```
1  //
2  //    FILE: dht.cpp
3  //  AUTHOR: Rob Tillaart
4  // VERSION: 0.1.14
5  // PURPOSE: DHT Temperature & Humidity Sensor library for Arduino
6  //     URL: http://arduino.cc/playground/Main/DHTLib
7  //
8  // HISTORY:
9  // 0.1.14 replace digital read with faster (~3x) code => more robust low MHz
   machines.
10 // 0.1.13 fix negative temperature
11 // 0.1.12 support DHT33 and DHT44 initial version
12 // 0.1.11 renamed DHTLIB_TIMEOUT
13 // 0.1.10 optimized faster WAKEUP + TIMEOUT
14 // 0.1.09 optimize size: timeout check + use of mask
15 // 0.1.08 added formula for timeout based upon clockspeed
16 // 0.1.07 added support for DHT21
17 // 0.1.06 minimize footprint (2012-12-27)
18 // 0.1.05 fixed negative temperature bug (thanks to Roseman)
19 // 0.1.04 improved readability of code using DHTLIB_OK in code
20 // 0.1.03 added error values for temp and humidity when read failed
21 // 0.1.02 added error codes
22 // 0.1.01 added support for Arduino 1.0, fixed typos (31/12/2011)
23 // 0.1.00 by Rob Tillaart (01/04/2011)
24 //
25 // inspired by DHT11 library
26 //
27 // Released to the public domain
28 //
29
30 #include "dhtlib_gpa788.h"
31
32 /////////////////////////////////////////////////
33 //
34 // PUBLIC
35 //
36
37 // return values:
38 // DHTLIB_OK
39 // DHTLIB_ERROR_CHECKSUM
40 // DHTLIB_ERROR_TIMEOUT
41
42
43
44 DHTLIB_ErrorCode dhtlib_gpa788::read11(uint8_t pin)
45 {
46     // READ VALUES
47     DHTLIB_ErrorCode rv = _readSensor(pin, DHTLIB_DHT11_WAKEUP);
48     if (rv != DHTLIB_ErrorCode::DHTLIB_OK)
49     {
50         humidity    = double(DHTLIB_ErrorCode::DHTLIB_INVALID_VALUE); // invalid
   value, or is NaN prefered?
51         temperature = double(DHTLIB_ErrorCode::DHTLIB_INVALID_VALUE); // invalid
   value
52         return rv;
53     }
54
55     // CONVERT AND STORE
56     humidity    = bits[0];  // bits[1] == 0;
```

```
57      temperature = bits[2];  // bits[3] == 0;
58
59      // TEST CHECKSUM
60      // bits[1] && bits[3] both 0
61      // Certains capteurs DHT11 transmettent le checksum dont la valeur inclut
62      // la partie fractionnaire de la température et l'humidité relative
63      // d'où la modification apportée ici. (TW, 2019)
64      uint8_t sum = bits[0] + bits[2] + bits[1] + bits[3];
65      if (bits[4] != sum) return DHTLIB_ErrorCode::DHTLIB_ERROR_CHECKSUM;
66
67      return DHTLIB_ErrorCode::DHTLIB_OK;
68  }
69
70
71  // return values:
72  // DHTLIB_OK
73  // DHTLIB_ERROR_CHECKSUM
74  // DHTLIB_ERROR_TIMEOUT
75  DHTLIB_ErrorCode dhtlib_gpa788::read(uint8_t pin)
76  {
77      // READ VALUES
78      DHTLIB_ErrorCode rv = _readSensor(pin, DHTLIB_DHT_WAKEUP);
79      if (rv != DHTLIB_ErrorCode::DHTLIB_OK)
80      {
81          humidity    = double(DHTLIB_ErrorCode::DHTLIB_INVALID_VALUE);  // invalid
    value, or is NaN prefered?
82          temperature = double(DHTLIB_ErrorCode::DHTLIB_INVALID_VALUE);  // invalid
    value
83          return rv; // propagate error value
84      }
85
86      // CONVERT AND STORE
87      humidity = word(bits[0], bits[1]) * 0.1;
88      temperature = word(bits[2] & 0x7F, bits[3]) * 0.1;
89      if (bits[2] & 0x80)  // negative temperature
90      {
91          temperature = -temperature;
92      }
93
94      // TEST CHECKSUM
95      uint8_t sum = bits[0] + bits[1] + bits[2] + bits[3];
96      if (bits[4] != sum)
97      {
98          return DHTLIB_ErrorCode::DHTLIB_ERROR_CHECKSUM;
99      }
100     return DHTLIB_ErrorCode::DHTLIB_OK;
101 }
102
103 /////////////////////////////////////////////////
104 //
105 // PRIVATE
106 //
107
108 // return values:
109 // DHTLIB_OK
110 // DHTLIB_ERROR_TIMEOUT
111 DHTLIB_ErrorCode dhtlib_gpa788::_readSensor(uint8_t pin, uint8_t wakeupDelay)
112 {
113     // INIT BUFFERVAR TO RECEIVE DATA
114     uint8_t mask = 128;
```

```
115     uint8_t idx = 0;
116
117     // replace digitalRead() with Direct Port Reads.
118     // reduces footprint ~100 bytes => portability issue?
119     // direct port read is about 3x faster
120   uint8_t bit = digitalPinToBitMask(pin);
121   uint8_t port = digitalPinToPort(pin);
122     volatile uint8_t *PIR = portInputRegister(port);
123
124     // EMPTY BUFFER
125     for (uint8_t i = 0; i < 5; i++) bits[i] = 0;
126
127     // REQUEST SAMPLE
128     pinMode(pin, OUTPUT);
129     digitalWrite(pin, LOW); // T-be
130     delay(wakeupDelay);
131     digitalWrite(pin, HIGH);   // T-go
132     delayMicroseconds(40);
133     pinMode(pin, INPUT);
134
135     // GET ACKNOWLEDGE or TIMEOUT
136     uint16_t loopCntLOW = DHTLIB_TIMEOUT;
137     while ((*PIR & bit) == LOW )  // T-rel
138     {
139         if (--loopCntLOW == 0) return DHTLIB_ErrorCode::DHTLIB_ERROR_TIMEOUT;
140     }
141
142     uint16_t loopCntHIGH = DHTLIB_TIMEOUT;
143     while ((*PIR & bit) != LOW )  // T-reh
144     {
145         if (--loopCntHIGH == 0) return DHTLIB_ErrorCode::DHTLIB_ERROR_TIMEOUT;
146     }
147
148     // READ THE OUTPUT - 40 BITS => 5 BYTES
149     for (uint8_t i = 40; i != 0; i--)
150     {
151         loopCntLOW = DHTLIB_TIMEOUT;
152         while ((*PIR & bit) == LOW )
153         {
154             if (--loopCntLOW == 0) return DHTLIB_ErrorCode::DHTLIB_ERROR_TIMEOUT;
155         }
156
157         uint32_t t = micros();
158
159         loopCntHIGH = DHTLIB_TIMEOUT;
160         while ((*PIR & bit) != LOW )
161         {
162             if (--loopCntHIGH == 0) return DHTLIB_ErrorCode::DHTLIB_ERROR_TIMEOUT;
163         }
164
165         if ((micros() - t) > 40)
166         {
167             bits[idx] |= mask;
168         }
169         mask >>= 1;
170         if (mask == 0)   // next byte?
171         {
172             mask = 128;
173             idx++;
174         }
```

```
175     }
176     pinMode(pin, OUTPUT);
177     digitalWrite(pin, HIGH);
178
179     return DHTLIB_ErrorCode::DHTLIB_OK;
180 }
181 //
182 // END OF FILE
183 //
184
```