino für die Übertragung verwendet wurde (Pin 0 und 1 am Arduino) darf die Verbindung zum SDS011 erst hergestellt werden, wenn das Hochladen des Programms abgeschlossen ist, sonst kommt es zu Fehlern beim Hochladen. Aus diesem Grund ist es sinnvoll am SD Card Shield eine steckbare Verbindung anzubringen.

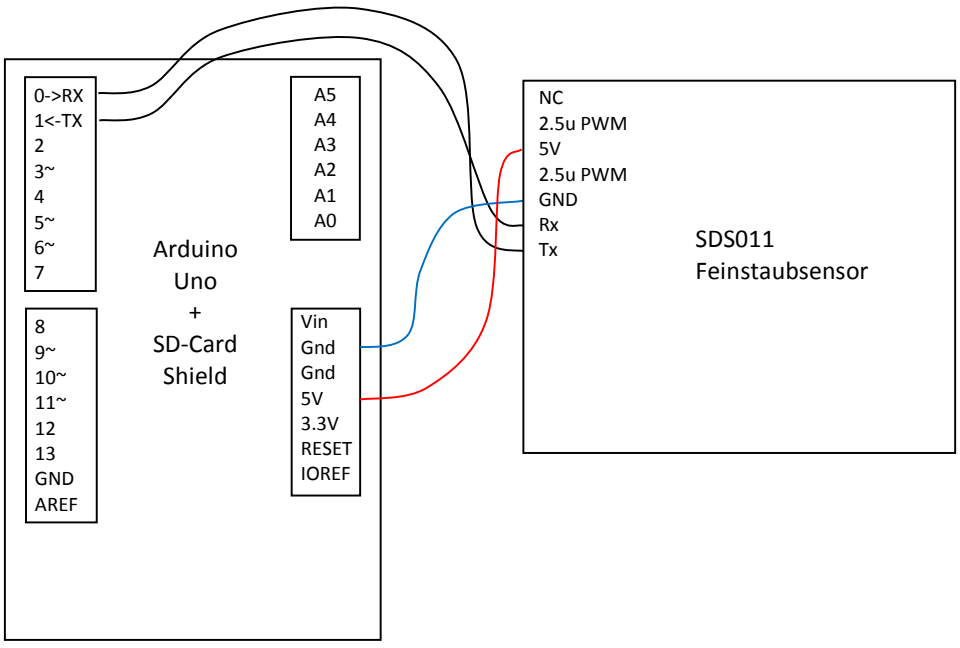

Abb. 2: Verdrahtung des Arduino + SD-Card Shield mit dem SDS011 Feinstaubsensor

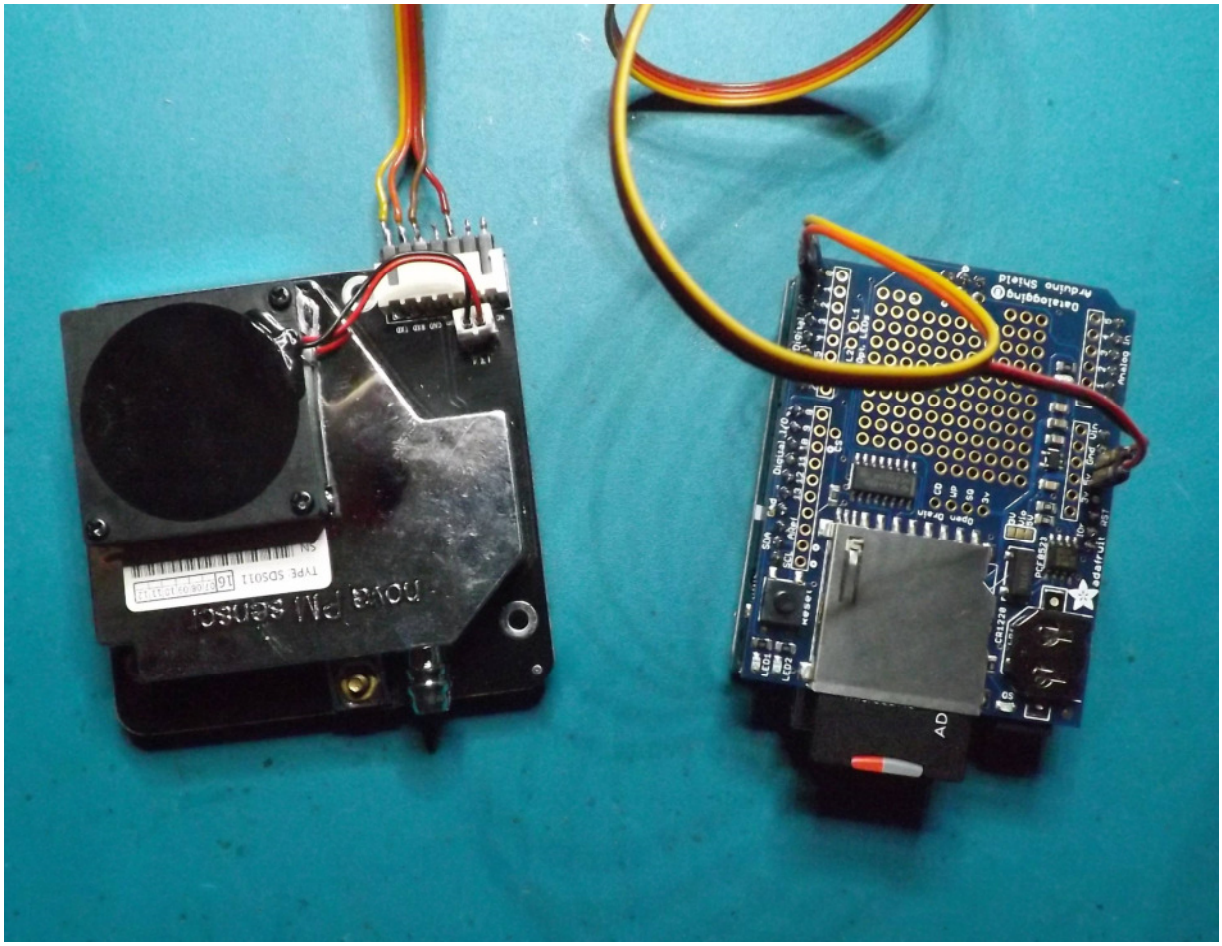

Abb. 3: SD011 Feinstaubsensor und Arduino Uno mit gestacktem SD-Card Shield

Literatur und Links

/1/ Charly Kühnast
Feinstaubmessung mit dem Raspi
In c't Make: IoT Special 1/2017

/2/ Nova Fitness Produktseite zum SDS011
http://inovafitness.com/en/Laser-PM2-5-Sensor-SDS011-35.html

/3/ Adafruit Produktseite zum Data-Logging Shield
https://www.adafruit.com/products/1141

/4/ hackspark Shop, Bezugsquelle für den SDS011 in Frankreich, aber teuer
https://hackspark.fr/en/nova-pm-sensor-dust-air-quality-detection-laser-sds011.html

Listings

Listing 1: Unix-Bash-Skript für die Ausgabe der SDS011 Messdaten auf dem Bildschirm (modifiziert aus /1/)

```
#!/bin/bash
#Anpassung für big endian unter RHEL 7
WDIR=~
while true; do
stty -F /dev/ttyUSB0 9600 raw
INPUT=$(dd conv=swab bs=10 count=1 </dev/ttyUSB0 2>/dev/null | od -x -N10 |head -n 1|cut -f2-10 -d" ");
#Ausgabe
#echo $INPUT
#echo " "
FIRST4BYTES=$(echo $INPUT|cut -b1-4);
#echo $FIRST4BYTES
if [ "$FIRST4BYTES" = "aac0" ]; then
        #  echo "check for correct intro characters: ok"
        echo " "
else
 echo "incorrect sequence, exiting"
 exit;
fi
PPM25LOW=$(echo $INPUT|cut -f2 -d " "|cut -b1-2);
PPM25HIGH=$(echo $INPUT|cut -f2 -d " "|cut -b3-4);
PPM10LOW=$(echo $INPUT|cut -f3 -d " "|cut -b1-2);
PPM10HIGH=$(echo $INPUT|cut -f3 -d " "|cut -b3-4);
#zu Dezimal konvertieren
PPM25LOWDEC=$( echo $((0x$PPM25LOW)) );
PPM25HIGHDEC=$( echo $((0x$PPM25HIGH)) );
PPM10LOWDEC=$( echo $((0x$PPM10LOW)) );
PPM10HIGHDEC=$( echo $((0x$PPM10HIGH)) );
PPM25=$(echo "scale=1;((( $PPM25HIGHDEC * 256 ) + $PPM25LOWDEC ) / 10 )"|bc -l );
PPM10=$(echo "scale=1;((( $PPM10HIGHDEC * 256 ) + $PPM10LOWDEC ) / 10 )"|bc -l );
echo "Feinstaub PPM25: $PPM25"
echo "Feinstaub PPM10: $PPM10"
done
```

Listing 2: Einfaches Arduino-Programm (Sketch) zur Ausgabe der SDS011 Messwerte auf dem Serial Monitor Fenster oder einem Terminal Programm (z.B. HTERM)

```
#define LEN 9
unsigned char incomingByte = 0;
unsigned char buf[LEN];
int PM2_5Val = 0;
int PM10Val = 0;

void setup() {
        Serial.begin(9600);
}

void loop() {
        int i;
        unsigned char checksum;
        // send data only when you receive data:
        if (Serial.available() > 0) {
          // read the incoming byte:
          incomingByte = Serial.read();
```

```
          if (incomingByte == 0xAA) {
            Serial.readBytes(buf, LEN);
            if ((buf[0] == 0xC0) && (buf[8] == 0xAB)) {
              for (i=1; i<=6; i++) {
                checksum = checksum + buf[i];
              }
              if (checksum == buf[7]) {
                PM2_5Val=((buf[2]<<8) + buf[1])/10;
                PM10Val=((buf[4]<<8) + buf[3])/10;

                Serial.print("PM2.5: ");
                Serial.print(PM2_5Val);
                Serial.println("  ug/m3");

                Serial.print("PM10 : ");
                Serial.print(PM10Val);
                Serial.println("  ug/m3");
                Serial.println();
             }
              else {
                Serial.println("checksum Error");
              }
            }
            else {
              Serial.println("frame error");
            }
          }
        }
}
```

Listing 3: Programm (Sketch) für einen Arduino-basierten Daten-Logger für den Feinstaub-Sensor SDS011 von Nova Fitness

```
#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include "RTClib.h"

#define LEN 9
#define windowSize 10


char fileName[15] = "datalog.txt";
File myFile;

RTC_PCF8523 rtc;

float aveArr[windowSize];
int loopcnt = 0;

unsigned char incomingByte = 0;   // for incoming serial data
unsigned char buf[LEN];
int PM2_5Val = 0;
int PM10Val = 0;
int PM2_5ave = 0;

void setup() {
  Serial.begin(9600);     // opens serial port, sets data rate to 9600 bps

  if (! rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }
  if (! rtc.initialized()) {
    Serial.println("RTC is NOT running!");
    // following line sets the RTC to the date & time this sketch was compiled
```

```
    // rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    // This line sets the RTC with an explicit date & time, for example to set
    // January 21, 2014 at 3am you would call:
    // rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));
  }


  pinMode(10, OUTPUT); // SD Card CS
  pinMode(9, OUTPUT); // status LED
  digitalWrite(9, HIGH);  // status not ready
  if (!SD.begin(10)) {
    Serial.println("SDcard not ready\n");
    return;
  }
  if (!SD.exists(fileName)) {
    myFile = SD.open(fileName, FILE_WRITE);
    myFile.println("###");
    myFile.flush();
  }
  else {
    myFile = SD.open(fileName, FILE_WRITE);
    myFile.println("-----");
    myFile.flush();
  }
  pinMode(9, LOW); // status ok

  for (int i=0; i<windowSize; i++) {
    aveArr[i] = 0.0;
  }

}

void loop() {
        int i;
        unsigned char checksum;

        // send data only when you receive data:
        if (Serial.available() > 0) {
          // read the incoming byte:
          incomingByte = Serial.read();
          if (incomingByte == 0xAA) {
            Serial.readBytes(buf, LEN);
            if ((buf[0] == 0xC0) && (buf[8] == 0xAB)) {
              for (i=1; i<=6; i++) {
                checksum = checksum + buf[i];
              }
              if (checksum == buf[7]) {
                PM2_5Val=((buf[2]<<8) + buf[1])/10;
                //PM10Val=((buf[4]<<8) + buf[3])/10;
                PM2_5ave = runAve(PM2_5Val, windowSize);

                DateTime now = rtc.now();

                Serial.print(loopcnt);
                Serial.println();
                Serial.print(now.year(), DEC);
                Serial.print('/');
                Serial.print(now.month(), DEC);
                Serial.print('/');
                Serial.print(now.day(), DEC);
                Serial.print(" ");
                Serial.print(now.hour(), DEC);
                Serial.print(':');
                Serial.print(now.minute(), DEC);
                Serial.print(':');
                Serial.print(now.second(), DEC);
                Serial.println();

                Serial.print("PM2.5: ");
```

```
                Serial.print(PM2_5ave);
                Serial.println("  ug/m3");
                Serial.println();

                if (loopcnt%windowSize == 0) {
                  myFile.print(now.year(), DEC);
                  myFile.print('/');
                  myFile.print(now.month(), DEC);
                  myFile.print('/');
                  myFile.print(now.day(), DEC);
                  myFile.print(" ");
                  myFile.print(now.hour(), DEC);
                  myFile.print(':');
                  myFile.print(now.minute(), DEC);
                  myFile.print(':');
                  myFile.print(now.second(), DEC);
                  myFile.print('\t');

                  myFile.print("PM2.5:\t");
                  myFile.print(PM2_5ave);
                  myFile.println();

                  myFile.flush();
                }
                loopcnt++;
              }
              else {
                Serial.println("checksum Error");
              }
            }
            else {
              Serial.println("frame error");
            }
          }
        }
}

float runAve(float x, int n) {
  float sum;
  int i;

  for (i=0; i<n-1; i++) {
    aveArr[i] = aveArr[i+1];
  }
  aveArr[n-1] = x;

  for (i=0; i<n; i++) {
    sum = sum + aveArr[i];
  }

  sum = sum/n;

  return sum;
}
```