A Low-cost Portable PM-Measurement System Based on the Alphasense OPC Sensor and an Arduino Microcontroller Platform

Bernd Laquai, Institute of Combustion and Power Plant Technology – University Stuttgart

July 11th, 2018, Version 1.0

Introduction

In many emerging economies, the focus on profitability is much stronger than the focus on conservation of ecology and living conditions of humans. In such countries, governmental regulations enforcing an environmental protection typically can't keep pace with production of waste and pollution. But even in highly developed economies and countries, the trend of forming overcrowded cities and regions, results in traffic congestion and densely built residential areas generating pollution from heating in winter, such that already established governmental regulations no longer are met. Especially, particulate matter concentrations tend to exceed limit values in many large cities. The result is increasing health issues of citizens that more and more become evident. However, the issues appear in a statistical way and with long latencies.

It is important that tackling these issues need to be accompanied with accurate air quality measurements. A big difficulty in this respect however is, that accurate measurement equipment is very costly, difficult to operate and clumsy. The consequence is, that even in wealthy countries accurate air quality measurements are only performed at few locations where very high pollution is expected. The situation in other locations such as in residential areas or the overall spatial distribution of air quality parameters across a larger area often remains uninvestigated. The situation in emerging countries is even worse due to the missing financial funding for air quality investigation projects and the necessary equipment.

However, with respect to measurement of particulate matter concentration as one key air quality parameter, a big improvement in sensor cost was achieved in the recent years. The higher integration and performance of electronics helped to construct extremely cheap and small PM-sensors based on the laser scattering principle. These very low-cost sensors are available at a cost of less than 30Euro per device. Whereas most of these consumer oriented devices are only able to adequately measure PM2.5 concentrations and just do very coarse estimations for PM10 by extrapolating the smaller particle concentration results, newer low-cost sensors are about to be available on the market that indeed are able to even measure PM10 accurately at a cost that is still by a factor of 10-30 lower compared to professional equipment.

Such low-cost sensors paired with modern highly integrated microcontrollers allow to develop low-cost portable PM-measurement systems that are well suited to do sufficiently accurate mobile and spatially distributed measurements for research purposes. At the cost of about 500 Euro per unit several measurement systems of this type may even be configured for use in a portable network that helps to more efficiently investigate the air pollution situation a whole city or industrial area. The integration of GPS for geo-referenced measurement as well as highly accurate humidity and temperature sensors enable to geographically map and interpolate PM concentrations along with the key meteorological parameters that are very important for interpreting PM results. In contrast to already existing very low-cost citizen science networks measuring PM-concentrations with a very high number of immobile nodes and poor data quality, particularly for the coarse particle fraction, such a portable system may indeed deliver a more accurate and thorough insight into the spatial and temporal distribution of air pollution caused by fine and coarse dust particles up to 10µm and beyond.

For our idea of creating such a low-cost mobile PM-measurement system, however, an additional aspect of great importance was education. When economies evolve rapidly, many engineers are educated in technical disciplines that promise profit. In contrast, the education of young professionals in environmental protection typically lags the highly profitable disciplines. In order to more aggressively address this issue, we must start with rebalancing the education of young professionals. Particle physics and the accurate measurement of particulate matter concentrations of course is a complex subject. Nevertheless, learning of a highly complex theory is much more easy and successful when it is accompanied with realistic practical work.

In this respect, the use of very expensive professional equipment is often prohibitive in training classes with many students, particularly in countries with low educational budgets. Therefore, a key goal for the development of our portable low-cost PM measurement system was to provide equipment for the education that can easily be understood and operated by students. The use of the Arduino microcontroller platform, initially developed at a University in Italy for teaching, is a very helpful contributor for this purpose, and it is already known by many students. Furthermore, the system should be affordable for universities despite tough budget constraints. We are convinced that the development of such a system is a significant step ahead in addressing the world-wide issue of air pollution with the education of more young professionals that later will be experts in the discipline of measuring and assessing particulate matter concentrations.

Fig. 1: The low-cost PM-measurement system

Overall System Concept

The mobile low-cost PM measurement system is based on a multi-processor system. Three Arduino microcontroller work asynchronously in parallel a main controller (Arduino Mega 2560), a controller for the and one for the GPS OPC (both Arduino Uno). Since the Alphasense OPC-N2 used as PM sensor requires a dedicated SPI bus and because an Arduino microcontroller provides only a single SPI bus, one Arduino Uno is dedicated to control the OPC. The OPC data lines use 3.3V logic-levels whereas the

Arduino Uno uses 5V. Therefore, a level-shifter is necessary. The power supply of the internal electronic of the OPC and its air fan requires 5V. Therefore, a dedicated regulated DC/DC is used to provide a separated 5V voltage to the OPC. Each Arduino has an own 5V linear regulator to provide a clean voltage to the microcontroller chip and the other electronics on the board and the shields.

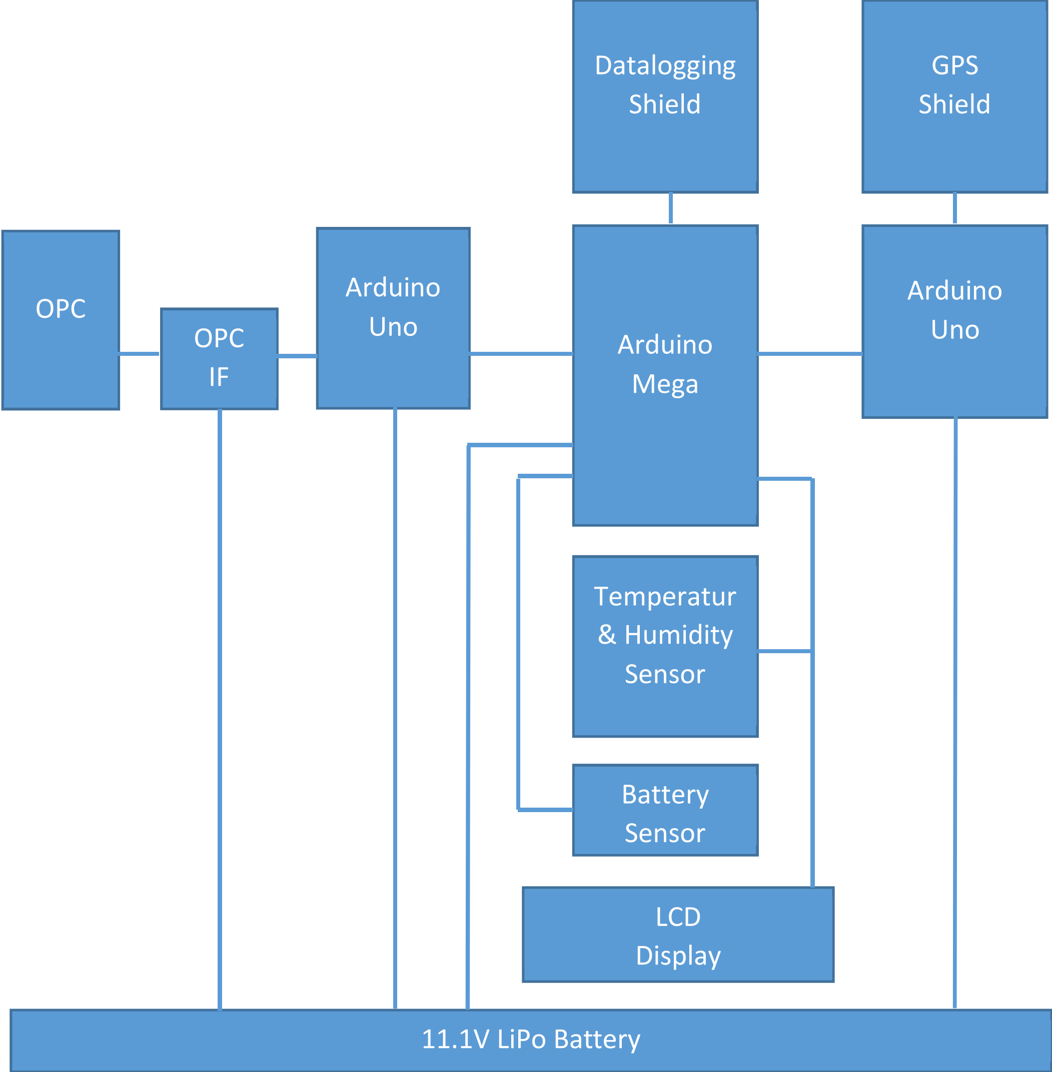

Fig. 2: Conceptual block diagram of the PM-measurement system

The GPS also uses its own Arduino Uno for control. The GPS shield communicates on a software serial interface with the Arduino. Due to the limited RAM memory of an Arduino Uno, the datalogging capability (micro SD-card) is not used, instead the data are passed to the larger main controller for data logging. In case the box is used for measurements at places where no GPS reception is possible but for stationary measurements where the geo-coordinates are known, the GPS can be deactivated with a switch before turning on the power switch. In this case, the real-time clock on the datalogging shield is used to create the necessary time-stamps for logging.

The main controller is an Arduino Mega 2560 that carries the data logging shield. It communicates with the two other Arduino Uno via a software serial interface and stores all data on a regular SD-card for later postprocessing. Since all processors work independently and asynchronously the main controller sends out two circuit select signals (OPC Cs and GPS Cs) to request information from both Arduino

Uno. Only during the time the circuit select signal is set as a semaphore, the respective Arduino transmits its locally processed data to the main controller.

In addition to this, the main controller manages an I2C bus for the LCD display and the temperature & humidity sensor. It also monitors the battery voltage and reads in the GPS switch position to decide between the mobile and the stationary mode.

The 3 Arduinos and the DC/DC for the OPC supply are powered from a 11.1V rechargeable LiPo battery as used in model aircrafts and cars. The LiPo battery must be charged externally from a dedicated charger. Since the system has no protection against deep discharge, the battery voltage is sensed by a battery monitor and the voltage is shown on the LCD-display. The operator has to ensure that the voltage does not drop below 9V. For stationary measurements without GPS reception the system can also be powered from an external voltage supply with a voltage in the range of 7-12V and a current capability of 2A max.

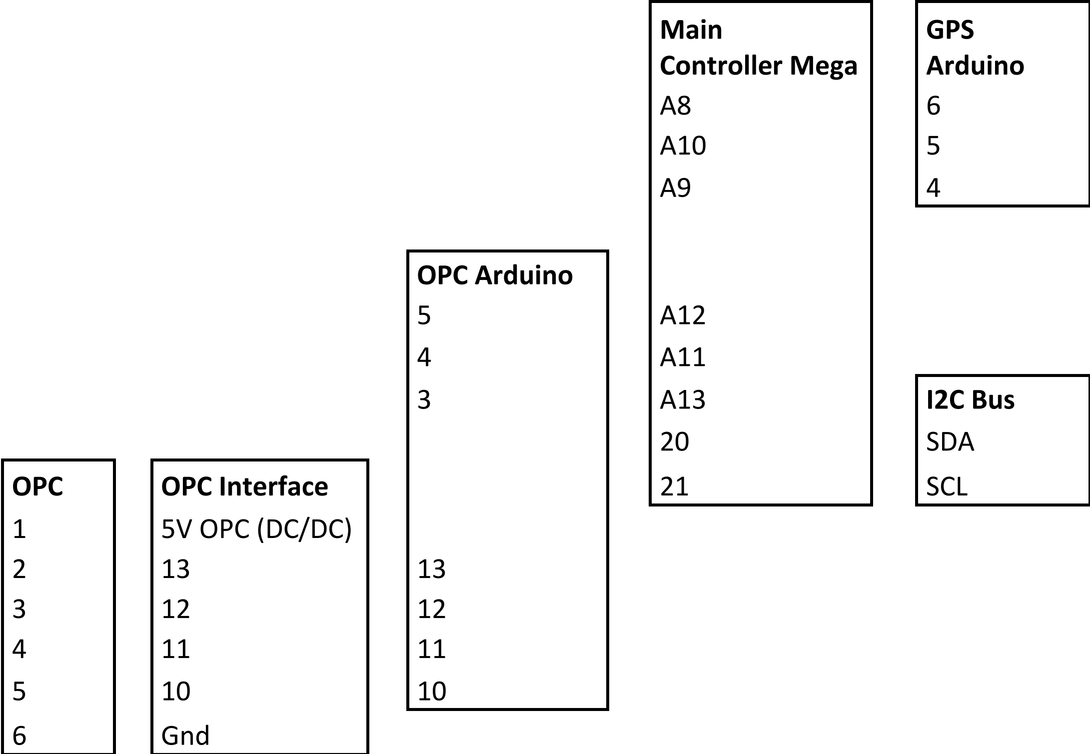

Table 1: Assignment of the used IO pins to the wiring between the functional blocks

Power Supply

The power supply is either possible from the rechargeable battery or via the power supply jack built into the housing. The LiPo battery is a 3S1P type that can be charged from a charging supply with balancer. Currently a 2400mAh (26.6Wh) LiPo battery from TopFuel (ECO-X 2400) is used that allows operation for about 6 hours. It has two connectors, one high current connector that is left unconnected for safety reasons and the smaller balancer connector with 4 pins. The electronics of the box is connected to the smaller low current balancer connector via a 1N5817 schottky diode.

The power supply jack is also connected to the electronics via a 1N5817 schottky diode. The two schottky diodes at the battery and the power jack help to prevent damage in case the power polarity is reversed. They also prevent a current flow from the power jack into the battery and vice versa.

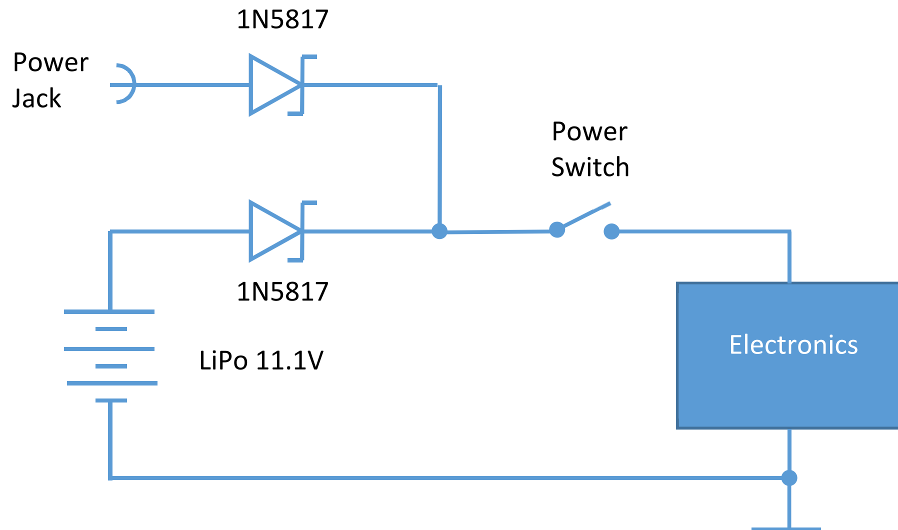

Fig. 3: Power supply from the LiPo battery and an external supply via a power jack at the enclosure

Main Controller (Arduino Mega 2560)

As a main controller, an Arduino Mega 2560 was chosen due to its large main memory. The first prototype of the box didn't have the option to work without GPS and it didn't use the real-time clock of the data logging shield. Under these conditions the main memory of an Arduino Uno was just sufficient. But as soon as we included the real-time clock, the software became instable, mainly during the string manipulations. Therefore, we replaced the Uno with a Mega. With the Mega, a further advantage is, that it enough memory is left to even add future features and sensors. Aside of the larger main memory and the higher IO pin-count, the Mega is similarly simple as the Uno and the data logging shield also works without any problems on the Mega. The I2C Bus has to be changed from the A4 and A5 pins on the Uno to the pins 20 (SDA) and 21 (SCL) on the Mega.

The data logging shield carried by the Mega is of Arduino Uno size and carries stackable headers. Similarly, the other shields on the Arduino Uno are also equipped with stackable headers. Therefore, some of the interconnection cables are connected through the stackable headers.

The power is supplied via the a 2.1mm center-positive plug plugged into the board's power jack. Note that all 3 Arduino boards are supplied in the same way from the LiPo battery, therefore the grounds of all boards are shorted via each board's power jack. No additional ground connection between the board should be established to avoid unnecessary ground loops.

For switching between GPS and RTC, a single pole double throw (SPDT) switch is used with the center contact connected to a digital input (9). One throw is connected to ground and the other throw is connected to a digital pin that permanently outputs a high level (7). When a logical HIGH is read by the digital input (9) the software reads data from the GPS and logs time and coordinates from the GPS shield after startup. If a logical LOW is read, the software reads the time and date from the real-time clock on the data logging shield.

The LCD display and the humidity sensor is connected to the I2C bus (SDA 20 and SCL 21). Power to both devices is also provided along with the data lines sourced from the Mega 5V linear regulator. The battery sensor makes use of the Vin pin that is connected to the board's power jack. It additionally uses a ground connection and a connection to the analog input pin A1.

The Mega controls two additional software serial links along with the semaphore signals to control the query of data. Towards the OPC Arduino, the pins A11 and A12 are used as Tx and Rx respectively and A13 is used as semaphore signal (OPC Cs). Towards the GPS Arduino the pins A9 and A10 are used as Tx and Rx respectively and A8 is used as semaphore signal (GPS Cs).

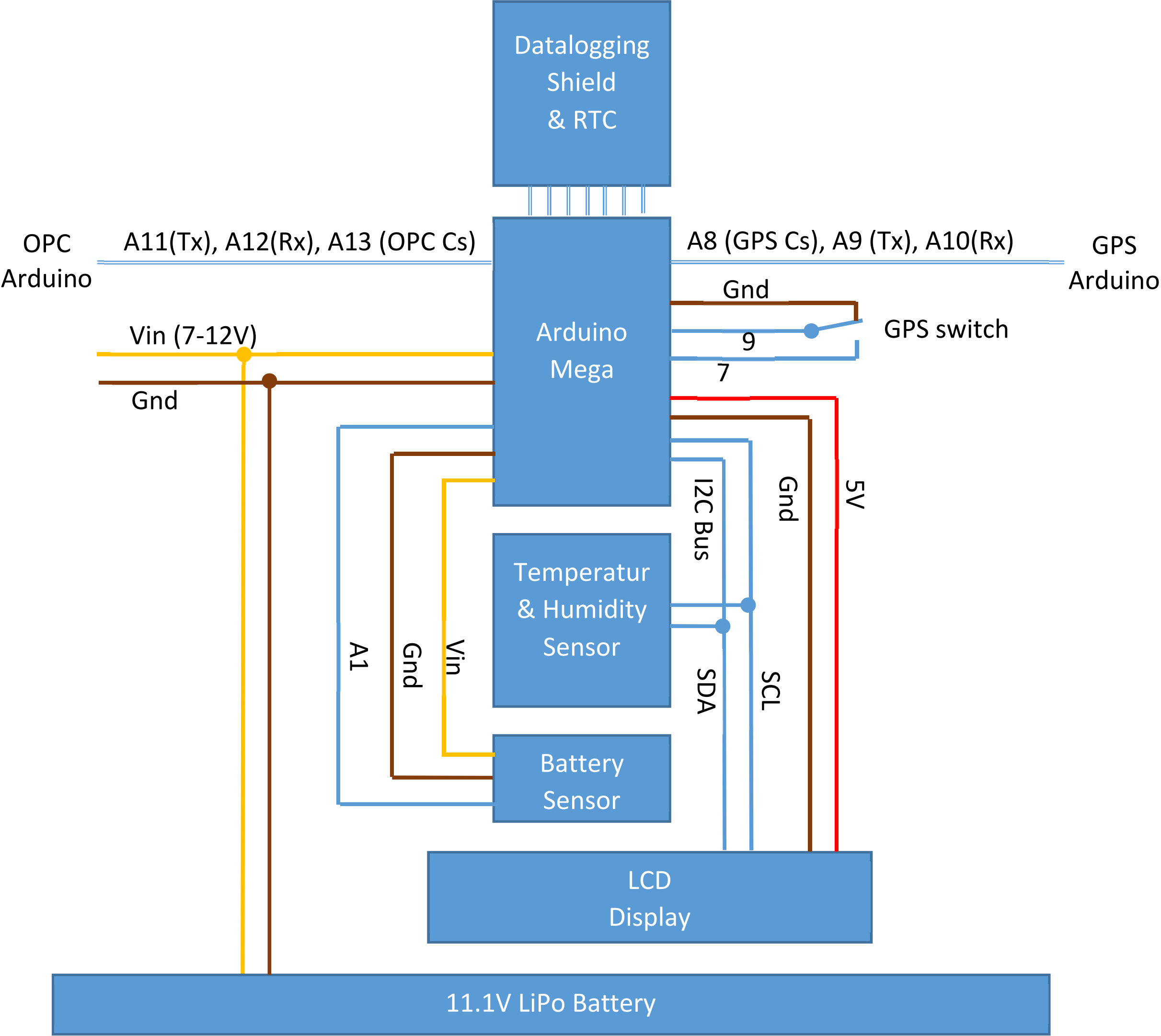

Fig. 4: Block diagram, signals and pin assignments of the main controller

Temperature & Humidity Sensor

The ability to monitor humidity is very important during PM-measurements since hygroscopic growth of particles as well as condensing water can strongly influence the PM measurement results of PM-sensors that aren't operated with an air dryer preceding the gas inlet. Monitoring the humidity as well as temperature enables postprocessing to correct the humidity influence or to mask time periods with very high humidity such as fog.

Many of the very low-cost temperature & humidity sensors are available on the market, particularly those provided from Chinese manufacturers e.g. the AM 2302 (DHT11). However, with the first prototype as well as with other applications, the experience was made that the very low-cost sensors show high variability and poor accuracy when the relative humidity comes close to 100%. Since for many humidity correction models the derivative of the growth factors become huge near 100%, the

error propagation of the inaccuracy of the humidity sensors is pretty strong. Therefore, we decided to deploy a costlier sensor manufactured by the Swiss company Innovative Sensor Technology (IST) named HYT 221. This device also provides a I2C bus connectivity (address 0x28).

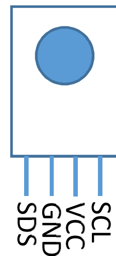

Fig. 5: Pin assignments of the temperature and humidity sensor HYT221

LCD

In first experiments with low-cost mobile PM measurement boxes it quickly turned out that a pure data logging functionality bears the risk, that a long-lasting measurement is done with the uncertainty that the box is not performing correctly e.g. when the SD card is not inserted correctly. Therefore, even when power consumption is increased, adding a display seemed to be beneficial because the operator can at least check some key parameters that indicate correct functionality. For this purpose, a 20 character x 4 line LCD display with backlight was added. To minimize the wiring, a LCD display with an I2C display was used.

The display actually is a standard parallel LCD display with a SPLC780 controller (Industry-standard HD44780 compatible controller) but is additionally equipped with a piggy-back parallel to I2C bus module (LCM1602). The I2C interface of the LCD uses the address 0x27. The LCM1602 module on the backside also contains a potentiometer to adjust the contrast. The 4-pin connectivity is printed as silk screen on the PCB between the potentiometer and the pin header.

During normal operation, the LCD displays the time and some key measurement data such as PM10, temperature, humidity and the battery voltage.

Data Logging Shield & Real Time Clock (RTC)

As a datalogging shield we used the Adafruit “Assembled Data Logging shield for Arduino” (product id 1141). This shield contains a standard SD-Card socket connected to the SPI bus and a PCF8523 standard I2C real time clock IC running on a 32.768kHz crystal. The SD-card interface uses the SPI bus connected internally to the pins 10-13.

The RTC chip is connected to the I2C bus through internal wires of the shield and uses address 0x68 as defined in the RTC library supplied by Adafruit. The RTC requires a CR1220 battery to keep the clock running during power off.

Fig. 6: The backside of the LCD display, it carries a parallel to I2C bus converter piggy-back module

When the measurement box is turned on and the SD-card is not inserted properly or the SD-card can't be written, the message "SD-card fail" is displayed for 5 seconds after startup. When this time has elapsed, the system starts even without data logging. When data logging is working properly the red LED named SD on the SD-card shield flashes all 5 seconds for a short moment.

Serial Communication System

The 3 Arduino controllers run on their own clock and work asynchronously. However, since data logging is performed centrally by the main controller a communication system had to be established. The two Arduino Uno boards use this communication system to report the results to the main controller. For the OPC Arduino the histogram bin data as well as the PM results are logged and for the GPS Arduino the time and the geo-coordinates are logged. Since serial UART interfaces based on the RS232 standard are easily implemented on Arduino, the communication between the processors is established in terms of software serial interfaces on regular digital pins.

In order to allow the main controller to keep control on the asynchronously provided information by the two Arduino Uno, a semaphore system is implemented. Two control lines for communication selection accompany each serial link. When the main controller sets the GPS Cs line high, the GPS Arduino starts sending data. When the main controller has successfully received the data via the serial link it sets the GPS Cs line low. This stops the GPS Arduino from sending further data. The same is the case for the OPC Arduino. The OPC Cs control line is used to have the OPC Arduino either sending data or being quiet. With respect to the software use of the pins for serial communication, it is important that RX and Tx are crossed between two Arduinos, which means that always a Tx pin on one Arduino has to be connected to a Rx pin on the other Arduino and vice versa.

On the side of the Arduino Mega acting as the main controller it is important to use Rx pins that are capable to handle change interrupts for the software serial interface. Not all pins can do that. The pins used for serial communication and communication control are the following:

A11(Tx), A12(Rx), A13 (OPC Cs)

A9 (Tx), A10 (Rx), A8 (GPS Cs)

In order to compress the data during transmission and in order to reduce the sending time, binary data are sent in terms of data structures using the SoftEasyTransfer-Arduino-Library of Bill Porter. This allows to even safely transmit the complete histogram bin data of the OPC via the serial link.

Battery Sensor

LiPo batteries are quiet sensitive to deep discharge. Even a single deep discharge may end the life of a LiPo battery prematurely. Therefore, monitoring the battery during operation is very important. No circuit is available in the system that automatically disconnects all the electronics from the battery. However, the battery voltage is sensed and displayed on the LCD display. So the operator has to regularly check the displayed battery voltage and when it falls below 9V he has to turn the main power off as soon as possible.

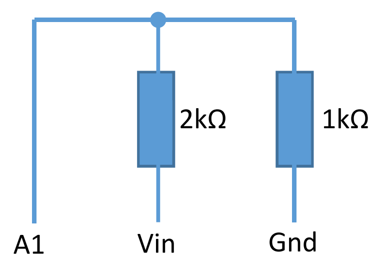

Fig. 7: The battery voltage sensor

The battery sensor is made up from a voltage divider, that divides the voltage that is available at the Vin pin. Vin is directly connected with the power jack on the Arduino board. The divided voltage is supplied to an analog pin of the main controller and is converted by the internal 10bit AD-converter to obtain a digital value for the battery voltage. The voltage divider ratio is used by the software to calculate the actual voltage available at Vin from the digital 10-bit value and the internal reference voltage of 5V.

OPC

The PM-measurement system is using the Alphasense OPC-N2 as a PM sensor. This device is a lower cost laser scattering sensor that calculates PM from pulse counts in a size histogram. The size histogram is made up from 16 count bins and the counts in each bin are converted into mass concentration per bin and are finally summed up to weighted sums as PM values. The weights represent the filter characteristics as they are defined for the different PM categories and further weights that represent the counting efficiency of the sensor. This ensures similar results compared to professional equipment that also used a bin based histogram approach. Furthermore, the OPC-N2 uses a voltage controlled fan to define the air-flow through the measurement chamber in contrast to many of the very low-cost sensors that do not control the air flow at all. The sensor is able to perform one measurement per second which is quite fast. Nevertheless, the communication and data logging also consumes time, therefore the overall sampling time in the PM-measurement system is between 5 and 6 seconds.

A certain drawback is, that the OPC-N2 requires an own dedicated SPI bus connectivity. The SPI data lines are not set to high impedance when there is no communication required for the OPC. Therefore, the SPI interface can't be shared with other devices in a bus type SPI communication. This is the main reason why a dedicated Arduino is required for controlling the OPC-N2. The second SPI bus on the main controller board is required for the SD-card interface operated through the internal SD card library.

OPC Arduino

The Arduino that controls the OPC is connected to the OPC interface via the standard SPI interface using the digital pins 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). The Arduino pins use 5V levels for this purpose and need to be converted to 3.3V on the OPC interface board for use with the OPC-N2.

The serial communication is established through the pins 5(Tx), 4(Rx) and 3 (OPC Cs) which acts as the semaphore control to indicate that the OPC Arduino is allowed to send data.

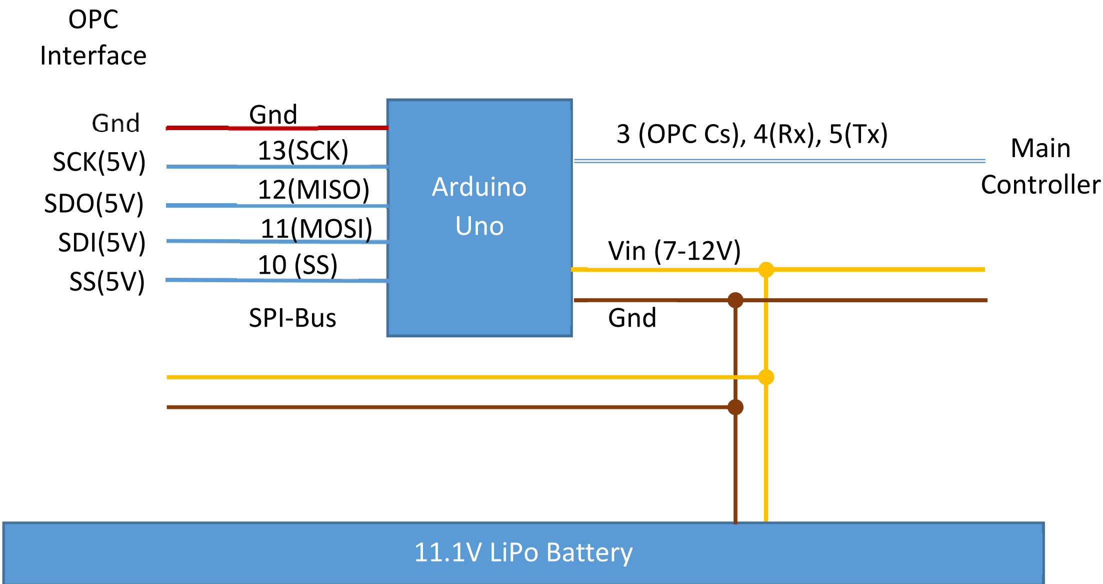

Fig. 8: Block diagram, signals and pin assignments of the OPC Arduino

OPC Interface

The OPC interface provides al level shifting capability and a power supply to the OPC. The level shifters are consisting of a voltage divider made up by a 2kΩ resistor to Gnd and a 1kΩ resistor towards the 5V signal side on each SPI pin.

To accommodate the high surge current when the fan of the OPC starts and to prevent conducted noise to couple into the other circuitry, a dedicated regulated DC/DC module is used to supply the 5V voltage to the OPC. For this purpose, a Traco Power TSR 1-2450 DC/DC-converter (or equivalent) is used on the OPC interface board. The DC/DC is able to provide a max. current of 1000 mA and runs from an input voltage as low as 6.5V. This module does not require any further additional external circuitry such as blocking capacitances.

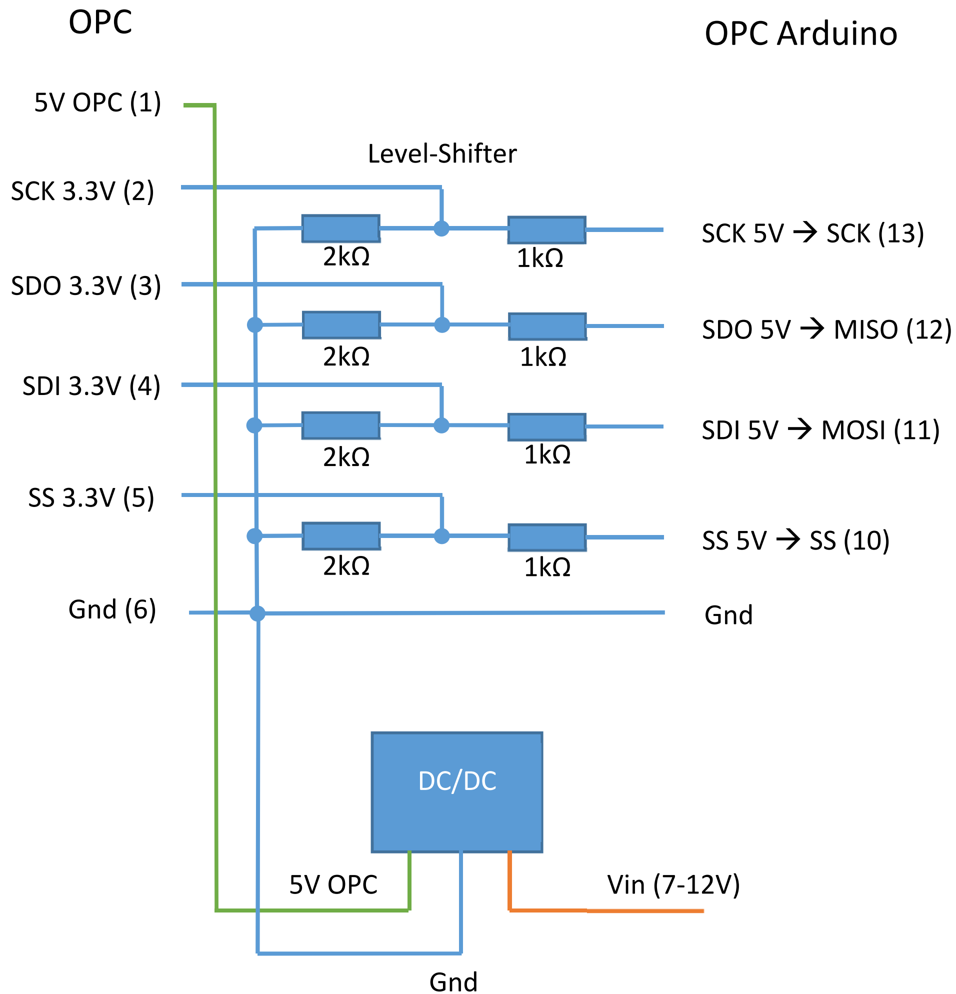

Fig. 9: The OPC interface board with level shifter and dedicated OPC supply

GPS

One purpose of the GPS shield is to provide the geo-coordinates of the location where a PM-measurement was taken. However, it is also used to assign a time stamp to a measurement value to allow synchronization with other equipment. The GPS provides a very highly accurate clock information in UTC-time. The disadvantage is that GPS reception requires a free sight to the sky. GPS signals become quickly shielded from building materials or even wet leaves in a dense forest. To allow measurements without GPS, a fallback is provided by use of the real-time clock (RTC) on the data logging shield. Therefore, a switch is available on the main controller to activate the use of the RTC and to ignore the GPS. This switch has to be set correctly before the power is turned on, since the switch is only read during the startup phase.

During startup, the GPS needs to first collect data from the GPS satellites that allow to calculate the position (a so-called fix). When the system was not used for days or it gets turned on far away from the previous measurement location, the ephemeris data of at least 4 satellites have to be collected and stored. This may take quite a while (up to half an hour). For faster startup, it is recommended to turn the PM-measurement system on sufficiently before a measurement is planned.

GPS Arduino Uno

The GPS Arduino Uno just controls the GPS on the GPS shield and provides all key data of a NMEA RMC message to the main controller via the software serial connection. The software serial connection

between the GPS Arduino and the main controller uses the pins (5) and (4) for the Tx and Rx connection respectively. Pin (6) on the GPS Arduino is used as a circuit select semaphore control. The power for the GPS Arduino is supplied via the a 2.1mm center-positive plug plugged into the board's power jack.

It is important that the GPS antenna on the GPS module of the shield sees a low satellite horizon, to receive a maximum number of satellites. The more satellites are in view, the better is the positioning accuracy. Therefore, the GPS Arduino is mounted in an elevated position with respect to the mounting plate of the enclosure. The transparent cover should not be covered with anything that attenuates RF signals.

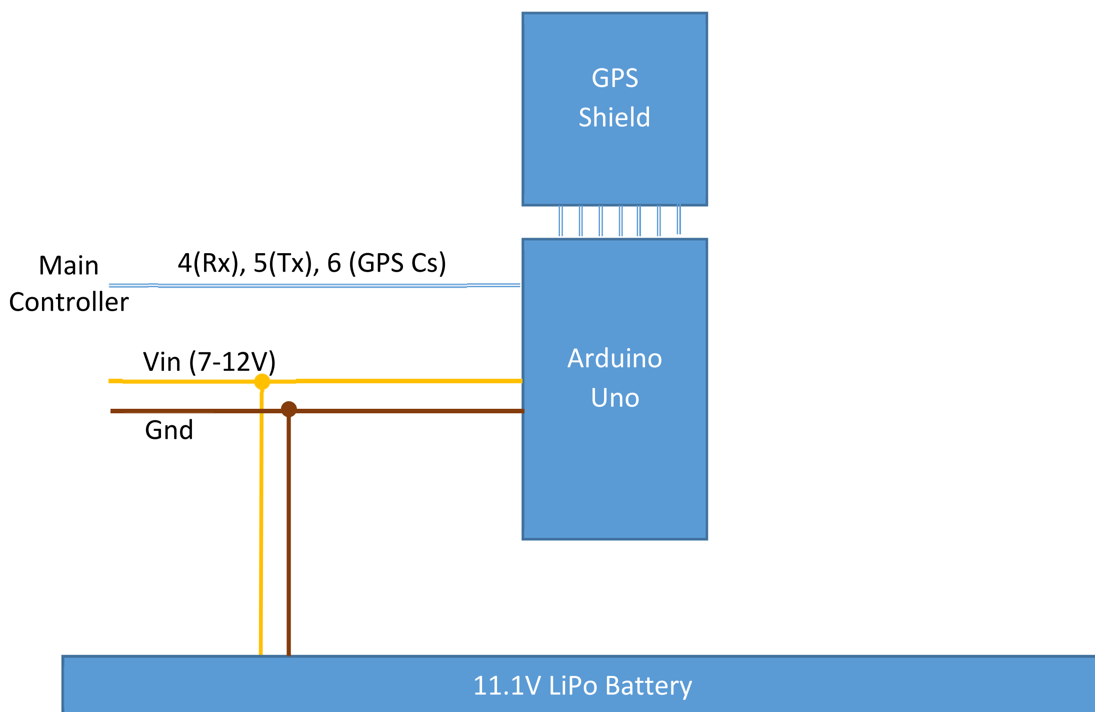

Fig. 10: Block diagram, signals and pin assignments of the GPS Arduino

GPS Shield

The GPS shied used in the low-cost PM measurement box is an “Adafruit Ultimate GPS Logger Shield” (product id 1272). It contains the MTK3339 GPS device providing a NMEA compatible output via a further shield internal serial interface. The GPS shield can either operate on the hardware serial connectivity of the Arduino (Pin 0 and 1) or on a software serial connection on pins (7) and (8). In our application we used the software serial connectivity provided on the GPS shield. For the GPS shield to work properly in this application, it must be ensured that the respective selector switch near the pin (2) is set correctly to the position “Soft. Serial” (and not to “Direct”). This shield internal software serial interface is related to the communication of the GPS module on the shield and the GPS Arduino that carries the shield. The software serial link between the GPS Arduino and the Mega is different from this and should not be confused with the GPS shield internal serial link.

The microSD-card interface on the GPS shield is not used for logging due to the more difficult mechanical handling of the small card and due to the fact that the main memory of the GPS Arduino Uno is already at its limits with the indispensable GPS related software.

The GPS shield requires a CR1220 data retention battery to store the ephemeris data of the GPS, this allows to obtain a faster first fix when the location hasn't changed too much or not too much time has elapsed since the last fix.

The GPS shield signals its status with a red and a green LED. During collection of ephemeris data, the red LED named FIX is blinking evenly with a duty cycle of 50%. During this time the LCD shows the message "Acquiring satellites". As soon as enough satellites are in sight and the ephemeris data are sufficient to calculate a fix, the red LED goes off for most of the time and remains flashing only for a short moment. From this point on the LCD displays the time and some additional measurement data.

Logging functionality

Whenever the measurement system is working with a SD-card, data will be logged in a file named DATALOG.TXT on the card. In a file system of a computer, the modification date of this file is always shown as "1.1.2000 00:00". The data file contains readable ASCII text, with tab spaced columns and a newline character at the end. Whenever the file is created, the first line written contains the three characters "###". When the measurement system is powered on and a file with the name DATALOG.TXT already exists on the SD-card, data will be appended. However, the first line written will contain the 5 characters "-----" only to indicate a separation from the previously stored data.

The data will be logged in the following sequence per line and separated with tabs:

Case 1, GPS operation

- Time (UTC format) hours, minutes, seconds, without separator and without leading zeros
- Day, month, year without separator and without leading zeros
- North coordinate (latitude) according to NMEA format
- East coordinate (longitude) according to NMEA format
- Bin 0-16 of the number count histogram
- PM10
- PM2.5
- PM1
- Temperature
- Humidity

Case 2, RTC operation

- Date-time (European format) with a space between date and time
- Bin 0-16 of the number count histogram
- PM10
- PM2.5
- PM1
- Temperature
- Humidity

The PM data as well as temperature and humidity data are represented as float numbers with two decimal digits. The NMEA format of the coordinates contain also float numbers, however, degrees and minutes of longitude and latitude are not separated with a character (see the NMEA coordinate data specification).

Mechanical Construction

Housing

A plastic housing with transparent cover was chosen as an enclosure for the electronics. The housing is a model EKJB 130 T made by FIBOX. It has an IP67 tightness specification but due to the fact that the OPC needs an air inlet and outlet the box is no longer water tight.

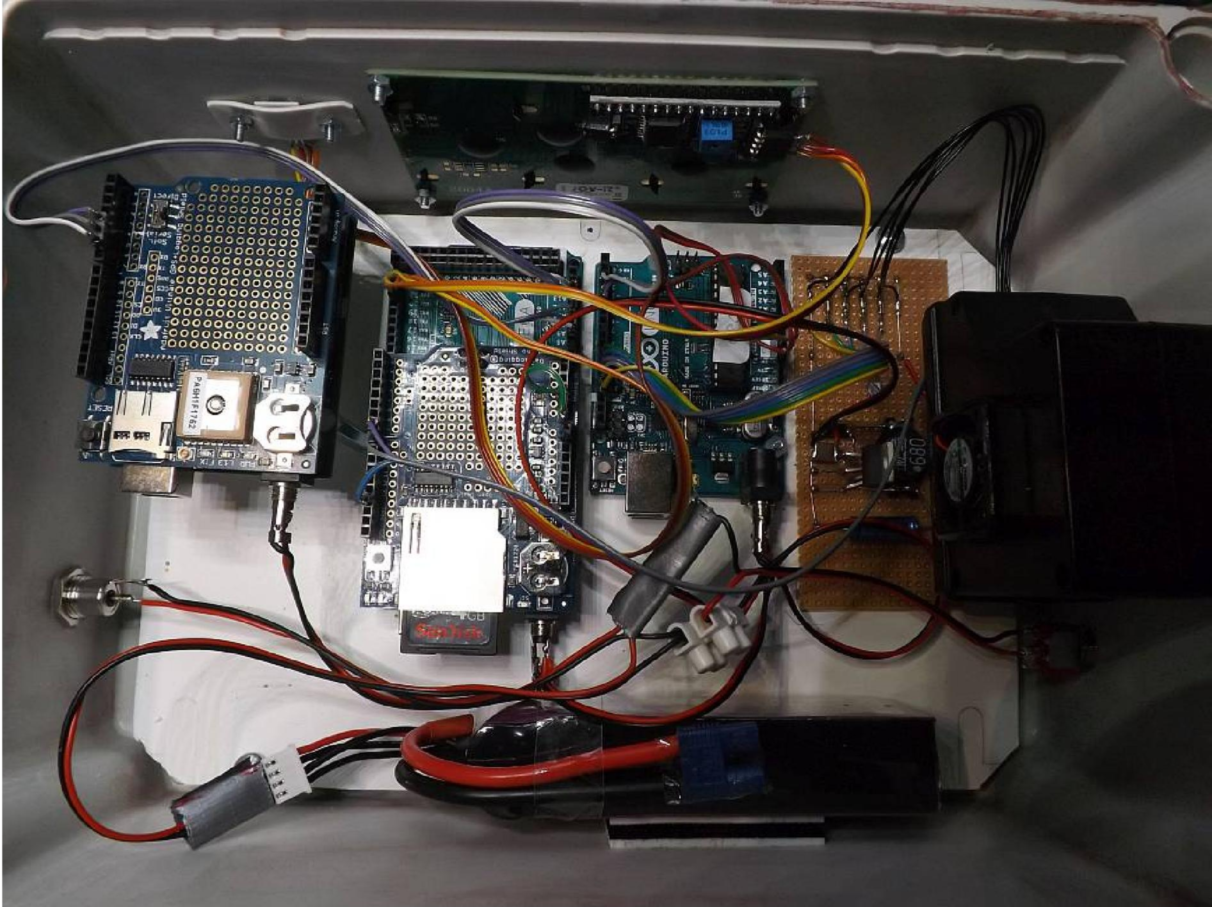

Fig. 11: Arrangement of the electronics inside of the enclosure

Fig. 12a: Cutouts and holes for the LCD display and the temperature & humidity sensor

Fig. 12b: Drill holes for the Alphasense OPC, the power and the GPS switch

Fig. 12c: Drill holes for the Alphasense OPC air outlet and the external power jack

EMI Coating

Since particle transport in the air is highly sensitive to electrostatically charged surfaces, we coated the bottom of the housing with highly conductive shielding lacquer on copper basis (EMV Kontakt Chemie). It is not possible to also coat the cover of the enclosure since this would prevent the GPS signals to reach the antenna.

Post-Processing Application for the PM-measurement system

For the postprocessing of the data we have developed a software that maps PM-measurement data on geographical maps obtained from either Open Street Map or Google Maps, to localize areas of high air pollution or sporadic events of high PM concentration. The PM-measurement system as well as the postprocessing software were tested in Ulaanbaatar, a city that suffers a lot from air pollution. As an example, the map in fig. 13 shows a section of the center city of Ulaanbaatar with color coded tracks. The color coding shows the PM10 concentration measured with the PM-measurement system mounted on a measurement car.

Fig. 13: Application of the low-cost PM measurement box in the city of Ulaanbaatar

Appendix: Arduino Sketches

Since the PM-measurement system was mainly developed for practical work of students in the subject of environmental engineering, we tried to keep the code development for the Arduino software maximally simple. The goal was to ensure that students not too much familiar with computer science are still able to understand and even to modify it. Of course, a larger processor like the Raspberry Pi or any other more powerful embedded controller would have been more suited to handle all feature on a single board while achieving an even higher integration.

The partitioning of the tasks between three individual, very simple Arduino boards however, has proven to be very helpful to give a clear structure to both, the hardware and the software. The concept of a serial communication between the 3 processors also contributed to establish well defined and distinct interfaces that additionally help the ease of understanding. So, from the didactic point of view, the Arduino based concept seems to be still the best with respect to teaching purposes in non-IT related disciplines.

A lot of debug output had been added to the code of the sketches. The debug information is output to the serial monitor when the respective Arduino is connected to the PC via an USB cable. The debug output via serial.print() statements not only serves to better understand issues during operation and debug but also helps to better understand the code.

Libraries

Aside of the libraries included in the Arduino SDK the following external libraries were used and must be located inside the library folder during compilation

LCD: New LiquidCrystal from Francisco Malpartida

RTC: Adafruit RTC library (see product folder of data logging shield)

Serial Communication: SoftEasyTransfer Library from Bill Porter

For the OPC no readily available library was found for Arduino platform. However, a lot of David Hagan's C++ library for the OPC-N2 could be leveraged to the Arduino SDK. This leveraged code is embedded in the sketch of the OPC Arduino and is not yet provided in terms of a further library.

All the sketches were compiled with the Arduino SDK version 1.6.13 .

Sketch for the Main Controller (Mega)

```
#include <LiquidCrystal_I2C.h>
#include <Wire.h>
#include <SoftEasyTransfer.h>
#include <SPI.h>
#include <SD.h>
#include <SoftwareSerial.h>
#include "RTClib.h"
#define gpsCsPin A8  //Mega
#define opcCsPin A13  //Mega
#define N 50
#define MAXCNT 50
#define HYT939_ADDR 0x28
#define TFACTOR 99.2909
#define TDELTA 40.0
#define HFACTOR 163.83

RTC_PCF8523 rtc;
DateTime now1;

char fileName[15] = "datalog.txt";
File myFile;
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Set the LCD I2C address

SoftwareSerial mySerial(A10, A9); //GPS rx, tx
SoftwareSerial opcSerial(A12, A11); //OPC rx, tx

//create object
SoftEasyTransfer ET;
SoftEasyTransfer ETopc;

struct RECEIVE_DATA_STRUCTURE{
  unsigned long tim;
  char nord[10]; //precision of float not sufficient!
  char east[11];
  unsigned long date;
};
RECEIVE_DATA_STRUCTURE mydata;

struct RECEIVE_DATA_STRUCTURE2 {
  unsigned int bin[16];
  unsigned long PM10val = 0;
  unsigned long PM2_5val = 0;
  unsigned long PM1val = 0;
};
RECEIVE_DATA_STRUCTURE2 mydataOPC;

int gpsOnPin = 9;   //controls if GPS is on
int gpsHigh = 7; // pin set constantly high for turning GPS on, substitute for +5V
int gpsOn = 0; //GPS or RTC

String timeStr;

void setup(){
  Wire.begin();
  lcd.begin(20,4);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Intisys PM Logger 3");
  pinMode(10, OUTPUT); // SD Card CS


  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(2017, 1, 21, 3, 0, 0));
  }
  //rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));


  pinMode(gpsCsPin, OUTPUT);
  pinMode(opcCsPin, OUTPUT);
  digitalWrite(gpsCsPin,LOW);
  digitalWrite(opcCsPin,LOW);
  mySerial.begin(9600);
  opcSerial.begin(9600);
  Serial.begin(9600);
  //start the library, pass in the data details and the name of the serial port.
  ET.begin(details(mydata), &mySerial);
  ETopc.begin(details(mydataOPC), &opcSerial);

  if (!SD.begin(10)) {
    Serial.println("SDcard not ready");
    lcd.setCursor(0,3);
    lcd.print("SD Card fail");
    delay(10000);
  }
  else
    Serial.println("SDcard ok");

  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(gpsHigh, OUTPUT); // substitute for +5V
  digitalWrite(gpsHigh, HIGH);  //constantly on
  delay(100);
  pinMode(gpsOnPin, INPUT);        // sets the GPS on off switch as input
  gpsOn = digitalRead(gpsOnPin);     // read the GPS on off switch
  lcd.setCursor(0,1);
  if (gpsOn) lcd.print("Acquiring Satellites");
  else {
    lcd.print("Using RTC...");
    delay(5000);
  }
  Serial.println("Logging started ...");

}

void loop(){

  digitalWrite(gpsCsPin, LOW);
  digitalWrite(opcCsPin, LOW);
  if (gpsOn) {
    digitalWrite(gpsCsPin,HIGH);
    digitalWrite(opcCsPin,LOW);
    mySerial.listen();

    if(ET.receiveData()){   //gps received
      timeStr = String(mydata.tim) + '\t' + String(mydata.nord) + '\t' + String(mydata.east) +
'\t' + Serial.println(mydata.date);
      digitalWrite(gpsCsPin, LOW);
      digitalWrite(opcCsPin,HIGH);
      opcSerial.listen();
      logData(gpsOn);
      mySerial.listen();
    }
  }
  else { //GPS off, use rtc
    now1 = rtc.now();
    String dayStr = String(now1.day());
    if (dayStr.length() == 1) dayStr = '0' + dayStr;
    String monthStr = String(now1.month());
    if (monthStr.length() == 1) monthStr = '0' + monthStr;
    String yearStr = String(now1.year());
    if (yearStr.length() == 1) yearStr = '0' + yearStr;
```

```
    String hourStr = String(now1.hour());
    if (hourStr.length() == 1) hourStr = '0' + hourStr;
    String minStr = String(now1.minute());
    if (minStr.length() == 1) minStr = '0' + minStr;
    String secStr = String(now1.second());
    if (secStr.length() == 1) secStr = '0' + secStr;
    timeStr = dayStr + '.' + monthStr  + '.' + yearStr + ' ' + hourStr + ':' + minStr + ':' +
secStr;

    digitalWrite(opcCsPin,HIGH);
    opcSerial.listen();
    logData(gpsOn);
  }
}

void logData(int gpsOn) {

  unsigned int traw;
  unsigned int hraw;
  double temp;
  double hum;
  int i;
  unsigned char buffer[4];

  //read OPC
  //Serial.println("Waiting for OPC");
  while (!ETopc.receiveData()); //this takes time

  lcd.clear();
  if (gpsOn) {
    Serial.println(mydata.tim);
    Serial.println(mydata.nord);
    Serial.println(mydata.east);
    Serial.println(mydata.date);
    myFile.print(mydata.tim);
    myFile.print('\t');
    myFile.print(mydata.date);
    myFile.print('\t');
    myFile.print(mydata.nord);
    myFile.print('\t');
    myFile.print(mydata.east);
    myFile.print('\t');
    lcd.setCursor(0,0);
    lcd.print("Time: ");
    lcd.print(mydata.tim);
  }
  else {
    Serial.print(now1.day());
    Serial.print('.');
    Serial.print(now1.month());
    Serial.print('.');
    Serial.print(now1.year());
    Serial.print(' ');
    Serial.print(now1.hour());
    Serial.print(':');
    Serial.print(now1.minute());
    Serial.print(':');
    Serial.print(now1.second());
    Serial.print(' ');
    myFile.print(timeStr);
    myFile.print('\t');
    lcd.setCursor(0,0);
    lcd.print("Time: ");
    lcd.print(now1.hour());
    lcd.print(':');
    lcd.print(now1.minute());
    lcd.print(':');
    lcd.print(now1.second());
  }

  for (int i = 0; i<16; i++) {
    Serial.print(mydataOPC.bin[i]);
    Serial.print(' ');
  }
  Serial.print(';');
  Serial.print(mydataOPC.PM10val);
  Serial.print(' ');
  Serial.print(mydataOPC.PM2_5val);
```

```
  Serial.print(' ');
  Serial.print(mydataOPC.PM1val);
  Serial.print(' ');

  for (int i = 0; i<16; i++) {
    myFile.print(mydataOPC.bin[i]);
    myFile.print('\t');
  }
  myFile.print(float(mydataOPC.PM10val)/100.0);
  myFile.print('\t');
  myFile.print(float(mydataOPC.PM2_5val)/100.0);
  myFile.print('\t');
  myFile.print(float(mydataOPC.PM1val)/100.0);
  myFile.print('\t');

  lcd.setCursor(0,1);
  lcd.print("PM10: ");
  lcd.print(float(mydataOPC.PM10val)/100.0);

  // read hyt221
  Wire.beginTransmission(HYT939_ADDR); // transmit to device #44 (0x2c)
  Wire.endTransmission(); // stop transmitting
  //100ms warten
  delay(100);
  //4 Bytes vom Sensor lesen
  Wire.requestFrom(HYT939_ADDR, 4,true);
  i=0;
  while(Wire.available()) {
    char c = Wire.read(); // receive a byte as character
    buffer[i]=c;
    i++;
  }
  traw=buffer[2]*256+buffer[3];
  hraw=buffer[0]*256+buffer[1];
  //Daten laut Datenblatt maskieren
  traw&=0xfffc;
  hraw&=0x3fff;
  traw=traw/4;
  temp=(double)traw/TFACTOR;
  temp=temp-TDELTA;
  hum=(double)hraw/HFACTOR;

  Serial.print(temp);
  Serial.print(' ');
  Serial.println(hum);

  myFile.print(temp);
  myFile.print('\t');
  myFile.print(hum);

  myFile.println();
  myFile.flush();

  lcd.setCursor(0,2);
  lcd.print("T: ");
  lcd.print(temp);
  lcd.print(" H: ");
  lcd.print(hum);
  lcd.setCursor(0,3);
  lcd.print("Batt: ");
  lcd.print(float(analogRead(1))*15.0/1024.0);
}
```

Sketch for the GPS Arduino

```
#include <SoftwareSerial.h>
#include <SoftEasyTransfer.h>
//#include "SD.h"
//#include "SPI.h"
#define LINEBUFSZ 256
#define CalFactor 1
#define PMTK_SET_NMEA_UPDATE_1HZ  "$PMTK220,1000*1F"
#define PMTK_SET_NMEA_UPDATE_01HZ  "$PMTK220,10000*2F"
#define PMTK_SET_NMEA_ALL_MSG "$PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0*28"
#define PMTK_SET_NMEA_RMCSLOW_MSG "$PMTK314,0,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*2D"
#define PMTK_SET_NMEA_RMC_MSG "$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29"
#define gpsCsPin 6
```

```
SoftwareSerial gpsSerial(8, 7);
SoftwareSerial mySerial(4, 5);
SoftEasyTransfer ET;
struct SEND_DATA_STRUCTURE{
  //put your variable definitions here for the data you want to send
  //THIS MUST BE EXACTLY THE SAME ON THE OTHER ARDUINO
  unsigned long tim;
  char nord[10]; //precision of float not sufficient!
  char east[11];
  unsigned long date;
};
SEND_DATA_STRUCTURE mydata;

int cCnt = 0;
char c;
char gpsBuf[LINEBUFSZ];

void setup() {
  pinMode(gpsCsPin, INPUT);

  Serial.begin(9600);
  gpsSerial.begin(9600);
  mySerial.begin(9600);
  ET.begin(details(mydata), &mySerial);
  delay(2000);
  waitFix();
  cCnt = 0;
}

void loop() {
  int bufLen;
  int j;
  char * pch;
  int nCnt;

  cCnt = 0;
  gpsSerial.listen();
  while (1) {
    if (gpsSerial.available()) { // wait for fix
      c = gpsSerial.read();

      if ((c != '\r') && (c != '\n')) {
        gpsBuf[cCnt]=c;
        cCnt++;
      }
      if (c == '\n') {
        gpsBuf[cCnt]=0;
        bufLen = cCnt;
        //Serial.println(bufLen);
        cCnt = 0;
        if (gpsBuf[3] == 'R') { // RMC msg
          if (gpsBuf[18] == 'A') {  //fix
            Serial.println("fix");
            Serial.println(gpsBuf);

            nCnt = 0;
            pch = strtok (gpsBuf,",");
            while (pch != NULL) {
              //Serial.print(nCnt);
              //Serial.print(' ');
              //Serial.println(pch);

              if (nCnt == 1) {
                mydata.tim=(unsigned long) atof(pch);
              }
              else if (nCnt == 3) {
                strncpy(mydata.nord,pch,9);
                mydata.nord[10]='\0';
              }
              else if (nCnt == 5) {
                strncpy(mydata.east,pch,10);
                mydata.east[11]='\0';
              }
              else if (nCnt == 9) {
                mydata.date=(unsigned long) atof(pch);
              }

              pch = strtok (NULL, ",");
```

```
            nCnt++;
          }
          Serial.println(mydata.tim);
          Serial.println(mydata.nord);
          Serial.println(mydata.east);
          Serial.println(mydata.date);
          if (digitalRead(gpsCsPin)== HIGH) {
            Serial.println("Sending");
            mySerial.listen();
            //send the data
            ET.sendData();
            gpsSerial.listen();
          }
          else
           Serial.println("Quiet");
        }
        else {
          Serial.println("no fix");
          Serial.println(gpsBuf);
          waitFix();
        }
      }
    }
  }
 }
}

void waitFix() {
  Serial.println("waitFix");
  gpsSerial.listen();
  Serial.println(PMTK_SET_NMEA_UPDATE_1HZ); //send high output freq
  gpsSerial.println(PMTK_SET_NMEA_UPDATE_1HZ);
  while (1) {
    if (gpsSerial.available()) {
      char c = gpsSerial.read();
      if (c != '\n') Serial.write(c);
      else {
        Serial.write(c);
        break;
      }
    }
  }
  waitCount();
  Serial.println(PMTK_SET_NMEA_ALL_MSG);  //send all msg
  gpsSerial.println(PMTK_SET_NMEA_ALL_MSG);
  while (1) {
    if (gpsSerial.available()) {
      c = gpsSerial.read();
      if (c != '\n') Serial.write(c);
      else {
        Serial.write(c);
        break;
      }
    }
  }
  waitCount();
  cCnt = 0;
  while (1) {
    if (gpsSerial.available()) { // wait for fix
      c = gpsSerial.read();

      if ((c != '\r') && (c != '\n')) {
        gpsBuf[cCnt]=c;
        cCnt++;
      }
      if (c == '\n') {
        gpsBuf[cCnt]=0;
        Serial.println(gpsBuf);
        cCnt = 0;
        if (gpsBuf[3] == 'R') { // RMC msg
          if (gpsBuf[18] == 'A') //fix
            break;
          else
            Serial.println("no fix");
        }
      }
    }
  }
```

```
  waitCount();
  Serial.println(PMTK_SET_NMEA_RMC_MSG); //send rmc msg
  gpsSerial.println(PMTK_SET_NMEA_RMC_MSG);
  while (1) {
    if (gpsSerial.available()) {
      c = gpsSerial.read();
      if (c != '\n') Serial.write(c);
      else {
        Serial.write(c);
        break;
      }
    }
  }
  Serial.println("end wait fix");
}

void waitCount() {
  int waitCnt = 5;

  cCnt = 0;
  while (1) { // check infinitely
    if (gpsSerial.available()) { // wait for fix
      c = gpsSerial.read();

      if ((c != '\r') && (c != '\n')) {
        gpsBuf[cCnt]=c;
        cCnt++;
      }
      if (c == '\n') {
        gpsBuf[cCnt]=0;
        Serial.print("***"); //indicates wait state
        Serial.println(gpsBuf);
        waitCnt--;
        if (waitCnt == 0) break;
        cCnt = 0;
      }
    }
  }
}
```

Sketch for the OPC Arduino

```
#include <SPI.h>
#include <SoftwareSerial.h>
#include <SoftEasyTransfer.h>
#include "Wire.h"

#define opcCsPin 3

unsigned char ret = 0;
byte retVal;
SoftwareSerial opcSerial(4, 5); //rx,tx
SoftEasyTransfer ET;
struct SEND_DATA_STRUCTURE{
  unsigned int bin[16];
  unsigned long PM10val = 0;
  unsigned long PM2_5val = 0;
  unsigned long PM1val = 0;
};
SEND_DATA_STRUCTURE mydata;

void setup() {
  delay(10000); // wait for power to stabilize
  Serial.begin(9600);
  opcSerial.begin(9600);
  ET.begin(details(mydata), &opcSerial);
  SPI.begin();
  Serial.println("Starting OPC");
  boolean OPCon = 0;

  SPI.beginTransaction(SPISettings(750000,MSBFIRST,SPI_MODE1)); //changed to SPI_Mode1 BL
  while (!OPCon)  {
    // Try to turn on the OPC<
    digitalWrite(SS,LOW);
    retVal=SPI.transfer(0x03);
    digitalWrite(SS,HIGH);
    if (retVal==0xF3) {
```

```
      delay(10);
      digitalWrite(SS,LOW);
      retVal=SPI.transfer(0x00);  // turn it ON
      digitalWrite(SS,HIGH);
      if (retVal==0x3) {
        delay(6000);
        OPCon = 1;
        Serial.println("OPC started");
      }
      else {
        Serial.println("OPC not ready");
      }
    }
    else {
      Serial.println("OPC bad response");
    }
    delay(5000);
  }
  Serial.println("start reading samples...");
}

unsigned long buf[31];
unsigned long bufPM[12];

void loop() {

  unsigned int bin[16];
  int i;
  float PM10val, PM2_5val, PM1val;

// Histo
  digitalWrite(SS,LOW);
  retVal=SPI.transfer(0x30);
  digitalWrite(SS,HIGH);
  delay(10);
  digitalWrite(SS,LOW);
  if (retVal==0xF3) {
    for (int i=0; i < 31; i++) {
      delayMicroseconds(20);
      buf[i]=SPI.transfer(0x30);
    }
  }
  digitalWrite(SS,HIGH);
  for (i=0; i< 15; i++) {
    mydata.bin[i] = buf[i*2]+(buf[i*2+1]<<8);
    Serial.print(mydata.bin[i]);
    Serial.print(' ');
  }
  mydata.bin[15] = buf[30];
  Serial.print(mydata.bin[15]);
  Serial.print("; ");
  delay(3000);

// PM
  digitalWrite(SS,LOW);
  retVal=SPI.transfer(0x32);
  digitalWrite(SS,HIGH);
  delay(10);
  digitalWrite(SS,LOW);
  if (retVal==0xF3) {
    for (int i=0; i < 12; i++) {
      delayMicroseconds(20);
      bufPM[i]=SPI.transfer(0x32);
    }
  }
  digitalWrite(SS,HIGH);

  PM10val = calculateFloat(bufPM[8], bufPM[9], bufPM[10], bufPM[11]);
  Serial.print("PM10: ");
  Serial.print(mydata.PM10val);
  PM2_5val = calculateFloat(bufPM[4], bufPM[5], bufPM[6], bufPM[7]);
  Serial.print("\tPM2.5: ");
  Serial.print(mydata.PM2_5val);
  PM1val = calculateFloat(bufPM[0], bufPM[1], bufPM[2], bufPM[3]);
  Serial.print("\tPM1: ");
  Serial.println(mydata.PM1val);

  if (digitalRead(opcCsPin)== HIGH) {
```

```
    mydata.PM10val = (unsigned long) (PM10val*100.0);
    mydata.PM2_5val = (unsigned long) (PM2_5val*100.0);
    mydata.PM1val = (unsigned long) (PM1val*100.0);
    Serial.println("Sending");
    opcSerial.listen();
    //send the data
    ET.sendData();
  }
  else
   Serial.println("Quiet");

  delay (1000);
}

float calculateFloat(byte val0, byte val1, byte val2, byte val3)
{
  // Return an IEEE754 float from an array of 4 bytes (from dhhhagan lib)
  union u_tag {
    byte b[4];
    float val;
  } u;

  u.b[0] = val0;
  u.b[1] = val1;
  u.b[2] = val2;
  u.b[3] = val3;

  return u.val;
}
```

Sources and Links

Arduino Home

https://www.arduino.cc/

External Libraries

http://www.billporter.info/2011/05/30/easytransfer-arduino-library/

https://bitbucket.org/fmalpartida/new-liquidcrystal/wiki/Home

https://github.com/dhhagan/opcn2

Shields

Adafruit Datalogging Shield:

https://www.adafruit.com/product/1141

Adafruit GPS Shield:

https://www.adafruit.com/product/1272

Alphasense OPC

http://www.alphasense.com/index.php/products/optical-particle-counter/

Temperatur & humidity sensor

https://www.ist-ag.com/sites/default/files/DHHYT221_D.pdf

LCD Display

http://anleitung.joy-it.net/wp-content/uploads/2016/09/SBC-LCD20x4-Anleitung.pdf

Battery
https://www.hacker-motor-shop.com/Akkus-und-Akkuzubehoer/TopFuel-EcoX/TopFuel-LiPo-20C-ECO-X-2400mAh-3S.htm

Enclosure
http://www.fibox.com/catalog/151/product/990/2538113_ENG3.html

Coating
http://www.kontaktchemie.com/KOC/KOCproductdetail.csp?product=EMI%2035