How to build an Anemometer with a RS485 MODBUS win sensor for Arduino and ESP8266

Author : Philippe de Craene
dcphilippe@yahoo.fr

Date : September 2019

Manuel version : 1

Program version : 3

The Anemometer will be a part of a bench of measures that will be added to the Wind Turbine MPPT Regulator. This bench of measures will work with a ESP8266, for its Wi-Fi availability.

For the moment, the objective is to find an easy way to implement RS485 on an Arduino Uno, them to adapt it to an ESP8266, the Wemos Lolin D1 mini for instance.

The code result seems very simple and cool, but I spent many and many hours to find a way to get something from this wind sensor.

So I think it will interest everyone that have to implement RS485.

Table of Contents

Introduction

There is many web pages about the RS485 theory; I will not explain it anymore, except that it is a serial communication protocol. One very good explanation is available here :
https://www.cupidcontrols.com/2015/10/software-serial-modbus-master-over-rs485-transceiver/

But the serial pins of the Arduino Uno are used for the console.

To add a serial port, a extra library will be need : SoftwareSerial.h that can be found there :
https://github.com/PaulStoffregen/SoftwareSerial

It is really the simplest way compare to any dedicated RS485 MODBUS library…

List of materials

1. One Arduino Uno / Wemos Lolin D1 mini
2. One RS485 MODBUS wind sensor
3. One MAX485 DIP8 integrated circuit
4. One LCD 1602 with I2C extension
5. One RTS module DS1307 (I2C communication)
6. One SD module

Around the wind sensor and Arduino Uno

Around the RS485 communication

Many cares must be taken to the wind sensor datasheet. It must contain :

- Communication baud rate,
- If any parity bit – normally there is no parity bits as there is a CRC

The RS485 communication is done by 2 wires called A and B. the length of these 2 wires can exceed 30 meters, in this case a 120R resistor placed at the top end is a good idea. There are 2 other wires : GND and the Power supply. In our case it is 12V.

An integrated circuit like MAX485 adapt the “A” and “B” signal to a 5V Rx and Tx understandable by the Arduino Uno.

PARAMETERS	CONTENT
Code	8-bit binary
Data bits	8 bit
Parity bit	No
Stop bit	1 bit
Error checking	CRC (redundant loop code)
Baud rate	2400 bps/ 4800 bps/ 9600 bps can be set factory defaults to 9600 bps

Thanks to the win sensor datasheet, that gives 2 communication examples: a request to the sensor and the answer:

Inquiry Frame

Address Code	Function Code	Start Address	Data Length	CRC_L	CRC_H
0x01	0x03	0x00 0x16	0x00 0x01	0x65	0xCE

Answer Frames

(For example, reading a wind speed of 2.3m/s)

Address Code	Function Code	The Number Of Valid Bytes	Wind Speed Value	CRC_L	CRC_H
0x01	0x03	0x02	0x00 0x17	0xF8	0x4A

Wind speed:

0017 H (hexadecimal) = 23 => Wind Speed = 2.3m/s

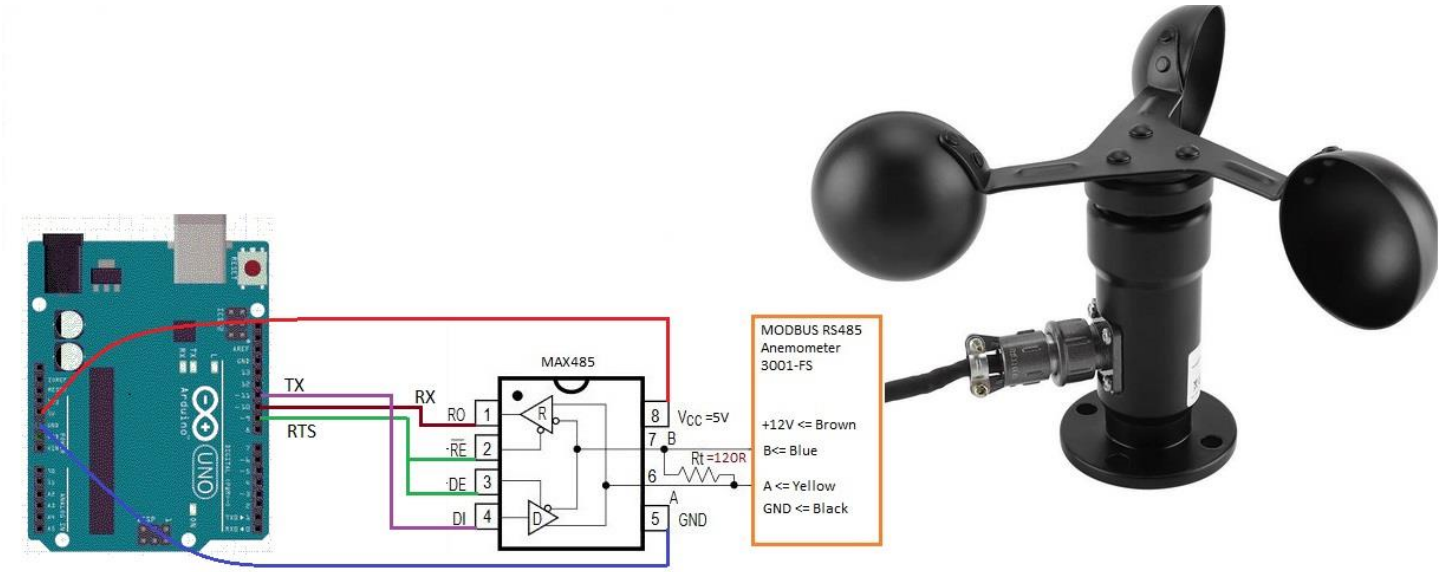
The address Code is the device address.

The function code depends of the device. For mine it means: “give me the wind speed”

The start address and the data length depend of the device…

The CRC depends of all above.

The diagram



The test program

```
/*
Anemometer with a RS485 wind sensor

from an idea of https://arduino.stackexchange.com/questions/62327/cannot-read-modbus-data-repetitively
https://www.cupidcontrols.com/2015/10/software-serial-modbus-master-over-rs485-transceiver/

_____________________________________________________________________
|                                                                    |
|       author : Philippe de Craene <dcphilippe@yahoo.fr             |
|       Any feedback is welcome                                      |
|                                                                    |
_____________________________________________________________________

Materials :
• 1* Arduino Uno R3 - tested with IDE version 1.8.7 and 1.8.9
• 1* wind sensor - RS485 MODBUS protocol of communication
• 1* MAX485 DIP8

Versions chronology:
version 1 - 7 sept  2019   - first test

*/

#include <SoftwareSerial.h>  // https://github.com/PaulStoffregen/SoftwareSerial

#define RX        10    //Serial Receive pin
#define TX        11    //Serial Transmit pin
#define RTS_pin    9    //RS485 Direction control
#define RS485Transmit    HIGH
#define RS485Receive     LOW

SoftwareSerial RS485Serial(RX, TX);

void setup() {

  pinMode(RTS_pin, OUTPUT);

  // Start the built-in serial port, for Serial Monitor
  Serial.begin(9600);
  Serial.println("Anemometer");

  // Start the Modbus serial Port, for anemometer
  RS485Serial.begin(9600);
  delay(1000);
}

void loop() {

  digitalWrite(RTS_pin, RS485Transmit);     // init Transmit
  byte Anemometer_request[] = {0x01, 0x03, 0x00, 0x16, 0x00, 0x01, 0x65, 0xCE}; // inquiry frame
  RS485Serial.write(Anemometer_request, sizeof(Anemometer_request));
  RS485Serial.flush();

  digitalWrite(RTS_pin, RS485Receive);      // Init Receive
  byte Anemometer_buf[8];
  RS485Serial.readBytes(Anemometer_buf, 8);

  Serial.print("wind speed : ");
  for( byte i=0; i<7; i++ ) {
  Serial.print(Anemometer_buf[i], HEX);     // this is for test only
  Serial.print(" ");
  }
  Serial.print(" ==> ");
  Serial.print(Anemometer_buf[4]);          // here is the result
  Serial.print(" m/s");
  Serial.println();
  delay(100);

}
```

Upgrade to ESP8266

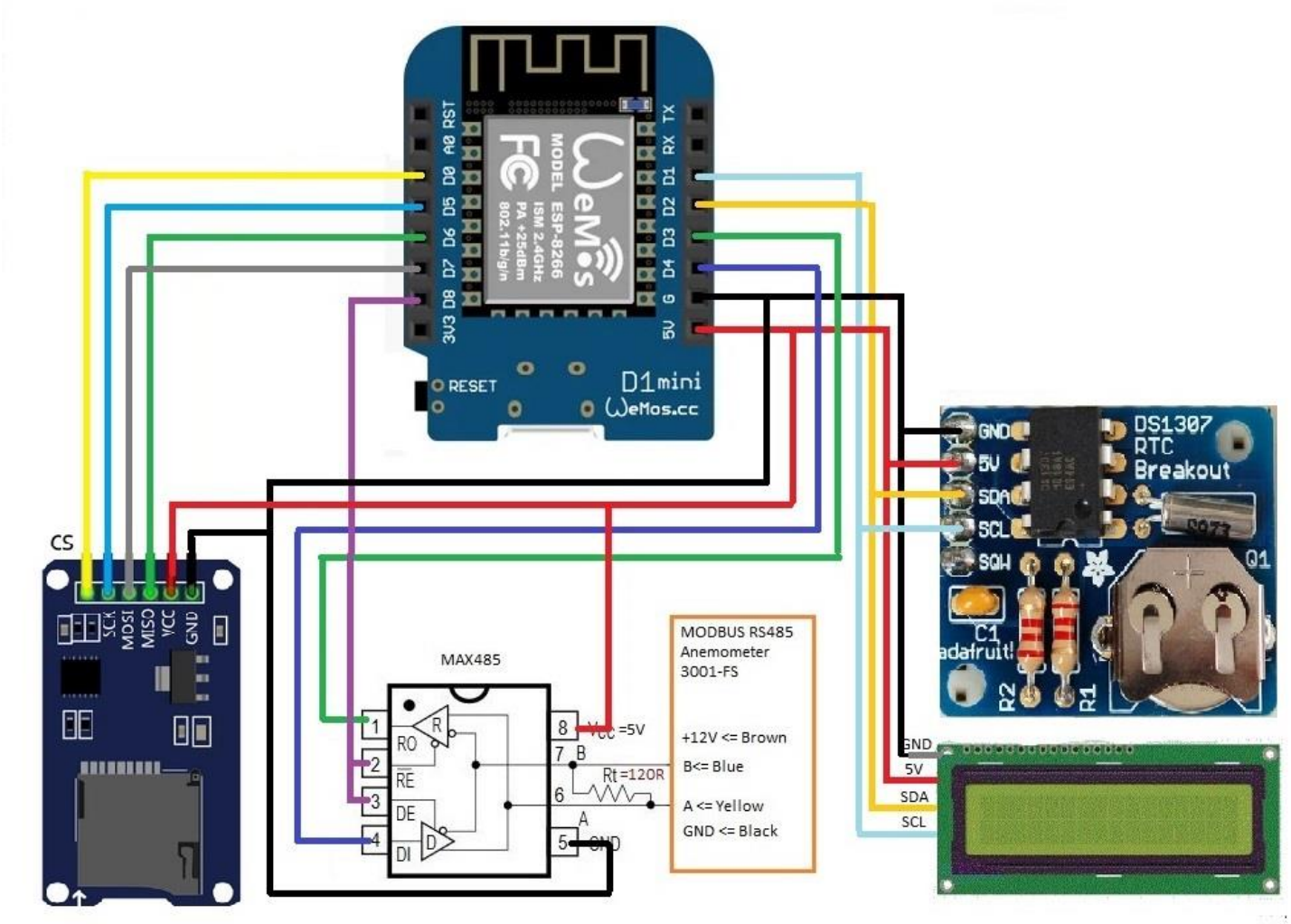
The advantage of the ESP8266 is the Wifi ! It is then possible to get a look anywhere in the world as the ESP8266 can be a mini web server!

It is very easy to move a code from an Arduino Uno to an ESP8266, full explanation is available here: https://randomnerdtutorials.com/getting-started-with-esp8266-wifi-transceiver-review

More over, it can be a good idea to store wind speed data on a SD card for statistics. In this case, we must get the date and the time.

The number of the inputs/outputs

Wemos D1 mini Anemometer diagram



The I2C allows several I2C components on the same 2 wires, here ths LCD and the RTC.

Some people will scream when they will see that I mix 5V supplied devices with the ESP8266 which works with a supply of 3.3V. In fact, you will definitively burn your esp8266 if you supply it with 5V – there is a 3.3V supply regulator on the Wemos D1 mini module – But, my anemometer is still working weeks after I built it. So it is possible to plug on an ESP8266 any 5V devices, when the 5V is only on the inputs / outputs of the microcontroller. In another words the ESP8266 is 5V tolerant, even if it must not been supplied over 3.3V. For the sceptics they can read this: https://www.ba0sh1.com/blog/2016/08/03/is-esp8266-io-really-5v-tolerant

In fact I get an IDE error when I download the code. After several research I've found that D8 must be at 0 during download. D8 is the default CS pin for the SD card. But I've seen that D8 stays stuck at Vcc… So I invert D8 with D0. However it is not enough. So I finally decide to power off the 5V supply to all components – except ESP8266 – each time I download the code, then I switch on them back again.

ESP8266 Anemometer code

Please note that the above code can run on Arduino once all internet instruction are deleted and pin connection adapted.

```
/*
Anemometer from an idea of https://arduino.stackexchange.com/questions/62327/cannot-read-modbus-data-repetitively
https://www.cupidcontrols.com/2015/10/software-serial-modbus-master-over-rs485-transceiver/

_____________________________________________________________________
|                                                                    |
|       author : Philippe de Craene <dcphilippe@yahoo.fr             |
|       Any feedback is welcome                                      |
|                                                                    |
_____________________________________________________________________

Materials :
• 1* Wemos D1 mini - tested with IDE version 1.8.7 and 1.8.9
• 1* wind sensor - RS485 MODBUS protocol of communication
• 1* MAX485 DIP8
• 1* RTC 1307
• 1* LCD1602 with I2C extension
• 1* SD card


Versions chronology:
version 1   - 7 sept 2019   - first test on Arduino Uno
Version 3   - 9 sept 2019   - ESP8266 based with RTC and SD card


ESP8266 pinup :

D1 => SCL for LCD1602 and DS1307 (Arduino A5)
D2 => SDA for LCD1602 and DS1307 (Arduino A4)

D3 => Rx = RO of MAX485 - pin 1
D4 => Tx = DI of MAX485 - pin 4
D8 => RTS = RE/DE of MAX485 - pins 2&3

D5 => SCK for SDcard  (Arduino 13)
D6 => MISO for SDcard (Arduino 12)
D7 => MOSI for SDcard (Arduino 11)
D0 => CS for SDcard (SDcard Arduino shield 10) CS should be in D8 but must be at 0
      during boot, but stay stuck at Vcc....

*/

#include <ESP8266WiFi.h>        // https://github.com/esp8266/Arduino
#include <WiFiUdp.h>
#include <ESP8266WebServer.h>   // required pour WifiManager.h
#include <DNSServer.h>          // required pour WifiManager.h
#include <WiFiManager.h>        // https://github.com/tzapu/WiFiManager
#include <ArduinoOTA.h>         // https://github.com/marcudanf/arduinoOTA
#include <TimeLib.h>            // https://github.com/PaulStoffregen/Time
#include <DS1307RTC.h>          // https://github.com/PaulStoffregen/DS1307RTC
#include <SD.h>                 // yet include : https://github.com/adafruit/SD
```

```
#include <SoftwareSerial.h>    // https://github.com/PaulStoffregen/SoftwareSerial
#include <LiquidCrystal_I2C.h> // https://github.com/lucasmaziero/LiquidCrystal_I2C

#define RX        D3    // Soft Serial RS485 Receive pin
#define TX        D4    // Soft Serial RS485 Transmit pin
#define RTS       D8    // RS485 Direction control
#define RS485Transmit    HIGH
#define RS485Receive     LOW
#define CS        D0    // CS for SDcard

SoftwareSerial RS485Serial(RX, TX);    // additional serial port for RS485
WiFiServer server(80);                 // web server on www default port 80
LiquidCrystal_I2C lcd(0x27, 16, 2);    // Set the LCD address to 0x27 for a 16 chars and 2 line
display
File dataFile;                         // initialisation of the SD card

// NTP server declaration
int TZ = 2;                            // timezone
unsigned int localPort = 2390;         // local port to listen for UDP packets
/* Don't hardwire the IP address or we won't get the benefits of the pool.
    Lookup the IP address for the host name instead */
//IPAddress timeServer(129, 6, 15, 28);   // time.nist.gov NTP server
IPAddress timeServerIP;                  // time.nist.gov NTP server address
const char* ntpServerName = "time.nist.gov";
const int NTP_PACKET_SIZE = 48;         // NTP time stamp is in the first 48 bytes of the message
byte packetBuffer[ NTP_PACKET_SIZE];    // buffer to hold incoming and outgoing packets
WiFiUDP udp;                            // A UDP instance to let us send and receive packets over
UDP

// Variables declaration
float Anemometer = 0, memo_Anemometer = 0;
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday"};
bool afficher = true;        // affichage sur LCD
unsigned int delai = 2000;   // delay between 2 measures in ms
unsigned int memo_actuel = 0;

//
// SETUP
//_____________________________________________________________________________________________

void setup() {

  pinMode(RTS, OUTPUT);
  pinMode(CS, OUTPUT);

// Start the built-in serial port, for Serial Monitor
  Serial.begin(9600);
  Serial.println("Anemometer");

// Start the Modbus serial Port, for anemometer
  RS485Serial.begin(9600);
  delay(100);

// initialize the LCD
  lcd.begin();                  // Init with pin default ESP8266 or ARDUINO
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Anemometer");
  lcd.setCursor(0, 1);

// see if the RTC is present and is set
  tmElements_t tm;
  if (RTC.read(tm)) {
    Serial.print("Ok, Time = ");
    Serial.print(tm.Hour); Serial.write(':');
    Serial.print(tm.Minute); Serial.write(':');
    Serial.print(tm.Second);
    Serial.print(", Date (D/M/Y) = ");
    Serial.print(tm.Day);
    Serial.write('/');
    Serial.print(tm.Month);
    Serial.write('/');
    Serial.print(tmYearToCalendar(tm.Year));
    Serial.println();
    setSyncProvider(RTC.get);     // to get the time from the RTC
    lcd.print("Time: OK    ");
  }
  else {
    if (RTC.chipPresent()) Serial.println("The DS1307 is stopped.  Please set time");
    else Serial.println("DS1307 read error!  Please check the circuitry.");
    lcd.print("Time: FAIL  ");
  }
  Serial.println();
  delay(1000);
  lcd.setCursor(0, 1);
```

```
// see if the card is present and can be initialized:
  if (!SD.begin(CS)) Serial.println("Card failed, or not present");
  else { Serial.println("card initialized."); }
  // Open up the file we're going to log to!
  dataFile = SD.open("datalog.txt", FILE_WRITE);
  if (!dataFile) {
    Serial.println("datalog.txt error !");
    lcd.print("SDcard: FAIL");
  }
  else {
    Serial.println(" datalog.txt ready ...");
    lcd.print("SDcard: OK  ");
  }
  Serial.println();
  delay(1000);
  lcd.setCursor(0, 0);
  lcd.print("WiFi is     ");
  lcd.setCursor(0, 1);
  lcd.print("starting ...");

// AP will start if no wifi identifiers in memory or wrong identification
// AP can be accessed from ssid "AutoConnectAP" then IP address 192.168.4.1 within 150 seconds
// in cas of unsuccess after 150 seconds the wifi will not be defined
// for local intialization. Once its business is done, there is no need to keep it around
  WiFiManager monwifi;
  monwifi.setConfigPortalTimeout(180);    // 150 seconds timeout
  byte i = 0;                          // counter of request to wifi connexion
  byte imax = 10;                      // max number of request to wifi connexion
// fetches ssid and pass from eeprom and tries to connect. If it does not connect it starts
// an access point with the specified name and goes into a blocking loop awaiting configuration
  if(!monwifi.autoConnect("AutoConnectAP")) Serial.println("non paramétré");
  else {
// Connect to Wi-Fi network with SSID and password
    Serial.print("connexion au Wifi en cours ");
    while( (WiFi.status() != WL_CONNECTED) && i < imax ) {
      i++;
      delay(500);
      Serial.print(".");
    }
  }     // end of else monwifi.autoConnect

// if wifi is connected
  if( i < imax ) {
// show IP address
    Serial.println();
    Serial.println("Wifi connecté.");
    Serial.print("Address IP : ");
    Serial.println(WiFi.localIP());
    lcd.setCursor(0, 1);
    lcd.print("started !   ");

// 2 of the 3 lines of code for OTA
    ArduinoOTA.setHostname("Anemometer");  // device name
    ArduinoOTA.begin();                    // OTA initialisation

// udp service startup
    Serial.println("Starting UDP");
    udp.begin(localPort);
    Serial.print("Local port: ");
    Serial.println(udp.localPort());

  }  // end of test i
  else {
    Serial.println();
    Serial.println("pas de réseau wifi");
    Serial.println("Récupération de l'heure en local");
    lcd.setCursor(0, 1);
    lcd.print("not started ");
    delay(1000);
  }
  server.begin();             // web server startup
 // getNTP();                  // NTP function to get the internet date and time
  lcd.setCursor(0, 0);
  lcd.print("Anemometer");
  lcd.setCursor(0, 1);

}  // end of setup

//
// LOOP
//_____________________________________________________________________________________________

void loop() {

// The 3rd code line for OTA
  ArduinoOTA.handle();
```

```
// to display data on a html page
  webserver();

// Daily time update
  if( hour() == 1 && minute() == 0 && second() < 2 ) getNTP();

// The above of the loop is done every waitdelay seconds only
  unsigned int actuel = millis();
  if( actuel - memo_actuel < delai ) return;
  memo_actuel = actuel;

// RS485 MODBUS Request and Receive with the anemometer
  byte Anemometer_buf[8];
  Anemometer_buf[1] = 0;
  while( Anemometer_buf[1] != 0x03 ) {    // if received message has an error
    // MODBUS Tramsmit by sending a request to the anemometer
    digitalWrite(RTS, RS485Transmit);     // init Transmit
    byte Anemometer_request[] = {0x01, 0x03, 0x00, 0x16, 0x00, 0x01, 0x65, 0xCE}; // inquiry frame
    RS485Serial.write(Anemometer_request, sizeof(Anemometer_request));
    RS485Serial.flush();
    // MODBUS Reception of the anemometer's answer
    digitalWrite(RTS, RS485Receive);      // init Receive
    RS485Serial.readBytes(Anemometer_buf, 8);
    // data treatment
    Serial.print("wind speed : ");
    for( byte i=0; i<7; i++ ) {
      Serial.print(Anemometer_buf[i], HEX);
      Serial.print(" ");
    }
    Serial.print(" ==> ");
    Serial.print(Anemometer_buf[4]);
    Serial.print(" /10 m/s");
    Serial.println();
    delay(500);
  }   // end of while
  memo_Anemometer = Anemometer;
  Anemometer = Anemometer_buf[4]/10.0;
  lcd.setCursor(0, 1);
  lcd.print(Anemometer);
  lcd.print(" m/s      ");

// Store on SDcard
  if( Anemometer != memo_Anemometer ) {  // if wind speed change
    String dataString = "";              // initialisation d'une chaine de caractères
    dataString += String(daysOfTheWeek[weekday()-1]);
    dataString += ";";
    dataString += String(day(), DEC);
    dataString += ";";
    dataString += String(month(), DEC);
    dataString += ";";
    dataString += String(year(), DEC);
    dataString += ";";
    dataString += String(hour(), DEC);
    dataString += ";";
    dataString += String(minute(), DEC);
    dataString += ";";
    dataString += String(second(), DEC);
    dataString += ";";
    dataString += String(Anemometer);
    dataString += ";";
    dataFile.println(dataString);     // record data on SD card
    dataFile.flush();                 // clean buffer
    Serial.println(dataString);       // show record on console
  }   // end test Anemomter
}     // end of loop

//
// webserver : display data on html page
//_____________________________________________________________________________________

void webserver() {

  WiFiClient client = server.available();   // Listen for incoming clients
  if( client ) {                            // If a new client connects,
    Serial.println("Nouveau client.");      // print a message out in the serial port

    String entete = client.readStringUntil('\r'); // read the header until \r
    Serial.print("header received => ");
    Serial.println(entete);

    String etat_afficher[] = {"non", "oui"};
    if( entete.indexOf("GET /?A=0") >= 0) afficher = false;
    if( entete.indexOf("GET /?A=1") >= 0) afficher = true;
    Serial.print("\n Etat de l'affichage du LCD : ");
    Serial.println(afficher);
    if( afficher == true ) lcd.backlight();
    else lcd.noBacklight();
```

```
    client.flush();                          //nettoie le tampon...
    // HTTP header
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:text/html");
    client.println("Connection: close");
    client.println();
    // Display the HTML web page with every 4 seconds refraish
    client.println("<!DOCTYPE html><html lang=fr-FR>");
    client.println("<head><meta http-equiv='refresh' content='4'/>");
    client.println("<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
    client.println("<link rel=\"icon\" href=\"data:,\">");
    // CSS to style the on/off buttons
    client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto;
text-align: center;}");
    client.println(".button { background-color: #8A0808; border: none; color: white; padding: 16px
40px;");
    client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}");
    client.println(".button2 {background-color: #32CD99;}");
    client.println(".button3 {background-color: #08298A;}</style></head>");
    // Web Page Heading
    client.println("<body><h1>An&eacute;mometre chez Fifi</h1>");
    String Minutes = "0";
    if( minute() < 10 ) Minutes += String(minute());
    else Minutes = String(minute());
    client.println("<p><h3>Il est " + String(hour()) + "h" + Minutes + " et " + String(second()) +
" secondes;</h3></p>");
    client.println("<HR size=2 align=center width=\"80%\">");
    client.println("<p><h3>Vitesse du vent " + String(Anemometer) + " m/s</h3></p>");
    client.println("<HR size=2 align=center width=\"80%\">");
    client.println("<p><h2>Affichage : " + etat_afficher[afficher] +"</h2></p>");
    client.println("<FORM>");
    client.println("<INPUT type=\"radio\" name=\"A\" value=\"1\">Allumer");
    client.println("<INPUT type=\"radio\" name=\"A\" value=\"0\">Eteindre");
    client.println("<INPUT class=\"button button3\" type=\"submit\" value=\"Actualiser\"></FORM>");
    client.println("</BODY></center></html>");
    client.println();              // The HTTP response ends with another blank line

    Serial.println("Fin de transmission web - Client disconnected.");
    Serial.println("");
  }  // end of client
}   // end of webserver

//
// getNTP : to get date and time from internet
//_____________________________________________________________________________________

void getNTP() {

  byte i = 0;             // NTP request counter
  byte imax = 40;         // max number of request

  WiFi.hostByName(ntpServerName, timeServerIP);  // get a random server from the pool

  do {
     i++;
     Serial.print("sending NTP packet... ");
     Serial.println(i);
     memset(packetBuffer, 0, NTP_PACKET_SIZE);   // set all bytes in the buffer to 0
     // Initialize values needed to form NTP request
     packetBuffer[0] = 0b11100011;   // LI, Version, Mode
     packetBuffer[1] = 0;            // Stratum, or type of clock
     packetBuffer[2] = 6;            // Polling Interval
     packetBuffer[3] = 0xEC;         // Peer Clock Precision
     // 8 bytes of zero for Root Delay & Root Dispersion
     packetBuffer[12] = 49;
     packetBuffer[13] = 0x4E;
     packetBuffer[14] = 49;
     packetBuffer[15] = 52;
     // all NTP fields have been given values, now you can send a packet requesting a timestamp:
     udp.beginPacket(timeServerIP, 123);       // NTP requests are to port 123
     udp.write(packetBuffer, NTP_PACKET_SIZE);
     udp.endPacket();
     delay(1000);                              // wait to see if a reply is available
  } while(!udp.parsePacket() && i<imax);

  if( i<imax ) {                      // We've received a packet, read the data from it
    udp.read(packetBuffer, NTP_PACKET_SIZE);     // read the packet into the buffer

    //the timestamp starts at byte 40 of the received packet and is four bytes,
    // or two words, long. First, esxtract the two words:
    unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
    unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
    // combine the four bytes (two words) into a long integer
    // this is NTP time (seconds since Jan 1 1900):
    unsigned long secsSince1900 = highWord << 16 | lowWord;
    Serial.print("Seconds since Jan 1 1900 = ");
```

```
    Serial.println(secsSince1900);

    // now convert NTP time into everyday time:
    Serial.print("Unix time = ");
    // Unix time starts on Jan 1 1970. In seconds, that's 2208988800:
    const unsigned long seventyYears = 2208988800UL;
    // subtract seventy years:
    unsigned long epoch = secsSince1900 - seventyYears;
    // print Unix time:
    Serial.println(epoch);

    // to see if it is summer or winter time
    int mois = month();
    int jour = day();
    int joursemaine = weekday();
    if( mois > 3 || mois < 10
      || (mois == 3  && (jour - joursemaine) > 22 )
      || (mois == 10 && (jour - joursemaine) < 23 ) ) TZ = 2;
    else TZ = 1;                    // heure d'hiver
    RTC.set(epoch + TZ*3600);
    setTime(epoch + TZ*3600);       // date and time adjust
  }
  else setSyncProvider(RTC.get);    // the function to get the time from the RTC
}  //   end of getNTP
```

Enjoy !