pt1000 tx

Übersicht

Das erste Projekt was einem im Sinne von IoT einfällt ist wohl ein Temperatur Sensor. Daher hier ein kleiner Sketch um einen PT1000 auszulesen und dessen Daten zum Gateway zu senden.

Alles wie immer ohne Gewähr und Garantie. Ich kenne mich mit den Funkstandards wenig aus!

Links

https://www.mikrocontroller.net/attachment/242568/pt1000.ino

Bitte die Librarys von hier installieren!, hier habe ich auch den Original Code her:Simple_temp
https://github.com/CongducPham/LowCostLoRaGw

http://www.libelium.com/development/waspmote/documentation/waspmote-lora-868mhz-915mhz-sx1272-networking-guide

Material

Folgendes wird benötigt:

- PT1000
- 1000Ohm Widerstand
- Arduino
- SX1278 (Oder SX1276)

Aufbau

Spannungsteiler: AREF → 1000Ohm A0 → PT1000→ GND Also ein 1:2 Spannungsteiler Oder von pin 8 → 1000Ohm ….. Für Low Power, der Sensor wird beim nicht gebrauch abgeschaltet.

Code

PT1000_TX.ino

```
/*
 *  temperature sensor on analog 8 to test the LoRa gateway
 *  extended version with AES and custom Carrier Sense features
 *
```

```
 *  Copyright (C) 2016 Congduc Pham, University of Pau, France
 *
 *  This program 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.

 *  This program 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 the program.  If not, see
<http://www.gnu.org/licenses/>.
 *
*****************************************************************
******
 * last update: Nov. 26th by C. Pham
 */
#include <SPI.h>
// Include the SX1272
#include "SX1272.h"

// IMPORTANT
///////////////////////////////////////////////////////////////////
///////////////////////////////////
// please uncomment only 1 choice
//
#define ETSI_EUROPE_REGULATION
//#define FCC_US_REGULATION
//#define SENEGAL_REGULATION
///////////////////////////////////////////////////////////////////
///////////////////////////////////

// IMPORTANT
///////////////////////////////////////////////////////////////////
///////////////////////////////////
// please uncomment only 1 choice
//#define BAND868
//#define BAND900
#define BAND433
///////////////////////////////////////////////////////////////////
///////////////////////////////////

#ifdef ETSI_EUROPE_REGULATION
#define MAX_DBM 14
// previous way for setting output power
```

```
// char powerLevel='M';
#elif defined SENEGAL_REGULATION
#define MAX_DBM 10
// previous way for setting output power
// 'H' is actually 6dBm, so better to use the new way to set output
power
// char powerLevel='H';
#elif defined FCC_US_REGULATION
#define MAX_DBM 14
#endif

#ifdef BAND868
#ifdef SENEGAL_REGULATION
const uint32_t DEFAULT_CHANNEL=CH_04_868;
#else
const uint32_t DEFAULT_CHANNEL=CH_10_868;
#endif
#elif defined BAND900
const uint32_t DEFAULT_CHANNEL=CH_05_900;
#elif defined BAND433
const uint32_t DEFAULT_CHANNEL=CH_00_433;
#endif

// IMPORTANT
/////////////////////////////////////////////////////////////////////
///////////////////////////////////
//
// uncomment if your radio is an HopeRF RFM92W, HopeRF RFM95W,
Modtronix inAir9B, NiceRF1276
// or you known from the circuit diagram that output use the PABOOST
line instead of the RFO line
//#define PABOOST
/////////////////////////////////////////////////////////////////////
///////////////////////////////////

/////////////////////////////////////////////////////////////////////
// COMMENT OR UNCOMMENT TO CHANGE FEATURES.
// ONLY IF YOU KNOW WHAT YOU ARE DOING!!! OTHERWISE LEAVE AS IT IS
#if not defined _VARIANT_ARDUINO_DUE_X_ && not defined __SAMD21G18A__
#define WITH_EEPROM
#endif
//#define WITH_APPKEY
#define FLOAT_TEMP
#define NEW_DATA_FIELD
#define LOW_POWER
#define LOW_POWER_HIBERNATE
#define CUSTOM_CS
//#define LORA_LAS
//#define WITH_AES
//#define LORAWAN
//#define TO_LORAWAN_GW
```

```
#define WITH_ACK
/////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////
// CHANGE HERE THE LORA MODE, NODE ADDRESS
#define LORAMODE  1
#define node_addr 6
/////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////
// CHANGE HERE THE THINGSPEAK FIELD BETWEEN 1 AND 4
#define field_index 3
/////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////
// CHANGE HERE THE READ PIN AND THE POWER PIN FOR THE TEMP. SENSOR
#define TEMP_PIN_READ  A0
// use digital 8 to power the temperature sensor if needed
#define TEMP_PIN_POWER 8
/////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////
// CHANGE HERE THE TIME IN MINUTES BETWEEN 2 READING & TRANSMISSION
unsigned int idlePeriodInMin = 10;
/////////////////////////////////////////////////////////////////

#ifdef WITH_APPKEY
/////////////////////////////////////////////////////////////////
// CHANGE HERE THE APPKEY, BUT IF GW CHECKS FOR APPKEY, MUST BE
// IN THE APPKEY LIST MAINTAINED BY GW.
uint8_t my_appKey[4]={5, 6, 7, 8};
/////////////////////////////////////////////////////////////////
#endif

// we wrapped Serial.println to support the Arduino Zero or M0
#if defined __SAMD21G18A__ && not defined ARDUINO_SAMD_FEATHER_M0
#define PRINTLN                   SerialUSB.println("")
#define PRINT_CSTSTR(fmt,param)   SerialUSB.print(F(param))
#define PRINT_STR(fmt,param)      SerialUSB.print(param)
#define PRINT_VALUE(fmt,param)    SerialUSB.print(param)
#define PRINT_HEX(fmt,param)      SerialUSB.print(param,HEX)
#define FLUSHOUTPUT               SerialUSB.flush();
#else
#define PRINTLN                   Serial.println("")
#define PRINT_CSTSTR(fmt,param)   Serial.print(F(param))
#define PRINT_STR(fmt,param)      Serial.print(param)
#define PRINT_VALUE(fmt,param)    Serial.print(param)
#define PRINT_HEX(fmt,param)      Serial.print(param,HEX)
#define FLUSHOUTPUT               Serial.flush();
#endif
```

```
#ifdef WITH_EEPROM
#include <EEPROM.h>
#endif

#define DEFAULT_DEST_ADDR 1

#ifdef WITH_ACK
#define NB_RETRIES 15
#endif

#if defined ARDUINO_AVR_PRO || defined ARDUINO_AVR_MINI || defined
__MK20DX256__ || defined __MKL26Z64__ || defined __MK64FX512__ ||
defined __MK66FX1M0__ || defined __SAMD21G18A__
  // if you have a Pro Mini running at 5V, then change here
  // these boards work in 3.3V
  // Nexus board from Ideetron is a Mini
  // __MK66FX1M0__ is for Teensy36
  // __MK64FX512__  is for Teensy35
  // __MK20DX256__ is for Teensy31/32
  // __MKL26Z64__ is for TeensyLC
  // __SAMD21G18A__ is for Zero/M0 and FeatherM0 (Cortex-M0)
  #define TEMP_SCALE  3300.0
#else // ARDUINO_AVR_NANO || defined ARDUINO_AVR_UNO || defined
ARDUINO_AVR_MEGA2560
  // also for all other boards, so change here if required.
  #define TEMP_SCALE  5000.0
#endif

#ifdef LOW_POWER
// this is for the Teensy36, Teensy35, Teensy31/32 & TeensyLC
// need v6 of Snooze library
#if defined __MK20DX256__ || defined __MKL26Z64__ || defined
__MK64FX512__ || defined __MK66FX1M0__
#define LOW_POWER_PERIOD 60
#include <Snooze.h>
SnoozeTimer timer;
SnoozeBlock sleep_config(timer);
#else
#define LOW_POWER_PERIOD 8
// you need the LowPower library from RocketScream
// https://github.com/rocketscream/Low-Power
#include "LowPower.h"

#ifdef __SAMD21G18A__
// use the RTC library
#include "RTCZero.h"
/* Create an rtc object */
RTCZero rtc;
#endif
#endif
```

```
unsigned int nCycle = idlePeriodInMin*60/LOW_POWER_PERIOD;
#endif

#ifdef WITH_AES

#include "AES-128_V10.h"
#include "Encrypt_V31.h"

unsigned char AppSkey[16] = {
  0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6,
  0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C
};

unsigned char NwkSkey[16] = {
  0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6,
  0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C
};

unsigned char DevAddr[4] = {
  0x00, 0x00, 0x00, node_addr
};

uint16_t Frame_Counter_Up = 0x0000;
// we use the same convention than for LoRaWAN as we will use the same
AES convention
// See LoRaWAN specifications
unsigned char Direction = 0x00;
#endif

#ifdef LORA_LAS
#include "LoRaActivitySharing.h"
// acting as an end-device
LASDevice loraLAS(node_addr,LAS_DEFAULT_ALPHA,DEFAULT_DEST_ADDR);
#endif

double temp;
unsigned long nextTransmissionTime=0L;
char float_str[20];
uint8_t message[100];

#ifdef TO_LORAWAN_GW
int loraMode=1;
#else
int loraMode=LORAMODE;
#endif

#ifdef WITH_EEPROM
struct sx1272config {

  uint8_t flag1;
```

```
  uint8_t flag2;
  uint8_t seq;
  // can add other fields such as LoRa mode,...
};

sx1272config my_sx1272config;
#endif

// receive window
uint16_t w_timer=1000;

#ifdef CUSTOM_CS
unsigned long startDoCad, endDoCad;
bool extendedIFS=true;
bool RSSIonSend=true;
uint8_t SIFS_cad_number;
uint8_t send_cad_number;
uint8_t SIFS_value[11]={0, 183, 94, 44, 47, 23, 24, 12, 12, 7, 4};
uint8_t CAD_value[11]={0, 62, 31, 16, 16, 8, 9, 5, 3, 1, 1};

// we could use the CarrierSense function added in the SX1272 library,
but it is more convenient to duplicate it here
// so that we could easily modify it for testing
void CarrierSense() {

  bool carrierSenseRetry=false;
  int e;

  if (send_cad_number) {
    do {
      do {

        // check for free channel (SIFS/DIFS)
        startDoCad=millis();
        e = sx1272.doCAD(send_cad_number);
        endDoCad=millis();

        PRINT_CSTSTR("%s","--> CAD duration ");
        PRINT_VALUE("%ld",endDoCad-startDoCad);
        PRINTLN;

        if (!e) {
          PRINT_CSTSTR("%s","OK1\n");

          if (extendedIFS)  {
            // wait for random number of CAD
#ifdef ARDUINO
            uint8_t w = random(1,8);
#else
            uint8_t w = rand() % 8 + 1;
#endif
```

```
            PRINT_CSTSTR("%s","--> waiting for ");
            PRINT_VALUE("%d",w);
            PRINT_CSTSTR("%s"," CAD = ");
            PRINT_VALUE("%d",CAD_value[loraMode]*w);
            PRINTLN;

            delay(CAD_value[loraMode]*w);

            // check for free channel (SIFS/DIFS) once again
            startDoCad=millis();
            e = sx1272.doCAD(send_cad_number);
            endDoCad=millis();

            PRINT_CSTSTR("%s","--> CAD duration ");
            PRINT_VALUE("%ld",endDoCad-startDoCad);
            PRINTLN;

            if (!e)
              PRINT_CSTSTR("%s","OK2");
            else
              PRINT_CSTSTR("%s","###2");

            PRINTLN;
          }
        }
        else {
          PRINT_CSTSTR("%s","###1\n");

          // wait for random number of DIFS
#ifdef ARDUINO
          uint8_t w = random(1,8);
#else
          uint8_t w = rand() % 8 + 1;
#endif

          PRINT_CSTSTR("%s","--> waiting for ");
          PRINT_VALUE("%d",w);
          PRINT_CSTSTR("%s"," DIFS (DIFS=3SIFS) = ");
          PRINT_VALUE("%d",SIFS_value[loraMode]*3*w);
          PRINTLN;

          delay(SIFS_value[loraMode]*3*w);

          PRINT_CSTSTR("%s","--> retry\n");
        }

      } while (e);

      // CAD is OK, but need to check RSSI
```

```
      if (RSSIonSend) {

          e=sx1272.getRSSI();

          uint8_t rssi_retry_count=10;

          if (!e) {

            PRINT_CSTSTR("%s","--> RSSI ");
            PRINT_VALUE("%d", sx1272._RSSI);
            PRINTLN;

            while (sx1272._RSSI > -90 && rssi_retry_count) {

              delay(1);
              sx1272.getRSSI();
              PRINT_CSTSTR("%s","--> RSSI ");
              PRINT_VALUE("%d",  sx1272._RSSI);
              PRINTLN;
              rssi_retry_count--;
            }
          }
          else
            PRINT_CSTSTR("%s","--> RSSI error\n");

          if (!rssi_retry_count)
            carrierSenseRetry=true;
          else
      carrierSenseRetry=false;
      }

    } while (carrierSenseRetry);
  }
}
#endif


#define DEF_BAUDRATE      19200
#define ADC_MAX           1023            // bei 10 Bit
#define PT_IN_0            0              // analog Eingang A0
#define UEBERLAUF         9999            // Fehler bei offenem Eingang

#define R_PT0             1000.0          // bei 0 Grad C
#define R_REF             1000.0          // ext. Widerstand mit 0,1%
#define R_ZULEITUNG       0               // bei laengeren Leitungen
unbedingt anpassen
#define PT_FAKTOR         3.85            // schon mit 1000.0 skaliert

int lese_PT1000(int ad_eingang)           // liest den ADC-Eingang und
rechnet auf Grad Celsius um
{
```

```
int temperatur, ADC_wert;
float PT_x;
    analogReference(DEFAULT);             // AREF einschalten
    analogRead(PT_IN_0);                  // ADC in Betrieb nehmen
    ADC_wert = analogRead(ad_eingang);    // und Kanal messen
    analogReference(EXTERNAL);
    analogRead(7);                        // AREF wieder abschalten:
geht nur so
    if(ADC_wert < ADC_MAX) {              // nur gueltige Werte
auswerten
      PT_x = R_REF * ADC_wert / (ADC_MAX-ADC_wert);   // Widerstand
ausrechnen
      PT_x = PT_x - R_PT0 - R_ZULEITUNG;  // Offset fuer 0 Grad
abziehen
      temperatur = PT_x / PT_FAKTOR;      // und auf Grad C skalieren
    } else temperatur = UEBERLAUF;        // falls PT1000-Zuleitung
offen
    return temperatur;
}

void setup()
{
  int e;

  // for the temperature sensor
  pinMode(TEMP_PIN_READ, INPUT);
  // and to power the temperature sensor
  pinMode(TEMP_PIN_POWER,OUTPUT);

#ifdef LOW_POWER
#ifdef __SAMD21G18A__
  rtc.begin();
#endif
#else
  digitalWrite(TEMP_PIN_POWER,HIGH);
#endif

  delay(3000);
  // Open serial communications and wait for port to open:
#if defined __SAMD21G18A__ && not defined ARDUINO_SAMD_FEATHER_M0
  SerialUSB.begin(38400);
#else
  Serial.begin(38400);
#endif
  // Print a start message
  PRINT_CSTSTR("%s","LoRa temperature sensor, extended version\n");

#ifdef ARDUINO_AVR_PRO
  PRINT_CSTSTR("%s","Arduino Pro Mini detected\n");
#endif
```

```
#ifdef ARDUINO_AVR_NANO
  PRINT_CSTSTR("%s","Arduino Nano detected\n");
#endif

#ifdef ARDUINO_AVR_MINI
  PRINT_CSTSTR("%s","Arduino MINI/Nexus detected\n");
#endif

#ifdef __MK20DX256__
  PRINT_CSTSTR("%s","Teensy31/32 detected\n");
#endif

#ifdef __MKL26Z64__
  PRINT_CSTSTR("%s","TeensyLC detected\n");
#endif

#ifdef __SAMD21G18A__
  PRINT_CSTSTR("%s","Arduino M0/Zero detected\n");
#endif

  // Power ON the module
  sx1272.ON();

#ifdef WITH_EEPROM
  // get config from EEPROM
  EEPROM.get(0, my_sx1272config);

  // found a valid config?
  if (my_sx1272config.flag1==0x12 && my_sx1272config.flag2==0x34) {
    PRINT_CSTSTR("%s","Get back previous sx1272 config\n");

    // set sequence number for SX1272 library
    sx1272._packetNumber=my_sx1272config.seq;
    PRINT_CSTSTR("%s","Using packet sequence number of ");
    PRINT_VALUE("%d", sx1272._packetNumber);
    PRINTLN;
  }
  else {
    // otherwise, write config and start over
    my_sx1272config.flag1=0x12;
    my_sx1272config.flag2=0x34;
    my_sx1272config.seq=sx1272._packetNumber;
  }
#endif

  // Set transmission mode and print the result
  e = sx1272.setMode(loraMode);
  PRINT_CSTSTR("%s","Setting Mode: state ");
  PRINT_VALUE("%d", e);
  PRINTLN;
```

```
  if (loraMode==1)
    w_timer=2500;

#ifdef LORA_LAS
  loraLAS.setSIFS(loraMode);
#endif

#ifdef CUSTOM_CS
  if (loraMode>7)
    SIFS_cad_number=6;
  else
    SIFS_cad_number=3;

  // SIFS=3CAD and DIFS=3SIFS
  // here we use a DIFS prior to data transmission
  send_cad_number=3*SIFS_cad_number;

#ifdef LOW_POWER
  // TODO: with low power, when setting the radio module in sleep mode
  // there seem to be some issue with RSSI reading
  RSSIonSend=false;
#endif

#else
  // enable carrier sense
  sx1272._enableCarrierSense=true;
#ifdef LOW_POWER
  // TODO: with low power, when setting the radio module in sleep mode
  // there seem to be some issue with RSSI reading
  sx1272._RSSIonSend=false;
#endif
#endif

  // Select frequency channel
  e = sx1272.setChannel(DEFAULT_CHANNEL);
  PRINT_CSTSTR("%s","Setting Channel: state ");
  PRINT_VALUE("%d", e);
  PRINTLN;

  // Select amplifier line; PABOOST or RFO
#ifdef PABOOST
  sx1272._needPABOOST=true;
  // previous way for setting output power
  // powerLevel='x';
#else
  // previous way for setting output power
  // powerLevel='M';
#endif
```

```
  // previous way for setting output power
  // e = sx1272.setPower(powerLevel);

  e = sx1272.setPowerDBM((uint8_t)MAX_DBM);

  PRINT_CSTSTR("%s","Setting Power: state ");
  PRINT_VALUE("%d", e);
  PRINTLN;

  // Set the node address and print the result
  e = sx1272.setNodeAddress(node_addr);
  PRINT_CSTSTR("%s","Setting node addr: state ");
  PRINT_VALUE("%d", e);
  PRINTLN;

#ifdef TO_LORAWAN_GW
  e = sx1272.setSyncWord(0x34);
  PRINT_CSTSTR("%s","Set sync word to 0x34 state ");
  PRINT_VALUE("%d", e);
  PRINTLN;
#endif

  // Print a success message
  PRINT_CSTSTR("%s","SX1272 successfully configured\n");

#ifdef LORA_LAS
  loraLAS.ON(LAS_ON_WRESET);
#endif

  //printf_begin();
  delay(500);
}

#if not defined _VARIANT_ARDUINO_DUE_X_ && defined FLOAT_TEMP

char *ftoa(char *a, double f, int precision)
{
 long p[] = {0,10,100,1000,10000,100000,1000000,10000000,100000000};

 char *ret = a;
 long heiltal = (long)f;
 itoa(heiltal, a, 10);
 while (*a != '\0') a++;
 *a++ = '.';
 long desimal = abs((long)((f - heiltal) * p[precision]));
 itoa(desimal, a, 10);
 return ret;
}
#endif

void loop(void)
```

```
{
  long startSend;
  long endSend;
  uint8_t app_key_offset=0;
  int e;

#ifdef LORA_LAS
  // call periodically to be able to detect the start of a new cycle
  loraLAS.checkCycle();
#else
  //if (loraLAS._has_init && (millis()-lastTransmissionTime > 120000))
{
#endif

#ifndef LOW_POWER
  // 600000+random(15,60)*1000
  if (millis() > nextTransmissionTime) {
#endif

#ifdef LOW_POWER
      digitalWrite(TEMP_PIN_POWER,HIGH);
      // security?
      delay(200);
      int value = analogRead(TEMP_PIN_READ);

#else
      int value = analogRead(TEMP_PIN_READ);
#endif

      // change here how the temperature should be computed depending
on your sensor type
      //
      temp = lese_PT1000(PT_IN_0);
      digitalWrite(TEMP_PIN_POWER,LOW);
      PRINT_CSTSTR("%s","Reading ");
      PRINT_VALUE("%d", value);
      PRINTLN;

      //temp = temp - 0.5;
      //temp = temp / 10.0;

      PRINT_CSTSTR("%s","Temp is ");
      PRINT_VALUE("%f", temp);
      PRINTLN;

#if defined WITH_APPKEY && not defined WITH_AES
      app_key_offset = sizeof(my_appKey);
      // set the app key in the payload
      memcpy(message,my_appKey,app_key_offset);
#endif
```

```
      uint8_t r_size;

      // then use app_key_offset to skip the app key
#ifdef _VARIANT_ARDUINO_DUE_X_
#ifdef NEW_DATA_FIELD
      r_size=sprintf((char*)message+app_key_offset, "\\!#%d#TC/%.2f",
field_index, temp);
#else
      r_size=sprintf((char*)message+app_key_offset, "\\!#%d#%.2f",
field_index, temp);
#endif
#else

#ifdef FLOAT_TEMP
      ftoa(float_str,temp,2);

#ifdef NEW_DATA_FIELD
      r_size=sprintf((char*)message+app_key_offset, "\\!#%d#TC/%s",
field_index, float_str);
#else
      // this is for testing, uncomment if you just want to test,
without a real temp sensor plugged
      //strcpy(float_str, "21.55567");
      r_size=sprintf((char*)message+app_key_offset, "\\!#%d#%s",
field_index, float_str);
#endif

#else

#ifdef NEW_DATA_FIELD
      r_size=sprintf((char*)message+app_key_offset, "\\!#%d#TC/%d",
field_index, (int)temp);
#else
      r_size=sprintf((char*)message+app_key_offset, "\\!#%d#%d",
field_index, (int)temp);
#endif
#endif
#endif

      PRINT_CSTSTR("%s","Sending ");
      PRINT_STR("%s",(char*)(message+app_key_offset));
      PRINTLN;

      PRINT_CSTSTR("%s","Real payload size is ");
      PRINT_VALUE("%d", r_size);
      PRINTLN;

      int pl=r_size+app_key_offset;

#ifdef WITH_AES
```

```
      // if encryption then we DO NOT use appkey
      //
      PRINT_STR("%s",(char*)message);
      PRINTLN;
      PRINT_CSTSTR("%s","plain payload hex\n");
      for (int i=0; i<pl;i++) {
        if (message[i]<16)
          PRINT_CSTSTR("%s","0");
        PRINT_HEX("%X", message[i]);
        PRINT_CSTSTR("%s"," ");
      }
      PRINTLN;

      PRINT_CSTSTR("%s","Encrypting\n");
      PRINT_CSTSTR("%s","encrypted payload\n");
      Encrypt_Payload((unsigned char*)message, pl, Frame_Counter_Up,
Direction);
      //Print encrypted message
      for(int i = 0; i < pl; i++)
      {
        if (message[i]<16)
          PRINT_CSTSTR("%s","0");
        PRINT_HEX("%X", message[i]);
        PRINT_CSTSTR("%s"," ");
      }
      PRINTLN;

      // with encryption, we use for the payload a LoRaWAN packet
format to reuse available LoRaWAN encryption libraries
      //
      unsigned char LORAWAN_Data[256];
      unsigned char LORAWAN_Package_Length;
      unsigned char MIC[4];
      //Unconfirmed data up
      unsigned char Mac_Header = 0x40;
      // no ADR, not an ACK and no options
      unsigned char Frame_Control = 0x00;
      // with application data so Frame_Port = 1..223
      unsigned char Frame_Port = 0x01;

      //Build the Radio Package, LoRaWAN format
      //See LoRaWAN specification
      LORAWAN_Data[0] = Mac_Header;

      LORAWAN_Data[1] = DevAddr[3];
      LORAWAN_Data[2] = DevAddr[2];
      LORAWAN_Data[3] = DevAddr[1];
      LORAWAN_Data[4] = DevAddr[0];

      LORAWAN_Data[5] = Frame_Control;
```

```
      LORAWAN_Data[6] = (Frame_Counter_Up & 0x00FF);
      LORAWAN_Data[7] = ((Frame_Counter_Up >> 8) & 0x00FF);

      LORAWAN_Data[8] = Frame_Port;

      //Set Current package length
      LORAWAN_Package_Length = 9;

      //Load Data
      for(int i = 0; i < r_size; i++)
      {
        // see that we don't take the appkey, just the encrypted data
that starts that message[app_key_offset]
        LORAWAN_Data[LORAWAN_Package_Length + i] = message[i];
      }

      //Add data Lenth to package length
      LORAWAN_Package_Length = LORAWAN_Package_Length + r_size;

      PRINT_CSTSTR("%s","calculate MIC with NwkSKey\n");
      //Calculate MIC
      Calculate_MIC(LORAWAN_Data, MIC, LORAWAN_Package_Length,
Frame_Counter_Up, Direction);

      //Load MIC in package
      for(int i=0; i < 4; i++)
      {
        LORAWAN_Data[i + LORAWAN_Package_Length] = MIC[i];
      }

      //Add MIC length to package length
      LORAWAN_Package_Length = LORAWAN_Package_Length + 4;

      PRINT_CSTSTR("%s","transmitted LoRaWAN-like packet:\n");
      PRINT_CSTSTR("%s","MHDR[1] | DevAddr[4] | FCtrl[1] | FCnt[2] |
FPort[1] | EncryptedPayload | MIC[4]\n");
      //Print transmitted data
      for(int i = 0; i < LORAWAN_Package_Length; i++)
      {
        if (LORAWAN_Data[i]<16)
          PRINT_CSTSTR("%s","0");
        PRINT_HEX("%X", LORAWAN_Data[i]);
        PRINT_CSTSTR("%s"," ");
      }
      PRINTLN;

      // copy back to message
      memcpy(message,LORAWAN_Data,LORAWAN_Package_Length);
      pl = LORAWAN_Package_Length;
```

```
#ifdef LORAWAN
      PRINT_CSTSTR("%s","end-device uses native LoRaWAN packet
format\n");
      // indicate to SX1272 lib that raw mode at transmission is
required to avoid our own packet header
      sx1272._rawFormat=true;
#else
      PRINT_CSTSTR("%s","end-device uses encapsulated LoRaWAN packet
format only for encryption\n");
#endif
      // in any case, we increment Frame_Counter_Up
      // even if the transmission will not succeed
      Frame_Counter_Up++;
#endif

#ifdef CUSTOM_CS
      CarrierSense();
#else
      sx1272.CarrierSense();
#endif

      startSend=millis();

#ifdef WITH_AES
      // indicate that payload is encrypted
      // DO NOT take into account appkey
      sx1272.setPacketType(PKT_TYPE_DATA | PKT_FLAG_DATA_ENCRYPTED);
#else
#ifdef WITH_APPKEY
      // indicate that we have an appkey
      sx1272.setPacketType(PKT_TYPE_DATA | PKT_FLAG_DATA_WAPPKEY);
#else
      // just a simple data packet
      sx1272.setPacketType(PKT_TYPE_DATA);
#endif
#endif

#ifdef LORA_LAS

      e = loraLAS.sendData(DEFAULT_DEST_ADDR, (uint8_t*)message, pl, 0,
              LAS_FIRST_DATAPKT+LAS_LAST_DATAPKT, LAS_NOACK);

      if (e==TOA_OVERUSE) {
          PRINT_CSTSTR("%s","Not sent, TOA_OVERUSE\n");
      }

      if (e==LAS_LBT_ERROR) {
          PRINT_CSTSTR("%s","LBT error\n");
      }
```

```
      if (e==LAS_SEND_ERROR || e==LAS_ERROR) {
          PRINT_CSTSTR("%s","Send error\n");
      }
#else
      // Send message to the gateway and print the result
      // with the app key if this feature is enabled
#ifdef WITH_ACK
      int n_retry=NB_RETRIES;

      do {
        e = sx1272.sendPacketTimeoutACK(DEFAULT_DEST_ADDR, message,
pl);

        if (e==3)
          PRINT_CSTSTR("%s","No ACK");

        n_retry--;

        if (n_retry)
          PRINT_CSTSTR("%s","Retry");
        else
          PRINT_CSTSTR("%s","Abort");

      } while (e && n_retry);
#else
      e = sx1272.sendPacketTimeout(DEFAULT_DEST_ADDR, message, pl);
#endif
#endif
      endSend=millis();

#ifdef LORAWAN
      // switch back to normal behavior
      sx1272._rawFormat=false;
#endif

#ifdef WITH_EEPROM
      // save packet number for next packet in case of reboot
      my_sx1272config.seq=sx1272._packetNumber;
      EEPROM.put(0, my_sx1272config);
#endif

      PRINT_CSTSTR("%s","LoRa pkt size ");
      PRINT_VALUE("%d", pl);
      PRINTLN;

      PRINT_CSTSTR("%s","LoRa pkt seq ");
      PRINT_VALUE("%d", sx1272.packet_sent.packnum);
      PRINTLN;

      PRINT_CSTSTR("%s","LoRa Sent in ");
      PRINT_VALUE("%ld", endSend-startSend);
```

```
      PRINTLN;

      PRINT_CSTSTR("%s","LoRa Sent w/CAD in ");
      PRINT_VALUE("%ld", endSend-sx1272._startDoCad);
      PRINTLN;

      PRINT_CSTSTR("%s","Packet sent, state ");
      PRINT_VALUE("%d", e);
      PRINTLN;

#ifdef LOW_POWER
      PRINT_CSTSTR("%s","Switch to power saving mode\n");

      e = sx1272.setSleepMode();

      if (!e)
        PRINT_CSTSTR("%s","Successfully switch LoRa module in sleep
mode\n");
      else
        PRINT_CSTSTR("%s","Could not switch LoRa module in sleep
mode\n");

      FLUSHOUTPUT
      delay(50);

#ifdef __SAMD21G18A__
      // For Arduino M0 or Zero we use the built-in RTC
      //LowPower.standby();
      rtc.setTime(17, 0, 0);
      rtc.setDate(1, 1, 2000);
      rtc.setAlarmTime(17, idlePeriodInMin, 0);
      // for testing with 20s
      //rtc.setAlarmTime(17, 0, 20);
      rtc.enableAlarm(rtc.MATCH_HHMMSS);
      //rtc.attachInterrupt(alarmMatch);
      rtc.standbyMode();

      PRINT_CSTSTR("%s","SAMD21G18A wakes up from standby\n");
      FLUSHOUTPUT
#else
      nCycle = idlePeriodInMin*60/LOW_POWER_PERIOD + random(2,4);

#if defined __MK20DX256__ || defined __MKL26Z64__ || defined
__MK64FX512__ || defined __MK66FX1M0__
      // warning, setTimer accepts value from 1ms to 65535ms max
      timer.setTimer(LOW_POWER_PERIOD*1000 + random(1,5)*1000);//
milliseconds

      nCycle = idlePeriodInMin*60/LOW_POWER_PERIOD;
#endif
```

```
      for (int i=0; i<nCycle; i++) {

#if defined ARDUINO_AVR_PRO || defined ARDUINO_AVR_NANO ||
ARDUINO_AVR_UNO || ARDUINO_AVR_MINI
          // ATmega328P, ATmega168
          LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);

          //LowPower.idle(SLEEP_8S, ADC_OFF, TIMER2_OFF, TIMER1_OFF,
TIMER0_OFF,
          //              SPI_OFF, USART0_OFF, TWI_OFF);
#elif defined ARDUINO_AVR_MEGA2560
          // ATmega2560
          LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);

          //LowPower.idle(SLEEP_8S, ADC_OFF, TIMER5_OFF, TIMER4_OFF,
TIMER3_OFF,
          //      TIMER2_OFF, TIMER1_OFF, TIMER0_OFF, SPI_OFF,
USART3_OFF,
          //      USART2_OFF, USART1_OFF, USART0_OFF, TWI_OFF);
#elif defined __MK20DX256__ || defined __MKL26Z64__ || defined
__MK64FX512__ || defined __MK66FX1M0__
          // Teensy31/32 & TeensyLC
#ifdef LOW_POWER_HIBERNATE
          Snooze.hibernate(sleep_config);
#else
          Snooze.deepSleep(sleep_config);
#endif
#else
          // use the delay function
          delay(LOW_POWER_PERIOD*1000);
#endif
          PRINT_CSTSTR("%s",".");
          FLUSHOUTPUT;
          delay(10);
      }

      delay(50);
#endif

#else
      PRINT_VALUE("%ld", nextTransmissionTime);
      PRINTLN;
      PRINT_CSTSTR("%s","Will send next value at\n");
      // use a random part also to avoid collision
      nextTransmissionTime=millis()+(unsigned
long)idlePeriodInMin*60*1000+(unsigned long)random(15,60)*1000;
      PRINT_VALUE("%ld", nextTransmissionTime);
      PRINTLN;
  }
#endif
```

```
#ifdef LORA_LAS
  // open a receive window
  // only if radio is on for receiving LAS control messages
  if (loraLAS._isRadioOn) {
      e = sx1272.receivePacketTimeout(w_timer);

      if (!e) {
         uint8_t tmp_length;

         if (loraLAS.isLASMsg(sx1272.packet_received.data)) {

           tmp_length=sx1272.packet_received.length-
OFFSET_PAYLOADLENGTH;

           int v=loraLAS.handleLASMsg(sx1272.packet_received.src,
                                      sx1272.packet_received.data,
                                      tmp_length);

           if (v==DSP_DATA) {
              PRINT_CSTSTR("%s","Strange to receive data from LR-
BS\n");
           }
         }
      }
  }
#endif
}
```