```
/*********************************************************************************************************************
* This program is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General    *
* Public License as published by the Free Software Foundation, either version 3 of the License, or 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 Lesser General Public License for more  *
* details. You should have received a copy of the GNU Lesser General Public License along with this program.          *
* If not, see <http://www.gnu.org/licenses/> or <http://blog.robotica.eng.br/gnu-lesser-general-public-license/>.      *
*                                                                                                                    *
* © Copyright 2019-2021 Filippo Pardini - filippo@robotica.eng.br - http://blog.robotica.eng.br/                     *
*********************************************************************************************************************/
/*
  Projeto CoMoSisFog V5.0.1 (15/08/2021)
  * Projeto para Controle e Monitoramento de um Sistema Fotovoltaico off grid para iluminação de um jardim

  * O sistema é constituído por 8 painéis “Canadian CS6K-270” de 270Wp cada. Eles estão interligados de forma a
  * estabelecer dois conjuntos iguais de 4 painéis. Cada conjunto tem cada 2 painéis interligados em série e por sua
  * vez interligados em paralelo. Cada conjunto alimenta um controlador “Controlador de Carga MPPT Epsolar Tracer-4210A
  * 40A 12/24V” através de um stringbox de proteção. Os 2 controladores fazem a carga de um banco de baterias
  * constituído de 6 baterias estacionarias “Heliar Freedom DF4100” de 240Ah cada. Estas baterias estão interligadas
  * de forma a obter um banco de 24V (dois conjuntos (cj1 e cj2) em série de três baterias em paralelo cada um,
  * balanceados por equalizador). Os controladores estão interligados ao banco de baterias através do stringbox de
  * proteção de forma invertida entre eles (um controlador alimenta o + do cj1 e o – do cj2 e o outro controlador
  * alimenta o – do cj1 e o + do cj2). Ao banco de baterias está ligado, via um disjuntor DC de 80A, um “Inversor
  * Senoidal Epsolar SHI2000-22 – 2000VA / 24Vcc / 220Vca” que alimenta os refletores LED do jardim (610W) em 220Vca
  * através de fusíveis de proteção. O controle e monitoramento deste sistema é feito através de duas Partes:
  * Parte 2 - um “controlador dos refletores” e Parte 1 - um “monitor dos parâmetros elétricos de carga e descarga do
  * banco de baterias”. O controlador e o monitor estão integrados no projeto CoMoSisFog que usa técnicas de IOT.

  Parte 1 - Quiosque (onde estão os painéis solares)

  ESP32 local - monitor
  * Esta parte utiliza uma placa DOIT ESP32 DEVKIT V1, display Oled, um RTC DS3231, uma eeprom 24C32 - 4K que vem com o
  * RTC DS3231, dois expansores I2C PCF8574, 3 placas de 4 relés com isolação ótica, 1 relé com isolação ótica,
  * uma fotocélula, 3 sensores INA226, um Xbee, um botão NA, um micro SD e um conversor de tensão 12Vdc/5Vdc.
  * Ela faz o monitoramento dos tres INA226 (INA1 - carga do banco de baterias via Tracer1, INA2 - carga do banco de
  * baterias via Tracer2, INA3 - descarga do banco de baterias via inversor) usando o RTC DS3231M, mostrando as
  * leituras atuais de corrente, voltagem, potência e energia armazenada e consumida no display Oled e enviando,
  * via Xbee, para o ESP32 da Parte 2. Grava no SD as voltagens máximas e mínimas das INAs e energia gerada e consumida
  * em 24h (para análise e calibração de ciclos liga/desliga). Em função desses dados, da fotocélula, dos dados
  * configurados pelo ESP32 remoto (Parte 2 via Xbee), controla e disponibiliza energia, em horários pré-configurados,
  * para os refletores, executa comandos e solicitações do ESP32 remoto e gerencia a energia fornecida pelas baterias.

  Parte 2 - Laboratório (na residência)

  ESP32 remoto - controlador
  * Esta parte utiliza uma placa DOIT ESP32 DEVKIT V1, uma eeprom At24c256 - 32KB, um Nanoshield ADC circuitar, um
  * display LCD03 com keypad, um buzzer, dois botões NA, um integrado MC1449 (debouncer), um Xbee, uma fonte 5Vdc
  * com bateria ion-litio 3S e uma micro-ventoinha 12Vdc. Ela atua como entrada de configurações, solicitações e
  * comandos, via keypad, para a parte 1 via Xbee. Faz o display, no LCD03, dos dados enviados pela parte 1 e comanda
  * os grupos de refletores pelo keypad.
*/

//*********************************************************************************************************************
//************************************** Este é o sketch da Parte 1 ***************************************************
//********************************************************************************************************* Bibliotecas

#include <Wire.h>                                       // I2C
#include <INA226_WE.h>                                  // INA226
#include "RTClib.h"                                     // DS3231M
#include <elapsedMillis.h>                              // Intervalos de tempo
#include "SSD1306Wire.h"                                // OLED
#include "FS.h"                                         // file system wrapper
#include "SD.h"                                         // micro SD
#include "SPI.h"                                        // Interface SPI

//_____________________________________________________________________________________________________________ Pinos I2C

// SPI MOSI - 23
// SPI MISO - 19
// SPI SCK  - 18
// SPI SS   - 5
#define SDApin 21                                       // I2C SDA
#define SCLpin 22                                       // I2C SCL
#define fotocelula 25                                   // Pino de entrada fotocélula
#define inputPin 26                                     // Pino para reinicialização da eeprom
#define RXpin 16                                        // UART RX
#define TXpin 17                                        // UART TX
#define i2c_addr_eeprom 0x57                            // Endereco I2C da 24C32 (interna ao RTC DS3231)
#define pcf1 0x20                                       // Endereço I2C PCF8574-1
#define pcf2 0x21                                       // Endereço I2C PCF8574-2
#define oled 0x3C                                       // Endereço I2C OLED
#define endRTC 0x68                                     // Endereço I2C RTC DS3231M
#define endINA1 0x40                                    // Endereço I2C INA1 Tracer1
#define endINA2 0x41                                    // Endereço I2C INA2 Tracer2
#define endINA3 0x44                                    // Endereço I2C INA3 Inversor

//_____________________________________________________________________________________________________ Instâncias
```

```
elapsedMillis timeElapsed1;                         // Instancia elapsedMillis
elapsedMillis timeElapsed2;                         // Instancia elapsedMillis
elapsedMillis timeElapsed;                          // Instancia elapsedMillis
elapsedMillis espera;                               // Instancia elapsedMillis
elapsedMillis timeElapsed24h;                       // Instancia elapsedMillis
INA226_WE ina1(endINA1);                            // Instancia INA226 Tracer1
INA226_WE ina2(endINA2);                            // Instancia INA226 Tracer2
INA226_WE ina3(endINA3);                            // Instancia INA226 Inversor
RTC_DS3231 rtc;                                     // Instancia RTC_DS3231
SSD1306Wire display(oled,SDApin,SCLpin);            // Instância OLED Para ESP32

//_____________________________________________________________________________________________________________ union

union
{
  byte byteArray[908];                              // Vetor unido à estrutura para gravação e leitura da eeprom
  struct                                            // Estrutura de variáveis para processamento
  {
    unsigned long valido;                           // Código que confirma eeprom válida
    float etracer1;                                 // Energia recebida no Tracer 1
    float etracer2;                                 // Energia recebida no Tracer 2
    float einversor;                                // Energia cedida pelo inversor
    unsigned long inter1;                           // Duração do primeiro ciclo em milissegundos
    unsigned long inter2;                           // Duração do segundo ciclo em milissegundos
    int grupo_rf[13][13];                           // grupo_rf[grupo][x] -> refletores que compõe o grupo
                                                    // grupo_rf[grupo][0] -> número de refletores no grupo
    int grupo_am[13][2];                            // grupo_am[grupo][0] -> ativo(0-inativo,1-ativo),
                                                    // grupo_am[grupo][1] -> modo(1-automático(fotocélula),
                                                    // 2-comando,3-programado(horario de liga e desliga))
    float grupo_if[13][2];                          // grupo_if[grupo][0] -> programação horaria - liga
                                                    // grupo_if[grupo][1] -> programação horaria - desliga
  };
}uniao;

//_____________________________________________________________________________________________________ Variáveis e constantes

String versao = "V5.0.1";                             // ************* Versão *********************************

#define timeout 10                                  // Timeout em segundos
bool ad1,ad2,ad3,ad4,ad5,ad6,ad7,ad8;
float delta1,delta2,delta3,coef1,vina1,cina1,pina1,vina2,cina2,pina2,vina3,cina3,pina3;
float etotalin_24h,etotalout_24h,ue1,ue2,ue3,temperatura;
char v_ina1[6],c_ina1[6],p_ina1[6],e_ina1[6];
char v_ina2[6],c_ina2[6],p_ina2[6],e_ina2[6];
char v_ina3[6],c_ina3[6],p_ina3[6],e_ina3[6];
String tempo,VB,AT,PT,WT,registro,str;

int grupo_LD[13];                                   // 1 -> grupo ligado, 0 -> grupo desligado
bool refletor_ligado[13];                           // true -> refletor ligado, false -> refletor desligado
bool tudo_ligado;
unsigned long key = 2863311530;                     // Código de eeprom válida
bool inversor_ligado;
bool pula;
File arquivo;

float vina1_max_dia;                                // Voltagem máxima INA1 no dia
float vina1_min_dia;                                // Voltagem mínima INA1 no dia
float vina2_max_dia;                                // Voltagem máxima INA2 no dia
float vina2_min_dia;                                // Voltagem mínima INA2 no dia
float vina3_max_dia;                                // Voltagem máxima INA3 no dia
float vina3_min_dia;                                // Voltagem mínima INA3 no dia

//_____________________________________________________________________________________________________ Declaração Funções

void telainicial();
bool toggle_grupo_refletores(int grp);
void desliga_todos_refletores(void);
void liga_todos_refletores(void);
void liga_desliga_grupos_automaticos(void);
void liga_desliga_grupos_programados(uint8_t ora,uint8_t mint);
unsigned int dias_desde_01_01_2019(unsigned int dia,unsigned int mes,unsigned int ano);
bool SD_readStringUntil(const char car);
bool tem_algo_ligado(void);
void liga_inversor(void);
void desliga_inversor(void);
bool toggle_refletor(long int rf);
void Serial2_print_flush(String *msg_in);
bool recebe_comando_de_P2(String *cmd);
bool decodifica_comando_recebido(String tmp,int *codcom,int *np,int *param);
void limpa_buffer_serial2(void);
void write_eeprom(unsigned int addr, byte data);
byte read_eeprom(unsigned int addr);
int grava_estrutura_eeprom(void);
int le_estrutura_eeprom(void);
bool aguarda_dados_serial2(int esp,int ponto);

```

```
//________________________________________________________________________________________________________________________
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//__________________________________________________ Processa uma vez ___________________________________________________ setup

void setup()
{
  byte error,addr;
  int nDevices,m;

  //______________________________________________________________________________________________________________________ definição pinos
  pinMode(fotocelula, INPUT_PULLUP);                          // Modo do pino fotocelula
  pinMode(inputPin, INPUT_PULLUP);                            // Modo do inputPin para eeprom default

  //______________________________________________________________________________________________________________________ monitor serial
  Serial.begin(115200);                                       // Define velocidade
  while (!Serial);                                            // aguarda

  //______________________________________________________________________________________________________________________ inicia I2C
  Wire.begin(SDApin,SCLpin);                                  // Define pinos I2C

  //______________________________________________________________________________________________________________________ varredura I2C

  Serial.println("\nVarredura I2C");
  Serial.println("\nProcurando...");
  nDevices = 0;
  ad1=ad2=ad3=ad4=ad5=ad6=ad7=ad8 = false;

  for(addr = 8;addr < 128;addr++)
  {
    Wire.beginTransmission(addr);
    error = Wire.endTransmission();
    if (error == 0)
    {
      switch (addr)
      {
        case 0x57:
          Serial.print("\nEncontrada eeprom 24C32 no endereço 0x57");
          ad1 = true;
          break;
        case 0x20:
          Serial.print("\nEncontrado PCF8574_1 no endereço 0x20");
          ad2 = true;
          break;
        case 0x21:
          Serial.print("\nEncontrado PCF8574_2 no endereço 0x21");
          ad3 = true;
          break;
        case 0x3C:
          Serial.print("\nEncontrado OLED no endereço 0x3C");
          ad4 = true;
          break;
        case 0x68:
          Serial.print("\nEncontrado RTC DS3231M no endereço 0x68");
          ad5 = true;
          break;
        case 0x40:
          Serial.print("\nEncontrado INA1 Tracer1 no endereço 0x40");
          ad6 = true;
          break;
        case 0x41:
          Serial.print("\nEncontrado INA2 Tracer2 no endereço 0x41");
          ad7 = true;
          break;
        case 0x44:
          Serial.print("\nEncontrado INA3 Inversor no endereço 0x44");
          ad8 = true;
          break;
        default:
          Serial.print("\nEndereço desconhecido ");
          Serial.print(addr,HEX);
      }
      nDevices++;
    }
  }
  if (nDevices == 8) Serial.println("\nTodos os equipamentos presentes");
  else
  {
    Serial.println("\nEquipamentos faltando:");
    if (!ad1) Serial.println("\nSem eeprom 24C32");
    if (!ad2) Serial.println("\nSem PCF8574_1");
    if (!ad3) Serial.println("\nSem PCF8574_2");
    if (!ad4) Serial.println("\nSem OLED");
    if (!ad5) Serial.println("\nSem RTC DS3231M");
    if (!ad6) Serial.println("\nSem INA1 Tracer1");
    if (!ad7) Serial.println("\nSem INA2 Tracer2");
```

```
  if (!ad8) Serial.println("\nSem INA3 Inversor");
}

// ____________________________________________________________________________________________________ inicia Xbee

Serial2.begin(9600, SERIAL_8N1, RXpin, TXpin);

//_____________________________________________________________________________________________________ inicia cartão SD
if(!SD.begin())
{
  str = String("<P1-Mount SD falhou>");
  Serial2_print_flush(&str);                          // Envia para P2 para display no LCD03
  while(1);
}
uint8_t cardType = SD.cardType();                     // Tipo do cartão
if(cardType == CARD_NONE){
  str = String("<P1-Sem SD inserido>");
  Serial2_print_flush(&str);                          // Envia para P2 para display no LCD03
  while(1);
}
Serial.print("\nSD inicializado");                     // Tipo do cartão SD
Serial.print("\nSD tipo: ");
if(cardType == CARD_MMC) Serial.print("MMC");
else if(cardType == CARD_SD) Serial.print("SDSC");
else if(cardType == CARD_SDHC) Serial.print("SDHC");
else Serial.print("DESCONHECIDO");

uint64_t cardSize = SD.cardSize() / (1024 * 1024);    // Tamanho do cartão SD
Serial.printf("\nSD tamanho: %lluMB", cardSize);

//*************************************************************************************************************
//SD.remove("/dados.txt");                            // *******Só quando precisa reinicializar /dados.txt********
//*************************************************************************************************************

arquivo = SD.open("/dados.txt", FILE_APPEND);         // Abre o arquivo para append
if(!arquivo)
{
  Serial.print("\nFalha - arquivo dados.txt");
  while(1);                                           // Para
}

//_____________________________________________________________________________________________________ inicia display Oled

display.init();

//_____________________________________________________________________________________________________ inicia rtc

if (!rtc.begin())
{
  Serial.print("\nFalha - RTC");
  while (1);                                          // Para
}

//_____________________________________________________________________________________________________ acertar RTC

//rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));     // Só para acertar o RTC, depois de carregar e executar,
                                                      // comentar (//) e recarregar

//_____________________________________________________________________________________________________ inicializa os refletores

Wire.beginTransmission(pcf1);
Wire.write(B11111111);                                // Os relés ligam com 0, então para desligar 1
Wire.endTransmission();
Wire.beginTransmission(pcf2);
Wire.write(B11111111);                                // Os relés ligam com 0, então para desligar 1
Wire.endTransmission();

for (m=1;m<=12;m++)
{
  refletor_ligado[m] = false;
}
inversor_ligado = false;

//_____________________________________________________________________________________________________ inicializa INAs 226

if (ad6 & ad7 & ad8)                                  // Configura INAs
{
  // Inicia INA1 endereco 0x40 - Tracer 1 - shunt 30A - 75mV
  ina1.init();
  // Configura
  ina1.setAverage(AVERAGE_4);
  ina1.setConversionTime(CONV_TIME_1100);
  ina1.setMeasureMode(TRIGGERED);
  // Calibra INA226 - shunt = 0.0025 ohm, corrente maxima esperada = 30A
  ina1.setResistorRange(0.0025,30.0);
  // ina1.setCorrectionFactor(0.95);  // medido/ina226

```

```
    // Inicia INA2 endereco 0x41 - Tracer 2 - shunt 30A - 75mV
    ina2.init();
    // Configura
    ina2.setAverage(AVERAGE_4);
    ina2.setConversionTime(CONV_TIME_1100);
    ina2.setMeasureMode(TRIGGERED);
    // Calibra INA226 - shunt = 0.0025 ohm, corrente maxima esperada = 30A
    ina2.setResistorRange(0.0025,30.0);
    // ina2.setCorrectionFactor(0.95);  // medido/ina226

    // Inicia INA3 endereco 0x44 - Inversor - shunt 50A - 75mV
    ina3.init();
    // Configura
    ina3.setAverage(AVERAGE_4);
    ina3.setConversionTime(CONV_TIME_1100);
    ina3.setMeasureMode(TRIGGERED);
    // Calibra INA226 - shunt = 0.0015 ohm, corrente maxima esperada = 30A
    ina3.setResistorRange(0.0015,30.0);
    // ina3.setCorrectionFactor(0.95);  // medido/ina226
    ina3.setAlertType(CURRENT_OVER,30000.0);
  }

  //_____________________________________________________________________________________________________________ display OLED

  telainicial();

  //_____________________________________________________________________________________________________________ lê eeprom

  m = le_estrutura_eeprom();
  delay(30);

  // Se eeprom não válida ou reinicialização forçada via inputPin - define eeprom default
  if ((uniao.valido != key) || (digitalRead(inputPin) == LOW))
  {
    uniao.valido = key;                                 // Valida a eeprom
    uniao.etracer1 = 0.0;
    uniao.etracer2 = 0.0;
    uniao.einversor = 0.0;
    uniao.inter1 = 60000;                               // 1 minuto
    uniao.inter2 = 300000;                              // 5 minutos

    for (m=1;m<=12;m++)                                 // Define tipo inicial dos grupos
    {
      uniao.grupo_am[m][0] = 0;                         // Inicialmente grupos inativos
      uniao.grupo_am[m][1] = 0;                         // Inicialmente grupos indefinidos
      uniao.grupo_if[m][0] = 0.0;                       // hora liga grupo, só para grupo programado
      uniao.grupo_if[m][1] = 0.0;                       // hora desliga grupo, só para grupo programado
    }

    m = grava_estrutura_eeprom();                       // Grava eeprom
    delay (30);

    Serial.print("\n\neeprom default");
    Serial.print("\ndeixar o pino inputPin aberto");
    Serial.print("\nBytes gravados na eeprom - ");
    Serial.print(m);
  }
  else  Serial.print("\n\neeprom valida");                  // Eeprom valida

  //_____________________________________________________________________________________________________________ Definição inicial
  //*****************************Atenção !!! rodar uma única vez e retirar*************************************** dos grupos
/*
  uniao.grupo_rf[1][0] = 1;                             // Número de refletores 1
  uniao.grupo_rf[1][1] = 1;                             // Monta grupo 1 com refletor 1
  uniao.grupo_am[1][0] = 1;                             // Inicialmente grupo 1 ativo
  uniao.grupo_am[1][1] = 3;                             // Inicialmente grupo 1 programado
  uniao.grupo_if[1][0] = 18.00;                         // Hora.minuto para ligar
  uniao.grupo_if[1][1] = 6.00;                          // Hora.minuto para desligar

  uniao.grupo_rf[2][0] = 3;                             // Número de refletores 3
  uniao.grupo_rf[2][1] = 2;                             // Monta grupo 2 com refletor 2
  uniao.grupo_rf[2][2] = 3;                             // Monta grupo 2 com refletor 3
  uniao.grupo_rf[2][3] = 4;                             // Monta grupo 2 com refletor 4
  uniao.grupo_am[2][0] = 1;                             // Inicialmente grupo 2 ativo
  uniao.grupo_am[2][1] = 2;                             // Inicialmente grupo 2 comandado

  uniao.grupo_rf[3][0] = 2;                             // Número de refletores 2
  uniao.grupo_rf[3][1] = 5;                             // Monta grupo 3 com refletor 5
  uniao.grupo_rf[3][2] = 6;                             // Monta grupo 3 com refletor 6
  uniao.grupo_am[3][0] = 1;                             // Inicialmente grupo 3 ativo
  uniao.grupo_am[3][1] = 2;                             // Inicialmente grupo 3 comandado

  uniao.grupo_rf[4][0] = 2;                             // Número de refletores 2
  uniao.grupo_rf[4][1] = 7;                             // Monta grupo 4 com refletor 7
  uniao.grupo_rf[4][2] = 8;                             // Monta grupo 4 com refletor 8
  uniao.grupo_am[4][0] = 1;                             // Inicialmente grupo 4 ativo
  uniao.grupo_am[4][1] = 2;                             // Inicialmente grupo 4 comandado
```

```
  uniao.grupo_rf[5][0] = 1;                                 // Número de refletores 1
  uniao.grupo_rf[5][1] = 9;                                 // Monta grupo 5 com refletor 9
  uniao.grupo_am[5][0] = 1;                                 // Inicialmente grupo 5 ativo
  uniao.grupo_am[5][1] = 2;                                 // Inicialmente grupo 5 comandado

  uniao.grupo_rf[6][0] = 1;                                 // Número de refletores 1
  uniao.grupo_rf[6][1] = 10;                                // Monta grupo 6 com refletor 10
  uniao.grupo_am[6][0] = 1;                                 // Inicialmente grupo 6 ativo
  uniao.grupo_am[6][1] = 2;                                 // Inicialmente grupo 6 comandado

  uniao.grupo_rf[7][0] = 2;                                 // Número de refletores 2
  uniao.grupo_rf[7][1] = 3;                                 // Monta grupo 7 com refletor 3
  uniao.grupo_rf[7][2] = 4;                                 // Monta grupo 7 com refletor 4
  uniao.grupo_am[7][0] = 1;                                 // Inicialmente grupo 7 ativo
  uniao.grupo_am[7][1] = 3;                                 // Inicialmente grupo 7 programado
  uniao.grupo_if[7][0] = 18.00;                             // Hora.minuto para ligar
  uniao.grupo_if[7][1] = 6.00;                              // Hora.minuto para desligar

  uniao.grupo_rf[8][0] = 0;                                 // Número de refletores 0
  uniao.grupo_am[8][0] = 0;                                 // Inicialmente grupo 8 inativo
  uniao.grupo_am[8][1] = 0;                                 // Inicialmente grupo indefinido

  uniao.grupo_rf[9][0] = 0;                                 // Número de refletores 0
  uniao.grupo_am[9][0] = 0;                                 // Inicialmente grupo 9 inativo
  uniao.grupo_am[9][1] = 0;                                 // Inicialmente grupo indefinido

  uniao.grupo_rf[10][0] = 0;                                // Número de refletores 0
  uniao.grupo_am[10][0] = 0;                                // Inicialmente grupo 10 inativo
  uniao.grupo_am[10][1] = 0;                                // Inicialmente grupo indefinido

  uniao.grupo_rf[11][0] = 0;                                // Número de refletores 0
  uniao.grupo_am[11][0] = 0;                                // Inicialmente grupo 11 inativo
  uniao.grupo_am[11][1] = 0;                                // Inicialmente grupo indefinido

  uniao.grupo_rf[12][0] = 0;                                // Número de refletores 0
  uniao.grupo_am[12][0] = 0;                                // Inicialmente grupo 12 inativo
  uniao.grupo_am[12][1] = 0;                                // Inicialmente grupo indefinido

  m = grava_estrutura_eeprom();                             // Grava eeprom
  delay (30);

  Serial.print("\n\neeprom gravada com grupos iniciais");
  Serial.print("\nretirar o trecho do programa");
  Serial.print("\nBytes gravados na eeprom - ");
  Serial.print(m);
*/
  //_____________________________________________________________________________________________________________

  Serial.print("\nconfiguracao na eeprom");
  Serial.print("\n\nBytes lidos na eeprom - ");Serial.print(m);
  Serial.print("\nKey = ");Serial.print(uniao.valido);
  Serial.print("\ninter1      " + String(uniao.inter1));
  Serial.print("\ninter2      " + String(uniao.inter2));
  Serial.print("\netracer1    " + String(uniao.etracer1));
  Serial.print("\netracer2    " + String(uniao.etracer2));
  Serial.print("\neinversor   " + String(uniao.einversor));

  //_____________________________________________________________________________________________________ inicializações

  coef1 = (float)uniao.inter1 / 3600000.0;                  // Para calculo da energia (Wh)

  timeElapsed1 = 0;                                         // Inicializa contagem
  timeElapsed2 = 0;                                         // Inicializa contagem
  timeElapsed24h = 0;                                       // Inicializa contagem às 5:00
  vina1_max_dia = 0.0;                                      // Voltagem máxima INA1 no dia
  vina1_min_dia = 30.0;                                     // Voltagem mínima INA1 no dia
  vina2_max_dia = 0.0;                                      // Voltagem máxima INA2 no dia
  vina2_min_dia = 30.0;                                     // Voltagem mínima INA2 no dia
  vina3_max_dia = 0.0;                                      // Voltagem máxima INA3 no dia
  vina3_min_dia = 30.0;                                     // Voltagem mínima INA3 no dia

  ue1 = 0.0;
  ue2 = 0.0;
  ue3 = 0.0;

  pula = false;
  tudo_ligado = false;

  delta1=delta2=delta3=vina1=cina1=pina1=vina2=cina2=pina2=vina3=cina3=pina3= 0.0;
  etotalin_24h=etotalout_24h=0.0;

  for (m=1;m<=12;m++)
  {
    grupo_LD[m] = 0;                                        // Grupos inicialmente desligados
  }

```

```
  Serial.print("\n\nFotocelula - ");
  if ((digitalRead(fotocelula) == LOW)) Serial.print("noite");  // Verifica a fotocelula - LOW => noite; HIGH => dia
  else Serial.print("dia");

  Serial.print("\n\nIniciando...");
  str = String("<P1-Iniciando...>");
  Serial2_print_flush(&str);
}

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//______________________________________________ Processa Continuamente ________________________________________________ loop

void loop()
{
  uint8_t i,ii=0,j,jj,k,hora,minuto,grupo;
  int codigo,numpar,parametros[20];

  DateTime now = rtc.now();                                 // Pega data e hora atual

  // Monta string com data e hora atual
  hora = now.hour();
  minuto = now.minute();
  tempo = String(now.day()) + '/' + String(now.month()) + '/' + String(now.year()) + ' ' + String(hora)
  + ':' + String(minuto) +  ':' + String(now.second());

  //____________________________________________________________________________________________________________________ inicia periodo de 24h

  if ((hora == 5) && !pula)                                 // Inicia periodo de 24h às 05h, executa uma única vez
  {
    // Vamos iniciar a contabilizar quanta energia foi acumulada e quanta foi consumida no periodo de 24h (dia/noite)
    timeElapsed24h = 0;
    pula = true;
  }

  //____________________________________________________________________________________________________________________ Verifica programados

  // Para verificar horario de ligar ou desligar os grupos de refletores programados

  liga_desliga_grupos_programados(hora,minuto);

  //____________________________________________________________________________________________________________________ verifica fotocelula

  // Para verificar se a fotocelula está ativa e quais grupos de refletores devem ser ligados ou desligados

  liga_desliga_grupos_automaticos();

  //____________________________________________________________________________________________________________________ P2 enviou comando?

  // Verifica se P2 enviou algum comando como texto <....>
  // Formato *c[*p1*...pn]# onde c => comando e [*p1*...pn] => parâmetros (opcional)
  // <*1*n> => n=1 => mostra data e hora; n=2 => mostra status da fotocélula; n=3 => mostra energias eeprom;
  //    n=4 => Mostra energias 24h; n=5 => zera energias na eeprom e correntes
  // <*3*dia*mes*ano*hora*minuto> => acerta relogio; ano => 4 digitos; hora => 24h
  // <*4*n> => solicita dados; n=1 => INAS e energias até o momento; n=2 => status grupos
  // <*5*n*v> => ciclos em P1; n=1 => inter1; n=2 => inter2; v => minutos
  // <*6*n*r[*r...*r]> => configuração de grupos; n => grupo; r => refletor; r¹ = 0 => limpa grupo
  // <*7*n*a*b*hl*ml*hd*md> => configuração de grupo; a=0 => inativo; a=1 => ativo; b=1 => fotocélula; b=2 => comando
  //    (P2,celular); b=3 => programado => hl,ml,hd,md (hora e minuto liga, hora e minuto desliga) só existem neste
  //    caso. Este comando pode ser usado de 3 formas: *7*n*a# => só ativa/desativa grupo; *7*n*a*b# => ativa/desativa
  //    e declara tipo; *7*n*a*b*hl*ml*hd*md# => completo
  // <*8*g1[*g2...*gn]> => toggle de grupos
  // <*10> => Liga todos os refletores
  // <*11> => Desliga todos os refletores
  // <*12*n> => Status L/D, n=1 => refletores; n=2 => grupos
  //____________________________________________________________________________________________________________________

  if (recebe_comando_de_P2(&str))
  {
    if (decodifica_comando_recebido(str,&codigo,&numpar,parametros))
    {
      switch (codigo)
      {
        case 1:                                             // Envia data e hora/status fotocélula - <*1*n>
          switch (parametros[1])
          {
            case 1:                                         // Envia data e hora
              str = String('<') + String(tempo) + String('>');
              Serial2_print_flush(&str);
              break;
            case 2:                                         // Envia status fotocélula
              if (digitalRead(fotocelula) == LOW) str = String("<P1-LOW - noite>");
              else str = String("<P1-HIGH - dia>");
              Serial2_print_flush(&str);
              break;
            case 3:                                         // Envia dados energia eeprom
```

```
            le_estrutura_eeprom();                          // Lê eeprom
            delay(30);

            str = String("tracer1 = ") + String(uniao.etracer1/1000.0,2) + String("Kwh");
            Serial.print("\n"); Serial.print(str);
            str = String("<") + str + String(">");
            Serial2_print_flush(&str);
            if (!aguarda_dados_serial2(timeout,9)) break;   //                                                  ponto 9

            str = String("tracer2 = ") + String(uniao.etracer2/1000.0,2) + String("Kwh");
            Serial.print("\n"); Serial.print(str);
            str = String("<") + str + String(">");
            Serial2_print_flush(&str);
            if (!aguarda_dados_serial2(timeout,10)) break;  //                                                  ponto 10

            str = String("inversor = ") + String(uniao.einversor/1000.0,2) + String("Kwh");
            Serial.print("\n"); Serial.print(str);
            str = String("<") + str + String(">");
            Serial2_print_flush(&str);
            if (!aguarda_dados_serial2(timeout,11)) break;  //                                                  ponto 11

            str = String('#');
            Serial2_print_flush(&str);
            break;
          case 4:                                           // Envia energias totais dia
            str = String("ein_24h = ") + String(etotalin_24h/1000.0,2) + String("Kwh");
            Serial.print("\n"); Serial.print(str);
            str = String("<") + str + String(">");
            Serial2_print_flush(&str);
            if (!aguarda_dados_serial2(timeout,4)) break;   //                                                  ponto 4

            str = String("eout_24h = ") + String(etotalout_24h/1000.0,2) + String("Kwh");
            Serial.print("\n"); Serial.print(str);
            str = String("<") + str + String(">");
            Serial2_print_flush(&str);
            if (!aguarda_dados_serial2(timeout,5)) break;   //                                                  ponto 5

            str = String('#');
            Serial2_print_flush(&str);
            break;
          case 5:                                           // Zera energias
            uniao.etracer1 = 0.0;
            uniao.etracer2 = 0.0;
            uniao.einversor = 0.0;
            ue1 = 0.0;
            ue2 = 0.0;
            ue3 = 0.0;

            grava_estrutura_eeprom();                       // Grava eeprom
            delay (30);

            Serial.print("\nEnergias zeradas na eeprom e as atuais");
            str = String("<P1-Energias zeradas>");
            Serial2_print_flush(&str);
            break;
        }
        break;
      case 3:                                               // Acerta relogio - <*3*dia*mes*ano*hora*minuto>
        if (numpar != 5)                                    // Erro no comando
        {
          Serial.print("\nErro no comando 3");
          str = String("<P1-Erro comando 3>");
          Serial2_print_flush(&str);
        }
        else
        {
          rtc.adjust(DateTime(parametros[3],parametros[2],parametros[1],parametros[4],parametros[5],0));
          delay(1000);
          str = String("<P1-Relogio OK>");
          Serial2_print_flush(&str);
        }
        break;
      case 4:                                               // Solicita dados INAS - <*4*1> e grupos - <*4*2>
        if (numpar != 1)                                    // Erro no comando
        {
          Serial.print("\nErro no comando 6");
        }
        else
        {
          switch (parametros[1])
          {
            case 1:                                         // Solicitação de dados INAs
              if (cina1 < 0) cina1 = 0.0;
              if (cina1 == 0) pina1 = 0.0;
              if (cina2 < 0) cina2 = 0.0;
              if (cina2 == 0) pina2 = 0.0;
              if (cina3 < 0) cina3 = 0.0;
```

```
            if (cina3 == 0) pina3 = 0.0;
            str = String('<')+String(vina1,2)+String(',')+String(cina1,2)+String(',')+String(pina1,2)+String('>');
            Serial2_print_flush(&str);
            if (!aguarda_dados_serial2(timeout,1)) break; //                                                    ponto 1
            str = String('<')+String(vina2,2)+String(',')+String(cina2,2)+String(',')+String(pina2,2)+String('>');
            Serial2_print_flush(&str);
            if (!aguarda_dados_serial2(timeout,2)) break; //                                                    ponto 2
            str = String('<')+String(vina3,2)+String(',')+String(cina3,2)+String(',')+String(pina3,2)+String('>');
            Serial2_print_flush(&str);
            if (!aguarda_dados_serial2(timeout,3)) break; //                                                    ponto 3
            str = String('<')+String((ue1 + ue2),2)+String(',')+String(ue3,2)+String('>');
            Serial2_print_flush(&str);
            break;
          case 2:                                       // Solicitação do status dos grupos
            for (j=1;j<=12;j++)                         // Varre grupos
            {
              str = String("<G") + String(j) + String('('); // Prepara linha do grupo
              for (jj=1;jj<=uniao.grupo_rf[j][0];jj++)      // Acrescenta os refletores
              {
                str += String(uniao.grupo_rf[j][jj]);
                if (jj < uniao.grupo_rf[j][0]) str += String(',');
                if (jj == uniao.grupo_rf[j][0])
                {
                  if (uniao.grupo_am[j][0] == 0) str = str + String(") I") + String(uniao.grupo_am[j][1]);
                  else str = str + String(") A") + String(uniao.grupo_am[j][1]);
                  if (uniao.grupo_am[j][1] == 3)
                  {
                    str = str + String(' ') + String(uniao.grupo_if[j][0],2) + String("/") +
                    String(uniao.grupo_if[j][1],2) + String(">");
                  }
                  else str = str + String(">");
                }
              }
              if (jj == 1)
              {
                if (uniao.grupo_am[j][0] == 0) str = str + String(") I") + String(uniao.grupo_am[j][1]);
                else str = str + String(") A") + String(uniao.grupo_am[j][1]);
                if (uniao.grupo_am[j][1] == 3)
                {
                  str = str + String(' ') + String(uniao.grupo_if[j][0],2) + String("/") +
                  String(uniao.grupo_if[j][1],2) + String(">");
                }
                else str = str + String(">");
              }
              Serial2_print_flush(&str);
              if (!aguarda_dados_serial2(timeout,6)) break;   //                                               ponto 6
            }
            str = String('#');
            Serial2_print_flush(&str);
            break;
        }
      }
      break;
    case 5:                                         // Ciclos em P1 - <*5*n*v>
      if (numpar != 2)                              // Erro no comando
      {
        Serial.print("\nErro no comando 5");
        str = String("<P1-Erro comando 5>");
        Serial2_print_flush(&str);
      }
      else
      {
        switch (parametros[1])
        {
          case 1:                                   // É interval1_p1
            uniao.inter1 = parametros[2] * 60000;
            break;
          case 2:                                   // É interval2_p1
            uniao.inter2 = parametros[2] * 60000;
            break;
        }

        // Avisa P2
        str = String("<P1-ciclos gravados>");
        Serial2_print_flush(&str);

        // Avisa no display Oled
        display.clear();
        display.setTextAlignment(TEXT_ALIGN_LEFT);
        display.setFont(ArialMT_Plain_10);
        display.drawString(1, 5, tempo);
        display.drawString(1, 15, "nova configuracao gravada");
        display.drawString(1, 25, "interval1       " + String(uniao.inter1));
        display.drawString(1, 35, "interval2       " + String(uniao.inter2));
        display.display();
        delay(1000);
```

```
            grava_estrutura_eeprom();                       // Grava eeprom
            delay (30);

            // Avisa no monitor serial
            Serial.print("\nnova configuracao gravada na eeprom");
            Serial.print("\ninterval1    " + String(uniao.inter1));
            Serial.print("\ninterval2    " + String(uniao.inter2));
          }
          break;
        case 6:                                             // Configuração de grupos - refletores - <*6*n*r[*r...*r]>
          grupo = parametros[1];
          if (parametros[2] == 0)                           // Limpa grupo
          {
            uniao.grupo_rf[grupo][0] = 0;                   // Sem refletores
            uniao.grupo_am[grupo][0] = 0;                   // Grupo inativo
            uniao.grupo_am[grupo][1] = 0;                   // Modo indefinido
          }
          else
          {
            i = 2;
            for (j=1;j<=numpar;j++)
            {
              uniao.grupo_rf[grupo][j] = parametros[i++];   // Armazena refletor
            }
            uniao.grupo_rf[grupo][0] = numpar - 1;          // Número de refletores no grupo

            str = String("<P1-grupo config OK>");
            Serial2_print_flush(&str);
          }
          grava_estrutura_eeprom();                         // Grava eeprom
          delay (30);

          break;
        case 7:                                             // Configuração de grupo - tipo - <*7*n*a*b*hl*ml*hd*md>
          if ((numpar != 2) && (numpar != 3)  && (numpar != 7))   // Erro no comando
          {
            Serial.print("\nErro no comando 7");
            str = String("<P1-Erro comando 7>");
            Serial2_print_flush(&str);
          }
          else
          {
            grupo = parametros[1];
            switch (numpar)
            {
              case 2:
                uniao.grupo_am[grupo][0] = parametros[2];   // Ativo/inativo
                break;
              case 3:
                uniao.grupo_am[grupo][0] = parametros[2];   // Ativo/inativo
                uniao.grupo_am[grupo][1] = parametros[3];   // Tipo
                break;
              case 7:
                uniao.grupo_am[grupo][0] = parametros[2];   // Ativo/inativo
                uniao.grupo_am[grupo][1] = parametros[3];   // Tipo
                if (uniao.grupo_am[grupo][1] == 3)          // É programado
                {
                  // Liga h.m (h - hora(0-24), m - minutos(0-60))
                  uniao.grupo_if[grupo][0] = (float)parametros[4] + (float)parametros[5]/100.;
                  // Desliga h.m (h - hora(0-24), m - minutos(0-60))
                  uniao.grupo_if[grupo][1] = (float)parametros[6] + (float)parametros[7]/100.;
                  if (uniao.grupo_if[grupo][1] < uniao.grupo_if[grupo][0]) uniao.grupo_if[grupo][1] += 24.0;
                }
                break;
            }
            str = String("<P1-grupo tipo OK>");
            Serial2_print_flush(&str);

            grava_estrutura_eeprom();                       // Grava eeprom
            delay (30);
          }
          break;
        case 8:                                             // Toggle de grupos - <*8*g1[*g2...*gn]>
          for (j=1;j<=numpar;j++)
          {
            k = parametros[j];
            if (!toggle_grupo_refletores(k))
            {
              Serial.print("\nToggle nao executado, grupo inativo ou não comandado - ");
              Serial.print(k);
              str = String("<P1-G") + String(k) + String(" NA>");
              Serial2_print_flush(&str);
            }
            else
            {
              if (grupo_LD[k] == 1) str = String("P1-G") + String(k) + String(" ligado");
              else str = String("P1-G") + String(k) + String(" desligado");
```

```
              Serial.print("\n");
              Serial.print(str);
              str = String("<") + str + String(">");
              Serial2_print_flush(&str);
              if (!aguarda_dados_serial2(timeout,8)) break; //                                                           ponto 8
            }
          }
          Serial2.print('#');                                 // Avisa encerramento a P2
          Serial2.flush();
          break;
        case 10:                                              // Liga todos os refletores - <*10>
          if (numpar != 0)                                    // Erro no comando
          {
            Serial.print("\nErro no comando 10");
            str = String("<P1-Erro comando 10>");
            Serial2_print_flush(&str);
          }
          else
          {
            liga_todos_refletores();
            Serial.print("\nTodos os refletores ligados");
            str = String("<P1-ligou tudo>");
            Serial2_print_flush(&str);
          }
          break;
        case 11:                                              // Desliga todos os refletores - <*11>
          if (numpar != 0)                                    // Erro no comando
          {
            Serial.print("\nErro no comando 11");
            str = String("<P1-Erro comando 11>");
            Serial2_print_flush(&str);
          }
          else
          {
            desliga_todos_refletores();
            Serial.print("\nTodos os refletores desligados");
            str = String("<P1-Desligou tudo>");
            Serial2_print_flush(&str);
          }
          break;
        case 12:                                              // Solicita status dos refletores e grupos
          switch (parametros[1])
          {
            case 1:                                           // Status dos refletores
              str = String("<");
              for (i=1;i<=12;i++)
              {
                if (refletor_ligado[i]) k = 1;
                else k = 0;
                if (i < 12) str = str + String(k) + String(",");
                else str = str + String(k) + String(">");
              }
              Serial2_print_flush(&str);
              break;
            case 2:                                           // Status dos grupos L/D
              str = String("<");
              for (i=1;i<=12;i++)
              {
                if (i < 12) str = str + String(grupo_LD[i]) + String(",");
                else str = str + String(grupo_LD[i]) + String(">");
              }
              Serial2_print_flush(&str);
              break;
          }
          break;
      }
      limpa_buffer_serial2();
    }
    else
    {
      Serial.print("\nFalha parsing");
      str = String("<P1-Falha parsing>");
      Serial2_print_flush(&str);
    }
  }

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

// ****************************************** Grava SD a cada 24H e atualiza eeprom ************************************ grava SD

  if (timeElapsed24h >= 86400000)                             // Executa a cada 24h começando às 05:00
  {
    etotalin_24h = ue1 + ue2;                                 // Energia total acumulada de entrada em 24h
    etotalout_24h = ue3;                                      // Energia total acumulada de saída em 24h
    timeElapsed24h = 0;
```

```
    // Vamos gravar o SD:
    // <datseq,dia,mes,ano,vina1_max_dia,vina1_min_dia,vina2_max_dia,vina2_min_dia,vina3_max_dia,
    // vina3_min_dia,etotalin_24h,etotalout_24h>

    registro = '<'+String(dias_desde_01_01_2019(now.day(),now.month(),now.year()))+','+String(now.day())+
    ','+String(now.month())+','+String(now.year())+','+String(vina1_max_dia)+','+String(vina1_min_dia)+
    ','+String(vina2_max_dia)+','+String(vina2_min_dia)+','+String(vina3_max_dia)+','+String(vina3_min_dia)+
    ','+String(etotalin_24h)+','+String(etotalout_24h)+'>';

    arquivo.close();
    arquivo = SD.open("/dados.txt", FILE_APPEND);
    if(!arquivo)
    {
      Serial.print("\nFalha arquivo 3");
      str = String("<P1-Falha arquivo 3>");
      Serial2_print_flush(&str);
      while(1);                                             // Para
    }
    arquivo.print(registro);
    Serial.print("\n" + registro);

//******************************************* Atualiza energias na eeprom ********************************************** energias na eeprom

    le_estrutura_eeprom();
    delay(30);

    uniao.etracer1 += ue1;
    uniao.etracer2 += ue2;
    uniao.einversor += ue3;

    ue1 = 0.0;
    ue2 = 0.0;
    ue3 = 0.0;

    grava_estrutura_eeprom();                               // Grava eeprom
    delay (30);

    // Reinicializa minimas e máximas
    vina1_max_dia = 0.0;                                    // Voltagem máxima INA1 no dia
    vina1_min_dia = 30.0;                                   // Voltagem mínima INA1 no dia
    vina2_max_dia = 0.0;                                    // Voltagem máxima INA2 no dia
    vina2_min_dia = 30.0;                                   // Voltagem mínima INA2 no dia
    vina3_max_dia = 0.0;                                    // Voltagem máxima INA3 no dia
    vina3_min_dia = 30.0;                                   // Voltagem mínima INA3 no dia

  }

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//_______________________________________ Ciclo de tempo timeElapsed1 - default 1 minuto _______________________________ timeElapsed1

  if (timeElapsed1 >= uniao.inter1)                         // Executa a cada ciclo de duracao inter1, default 1 minuto
  {
    //______________________________________________________________________________________________________________ lê os sensores INA226

    if (ad6 & ad7 & ad8)                                    // Se os sensores INA foram detectados, vamos lê-los
    {
      Serial.print("\nINA1-medição");
      ina1.startSingleMeasurement();
      ina1.readAndClearFlags();
      vina1 = ina1.getBusVoltage_V();                       // Voltagem INA1
      cina1 = ina1.getCurrent_mA() / 1000.0;                // Corrente INA1
      pina1 = ina1.getBusPower() / 1000.0;                  // Potência INA1
      delta1 = pina1 * coef1;                               // Delta energia
      if(ina1.overflow)
      {
        Serial.print("\nINA1-overflow! Use uma corrente maior");
        str = String("<P1-ALERTA-INA1>");
        Serial2_print_flush(&str);
      }

      Serial.print("\nINA2-medição");
      ina2.startSingleMeasurement();
      ina2.readAndClearFlags();
      vina2 = ina2.getBusVoltage_V();                       // Voltagem INA2
      cina2 = ina2.getCurrent_mA() / 1000.0;                // Corrente INA2
      pina2 = ina2.getBusPower() / 1000.0;                  // Potência INA2
      delta2 = pina2 * coef1;                               // Delta energia
      if(ina2.overflow)
      {
        Serial.print("\nINA2-overflow! Use uma corrente maior");
        str = String("<P1-ALERTA-INA2>");
        Serial2_print_flush(&str);
      }

```

```
      Serial.print("\nINA3-medição");
      ina3.startSingleMeasurement();
      ina3.readAndClearFlags();
      vina3 = ina3.getBusVoltage_V();                     // Voltagem INA3
      cina3 = ina3.getCurrent_mA() / 1000.0;              // Corrente INA3
      pina3 = ina3.getBusPower() / 1000.0;                // Potência INA3
      delta3 = pina3 * coef1;                             // Delta energia
      if(ina3.overflow)
      {
        Serial.print("\nINA3-overflow! Use uma corrente maior");
        str = String("<P1-ALERTA-INA3>");
        Serial2_print_flush(&str);
      }
      if (ina3.limitAlert)
      {
        Serial.print("\nALERTA!!! Corrente máxima no inversor - ");
        Serial.print(cina3, 2);
        Serial.print("A");
        str = String("<P1-ALERTA-Inversor>");
        Serial2_print_flush(&str);
      }

      ue1 += delta1;                                      // Integra energia
      ue2 += delta2;                                      // Integra energia
      ue3 += delta3;                                      // Integra energia
    }

    //_____________________________________________________________________________________________________ Imprime data e hora atual

    Serial.println("\n");
    Serial.print(tempo);
    Serial.print("\n");

    //_____________________________________________________________________________________________________ Imprime dados INAs

    // Imprime leituras relativas ao Tracer1
    Serial.print("\nINA226-1 Tracer1");
    Serial.print("\n________________");
    Serial.print("\nBus voltage:   ");
    Serial.print(vina1, 2);
    Serial.print("V");
    Serial.print("\nShunt current: ");
    Serial.print(cina1, 2);
    Serial.print("A");
    Serial.print("\nBus power:     ");
    Serial.print(pina1, 2);
    Serial.print("W");
    Serial.print("\nBus energy:    ");
    Serial.print(delta1, 2);
    Serial.print("Wh");
    // Imprime leituras relativas ao Tracer2
    Serial.print("\n");
    Serial.print("\nINA226-2 Tracer2");
    Serial.print("\n________________");
    Serial.print("\nBus voltage:   ");
    Serial.print(vina2, 2);
    Serial.print("V");
    Serial.print("\nShunt current: ");
    Serial.print(cina2, 2);
    Serial.print("A");
    Serial.print("\nBus power:     ");
    Serial.print(pina2, 2);
    Serial.print("W");
    Serial.print("\nBus energy:    ");
    Serial.print(delta2, 2);
    Serial.print("Wh");
    // Imprime leituras relativas ao Inversor
    Serial.print("\n");
    Serial.print("\nINA226-3 Inversor");
    Serial.print("\n________________");
    Serial.print("\nBus voltage:   ");
    Serial.print(vina3, 2);
    Serial.print("V");
    Serial.print("\nShunt current: ");
    Serial.print(cina3, 2);
    Serial.print("A");
    Serial.print("\nBus power:     ");
    Serial.print(pina3, 2);
    Serial.print("W");
    Serial.print("\nBus energy:    ");
    Serial.print(delta3, 2);
    Serial.print("Wh");
    Serial.print("\n");

    //_____________________________________________________________________________________________________ Display no OLED

    dtostrf(vina1, 5, 2, v_ina1);
```

```
      dtostrf(cina1, 5, 2, c_ina1);
      dtostrf(pina1, 5, 2, p_ina1);
      dtostrf(ue1, 5, 2, e_ina1);

      dtostrf(vina2, 5, 2, v_ina2);
      dtostrf(cina2, 5, 2, c_ina2);
      dtostrf(pina2, 5, 2, p_ina2);
      dtostrf(ue2, 5, 2, e_ina2);

      dtostrf(vina3, 5, 2, v_ina3);
      dtostrf(cina3, 5, 2, c_ina3);
      dtostrf(pina3, 5, 2, p_ina3);
      dtostrf(ue3, 5, 2, e_ina3);

      ii++;
      if (ii > 3) ii = 1;
      switch (ii)
      {
        case 1:
        VB = "V-Banco     " + String(v_ina1);
        AT = "A-Tracer1   " + String(c_ina1);
        PT = "P-Tracer1   " + String(p_ina1);
        WT = "Wh-Tracer1 " + String(e_ina1);
        break;
        case 2:
        VB = "V-Banco     " + String(v_ina2);
        AT = "A-Tracer2   " + String(c_ina2);
        PT = "P-Tracer2   " + String(p_ina2);
        WT = "Wh-Tracer2 " + String(e_ina2);
        break;
        case 3:
        VB = "V-Banco      " + String(v_ina3);
        AT = "A-Inversor   " + String(c_ina3);
        PT = "P-Inversor   " + String(p_ina3);
        WT = "Wh-Inversor " + String(e_ina3);
        break;
      }

      display.clear();
      display.setTextAlignment(TEXT_ALIGN_LEFT);
      display.setFont(ArialMT_Plain_10);
      display.drawString(1, 5, tempo);
      display.drawString(1, 15, VB);
      display.drawString(1, 25, AT);
      display.drawString(1, 35, PT);
      display.drawString(1, 45, WT);
      display.display();

      //__________________________________________________________________________________________________________________ verifica fotocelula

      // Para verificar se a fotocelula está ativa e quais grupos de refletores devem ser ligados ou desligados

      liga_desliga_grupos_automaticos();

      //__________________________________________________________________________________________________________________ Verifica programados

      // Para verificar horario de ligar ou desligar os grupos de refletores programados

      liga_desliga_grupos_programados(hora,minuto);

      //__________________________________________________________________________________________________________________ reset do contador

      timeElapsed1 = 0;
    }

  //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  //___________________________________ Ciclo de tempo timeElapsed2 - default 5 minutos __________________________________ timeElapsed2

    if (timeElapsed2 >= uniao.inter2)                         // Executa quando completa um ciclo de duracao inter2
    {                                                         // Default 5 minutos
      temperatura = rtc.getTemperature();                     // Pega temperatura

      // Vamos pegar máximos e mínimos da voltagem das INAs (no dia)
      if (vina1 > vina1_max_dia) vina1_max_dia = vina1;
      else if (vina1 < vina1_min_dia) vina1_min_dia = vina1;
      else;
      if (vina2 > vina2_max_dia) vina2_max_dia = vina2;
      else if (vina2 < vina2_min_dia) vina2_min_dia = vina2;
      else;
      if (vina3 > vina3_max_dia) vina3_max_dia = vina3;
      else if (vina3 < vina3_min_dia) vina3_min_dia = vina3;
      else;

      // Imprime data e hora atual
      Serial.println("\n");
```

```
    Serial.print(tempo);
    Serial.print("\nTemperatura: ");
    Serial.print(temperatura);
    Serial.print(" Cº\n");
    Serial.print("\nEnergia Tracer1:   ");
    Serial.print(ue1, 2);
    Serial.print("Wh");
    Serial.print("\nEnergia Tracer2:   ");
    Serial.print(ue2, 2);
    Serial.print("Wh");
    Serial.print("\nEnergia Inversor:  ");
    Serial.print(ue3, 2);
    Serial.print("Wh");
    Serial.print("\nEnergia acumulada: ");
    Serial.print((ue1 + ue2), 2);
    Serial.print("Wh");
    Serial.print("\nEnergia consumida: ");
    Serial.print(ue3, 2);
    Serial.print("Wh");

    timeElapsed2 = 0;                                       // Reset do contador
  }
}

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//______________________________________________________ Funções _______________________________________________________ telainicial

void telainicial()                                          // Tela inicial do OLED
{
  str = "CoMoSisFog " + versao;
  display.clear();
  display.setTextAlignment(TEXT_ALIGN_CENTER);
  display.setFont(ArialMT_Plain_10);
  display.drawString(63, 5, str);
  display.drawString(63, 25, "ESP32");
  display.drawString(63, 45, "Filippo Pardini");
  display.display();
}

//______________________________________________________________________________________________________________________ toggle_grupo_refletores

bool toggle_grupo_refletores(int grp)
{
  // Faz o toggle (inversão) de um grupo de refletores: se estava ligado -> desliga e se estava desligado -> liga
  // Nota: Só pode executar para grupos do tipo comandado, automáticos ou programados não

  int m;

  // Vamos verificar se é um grupo ativo e do tipo comandado
  if ((uniao.grupo_am[grp][0] == 0) || (uniao.grupo_am[grp][1] != 2)) return false;   // grupo inativo ou não comandado

  // É ativo, verifica se está ligado ou não
  if (grupo_LD[grp] == 0)                                   // Está desligado, liga
  {
    for (m=1;m<=uniao.grupo_rf[grp][0];m++)                 // Varre os refletores do grupo
    {
      toggle_refletor(uniao.grupo_rf[grp][m]);              // Toggle refletor
    }
    grupo_LD[grp] = 1;                                      // Marca grupo como ligado
  }
  else                                                      // Está ligado, desliga
  {
    for (m=1;m<=uniao.grupo_rf[grp][0];m++)                 // Varre os refletores do grupo
    {
      toggle_refletor(uniao.grupo_rf[grp][m]);              // Toggle refletor
    }
    grupo_LD[grp] = 0;                                      // Marca grupo como desligado
  }
  return true;
}

//______________________________________________________________________________________________________________________ desliga_todos_refletores

void desliga_todos_refletores(void)
{
  uint8_t m;

  Wire.beginTransmission(pcf1);
  Wire.write(B11111111);                                    // Os relés ligam com 0, então para desligar 1
  Wire.endTransmission();
  Wire.beginTransmission(pcf2);
  Wire.write(B11111111);                                    // Os relés ligam com 0, então para desligar 1
  Wire.endTransmission();

  for (m=1;m<=12;m++)                                       // Marca todos os grupos como desligados
```

```
  {
    refletor_ligado[m] = false;                         // Marca refletor  m como desligado
    grupo_LD[m] = 0;                                    // Marca grupo m como desligado
  }

  tudo_ligado = false;
  desliga_inversor();
}

//_____________________________________________________________________________________________________________ liga todos os refletores

void liga_todos_refletores(void)
{
  uint8_t m;

  Wire.beginTransmission(pcf1);
  Wire.write(B00000000);                                // Os relés ligam com 0
  Wire.endTransmission();
  Wire.beginTransmission(pcf2);
  Wire.write(B00000000);                                // Os relés ligam com 0
  Wire.endTransmission();

  for (m=1;m<=12;m++)                                   // Marca todos os grupos como ligados
  {
    refletor_ligado[m] = true;                          // Marca refletor como ligado
    grupo_LD[m] = 1;                                    // Marca grupo como ligado
  }

  tudo_ligado = true;
  liga_inversor();
}

//_____________________________________________________________________________________________________ liga_desliga_grupos_automaticos

void liga_desliga_grupos_automaticos(void)
{
  int adr,ref,rf,n,m,ld;
  byte atual;

  if (!tudo_ligado)
  {
    for (n=1;n<=12;n++)                                 // Varre os grupos
    {
      if ((uniao.grupo_am[n][0] == 1) && (uniao.grupo_am[n][1] == 1)) // Grupo ativo e automático
      {
        if (grupo_LD[n] == 0)                           // Grupo está desligado
        {
          // Tem que ligar?
          if (digitalRead(fotocelula) == LOW) ld = 1; // Sim
          else ld = 2; // Não
        }
        else                                            // Está ligado
        {
          // Tem que desligar?
          if (digitalRead(fotocelula) == HIGH) ld = 0;  // Sim
          else ld = 2; // Não
        }
        switch (ld)
        {
          case 2:                                       // Sem ação, aguarda
            break;
          case 1:                                       // Liga grupo
            for (m=1;m<=uniao.grupo_rf[n][0];m++)       // Varre os refletores do grupo
            {
              rf = uniao.grupo_rf[n][m];                // refletor (entre 1 e 12)
              if (rf <= 8) {adr = pcf1; ref = rf;}      // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
              else {adr = pcf2; ref = rf - 8;}          // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref

              Wire.requestFrom(adr,1);                  // Solicita 1 byte ao Pcf correto
              while (Wire.available() == 0);            // Aguarda estar no buffer
              atual = Wire.read();                      // Lê o byte, bit 0 -> refletor 1 até bit 7 -> refletor
                                                        // 8 se for Pcf1. No Pcf2, bit 0 -> refletor 9 até bit 3
                                                        // -> refletor 12
              atual &= ~(1 << (ref - 1));               // ld = 1 => liga (bit = 0 no pcf)
              refletor_ligado[rf] = true;               // Marca como refletor ligado

              // Atualiza Pcf para acionar refletor do grupo nesse Pcf
              Wire.beginTransmission(adr);              // Inicia transmissão I2C para o Pcf correto
              Wire.write(atual);                        // Novo byte de acionamento dos refletores do grupo
              Wire.endTransmission();                   // Encerra transmissão I2C
            }
            grupo_LD[n] = 1;                            // Marca grupo como ligado
            if (!inversor_ligado) liga_inversor();      // Se inversor desligado, liga inversor
            break;
          case 0:                                       // Desliga grupo
            for (m=1;m<=uniao.grupo_rf[n][0];m++)       // Varre os refletores do grupo
            {
```

```
              rf = uniao.grupo_rf[n][m];                    // refletor (entre 1 e 12)
              if (rf <= 8) {adr = pcf1; ref = rf;}          // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
              else {adr = pcf2; ref = rf - 8;}              // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref

              Wire.requestFrom(adr,1);                      // Solicita 1 byte ao Pcf correto
              while (Wire.available() == 0);                // Aguarda estar no buffer
              atual = Wire.read();                          // Lê o byte, bit 0 -> refletor 1 até bit 7 -> refletor
                                                            // 8 se for Pcf1. No Pcf2, bit 0 -> refletor 9 até bit 3
                                                            // -> refletor 12
              atual |= 1 << (ref - 1);                      // ld = 0 => desliga (bit = 1 no pcf)
              refletor_ligado[rf] = false;                  // Marca como refletor desligado

              // Atualiza Pcf para acionar refletor do grupo nesse Pcf
              Wire.beginTransmission(adr);                  // Inicia transmissão I2C para o Pcf correto
              Wire.write(atual);                            // Novo byte de acionamento dos refletores do grupo
              Wire.endTransmission();                       // Encerra transmissão I2C
            }
            grupo_LD[n] = 0;                                // Marca grupo como desligado
            if (!tem_algo_ligado()) desliga_inversor();     // Se não tem nada ligado, desliga inversor
            break;
          }
        }
      }
    }
  }
}

//_____________________________________________________________________________________________________________ liga_desliga_grupos_programados

void liga_desliga_grupos_programados(uint8_t ora,uint8_t mint)
{
  // Para grupos programados iniciando em um dia e terminando no seguinte, à hora de desligamento (máximo 8.0) deverá
  // ser somado 24.0  Ex: ligar às 22:00 e desligar às 6:00 => ligar - 22.00; desligar - 30.00

  int adr,ref,rf,n,m,ld;
  byte atual;
  float agora;
  ld = 2;

  agora = (float)ora + ((float)mint / 100.0);

  for (n=1;n<=12;n++)                                       // Varre os grupos
  {
    if ((uniao.grupo_am[n][0] == 1) && (uniao.grupo_am[n][1] == 3)) // Grupo ativo e programado
    {
      if (grupo_LD[n] == 0)                                 // Grupo está desligado
      {
        // Está na hora de ligar?
        if ((agora >= uniao.grupo_if[n][0]) && (agora < uniao.grupo_if[n][1])) ld = 1;  // Sim
        else ld = 2;                                        // Não
      }
      else                                                  // Está ligado
      {
        // É madrugada?
        if ((agora <= 8.0)  &&  (uniao.grupo_if[n][1] > 24.0))  // Sim
        {
          if ((agora + 24.0) >= uniao.grupo_if[n][1]) ld = 0;   // Está na hora de desligar
          else ld = 2;                                          // Não
        }
        else // Não é madrugada
        {
          if (agora >= uniao.grupo_if[n][1]) ld = 0;        // Está na hora de desligar
          else ld = 2;                                      //Não
        }
      }
      switch (ld)
      {
        case 2:                                             // Sem ação, aguarda
          break;
        case 1:                                             // Liga grupo
          for (m=1;m<=uniao.grupo_rf[n][0];m++)             // Varre os refletores do grupo
          {
            rf = uniao.grupo_rf[n][m];                      // refletor (entre 1 e 12)
            if (rf <= 8) {adr = pcf1; ref = rf;}            // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
            else {adr = pcf2; ref = rf - 8;}                // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref

            Wire.requestFrom(adr,1);                        // Solicita 1 byte ao Pcf correto
            while (Wire.available() == 0);                  // Aguarda estar no buffer
            atual = Wire.read();                            // Lê o byte, bit 0 -> refletor 1 até bit 7 -> refletor
                                                            // 8 se for Pcf1. No Pcf2, bit 0 -> refletor 9 até bit 3
                                                            // -> refletor 12
            atual &= ~(1 << (ref - 1));                     // ld = 1 => liga (bit = 0 no pcf)
            refletor_ligado[rf] = true;                     // Marca como refletor ligado

            // Atualiza Pcf para acionar refletor do grupo nesse Pcf
            Wire.beginTransmission(adr);                    // Inicia transmissão I2C para o Pcf correto
            Wire.write(atual);                              // Novo byte de acionamento dos refletores do grupo
            Wire.endTransmission();                         // Encerra transmissão I2C
```

```
          }
          grupo_LD[n] = 1;                                  // Marca grupo como ligado
          if (!inversor_ligado) liga_inversor();            // Se inversor desligado, liga inversor
          break;
        case 0:                                             // Desliga grupo
          for (m=1;m<=uniao.grupo_rf[n][0];m++)             // Varre os refletores do grupo
          {
            rf = uniao.grupo_rf[n][m];                      // refletor (entre 1 e 12)
            if (rf <= 8) {adr = pcf1; ref = rf;}            // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
            else {adr = pcf2; ref = rf - 8;}                // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref

            Wire.requestFrom(adr,1);                        // Solicita 1 byte ao Pcf correto
            while (Wire.available() == 0);                  // Aguarda estar no buffer
            atual = Wire.read();                            // Lê o byte, bit 0 -> refletor 1 até bit 7 -> refletor
                                                            // 8 se for Pcf1. No Pcf2, bit 0 -> refletor 9 até bit 3
                                                            // -> refletor 12
            atual |= 1 << (ref - 1);                        // ld = 0 => desliga (bit = 1 no pcf)
            refletor_ligado[rf] = false;                    // Marca como refletor desligado

            // Atualiza Pcf para acionar refletor do grupo nesse Pcf
            Wire.beginTransmission(adr);                    // Inicia transmissão I2C para o Pcf correto
            Wire.write(atual);                              // Novo byte de acionamento dos refletores do grupo
            Wire.endTransmission();                         // Encerra transmissão I2C
          }
          grupo_LD[n] = 0;                                  // Marca grupo como desligado
          if (!tem_algo_ligado()) desliga_inversor();       // Se não tem nada ligado, desliga inversor
          break;
      }
      ld = 2;
    }
  }
}

//_____________________________________________________________________________________________________________ dias_desde_01_01_2019

unsigned int dias_desde_01_01_2019(unsigned int dia,unsigned int mes,unsigned int ano)
{
  unsigned int an,nd = 0;
  unsigned int dd[14] = {0,0,31,59,90,120,151,181,212,243,273,304,334,365};

  for (an=2019;an<ano;an++)
  {
    if ((an % 4) == 0) nd += 366;                               // É ano bissexto (vale até 2100)
    else nd += 365;
  }
  if (((an % 4) == 0) && (mes > 2)) nd += (dd[mes] + dia + 1);  // É ano bissexto (vale até 2100)
  else nd += (dd[mes] + dia);
  return nd;
}

//_____________________________________________________________________________________________________________ SD_readStringUntil

bool SD_readStringUntil(const char car)
{
  // Lê o string de caracteres até o primeiro caractere car encontrado inclusive

  str = "";
  char cc;

  while(1)
  {
    cc = arquivo.read();
    if ((uint8_t)cc == -1) return false;
    if (cc != car) str += cc;
    else
    {
      str += cc;
      return true;
    }
  }
}

//_____________________________________________________________________________________________________________ tem_algo_ligado

bool tem_algo_ligado(void)
{
  uint8_t kk;
  bool ret = false;

  for (kk=1;kk<=12;kk++)
  {
    ret |= refletor_ligado[kk];
  }
  return ret;
}

//_____________________________________________________________________________________________________________ liga_inversor
```

```
1567
1568  void liga_inversor(void)
1569  {
1570    // O inversor tem que estar ligado no pino P7 (bit 7)do pcf2 através de um relé externo
1571    byte este;
1572
1573    Wire.requestFrom(pcf2,1);                                  // Solicita 1 byte ao Pcf2
1574    while (Wire.available() == 0);                             // Aguarda estar no buffer
1575    este = Wire.read();                                        // Lê o byte, bit 7 -> inversor
1576    este &= ~(1 << 7);                                         // Liga (bit 7 = 0)
1577    // Atualiza Pcf2 para ligar inversor
1578    Wire.beginTransmission(pcf2);                              // Inicia transmissão I2C para o Pcf2
1579    Wire.write(este);                                          // Novo byte de acionamento
1580    Wire.endTransmission();                                    // Encerra transmissão I2C
1581    delay(1000);                                               // Tempo para o inversor ligar fisicamente
1582    inversor_ligado = true;
1583  }
1584
1585  //_____________________________________________________________________________________________________________ desliga_inversor
1586
1587  void desliga_inversor(void)
1588  {
1589    // O inversor tem que estar ligado no pino P7 (bit 7)do pcf2 através de um relé externo
1590    byte este;
1591
1592    Wire.requestFrom(pcf2,1);                                  // Solicita 1 byte ao Pcf2
1593    while (Wire.available() == 0);                             // Aguarda estar no buffer
1594    este = Wire.read();                                        // Lê o byte, bit 7 -> inversor
1595    este |= 1 << 7;                                            // Desliga (bit 7 = 1)
1596    // Atualiza Pcf2 para desligar inversor
1597    Wire.beginTransmission(pcf2);                              // Inicia transmissão I2C para o Pcf correto
1598    Wire.write(este);                                          // Novo byte de acionamento
1599    Wire.endTransmission();                                    // Encerra transmissão I2C
1600    delay(1000);                                               // Tempo para o inversor desligar fisicamente
1601    inversor_ligado = false;
1602  }
1603
1604  //_____________________________________________________________________________________________________________ toggle_refletor
1605
1606  bool toggle_refletor(long int rf)
1607  {
1608    // Faz o toggle (inversão) de um refletor: se estava ligado -> desliga e se estava desligado -> liga
1609    // Para até 12 refletores (12 relés)
1610
1611    int adr,ref;
1612    byte atual,tmp;
1613
1614    if (rf <= 8) {adr = pcf1; ref = rf;}                       // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
1615    else if (rf <= 12) {adr = pcf2; ref = rf - 8;}             // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
1616    else return false;
1617
1618    Wire.requestFrom(adr,1);                                   // Solicita 1 byte ao Pcf correto
1619    while (Wire.available() == 0);                             // Aguarda estar no buffer
1620    atual = Wire.read();                                       // Lê o byte, bit 0 -> refletor 1 até bit 7 -> refletor
1621                                                               // 8 se for Pcf1. No Pcf2, bit 0 -> refletor 9 até bit 3
1622                                                               // -> refletor 12
1623
1624    tmp = atual;                                               // Temporario
1625    tmp <<= (8 - ref);                                         // Pega o bit do refletor
1626    tmp >>= 7;                                                 // Pega o bit do refletor
1627    if (tmp == 0)                                              // Se refletor ligado -> desliga
1628    {
1629      atual |= 1 << (ref - 1);                                 // Toggle do refletor rf
1630      refletor_ligado[rf] = false;                             // Marca como refletor desligado
1631      // Atualiza Pcf para acionar refletor nesse Pcf
1632      Wire.beginTransmission(adr);                             // Inicia transmissão I2C para o Pcf correto
1633      Wire.write(atual);                                       // Novo byte de acionamento dos refletores do grupo
1634      Wire.endTransmission();                                  // Encerra transmissão I2C
1635      if (!tem_algo_ligado()) desliga_inversor();              // Se não tem nada ligado, desliga inversor
1636    }
1637    else                                                       // Se refletor desligado -> liga
1638    {
1639      atual &= ~(1 << (ref - 1));                              // Toggle do refletor rf
1640      refletor_ligado[rf] = true;                              // Marca como refletor ligado
1641      // Atualiza Pcf para acionar refletor nesse Pcf
1642      Wire.beginTransmission(adr);                             // Inicia transmissão I2C para o Pcf correto
1643      Wire.write(atual);                                       // Novo byte de acionamento dos refletores do grupo
1644      Wire.endTransmission();                                  // Encerra transmissão I2C
1645      if (!inversor_ligado) liga_inversor();                   // Se inversor desligado, liga inversor
1646    }
1647    return true;
1648  }
1649
1650  //_____________________________________________________________________________________________________________ Serial2_print_flush
1651
1652  void Serial2_print_flush(String *msg_in)
1653  {
```

```
  Serial2.print(*msg_in);
  Serial2.flush();
}

//_____________________________________________________________________________________________________ recebe_comando_de_P2

bool recebe_comando_de_P2(String *cmd)
{
  if (Serial2.available() >= 1)
  {
    if (Serial2.peek() != '<')
    {
      Serial.print("\nComando invalido");
      String(xy) = String("<Comando invalido>");
      Serial2_print_flush(&xy);
      limpa_buffer_serial2();
      return false;
    }
    else
    {
      *cmd = Serial2.readStringUntil('>') + String('>');
      Serial.print("\n");
      Serial.print(*cmd);
      return true;
    }
  }
  else return false;
}

//_____________________________________________________________________________________________________ decodifica_comando_recebido

bool decodifica_comando_recebido(String tmp,int *codcom,int *np,int *param)
{
  // Decodifica um comando recebido no formato : <*c[*p1...*pn]>
  // onde:
  // c => comando (obrigatório); p1...pn => parametros (opcional); * => divisor (primeiro obrigatório,
  // outros se houver parametros); < e > => delimitadores (obrigatório)

  int ni,nf,n;  // ni => indice inicial; nf => indice final; n => número de parametros

  n = 0;

  if ((tmp.indexOf('<') != -1) && (tmp.indexOf('>') != -1) && (tmp.indexOf('*') == 1)) // Garante <* e >
  {
    tmp = tmp.substring(0,tmp.indexOf('>')) + String('>');  // Corta além do >
    ni = tmp.indexOf('*') + 1;                              // Indice inicio codigo de comando
    nf = tmp.indexOf('*',ni);                               // Indice fim codigo de comando
    if (nf == -1)                                           // Se não achou (-1), não tem parametros
    {
      nf = tmp.indexOf('>',ni);                             // Novo indice fim codigo de comando
      *codcom = tmp.substring(ni,nf).toInt();               // Codigo de comando sem parametros
      *np = n;                                              // Numero de parametros => 0
      return true;                                          // Retorna verdadeiro
    }

    // Tem parametros
    *codcom = tmp.substring(ni,nf).toInt();                 // Codigo de comando com parametros
    while(1)                                                // Cicla
    {
      n++;                                                  // Conta parametros
      ni = nf + 1;                                          // Indice inicial parametro
      nf = tmp.indexOf('*',ni);                             // Indice final parametro
      if (nf == -1)                                         // Se -1 é ultimo parametro
      {
        nf = tmp.indexOf('>',ni);                           // Novo indice final parametro
        param[n] = tmp.substring(ni,nf).toInt();            // Parametro n
        *np = n;                                            // número de parametros
        return true;                                        // Retorna verdadeiro
      }
      else param[n] = tmp.substring(ni,nf).toInt();         // Tem mais parametros
    }
  }
  else return false;                                        // Comando invalido, retorna falso
}

//_____________________________________________________________________________________________________ limpa_buffer_serial2

void limpa_buffer_serial2(void)
{
  while (Serial2.available()) Serial2.read();
}

//_____________________________________________________________________________________________________ write_eeprom

void write_eeprom(unsigned int addr, byte data)
{
  Wire.beginTransmission(0x57);                             // Endereço I2C da eeprom
```

```
  Wire.write((int)(addr >> 8));                 // Envia MSB do endereço na eeprom
  Wire.write((int)(addr & 0xFF));               // Envia LSB do endereço na eeprom
  Wire.write(data);                             // Envia byte de dados a ser gravado
  Wire.endTransmission();
  delay(5);                                     // Tempo necessário à gravação na eeprom (min 3,5mseg)
}

//_____________________________________________________________________________________________________ read_eeprom

byte read_eeprom(unsigned int addr)
{
  byte rdata = 0xFF;

  Wire.beginTransmission(0x57);                 // Endereço I2C da eeprom
  Wire.write((int)(addr >> 8));                 // Envia MSB do endereço na eeprom
  Wire.write((int)(addr & 0xFF));               // Envia LSB do endereço na eeprom
  Wire.endTransmission();
  Wire.requestFrom(0x57,1);                     // Solicita o byte
  while (!Wire.available());
  rdata = Wire.read();                          // Se disponivel, lê byte
  delay(5);
  return rdata;                                 // Retorna o byte lido ou 0xFF se não conseguiu ler
}

//_____________________________________________________________________________________________ grava_estrutura_eeprom

int grava_estrutura_eeprom(void)
{
  int n;
  byte *p = uniao.byteArray;

  for (n=0;n<sizeof(uniao.byteArray);n++)
  {
    write_eeprom(n,*p++);
  }
  return n;
}

//_____________________________________________________________________________________________ le_estrutura_eeprom

int le_estrutura_eeprom(void)
{
  int n;
  byte *p = uniao.byteArray;

  for (n=0;n<sizeof(uniao.byteArray);n++)
  {
    *p++ = read_eeprom(n);
  }
  return n;
}

//_____________________________________________________________________________________________ aguarda_dados_serial2

bool aguarda_dados_serial2(int esp,int ponto)
{
  unsigned long tim;
  String to;

  tim = (unsigned long)esp * 1000;
  espera = 0;

  while ((Serial2.available() == 0) && (espera <= tim));    // Aguarda chegar OK('#') de P2
  if (Serial2.available() > 0)
  {
    Serial2.read();                             // Retira o '#'
    return true;
  }
  else
  {
    to = String("Timeout - ponto ") + String(ponto);
    Serial.print("\n"); Serial.print(to);
    return false;
  }
}

//_____________________________________________________________________________________________________
```