```
/* TCC - Medidor de consumo elétrico - PROJETO 8
/*
   Testes de utilização do ESP32 com o sensor PZEM
*/

// --------------------- SENDER --------------------------

// --------------------- Bibliotecas utilizadas --------------------------

#include <SPI.h> //responsável pela comunicação serial
#include <LoRa.h> //responsável pela comunicação com o WIFI Lora
#include <Wire.h>  //responsável pela comunicação i2c
#include <HardwareSerial.h> // Hardware do PZEM
#include <PZEM004T.h> // Biblioteca de comunicação do PZEM
#include <RTClib.h> // Biblioteca de comunidação RTC que funcionou com
Arduino/ ESP
#include <LiquidCrystal_I2C.h> // Biblioteca do LCD


// --------------------- Observações --------------------------
/*
* Deve-se usar o código acima para identificar os endereço I2C
* D1 (05) = SCL
* D2 (04) = SDA
* Endereços scaneados:
* 0x27 = LCD Display
* 0x57 = EEPROM RTC DS3231
* 0x68 = PCF8574
*/
// --------------------- PZEM --------------------------

// Definição do Hardware do PZEM
HardwareSerial PzemSerial2(2);      // Utilizar o hwserial UART2 nos pinos
IO-16 (RX2) e IO-17 (TX2)
PZEM004T pzem(&PzemSerial2);
IPAddress ip(192,168,1,1);

bool pzemrdy = false; // Variável da conexão do PZEM

// Variáveis para armazenar e tratar leituras do PZEM
float v, i, p, e, maior, menor, x, y, z, a = 0;
float v1,v2,v3 = 0;
float c1,c2,c3 = 0;
float p1,p2,p3 = 0;
float e1,e2,e3 = 0;
float e_aux=0;
```

```
float v_offset = 1;
float consumo_mensal, custo_mensal=0;
float cons_aux=0;

// --------------------- LoRA --------------------------

// Definição dos pinos LORA
#define SCK     5    // GPIO5  -- SX127x's SCK
#define MISO    19   // GPIO19 -- SX127x's MISO
#define MOSI    27   // GPIO27 -- SX127x's MOSI
#define SS      18   // GPIO18 -- SX127x's CS
#define RST     14   // GPIO14 -- SX127x's RESET
#define DI00    26   // GPIO26 -- SX127x's IRQ(Interrupt Request)

#define BAND    915E6  //Frequencia do radio - podemos utilizar ainda :
433E6, 868E6, 915E6
#define PABOOST true



// --------------------- LCD --------------------------

LiquidCrystal_I2C lcd(0x27,20,4); // set the LCD address to 0x3F for a 16
chars and 2 line display
// Entradas do LCD
#define SDA1 21
#define SCL1 22


// --------------------- RTC --------------------------
// Variáveis do tempo
uint32_t ts1, ts2, ts3, ts, tfinal, tint=0;
RTC_DS3231 rtc; // Definição da imagem do RTC
char daysOfTheWeek[7][12] = {"Domingo", "Segunda", "Terça", "Quarta",
"Quinta", "Sexta", "Sábado"};
// Entradas do RTC (Real Time Clock)
//#define SDA2 21
//#define SCL2 22

//======================== SETUP
================================================================================
==============

void setup()
{
// Inicialização da comunicação serial
```

```
    Serial.begin(115200);

// ============================ Inicialização do LCD
============================
    Serial.println("Inicialização LCD");
    lcd.begin();

// ============================ Inicialização do LORA
============================
    SPI.begin(SCK,MISO,MOSI,SS); // Inicia a comunicação serial com o Lora
    LoRa.setPins(SS,RST,DI00); // Configura os pinos que serão utlizados
pela biblioteca (deve ser chamado antes do LoRa.begin)

    Serial.print ("Iniciando LoRa ...");
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Iniciando LoRa ...");
    delay(1000);


// Inicializa o Lora com a frequencia específica.
   if (!LoRa.begin(BAND))//PABOOST
   {
      Serial.println("Inicialização do LoRa falhou");
//Acompanhamento no serial
      lcd.setCursor(0,1);
      lcd.print("Inicialização do LoRa falhou");
     while (1);
   }

// Indica no display que iniciou corretamente.
   Serial.println("LoRa Conectado!");            //Acompanhamento no serial
   lcd.setCursor(0,0);
   lcd.print("LoRa Conectado!");
   delay(3000);

// ============================  Inicialização do PZEM
============================
   pzem.setAddress(ip);
   while (!pzemrdy)
   {
      Serial.println("Iniciando o PZEM...");           //Acompanhamento no
serial
      lcd.setCursor(0,2);
      lcd.print("Iniciando o PZEM ...");
      pzemrdy= pzem.setAddress(ip);
```

```
     delay(500); // Aguarda 1/2 seg
   }

    Serial.println("PZEM Conectado!");
    lcd.setCursor(0,3);
    lcd.print("PZEM Conectado!");
    delay(3000);
    lcd.clear();

// ============================ Inicialização do RTC
============================
  while (!rtc.begin()) {
     Serial.println("RTC NÃO encontrado!");
     lcd.setCursor(0,0);
     lcd.print("RTC NAO encontrado..");
  }
     Serial.println("RTC encontrado!");
     lcd.setCursor(0,0);
     lcd.print("RTC encontrado!");
     delay(1000);

     //lcd.setCursor(0,1);
     //lcd.print("Ajuste data-hora");
     //rtc.adjust(DateTime(2019, 11, 02, 13, 24, 0));

  if (rtc.lostPower()) {
     Serial.println("RTC perdeu alimentação, vamos ajustar data-hora!");
     lcd.clear();
     lcd.setCursor(0,1);
     lcd.print("Ajuste data-hora");
      //http://www.esp32learning.com/code/esp32-and-ds3231-rtc-example.php
     // following line sets the RTC to the date &amp;amp; time this sketch was
compiled
     // rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
     // This line sets the RTC with an explicit date & time, for example to
set
     // January 21, 2014 at 3am you would call:
     rtc.adjust(DateTime(2019, 10, 31, 23, 58, 0));
     lcd.setCursor(0,1);
     lcd.print("Manual Adj...");
     delay(1000);
  }
// ============================ Aquisição da data e hora e início da rotina
============================
  DateTime now = rtc.now();
  Serial.print ("Aquisição de data e hora");
```

```
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Energy monitoring");
  lcd.setCursor(0,1);
  lcd.print("Data: ");lcd.print(now.day(), DEC);lcd.print('/');
lcd.print(now.month(), DEC);lcd.print('/');lcd.print(now.year(), DEC);
  lcd.setCursor(0,2);
  lcd.print("Hora: "); lcd.print(now.hour(), DEC);lcd.print(':');
lcd.print(now.minute(), DEC);lcd.print(':');lcd.print(now.second(), DEC);
   Serial.print(now.day(), DEC);Serial.print('/');Serial.print(now.month(),
DEC); Serial.print('/');Serial.print(now.year(), DEC);
  Serial.print ("Início das medições...");
  lcd.setCursor(0,3);
  lcd.print("Medindo....");
  delay(500);
}

//================================== LOOP
==========================================================================
==========
void loop()
{
  DateTime now = rtc.now();
  if(((now.day(), DEC)==1)&& ((now.minute(), DEC)==00) && ((now.hour(),
DEC)==00) && (e > 0) ){ // Reajuste do consumo de energia para todo dia 1
às 00:00
    cons_aux=e;
  }
  else{

// ++++++++++++++++++++++ Loop de aquisição dos dados do PZEM
++++++++++++++++++++

// Aquisição dos dados a serem tratados - 3 resultados para que sejam
filtrados posteriormente

// ------------ Tensão ---------------------------------
  ts1=millis(); // Início da contagem do loop
  v1 = pzem.voltage(ip)+ v_offset; // Aquisição da medição de tensão
  delay(500);
  while (v1 <= 0.0){ // Loop condicional enquanto não for detectado nenhuma
tensão
    delay(500);
    v1 = pzem.voltage(ip) + v_offset; // V_offset = Utilizado porque o PZEM
devolve um valor igual a -1 no resultado de tensão igual a 0
    ts=millis(); // Contagem do final do loop
```

```
    if(ts-ts1 > 10000){ // Caso o loop ultrapassar 10 segundos, será
considerado como falta de energia de fase
      faltadeenergia();
    }
  }
  ts2=millis(); // Repetição do loop para as outras duas medições
  v2 = pzem.voltage(ip)+ v_offset;
  delay(500);
  while (v2 <= 0.0){
    delay(500);
    v2 = pzem.voltage(ip)+ v_offset;
        ts=millis();
    if(ts-ts2 > 10000){
      faltadeenergia();
    }
  }
  ts3=millis();
  v3 = pzem.voltage(ip)+ v_offset;
  delay(500);
  while (v3 <= 0.0){
    delay(500);
    v3 = pzem.voltage(ip)+ v_offset;
    ts=millis();
    if(ts-ts3 > 10000){
      faltadeenergia();
    }
  }

// ------------- Corrente ---------------------------------- // Repetição
da aquisição de dados
  c1 = pzem.current(ip);
    delay(500);
    while (c1 < 0.0){
      delay(500);
      c1 = pzem.current(ip);
    }
  c2 = pzem.current(ip);
    delay(500);
    while (c2 < 0.0){
      delay(500);
      c2 = pzem.current(ip);
    }
  c3 = pzem.current(ip);
    delay(500);
    while (c3 < 0.0){
      delay(500);
```

```
      c3 = pzem.current(ip);
    }

// ------------ Potência ----------------------------------
  p1 = pzem.power(ip);
    delay(500);
    while (p1 < 0.0){
      delay(500);
      p1 = pzem.power(ip);
    }
  p2 = pzem.power(ip);
    delay(500);
    while (p2 < 0.0){
      delay(500);
      p2 = pzem.power(ip);
    }
  p3 = pzem.power(ip);
    delay(500);
    while (p3 < 0.0){
      delay(500);
      p3 = pzem.power(ip);
    }

// ------------ Consumo de energia ------------------------------------
  e1 = pzem.energy(ip);
    delay(500);
    while (e1 < 0.0){
      delay(500);
      e1 = pzem.energy(ip);
    }
  e2 = pzem.energy(ip);
    delay(500);
    while (e2 < 0.0){
      delay(500);
      e2 = pzem.energy(ip);
    }
  e3 = pzem.energy(ip);
    delay(500);
    while (p3 < 0.0){
      delay(500);
      e3 = pzem.energy(ip);
    }

// ------------------------------- Entrada do Loop do tratamento de dados
espúrios  ---------------------------------
  espurgov(v1, v2, v3);
```

```
  espurgoi(c1, c2, c3);
  espurgop(p1, p2, p3);
  espurgoe(e1, e2, e3);
// Retorno dos dados filtrados (v,i,p,e) ---> e convertido para kWh

// ---------------------------------- Reajuste do consumo mensal de energia
----------------------------------
  if (e > e_aux){
    e_aux=e; // Recebe o valor do consumo caso for maior que o valor medido
anteriormente para se manter atualizado
  }
  else {
    e=e_aux; // Caso o valor medido for zero ( representando que não tem
carga ) ou menor que o anterior, atualiza o valor do consumo para o anterio
  }


// ---------------------------------- Cálculo do consumo mensal de energia
----------------------------------

  //  cons_aux=e; // Condição inicial utilizada apenas para definição do
consumo mensal inicial
  // ** DEVE SER RETIRADO APÓS A PRIMEIRA ITERAÇÃO

  if (e < cons_aux){ // Caso a constante de consumo auxiliar for maior que
o consumo atual
    //consumo_mensal=cons_aux;
    Serial.println("Problemas com o consumo mensal");
    Serial.print("Valor da cons_aux: "); Serial.println(cons_aux);
    Serial.print("Valor da consumo mensal: "); Serial.
println(consumo_mensal);
  }
  else{
    consumo_mensal= e - cons_aux;  // Subtração do consumo base do mês com
o consumo decorrido do mês
  }

// Cálculo do custo mensal
  if(consumo_mensal >= 0){
    custo_mensal=0.65823*consumo_mensal; // Multiplicação R$ 0,65823 por
kWh (Dados obtidos em SP: 2019)
  }
  else{
    Serial.print("Problemas com o custo mensal");
  }
```

```
// ---------------------------------------- PZEM Serial Monitor
----------------------------------------
  Serial.print("Tensão(V): "); Serial.println(v);  // Envio de dados para a
comunicação serial
  Serial.print("Corr.(A): "); Serial.println(i); // Envio de dados para a
comunicação serial
  Serial.print("Pot. (W):"); Serial.println(p); // Envio de dados para a
comunicação serial
  Serial.print("Consumo (kWh):"); Serial.println(e); // Envio de dados para
a comunicação serial
  Serial.print("Consumo Mensal (kWh):"); Serial.println(consumo_mensal); //
Envio de dados para a comunicação serial
  Serial.print("Custo Mensal (R$):"); Serial.println(custo_mensal); //
Envio de dados para a comunicação serial

  Serial.println(); // Pula uma tabulação após a finalização dos dados
  delay(3000); // Aguarda 3 milisegundos para a nova iteração

// ---------------------------------------- Configuração do LCD para
apresentar os dados ----------------------------------------
  lcd.clear();
  /*
  lcd.setCursor(0,0);
  lcd.print("Data: ");lcd.print(now.day(), DEC);lcd.print('/');
lcd.print(now.month(), DEC);lcd.print('/');lcd.print(now.year(), DEC);
  lcd.setCursor(0,1);
  lcd.print("Hora: "); lcd.print(now.hour(), DEC);lcd.print(':');
lcd.print(now.minute(), DEC);lcd.print(':');lcd.print(now.second(), DEC);
  */

  lcd.setCursor(0,0);
  lcd.print("Tensao  (V):  ");lcd.print(v);
  lcd.setCursor(0,1);
  lcd.print("Corrente (A): ");lcd.print(i);
  lcd.setCursor(0,2);
  lcd.print("Potencia (W): ");lcd.print(p);
  lcd.setCursor(0,3);
  lcd.print("Consumo(kWh): ");lcd.print(e);

  //lcd.setCursor(0,3);
  //lcd.print("Cons. Mensal: ");lcd.print(consumo_mensal);lcd.print("Wh");
  //lcd.setCursor(0,3);
  //lcd.print("Custo mensal(R$/mês): ");lcd.print(custo_mensal);


// ----------------------------------------  Envio dos dados através do
```

```
LoRa  ----------------------------------------
 //beginPacket : abre um pacote para adicionarmos os dados para envio
 //print: adiciona os dados no pacote
 //endPacket : fecha o pacote e envia -> retorno= 1:sucesso | 0: falha

  LoRa.beginPacket();
  LoRa.print("Tensão (V): ");
  LoRa.print(v);       //Tensão
  LoRa.endPacket();
  Serial.println("Tensao enviada");
  delay(2000);

  LoRa.beginPacket();
  LoRa.print("Corrente (A):  ");
  LoRa.print(i);       //Corrente
  LoRa.endPacket();
  Serial.println("Corrente enviada");
  delay(2000);

  LoRa.beginPacket();
  LoRa.print("Potência (W): ");
  LoRa.print(p);
  LoRa.endPacket();
  Serial.println("Potencia enviada");
  delay(2000);

  LoRa.beginPacket();
  LoRa.print("Consumo (kWh): ");
  LoRa.print(e);       //Watts hora
  LoRa.endPacket();
  Serial.println("Consumo enviado");
  delay(2000);

  LoRa.beginPacket();
  LoRa.print("Consumo mensal (kWh/mês): ");
  LoRa.print(consumo_mensal);       //Watts hora
  LoRa.endPacket();
  Serial.println("Consumo mensal enviado");
  delay(2000);

  LoRa.beginPacket();
  LoRa.print("Custo mensal (R$/mês): ");
  LoRa.print(custo_mensal);       //Watts hora
  LoRa.endPacket();
  Serial.println("Custo mensal enviado");
  delay(2000);
```

```
  LoRa.beginPacket();
  LoRa.print("Data (DD/MM/YYYY): ");
   LoRa.print(now.day(), DEC);LoRa.print('/');LoRa.print(now.month(), DEC);
LoRa.print('/');LoRa.print(now.year(), DEC);     // Data RTC
   Serial.print(now.day(), DEC);Serial.print('/');Serial.print(now.month(),
DEC); Serial.print('/');Serial.println(now.year(), DEC);
  LoRa.endPacket();  // Envio das horas RTC
  Serial.println("Data enviada");
  delay(2000);

  LoRa.beginPacket();
   Serial.print(now.hour(), DEC);Serial.print(':'); Serial.print(now.
minute(), DEC);Serial.print(':');Serial.println(now.second(), DEC);
  LoRa.print("Hora (HH:MM:SS): ");
   LoRa.print(now.hour(), DEC);LoRa.print(':');LoRa.print(now.minute(), DEC)
LoRa.print(':');LoRa.println(now.second(), DEC);
  LoRa.endPacket();  // Envio das horas RTC
  Serial.println("Hora enviada");
  delay(2000);
  Serial.println();
  } // Fechamento do else da potência mensal

 // ---------------------------------- Cálculo do tempo de iteração do
sistema  ----------------------------------
  tfinal=millis(); // Início da contagem do loop
  tint=tfinal-ts1;
  Serial.print("Tempo de iteração do sistema (seg): "); Serial.
println(tint);
}

/*=========================================================================
=======================================
* BLOCO DE FUNÇÕES UTILIZADAS
* Funções de espurdo dados quebrados
*/

/* Tratamento de valores espúrios (a lib do PZEM foi escrita para
arquitetura AVR, não ESP
* Armazena 3 valores em sequência e prepara para comparação/ limpeza de
valores espúrios
*/

// Leitura e Tratamento dos valores de Tensão
void *espurgov(float x, float y, float z){
  float maior = x; float menor = x;
```

```
  if (y > menor){
    menor = y;
  }
  else {
    menor = 0.0;
  }
  if (y > maior && y >= 0.0){
    maior = y;
  }
  if (z > menor){
    menor = z;
  }
  else {
    menor = 0.0;
  }
  if (z > maior && z >= 0.0){
    maior = z;
  }
  v = maior;
}

// Leitura e Tratamento dos valores de Corrente
void *espurgoi(float x, float y, float z){
  float maior = x; float menor = x;
  if (y > menor){
    menor = y;
  }
  else {
    menor = 0.0;
  }
  if (y > maior && y >= 0.0){
    maior = y;
  }
  if (z > menor){
    menor = z;
  }
  else {
    menor = 0.0;
  }
  if (z > maior && z >= 0.0){
    maior = z;
  }

  i = maior;
}
```

```
// Leitura e Tratamento dos valores de Potência Instantânea
void *espurgop(float x, float y, float z){
  float maior = x; float menor = x;
  if (y > menor){
    menor = y;
  }
  else {
    menor = 0.0;
  }
  if (y > maior && y >= 0.0){
    maior = y;
  }
  if (z > menor){
    menor = z;
  }
  else {
    menor = 0.0;
  }
  if (z > maior && z >= 0.0){
    maior = z;
  }
  p = maior;
}

// Leitura e Tratamento dos valores de Energia Consumida
void *espurgoe(float x, float y, float z){
float maior = x; float menor = x;
  if (y > menor){
    menor = y;
  }
  else {
    menor = 0.0;
  }
  if (y > maior && y >= 0.0){
    maior = y;
  }
  if (z > menor){
    menor = z;
  }
  else {
    menor = 0.0;
  }
  if (z > maior && z >= 0.0){
    maior = z;
  }
  //e = maior/1000; // Conversão para kWh
```

```
  e = maior/10;
}

// Tratamento de dados para execução do ciclo de detecção de falta de
energia de fase
void  *faltadeenergia(){
   LoRa.beginPacket();
   LoRa.print("Falta de energia detectada ! ");
   LoRa.endPacket();

   lcd.clear();
   lcd.setCursor(0,0);
   lcd.print("Falta de energia ");
   lcd.setCursor(4,1);
   lcd.print("detectada ! ");

   Serial.println("Falta de energia detectada ! ");
   delay(2000);
   a=0;
   v1 = pzem.voltage(ip)+ v_offset;
   delay(500);
   while (v1 <= 0.0){ // Loop condicional enquanto não retorna a energia ao
sistema
     v1 = pzem.voltage(ip)+ v_offset;
     delay(500);
     a++;
     Serial.println("Tentativa de reconexão:  "); Serial.print(a);
     lcd.setCursor(0,2);
     lcd.print("Reconexao.... ");
     lcd.setCursor(0,3);
     lcd.print(a);
   }
}
```