~\OneDrive\Documentos\PIP Lora\PIC\MOTEapp.c

```
/**
  ADC Generated Driver File

  @Company
    Microchip Technology Inc.

  @File Name
    adc.c

  @Summary
    This is the generated driver implementation file for the ADC driver using MPLAB® Code Configurator

  @Description
    This source file provides implementations for driver APIs for ADC.
    Generation Information :
        Product Revision  :  MPLAB® Code Configurator - v2.10.2
        Device            :  PIC16F1509
        Driver Version    :  2.00
    The generated drivers are tested against the following:
        Compiler          :  XC8 v1.33
        MPLAB             :  MPLAB X 2.26
*/

/*
Copyright (c) 2013 - 2015 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
*/

/**
```

```c
  Section: Included Files
*/

#include <xc.h>
#include "adc.h"
#include "mcc.h"

/**
  Section: Macro Declarations
*/

#define ACQ_US_DELAY 5

/**
  Section: ADC Module APIs
*/

void ADC_Initialize(void)
{
    // set the ADC to the options selected in the User Interface
    VREFCON0 = 0b10100000;
    // GO_nDONE stop; ADON enabled; CHS AN0;
    ADCON0 = 0x01;

    // TRIGSEL no_auto_trigger;
    ADCON1 = 0x08;

    // ADPREF VDD; ADFM right; ADCS FOSC/64;
    ADCON2 = 0b10001110;

    // ADRESL 0x0;
    ADRESL = 0x00;

    // ADRESH 0x0;
    ADRESH = 0x00;

}

void ADC_StartConversion(adc_channel_t channel)
{
    // select the A/D channel
    ADCON0bits.CHS = channel;

    // Turn on the ADC module
    ADCON0bits.ADON = 1;

    // Acquisition time delay
    __delay_us(ACQ_US_DELAY);

    // Start the conversion
```

```
98      ADCON0bits.GO_nDONE = 1;
99  }
100
101 bool ADC_IsConversionDone()
102 {
103     // Start the conversion
104     return (!ADCON0bits.GO_nDONE);
105 }
106
107 adc_result_t ADC_GetConversionResult(void)
108 {
109     // Conversion finished, return the result
110     return ((ADRESH << 8) + ADRESL);
111 }
112
113  adc_result_t ADC_GetConversion(adc_channel_t channel)
114 {
115     // Select the A/D channel
116     ADCON0bits.CHS = channel;
117
118     // Turn on the ADC module
119     ADCON0bits.ADON = 1;
120
121     // Acquisition time delay
122     __delay_us(ACQ_US_DELAY);
123
124     // Start the conversion
125     ADCON0bits.GO_nDONE = 1;
126
127     // Wait for the conversion to finish
128     while (ADCON0bits.GO_nDONE)
129     {
130     }
131
132     // Conversion finished, return the result
133     return ((ADRESH << 8) + ADRESL);
134 }
135
136  uint8_t ADC_TempConversion(adc_channel_t tempRaw)
137  {
138      uint8_t temp;
139
140      double val = ((tempRaw * 3.2)/1024);
141      val = ((val - 0.5)*100);
142      temp = (uint8_t) val;
143
144      return temp;
145  }
146  uint8_t ADC_HumSueConversion(uint16_t humsueRaw)
147  {
```

```
148    /* Datos optenidos del sensor prueba 11/12/2021
149     Sensor al aire= 3,000 V
150     Sustrato Húmedo= 1,12 V
151     Sustrato Húmedo+ = 0,88 V
152     Sustrato Húmedo ++ = 0,76 V
153     Sensor en Agua = 0,75 V
154     */
155      const double X1=416, X2=959, Y1=100, Y2=0;
156              uint8_t hums;
157
158      //double val = (((Y2-Y1)/(X2-X1))*(humsueRaw- X1) + Y1); // Desmenuzar Ecuación y ver
    resultados
159      double val =0.0;
160      val=(Y2-Y1)/(X2-X1);
161      val= val*(humsueRaw- X1);
162      val= val + Y1;
163      //val = ((val - 0.5)*100);
164      hums = (uint8_t) val;
165
166      return hums;
167  }
168 /**
169  End of File
170 */
```

~\OneDrive\Documentos\PIP Lora\PIC\DHT22_Ult.c

```
/*
 * File:   DHT22.c
 * Author: Fedhe
 *
 * Created on 15 de noviembre de 2021, 23:50
 */
/*
 * Interfacing DHT22 (AM2302) sensor with PIC18LF45k50
 * C Code for MPLAB XC8 compiler
 *
 *
 * Modificado de http://simple-circuit.com/
*/

#include <xc.h>
#include <stdint.h>
#include <string.h>
#include <stddef.h>
// Application Includes
#include "HardwareProfile.h"
#include "pin_manager.h"
/*#include "MOTEapp.h"
#include "USBapp.h"
#include "SSD1306oLED.h"*/

// Communication Includes
/*#include "eusart.h"
#include "buttons.h"
#include "TMRapp.h"*/
#include "tmr2.h"
//#include "tmr2.h"
#include "input.h"
#include "adc.h"
//#include "SSD1306oLED.h"
#include "memory.h"


// variables declaration
/*char Temperature[] = "Temp = 00.0 C  ";
char Humidity[]    = "RH   = 00.0 %  ";
*/
//  char Temperature[] = " 00.0 ";
//  char Humidity[]    = " 00.0 ";
  uint8_t TempHum[4];
  unsigned char T_Byte1, T_Byte2, RH_Byte1, RH_Byte2, CheckSum ;
  unsigned int Temp, RH;


```

```
49 void Start_Signal(void) {
50   DHT22_PIN_DIR = 0;     // configure DHT22_PIN as output
51   DHT22_PIN = 0;         // clear DHT22_PIN output (logic 0)
52
53   __delay_ms(25);        // wait 25 ms
54   DHT22_PIN = 1;         // set DHT22_PIN output (logic 1)
55
56   __delay_us(30);        // wait 30 us
57   DHT22_PIN_DIR = 1;     // configure DHT22_PIN as input
58 }
59 // Check sensor response
60 __bit Check_Response() {
61   TMR2 = 0;                 // reset Timer2
62
63   //TMR2ON = 1;                // enable Timer22 module
64  TMR2_StartTimer();
65
66   while(!DHT22_PIN && TMR2 < 75)  // wait until DHT22_PIN becomes high (checking of 80µs low
time response)
67   {                                  //EL ORIGINAL TENIA 100 SE CAMBIO A 150 QUE ES EL TIEMPO
EQUIVALENTE CON CRISTAL DE 48MHZ
68   if(TMR2 > 73)                     // if response time > 99µS  ==> Response error
69   {
70       return 0;                       // return 0 (Device has a problem with response)
71   }
72      }
73
74
75
76  TMR2 = 0;
77     while(DHT22_PIN && TMR2 < 75) // wait until DHT22_PIN becomes low (checking of 80µs high
time response)
78     {                                  //EL ORIGINAL TENIA 100 SE CAMBIO A 150 QUE ES EL
TIEMPO EQUIVALENTE CON CRISTAL DE 48MHZ
79     if(TMR2 > 73)                   // if response time > 99µS  ==> Response error
80     {
81          return 0;                     // return 0 (Device has a problem with response)
82     }
83          }
84
85       return 1;                      // return 1 (response OK)
86
87
88 }
89
90 // Data read function
91 __bit Read_Data(unsigned char* dht_data)
92 {
93   *dht_data = 0;
94
95   for(char i = 0; i < 8; i++)
```

```
  {
    TMR2 = 0;             // reset Timer1
    TMR2_StartTimer();
    while(!DHT22_PIN)      // wait until DHT22_PIN becomes high
    {
        if(TMR2 > 75) {    // if low time > 100  ==>  Time out error (Normally it takes 50µs)
        return 1;          //EL ORIGINAL TENIA 100 SE CAMBIO A 150 QUE ES EL TIEMPO EQUIVALENTE CON CRISTAL DE 48MHZ
      }
    }
    TMR2 = 0;             // reset Timer1


    while(DHT22_PIN)       // wait until DHT22_PIN becomes low

        if(TMR2 > 75) {    // if high time > 100  ==>  Time out error (Normally it takes 26-28µs for 0 and 70µs for 1)
        return 1;          // return 1 (timeout error)
      }                   //EL ORIGINAL TENIA 100 SE CAMBIO A 150 QUE ES EL TIEMPO EQUIVALENTE CON CRISTAL DE 48MHZ

     if(TMR2 > 38) {                // if high time > 50  ==>  Sensor sent 1
       *dht_data |= (1 << (7 - i));  // set bit (7 - i)
  }                                   //EL ORIGINAL TENIA 50 SE CAMBIO A 37 QUE ES EL TIEMPO EQUIVALENTE CON CRISTAL DE 48MHZ
  }
  return 0;                          // return 0 (data read OK)
}
void backUp(void)
{
    NOP();

}

void DHT22_GetTempHum(uint8_t * vDest)
{
    /*

  unsigned char T_Byte1, T_Byte2, RH_Byte1, RH_Byte2, CheckSum ;
  unsigned int Temp, RH;*/
   //static char Temp1[]=" 00.0 " ;

   //static char dataDHT[2][6];
  /*OSCCON = 0X70;        // set internal oscillator to 8MHz
  ANSELH = 0;           // configure all PORTB pins as digital
  */
  //T1CON  = 0x10;        // set Timer3 clock source to internal with 1:2 prescaler (Timer1 clock = 1MHz)
  TMR2 = 0;           // reset Timer1


```

```
142   __delay_ms(10);     // wait 1 second
143
144      Start_Signal();     // send start signal to the sensor
145
146     if(Check_Response())    // check if there is a response from sensor (If OK start reading
humidity and temperature data)
147     {
148       // read (and save) data from the DHT22 sensor and check time out errors
149       if(Read_Data(&RH_Byte1) || Read_Data(&RH_Byte2) || Read_Data(&T_Byte1) ||
Read_Data(&T_Byte2) || Read_Data(&CheckSum))
150       {
151         /*LCD_Cmd(LCD_CLEAR);       // clear LCD
152         LCD_Goto(5, 1);           // go to column 5, row 2
153         LCD_Print("Time out!");   // display "Time out!"
154         */
155        // Code para falla de lectura
156           /*
157           Temperature[1] ='T';
158           Temperature[2] ='m';
159           Temperature[3] ='O';*/
160
161            TempHum[2]=0x6E; //110 significa Time Out
162            TempHum[3]=0x00;
163
164       }
165
166       else         // if there is no time out error
167       {
168         if(CheckSum == ((RH_Byte1 + RH_Byte2 + T_Byte1 + T_Byte2) & 0xFF))
169         {             // if there is no checksum error
170             TempHum[0]=T_Byte1;
171             TempHum[1]=T_Byte2;
172             TempHum[2]=RH_Byte1;
173             TempHum[3]=RH_Byte2;
174
175
176           /*
177           RH = RH_Byte1;
178           RH = (RH << 8) | RH_Byte2;
179           Temp = T_Byte1;
180           Temp = (Temp << 8) | T_Byte2;
181
182           if (Temp > 0X8000)
183           {                         // if temperature < 0
184             Temperature[0] = '-';   // put minus sign '-'
185             Temp = Temp & 0X7FFF;
186           }
187           else                     // otherwise (temperature > 0)
188           {
189           Temperature[0] = ' ';  // put space ' '
```

```
          Temperature[1]  = (Temp / 100) % 10  + '0';
          Temperature[2]  = (Temp / 10) % 10   + '0';
          Temperature[4] =  Temp % 10  + '0';
          //Temperature[5] = 223;    // put degree symbol (°)
          }
         // DataDHT22[1]={Temperature[0], Temperature[1], Temperature[2], Temperature[3], Temperature[4], Temperature[5]};

          if(RH == 1000)              // if the relative humidity = 100.0 %
           {
            Humidity[0]  = 1 + '0';   // put 1 of hundreds
           }
          else
          {
          Humidity[0]  = ' ';       // put space ' '
          Humidity[1]  = (RH / 100) % 10 + '0';
          Humidity[2]  = (RH / 10) % 10  + '0';
          Humidity[4] = RH % 10 + '0';
          // DataDHT22[2][]=Humidity;
          }
         */
        }

        // if there is a checksum error
        else
        {

         // Code para falla de checksum
            NOP();
            /*Temperature[1] ='C';
            Temperature[2] ='h';
            Temperature[3] ='S';*/
            TempHum[2]=0x78; //120 significa error CheckSum
            TempHum[3]=0x00;

        }

      }
    }

    // if there is a response (from the sensor) problem
    else
    {
        NOP();
         /* Temperature[1] ='N';
          Temperature[2] ='o';
          Temperature[3] ='R';
          Temperature[3] =' ';*/
          TempHum[2]=0x82; //130 significa No Response
          TempHum[3]=0x00;
```

```
239
240
241      // Code por falla de comunicación
242     }
243
244     //TMR1ON = 0;        // disable Timer1 module
245     //TMR2_StopTimer();
246     __delay_ms(20);  // wait 1 second
247
248
249  //DataDHT[2][6]={{Temperature[0], Temperature[1], Temperature[2], Temperature[3],
    Temperature[4], Temperature[5]},
250   //{Humidity[0], Humidity[1], Humidity[2], Humidity[3], Humidity[4], Humidity[5]}};
251   //DataDHT22[1]=Temperature;
252   //DataDHT22[2]=Humidity;
253 //  DataDHT22={{Temperature},{Humidity}};
254    // TempHum = {Temperature,Humidity};
255    // TempHum= strcat(TempHum, Humidity);
256           /*   TempHum[0]=0x89;
257             TempHum[1]=0x3E;
258             TempHum[2]=0x19;
259             TempHum[3]=0x10;*/
260     for(int i=0;i<4;i++){
261         *vDest++ = TempHum[i];
262
263     }
264
265
266
267
268 }
269 // End of code.
```

~\OneDrive\Documentos\PIP Lora\PIC\adc.c

```
/**
  ADC Generated Driver File

  @Company
    Microchip Technology Inc.

  @File Name
    adc.c

  @Summary
    This is the generated driver implementation file for the ADC driver using MPLAB® Code Configurator

  @Description
    This source file provides implementations for driver APIs for ADC.
    Generation Information :
        Product Revision  :  MPLAB® Code Configurator - v2.10.2
        Device            :  PIC16F1509
        Driver Version    :  2.00
    The generated drivers are tested against the following:
        Compiler          :  XC8 v1.33
        MPLAB             :  MPLAB X 2.26
*/

/*
Copyright (c) 2013 - 2015 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
*/

/**
```

```c
  Section: Included Files
*/

#include <xc.h>
#include "adc.h"
#include "mcc.h"

/**
  Section: Macro Declarations
*/

#define ACQ_US_DELAY 5

/**
  Section: ADC Module APIs
*/

void ADC_Initialize(void)
{
    // set the ADC to the options selected in the User Interface
    VREFCON0 = 0b10100000;
    // GO_nDONE stop; ADON enabled; CHS AN0;
    ADCON0 = 0x01;

    // TRIGSEL no_auto_trigger;
    ADCON1 = 0x08;

    // ADPREF VDD; ADFM right; ADCS FOSC/64;
    ADCON2 = 0b10001110;

    // ADRESL 0x0;
    ADRESL = 0x00;

    // ADRESH 0x0;
    ADRESH = 0x00;

}

void ADC_StartConversion(adc_channel_t channel)
{
    // select the A/D channel
    ADCON0bits.CHS = channel;

    // Turn on the ADC module
    ADCON0bits.ADON = 1;

    // Acquisition time delay
    __delay_us(ACQ_US_DELAY);

    // Start the conversion
```

```
98      ADCON0bits.GO_nDONE = 1;
99  }
100
101 bool ADC_IsConversionDone()
102 {
103     // Start the conversion
104     return (!ADCON0bits.GO_nDONE);
105 }
106
107 adc_result_t ADC_GetConversionResult(void)
108 {
109     // Conversion finished, return the result
110     return ((ADRESH << 8) + ADRESL);
111 }
112
113  adc_result_t ADC_GetConversion(adc_channel_t channel)
114 {
115     // Select the A/D channel
116     ADCON0bits.CHS = channel;
117
118     // Turn on the ADC module
119     ADCON0bits.ADON = 1;
120
121     // Acquisition time delay
122     __delay_us(ACQ_US_DELAY);
123
124     // Start the conversion
125     ADCON0bits.GO_nDONE = 1;
126
127     // Wait for the conversion to finish
128     while (ADCON0bits.GO_nDONE)
129     {
130     }
131
132     // Conversion finished, return the result
133     return ((ADRESH << 8) + ADRESL);
134 }
135
136  uint8_t ADC_TempConversion(adc_channel_t tempRaw)
137  {
138      uint8_t temp;
139
140      double val = ((tempRaw * 3.2)/1024);
141      val = ((val - 0.5)*100);
142      temp = (uint8_t) val;
143
144      return temp;
145  }
146  uint8_t ADC_HumSueConversion(uint16_t humsueRaw)
147  {
```

```
148     /* Datos optenidos del sensor prueba 11/12/2021
149      Sensor al aire= 3,000 V
150      Sustrato Húmedo= 1,12 V
151      Sustrato Húmedo+ = 0,88 V
152      Sustrato Húmedo ++ = 0,76 V
153      Sensor en Agua = 0,75 V
154      */
155       const double X1=416, X2=959, Y1=100, Y2=0;
156               uint8_t hums;
157
158       //double val = (((Y2-Y1)/(X2-X1))*(humsueRaw- X1) + Y1); // Desmenuzar Ecuación y ver
    resultados
159       double val =0.0;
160       val=(Y2-Y1)/(X2-X1);
161       val= val*(humsueRaw- X1);
162       val= val + Y1;
163       //val = ((val - 0.5)*100);
164       hums = (uint8_t) val;
165
166       return hums;
167   }
168 /**
169   End of File
170 */
```