/\*!

\* @file DHT.cpp

\*

\* @mainpage DHT series of low cost temperature/humidity sensors.

\*

\* @section intro\_sec Introduction

\*

\* This is a library for DHT series of low cost temperature/humidity sensors.

\*

\* You must have Adafruit Unified Sensor Library library installed to use this

\* class.

\*

\* Adafruit invests time and resources providing this open source code,

\* please support Adafruit andopen-source hardware by purchasing products

\* from Adafruit!

\*

\* @section author Author

\*

\* Written by Adafruit Industries.

\*

\* @section license License

\*

\* MIT license, all text above must be included in any redistribution

\*/

#include "DHT.h" //引入DHT

#define MIN\_INTERVAL 2000 /\*\*< min interval value \*/ //設定最小間隔時間

#define TIMEOUT -1 /\*\*< timeout on \*/ //設定timeout

/\*!

\* @brief Instantiates a new DHT class

\* @param pin

\* pin number that sensor is connected

\* @param type

\* type of sensor

\* @param count

\* number of sensors

\*/

DHT::DHT(uint8\_t pin, uint8\_t type, uint8\_t count) {

\_pin = pin;

\_type = type;

#ifdef \_\_AVR

\_bit = digitalPinToBitMask(pin); //把Pin轉轉成bitmask給\_bit

\_port = digitalPinToPort(pin);//把pin轉成Port給\_port

#endif

\_maxcycles =

microsecondsToClockCycles(1000); // 1 millisecond timeout for

// reading pulses from DHT sensor.

// Note that count is now ignored as the DHT reading algorithm adjusts itself

// based on the speed of the processor.

//間隔時間超過1毫秒時，從DHT傳感器讀取脈衝

}

/\*!

\* @brief Setup sensor pins and set pull timings

\* @param usec

\* Optionally pass pull-up time (in microseconds) before DHT reading

\*starts. Default is 55 (see function declaration in DHT.h).

\*/

void DHT::begin(uint8\_t usec) {

// set up the pins!

//設定pin

pinMode(\_pin, INPUT\_PULLUP);

// Using this value makes sure that millis() - lastreadtime will be

// >= MIN\_INTERVAL right away. Note that this assignment wraps around,

// but so will the subtraction.

\_lastreadtime = millis() - MIN\_INTERVAL;

DEBUG\_PRINT("DHT max clock cycles: ");

DEBUG\_PRINTLN(\_maxcycles, DEC);

pullTime = usec;

//利用這確保 millis()- \_lastreadtime >= 最小間隔

}

/\*!

\* @brief Read temperature

\* @param S

\* Scale. Boolean value:

\* - true = Fahrenheit

\* - false = Celcius

\* @param force

\* true if in force mode

\* @return Temperature value in selected scale

\*/

float DHT::readTemperature(bool S, bool force) {

//讀取溫度

float f = NAN;

//把f設定成NAN(空值)

if (read(force)) {

//如果有讀到數值

switch (\_type) {//切換DHT11 DHT12 DHT21 DHT22

case DHT11:

f = data[2];

if (data[3] & 0x80) {

f = -1 - f;

}

f += (data[3] & 0x0f) \* 0.1;

if (S) {//轉換成F

f = convertCtoF(f);

}

break;

case DHT12:

f = data[2];

f += (data[3] & 0x0f) \* 0.1;

if (data[2] & 0x80) {

f \*= -1;

}

if (S) {//轉換成F

f = convertCtoF(f);

}

break;

case DHT22:

case DHT21:

f = ((word)(data[2] & 0x7F)) << 8 | data[3];

f \*= 0.1;

if (data[2] & 0x80) {

f \*= -1;

}

if (S) {//轉換成F

f = convertCtoF(f);

}

break;

}

}

return f; //回傳f

}

/\*!

\* @brief Converts Celcius to Fahrenheit

\* @param c

\* value in Celcius

\* @return float value in Fahrenheit

\*/

float DHT::convertCtoF(float c) { return c \* 1.8 + 32; }

//把攝氏溫度換成華氏溫度

/\*!

\* @brief Converts Fahrenheit to Celcius

\* @param f

\* value in Fahrenheit

\* @return float value in Celcius

\*/

float DHT::convertFtoC(float f) { return (f - 32) \* 0.55555; }

//把華氏溫度改成攝氏溫度

/\*!

\* @brief Read Humidity

\* @param force

\* force read mode

\* @return float value - humidity in percent

\*/

float DHT::readHumidity(bool force) {

float f = NAN;//設定f=NAN

if (read(force)) {//如果有讀到數值

switch (\_type) {//切換DHT版本

case DHT11:

case DHT12:

f = data[0] + data[1] \* 0.1;

break;

case DHT22:

case DHT21:

f = ((word)data[0]) << 8 | data[1];

f \*= 0.1;

break;

}

}

return f;//回傳f

}

/\*!

\* @brief Compute Heat Index

\* Simplified version that reads temp and humidity from sensor

\* @param isFahrenheit

\* true if fahrenheit, false if celcius (default

\*true)

\* @return float heat index

\*/

float DHT::computeHeatIndex(bool isFahrenheit) {//計算熱源

float hi = computeHeatIndex(readTemperature(isFahrenheit), readHumidity(),

isFahrenheit);//讀取溫度與濕度計算溫度

return hi;//回傳hi

}

/\*!

\* @brief Compute Heat Index

\* Using both Rothfusz and Steadman's equations

\* (http://www.wpc.ncep.noaa.gov/html/heatindex\_equation.shtml)

\* @param temperature

\* temperature in selected scale

\* @param percentHumidity

\* humidity in percent

\* @param isFahrenheit

\* true if fahrenheit, false if celcius

\* @return float heat index

\*/

float DHT::computeHeatIndex(float temperature, float percentHumidity,

bool isFahrenheit) {//輸入溫度,濕度 確認是否為華氏溫度

float hi;

//確認是否為華氏溫度

if (!isFahrenheit)

temperature = convertCtoF(temperature);//轉換華氏溫度

hi = 0.5 \* (temperature + 61.0 + ((temperature - 68.0) \* 1.2) +

(percentHumidity \* 0.094));//計算熱指數

if (hi > 79) { //如果高於79

hi = -42.379 + 2.04901523 \* temperature + 10.14333127 \* percentHumidity +

-0.22475541 \* temperature \* percentHumidity +

-0.00683783 \* pow(temperature, 2) +

-0.05481717 \* pow(percentHumidity, 2) +

0.00122874 \* pow(temperature, 2) \* percentHumidity +

0.00085282 \* temperature \* pow(percentHumidity, 2) +

-0.00000199 \* pow(temperature, 2) \* pow(percentHumidity, 2);

if ((percentHumidity < 13) && (temperature >= 80.0) &&

(temperature <= 112.0)) //溫度及濕度對於熱指數的調整

hi -= ((13.0 - percentHumidity) \* 0.25) \*

sqrt((17.0 - abs(temperature - 95.0)) \* 0.05882);

else if ((percentHumidity > 85.0) && (temperature >= 80.0) &&

(temperature <= 87.0))//溫度及濕度對於熱指數的調整

hi += ((percentHumidity - 85.0) \* 0.1) \* ((87.0 - temperature) \* 0.2);

}

return isFahrenheit ? hi : convertFtoC(hi); //如果是華氏溫度回傳fi如果是攝氏溫度->進行更改

}

/\*!

\* @brief Read value from sensor or return last one from less than two

\*seconds.

\* @param force

\* true if using force mode

\* @return float value

\*/

bool DHT::read(bool force) {//讀取值

// Check if sensor was read less than two seconds ago and return early

// to use last reading.

uint32\_t currenttime = millis();

if (!force && ((currenttime - \_lastreadtime) < MIN\_INTERVAL)) {

return \_lastresult; // return last correct measurement

}//如果沒有讀到值或是超時 回傳上一個結果

\_lastreadtime = currenttime;

// Reset 40 bits of received data to zero.

data[0] = data[1] = data[2] = data[3] = data[4] = 0;

//把data重製為0

#if defined(ESP8266)

yield(); // Handle WiFi / reset software watchdog 處理wifi/重製軟體watchdog

#endif

// Send start signal. See DHT datasheet for full signal diagram:

// http://www.adafruit.com/datasheets/Digital%20humidity%20and%20temperature%20sensor%20AM2302.pdf

// Go into high impedence state to let pull-up raise data line level and

// start the reading process.

//進入高阻態讓上拉提高數據線電平並開始讀取過

pinMode(\_pin, INPUT\_PULLUP);

delay(1);

// First set data line low for a period according to sensor type

pinMode(\_pin, OUTPUT);

digitalWrite(\_pin, LOW);

//首先根據傳感器類型將數據線設置為低電平一段時間

switch (\_type) {//切換DHT狀態

case DHT22:

case DHT21:

delayMicroseconds(1100); // data sheet says "at least 1ms" 延遲1100ms

break;

case DHT11:

default:

delay(20); // data sheet says at least 18ms, 20ms just to be safe 延遲20ms

break;

}

uint32\_t cycles[80];

{

// End the start signal by setting data line high for 40 microseconds.

pinMode(\_pin, INPUT\_PULLUP);

// Delay a moment to let sensor pull data line low.

delayMicroseconds(pullTime);//延遲一些時間讓傳感氣

// Now start reading the data line to get the value from the DHT sensor.

// Turn off interrupts temporarily because the next sections

// are timing critical and we don't want any interruptions.

InterruptLock lock;

// First expect a low signal for ~80 microseconds followed by a high signal

// for ~80 microseconds again.

//高低電位tomeout訊息處理

if (expectPulse(LOW) == TIMEOUT) {

DEBUG\_PRINTLN(F("DHT timeout waiting for start signal low pulse."));

\_lastresult = false;

return \_lastresult;

}

if (expectPulse(HIGH) == TIMEOUT) {

DEBUG\_PRINTLN(F("DHT timeout waiting for start signal high pulse."));

\_lastresult = false;

return \_lastresult;

}

// Now read the 40 bits sent by the sensor. Each bit is sent as a 50

// microsecond low pulse followed by a variable length high pulse. If the

// high pulse is ~28 microseconds then it's a 0 and if it's ~70 microseconds

// then it's a 1. We measure the cycle count of the initial 50us low pulse

// and use that to compare to the cycle count of the high pulse to determine

// if the bit is a 0 (high state cycle count < low state cycle count), or a

// 1 (high state cycle count > low state cycle count). Note that for speed

// all the pulses are read into a array and then examined in a later step.

//讀取傳感器發送的40位。每一次發送為50

//微秒低脈衝，然後是可變長度的高脈衝。如果

//高脈衝約為28微秒，則為0；如果約為70微秒，則為1。

//測量初始50us低脈衝的周期數並將其與高脈衝的周期計數進行比較，以確定

//如果該位為0（高狀態循環計數<低狀態循環計數），或1（高狀態循環計數>低狀態循環計數）。

//注意速度將所有脈衝讀入陣列，然後在後續步驟中進行檢查。

for (int i = 0; i < 80; i += 2) {

cycles[i] = expectPulse(LOW);

cycles[i + 1] = expectPulse(HIGH);

}

} // Timing critical code is now complete.

// Inspect pulses and determine which ones are 0 (high state cycle count < low

// state cycle count), or 1 (high state cycle count > low state cycle count).

//檢查脈衝並確定哪一個為0（高狀態循環計數<低狀態週期計數）或1（高狀態週期計數>低狀態週期計數）。

for (int i = 0; i < 40; ++i) {

uint32\_t lowCycles = cycles[2 \* i];

uint32\_t highCycles = cycles[2 \* i + 1];

if ((lowCycles == TIMEOUT) || (highCycles == TIMEOUT)) {

DEBUG\_PRINTLN(F("DHT timeout waiting for pulse."));

\_lastresult = false;

return \_lastresult;

}

data[i / 8] <<= 1;

// Now compare the low and high cycle times to see if the bit is a 0 or 1.

if (highCycles > lowCycles) {

// High cycles are greater than 50us low cycle count, must be a 1.

data[i / 8] |= 1;

}

// Else high cycles are less than (or equal to, a weird case) the 50us low

// cycle count so this must be a zero. Nothing needs to be changed in the

// stored data.

}

//印出數值

DEBUG\_PRINTLN(F("Received from DHT:"));

DEBUG\_PRINT(data[0], HEX);

DEBUG\_PRINT(F(", "));

DEBUG\_PRINT(data[1], HEX);

DEBUG\_PRINT(F(", "));

DEBUG\_PRINT(data[2], HEX);

DEBUG\_PRINT(F(", "));

DEBUG\_PRINT(data[3], HEX);

DEBUG\_PRINT(F(", "));

DEBUG\_PRINT(data[4], HEX);

DEBUG\_PRINT(F(" =? "));

DEBUG\_PRINTLN((data[0] + data[1] + data[2] + data[3]) & 0xFF, HEX);

// Check we read 40 bits and that the checksum matches.

//檢查讀取了40個，並且校驗和匹配。

if (data[4] == ((data[0] + data[1] + data[2] + data[3]) & 0xFF)) {

\_lastresult = true;

return \_lastresult;

} else {

DEBUG\_PRINTLN(F("DHT checksum failure!"));

\_lastresult = false;

return \_lastresult;

}

}

// Expect the signal line to be at the specified level for a period of time and

// return a count of loop cycles spent at that level (this cycle count can be

// used to compare the relative time of two pulses). If more than a millisecond

// ellapses without the level changing then the call fails with a 0 response.

// This is adapted from Arduino's pulseInLong function (which is only available

// in the very latest IDE versions):

// https://github.com/arduino/Arduino/blob/master/hardware/arduino/avr/cores/arduino/wiring\_pulse.c

uint32\_t DHT::expectPulse(bool level) {

//期望信號線在一段時間內保持在指定水平，並且返回在該級別花費的循環週期數（此週期數可以是用於比較兩個脈衝的相對時間）。

//如果超過一毫秒

//在沒有更改級別的情況下消失，然後調用失敗，響應為0。

#if (F\_CPU > 16000000L)

uint32\_t count = 0;

#else

uint16\_t count = 0; // To work fast enough on slower AVR boards

#endif

// On AVR platforms use direct GPIO port access as it's much faster and better

// for catching pulses that are 10's of microseconds in length:

#ifdef \_\_AVR

uint8\_t portState = level ? \_bit : 0; //確認 portState = level true 回傳\_bit F ㄑ 回傳0

while ((\*portInputRegister(\_port) & \_bit) == portState) { //Register port值與\_bits值都等於 portState時

if (count++ >= \_maxcycles) {//如果數量 >= \_maxcycles

return TIMEOUT; // Exceeded timeout, fail.

}//回傳超時

}

// Otherwise fall back to using digitalRead (this seems to be necessary on

// ESP8266 right now, perhaps bugs in direct port access functions?).

#else

while (digitalRead(\_pin) == level) {//當 \_pin = level 時

if (count++ >= \_maxcycles) {//如果數量 >= \_maxcycles

return TIMEOUT; // Exceeded timeout, fail.

}//回傳超時

}

#endif

return count;//回傳數量

}