Third example: logging time stamped data to file

by Mauri Favaron

Introduction

In previous examples we have seen how it is possible to construct a very simple temperature and relative humidity reader, and how to add it a time stamp generator making some sense.

We concentrated on *raw* data, that is, data as coming directly from a sensor. In out case the "sensor" is quite a complex device with an on-board micro-controller, performing actual reads from sensor devices, converting them directly to physical units, and making them available for read as digital data.

So, much of what really happens at sensor-level is hidden between the programming of the DHT-22 "sensor" device. This may or may not be good - in some applications it is, as people is not concerned about the details of the measurement process, and simply trust the sensors; in other situations, a closer view would be welcome, as for example when the temperature measurement is made within some medical device, or more generally in which temperature and relative humidity data are of great value.

Now, curiosity get stronger on undersaìtanding what kind of data are really produced by the sensor (and received by the reader). But, we have a practical difficulty: we have no easy way to save the data somewhere on disk, in order to analyze them.

Sure we could, in principle, copy the serial output window to the clipboard of the computer we're using to program and manage the logger, but this procedure is quite risky and error prone. Better would be if data are actually written to some long term memory.

The SD-card solution

The hardware platform we're using in fact allows writing data to a long-term memory: an SD-card.

To make (for the moment) things simple, we can decide to write data to a fixed-name file. If two data gatherings will be made, the second will overwrite the first. But, we'll not have to deal with the nitty-gritty details of file naming (we'll do also this, but in a following example).

Storing data to an SD card is easy, using the Arduino standard SD library: the interface it exposes very closely resembles that of Serial, plus functions to open/create the output file and close it; of course, functions to read from an SD file, but this is not something we'll use.

By writing raw data to the SD card, we complete the concept data logger to the point we may begin to deal with interesting user-oriented "scientific" aspects, and not just Arduino coding (as interesting and nice it may be): the data writer constitutes the basis for future developments.

So a bit of additional patience is welcome: but, stay assured, we've suffered a bit until now, only to see more interesting things in the next future.

The code

Here is the code of the SD card raw temperature and relative humidity data writer. Code text is (I hope) straightforward, especially if you consider the preceding examples.

```
// Settable (but still "very wrong") temperature reader.
//
// This Arduino sketch is part of the Hypatia project.
//
// Parts:
//
// - An Arduino (an "Uno" is just sufficient; I'm using a Feather M0
AdaLogger, by AdaFruit)
// - A DHT-22 temperature and relative humidity sensor (good for indoor,
mainstream accuracy)
// - A switch with two stable open-close states (or a pin header)
// - A breadboard
//
// Libraries (if you have not yet installed):
//
// - DHT sensor library (from AdaFruit)
// - Adafruit Unified Sensor library (it may be necessary to install
//   this dependency of DHT lib by hand).
// - RTCZero (for RTC functionality)
// - SD card file write functionality.
//
// What you may expect from this sketch:
//
// 1) A simple data acquisition with time stamping example
//
// 2) An example on how to read safely a string from the USB-emulated
serial port
//
// 3) Indirectly, a test that your DHT-22 and the RTC
//
// 4) A way to change the system behavior based on a switch
//
// 5) How-to-read reliably from a serial port
//
// 6) How-to-write to an SD card.
//
// This is open-source software, covered by the MIT license.
//
//
// Written by: Mauri Favaron
```

```
// Library imports:
#include <DHT.h>      // DHT-11, DHT-22 temp/relh sensors
#include <RTCZero.h>  // Accessing the logger's Cortex-M0+ RTC functions
#include <SD.h>       // SD card reader-writer
#include <SPI.h>      // Used by the SD card reader-writer

// Global symbols, referring to SD card
#define cardChipSelect 4
File dataFile;
// No need to declare SD explicitly: this is made by "SD.h" include file.

// Global variable, referring to the DHT-22 used in this project
#define DHT_PIN 6
#define DHT_TYPE DHT22
DHT dht(DHT_PIN, DHT_TYPE);

// Global variable, referring to the Cortex-M0+ RTC
RTCZero rtc;

// Global variables, containing the values of temperature and
// relative humidity just as coming from the DHT-22 sensor
float rTemperature;
float rRelHumidity;

// Mode selector, and mode-related data (date and time)
#define PROGRAMMING_PIN 12

// Buffer, for reading initial configuration
#define MAX_CHAR 128
char buffer[MAX_CHAR];
int  idx;

// Get the number of pending character over serial port
int Serial_available(const int port=0) {
  int numCharsWaiting = 0;
  switch(port) {
    case(0):
      numCharsWaiting = Serial.available();
      break;
    case(1):
      numCharsWaiting = Serial1.available();
      break;
  }
  return(numCharsWaiting);
}

char Serial_read(const int port=0) {
  char ch;
```

```
  switch(port) {
    case(0):
      ch = Serial.read();
      break;
    case(1):
      ch = Serial1.read();
      break;
  }
  return(ch);
}

// Read a string from serial USB line, until its terminating character.
// This function replaces the standard Serial.readBytesUntil(), which I
// wasn't able to convince yielding characters - surely I did something,
// but wasn't able understanding what from the documentation. This routine
// works instead, provided it is preceded by a wait-for-first character
// command.
void receiveUntilLineEnd(const int port=0, const char terminator = '\n') {

  char        ch;                     // Character received
  size_t      n   = sizeof(buffer);  // Number of characters available
  boolean     lineEnd = false;

  // Get pending characters
  while (Serial_available(port) > 0 && lineEnd == false) {

    // Get one pending character
    ch = Serial_read(port);

    // Add character to string, if not the terminator; otherwise,
    // declare the string complete.
    if (ch != terminator) {
      buffer[idx] = ch;
      idx++;
      if (idx >= n) {
        idx = n - 1;
      }
    }
    else {
      // terminate the string
      buffer[idx] = '\0';
      idx = 0;
      lineEnd = true;
    }

  }

}
```

```
// Print a number with leading zero, if 2 digits; if more than 2 digits,
print as it is
void print2digits(int number) {
  if (number < 10) {
    Serial.print("0"); // print a 0 before if the number is < than 10
  }
  Serial.print(number);
}
void write2digits(int number) {
  if (number < 10) {
    dataFile.print("0"); // print a 0 before if the number is < than 10
  }
  dataFile.print(number);
}


void blink(const int n) {
  for(int i = 0; i < n; i++) {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(100);
    digitalWrite(LED_BUILTIN, LOW);
    delay(100);
  }
}


// Initialization
void setup() {

  // Setup sensors
  dht.begin();        // Clean-up and start DHT-22

  // Initialize serial port preliminarily (will be refined, if in
programming mode)
  Serial.begin(9600); // Will be used to monitor the system activity

  // Start SD card
  if(!SD.begin(cardChipSelect)) {
    Serial.println("SD Card not found or not working");
  }
  char fileName[15];
  strcpy(fileName, "TestFile.csv");
  dataFile = SD.open(fileName, FILE_WRITE);
  if(!dataFile) {
    Serial.println("Data files not opened.");
  }
  dataFile.println("date.time,Temp,RelH");
```

```
  // Setup internal LED so that it may be used to inform about the
  // programming mode on start
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);

  // Setup and read programming mode switch
  pinMode(PROGRAMMING_PIN, INPUT_PULLUP);
  int iProgState = digitalRead(PROGRAMMING_PIN);
  if(iProgState == 1) {
    // Wait for Arduino IDE terminal window to open
    // and reset serial port circuitry, allowing users to
    // see the informative messages
    while(!Serial) {}
  }
  Serial.print("Programming switch reading:");
  Serial.println(iProgState);

  // Setup serial, taking programming mode into account
  if(iProgState == HIGH) {

    // Inform we're in programming mode
    blink(5);

    // Get programming
    Serial.setTimeout(65535L);
    while(true) {

      // Clean serial buffer before to proceed
      for(int i = 0; i < MAX_CHAR; i++) buffer[i] = '\0';
      idx = 0;

      // Ask interactively the user to provide a date and time for
      // initializing the RTC.
      Serial.println("Please enter current (local, no time saving) date and
time");
      Serial.println("in ISO form (e.g. 2018-03-08 10:11:12) >> ");
      while(Serial.available() <= 0);   // Wait until a character becomes
available
      receiveUntilLineEnd(0, '\n');

      // Check the line just read has the correct number of characters (19
for an ISO string)
      int len = 0;
      for(int i = 0; i < MAX_CHAR+1; i++) {
        if(buffer[i] != '\0') {
          len++;
        }
        else {
          break;
```

```
    }
  }
  if(len == 19) {

    // Extract all parts of date-time to strings on their own
    char sYear[5];
    char sMonth[3];
    char sDay[3];
    char sHour[3];
    char sMinute[3];
    char sSecond[3];
    int i, j;
    j=0;
    for(i = 0; i < 4; i++) {
      sYear[i] = buffer[j];
      j++;
    }
    sYear[4] = '\0';
    j=5;
    for(i = 0; i < 2; i++) {
      sMonth[i] = buffer[j];
      j++;
    }
    sMonth[3] = '\0';
    j=8;
    for(i = 0; i < 2; i++) {
      sDay[i] = buffer[j];
      j++;
    }
    sDay[3] = '\0';
    j=11;
    for(i = 0; i < 2; i++) {
      sHour[i] = buffer[j];
      j++;
    }
    sHour[3] = '\0';
    j=14;
    for(i = 0; i < 2; i++) {
      sMinute[i] = buffer[j];
      j++;
    }
    sMinute[3] = '\0';
    j=17;
    for(i = 0; i < 2; i++) {
      sSecond[i] = buffer[j];
      j++;
    }
    sSecond[3] = '\0';
```

```
        // Try converting parts of date-time to numbers, and check they
more or less make sense
        byte iYear   = atoi(sYear) % 100;
        byte iMonth  = atoi(sMonth);
        byte iDay    = atoi(sDay);
        byte iHour   = atoi(sHour);
        byte iMinute = atoi(sMinute);
        byte iSecond = atoi(sSecond);
        iYear   = constrain(iYear, 1, 9999);
        iMonth  = constrain(iMonth, 1, 12);
        iDay    = constrain(iDay, 1, 31);
        iHour   = constrain(iHour, 0, 23);
        iMinute = constrain(iMinute, 0, 59);
        iSecond = constrain(iSecond, 0, 59);

        // Set the Cortex M0+ RTC to reflect the desired date and time
        rtc.begin();
        rtc.setYear(iYear);
        rtc.setMonth(iMonth);
        rtc.setDay(iDay);
        rtc.setHours(iHour);
        rtc.setMinutes(iMinute);
        rtc.setSeconds(iSecond);

        // Resume initialization code, exiting from current while() loop
        break;

      }
      else {
        Serial.println("The string just read is not an ISO date-time:
please repeat");
      }
    }
  }
  else{

    // Inform the user we're about to start immediately
    blink(4);

  }

}

void loop() {

  // Wait some time (the amount should be larger than the interval
  // the slowest sensor takes to get a new reading; 1s is sufficient
  // for the DHT-22.
  delay(1000);
```

```
  // Get readings
  rTemperature = dht.readTemperature();
  rRelHumidity = dht.readHumidity();

  // Show us what the sensor did read
  print2digits(rtc.getYear()+2000);
  Serial.print("-");
  print2digits(rtc.getMonth());
  Serial.print("-");
  print2digits(rtc.getDay());
  Serial.print(" ");
  print2digits(rtc.getHours());
  Serial.print(":");
  print2digits(rtc.getMinutes());
  Serial.print(":");
  print2digits(rtc.getSeconds());
  Serial.print(" - ");
  Serial.print("Ta: ");
  Serial.print(rTemperature);
  Serial.print(" °C    RH: ");
  Serial.print(rRelHumidity);
  Serial.println(" %");

  // Write to SD
  write2digits(rtc.getYear()+2000);
  dataFile.print("-");
  write2digits(rtc.getMonth());
  dataFile.print("-");
  write2digits(rtc.getDay());
  dataFile.print(" ");
  write2digits(rtc.getHours());
  dataFile.print(":");
  write2digits(rtc.getMinutes());
  dataFile.print(":");
  write2digits(rtc.getSeconds());
  dataFile.print(",");
  dataFile.print(rTemperature);
  dataFile.print(",");
  dataFile.println(rRelHumidity);
  dataFile.flush();

  // Successful loop instance completion: inform users with a single blink
  blink(1);

}
```