```c
/* ********************************************************************************** */
/*                                                                                    */
/*   Nome do arquivo:        communicationStateMachine.c                              */
/*                                                                                    */
/*   Descricao:              Arquivo contendo as funcoes de que implementam uma       */
/*                         maquina de estados para comunicacao UART                   */
/*                                                                                    */
/*   Autores:                Gustavo Lino e Giacomo Dollevedo                         */
/*   Criado em:              21/05/2020                                               */
/*   Ultima revisao em:      29/07/2020                                               */
/* ********************************************************************************** */

/*REVISAO:*/
/*ALTERADO O FUNCIONAMENTO DA FUNCAO "setParam"*/
/*ALTERADA AS -VARIAVEIS DE TESTE- */

/*Revisão: 28/07/2020 23:18*/
/*Inserido a possibilidade de setar os parametros de ganho Kp, Ki e Kd*/

/* Comandos utilizados (dicionario):
   "#gt;" Get valor de temperatura atual
   "#gc;" Get Duty Cycle Cooler
   "#gh;" Get Duty Cycle Heater

   "#sb<N>;" Set Button On/OFF, onde N é qualquer número de até de 7 bytes.
   "#st<N>;" Set Temperatura Máxima desejada para controle, onde N é qualquer número de até de 7 byte.
   "#sc<N>;" Set duty cycle do cooler, onde N é qualquer número de até de 7 bytes.
   "#sh<N>;" Set duty cycle do heater, onde N é qualquer número de até de 7 bytes.
   "#sp<N>;" Set ganho prporcional do controlador PID, onde N é qualquer número de até 7 bytes.
   "#si<N>;" Set ganho integrativo do controlador PID, onde N é qualquer número de até 7 bytes.
   "#sd<N>;" Set ganho derivativo do controlador PID, onde N é qualquer número de até 7 bytes.



   "<#a<p>;" Respostas do parametro solicitado, onde p pode ser t,c ou h.

   É previsto nas próximas versões do código que haja a solicitação de todos os parametros e sua respostas

*/

#include "aquecedorECooler.h"
#include "variaveis_globais.h"
#include <stdlib.h>



#define IDLE    0
#define READY   1
#define GET     2
#define SET     3
#define PARAM   4
#define VALUE   5

#define MAX_VALUE_LENGTH  7

unsigned char ucCurrentState = IDLE;
unsigned char ucValueCounter = 0;

/*VARIAVEIS DE TESTE*/
unsigned char ucMaxTempTest[MAX_VALUE_LENGTH+1] = "30";
unsigned char ucCurrTempTest[MAX_VALUE_LENGTH+1]= "25";
unsigned char fCoolerDutyTest[5] = "0,75";
unsigned char fHeaterDutyTest[5] = "0,50";


/* ********************************************************************************** */
/* Nome do metodo:          returnParam                                               */
/* Descricao:               Imprime no terminal de comunicacao UART os parametros     */
/*                        solicitados pelo comando Get                                */
/*                                                                                    */
/* Parametros de entrada:   ucParam -> Parametro solicitado (de acordo com dicionario)*/
/*                                                                                    */
/* Parametros de saida:     n/a                                                       */
/* ********************************************************************************** */
void returnParam(unsigned char ucParam){

   switch(ucParam){

      case 't':
         debug_printf("#a%dºC;", ucTempAtual);
         break;

```

```
80        case 'c':
81          debug_printf("#a%c%c%c%c;", fCoolerDutyTest[0], fCoolerDutyTest[1], fCoolerDutyTest[2], fCoolerDutyTest[3]);
82          break;
83
84        case 'h':
85          debug_printf("#a%c%c%c%c;", fHeaterDutyTest[0], fHeaterDutyTest[1] ,fHeaterDutyTest[2], fHeaterDutyTest[3]);
86          break;
87
88        case 'p':
89          debug_printf("#a%f;", fKp);
90          break;
91
92        case 'i':
93          debug_printf("#a%f;", fKi);
94          break;
95
96        case 'd':
97          debug_printf("#a%f;", fKd);
98          break;
99
100     }
101
102
103 }
104
105
106 /* ************************************************************************** */
107 /* Nome do metodo:         setParam                                           */
108 /* Descricao:              Define valores de controle/usabilidade necessarios para */
109 /*                         garantir a interface e funcionamento adequado do uC: */
110 /*       Temperatura Máxima          */
111 /*       Disponibilidade dos botoes          */
112 /*                                                                            */
113 /* Parametros de entrada:   ucParam -> Parametro que sera alterado      */
114 /*        *ucValue ->  Array com valores de alteracao             */
115 /*                                                                            */
116 /* Parametros de saida:     n/a              */
117 /* ************************************************************************** */
118 void setParam(unsigned char ucParam, unsigned char *ucValue){
119
120     unsigned char ucContador = 0;
121     unsigned char ucFlag = 0;
122     unsigned char ucStrValue[5] = "0,00\0";
123     float fAux = 0;
124
125     switch(ucParam){
126
127         case 't':
128           while('\0' != ucValue[ucContador]){
129           //Pega o valor da dezena
130               if(0 == ucContador){
131                   if(9 == ucValue[ucContador]){
132                       //Limita em 90 a temperatura maxima
133                       fAux = 90;
134                       ucTempAlvo = fAux;
135                       ucDezTempAlvo = 9;
136                       ucUnTempAlvo = 0;
137                       break;
138                   }
139                   fAux = ucValue[ucContador] -48;
140                   fAux = fAux*10;
141               }
142           //Pega o valor da unidade
143               else if(1 == ucContador){
144                   fAux = fAux + (ucValue[ucContador] -48);
145                   ucTempAlvo = fAux;
146               }
147               if(2 < ucContador){
148           //Caso o usuario tente inserir uma temperatura maior que 2 digitos, seta a temperatura alvo par ao padrão
149                   ucTempAlvo = 30;
150               }
151               ucContador++;
152           }
153           ucDezTempAlvo = ucTempAlvo/10;
154           ucUnTempAlvo = ucTempAlvo%10;
155           break;
156
157       //Habilitar ou desabilitar os botões da interface do microcontrolador.
158       case 'b':
159           /* Espera ucValue num formato especifico*/
160           if('\0' != ucValue[4]) {
161
```

```
            if((0 != ucValue[0] && 1 != ucValue[0]) && (0 != ucValue[1] && 1 != ucValue[1]))
                if((0 != ucValue[2] && 1 != ucValue[2]) && (0 != ucValue[3] && 1 != ucValue[3]))
                    ledSwi_init(ucValue[0], ucValue[1], ucValue[2], ucValue[3]);
                    break;
        }

        else{
            break;
        }

    /* Caso para setar Duty Cycle do cooler */
    case 'c':
        ucContador  = 0;
        ucFlag      = 0;
        fAux        = 0;

        while('\0' != ucValue[ucContador]){
            if('1' == ucValue[0]){
                coolerfan_PWMDuty(1.0);
                break;
            }

            if(',' == ucValue[ucContador]){
                ucFlag = 1;
            }

            else if (1 == ucFlag){
                fAux += ucValue[ucContador] - 48;
                fAux = fAux*10;
            }

            ucContador++;
        }

        if(1 == ucFlag){
            while(fAux > 0)
                fAux = fAux/10;

        coolerfan_PWMDuty(fAux);
        }

        break;

    /* Caso para setar Duty Cycle do heater */
    case 'h':
        ucContador  = 0;
        ucFlag      = 0;
        fAux        = 0;
        while('\0' != ucValue[ucContador]){
            if('1' == ucValue[0]){ //Seria melhor generalizar para qualquer valor diferente de zero?
                heater_PWMDuty(0.5);
                fDutyCycle_Heater = 0.5;
                break;
            }

            if(',' == ucValue[ucContador]){
                ucFlag = 1;
            }

            else if (1 == ucFlag){
                fAux += ucValue[ucContador] - 48;
                fAux = fAux*10;
            }

            ucContador++;
        }

        if(1 == ucFlag){
            while(fAux > 0)
                fAux = fAux/10;

            if(0.5 < fAux){
                heater_PWMDuty(0.5);
                fDutyCycle_Heater = 0.5;
            }

            else{
                heater_PWMDuty(fAux);
                fDutyCycle_Heater = fAux;
            }

        break;
```

```
245         break;
246
247     case 'p':
248
249
250         ucContador  = 0;
251         fAux        = 0;
252
253         while('\0' != ucValue[ucContador]){
254
255             if(',' != ucValue[ucContador]){
256                 ucStrValue[ucContador] = ucValue[ucContador];
257             }
258             //Converte virgula para ponto
259             else{
260
261                 ucStrValue[ucContador] = '.';
262
263             }
264             ucContador++;
265         }
266
267         fAux = strtof(ucStrValue, NULL);
268         fKp = fAux;
269
270         break;
271
272     case 'i':
273
274
275         ucContador  = 0;
276         fAux        = 0;
277
278         while('\0' != ucValue[ucContador]){
279
280             if(',' != ucValue[ucContador]){
281                 ucStrValue[ucContador] = ucValue[ucContador];
282             }
283             //Converte virgula para ponto
284             else{
285
286                 ucStrValue[ucContador] = '.';
287
288             }
289             ucContador++;
290         }
291
292         fAux = strtof(ucStrValue, NULL);
293         fKi = fAux;
294
295         break;
296
297     case 'd':
298
299
300         ucContador  = 0;
301         fAux        = 0;
302
303         while('\0' != ucValue[ucContador]){
304
305             if(',' != ucValue[ucContador]){
306                 ucStrValue[ucContador] = ucValue[ucContador];
307             }
308             //Converte virgula para ponto
309             else{
310
311                 ucStrValue[ucContador] = '.';
312
313             }
314             ucContador++;
315         }
316
317         fAux = strtof(ucStrValue, NULL);
318         fKd = fAux;
319
320         break;
321
322
323   }
324
325 }
326 }
327
```

```
/* ********************************************************************************** */
/* Nome do metodo:         processByteCommUART                                        */
/* Descricao:              Realiza todos os processos para que a comunicacao UART     */
/*                      ocorra, baseado numa maquina de estados                       */
/*                                                                                    */
/* Parametros de entrada:   ucCmdByte-> Comandos em Bytes enviados por UART           */
/*                                                                                    */
/* Parametros de saida:     n/a                                                       */
/* ********************************************************************************** */
void processByteCommUART(unsigned char ucCmdByte){


   static unsigned char ucParam;
   static unsigned char ucValue[MAX_VALUE_LENGTH + 1];

   if('#' == ucCmdByte)
      ucCurrentState = READY;

   else
      if(IDLE != ucCurrentState)
         switch(ucCurrentState ){

            case READY:
               if('g' == ucCmdByte)
                  ucCurrentState = GET;

               if('s' == ucCmdByte)
                  ucCurrentState = SET;

               else
                  ucCurrentState = IDLE;

               break;


            case GET:
               if('t' == ucCmdByte || 'c' == ucCmdByte || 'h' == ucCmdByte || 'i' == ucCmdByte || 'p' == ucCmdByte || 'd' == ucCmdByte){
                  ucParam = ucCmdByte;
                  ucCurrentState = PARAM;
               }

               else
                  ucCurrentState = IDLE;

               break;

            case SET:
               if('t' == ucCmdByte || 'b' == ucCmdByte || 'c' == ucCmdByte || 'h' == ucCmdByte || 'p' == ucCmdByte || 'i' == ucCmdByte || 'd' == ucCmdByte){
                  ucParam = ucCmdByte;
                  ucValueCounter = 0;
                  ucCurrentState = VALUE;
               }

               else
                  ucCurrentState = IDLE;

               break;

            case PARAM:
               if(';' == ucCmdByte)
                  returnParam(ucParam);

               ucCurrentState = IDLE;

               break;

            case VALUE:
               if((ucCmdByte >= '0' && ucCmdByte <= '9') || ','){
                  if(ucValueCounter < MAX_VALUE_LENGTH){
                     ucValue[ucValueCounter] = ucCmdByte;
                     ucValueCounter++;
                  }
               }

               else{
                  if(';' == ucCmdByte){
                     ucValue[ucValueCounter] = '\0';
                     setParam(ucParam, ucValue);
                  }

                  ucCurrentState = IDLE;
```

```
410                     ucCurrentState = IDLE;
411
412                 }
413                 break;
414
415             default:
416                 ucCurrentState = IDLE;
            }
    }
```