HEvL ISET 2e IS

Rapport 3 Electronique application

Station météo sur Arduino nano + communication ordinateur

Tibério GAIDE CHEVRONNAY
14/06/2018

Table des matières

But ..... **Erreur ! Signet non défini.**
Composant 3
Cablage 3
Méthode de programmation 4
Arduino 4
Libraire 4
Point intéressant du code 4
C# 5
Libraire 5
Point intéressant du code 5

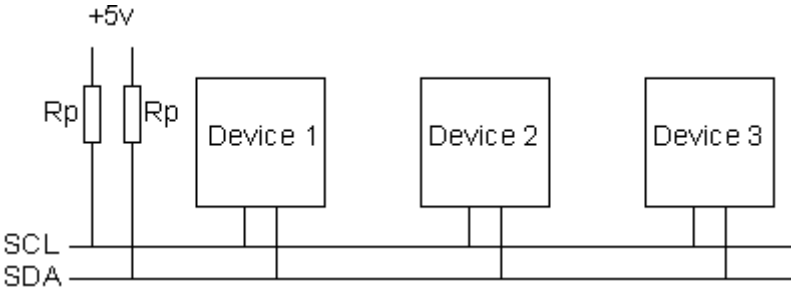
Présentation

A l'aide d'un µContrôleur (Arduino – nano ici), je réalise une station de météo calculant la pression (Pa) , la température (°C) et l'humidité (%).
A l'aide d'un programme sur Arduino, je calcul ces données que j'envoie en communication série vers un programme en c#.

Composant

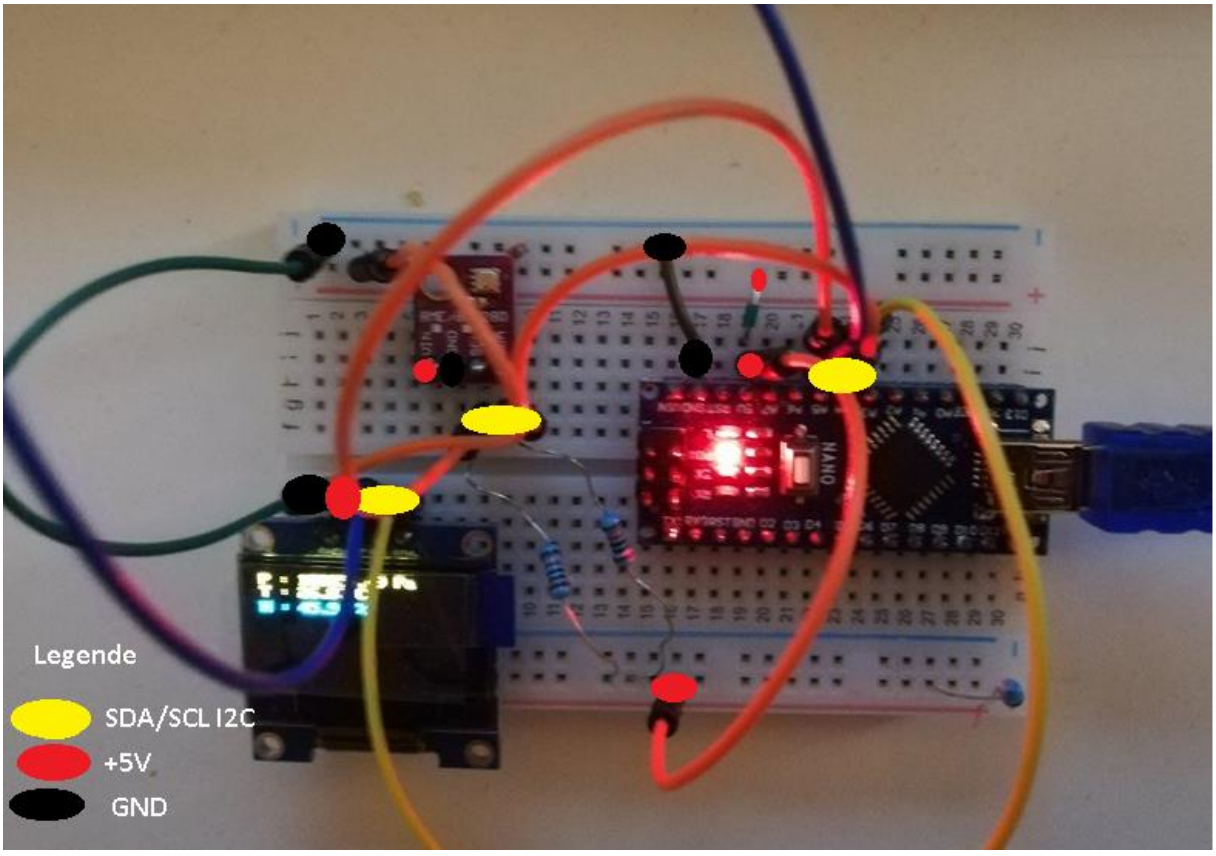
- Arduino-nano
- Afficheur O-led 128*64
- Bme/bmp 280
- Câbles male-male
- 2 Résistances de 10k

Cablage



Les résistances sont en pull-up

Les deux composant sont connectés en I2C sur le port SCL et SDA de l'Arduino (A4 (SDA), A5 (SCL)).



Méthode de programmation

Arduino

Libraire

```
#include "SparkFunBME280.h"
#include "Wire.h"
#include "SPI.h"
//#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

Elle me permet de gérer le calcul du bme 280, l'affichage sur l'écran OLed , l'I2C et la communication série.

Point intéressant du code

Dans notre setup, on configure notre capteur ainsi que notre display. Notons que Baud Rate est de 9600, il sera intéressant de le relever pour tout à l'heure lors du programme C#.

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

      //configuration du capteur
      capteur.settings.commInterface = I2C_MODE;
      capteur.settings.I2CAddress = 0x76;
      capteur.settings.runMode = 3;
      capteur.settings.tStandby = 0;
      capteur.settings.filter = 0;
      capteur.settings.tempOverSample = 1 ;
      capteur.settings.pressOverSample = 1;
      capteur.settings.humidOverSample = 1;

      delay(10);  // attente de la mise en route du capteur. 2 ms minimum
      // chargement de la configuration du capteur
      capteur.begin();
      display.begin(SSD1306_SWITCHCAPVCC, 0x3C);

}
```

L'affichage sur l'écran et le calcul des différents paramètres du capteur.
Notons que vu la place occupé par les lbr, je préfère tirer un peu plus sur le µprocesseur que de gâcher des mémoires en plus.

```
  display.setTextSize(1);
  display.setTextColor(WHITE, WHITE);
  display.setCursor(0,0);
  display.print("P : ");

  display.print(capteur.readFloatPressure());
  display.println(" Pa");

  display.print("T : ");
  display.print(capteur.readTempC()/*, 2*/);
  display.println(" C");

  display.print("H : ");
  display.print(capteur.readFloatHumidity());
  display.println(" %");
```

L'envoie des données en série est donc gérer par un simple Serial.Print(« ») ;

```
Serial.print(capteur.readFloatPressure());
Serial.print(";");
Serial.print(capteur.readTempC());
Serial.print(";");
Serial.print(capteur.readFloatHumidity());
  Serial.print(";");
```

Niveau mémoire je suis tout juste sur le nano.

```
Le croquis utilise 18578 octets (60%) de l'espace de stockage de programmes. Le maximum est de 30720 octets.
Les variables globales utilisent 1620 octets (79%) de mémoire dynamique, ce qui laisse 428 octets pour les variables lo
La mémoire disponible faible, des problèmes de stabilité pourraient survenir.
```

Notons que je n'ai pas rencontrer de disfonctionnement lors de mes tests de l'Arduino-nano.

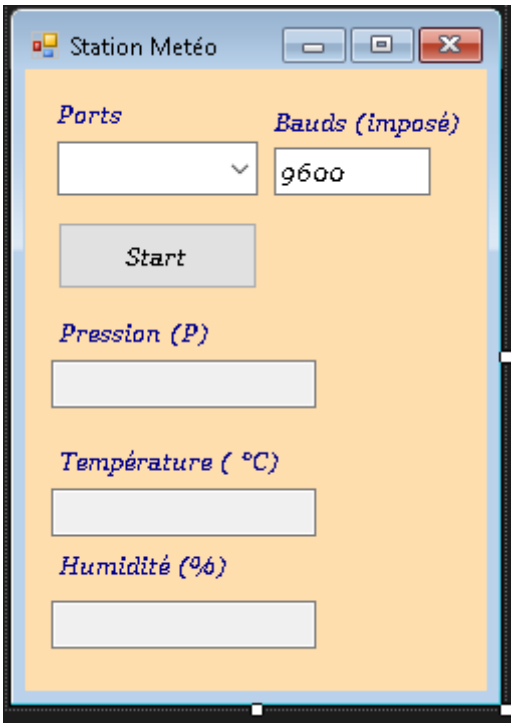
C#

Libraire

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using System.Threading;
```

Point intéressant du code

Je fais ça sur un Win Forms et non une console.



Dans le code, nous retrouvons l'instanciation d'un port Série et des méthodes qui permettront de le mettre en marche.

Il faut d'abord choisir son port, puis ensuite on peut lancer la communication série.

Afin de pouvoir couper la lecture quand bon me semble, j'utilise des threads qui me permette alors de pouvoir arrêter la fonction selon mon bon vouloir via thread.abort() .

La lecture de données et l'affichage se fait ainsi.

```
private void LectureData()
{
    try
    {
        while (calcul.IsAlive)
        {
            string a = SP1.ReadExisting();
            Thread.Sleep(20);
            BeginInvoke(new EventHandler(delegate
            {
                Affichage(a);
            }));

            Application.DoEvents();
        }
    }
    catch (ThreadAbortException abortException)
    {
        Console.WriteLine((string)abortException.ExceptionState);
    }
}
private void Affichage(string s)
{
    if (s.Count() > 0)
    {
        string mess="";
        s= s.Replace('.', ',');

        float Pression, Temperature, Humidite;
        string[] tab = s.Split(';'); // ici on a alors la pression

        Pression = float.Parse(tab[0].ToString());
        Temperature = float.Parse(tab[1]);
        Humidite = float.Parse(tab[2]);

        tbPression.Text = (Pression.ToString());
        tbTemperature.Text = (Temperature.ToString());
        tbHumidite.Text = (Humidite.ToString());
    }
}
private void activerbtnlecture (bool active)
```