

```
1 * This program is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General *
2 * Public License as published by the Free Software Foundation, either version 3 of the License, or any later version. *
3 * This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied *
4 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more *
5 * details. You should have received a copy of the GNU Lesser General Public License along with this program. *
6 * If not, see <http://www.gnu.org/licenses/> or <http://blog.robotica.eng.br/gnu-lesser-general-public-license/>. *
7 *
8 *
9 * © Copyright 2019-2021 Filippo Pardini - filippo@robotica.eng.br - http://blog.robotica.eng.br/ *
10 *****/
11 /*
12 Projeto CoMoSisFog V5.0.1 (15/08/2021)
13 * Projeto para Controle e Monitoramento de um Sistema Fotovoltaico off grid para iluminação de um jardim
14
15 * 0 sistema é constituído por 8 painéis "Canadian CS6K-270" de 270Wp cada. Eles estão interligados de forma a
16 * estabelecer dois conjuntos iguais de 4 painéis. Cada conjunto tem cada 2 painéis interligados em série e por sua
17 * vez interligados em paralelo. Cada conjunto alimenta um controlador "Controlador de Carga MPPT Epsolar Tracer-4210A
18 * 40A 12/24V" através de um stringbox de proteção. Os 2 controladores fazem a carga de um banco de baterias
19 * constituído de 6 baterias estacionárias "Heliar Freedom DF4100" de 240Ah cada. Estas baterias estão interligadas
20 * de forma a obter um banco de 24V (dois conjuntos (cjl e cj2) em série de três baterias em paralelo cada um,
21 * balanceados por equalizador). Os controladores estão interligados ao banco de baterias através do stringbox de
22 * proteção de forma invertida entre eles (um controlador alimenta o + do cjl e o - do cj2 e o outro controlador
23 * alimenta o - do cjl e o + do cj2). Ao banco de baterias está ligado, via um disjuntor DC de 80A, um "Inversor
24 * Senoidal Epsolar SHI2000-22 - 2000VA / 24Vcc / 220Vca" que alimenta os refletores LED do jardim (610W) em 220Vca
25 * através de fusíveis de proteção. O controle e monitoramento deste sistema é feito através de duas Partes:
26 * Parte 2 - um "controlador dos refletores" e Parte 1 - um "monitor dos parâmetros elétricos de carga e descarga do
27 * banco de baterias". O controlador e o monitor estão integrados no projeto CoMoSisFog que usa técnicas de IOT.
28
29
30
31 Parte 1 - Quiosque (onde estão os painéis solares)
32
33 ESP32 local - monitor
34 * Esta parte utiliza uma placa DOIT ESP32 DEVKIT V1, display Oled, um RTC DS3231, uma eeprom 24C32 - 4K que vem com o
35 * RTC DS3231, dois expansores I2C PCF8574, 3 placas de 4 relés com isolação ótica, 1 relé com isolação ótica,
36 * uma fotocélula, 3 sensores INA226, um Xbee, um botão NA, um micro SD e um conversor de tensão 12Vdc/5Vdc.
37 * Ela faz o monitoramento dos tres INA226 (INA1 - carga do banco de baterias via Tracer1, INA2 - carga do banco de
38 * baterias via Tracer2, INA3 - descarga do banco de baterias via inversor) usando o RTC DS3231M, mostrando as
39 * leituras atuais de corrente, voltagem, potência e energia armazenada e consumida no display Oled e enviando,
40 * via Xbee, para o ESP32 da Parte 2. Grava no SD as voltagens máximas e mínimas das INAs e energia gerada e consumida
41 * em 24h (para análise e calibração de ciclos liga/desliga). Em função desses dados, da fotocélula, dos dados
42 * configurados pelo ESP32 remoto (Parte 2 via Xbee), controla e disponibiliza energia, em horários pré-configurados,
43 * para os refletores, executa comandos e solicitações do ESP32 remoto e gerencia a energia fornecida pelas baterias.
44
45
46 Parte 2 - Laboratório (na residência)
47
48 ESP32 remoto - controlador
49 * Esta parte utiliza uma placa DOIT ESP32 DEVKIT V1, uma eeprom At24c256 - 32KB, um Nanoshield ADC circuitar, um
50 * display LCD03 com keypad, um buzzer, dois botões NA, um integrado MC1449 (debouncer), um Xbee, uma fonte 5Vdc
51 * com bateria ion-lítio 3S e uma micro-ventoinha 12Vdc. Ela atua como entrada de configurações, solicitações e
52 * comandos, via keypad, para a parte 1 via Xbee. Faz o display, no LCD03, dos dados enviados pela parte 1 e comanda
53 * os grupos de refletores pelo keypad.
54 */
55
56 //*****
57 //***** Este é o sketch da Parte 1 *****
58 //***** Bibliotecas
59
60 #include <Wire.h> // I2C
61 #include <INA226_WE.h> // INA226
62 #include "RTClib.h" // DS3231M
63 #include <elapsedMillis.h> // Intervalos de tempo
64 #include "SSD1306Wire.h" // OLED
65 #include "FS.h" // file system wrapper
66 #include "SD.h" // micro SD
67 #include "SPI.h" // Interface SPI
68
69 // ----- Pinos I2C
70
71 // SPI MOSI - 23
72 // SPI MISO - 19
73 // SPI SCK - 18
74 // SPI SS - 5
75
76 #define SDApin 21 // I2C SDA
77 #define SCLpin 22 // I2C SCL
78 #define fotocelula 25 // Pino de entrada fotocélula
79 #define inputPin 26 // Pino para reinicialização da eeprom
80 #define RXpin 16 // UART RX
81 #define TXpin 17 // UART TX
82 #define i2c_addr_eeprom 0x57 // Endereço I2C da 24C32 (interna ao RTC DS3231)
83 #define pcf1 0x20 // Endereço I2C PCF8574-1
84 #define pcf2 0x21 // Endereço I2C PCF8574-2
85 #define oLed 0x3C // Endereço I2C OLED
86 #define endRTC 0x68 // Endereço I2C RTC DS3231M
87 #define endINA1 0x40 // Endereço I2C INA1 Tracer1
88 #define endINA2 0x41 // Endereço I2C INA2 Tracer2
89 #define endINA3 0x44 // Endereço I2C INA3 Inversor
90
91 // ----- Instâncias
```

```

89 elapsedMillis timeElapsed1; // Instancia elapsedMillis
90 elapsedMillis timeElapsed2; // Instancia elapsedMillis
91 elapsedMillis timeElapsed; // Instancia elapsedMillis
92 elapsedMillis espera; // Instancia elapsedMillis
93 elapsedMillis timeElapsed24h; // Instancia elapsedMillis
94 INA226_WE ina1(endINA1); // Instancia INA226 Tracer1
95 INA226_WE ina2(endINA2); // Instancia INA226 Tracer2
96 INA226_WE ina3(endINA3); // Instancia INA226 Inversor
97 RTC_DS3231 rtc; // Instancia RTC_DS3231
98 SSD1306Wire display(oled,SDApin,SCLpin); // Instancia OLED Para ESP32
99
100 // ----- union
101
102 union
103 {
104     byte byteArray[908]; // Vetor unido à estrutura para gravação e leitura da eeprom
105     struct // Estrutura de variáveis para processamento
106     {
107         unsigned long valido; // Código que confirma eeprom válida
108         float etracer1; // Energia recebida no Tracer 1
109         float etracer2; // Energia recebida no Tracer 2
110         float einversor; // Energia cedida pelo inversor
111         unsigned long inter1; // Duração do primeiro ciclo em milissegundos
112         unsigned long inter2; // Duração do segundo ciclo em milissegundos
113         int grupo_rf[13][13]; // grupo_rf[grupo][x] -> refletores que compõe o grupo
114         // grupo_rf[grupo][0] -> número de refletores no grupo
115         int grupo_am[13][2]; // grupo_am[grupo][0] -> ativo(0-inativo,1-ativo),
116         // grupo_am[grupo][1] -> modo(1-automático(fotocélula),
117         // 2-comando,3-programado(horario de liga e desliga))
118         float grupo_if[13][2]; // grupo_if[grupo][0] -> programação horaria - liga
119         // grupo_if[grupo][1] -> programação horaria - desliga
120     };
121 }uniao;
122
123 // ----- Variáveis e constantes
124
125 String versao = "V5.0.1"; // ***** Versão *****
126
127 #define timeout 10 // Timeout em segundos
128 bool ad1,ad2,ad3,ad4,ad5,ad6,ad7,ad8;
129 float delta1,delta2,delta3,coef1,vina1,cina1,pina1,vina2,cina2,pina2,vina3,cina3,pina3;
130 float etotalin_24h,etotalout_24h,ue1,ue2,ue3,temperatura;
131 char v_ina1[6],c_ina1[6],p_ina1[6],e_ina1[6];
132 char v_ina2[6],c_ina2[6],p_ina2[6],e_ina2[6];
133 char v_ina3[6],c_ina3[6],p_ina3[6],e_ina3[6];
134 String tempo,VB,AT,PT,WT,registro,str;
135
136 int grupo_LD[13]; // 1 -> grupo ligado, 0 -> grupo desligado
137 bool refletor_ligado[13]; // true -> refletor ligado, false -> refletor desligado
138 bool tudo_ligado;
139 unsigned long key = 2863311530; // Código de eeprom válida
140 bool inversor_ligado;
141 bool pula;
142 File arquivo;
143
144 float vina1_max_dia; // Voltagem máxima INA1 no dia
145 float vina1_min_dia; // Voltagem mínima INA1 no dia
146 float vina2_max_dia; // Voltagem máxima INA2 no dia
147 float vina2_min_dia; // Voltagem mínima INA2 no dia
148 float vina3_max_dia; // Voltagem máxima INA3 no dia
149 float vina3_min_dia; // Voltagem mínima INA3 no dia
150
151 // ----- Declaração Funções
152
153 void telainicial();
154 bool toggle_grupo_refletores(int grp);
155 void desliga_todos_refletores(void);
156 void liga_todos_refletores(void);
157 void liga_desliga_grupos_automáticos(void);
158 void liga_desliga_grupos_programados(uint8_t ora,uint8_t mint);
159 unsigned int dias_desde_01_01_2019(unsigned int dia,unsigned int mes,unsigned int ano);
160 bool SD_readStringUntil(const char car);
161 bool tem_algo_ligado(void);
162 void liga_inversor(void);
163 void desliga_inversor(void);
164 bool toggle_refletor(long int rf);
165 void Serial2_print_flush(String *msg_in);
166 bool recebe_comando_de_P2(String *cmd);
167 bool decodifica_comando_recebido(String tmp,int *codcom,int *np,int *param);
168 void limpa_buffer_serial2(void);
169 void write_eeprom(unsigned int addr, byte data);
170 byte read_eeprom(unsigned int addr);
171 int grava_estrutura_eeprom(void);
172 int le_estrutura_eeprom(void);
173 bool aguarda_dados_serial2(int esp,int ponto);
174

```

```

176 //
177 //
178 //
179 // _____ Processa uma vez _____ setup
180
181 void setup()
182 {
183   byte error,addr;
184   int nDevices,m;
185
186   // _____ definição pinos
187   pinMode(fotocelula, INPUT_PULLUP); // Modo do pino fotocelula
188   pinMode(inputPin, INPUT_PULLUP); // Modo do inputPin para eeprom default
189
190   // _____ monitor serial
191   Serial.begin(115200); // Define velocidade
192   while (!Serial); // aguarda
193
194   // _____ inicia I2C
195   Wire.begin(SDApin,SCLpin); // Define pinos I2C
196
197   // _____ varredura I2C
198
199   Serial.println("\nVarredura I2C");
200   Serial.println("\nProcurando...");
201   nDevices = 0;
202   ad1=ad2=ad3=ad4=ad5=ad6=ad7=ad8 = false;
203
204   for(addr = 8;addr < 128;addr++)
205   {
206     Wire.beginTransmission(addr);
207     error = Wire.endTransmission();
208     if (error == 0)
209     {
210       switch (addr)
211       {
212         case 0x57:
213           Serial.print("\nEncontrada eeprom 24C32 no endereço 0x57");
214           ad1 = true;
215           break;
216         case 0x20:
217           Serial.print("\nEncontrado PCF8574_1 no endereço 0x20");
218           ad2 = true;
219           break;
220         case 0x21:
221           Serial.print("\nEncontrado PCF8574_2 no endereço 0x21");
222           ad3 = true;
223           break;
224         case 0x3C:
225           Serial.print("\nEncontrado OLED no endereço 0x3C");
226           ad4 = true;
227           break;
228         case 0x68:
229           Serial.print("\nEncontrado RTC DS3231M no endereço 0x68");
230           ad5 = true;
231           break;
232         case 0x40:
233           Serial.print("\nEncontrado INA1 Tracer1 no endereço 0x40");
234           ad6 = true;
235           break;
236         case 0x41:
237           Serial.print("\nEncontrado INA2 Tracer2 no endereço 0x41");
238           ad7 = true;
239           break;
240         case 0x44:
241           Serial.print("\nEncontrado INA3 Inversor no endereço 0x44");
242           ad8 = true;
243           break;
244         default:
245           Serial.print("\nEndereço desconhecido ");
246           Serial.print(addr,HEX);
247       }
248       nDevices++;
249     }
250   }
251   if (nDevices == 8) Serial.println("\nTodos os equipamentos presentes");
252   else
253   {
254     Serial.println("\nEquipamentos faltando:");
255     if (!ad1) Serial.println("\nSem eeprom 24C32");
256     if (!ad2) Serial.println("\nSem PCF8574_1");
257     if (!ad3) Serial.println("\nSem PCF8574_2");
258     if (!ad4) Serial.println("\nSem OLED");
259     if (!ad5) Serial.println("\nSem RTC DS3231M");
260     if (!ad6) Serial.println("\nSem INA1 Tracer1");
261     if (!ad7) Serial.println("\nSem INA2 Tracer2");

```

```

263 }
264 }
265 // _____ inicia Xbee
266
267 Serial2.begin(9600, SERIAL_8N1, RXpin, TXpin);
268
269 // _____ inicia cartão SD
270 if(!SD.begin())
271 {
272   str = String("<P1-Mount SD falhou>");
273   Serial2_print_flush(&str); // Envia para P2 para display no LCD03
274   while(1);
275 }
276 uint8_t cardType = SD.cardType(); // Tipo do cartão
277 if(cardType == CARD_NONE){
278   str = String("<P1-Sem SD inserido>");
279   Serial2_print_flush(&str); // Envia para P2 para display no LCD03
280   while(1);
281 }
282 Serial.print("\nSD inicializado"); // Tipo do cartão SD
283 Serial.print("\nSD tipo: ");
284 if(cardType == CARD_MMC) Serial.print("MMC");
285 else if(cardType == CARD_SD) Serial.print("SDSC");
286 else if(cardType == CARD_SDHC) Serial.print("SDHC");
287 else Serial.print("DESCONHECIDO");
288
289 uint64_t cardSize = SD.cardSize() / (1024 * 1024); // Tamanho do cartão SD
290 Serial.printf("\nSD tamanho: %lluMB", cardSize);
291
292 //*****Só quando precisa reinicializar /dados.txt*****
293 //SD.remove("/dados.txt");
294 //*****
295
296 arquivo = SD.open("/dados.txt", FILE_APPEND); // Abre o arquivo para append
297 if(!arquivo)
298 {
299   Serial.print("\nFalha - arquivo dados.txt");
300   while(1); // Para
301 }
302
303 // _____ inicia display Oled
304
305 display.init();
306
307 // _____ inicia rtc
308
309 if (!rtc.begin())
310 {
311   Serial.print("\nFalha - RTC");
312   while (1); // Para
313 }
314
315 // _____ acertar RTC
316
317 //rtc.adjust(DateTime(F(__DATE__), F(__TIME__))); // Só para acertar o RTC, depois de carregar e executar,
318 // comentar (//) e recarregar
319
320 // _____ inicializa os refletores
321
322 Wire.beginTransmission(pcf1);
323 Wire.write(B11111111); // Os relés ligam com 0, então para desligar 1
324 Wire.endTransmission();
325 Wire.beginTransmission(pcf2);
326 Wire.write(B11111111); // Os relés ligam com 0, então para desligar 1
327 Wire.endTransmission();
328
329 for (m=1;m<=12;m++)
330 {
331   refletor_ligado[m] = false;
332 }
333 inversor_ligado = false;
334
335 // _____ inicializa INAs 226
336
337 if (ad6 & ad7 & ad8) // Configura INAs
338 {
339   // Inicia INA1 endereço 0x40 - Tracer 1 - shunt 30A - 75mV
340   ina1.init();
341   // Configura
342   ina1.setAverage(AVERAGE_4);
343   ina1.setConversionTime(CONV_TIME_1100);
344   ina1.setMeasureMode(TRIGGERED);
345   // Calibra INA226 - shunt = 0.0025 ohm, corrente maxima esperada = 30A
346   ina1.setResistorRange(0.0025,30.0);
347   // ina1.setCorrectionFactor(0.95); // medido/ina226
348

```

```

350 // Inicia INA2 endereco 0x41 - Tracer 2 - shunt 30A - 75mV
351 ina2.init();
352 // Configura
353 ina2.setAverage(AVERAGE_4);
354 ina2.setConversionTime(CONV_TIME_1100);
355 ina2.setMeasureMode(TRIGGERED);
356 // Calibra INA226 - shunt = 0.0025 ohm, corrente maxima esperada = 30A
357 ina2.setResistorRange(0.0025,30.0);
358 // ina2.setCorrectionFactor(0.95); // medido/ina226
359
360 // Inicia INA3 endereco 0x44 - Inversor - shunt 50A - 75mV
361 ina3.init();
362 // Configura
363 ina3.setAverage(AVERAGE_4);
364 ina3.setConversionTime(CONV_TIME_1100);
365 ina3.setMeasureMode(TRIGGERED);
366 // Calibra INA226 - shunt = 0.0015 ohm, corrente maxima esperada = 30A
367 ina3.setResistorRange(0.0015,30.0);
368 // ina3.setCorrectionFactor(0.95); // medido/ina226
369 ina3.setAlertType(CURRENT_OVER,30000.0);
370 }
371 // _____ display OLED
372
373 telainicial();
374
375 // _____ lê eeprom
376
377 m = le_estrutura_eeprom();
378 delay(30);
379
380 // Se eeprom não válida ou reinicialização forçada via inputPin - define eeprom default
381 if ((uniao.valido != key) || (digitalRead(inputPin) == LOW))
382 {
383     uniao.valido = key; // Valida a eeprom
384     uniao.etracer1 = 0.0;
385     uniao.etracer2 = 0.0;
386     uniao.einversor = 0.0;
387     uniao.inter1 = 60000; // 1 minuto
388     uniao.inter2 = 300000; // 5 minutos
389
390     for (m=1;m<=12;m++) // Define tipo inicial dos grupos
391     {
392         uniao.grupo_am[m][0] = 0; // Inicialmente grupos inativos
393         uniao.grupo_am[m][1] = 0; // Inicialmente grupos indefinidos
394         uniao.grupo_if[m][0] = 0.0; // hora liga grupo, só para grupo programado
395         uniao.grupo_if[m][1] = 0.0; // hora desliga grupo, só para grupo programado
396     }
397
398     m = grava_estrutura_eeprom(); // Grava eeprom
399     delay (30);
400
401     Serial.print("\n\neeprom default");
402     Serial.print("\ndeixar o pino inputPin aberto");
403     Serial.print("\nBytes gravados na eeprom - ");
404     Serial.print(m);
405 }
406 else Serial.print("\n\neeprom valida"); // Eeprom valida
407
408 // _____ Definição inicial
409 // *****Atenção !!! rodar uma única vez e retirar***** dos grupos
410 /*
411 uniao.grupo_rf[1][0] = 1; // Número de refletores 1
412 uniao.grupo_rf[1][1] = 1; // Monta grupo 1 com refletor 1
413 uniao.grupo_am[1][0] = 1; // Inicialmente grupo 1 ativo
414 uniao.grupo_am[1][1] = 3; // Inicialmente grupo 1 programado
415 uniao.grupo_if[1][0] = 18.00; // Hora.minuto para ligar
416 uniao.grupo_if[1][1] = 6.00; // Hora.minuto para desligar
417
418 uniao.grupo_rf[2][0] = 3; // Número de refletores 3
419 uniao.grupo_rf[2][1] = 2; // Monta grupo 2 com refletor 2
420 uniao.grupo_rf[2][2] = 3; // Monta grupo 2 com refletor 3
421 uniao.grupo_rf[2][3] = 4; // Monta grupo 2 com refletor 4
422 uniao.grupo_am[2][0] = 1; // Inicialmente grupo 2 ativo
423 uniao.grupo_am[2][1] = 2; // Inicialmente grupo 2 comandado
424
425 uniao.grupo_rf[3][0] = 2; // Número de refletores 2
426 uniao.grupo_rf[3][1] = 5; // Monta grupo 3 com refletor 5
427 uniao.grupo_rf[3][2] = 6; // Monta grupo 3 com refletor 6
428 uniao.grupo_am[3][0] = 1; // Inicialmente grupo 3 ativo
429 uniao.grupo_am[3][1] = 2; // Inicialmente grupo 3 comandado
430
431 uniao.grupo_rf[4][0] = 2; // Número de refletores 2
432 uniao.grupo_rf[4][1] = 7; // Monta grupo 4 com refletor 7
433 uniao.grupo_rf[4][2] = 8; // Monta grupo 4 com refletor 8
434 uniao.grupo_am[4][0] = 1; // Inicialmente grupo 4 ativo
435 uniao.grupo_am[4][1] = 2; // Inicialmente grupo 4 comandado

```

```

437 uniao.grupo_rf[5][0] = 1; // Número de refletores 1
438 uniao.grupo_rf[5][1] = 9; // Monta grupo 5 com refletor 9
439 uniao.grupo_am[5][0] = 1; // Inicialmente grupo 5 ativo
440 uniao.grupo_am[5][1] = 2; // Inicialmente grupo 5 comandado
441
442 uniao.grupo_rf[6][0] = 1; // Número de refletores 1
443 uniao.grupo_rf[6][1] = 10; // Monta grupo 6 com refletor 10
444 uniao.grupo_am[6][0] = 1; // Inicialmente grupo 6 ativo
445 uniao.grupo_am[6][1] = 2; // Inicialmente grupo 6 comandado
446
447 uniao.grupo_rf[7][0] = 2; // Número de refletores 2
448 uniao.grupo_rf[7][1] = 3; // Monta grupo 7 com refletor 3
449 uniao.grupo_rf[7][2] = 4; // Monta grupo 7 com refletor 4
450 uniao.grupo_am[7][0] = 1; // Inicialmente grupo 7 ativo
451 uniao.grupo_am[7][1] = 3; // Inicialmente grupo 7 programado
452 uniao.grupo_if[7][0] = 18.00; // Hora.minuto para ligar
453 uniao.grupo_if[7][1] = 6.00; // Hora.minuto para desligar
454
455 uniao.grupo_rf[8][0] = 0; // Número de refletores 0
456 uniao.grupo_am[8][0] = 0; // Inicialmente grupo 8 inativo
457 uniao.grupo_am[8][1] = 0; // Inicialmente grupo indefinido
458
459 uniao.grupo_rf[9][0] = 0; // Número de refletores 0
460 uniao.grupo_am[9][0] = 0; // Inicialmente grupo 9 inativo
461 uniao.grupo_am[9][1] = 0; // Inicialmente grupo indefinido
462
463 uniao.grupo_rf[10][0] = 0; // Número de refletores 0
464 uniao.grupo_am[10][0] = 0; // Inicialmente grupo 10 inativo
465 uniao.grupo_am[10][1] = 0; // Inicialmente grupo indefinido
466
467 uniao.grupo_rf[11][0] = 0; // Número de refletores 0
468 uniao.grupo_am[11][0] = 0; // Inicialmente grupo 11 inativo
469 uniao.grupo_am[11][1] = 0; // Inicialmente grupo indefinido
470
471 uniao.grupo_rf[12][0] = 0; // Número de refletores 0
472 uniao.grupo_am[12][0] = 0; // Inicialmente grupo 12 inativo
473 uniao.grupo_am[12][1] = 0; // Inicialmente grupo indefinido
474
475 m = grava_estrutura_eeprom(); // Grava eeprom
476 delay (30);
477
478 Serial.print("\n\neeprom gravada com grupos iniciais");
479 Serial.print("\n\retirar o trecho do programa");
480 Serial.print("\n\nBytes gravados na eeprom - ");
481 Serial.print(m);
482 */
483 // _____
484
485 Serial.print("\n\nconfiguracao na eeprom");
486 Serial.print("\n\nBytes lidos na eeprom - ");Serial.print(m);
487 Serial.print("\n\nKey = ");Serial.print(uniao.valido);
488 Serial.print("\n\ninter1 " + String(uniao.inter1));
489 Serial.print("\n\ninter2 " + String(uniao.inter2));
490 Serial.print("\n\netracer1 " + String(uniao.etracer1));
491 Serial.print("\n\netracer2 " + String(uniao.etracer2));
492 Serial.print("\n\neinversor " + String(uniao.einversor));
493
494 // _____ inicializações
495
496 coef1 = (float)uniao.inter1 / 3600000.0; // Para calculo da energia (Wh)
497
498 timeElapsed1 = 0; // Inicializa contagem
499 timeElapsed2 = 0; // Inicializa contagem
500 timeElapsed24h = 0; // Inicializa contagem às 5:00
501 vina1_max_dia = 0.0; // Voltagem máxima INA1 no dia
502 vina1_min_dia = 30.0; // Voltagem mínima INA1 no dia
503 vina2_max_dia = 0.0; // Voltagem máxima INA2 no dia
504 vina2_min_dia = 30.0; // Voltagem mínima INA2 no dia
505 vina3_max_dia = 0.0; // Voltagem máxima INA3 no dia
506 vina3_min_dia = 30.0; // Voltagem mínima INA3 no dia
507
508 ue1 = 0.0;
509 ue2 = 0.0;
510 ue3 = 0.0;
511
512 pula = false;
513 tudo_ligado = false;
514
515 delta1=delta2=delta3=vina1=cina1=pina1=vina2=cina2=pina2=vina3=cina3=pina3= 0.0;
516 etotalin_24h=etotalout_24h=0.0;
517
518 for (m=1;m<=12;m++)
519 {
520 grupo_LD[m] = 0; // Grupos inicialmente desligados
521 }
522

```

```

523 Serial.print("\n\nFotocelula - ");
524 if ((digitalRead(fotocelula) == LOW)) Serial.print("noite"); // Verifica a fotocelula - LOW => noite; HIGH => dia
525 else Serial.print("dia");
526
527 Serial.print("\n\nIniciando...");
528 str = String("<P1-Iniciando...>");
529 Serial2_print_flush(&str);
530 }
531
532 //----- Processa Continuamente ----- Loop
533
534 void loop()
535 {
536     uint8_t i,ii=0,j,jj,k,hora,minuto,grupo;
537     int código,numpar,parametros[20];
538
539     DateTime now = rtc.now(); // Pega data e hora atual
540
541     // Monta string com data e hora atual
542     hora = now.hour();
543     minuto = now.minute();
544     tempo = String(now.day()) + '/' + String(now.month()) + '/' + String(now.year()) + ' ' + String(hora)
545     + ':' + String(minuto) + ':' + String(now.second());
546
547     //----- inicia periodo de 24h -----
548     if ((hora == 5) && !pula) // Inicia periodo de 24h às 05h, executa uma única vez
549     {
550         // Vamos iniciar a contabilizar quanta energia foi acumulada e quanta foi consumida no periodo de 24h (dia/noite)
551         timeElapsed24h = 0;
552         pula = true;
553     }
554
555     //----- Verifica programados -----
556     // Para verificar horario de ligar ou desligar os grupos de refletores programados
557     liga_desliga_grupos_programados(hora,minuto);
558
559     //----- verifica fotocelula -----
560     // Para verificar se a fotocelula está ativa e quais grupos de refletores devem ser ligados ou desligados
561     liga_desliga_grupos_automáticos();
562
563     //----- P2 enviou comando? -----
564     // Verifica se P2 enviou algum comando como texto <....>
565     // Formato *c[*p1*...pn]# onde c => comando e [*p1*...pn] => parâmetros (opcional)
566     // <*1*n> => n=1 => mostra data e hora; n=2 => mostra status da fotocélula; n=3 => mostra energias eeprom;
567     // n=4 => Mostra energias 24h; n=5 => zera energias na eeprom e correntes
568     // <*3*dia*mes*ano*hora*minuto> => ajusta relógio; ano => 4 dígitos; hora => 24h
569     // <*4*n> => solicita dados; n=1 => INAS e energias até o momento; n=2 => status grupos
570     // <*5*n*v> => ciclos em P1; n=1 => inter1; n=2 => inter2; v => minutos
571     // <*6*n*r[*r...r]> => configuração de grupos; n => grupo; r => refletor; r1 = 0 => limpa grupo
572     // <*7*n*a*b*hl*ml*hd*md> => configuração de grupo; a=0 => inativo; a=1 => ativo; b=1 => fotocélula; b=2 => comando
573     // (P2,celular); b=3 => programado => hl,ml,hd,md (hora e minuto liga, hora e minuto desliga) só existem neste
574     // caso. Este comando pode ser usado de 3 formas: *7*n*a# => só ativa/desativa grupo; *7*n*a*b# => ativa/desativa
575     // e declara tipo; *7*n*a*b*hl*ml*hd*md# => completo
576     // <*8*g1[*g2...*gn]> => toggle de grupos
577     // <*10> => Liga todos os refletores
578     // <*11> => Desliga todos os refletores
579     // <*12*n> => Status L/D, n=1 => refletores; n=2 => grupos
580
581     if (recebe_comando_de_P2(&str))
582     {
583         if (decodifica_comando_recebido(str,&código,&numpar,parametros))
584         {
585             switch (código)
586             {
587                 case 1: // Envia data e hora/status fotocélula - <*1*n>
588                     switch (parametros[1])
589                     {
590                         case 1: // Envia data e hora
591                             str = String('<') + String(tempo) + String('>');
592                             Serial2_print_flush(&str);
593                             break;
594                         case 2: // Envia status fotocélula
595                             if (digitalRead(fotocelula) == LOW) str = String("<P1-LOW - noite>");
596                             else str = String("<P1-HIGH - dia>");
597                             Serial2_print_flush(&str);
598                             break;
599                         case 3: // Envia dados energia eeprom

```

```

le_estrutura_eeeprom(); // Lê eeprom
611 delay(30);
612
613 str = String("tracer1 = ") + String(uniao.etracer1/1000.0,2) + String("Kwh");
614 Serial.print("\n"); Serial.print(str);
615 str = String("<") + str + String(">");
616 Serial2_print_flush(&str);
617 if (!aguarda_dados_serial2(timeout,9)) break; // ponto 9
618
619 str = String("tracer2 = ") + String(uniao.etracer2/1000.0,2) + String("Kwh");
620 Serial.print("\n"); Serial.print(str);
621 str = String("<") + str + String(">");
622 Serial2_print_flush(&str);
623 if (!aguarda_dados_serial2(timeout,10)) break; // ponto 10
624
625 str = String("inversor = ") + String(uniao.einversor/1000.0,2) + String("Kwh");
626 Serial.print("\n"); Serial.print(str);
627 str = String("<") + str + String(">");
628 Serial2_print_flush(&str);
629 if (!aguarda_dados_serial2(timeout,11)) break; // ponto 11
630
631 str = String('#');
632 Serial2_print_flush(&str);
633 break;
634 case 4: // Envia energias totais dia
635 str = String("ein_24h = ") + String(etotalin_24h/1000.0,2) + String("Kwh");
636 Serial.print("\n"); Serial.print(str);
637 str = String("<") + str + String(">");
638 Serial2_print_flush(&str);
639 if (!aguarda_dados_serial2(timeout,4)) break; // ponto 4
640
641 str = String("eout_24h = ") + String(etotalout_24h/1000.0,2) + String("Kwh");
642 Serial.print("\n"); Serial.print(str);
643 str = String("<") + str + String(">");
644 Serial2_print_flush(&str);
645 if (!aguarda_dados_serial2(timeout,5)) break; // ponto 5
646
647 str = String('#');
648 Serial2_print_flush(&str);
649 break;
650 case 5: // Zera energias
651 uniao.etracer1 = 0.0;
652 uniao.etracer2 = 0.0;
653 uniao.einversor = 0.0;
654 ue1 = 0.0;
655 ue2 = 0.0;
656 ue3 = 0.0;
657
658 grava_estrutura_eeeprom(); // Grava eeprom
659 delay(30);
660
661 Serial.print("\nEnergias zeradas na eeprom e as atuais");
662 str = String("<P1-Energias zeradas>");
663 Serial2_print_flush(&str);
664 break;
665 }
666 break;
667 case 3: // Acerta relógio - <*3*dia*mes*ano*hora*minuto>
668 if (numpar != 5) // Erro no comando
669 {
670 Serial.print("\nErro no comando 3");
671 str = String("<P1-Erro comando 3>");
672 Serial2_print_flush(&str);
673 }
674 else
675 {
676 rtc.adjust(DateTime(parametros[3],parametros[2],parametros[1],parametros[4],parametros[5],0));
677 delay(1000);
678 str = String("<P1-Relógio OK>");
679 Serial2_print_flush(&str);
680 }
681 break;
682 case 4: // Solicita dados INAS - <*4*1> e grupos - <*4*2>
683 if (numpar != 1) // Erro no comando
684 {
685 Serial.print("\nErro no comando 6");
686 }
687 else
688 {
689 switch (parametros[1])
690 {
691 case 1: // Solicitação de dados INAs
692 if (cina1 < 0) cina1 = 0.0;
693 if (cina1 == 0) pina1 = 0.0;
694 if (cina2 < 0) cina2 = 0.0;
695 if (cina2 == 0) pina2 = 0.0;
696 if (cina3 < 0) cina3 = 0.0;

```



```

698 if (cina3 == 0) pina3 = 0.0;
699 str = String('<') + String(vinal,2) + String(',') + String(cina1,2) + String(',') + String(pina1,2) + String('>');
700 Serial2_print_flush(&str);
701 if (!aguarda_dados_serial2(timeout,1)) break; // ponto 1
702 str = String('<') + String(vina2,2) + String(',') + String(cina2,2) + String(',') + String(pina2,2) + String('>');
703 Serial2_print_flush(&str);
704 if (!aguarda_dados_serial2(timeout,2)) break; // ponto 2
705 str = String('<') + String(vina3,2) + String(',') + String(cina3,2) + String(',') + String(pina3,2) + String('>');
706 Serial2_print_flush(&str);
707 if (!aguarda_dados_serial2(timeout,3)) break; // ponto 3
708 str = String('<') + String((ue1 + ue2),2) + String(',') + String(ue3,2) + String('>');
709 Serial2_print_flush(&str);
710 break;
711 case 2: // Solicitação do status dos grupos
712 for (j=1;j<=12;j++) // Varre grupos
713 {
714 str = String("<G") + String(j) + String(',') // Prepara linha do grupo
715 for (jj=1;jj<=uniao.grupo_rf[j][0];jj++) // Acrescenta os refletores
716 {
717 str += String(uniao.grupo_rf[j][jj]);
718 if (jj < uniao.grupo_rf[j][0]) str += String(',')
719 if (jj == uniao.grupo_rf[j][0])
720 {
721 if (uniao.grupo_am[j][0] == 0) str = str + String(" I") + String(uniao.grupo_am[j][1]);
722 else str = str + String(" A") + String(uniao.grupo_am[j][1]);
723 if (uniao.grupo_am[j][1] == 3)
724 {
725 str = str + String(' ') + String(uniao.grupo_if[j][0],2) + String("/") +
726 String(uniao.grupo_if[j][1],2) + String(">");
727 }
728 else str = str + String(">");
729 }
730 }
731 if (jj == 1)
732 {
733 if (uniao.grupo_am[j][0] == 0) str = str + String(" I") + String(uniao.grupo_am[j][1]);
734 else str = str + String(" A") + String(uniao.grupo_am[j][1]);
735 if (uniao.grupo_am[j][1] == 3)
736 {
737 str = str + String(' ') + String(uniao.grupo_if[j][0],2) + String("/") +
738 String(uniao.grupo_if[j][1],2) + String(">");
739 }
740 else str = str + String(">");
741 }
742 Serial2_print_flush(&str);
743 if (!aguarda_dados_serial2(timeout,6)) break; // ponto 6
744 }
745 str = String('#');
746 Serial2_print_flush(&str);
747 break;
748 }
749 break;
750 case 5: // Ciclos em P1 - <*5*n*v>
751 if (numpar != 2) // Erro no comando
752 {
753 Serial.print("\nErro no comando 5");
754 str = String("<P1-Erro comando 5>");
755 Serial2_print_flush(&str);
756 }
757 else
758 {
759 switch (parametros[1])
760 {
761 case 1: // É interval1_p1
762 uniao.inter1 = parametros[2] * 60000;
763 break;
764 case 2: // É interval2_p1
765 uniao.inter2 = parametros[2] * 60000;
766 break;
767 }
768 // Avisa P2
769 str = String("<P1-ciclos gravados>");
770 Serial2_print_flush(&str);
771 // Avisa no display 0led
772 display.clear();
773 display.setTextAlignment(TEXT_ALIGN_LEFT);
774 display.setFont(ArialMT_Plain_10);
775 display.drawString(1, 5, tempo);
776 display.drawString(1, 15, "nova configuracao gravada");
777 display.drawString(1, 25, "interval1 " + String(uniao.inter1));
778 display.drawString(1, 35, "interval2 " + String(uniao.inter2));
779 display.display();
780 delay(1000);
781
782
783

```

```

784 grava_estrutura_eeprom(); // Grava eeprom
785 delay (30);
786
787 // Avisar no monitor serial
788 Serial.print("\nnova configuracao gravada na eeprom");
789 Serial.print("\ninterval1 " + String(uniao.inter1));
790 Serial.print("\ninterval2 " + String(uniao.inter2));
791 }
792 break;
793
794 case 6: // Configuração de grupos - refletores - <*n*r[*...*r]>
795 grupo = parametros[1];
796 if (parametros[2] == 0) // Limpa grupo
797 {
798 uniao.grupo_rf[grupo][0] = 0; // Sem refletores
799 uniao.grupo_am[grupo][0] = 0; // Grupo inativo
800 uniao.grupo_am[grupo][1] = 0; // Modo indefinido
801 }
802 else
803 {
804 i = 2;
805 for (j=1;j<=numpar;j++)
806 {
807 uniao.grupo_rf[grupo][j] = parametros[i++]; // Armazena refletor
808 }
809 uniao.grupo_rf[grupo][0] = numpar - 1; // Número de refletores no grupo
810
811 str = String("<P1-grupo config OK>");
812 Serial2_print_flush(&str);
813 }
814 grava_estrutura_eeprom(); // Grava eeprom
815 delay (30);
816
817 break;
818 case 7: // Configuração de grupo - tipo - <*7*n*a*b*hl*ml*hd*md>
819 if ((numpar != 2) && (numpar != 3) && (numpar != 7)) // Erro no comando
820 {
821 Serial.print("\nErro no comando 7");
822 str = String("<P1-Erro comando 7>");
823 Serial2_print_flush(&str);
824 }
825 else
826 {
827 grupo = parametros[1];
828 switch (numpar)
829 {
830 case 2:
831 uniao.grupo_am[grupo][0] = parametros[2]; // Ativo/inativo
832 break;
833 case 3:
834 uniao.grupo_am[grupo][0] = parametros[2]; // Ativo/inativo
835 uniao.grupo_am[grupo][1] = parametros[3]; // Tipo
836 break;
837 case 7:
838 uniao.grupo_am[grupo][0] = parametros[2]; // Ativo/inativo
839 uniao.grupo_am[grupo][1] = parametros[3]; // Tipo
840 if (uniao.grupo_am[grupo][1] == 3) // É programado
841 {
842 // Liga h.m (h - hora(0-24), m - minutos(0-60))
843 uniao.grupo_if[grupo][0] = (float)parametros[4] + (float)parametros[5]/100.;
844 // Desliga h.m (h - hora(0-24), m - minutos(0-60))
845 uniao.grupo_if[grupo][1] = (float)parametros[6] + (float)parametros[7]/100.;
846 if (uniao.grupo_if[grupo][1] < uniao.grupo_if[grupo][0]) uniao.grupo_if[grupo][1] += 24.0;
847 }
848 break;
849 }
850 str = String("<P1-grupo tipo OK>");
851 Serial2_print_flush(&str);
852
853 grava_estrutura_eeprom(); // Grava eeprom
854 delay (30);
855 }
856 break;
857 case 8: // Toggle de grupos - <*8*g1[*g2...*gn]>
858 for (j=1;j<=numpar;j++)
859 {
860 k = parametros[j];
861 if (!toggle_grupo_refletores(k))
862 {
863 Serial.print("\nToggle nao executado, grupo inativo ou não comandado - ");
864 Serial.print(k);
865 str = String("<P1-G" + String(k) + String(" NA>");
866 Serial2_print_flush(&str);
867 }
868 else
869 {
870 if (grupo_LD[k] == 1) str = String("P1-G" + String(k) + String(" ligado"));
871 else str = String("P1-G" + String(k) + String(" desligado"));

```



```

959 // Vamos gravar o SD:
960 // <dateq,dia,mes,ano,vinal_max_dia,vinal_min_dia,vina2_max_dia,vina2_min_dia,vina3_max_dia,
961 // vina3_min_dia,etotalin_24h,etotalout_24h>
962
963 registro = '<' +String(dias_desde_01_01_2019(now.day()),now.month(),now.year()))+'>'+String(now.day())+
964 '+String(now.month())+'>'+String(now.year())+'>'+String(vinal_max_dia)+'>'+String(vinal_min_dia)+
965 '+String(vina2_max_dia)+'>'+String(vina2_min_dia)+'>'+String(vina3_max_dia)+'>'+String(vina3_min_dia)+
966 '+String(etotalin_24h)+'>'+String(etotalout_24h)+'>';
967
968 arquivo.close();
969 arquivo = SD.open("/dados.txt", FILE_APPEND);
970 if(!arquivo)
971 {
972     Serial.print("\nFalha arquivo 3");
973     str = String("<P1-Falha arquivo 3>");
974     Serial2_print_flush(&str);
975     while(1); // Para
976 }
977 arquivo.print(registro);
978 Serial.print("\n" + registro);
979
980 //***** Atualiza energias na eeprrom ***** energias na eeprrom
981
982 le_estrutura_eeprrom();
983 delay(30);
984
985 uniao.etracer1 += ue1;
986 uniao.etracer2 += ue2;
987 uniao.einversor += ue3;
988
989 ue1 = 0.0;
990 ue2 = 0.0;
991 ue3 = 0.0;
992
993 grava_estrutura_eeprrom(); // Grava eeprrom
994 delay (30);
995
996 // Reinicializa minimas e máximas
997 vinal_max_dia = 0.0; // Voltagem máxima INA1 no dia
998 vinal_min_dia = 30.0; // Voltagem mínima INA1 no dia
999 vina2_max_dia = 0.0; // Voltagem máxima INA2 no dia
1000 vina2_min_dia = 30.0; // Voltagem mínima INA2 no dia
1001 vina3_max_dia = 0.0; // Voltagem máxima INA3 no dia
1002 vina3_min_dia = 30.0; // Voltagem mínima INA3 no dia
1003
1004 }
1005
1006 //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
1007 //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
1008 //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
1009 //_____ Ciclo de tempo timeElapsed1 - default 1 minuto _____ timeElapsed1
1010
1011 if (timeElapsed1 >= uniao.inter1) // Executa a cada ciclo de duracao inter1, default 1 minuto
1012 {
1013 //_____ lê os sensores INA226 _____
1014
1015 if (ad6 & ad7 & ad8) // Se os sensores INA foram detectados, vamos lê-los
1016 {
1017     Serial.print("\nINA1-medicação");
1018     ina1.startSingleMeasurement();
1019     ina1.readAndClearFlags();
1020     vinal = ina1.getBusVoltage_V(); // Voltagem INA1
1021     cinal = ina1.getCurrent_mA() / 1000.0; // Corrente INA1
1022     pinal = ina1.getBusPower() / 1000.0; // Potência INA1
1023     delta1 = pinal * coef1; // Delta energia
1024     if(ina1.overflow)
1025     {
1026         Serial.print("\nINA1-overflow! Use uma corrente maior");
1027         str = String("<P1-ALERTA-INA1>");
1028         Serial2_print_flush(&str);
1029     }
1030
1031     Serial.print("\nINA2-medicação");
1032     ina2.startSingleMeasurement();
1033     ina2.readAndClearFlags();
1034     vina2 = ina2.getBusVoltage_V(); // Voltagem INA2
1035     cina2 = ina2.getCurrent_mA() / 1000.0; // Corrente INA2
1036     pina2 = ina2.getBusPower() / 1000.0; // Potência INA2
1037     delta2 = pina2 * coef1; // Delta energia
1038     if(ina2.overflow)
1039     {
1040         Serial.print("\nINA2-overflow! Use uma corrente maior");
1041         str = String("<P1-ALERTA-INA2>");
1042         Serial2_print_flush(&str);
1043     }
1044 }

```

```

1046 Serial.print("\nINA3-medição");
1047 ina3.startSingleMeasurement();
1048 ina3.readAndClearFlags();
1049 vina3 = ina3.getBusVoltage_V(); // Voltagem INA3
1050 cina3 = ina3.getCurrent_mA() / 1000.0; // Corrente INA3
1051 pina3 = ina3.getBusPower() / 1000.0; // Potência INA3
1052 delta3 = pina3 * coef1; // Delta energia
1053 if(ina3.overflow)
1054 {
1055 Serial.print("\nINA3-overflow! Use uma corrente maior");
1056 str = String("<P1-ALERTA-INA3>");
1057 Serial2_print_flush(&str);
1058 }
1059 if (ina3.limitAlert)
1060 {
1061 Serial.print("\nALERTA!!! Corrente máxima no inversor - ");
1062 Serial.print(cina3, 2);
1063 Serial.print("A");
1064 str = String("<P1-ALERTA-Inversor>");
1065 Serial2_print_flush(&str);
1066 }
1067 ue1 += delta1; // Integra energia
1068 ue2 += delta2; // Integra energia
1069 ue3 += delta3; // Integra energia
1070 }
1071 // _____ Imprime data e hora atual
1072
1073 Serial.println("\n");
1074 Serial.print(tempo);
1075 Serial.print("\n");
1076 // _____ Imprime dados INAs
1077
1078 // Imprime leituras relativas ao Tracer1
1079 Serial.print("\nINA226-1 Tracer1");
1080 Serial.print("\n");
1081 Serial.print("-----");
1082 Serial.print("\nBus voltage: ");
1083 Serial.print(vina1, 2);
1084 Serial.print("V");
1085 Serial.print("\nShunt current: ");
1086 Serial.print(cina1, 2);
1087 Serial.print("A");
1088 Serial.print("\nBus power: ");
1089 Serial.print(pina1, 2);
1090 Serial.print("W");
1091 Serial.print("\nBus energy: ");
1092 Serial.print(delta1, 2);
1093 Serial.print("Wh");
1094 // Imprime leituras relativas ao Tracer2
1095 Serial.print("\n");
1096 Serial.print("\nINA226-2 Tracer2");
1097 Serial.print("\n");
1098 Serial.print("-----");
1099 Serial.print("\nBus voltage: ");
1100 Serial.print(vina2, 2);
1101 Serial.print("V");
1102 Serial.print("\nShunt current: ");
1103 Serial.print(cina2, 2);
1104 Serial.print("A");
1105 Serial.print("\nBus power: ");
1106 Serial.print(pina2, 2);
1107 Serial.print("W");
1108 Serial.print("\nBus energy: ");
1109 Serial.print(delta2, 2);
1110 Serial.print("Wh");
1111 // Imprime leituras relativas ao Inversor
1112 Serial.print("\n");
1113 Serial.print("\nINA226-3 Inversor");
1114 Serial.print("\n");
1115 Serial.print("-----");
1116 Serial.print("\nBus voltage: ");
1117 Serial.print(vina3, 2);
1118 Serial.print("V");
1119 Serial.print("\nShunt current: ");
1120 Serial.print(cina3, 2);
1121 Serial.print("A");
1122 Serial.print("\nBus power: ");
1123 Serial.print(pina3, 2);
1124 Serial.print("W");
1125 Serial.print("\nBus energy: ");
1126 Serial.print(delta3, 2);
1127 Serial.print("Wh");
1128 Serial.print("\n");
1129 // _____ Display no OLED
1130
1131 dtostrf(vina1, 5, 2, v_ina1);

```

```
1132 dtostrf(cina1, 5, 2, c_ina1);
1133 dtostrf(pina1, 5, 2, p_ina1);
1134 dtostrf(ue1, 5, 2, e_ina1);
1135
1136 dtostrf(vina2, 5, 2, v_ina2);
1137 dtostrf(cina2, 5, 2, c_ina2);
1138 dtostrf(pina2, 5, 2, p_ina2);
1139 dtostrf(ue2, 5, 2, e_ina2);
1140
1141 dtostrf(vina3, 5, 2, v_ina3);
1142 dtostrf(cina3, 5, 2, c_ina3);
1143 dtostrf(pina3, 5, 2, p_ina3);
1144 dtostrf(ue3, 5, 2, e_ina3);
1145
1146 ii++;
1147 if (ii > 3) ii = 1;
1148 switch (ii)
1149 {
1150     case 1:
1151         VB = "V-Banco " + String(v_ina1);
1152         AT = "A-Tracer1 " + String(c_ina1);
1153         PT = "P-Tracer1 " + String(p_ina1);
1154         WT = "Wh-Tracer1 " + String(e_ina1);
1155         break;
1156     case 2:
1157         VB = "V-Banco " + String(v_ina2);
1158         AT = "A-Tracer2 " + String(c_ina2);
1159         PT = "P-Tracer2 " + String(p_ina2);
1160         WT = "Wh-Tracer2 " + String(e_ina2);
1161         break;
1162     case 3:
1163         VB = "V-Banco " + String(v_ina3);
1164         AT = "A-Inversor " + String(c_ina3);
1165         PT = "P-Inversor " + String(p_ina3);
1166         WT = "Wh-Inversor " + String(e_ina3);
1167         break;
1168 }
1169
1170 display.clear();
1171 display.setTextAlignment(TEXT_ALIGN_LEFT);
1172 display.setFont(ArialMT_Plain_10);
1173 display.drawString(1, 5, tempo);
1174 display.drawString(1, 15, VB);
1175 display.drawString(1, 25, AT);
1176 display.drawString(1, 35, PT);
1177 display.drawString(1, 45, WT);
1178 display.display();
1179
1180 // _____ verifica fotocelula
1181
1182 // Para verificar se a fotocelula está ativa e quais grupos de refletores devem ser ligados ou desligados
1183
1184 liga_desliga_grupos_automaticos();
1185
1186 // _____ Verifica programados
1187
1188 // Para verificar horario de ligar ou desligar os grupos de refletores programados
1189
1190 liga_desliga_grupos_programados(hora,minuto);
1191
1192 // _____ reset do contador
1193
1194 timeElapsed1 = 0;
1195 }
1196
1197 //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
1198 //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
1199 //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
1200 // _____ Ciclo de tempo timeElapsed2 - default 5 minutos _____ timeElapsed2
1201
1202 if (timeElapsed2 >= uniao.inter2) // Executa quando completa um ciclo de duracao inter2
1203 { // Default 5 minutos
1204     temperatura = rtc.getTemperature(); // Pega temperatura
1205
1206     // Vamos pegar máximos e mínimos da voltagem das INAs (no dia)
1207     if (vina1 > vina1_max_dia) vina1_max_dia = vina1;
1208     else if (vina1 < vina1_min_dia) vina1_min_dia = vina1;
1209     else;
1210     if (vina2 > vina2_max_dia) vina2_max_dia = vina2;
1211     else if (vina2 < vina2_min_dia) vina2_min_dia = vina2;
1212     else;
1213     if (vina3 > vina3_max_dia) vina3_max_dia = vina3;
1214     else if (vina3 < vina3_min_dia) vina3_min_dia = vina3;
1215     else;
1216
1217     // Imprime data e hora atual
1218     Serial.println("\n");

```

```

1220 Serial.print(tempo);
1221 Serial.print("\nTemperatura: ");
1222 Serial.print(" Cº\n");
1223 Serial.print("\nEnergia Tracer1:  ");
1224 Serial.print(ue1, 2);
1225 Serial.print("Wh");
1226 Serial.print("\nEnergia Tracer2:  ");
1227 Serial.print(ue2, 2);
1228 Serial.print("Wh");
1229 Serial.print("\nEnergia Inversor:  ");
1230 Serial.print(ue3, 2);
1231 Serial.print("Wh");
1232 Serial.print("\nEnergia acumulada: ");
1233 Serial.print((ue1 + ue2), 2);
1234 Serial.print("Wh");
1235 Serial.print("\nEnergia consumida: ");
1236 Serial.print(ue3, 2);
1237 Serial.print("Wh");
1238
1239 timeElapsed2 = 0; // Reset do contador
1240 }
1241 }
1242
1243 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1244 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1245 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1246 //_____ Funções _____ telainicial
1247
1248 void telainicial() // Tela inicial do OLED
1249 {
1250 str = "CoMoSisFog " + versao;
1251 display.clear();
1252 display.setTextAlignment(TEXT_ALIGN_CENTER);
1253 display.setFont(ArialMT_Plain_10);
1254 display.drawString(63, 5, str);
1255 display.drawString(63, 25, "ESP32");
1256 display.drawString(63, 45, "Filippo Pardini");
1257 display.display();
1258 }
1259
1260 //_____ toggle_grupo_refletores
1261
1262 bool toggle_grupo_refletores(int grp)
1263 {
1264 // Faz o toggle (inversão) de um grupo de refletores: se estava ligado -> desliga e se estava desligado -> liga
1265 // Nota: Só pode executar para grupos do tipo comandado, automáticos ou programados não
1266
1267 int m;
1268
1269 // Vamos verificar se é um grupo ativo e do tipo comandado
1270 if ((uniaio.grupo_am[grp][0] == 0) || (uniaio.grupo_am[grp][1] != 2)) return false; // grupo inativo ou não comandado
1271
1272 // É ativo, verifica se está ligado ou não
1273 if (grupo_LD[grp] == 0) // Está desligado, liga
1274 {
1275 for (m=1;m<=uniaio.grupo_rf[grp][0];m++) // Varre os refletores do grupo
1276 {
1277 toggle_refletor(uniaio.grupo_rf[grp][m]); // Toggle refletor
1278 }
1279 grupo_LD[grp] = 1; // Marca grupo como ligado
1280 }
1281 else // Está ligado, desliga
1282 {
1283 for (m=1;m<=uniaio.grupo_rf[grp][0];m++) // Varre os refletores do grupo
1284 {
1285 toggle_refletor(uniaio.grupo_rf[grp][m]); // Toggle refletor
1286 }
1287 grupo_LD[grp] = 0; // Marca grupo como desligado
1288 }
1289 return true;
1290 }
1291
1292 //_____ desliga_todos_refletores
1293
1294 void desliga_todos_refletores(void)
1295 {
1296 uint8_t m;
1297
1298 Wire.beginTransmission(pcf1);
1299 Wire.write(B11111111); // Os relés ligam com 0, então para desligar 1
1300 Wire.endTransmission();
1301 Wire.beginTransmission(pcf2);
1302 Wire.write(B11111111); // Os relés ligam com 0, então para desligar 1
1303 Wire.endTransmission();
1304
1305 for (m=1;m<=12;m++) // Marca todos os grupos como desligados

```

```

1307 { refletor_ligado[m] = false; // Marca refletor m como desligado
1308 grupo_LD[m] = 0; // Marca grupo m como desligado
1309 }
1310
1311 tudo_ligado = false;
1312 desliga_inversor();
1313 }
1314
1315 // _____ liga todos os refletores
1316
1317 void liga_todos_refletores(void)
1318 {
1319     uint8_t m;
1320
1321     Wire.beginTransmission(pcf1);
1322     Wire.write(0x00); // Os relés ligam com 0
1323     Wire.endTransmission();
1324     Wire.beginTransmission(pcf2);
1325     Wire.write(0x00); // Os relés ligam com 0
1326     Wire.endTransmission();
1327
1328     for (m=1;m<=12;m++) // Marca todos os grupos como ligados
1329     {
1330         refletor_ligado[m] = true; // Marca refletor como ligado
1331         grupo_LD[m] = 1; // Marca grupo como ligado
1332     }
1333
1334     tudo_ligado = true;
1335     liga_inversor();
1336 }
1337
1338 // _____ liga_desliga_grupos_automaticos
1339
1340 void liga_desliga_grupos_automaticos(void)
1341 {
1342     int adr, ref, rf, n, m, ld;
1343     byte atual;
1344
1345     if (!tudo_ligado)
1346     {
1347         for (n=1;n<=12;n++) // Varre os grupos
1348         {
1349             if ((uniao.grupo_am[n][0] == 1) && (uniao.grupo_am[n][1] == 1)) // Grupo ativo e automático
1350             {
1351                 if (grupo_LD[n] == 0) // Grupo está desligado
1352                 {
1353                     // Tem que ligar?
1354                     if (digitalRead(fotocelula) == LOW) ld = 1; // Sim
1355                     else ld = 2; // Não
1356                 }
1357                 else // Está ligado
1358                 {
1359                     // Tem que desligar?
1360                     if (digitalRead(fotocelula) == HIGH) ld = 0; // Sim
1361                     else ld = 2; // Não
1362                 }
1363                 switch (ld)
1364                 {
1365                     case 2: // Sem ação, aguarda
1366                         break;
1367                     case 1: // Liga grupo
1368                         for (m=1;m<=uniao.grupo_rf[n][0];m++) // Varre os refletores do grupo
1369                         {
1370                             rf = uniao.grupo_rf[n][m]; // refletor (entre 1 e 12)
1371                             if (rf <= 8) {adr = pcf1; ref = rf;} // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
1372                             else {adr = pcf2; ref = rf - 8;} // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
1373
1374                             Wire.requestFrom(adr, 1); // Solicita 1 byte ao Pcf correto
1375                             while (Wire.available() == 0); // Aguarda estar no buffer
1376                             atual = Wire.read(); // Lê o byte, bit 0 -> refletor 1 até bit 7 -> refletor
1377                             // 8 se for Pcf1. No Pcf2, bit 0 -> refletor 9 até bit 3
1378                             // -> refletor 12
1379                             // ld = 1 => liga (bit = 0 no pcf)
1380                             refletor_ligado[rf] = true; // Marca como refletor ligado
1381
1382                             // Atualiza Pcf para acionar refletor do grupo nesse Pcf
1383                             Wire.beginTransmission(adr); // Inicia transmissão I2C para o Pcf correto
1384                             Wire.write(atual); // Novo byte de acionamento dos refletores do grupo
1385                             Wire.endTransmission(); // Encerra transmissão I2C
1386                         }
1387                         grupo_LD[n] = 1; // Marca grupo como ligado
1388                         if (!inversor_ligado) liga_inversor(); // Se inversor desligado, liga inversor
1389                         break;
1390                     case 0: // Desliga grupo
1391                         for (m=1;m<=uniao.grupo_rf[n][0];m++) // Varre os refletores do grupo
1392                         {

```



```

1394 rf = uniao.grupo_rf[n][m]; // refletor (entre 1 e 12)
1395 if (rf <= 8) {adr = pcf1; ref = rf;} // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
1396 else {adr = pcf2; ref = rf - 8;} // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
1397
1398 Wire.requestFrom(adr,1); // Solicita 1 byte ao Pcf correto
1399 while (Wire.available() == 0); // Aguarda estar no buffer
1400 atual = Wire.read(); // Lê o byte, bit 0 -> refletor 1 até bit 7 -> refletor
1401 // 8 se for Pcf1. No Pcf2, bit 0 -> refletor 9 até bit 3
1402 // -> refletor 12
1403 atual |= 1 << (ref - 1); // ld = 0 => desliga (bit = 1 no pcf)
1404 refletor_ligado[rf] = false; // Marca como refletor desligado
1405
1406 // Atualiza Pcf para acionar refletor do grupo nesse Pcf
1407 Wire.beginTransmission(adr); // Inicia transmissão I2C para o Pcf correto
1408 Wire.write(atual); // Novo byte de acionamento dos refletores do grupo
1409 Wire.endTransmission(); // Encerra transmissão I2C
1410 }
1411 grupo_LD[n] = 0; // Marca grupo como desligado
1412 if (!tem_algo_ligado()) desliga_inversor(); // Se não tem nada ligado, desliga inversor
1413 break;
1414 }
1415 }
1416 }
1417 }
1418
1419 // _____ liga_desliga_grupos_programados
1420
1421 void liga_desliga_grupos_programados(uint8_t ora,uint8_t mint)
1422 {
1423 // Para grupos programados iniciando em um dia e terminando no seguinte, à hora de desligamento (máximo 8.0) deverá
1424 // ser somado 24.0 Ex: ligar às 22:00 e desligar às 6:00 => ligar - 22.00; desligar - 30.00
1425
1426 int adr,ref,rf,n,m,ld;
1427 byte atual;
1428 float agora;
1429 ld = 2;
1430
1431 agora = (float)ora + ((float)mint / 100.0);
1432
1433 for (n=1;n<=12;n++) // Varre os grupos
1434 {
1435 if ((uniao.grupo_am[n][0] == 1) && (uniao.grupo_am[n][1] == 3)) // Grupo ativo e programado
1436 {
1437 if (grupo_LD[n] == 0) // Grupo está desligado
1438 {
1439 // Está na hora de ligar?
1440 if ((agora >= uniao.grupo_if[n][0]) && (agora < uniao.grupo_if[n][1])) ld = 1; // Sim
1441 else ld = 2; // Não
1442 }
1443 else // Está ligado
1444 {
1445 // É madrugada?
1446 if ((agora <= 8.0) && (uniao.grupo_if[n][1] > 24.0)) // Sim
1447 {
1448 if ((agora + 24.0) >= uniao.grupo_if[n][1]) ld = 0; // Está na hora de desligar
1449 else ld = 2; // Não
1450 }
1451 else // Não é madrugada
1452 {
1453 if (agora >= uniao.grupo_if[n][1]) ld = 0; // Está na hora de desligar
1454 else ld = 2; //Não
1455 }
1456 }
1457 switch (ld)
1458 {
1459 case 2: // Sem ação, aguarda
1460 break;
1461 case 1: // Liga grupo
1462 for (m=1;m<=uniao.grupo_rf[n][0];m++) // Varre os refletores do grupo
1463 {
1464 rf = uniao.grupo_rf[n][m]; // refletor (entre 1 e 12)
1465 if (rf <= 8) {adr = pcf1; ref = rf;} // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
1466 else {adr = pcf2; ref = rf - 8;} // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
1467
1468 Wire.requestFrom(adr,1); // Solicita 1 byte ao Pcf correto
1469 while (Wire.available() == 0); // Aguarda estar no buffer
1470 atual = Wire.read(); // Lê o byte, bit 0 -> refletor 1 até bit 7 -> refletor
1471 // 8 se for Pcf1. No Pcf2, bit 0 -> refletor 9 até bit 3
1472 // -> refletor 12
1473 atual &= ~(1 << (ref - 1)); // ld = 1 => liga (bit = 0 no pcf)
1474 refletor_ligado[rf] = true; // Marca como refletor ligado
1475
1476 // Atualiza Pcf para acionar refletor do grupo nesse Pcf
1477 Wire.beginTransmission(adr); // Inicia transmissão I2C para o Pcf correto
1478 Wire.write(atual); // Novo byte de acionamento dos refletores do grupo
1479 Wire.endTransmission(); // Encerra transmissão I2C

```

```

1481     grupo_LD[n] = 1; // Marca grupo como ligado
1482     if (!inversor_ligado) liga_inversor(); // Se inversor desligado, liga inversor
1483     break;
1484     case 0: // Desliga grupo
1485         for (m=1;m<=uniaio.grupo_rf[n][0];m++) // Varre os refletores do grupo
1486         {
1487             rf = uniao.grupo_rf[n][m]; // refletor (entre 1 e 12)
1488             if (rf <= 8) {adr = pcf1; ref = rf;} // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
1489             else {adr = pcf2; ref = rf - 8;} // Calcula qual Pcf (1 ou 2) endereço adr e refletor ref
1490
1491             Wire.requestFrom(adr,1); // Solicita 1 byte ao Pcf correto
1492             while (Wire.available() == 0); // Aguarda estar no buffer
1493             atual = Wire.read(); // Lê o byte, bit 0 -> refletor 1 até bit 7 -> refletor
1494             // 8 se for Pcf1. No Pcf2, bit 0 -> refletor 9 até bit 3
1495             // -> refletor 12
1496             atual |= 1 << (ref - 1); // ld = 0 => desliga (bit = 1 no pcf)
1497             refletor_ligado[rf] = false; // Marca como refletor desligado
1498
1499             // Atualiza Pcf para acionar refletor do grupo nesse Pcf
1500             Wire.beginTransmission(adr); // Inicia transmissão I2C para o Pcf correto
1501             Wire.write(atual); // Novo byte de acionamento dos refletores do grupo
1502             Wire.endTransmission(); // Encerra transmissão I2C
1503         }
1504         grupo_LD[n] = 0; // Marca grupo como desligado
1505         if (!tem_algo_ligado()) desliga_inversor(); // Se não tem nada ligado, desliga inversor
1506         break;
1507     }
1508     ld = 2;
1509 }
1510 }
1511 }
1512
1513 // _____ dias_desde_01_01_2019
1514
1515 unsigned int dias_desde_01_01_2019(unsigned int dia,unsigned int mes,unsigned int ano)
1516 {
1517     unsigned int an,nd = 0;
1518     unsigned int dd[14] = {0,0,31,59,90,120,151,181,212,243,273,304,334,365};
1519
1520     for (an=2019;an<ano;an++)
1521     {
1522         if ((an % 4) == 0) nd += 366; // É ano bissexto (vale até 2100)
1523         else nd += 365;
1524     }
1525     if (((an % 4) == 0) && (mes > 2)) nd += (dd[mes] + dia + 1); // É ano bissexto (vale até 2100)
1526     else nd += (dd[mes] + dia);
1527     return nd;
1528 }
1529
1530 // _____ SD_readStringUntil
1531
1532 bool SD_readStringUntil(const char car)
1533 {
1534     // Lê o string de caracteres até o primeiro caractere car encontrado inclusive
1535
1536     str = "";
1537     char cc;
1538
1539     while(1)
1540     {
1541         cc = arquivo.read();
1542         if ((uint8_t)cc == -1) return false;
1543         if (cc != car) str += cc;
1544         else
1545         {
1546             str += cc;
1547             return true;
1548         }
1549     }
1550 }
1551
1552 // _____ tem_algo_ligado
1553
1554 bool tem_algo_ligado(void)
1555 {
1556     uint8_t kk;
1557     bool ret = false;
1558
1559     for (kk=1;kk<=12;kk++)
1560     {
1561         ret |= refletor_ligado[kk];
1562     }
1563     return ret;
1564 }
1565
1566 // _____ liga_inversor

```

```

1568 void liga_inversor(void)
1569 {
1570     // 0 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     // 0 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
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 {

```

```

1654 Serial2.print(*msg_in);
1655 Serial2.flush();
1656 }
1657 }
1658 // _____ recebe_comando_de_P2
1659
1660 bool recebe_comando_de_P2(String *cmd)
1661 {
1662   if (Serial2.available() >= 1)
1663   {
1664     if (Serial2.peek() != '<')
1665     {
1666       Serial.print("\nComando invalido");
1667       String(xy) = String("<Comando invalido>");
1668       Serial2_print_flush(&xy);
1669       limpa_buffer_serial2();
1670       return false;
1671     }
1672     else
1673     {
1674       *cmd = Serial2.readStringUntil('>') + String('>');
1675       Serial.print("\n");
1676       Serial.print(*cmd);
1677       return true;
1678     }
1679   }
1680   else return false;
1681 }
1682
1683 // _____ decodifica_comando_recebido
1684
1685 bool decodifica_comando_recebido(String tmp,int *codcom,int *np,int *param)
1686 {
1687   // Decodifica um comando recebido no formato : <*[*p1...*pn]>
1688   // onde:
1689   // c => comando (obrigatório); p1...pn => parametros (opcional); * => divisor (primeiro obrigatório,
1690   // outros se houver parametros); < e > => delimitadores (obrigatório)
1691
1692   int ni,nf,n; // ni => indice inicial; nf => indice final; n => número de parametros
1693
1694   n = 0;
1695
1696   if ((tmp.indexOf('<') != -1) && (tmp.indexOf('>') != -1) && (tmp.indexOf('*') == 1)) // Garante <* e >
1697   {
1698     tmp = tmp.substring(0,tmp.indexOf('>')) + String('>'); // Corta além do >
1699     ni = tmp.indexOf('*') + 1; // Indice inicio codigo de comando
1700     nf = tmp.indexOf('*',ni); // Indice fim codigo de comando
1701     if (nf == -1) // Se não achou (-1), não tem parametros
1702     {
1703       nf = tmp.indexOf('>',ni); // Novo indice fim codigo de comando
1704       *codcom = tmp.substring(ni,nf).toInt(); //Codigo de comando sem parametros
1705       *np = n; // Numero de parametros => 0
1706       return true; // Retorna verdadeiro
1707     }
1708
1709     // Tem parametros
1710     *codcom = tmp.substring(ni,nf).toInt(); //Codigo de comando com parametros
1711     while(1) // Cicla
1712     {
1713       n++; // Conta parametros
1714       ni = nf + 1; // Indice inicial parametro
1715       nf = tmp.indexOf('*',ni); // Indice final parametro
1716       if (nf == -1) // Se -1 é ultimo parametro
1717       {
1718         nf = tmp.indexOf('>',ni); // Novo indice final parametro
1719         param[n] = tmp.substring(ni,nf).toInt(); // Parametro n
1720         *np = n; // número de parametros
1721         return true; // Retorna verdadeiro
1722       }
1723       else param[n] = tmp.substring(ni,nf).toInt(); // Tem mais parametros
1724     }
1725   }
1726   else return false; // Comando invalido, retorna falso
1727 }
1728
1729 // _____ limpa_buffer_serial2
1730
1731 void limpa_buffer_serial2(void)
1732 {
1733   while (Serial2.available()) Serial2.read();
1734 }
1735
1736 // _____ write_eeprom
1737
1738 void write_eeprom(unsigned int addr, byte data)
1739 {
1740   Wire.beginTransmission(0x57); // Endereço I2C da eeprom

```

```

1741 Wire.write((int)(addr >> 8)); // Envia MSB do endereço na eeprom
1742 Wire.write((int)(addr & 0xFF)); // Envia LSB do endereço na eeprom
1743 Wire.write(data); // Envia byte de dados a ser gravado
1744 Wire.endTransmission();
1745 delay(5); // Tempo necessário à gravação na eeprom (min 3,5mseg)
1746 }
1747
1748 // _____ read_eeprom
1749
1750 byte read_eeprom(unsigned int addr)
1751 {
1752     byte rdata = 0xFF;
1753
1754     Wire.beginTransmission(0x57); // Endereço I2C da eeprom
1755     Wire.write((int)(addr >> 8)); // Envia MSB do endereço na eeprom
1756     Wire.write((int)(addr & 0xFF)); // Envia LSB do endereço na eeprom
1757     Wire.endTransmission();
1758     Wire.requestFrom(0x57,1); // Solicita o byte
1759     while (!Wire.available());
1760     rdata = Wire.read(); // Se disponível, lê byte
1761     delay(5);
1762     return rdata; // Retorna o byte lido ou 0xFF se não conseguiu ler
1763 }
1764
1765 // _____ grava_estrutura_eeprom
1766
1767 int grava_estrutura_eeprom(void)
1768 {
1769     int n;
1770     byte *p = uniao.byteArray;
1771
1772     for (n=0;n<sizeof(uniao.byteArray);n++)
1773     {
1774         write_eeprom(n,*p++);
1775     }
1776     return n;
1777 }
1778
1779 // _____ le_estrutura_eeprom
1780
1781 int le_estrutura_eeprom(void)
1782 {
1783     int n;
1784     byte *p = uniao.byteArray;
1785
1786     for (n=0;n<sizeof(uniao.byteArray);n++)
1787     {
1788         *p++ = read_eeprom(n);
1789     }
1790     return n;
1791 }
1792
1793 // _____ aguarda_dados_serial2
1794
1795 bool aguarda_dados_serial2(int esp,int ponto)
1796 {
1797     unsigned long tim;
1798     String to;
1799
1800     tim = (unsigned long)esp * 1000;
1801     espera = 0;
1802
1803     while ((Serial2.available() == 0) && (espera <= tim)); // Aguarda chegar OK('#') de P2
1804     if (Serial2.available() > 0)
1805     {
1806         Serial2.read(); // Retira o '#'
1807         return true;
1808     }
1809     else
1810     {
1811         to = String("Timeout - ponto ") + String(ponto);
1812         Serial.print("\n"); Serial.print(to);
1813         return false;
1814     }
1815 }
1816
1817 // _____
1818
1819
1820
1821

```