```cpp
/*
ThermistorNTC.cpp - Library to used to derive a precise temperature of a thermistor,
fastest Calc (26~18% faster)
v0.2

Copyright © 2021 Francisco Rafael Reyes Carmona.
All rights reserved.

rafael.reyes.carmona@gmail.com


  This file is part of ThermistorNTC.

  ThermistorNTC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  ThermistorNTC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with ThermistorNTC.  If not, see <https://www.gnu.org/licenses/>.

*/

#include "ThermistorNTC.h"
#include <math.h>

// Constructor para 4 parametros (A,B,C,D).
Thermistor::Thermistor(int PIN,
                       long RESISTOR,
                       long NTC_25C,
                       double A,
                       double B,
                       double C,
                       double D,
                       float VREF){
  _PIN = PIN;
  _RESISTOR = RESISTOR;
  _NTC_25C = NTC_25C;
  _A = A;
  _B = B;
  _C = C;
  _D = D;
  _VREF = VREF;

  pinMode( PIN, INPUT);
```

```
  pinMode(_PIN, INPUT);
}

// Constructor para 3 parametros (A,B,D.. C = 0).
Thermistor::Thermistor(int PIN,
                       long RESISTOR,
                       long NTC_25C,
                       double A,
                       double B,
                       double D,
                       float VREF){
  _PIN = PIN;
  _RESISTOR = RESISTOR;
  _NTC_25C = NTC_25C;
  _A = A;
  _B = B;
  _D = D;
  _VREF = VREF;

  pinMode(_PIN, INPUT);
}

// Constructor para parametro BETA del termistor.
Thermistor::Thermistor(int PIN,
                       long RESISTOR,
                       long NTC_25C,
                       float BETA,
                       float VREF){
  _PIN = PIN;
  _RESISTOR = RESISTOR;
  _NTC_25C = NTC_25C;
  _BETA = BETA;
  _VREF = VREF;

  pinMode(_PIN, INPUT);
}

// Constructor cuando se desconoce los parámetros del termistor. 3
Coeficientes.
Thermistor::Thermistor(int PIN,
                       long RESISTOR,
                       long NTC_1,
                       float TEMP_1,
                       long NTC_2,
                       float TEMP_2,
                       long NTC_3,
                       float TEMP_3,
                       float VREF){
  _PIN = PIN;
  _RESISTOR = RESISTOR;
  _VREF = VREF;
```

```
100
101     calcCoefficients3(TEMP_1, NTC_1, TEMP_2, NTC_2, TEMP_3, NTC_3);
102
103     pinMode(_PIN, INPUT);
104   }
105
106   // Constructor cuando se desconoce los parámetros del termistor. 4
      Coeficientes.
107   Thermistor::Thermistor(int PIN,
108                          long RESISTOR,
109                          long NTC_1,
110                          float TEMP_1,
111                          long NTC_2,
112                          float TEMP_2,
113                          long NTC_3,
114                          float TEMP_3,
115                          long NTC_4,
116                          float TEMP_4,
117                          float VREF){
118     _PIN = PIN;
119     _RESISTOR = RESISTOR;
120     _VREF = VREF;
121
122     calcCoefficients4(TEMP_1, NTC_1, TEMP_2, NTC_2, TEMP_3, NTC_3, TEMP_4,
      NTC_4);
123
124     pinMode(_PIN, INPUT);
125   }
126
127
128   void Thermistor::SteinhartHart(Thermistor_connection ConType){
129     float E = log(calcNTC(ConType));
130     _temp_k = _A + (_B*E) + (_C*(E*E)) + (_D*(E*E*E));
131     _temp_k = 1.0 / _temp_k;
132     _temp_c = _temp_k - 273.15;
133   }
134
135
136   void Thermistor::SteinhartHart_beta(Thermistor_connection ConType){
137     _temp_k = log(calcNTC(ConType)/(float)_NTC_25C);
138     _temp_k /= _BETA;
139     _temp_k += 1.0 / 298.15;
140     _temp_k = 1.0 / _temp_k;
141     _temp_c = _temp_k - 273.15;
142   }
143
144
145   void Thermistor::SteinhartHart_fast(Thermistor_connection ConType){
146     _temp_k = log(calcNTC(ConType)/(float)_NTC_25C);
147     _temp_k *= 298.15;
148     _temp_k += _BETA;
```

```
149     _temp_k = (_BETA * 298.15) / _temp_k;
150     _temp_c = _temp_k - 273.15;
151   }
152
153
154   double Thermistor::getTempKelvin_SteinHart(Thermistor_connection ConType) {
155     SteinhartHart(ConType);
156     return _temp_k;
157   }
158
159
160   double Thermistor::getTempCelsius_SteinHart(Thermistor_connection ConType)
      {
161     SteinhartHart(ConType);
162     return _temp_c;
163   }
164
165
166   double Thermistor::getTempFahrenheit_SteinHart(Thermistor_connection
      ConType){
167     return getTempCelsius_SteinHart(ConType) * 9/5 + 32;
168   }
169
170
171   double Thermistor::getTempKelvin(Thermistor_connection ConType) {
172     _BETA > 0.0 ? SteinhartHart_beta(ConType) : SteinhartHart(ConType);
173     return _temp_k;
174   }
175
176
177   double Thermistor::getTempCelsius(Thermistor_connection ConType) {
178     _BETA > 0.0 ? SteinhartHart_beta(ConType) : SteinhartHart(ConType);
179     return _temp_c;
180   }
181
182
183   double Thermistor::getTempFahrenheit(Thermistor_connection ConType){
184     return getTempCelsius(ConType) * 9/5 + 32;
185   }
186
187
188   double Thermistor::fastTempKelvin(Thermistor_connection ConType) {
189     _BETA > 0.0 ? SteinhartHart_fast(ConType) : SteinhartHart(ConType);
190     return _temp_k;
191   }
192
193
194   double Thermistor::fastTempCelsius(Thermistor_connection ConType) {
195     _BETA > 0.0 ? SteinhartHart_fast(ConType) : SteinhartHart(ConType);
196     return _temp_c;
```

```
}


double Thermistor::fastTempFahrenheit(Thermistor_connection ConType){
  return fastTempCelsius(ConType) * 9/5 + 32;
}


float Thermistor::getADC(int numsamples){
  float EMA_LOW;
  int microdelay;

  microdelay = (1 <<((1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0)));
  microdelay = microdelay * 2000000 / F_CPU;

  EMA_LOW = analogRead(_PIN);

  for (byte i = numsamples; i--; ){
    delayMicroseconds(microdelay);
    EMA_LOW = (_alphaEMA_LOW * (float)analogRead(_PIN)) + ((1.0 -
    _alphaEMA_LOW) * EMA_LOW);
  }

  return EMA_LOW;
}


double Thermistor::calcNTC(Thermistor_connection ConType){
  double NTC;
  float ADC_VALUE = getADC();
  if (ConType == VCC){
    NTC = (float)_ADC_MAX * (float)_RESISTOR;
    NTC -= ADC_VALUE * (float)_RESISTOR;
    NTC /= ADC_VALUE;
    return NTC;
  }
    NTC = ADC_VALUE * _VREF / (float)_ADC_MAX;
    NTC /= (_VREF - NTC);
    NTC *= (float)_RESISTOR;
    return NTC;
}


void Thermistor::calcCoefficients3(float T1, long RT1, float T2, long RT2,
float T3, long RT3){
  float _T1 = T1 + 273.15f;
  float _T2 = T2 + 273.15f;
  float _T3 = T3 + 273.15f;
  float L1 = log((float)RT1);
  float L2 = log((float)RT2);
  float L3 = log((float)RT3);
```

```
246
247     float BETA1 = _T1 * _T2 * (L1 - L2) / (_T2-_T1);
248     float BETA2 = _T2 * _T3 * (L2 - L3) / (_T3-_T2);
249     _BETA = (BETA1 + BETA2) / 2.0f;
250
251     long NTC_25C1 = RT1 / exp(- _BETA * (_T1 - 298.15f)/ _T1 / 298.15f);
252     long NTC_25C2 = RT2 / exp(- _BETA * (_T2 - 298.15f)/ _T2 / 298.15f);
253     long NTC_25C3 = RT3 / exp(- _BETA * (_T3 - 298.15f)/ _T3 / 298.15f);
254     _NTC_25C = (NTC_25C1 + NTC_25C2 + NTC_25C3) / 3L ;
255
256
257     float Y1 = 1.0f / (_T1);
258     float Y2 = 1.0f / (_T2);
259     float Y3 = 1.0f / (_T3);
260     double yY2 = (Y2 - Y1)/(L2 - L1);
261     double yY3 = (Y3 - Y1)/(L3 - L1);
262
263     _D = (yY3 - yY2);
264     _D /= ((L3 -L2) * (L1 + L2 + L3));
265     _B = (L1 * L1) + (L1 * L2) + (L2 *L2);
266     _B *= _D;
267     _B = yY2 - _B;
268     _A = _D * L1 * L1;
269     _A += _B;
270     _A *= L1;
271     _A = Y1 - _A;
272   }
273
274
275   void Thermistor::calcCoefficients4(float T1, long RT1, float T2, long RT2,
  •   float T3, long RT3, float T4, long RT4){
276     float _T1 = T1 + 273.15f;
277     float _T2 = T2 + 273.15f;
278     float _T3 = T3 + 273.15f;
279     float _T4 = T4 + 273.15f;
280     float L1 = log((float)RT1);
281     float L2 = log((float)RT2);
282     float L3 = log((float)RT3);
283     float L4 = log((float)RT4);
284
285     float BETA1 = _T1 * _T2 * (L1 - L2) / (_T2-_T1);
286     float BETA2 = _T2 * _T3 * (L2 - L3) / (_T3-_T2);
287     float BETA3 = _T3 * _T4 * (L3 - L4) / (_T4-_T3);
288     _BETA = (BETA1 + BETA2 + BETA3) / 3.0f;
289
290     long NTC_25C1 = RT1 / exp(- _BETA * (_T1 - 298.15f)/ _T1 / 298.15f);
291     long NTC_25C2 = RT2 / exp(- _BETA * (_T2 - 298.15f)/ _T2 / 298.15f);
292     long NTC_25C3 = RT3 / exp(- _BETA * (_T3 - 298.15f)/ _T3 / 298.15f);
293     long NTC_25C4 = RT4 / exp(- _BETA * (_T4 - 298.15f)/ _T4 / 298.15f);
294     _NTC_25C = (NTC_25C1 + NTC_25C2 + NTC_25C3 + NTC_25C4) / 4L ;
295
```

```
296     float L1_2 = L1*L1;
297     float L2_2 = L2*L2;
298     float L3_2 = L3*L3;
299     float L4_2 = L4*L4;
300     float L1_3 = L1*L1_2;
301     float L2_3 = L2*L2_2;
302     float L3_3 = L3*L3_2;
303     float L4_3 = L4*L4_2;
304     float Y1 = 1.0f / (_T1);
305     float Y2 = 1.0f / (_T2);
306     float Y3 = 1.0f / (_T3);
307     float Y4 = 1.0f / (_T4);
308
309    /*
310     double L2_L1 = L2 - L1;
311     double yy2 = (L2_2 - L1_2) / L2_L1;
312     double yY3 = ((L3_2 - L1_2) / (L3 - L1)) - yy2;
313     double yY4 = ((L4_2 - L1_2) / (L4 - L1)) - yy2;
314
315     double Dd3 = (((L3_3 - L1_3) / (L3 - L1)) - ((L2_3 - L1_3) / L2_L1)) /
    •    yY3;
316     double Dd4 = (((L4_3 - L1_3) / (L4 - L1)) - ((L2_3 - L1_3) / L2_L1)) /
    •    yY4;
317     double Dy3 = (((Y3 - Y1) / (L3 - L1) - yy2) / yY3);
318     double Dy4 = (((Y4 - Y1) / (L4 - L1) - yy2) / yY4);
319
320     _D = (Dy4 - Dy3) / (Dd4 - Dd3);
321     _C = Dy3 - (Dd3 * _D);
322
323     double Z1 = Y1 - (_C * L1_2) - (_D * L1_3);
324     double Z2 = Y2 - (_C * L2_2) - (_D * L2_3);
325
326     _A = (Z1 * L2) - (Z2 * L1);
327     _A /= L2_L1;
328     _B = Z2 - Z1;
329     _B /= L2_L1;
330     */
331   //  /*
332     float L2_L1 = L2 - L1;
333     float L3_L1 = L3 - L1;
334     float L4_L1 = L4 - L1;
335
336     double DS1_1 = (L3_2-L1_2) * L2_L1 - (L2_2-L1_2) * L3_L1;
337     double DS1_2 = (L4_3-L1_3) * L2_L1 - (L2_3-L1_3) * L4_L1;
338     double DS2_1 = (L4_2-L1_2) * L2_L1 - (L2_2-L1_2) * L4_L1;
339     double DS2_2 = (L3_3-L1_3) * L2_L1 - (L2_3-L1_3) * L3_L1;
340     double DY1 = (Y3-Y1) * L2_L1 - (Y2-Y1) * L3_L1;
341     double DY2 = (Y4-Y1) * L2_L1 - (Y2-Y1) * L4_L1;
342
343     double DS = (DS1_1 * DS1_2) - (DS2_1 * DS2_2);
```

```
344     double DC = (DY1 * DS1_2) - (DY2 * DS2_2);
345     double DD = (DY2 * DS1_1) - (DY1 * DS2_1);
346
347     _D = DD / DS;
348     _C = DC / DS;
349
350     double Z1 = Y1 - (_C * L1_2) - (_D * L1_3);
351     double Z2 = Y2 - (_C * L2_2) - (_D * L2_3);
352
353     _A = (Z1 * L2) - (Z2 * L1);
354     _A /= L2_L1;
355     _B = Z2 - Z1;
356     _B /= L2_L1;
357   //  */
358   }
359
360
361   void Thermistor::setADC(int ADC_MAX){
362     _ADC_MAX = ADC_MAX;
363   }
364
365
366   void Thermistor::setEMA(float EMA){
367     _alphaEMA_LOW = EMA;
368   }
369
```