WeatherShield 1

Introduction

The WeatherShield1 is a shield for Arduino or Netduino boards. It lets possible to easily implement a simple weather station able to read temperature, ambient pressure and relative humidity. The shield is equipped with an on board controller responsible for measuring, averaging and conditioning sampled data and to present them to the user in the internationally known units (Celsius, hPa and relative %).

The connection between Arduino or Netduino and the WeatherShield1 is made by a two directional synchronous serial bus using only two digital pins (the line D2, bidirectional, for the data and the line D7, unidirectional from Arduino to the WeatherShield1, for the clock). Is possible to modify the pins used by the libraries in order to adapt the code for different needs.

Each WeatherShield1 is provided with an internal, software programmable, address. This is useful when more than one WeatherShield is connected to the same Arduino or Netduino.

How to use it

Using the WeatherShield1 is easy and is much more simple thanks to the freely downloadable libraries present on the EtherMania website.

When decompressed the .zip file and placed the libraries in the correct place (see the Arduino IDE FAQ for the how to install the libraries) is possible to start to connect to the WeatherShield simply calling the provided API. Below there is an example on how to use these APIs to properly connect to the WeatherShield1.

The first step to be performed is to include the libraries header at the very beginning of your sketches:

```
#include <WeatherShield1.h>
```

is now possible to instantiate a global WeatherShield type object:

```
WeatherShield1 weatherShield;
```

Optionally, is possible to pass some information to the WeatherShield constructor. This is useful if you need to specify a different I/O setup or a different WeatherShield1 address and could be done replacing the previous line with the next code snipset:

```
#define IODATA_PIN    2

#define CLOCK_PIN     7

#define MY_WEATHERSHIELD_ADDRESS   10

WeatherShield1 weatherShield(CLOCK_PIN, IODATA_PIN, MY_WEATHERSHIELD_ADDRESS);
```

and modifying the values associated to IODATA_PIN, CLOCK_PIN o MY_WEATHERSHIELD_ADDRESS in order to match your needs.

This is all is required to initialize the communications between Arduino and the WeatherShield1. Is then possible to start to read temperature, pressure and humidity values.

As already described, all communications between Arduino and the WeatherShield1 are performed through a bidirectional synchronous serial communication protocol. The Arduino side is implemented by the provided libraries. All values to and from the WeatherShield should be temporarily stored in a four bytes size buffer that should be defined in the loop function and passed to each library calls.

```
void loop() {
   unsigned char ucBuffer[4];
...
```

Communicating with the WeatherShield is generally made in two steps:

1. the user code asks for a command execution. The WeatherShield answer is stored in the temporary buffer
2. the temporary buffer is then decoded by a particular API call that returns the read value in the required format (float, integer etc.).

The above sequence should be replicated for each command. Some exception applies for particular commands that don't return any value.

Sending a command

The SendCommand function triggers a particular command to the WeatherShield1. This function has to be called specifying three parameters:

1. The command identifier (see below)
2. A parameter. The semantical value of this parameter is dependent by the command type
3. The buffer (pointer) where the result should be stored.

Here is a list of available commands:

- CMD_ECHO_PAR: The WeatherShield1 answers replicating the same character specified as parameter when calling this function. It could be used for a communication check and for a WeatherShield1 presence check.
- CMD_SET_SAMPLETIME: set the sample time used by the WeatherShield1 to collect weather information. The parameter specifies the sampling time (in seconds). Values from 0 (1 second period) to 255 (256 seconds period) are allowed. This value is stored in a EEPROM area in the WeatherShield1 and used for all subsequent runs until a new value will be specified. Note: The factory sample time value is 30 seconds.
- CMD_GETTEMP_C_AVG: read the temperature value calculated as moving average for last 8 samples and corrected based on sensor datasheet suggestions. The returned value is measured in Celsius grades with two decimals precision. The specified parameter will be ignored by the WeatherShield1.
- CMD_GETTEMP_C_RAW: read a 10 bit value coming from the temperature sensor. The parameter specifies which value should be read. Setting the parameter to PAR_GET_LAST_SAMPLE the returned valued will be the last sample; PAR_GET_AVG_SAMPLE forces the WeatherShield1 to return the 10 bit value calculated by the last 8 samples moving average (without any correction); setting the parameter to a value between 0 and 7 forces the WeatherShield1 to return one of the eight stored samples.
- CMD_GETPRESS_AVG: read the pressure value calculated as moving average for last 8 samples and corrected based on sensor datasheet suggestions. The returned value is measured in hPa with two decimals precision. The specified parameter will be ignored by the WeatherShield1.
- CMD_GETPRESS_RAW: read a 10 bit value coming from the pressure sensor. The parameter specifies which value should be read. Setting the parameter to PAR_GET_LAST_SAMPLE the returned valued will be the last sample; PAR_GET_AVG_SAMPLE forces the WeatherShield1 to return the 10 bit value calculated by the last 8 samples moving average (without any correction); setting the parameter to a value between 0 and 7 forces the WeatherShield1 to return one of the eight stored samples.
- CMD_GETHUM_AVG: read the humidity value calculated as moving average for last 8 samples and corrected based on sensor datasheet suggestions. The returned value is measured in hPa with two decimals precision. The specified parameter will be ignored by the WeatherShield1.
- CMD_GETHUM_RAW: read a 10 bit value coming from the humidity sensor. The parameter specifies which value should be read. Setting the parameter to the value PAR_GET_LAST_SAMPLE the returned valued will be the last sample; the value PAR_GET_AVG_SAMPLE forces the WeatherShield1 to return the 10 bit value calculated by the last 8 samples moving average (without any correction); setting the parameter to a value between 0 and 7 forces the WeatherShield1 to return one of the eight stored samples.
- CMD_SETADDRESS: changes the address associated to the WeatherShield1. The new address will be equal to the specified parameter and will be stored in a EEPROM area present on the WeatherShield1 that will be retained after a power loss. This function is useful when more than one WeatherShield1 share the same I/O communication lines. N.B. The factory address is equal to 1.

The function returns a boolean “true” value if the communication with the WeatherShield1 was correctly performed; false otherwise.

Decoding results

The results coming from the WeatherShield1 after a command execution could be read through a set of decoding functions. The APIs provide functions able to convert the results in three different formats:

- `float decodeFloatValue(unsigned char *pucBuffer)`: it should be used mainly when reading averaged values (CMD_GETTEMP_C_AVG, CMD_GETPRESS_AVG, CMD_GETHUM_AVG). It returns the read value in a float format. It's mandatory to pass the buffer used to store the command result (a pointer of the temporary buffer).
- `void decodeFloatAsString(unsigned char *pucBuffer, char *chString)`: it should be used mainly

when reading averaged values (CMD_GETTEMP_C_AVG, CMD_GETPRESS_AVG, CMD_GETHUM_AVG). It's mandatory to pass the buffer used to store the command result (a pointer of the temporary buffer) and a second buffer where the result will be stored in ASCII format.

- `unsigned short decodeShortValue(unsigned char *pucBuffer)`: it should be used mainly when reading RAW values (CMD_GETTEMP_C_RAW, CMD_GETPRESS_RAW, CMD_GETHUM_RAW). It converts the result in a 16 bit unsigned short (integer). It's mandatory to pass the buffer used to store the command result (a pointer to the temporary buffer)

Other functions

Is possible to force a serial communication reset through the resetConnection() function. This is useful when, for some reason, the WeatherShield1 is not able to answer to Arduino requests and the SendCommand function starts returning “false”.

The resetConnection() function resets the serial link without affecting the already sampled values and calculated moving average values.

A simple Sketch

As example is here reported a simple sketch required to read the averaged temperature and the last RAW sample, and to send them through the Serial Monitor.

```
#include <WeatherShield1.h>
#define RXBUFFERLENGTH          4
WeatherShield1 weatherShield;

/* This is the main sketch setup handler */
void setup(){
  /* Initialize the serial port */
  Serial.begin(9600);
}

/* This is the main sketch loop handler */
void loop() {
    /* This is the buffer for the answers */
  unsigned char ucBuffer[RXBUFFERLENGTH];

  /* Check for the weather shield connection */
  if (weatherShield.sendCommand(CMD_ECHO_PAR, 100, ucBuffer)) {
    Serial.println("Connection With Shield Performed OK");
  }

  /* Start reading temperature */
  float fTemperature = 0.0f;
  unsigned short shTemperature = 0;
  if (weatherShield.sendCommand(CMD_GETTEMP_C_AVG, 0, ucBuffer))
    fTemperature = weatherShield.decodeFloatValue(ucBuffer);
  if (weatherShield.sendCommand(CMD_GETTEMP_C_RAW, PAR_GET_LAST_SAMPLE, ucBuffer))
    shTemperature = weatherShield.decodeShortValue(ucBuffer);

  /* Send all data through the serial line */
  Serial.print("Temperature ");
  Serial.print(fTemperature);
  Serial.print(" Celsius (");
  Serial.print(shTemperature);
  Serial.println(")");

  /* Wait some time before running again */
  delay(1000);
}
```