```c
/* ************************************************************** */
/* File name:       adc.c                                         */
/* File description: This file has a couple of useful functions to   */
/*                  control the ADC from the peripheral board.      */
/*                  The converter is connected to the Temperature   */
/*                  sensor.                                       */
/* Author name:      dloubach, julioalvesMS, lagoAF e rbacurau       */
/* Creation date:    07jun2018                                     */
/* Revision date:    20mai2020                                     */
/* ************************************************************** */

#include "board.h"
#include "adc.h"

#define ADC0_SC1A_COCO (ADC0_SC1A >> 7)
#define ADC0_SC2_ADACT (ADC0_SC2 >> 7)

#define ADC_CFG1_BUS_CLK_2   01U
#define ADC_CFG1_CONVERSION  00U
#define ADC_CFG1_SAMPLE_TIME  0U
#define ADC_CFG1_CLK_DIVIDER 00U
#define ADC_CFG1_LOW_POWER    0U

#define ADC_SC2_VOLT_REF    00U
#define ADC_SC2_DMA          0U
#define ADC_SC2_COMPARE      0U
#define ADC_SC2_TRIGGER_CONV 0U

#define ADC_CFG2_LONG_SAMPLE 00U
#define ADC_CFG2_HIGH_SPEED   0U
#define ADC_CFG2_ASYNC_CLK    0U
#define ADC_CFG2_MUX_SELECT   0U

#define ADC_SC1A_COMPLETE     4U
#define ADC_SC1A_INTERRUPT    0U
#define ADC_SC1A_DIFFERENTIAL 0U

#define CGC_CLOCK_ENABLED 1 //ASSUMINDO QUE SEJA 1 OU 0 (NAO TINHA NOS DEFINES).
      //pra arrumar (se n funcionar) pagina 206 do KL25 Sub-Family Reference Manual

/* ************************************************** */
/* Method name:       adc_initADCModule               */
/* Method description: Init a the ADC converter device */
/* Input params:      n/a                             */
/* Output params:     n/a                             */
/* ************************************************** */
void adc_initADCModule(void)
{
  /* un-gate port clock*/
  SIM_SCGC6 |= SIM_SCGC6_ADC0(CGC_CLOCK_ENABLED);    //Enable clock for ADC

  /* un-gate port clock*/
  SIM_SCGC5 |= SIM_SCGC5_PORTE(CGC_CLOCK_ENABLED);

  /* set pin as ADC In */
  PORTE_PCR21 |= PORT_PCR_MUX(THERMOMETER_ALT);   //Temperature Sensor

  /*
  ADC_CFG1_ADICLK(x)// bus/2 clock selection
  ADC_CFG1_MODE(x)  // 8-bit Conversion mode selection
  ADC_CFG1_ADLSMP(x)// Short sample time configuration
  ADC_CFG1_ADIV(x)  // Clock Divide Select (Divide by 1)
  ADC_CFG1_ADLPC(x) // Normal power Configuration
  */
  ADC0_CFG1 |= (ADC_CFG1_ADICLK(ADC_CFG1_BUS_CLK_2) | ADC_CFG1_MODE(ADC_CFG1_CONVERSION) | ADC_CFG1_ADLSMP(ADC_CFG1_SAMPLE_TIME) | ADC_CFG1_ADIV(ADC_C

  /*
  ADC_SC2_REFSEL(x)// reference voltage selection - external pins
  ADC_SC2_DMAEN(x) // dma disabled
  ADC_SC2_ACREN(x) // dont care - range function
  ADC_SC2_ACFGT(x) // dont care - 0 -> Less than, 1 -> Greater Than
  ADC_SC2_ACFE(x)  // compare function disabled
  ADC_SC2_ADTRG(x) // When software trigger is selected, a conversion is initiated following a write to SC1A
  ADC_SC2_ADACT(x) // HW-set indicates if a conversion is being held, is cleared when conversion is done
  */
  ADC0_SC2 |= (ADC_SC2_REFSEL(ADC_SC2_VOLT_REF) | ADC_SC2_DMAEN(ADC_SC2_DMA) | ADC_SC2_ACFE(ADC_SC2_COMPARE) | ADC_SC2_ADTRG(ADC_SC2_TRIGGER_CONV));

  /*
  ADC_CFG2_ADLSTS(x)  // default time
  ADC_CFG2_ADHSC(x)   // normal conversion sequence
  DC_CFG2_ADACKEN(x) // disable adack clock
  ADC_CFG2_MUXSEL(x)  // select 'a' channels
  */
  ADC0_CFG2 |= (ADC_CFG2_ADLSTS(ADC_CFG2_LONG_SAMPLE) | ADC_CFG2_ADHSC(ADC_CFG2_HIGH_SPEED) | ADC_CFG2_ADACKEN(ADC_CFG2_ASYNC_CLK) | ADC_CFG2_MUXSEL
}


/* ************************************************** */
/* Method name:       adc_initConvertion              */
/* Method description: init a conversion from A to D  */
/* Input params:      n/a                             */
/* Output params:     n/a                             */
/* ************************************************** */
void adc_initConvertion(void)
{
   /*
   ADC_SC1_COCO(x) // conversion complete flag HW-set
   ADC_SC1_AIEN(x) // conversion complete interrupt disables
   ADC_SC1_DIFF(x) // selects single-ended convertion
   ADC_SC1_ADCH(x) // selects channel, view 3.7.1.3.1 ADC0 Channel Assignment ADC0_SE4a from datasheet
   */
 ADC0_SC1A &= (ADC_SC1_ADCH(ADC_SC1A_COMPLETE) | ADC_SC1_DIFF(ADC_SC1A_DIFFERENTIAL) | ADC_SC1_AIEN(ADC_SC1A_INTERRUPT));
}

/* ************************************************** */
/* Method name:       adc_isAdcDone                   */
/* Method description: check if conversion is done    */
```

```
/* Method description: check if conversion is done     */
/* Input params:       n/a                             */
/* Output params:      char: 1 if Done, else 0         */
/* ************************************************** */
char adc_isAdcDone(void)
{
   if(ADC0_SC1A_COCO) // watch complete convertion flag
      return 1; // if the conversion is complete, return 1
   else
      return 0; // if the conversion is still taking place, return 0
}

/* ************************************************** */
/* Method name:        adc_getConvertionValue         */
/* Method description: Retrieve converted value       */
/* Input params:       n/a                            */
/* Output params:      int: Result from convertion    */
/* ************************************************** */
int adc_getConvertionValue(void)
{
   return ADC0_RA; // return the register value that keeps the result of convertion
}
```

```c
/* ************************************************************ */
/* File name:        adc.h                                      */
/* File description: This file has a couple of useful functions to   */
/*                   control the ADC from the peripheral board.      */
/*                   The converter is connected to the Temperature   */
/*                   sensor.                                    */
/* Author name:      dloubach, julioalvesMS, lagoAF e rbacurau       */
/* Creation date:    07jun2018                                  */
/* Revision date:    20mai2020                                  */
/* ************************************************************ */

#ifndef SOURCES_ADC_H_
#define SOURCES_ADC_H_


/* ************************************************ */
/* Method name:        adc_initADCModule              */
/* Method description: Init a the ADC converter device */
/* Input params:       n/a                            */
/* Output params:      n/a                            */
/* ************************************************ */
void adc_initADCModule(void);


/* ************************************************ */
/* Method name:        adc_initConvertion             */
/* Method description: init a conversion from A to D  */
/* Input params:       n/a                            */
/* Output params:      n/a                            */
/* ************************************************ */
void adc_initConvertion(void);


/* ************************************************ */
/* Method name:        adc_isAdcDone                  */
/* Method description: check if conversion is done    */
/* Input params:       n/a                            */
/* Output params:      char: 1 if Done, else 0        */
/* ************************************************ */
char adc_isAdcDone(void);


/* ************************************************ */
/* Method name:        adc_getConvertionValue         */
/* Method description: Retrieve converted value       */
/* Input params:       n/a                            */
/* Output params:      int: Result from convertion    */
/* ************************************************ */
int adc_getConvertionValue(void);


#endif /* SOURCES_ADC_H_ */
```

```c
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:       aquecedorECooler.c                                */
/*                                                                            */
/*   Descricao:             Arquivo contendo a implementacao                  */
/*                        das funcoes de interface do microcontrolador        */
/*                        com a resistencia de aquecimento e cooler do kit    */
/*                                                                            */
/*   Autores:              Gustavo Lino e Giacomo Dollevedo                   */
/*   Criado em:             21/04/2020                                        */
/*   Ultima revisao em:     29/07/2020                                        */
/* ************************************************************************** */

/* REVISÃƒO: */
/* ALTERADO PARA "&=" no UP_COUNTING, e sua mascara no board.h */
/* ADICIONADO DISABLE_COUNTER */
/* ALTERADO PARA "&=" no EDGE_ALIGNED, e sua mascara no board.h */
/* ADICIONADO UM CLEAR PARA "TPM_CNT" na inicializacao*/
/* ADICIONADA UMA FUNCAO PARA RESETAR O "TPM_CNT" (PWM_clearCounter)*/
/* ADICIONADO "CLEAR_16" no board.h */
/* ALTERADAS AS MASCARAS PARA "CLOCK_DIVIDER" no board.h*/
/* ALTERADO PARA "&=" no CLOCK_DIVIDER */


#include "board.h"
#include "aquecedorECooler.h"


/* ************************************************************************** */
/* Nome do metodo:         PWM_init                                           */
/* Descricao:              Inicializa os registradores para funcionamento do PWM   */
/*                       entre 5 e 20Hz                                       */
/*                                                                            */
/* Parametros de entrada:    n/a                                              */
/*                                                                            */
/* Parametros de saida:      n/a                                              */
/* ************************************************************************** */
void PWM_init(void){


    /* Liberando o Clock para o timer/pwm*/
    SIM_SCGC6 |= TPM1_CLOCK_GATE;

    /*Divisor pro clock*/
    TPM1_SC   |= CLOCK_DIVIDER_64;

    /*Selecao do clock de 32kHz*/
    /*MCGIRCLK == internal reference clock*/
    SIM_SOPT2 |= MCGIRCLK_SELECT;


    /*Desabilitando o LPTPM Counter para poder alterar o modo de contagem*/
    TPM1_SC &= DISABLE_COUNTER;

    /*Modo de up-counting */
    TPM1_SC &= PWM_UP_COUNTING;

    /*Incrementa a cada pulso*/
    TPM1_SC   |= PWM_EVERY_CLOCK;

    /*Modulo configurado para 49 (chegar numa freq de 10Hz)*/
    /*Portanto, conta ate 50 (0 a 49)*/
    TPM1_MOD  |= 0x0031;

    /*Configurando modo Edge Aligned PWM e High True Pulses nos canais 0 e 1*/
    TPM1_C0SC &= EDGE_ALIGNED_HIGH_TRUE;
```

```
67      TPM1_C1SC &= EDGE_ALIGNED_HIGH_TRUE;
68
69      /*DUTY CYCLE 50%*/
70      /*Inverte o sinal apos contar 25 vezes*/
71      TPM1_C0V |= 0x0019;
72      TPM1_C1V |= 0x0019;
73
74      TPM1_CNT &= CLEAR_16;
75
76      coolerfan_init();
77      heater_init();
78
79
80  }
81
82
83  /* ************************************************************************** */
84  /* Nome do metodo:          PWM_clearCounter                                  */
85  /* Descricao:               Reseta o contador TPM1_CNT para nao haver overflow */
86  /*                                                                            */
87  /* Parametros de entrada:    n/a                                              */
88  /*                                                                            */
89  /* Parametros de saida:      n/a                                              */
90  /* ************************************************************************** */
91  void PWM_clearCounter(void){
92
93      if(TPM1_CNT >= 0x7FFF)
94          TPM1_CNT &= CLEAR_16;
95
96  }
97
98  /* ************************************************************************** */
99  /* Nome do metodo:          coolerfan_init                                    */
100 /* Descricao:               Configura a liberacao do sinal PWM no pino PTA13  */
101 /*                                                                            */
102 /* Parametros de entrada:    n/a                                              */
103 /*                                                                            */
104 /* Parametros de saida:      n/a                                              */
105 /* ************************************************************************** */
106 void coolerfan_init(void){
107
108     SIM_SCGC5 |= PORTA_CLOCK_GATE;
109
110     PORTA_PCR13 |= MUX_ALT3;
111
112 }
113
114 /* ************************************************************************** */
115 /* Nome do metodo:          heater_init                                       */
116 /* Descricao:               Configura a liberacao do sinal PWM no pino PTA12  */
117 /*                                                                            */
118 /* Parametros de entrada:    n/a                                              */
119 /*                                                                            */
120 /* Parametros de saida:      n/a                                              */
121 /* ************************************************************************** */
122 void heater_init(void){
123
124     SIM_SCGC5 |= PORTA_CLOCK_GATE;
125
126     PORTA_PCR12 |= MUX_ALT3;
127 }
128
129
130 /* ************************************************************************** */
131 /* Nome do metodo:          coolerfan_PWMDuty                                 */
132 /* Descricao:               Configura o Duty Cycle do PWM para o cooler       */
133 /*                                                                            */
```

```
/* Parametros de entrada:   fCoolerDuty -> valor entre 0 e 1 indicando o Duty Cycle */
/*                                                                                  */
/* Parametros de saida:     n/a                                                     */
/* ******************************************************************************** */
void coolerfan_PWMDuty(float fCoolerDuty){

    unsigned char ucDuty = 0;

    if(0 <= fCoolerDuty && 1 >= fCoolerDuty){
        ucDuty = 50*fCoolerDuty;
    }

    else{
        ucDuty = 0x0019;
    }

    TPM1_C1V &= CLEAR_16;
    TPM1_C1V |= ucDuty;

}

/* ******************************************************************************** */
/* Nome do metodo:          heater_PWMDuty                                          */
/* Descricao:               Configura o Duty Cycle do PWM para o cooler             */
/*                                                                                  */
/* Parametros de entrada:   fHeaterDuty -> valor entre 0 e 1 indicando o Duty Cycle */
/*                                                                                  */
/* Parametros de saida:     n/a                                                     */
/* ******************************************************************************** */
void heater_PWMDuty(float fHeaterDuty){

    unsigned char ucDuty = 0;

    if(0 <= fHeaterDuty && 1 >= fHeaterDuty){
         ucDuty = 50*fHeaterDuty;
    }

    else{
        ucDuty = 0x0019;
    }

    TPM1_C0V &= CLEAR_16;
    TPM1_C0V |= ucDuty;

}
```

```
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:       aquecedorECooler.h                                */
/*                                                                            */
/*   Descricao:             Arquivo Header contendo a declaracao              */
/*                        das funcoes de interface do microcontrolador        */
/*                        com a resistencia de aquecimento e cooler do kit    */
/*                                                                            */
/*   Autores:              Gustavo Lino e Giacomo Dollevedo                   */
/*   Criado em:             21/04/2020                                        */
/*   Ultima revisao em:     24/07/2020                                        */
/* ************************************************************************** */

#ifndef SOURCES_COOLER_HEATER_
#define SOURCES_COOLER_HEATER_


/* ************************************************************************** */
/* Nome do metodo:         PWM_init                                           */
/* Descricao:             Inicializa os registradores para funcionamento do PWM */
/*                      entre 5 e 20Hz                                        */
/*                                                                            */
/* Parametros de entrada:    n/a                                              */
/*                                                                            */
/* Parametros de saida:      n/a                                              */
/* ************************************************************************** */
void PWM_init(void);


/* ************************************************************************** */
/* Nome do metodo:         PWM_clearCounter                                   */
/* Descricao:             Reseta o contador TPM1_CNT para nao haver overflow  */
/*                                                                            */
/* Parametros de entrada:    n/a                                              */
/*                                                                            */
/* Parametros de saida:      n/a                                              */
/* ************************************************************************** */
void PWM_clearCounter(void);

/* ************************************************************************** */
/* Nome do metodo:         coolerfan_init                                     */
/* Descricao:             Configura a liberacao do sinal PWM no pino PTA13    */
/*                                                                            */
/* Parametros de entrada:    n/a                                              */
/*                                                                            */
/* Parametros de saida:      n/a                                              */
/* ************************************************************************** */
void coolerfan_init(void);


/* ************************************************************************** */
/* Nome do metodo:         heater_init                                        */
/* Descricao:             Configura a liberacao do sinal PWM no pino PTA12    */
/*                                                                            */
/* Parametros de entrada:    n/a                                              */
/*                                                                            */
/* Parametros de saida:      n/a                                              */
/* ************************************************************************** */
void heater_init(void);

/* ************************************************************************** */
/* Nome do metodo:         coolerfan_PWMDuty                                  */
/* Descricao:             Configura o Duty Cycle do PWM para o cooler         */
/*                                                                            */
/* Parametros de entrada:   fCoolerDuty -> valor entre 0 e 1 indicando o Duty Cycle */
/*                                                                            */
```

```
67  /* Parametros de saida:     n/a                                              */
68  /* ******************************************************************************** */
69  void coolerfan_PWMDuty(float fCoolerDuty);
70
71  /* ******************************************************************************** */
72  /* Nome do metodo:          heater_PWMDuty                                   */
73  /* Descricao:               Configura o Duty Cycle do PWM para o cooler      */
74  /*                                                                           */
75  /* Parametros de entrada:   fHeaterDuty -> valor entre 0 e 1 indicando o Duty Cycle */
76  /*                                                                           */
77  /* Parametros de saida:     n/a                                              */
78  /* ******************************************************************************** */
79  void heater_PWMDuty(float fHeaterDuty);
80
81  #endif /* SOURCES_COOLER_HEATER_ */
```

```
/* ********************************************************************** */
/*                                                                        */
/*   Nome do arquivo:       board.h                                       */
/*                                                                        */
/*   Descrição:             Arquivo header contendo as definicoes de      */
/*                       pinos e mascaras utilizadas durante o            */
/*                       projeto                                          */
/*                                                                        */
/*   Autores:              Gustavo Lino e Giácomo Dollevedo               */
/*   Criado em:             31/03/2020                                    */
/*   Ultima revisão em:     03/06/2020                                    */
/* ********************************************************************** */


#ifndef SOURCES_BOARD_H_
#define SOURCES_BOARD_H_

/* system includes */
#include <MKL25Z4.h>

/* FDRMKL25Z RGB LED pins defintions */
#define RED_LED_PORT_BASE_PNT    PORTB /* peripheral port base pointer */
#define RED_LED_GPIO_BASE_PNT    PTB   /* peripheral gpio base pointer */
#define RED_LED_PIN              (uint32_t) 18u
#define GREEN_LED_PORT_BASE_PNT  PORTB /* peripheral port base pointer */
#define GREEN_LED_GPIO_BASE_PNT  PTB   /* peripheral gpio base pointer */
#define GREEN_LED_PIN            (uint32_t) 19u
#define BLUE_LED_PORT_BASE_PNT   PORTD /* peripheral port base pointer */
#define BLUE_LED_GPIO_BASE_PNT   PTD   /* peripheral gpio base pointer */
#define BLUE_LED_PIN             (uint32_t) 1u




/*Mascaras de ativação de pinos de portas como saída ou entrada*/

#define uiPin0MaskEnable      0x01
#define uiPin0MaskDisable     0xFFFFFFFE
#define uiPin1MaskEnable      0x02
#define uiPin1MaskDisable     0xFFFFFFFD
#define uiPin2MaskEnable      0x04
#define uiPin2MaskDisable     0xFFFFFFFB
#define uiPin3MaskEnable      0x08
#define uiPin3MaskDisable     0xFFFFFFF7

#define uiPin4MaskEnable      0x010
#define uiPin4MaskDisable     0xFFFFFFEF
#define uiPin5MaskEnable      0x020
#define uiPin5MaskDisable     0xFFFFFFDF
#define uiPin6MaskEnable      0x040
#define uiPin6MaskDisable     0xFFFFFFBF
#define uiPin7MaskEnable      0x080
#define uiPin7MaskDisable     0xFFFFFF7F

#define uiPin8MaskEnable      0x100
#define uiPin8MaskDisable     0xFFFFFEFF
#define uiPin9MaskEnable      0x200
#define uiPin9MaskDisable     0xFFFFFDFF


#define uiSetPinAsGPIO        0x100
#define uiSetClockPort        0x0200

/*CLOCK PORT ENABLE*/
#define PORTA_CLOCK_GATE    0x0200
```

```
67  #define PORTB_CLOCK_GATE       0x0400
68  #define PORTC_CLOCK_GATE       0x0800
69  #define PORTD_CLOCK_GATE       0x1000
70  #define PORTE_CLOCK_GATE       0x2000
71
72  #define MUX_ALT3    0x300
73  #define MUX_ALT4    0x400
74
75
76  /* Configuração dos set-up para utilziar o LCD
77   *
78   * Quando LCD_RS = LCD_RS_HIGH => Registrador de dados é selecionados
79   * Quando (LCD_RS = LCD_RS_LOW => Registrador de instruções é selecionado.
80   */
81
82
83  #define LCD_GPIO_BASE_PNT       PTC
84
85  #define LCD_RS_PIN              8U
86  #define LCD_RS_DIR              kGpioDigitalOutput
87  #define LCD_RS_ALT              kPortMuxAsGpio
88
89  #define LCD_ENABLE_DIR          kGipioDigitalOutput
90  #define LCD_ENABLE_ALT          kPortMuxAsGpio
91
92  #define LCD_RS_HIGH             0x0100
93  #define LCD_RS_DATA             LCD_RS_HIGH
94
95  #define LCD_RS_LOW              0x0000
96  #define LCD_RS_CMD              LCD_RS_LOW
97
98  #define LCD_RS_WAITING    0xFFFFEFF
99
100 #define LCD_ENABLED             uiPin9MaskEnable
101 #define LCD_DISABLED            uiPin9MaskDisable
102
103
104 #define LCD_DATA_DIR            kGpioDigitalOutput
105 #define LCD_DATA_ALT            kPortMuxAsGpio
106
107 #define LCD_DATA_DB0_PIN        0U
108 #define LCD_DATA_DB1_PIN        1U
109 #define LCD_DATA_DB2_PIN        2U
110 #define LCD_DATA_DB3_PIN        3U
111 #define LCD_DATA_DB4_PIN        4U
112 #define LCD_DATA_DB5_PIN        5U
113 #define LCD_DATA_DB6_PIN        6U
114 #define LCD_DATA_DB7_PIN        7U
115
116 /* Configuracao dos setups para utilizar o D7S*/
117
118 #define D7S_GPIO_CONFIG    0x3CFF
119
120 /* formato letras display 7 seg */
121 /* DP   g   f   e   d   c   b   a*/
122
123 #define DISP_0      0x003F
124 #define DISP_1      0x0006
125 #define DISP_2      0x005B
126 #define DISP_3      0x004F
127 #define DISP_4      0x0066
128 #define DISP_5      0x006D
129 #define DISP_6      0x007D
130 #define DISP_7      0x0007
131 #define DISP_8      0x007F
132 #define DISP_9      0x006F
133 #define DISP_A      0x0077
```

```
#define DISP_B      0x007C
#define DISP_C      0x0039
#define DISP_D      0x005E
#define DISP_E      0x0079
#define DISP_F      0x0071
#define DISP_DP     0x0080
#define DISP_CLEAR  0xFFFFFF00
#define DISP_ALL    0x00FF

/* Final das definições do LCD para a placa */

/* Configuracao dos setups para utilizar o Timer/PWM*/

#define TPM1_CLOCK_GATE   0x2000000

#define MCGIRCLK_SELECT   0x3000000


#define PWM_EVERY_CLOCK   0x08



/* Final das definições do LCD para a placa */

/*Configuração dos set-ups para utilziar o Tacometro*/
#define SET_LTPMR0     0X01
#define TPM_EXTERNAL_CLOCK  0x10
#define TPM_MAX_VALUE_COUNT  0x0FFFF
#define TPM0CLKSEL_AS_CLKIN1 0x01000000

/*              TEMPERATURE SENSOR DIODE DEFINITIONS          */

#define  THERMOMETER_PORT_BASE_PNT  PORTE                    /* peripheral port base pointer */
#define  THERMOMETER_GPIO_BASE_PNT  PTE                     /* peripheral gpio base pointer */
#define  THERMOMETER_PIN          21U                    /* thermometer pin */
#define  THERMOMETER_DIR          (GPIO_INPUT << TERMOMETER_PIN)
#define  THERMOMETER_ALT          0x00u

/*              END OF TEMPERATURE SENSOR DIODE DEFINITIONS    */


/* REVISAO PWM - AULA 12 */

#define DISABLE_COUNTER   0xFFFFFFE7
#define PWM_UP_COUNTING   0xFFFFFFDF
#define EDGE_ALIGNED_HIGH_TRUE  0xFFFFFFAB
#define CLEAR_16    0xFFFF0000

#define CLOCK_DIVIDER_1   0x1F8
#define CLOCK_DIVIDER_2   0x1F9
#define CLOCK_DIVIDER_4   0x1FA
#define CLOCK_DIVIDER_8   0x1FB
#define CLOCK_DIVIDER_16   0x1FC
#define CLOCK_DIVIDER_32  0x1FD
#define CLOCK_DIVIDER_64  0x1FE
#define CLOCK_DIVIDER_128   0x1FF

/* REVISAO TACOMETRO - AULA 14 */

#define TPM0_CLOCK_GATE   0x800000
#define TPM0CLKSEL_AS_CLKIN0  0xFEFFFFFF


#endif /* SOURCES_BOARD_H_ */
```

```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;
            }
    }
```

```c
/* ********************************************************************************** */
/*                                                                                    */
/*   Nome do arquivo:        communicationStateMachine.h                              */
/*                                                                                    */
/*   Descricao:              Arquivo contendo a bilioteca das 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:      25/07/2020                                               */
/* ********************************************************************************** */


#ifndef _COMMUNICATIONSTATEMACHINE_H
#define _COMMUNICATIONSTATEMACHINE_H




/* ********************************************************************************** */
/* 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);


/* ********************************************************************************** */
/* Nome do metodo:          setParam                                                  */
/* Descricao:               Define valores de controle/usabilidade necessarios para   */
/*                          garantir a interface e funcionamento adequado do uC:      */
/*       Temperatura MÃĄxima                                                          */
/*       Disponibilidade dos botoÃľes                                                */
/*                                                                                    */
/* Parametros de entrada:   ucParam  -> Parametro que sera alterado                   */
/*        *ucValue ->  Array com valores de alteracao                                 */
/*                                                                                    */
/* Parametros de saida:     n/a                                                       */
/* ********************************************************************************** */
void setParam(unsigned char ucParam, unsigned char* ucValue);

/* ********************************************************************************** */
/* 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);



#endif /* _COMMUNICATIONSTATEMACHINE_H */
```

```c
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:       display7seg.c                                     */
/*                                                                            */
/*   Descricao:             Arquivo contendo a implementacao                  */
/*                       das funcoes de interface do microcontrolador         */
/*                       com o display de 7 segmentos do kit                  */
/*                                                                            */
/*   Autores:              Gustavo Lino e Giacomo Dollevedo                   */
/*   Criado em:             13/04/2020                                        */
/*   Ultima revisao em:     31/07/2020                                        */
/* ************************************************************************** */

/* Correções implementadas:

Pinos foram definidos no arquivo display7seg.h e não no board.h, resolvido declarando
as constantes no arquivo board.h;
Função tc_installLptmr0 não foi chamada, resolvido implementado dentro da board_init na main;
Período em microsegundos e não em milisegundos, resolvido adequando a constante de tempo
para 400;*/

#include "board.h"
#include "display7seg.h"
#include "lptmr.h"

//Variaveis para controle dos displays









/* ************************************************************************** */
/* Nome do metodo:         display7Seg_init                                   */
/* Descricao:              Inicializa os registradores para funcionamento do D7S   */
/*                                                                            */
/* Parametros de entrada:    n/a                                              */
/*                                                                            */
/* Parametros de saida:      n/a                                              */
/* ************************************************************************** */
void display7seg_init(void){


    /* Liberando o Clock para porta C*/
    SIM_SCGC5 |= PORTC_CLOCK_GATE;

    /* Declarando os pinos como GPIO */
    PORTC_PCR0 |= uiSetPinAsGPIO;   //Segmento A
    PORTC_PCR1 |= uiSetPinAsGPIO;   //Segmento B
    PORTC_PCR2 |= uiSetPinAsGPIO;   //Segmento C
    PORTC_PCR3 |= uiSetPinAsGPIO;   //Segmento D
    PORTC_PCR4 |= uiSetPinAsGPIO;   //Segmento E
    PORTC_PCR5 |= uiSetPinAsGPIO;   //Segmento F
    PORTC_PCR6 |= uiSetPinAsGPIO;   //Segmento G
    PORTC_PCR7 |= uiSetPinAsGPIO;   //Segmento DP

    PORTC_PCR13 |= uiSetPinAsGPIO;  //Display 1
    PORTC_PCR11 |= uiSetPinAsGPIO;  //Display 2
    PORTC_PCR12 |= uiSetPinAsGPIO;  //Display 3
    PORTC_PCR10 |= uiSetPinAsGPIO;  //Display 4

    /* Declarando os pinos como Saida*/
    GPIOC_PDDR |= D7S_GPIO_CONFIG;

    /*Incializa o temporizador de interrupção*/

```

```
65
66  }
67
68  /* ************************************************************************** */
69  /* Nome do metodo:         display7seg_writeChar          */
70  /* Descricao:              Escreve uma letra em um D7S                        */
71  /*                                                                            */
72  /* Parametros de entrada:   ucDisplay -> indica o D7S no qual sera escrito (1 a 4)  */
73  /*                          ucValue -> valor indicando a letra que sera escrita  */
74  /*                                                                            */
75  /* Parametros de saida:     n/a                                               */
76  /* ************************************************************************** */
77  void display7seg_writeChar(unsigned char ucDisplay, unsigned char ucChar){
78
79   selectDisp(ucDisplay);
80   GPIOC_PDOR &= DISP_CLEAR;
81
82   if(97 <= ucChar && 102 >= ucChar){
83    switch(ucChar){
84    case 'a':
85     GPIOC_PDOR |= DISP_A;
86     break;
87    case 'b':
88     GPIOC_PDOR |= DISP_B;
89     break;
90    case 'c':
91     GPIOC_PDOR |= DISP_C;
92     break;
93    case 'd':
94     GPIOC_PDOR |= DISP_D;
95     break;
96    case 'f':
97     GPIOC_PDOR |= DISP_F;
98     break;
99    }
100  }
101 }
102
103 /* ************************************************************************** */
104 /* Nome do metodo:         display7seg_writeSymbol                            */
105 /* Descricao:              Escreve um caracter em um D7S                      */
106 /*                                                                            */
107 /* Parametros de entrada:   ucDisplay -> indica o D7S no qual sera escrito (1 a 4)  */
108 /*                          ucValue -> valor indicando o caracter que sera escrito  */
109 /*                                                                            */
110 /* Parametros de saida:     n/a                                               */
111 /* ************************************************************************** */
112 void display7seg_writeSymbol(unsigned char ucDisplay,  unsigned char ucValue){
113
114    selectDisp(ucDisplay);
115
116    /*CLEAR no display*/
117    /* A mascara mantem o estado dos outros pinos */
118
119    GPIOC_PDOR &= DISP_CLEAR;
120
121    if(ucValue < 20){
122       switch(ucValue%10){
123          case 0:
124             GPIOC_PDOR |= DISP_0;
125             break;
126
127          case 1:
128             GPIOC_PDOR |= DISP_1;
129             break;
130
131
```

```
132             case 2:
133                 GPIOC_PDOR |= DISP_2;
134                 break;
135 
136             case 3:
137                 GPIOC_PDOR |= DISP_3;
138                 break;
139 
140             case 4:
141                 GPIOC_PDOR |= DISP_4;
142                 break;
143 
144             case 5:
145                 GPIOC_PDOR |= DISP_5;
146                 break;
147 
148             case 6:
149                 GPIOC_PDOR |= DISP_6;
150                 break;
151 
152             case 7:
153                 GPIOC_PDOR |= DISP_7;
154                 break;
155 
156             case 8:
157                 GPIOC_PDOR |= DISP_8;
158                 break;
159 
160             case 9:
161                 GPIOC_PDOR |= DISP_9;
162                 break;
163         }
164 
165     }
166 
167     /* Acendendo o ponto decimal */
168     if(ucValue >= 10 && ucValue <= 20){
169         GPIOC_PDOR |= DISP_DP;
170 
171     }
172 
173     /* Caso CLEAR do display */
174     else if(ucValue == 21){
175         GPIOC_PDOR &= DISP_CLEAR;
176     }
177 
178     /* Caso acenda todos os segmentos do display */
179     else if(ucValue == 22){
180         GPIOC_PDOR |= DISP_ALL;
181     }
182 }
183 
184 /* ************************************************************************** */
185 /* Nome do metodo:         selectDisp                                         */
186 /* Descricao:              Seleciona o D7S que sera escrito                   */
187 /*                                                                            */
188 /* Parametros de entrada:   ucDisplay -> indica o D7S no qual sera escrito (1 a 4)  */
189 /*                                                                            */
190 /* Parametros de saida:     n/a                                               */
191 /* ************************************************************************** */
192 void selectDisp(unsigned char ucDisplay){
193 
194     /* CLEAR, zerando pinos de 10 a 13 (1111 1111 1111 1111 1111 0000 1111 1111) */
195     /* A mascara mantem o estado dos outros pinos */
196     GPIOC_PDOR &= 0xFFFFF0FF;
197 
198 
```

```
199     switch(ucDisplay){
200         /* Display 1, pino 13*/
201         case 1:
202             GPIOC_PDOR |= 0x000C;
203             break;
204 
205         /* Display 2, pino 11*/
206         case 2:
207             GPIOC_PDOR |= 0x000A;
208             break;
209 
210         /* Display 3, pino 12*/
211         case 3:
212             GPIOC_PDOR |= 0x000B;
213 
214             break;
215 
216         /* Display 4, pino 10*/
217         case 4:
218             GPIOC_PDOR |= 0x0009;
219             break;
220 
221         default:
222             break;
223     }
224 
225 
226 }
```

```
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:       display7seg.h                                     */
/*                                                                            */
/*   Descricao:             Arquivo Header contendo a declaracao              */
/*                       das funcoes de interface do microcontrolador         */
/*                       com o display de 7 segmentos do kit                  */
/*                                                                            */
/*   Autores:               Gustavo Lino e Giacomo Dollevedo                  */
/*   Criado em:             13/04/2020                                        */
/*   Ultima revisao em:     13/04/2020                                        */
/* ************************************************************************** */

#ifndef SOURCES_D7S_
#define SOURCES_D7S_


/* ************************************************************************** */
/* Nome do metodo:        display7Seg_init                                    */
/* Descricao:             Inicializa os registradores para funcionamento do D7S   */
/*                                                                            */
/* Parametros de entrada:    n/a                                              */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void display7seg_init(void);

/* ************************************************************************** */
/* Nome do metodo:        display7seg_writeChar         */
/* Descricao:             Escreve uma letra em um D7S                         */
/*                                                                            */
/* Parametros de entrada:   ucDisplay -> indica o D7S no qual sera escrito (1 a 4)  */
/*                       ucValue -> valor indicando a letra que sera escrita  */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void display7seg_writeChar(unsigned char ucDisplay, unsigned char ucChar);


/* ************************************************************************** */
/* Nome do metodo:        display7seg_writeSymbol                             */
/* Descricao:             Escreve um caracter em um D7S                       */
/*                                                                            */
/* Parametros de entrada:   ucDisplay -> indica o D7S no qual sera escrito (1 a 4)  */
/*                       ucValue -> valor indicando o caracter que sera escrito  */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void display7seg_writeSymbol(unsigned char ucDisplay,  unsigned char ucValue);


/* ************************************************************************** */
/* Nome do metodo:        selectDisp                                          */
/* Descricao:             Seleciona o D7S que sera escrito                    */
/*                                                                            */
/* Parametros de entrada:   ucDisplay -> indica o D7S no qual sera escrito (1 a 4)  */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void selectDisp(unsigned char ucDisplay);



#endif /* SOURCES_D7S_  */
```

```c
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:       display7Temp.c                                    */
/*                                                                            */
/*   Descricao:             Funcoes para operar o D7S uteis no contexto do projeto  */
/*       do controlador de temperatura        */
/*                                                                            */
/*   Autores:               Gustavo Lino e Giacomo A. Dollevedo                */
/*   Criado em:             29/07/2020                                        */
/*   Ultima revisao em:     31/07/2020                                        */
/* ************************************************************************** */

/* Incluindo bibliotecas */
#include "display7seg.h"
#include "variaveis_globais.h"

/* ************************************************************************** */
/* Nome do metodo:          display7Temp_init                                 */
/* Descricao:               Inicializa os registradores para funcionamento do D7S   */
/*                                                                            */
/* Parametros de entrada:    n/a                                              */
/*                                                                            */
/* Parametros de saida:      n/a                                              */
/* ************************************************************************** */
void display7Temp_init(void){

 display7seg_init();

}


/* ************************************************************************** */
/* Nome do metodo:          attDisp7Temp                                      */
/* Descricao:               Atualiza o D7S com a temperatura atual da resistencia   */
/*                                                                            */
/* Parametros de entrada:    n/a                                              */
/*                                                                            */
/* Parametros de saida:      n/a                                              */
/* ************************************************************************** */
void attDisp7Temp(void){

    display7seg_writeSymbol(1, ucDezTempAtual);
    display7seg_writeSymbol(2, ucUnTempAtual);
    display7seg_writeSymbol(3, 20);
    display7seg_writeChar(4, 'c');

}
```

```c
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:       display7Temp.c                                    */
/*                                                                            */
/*   Descricao:             Declaracao das funcoes para operar o D7S uteis no */
/*                        contexto do projeto do controlador de temperatura   */
/*                                                                            */
/*   Autores:               Gustavo Lino e Giacomo A. Dollevedo               */
/*   Criado em:             29/07/2020                                        */
/*   Ultima revisao em:     29/07/2020                                        */
/* ************************************************************************** */

#ifndef SOURCES_D7TEMP_H_
#define SOURCES_D7TEMP_H_

/* ************************************************************************** */
/* Nome do metodo:          display7Temp_init                                 */
/* Descricao:               Inicializa os registradores para funcionamento do D7S */
/*                                                                            */
/* Parametros de entrada:    n/a                                              */
/*                                                                            */
/* Parametros de saida:      n/a                                              */
/* ************************************************************************** */
void display7Temp_init(void);


/* ************************************************************************** */
/* Nome do metodo:          attDisp7Temp                                      */
/* Descricao:               Atualiza o D7S com a temperatura atual da resistencia */
/*                                                                            */
/* Parametros de entrada:    n/a                                              */
/*                                                                            */
/* Parametros de saida:      n/a                                              */
/* ************************************************************************** */
void attDisp7Temp(void);

#endif /*SOURCES_D7TEMP_H_*/
```

```c
/*
 * Copyright (c) 2013 - 2014, Freescale Semiconductor, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright notice, this
 *   list of conditions and the following disclaimer in the documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of Freescale Semiconductor, Inc. nor the names of its
 *   contributors may be used to endorse or promote products derived from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include "fsl_device_registers.h"
#include "fsl_debug_console.h"
#if defined(UART_INSTANCE_COUNT)
#include "fsl_uart_hal.h"
#endif
#if defined(LPUART_INSTANCE_COUNT)
#include "fsl_lpuart_hal.h"
#endif
#if defined(UART0_INSTANCE_COUNT)
#include "fsl_lpsci_hal.h"
#endif
#include "fsl_clock_manager.h"
#include "fsl_os_abstraction.h"
#include "print_scan.h"

#if (defined(USB_INSTANCE_COUNT) &&  (defined(BOARD_USE_VIRTUALCOM)))
  #include "usb_device_config.h"
  #include "usb.h"
  #include "usb_device_stack_interface.h"
  #include "usb_descriptor.h"
  #include "virtual_com.h"
#endif

extern uint32_t g_app_handle;
#if __ICCARM__
#include <yfuns.h>
#endif

static int debug_putc(int ch, void* stream);

/*************************************************************************
 * Definitions
 *************************************************************************/
```

```
64
65  /*! @brief Operation functions definiations for debug console. */
66  typedef struct DebugConsoleOperationFunctions {
67      union {
68          void (* Send)(void *base, const uint8_t *buf, uint32_t count);
69  #if defined(UART_INSTANCE_COUNT)
70          void (* UART_Send)(UART_Type *base, const uint8_t *buf, uint32_t count);
71  #endif
72  #if defined(LPUART_INSTANCE_COUNT)
73          void (* LPUART_Send)(LPUART_Type* base, const uint8_t *buf, uint32_t count);
74  #endif
75  #if defined(UART0_INSTANCE_COUNT)
76          void (* UART0_Send)(UART0_Type* base, const uint8_t *buf, uint32_t count);
77  #endif
78  #if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))
79          void (* USB_Send)(uint32_t base, const uint8_t *buf, uint32_t count);
80  #endif
81      } tx_union;
82      union{
83          void (* Receive)(void *base, uint8_t *buf, uint32_t count);
84  #if defined(UART_INSTANCE_COUNT)
85          uart_status_t (* UART_Receive)(UART_Type *base, uint8_t *buf, uint32_t count);
86  #endif
87  #if defined(LPUART_INSTANCE_COUNT)
88          lpuart_status_t (* LPUART_Receive)(LPUART_Type* base, uint8_t *buf, uint32_t count);
89  #endif
90  #if defined(UART0_INSTANCE_COUNT)
91          lpsci_status_t (* UART0_Receive)(UART0_Type* base, uint8_t *buf, uint32_t count);
92  #endif
93  #if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))
94          usb_status_t (* USB_Receive)(uint32_t base, uint8_t *buf, uint32_t count);
95  #endif
96
97      } rx_union;
98  } debug_console_ops_t;
99
100 /*! @brief State structure storing debug console. */
101 typedef struct DebugConsoleState {
102     debug_console_device_type_t type;/*<! Indicator telling whether the debug console is inited. */
103     uint8_t instance;              /*<! Instance number indicator. */
104     void*   base;                 /*<! Base of the IP register. */
105     debug_console_ops_t ops;       /*<! Operation function pointers for debug uart operations. */
106 } debug_console_state_t;
107
108 /*******************************************************************************
109  * Variables
110  ******************************************************************************/
111 /*! @brief Debug UART state information.*/
112 static debug_console_state_t s_debugConsole;
113
114 /*******************************************************************************
115  * Code
116  ******************************************************************************/
117 /* See fsl_debug_console.h for documentation of this function.*/
118 debug_console_status_t DbgConsole_Init(
119         uint32_t uartInstance, uint32_t baudRate, debug_console_device_type_t device)
120 {
121     if (s_debugConsole.type != kDebugConsoleNone)
122     {
123         return kStatus_DEBUGCONSOLE_Failed;
124     }
125
126     /* Set debug console to initialized to avoid duplicated init operation.*/
127     s_debugConsole.type = device;
128     s_debugConsole.instance = uartInstance;
129
130
```

```
    /* Switch between different device. */
    switch (device)
    {
#if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))  /*&& defined()*/
        case kDebugConsoleUSBCDC:
        {
            VirtualCom_Init();
            s_debugConsole.base = (void*)g_app_handle;
            s_debugConsole.ops.tx_union.USB_Send = VirtualCom_SendDataBlocking;
            s_debugConsole.ops.rx_union.USB_Receive = VirtualCom_ReceiveDataBlocking;
        }
        break;
#endif
#if defined(UART_INSTANCE_COUNT)
        case kDebugConsoleUART:
          {
            UART_Type * g_Base[UART_INSTANCE_COUNT] = UART_BASE_PTRS;
            UART_Type * base = g_Base[uartInstance];
            uint32_t uartSourceClock;

            s_debugConsole.base = base;
            CLOCK_SYS_EnableUartClock(uartInstance);

            /* UART clock source is either system or bus clock depending on instance */
            uartSourceClock = CLOCK_SYS_GetUartFreq(uartInstance);

            /* Initialize UART baud rate, bit count, parity and stop bit. */
            UART_HAL_SetBaudRate(base, uartSourceClock, baudRate);
            UART_HAL_SetBitCountPerChar(base, kUart8BitsPerChar);
            UART_HAL_SetParityMode(base, kUartParityDisabled);
#if FSL_FEATURE_UART_HAS_STOP_BIT_CONFIG_SUPPORT
            UART_HAL_SetStopBitCount(base, kUartOneStopBit);
#endif

            /* Finally, enable the UART transmitter and receiver*/
            UART_HAL_EnableTransmitter(base);
            UART_HAL_EnableReceiver(base);

            /* Set the funciton pointer for send and receive for this kind of device. */
            s_debugConsole.ops.tx_union.UART_Send = UART_HAL_SendDataPolling;
            s_debugConsole.ops.rx_union.UART_Receive = UART_HAL_ReceiveDataPolling;
          }
          break;
#endif
#if defined(UART0_INSTANCE_COUNT)
        case kDebugConsoleLPSCI:
          {
            /* Declare config sturcuture to initialize a uart instance. */
            UART0_Type * g_Base[UART0_INSTANCE_COUNT] = UART0_BASE_PTRS;
            UART0_Type * base = g_Base[uartInstance];
            uint32_t uartSourceClock;

            s_debugConsole.base = base;
            CLOCK_SYS_EnableLpsciClock(uartInstance);

            uartSourceClock = CLOCK_SYS_GetLpsciFreq(uartInstance);

            /* Initialize LPSCI baud rate, bit count, parity and stop bit. */
            LPSCI_HAL_SetBaudRate(base, uartSourceClock, baudRate);
            LPSCI_HAL_SetBitCountPerChar(base, kLpsci8BitsPerChar);
            LPSCI_HAL_SetParityMode(base, kLpsciParityDisabled);
#if FSL_FEATURE_LPSCI_HAS_STOP_BIT_CONFIG_SUPPORT
            LPSCI_HAL_SetStopBitCount(base, kLpsciOneStopBit);
#endif

            /* Finally, enable the LPSCI transmitter and receiver*/
            LPSCI_HAL_EnableTransmitter(base);
```

```
198             LPSCI_HAL_EnableReceiver(base);
199
200             /* Set the funciton pointer for send and receive for this kind of device. */
201             s_debugConsole.ops.tx_union.UART0_Send = LPSCI_HAL_SendDataPolling;
202             s_debugConsole.ops.rx_union.UART0_Receive = LPSCI_HAL_ReceiveDataPolling;
203         }
204         break;
205 #endif
206 #if defined(LPUART_INSTANCE_COUNT)
207       case kDebugConsoleLPUART:
208         {
209             LPUART_Type* g_Base[LPUART_INSTANCE_COUNT] = LPUART_BASE_PTRS;
210             LPUART_Type* base = g_Base[uartInstance];
211             uint32_t lpuartSourceClock;
212
213             s_debugConsole.base = base;
214             CLOCK_SYS_EnableLpuartClock(uartInstance);
215
216             /* LPUART clock source is either system or bus clock depending on instance */
217             lpuartSourceClock = CLOCK_SYS_GetLpuartFreq(uartInstance);
218
219             /* initialize the parameters of the LPUART config structure with desired data */
220             LPUART_HAL_SetBaudRate(base, lpuartSourceClock, baudRate);
221             LPUART_HAL_SetBitCountPerChar(base, kLpuart8BitsPerChar);
222             LPUART_HAL_SetParityMode(base, kLpuartParityDisabled);
223             LPUART_HAL_SetStopBitCount(base, kLpuartOneStopBit);
224
225             /* finally, enable the LPUART transmitter and receiver */
226             LPUART_HAL_SetTransmitterCmd(base, true);
227             LPUART_HAL_SetReceiverCmd(base, true);
228
229             /* Set the funciton pointer for send and receive for this kind of device. */
230             s_debugConsole.ops.tx_union.LPUART_Send = LPUART_HAL_SendDataPolling;
231             s_debugConsole.ops.rx_union.LPUART_Receive = LPUART_HAL_ReceiveDataPolling;
232
233         }
234         break;
235 #endif
236       /* If new device is requried as the low level device for debug console,
237        * Add the case branch and add the preprocessor macro to judge whether
238        * this kind of device exist in this SOC. */
239       default:
240         /* Device identified is invalid, return invalid device error code. */
241         return kStatus_DEBUGCONSOLE_InvalidDevice;
242   }
243
244   /* Configure the s_debugConsole structure only when the inti operation is successful. */
245   s_debugConsole.instance = uartInstance;
246
247   return kStatus_DEBUGCONSOLE_Success;
248 }
249
250 /* See fsl_debug_console.h for documentation of this function.*/
251 debug_console_status_t DbgConsole_DeInit(void)
252 {
253   if (s_debugConsole.type == kDebugConsoleNone)
254   {
255     return kStatus_DEBUGCONSOLE_Success;
256   }
257
258   switch(s_debugConsole.type)
259   {
260 #if defined(UART_INSTANCE_COUNT)
261       case kDebugConsoleUART:
262         CLOCK_SYS_DisableUartClock(s_debugConsole.instance);
263         break;
264
```

```
#endif
#if defined(UART0_INSTANCE_COUNT)
        case kDebugConsoleLPSCI:
            CLOCK_SYS_DisableLpsciClock(s_debugConsole.instance);
            break;
#endif
#if defined(LPUART_INSTANCE_COUNT)
        case kDebugConsoleLPUART:
             CLOCK_SYS_DisableLpuartClock(s_debugConsole.instance);
            break;
#endif
        default:
            return kStatus_DEBUGCONSOLE_InvalidDevice;
    }

    s_debugConsole.type = kDebugConsoleNone;

    return kStatus_DEBUGCONSOLE_Success;
}

#if (defined(__KSDK_STDLIB__))
int _WRITE(int fd, const void *buf, size_t nbytes)
{
    if (buf == 0)
    {
        /* This means that we should flush internal buffers.  Since we*/
        /* don't we just return.  (Remember, "handle" == -1 means that all*/
        /* handles should be flushed.)*/
        return 0;
    }


    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }

    /* Send data.*/
    s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (uint8_t const *)buf, nbytes);
    return nbytes;

}

int _READ(int fd, void *buf, size_t nbytes)
{

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }

    /* Receive data.*/
    s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, buf, nbytes);
    return nbytes;
}
#elif __ICCARM__

#pragma weak __write
size_t __write(int handle, const unsigned char * buffer, size_t size)
{
    if (buffer == 0)
    {
        /* This means that we should flush internal buffers.  Since we*/
        /* don't we just return.  (Remember, "handle" == -1 means that all*/
```

```
        /* handles should be flushed.)*/
        return 0;
    }

    /* This function only writes to "standard out" and "standard err",*/
    /* for all other file handles it returns failure.*/
    if ((handle != _LLIO_STDOUT) && (handle != _LLIO_STDERR))
    {
        return _LLIO_ERROR;
    }

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return _LLIO_ERROR;
    }

    /* Send data.*/
    s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (uint8_t const *)buffer, size);
    return size;
}

#pragma weak __read
size_t __read(int handle, unsigned char * buffer, size_t size)
{
    /* This function only reads from "standard in", for all other file*/
    /* handles it returns failure.*/
    if (handle != _LLIO_STDIN)
    {
        return _LLIO_ERROR;
    }

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return _LLIO_ERROR;
    }

    /* Receive data.*/
    s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, buffer, size);

    return size;
}

#elif (defined(__GNUC__))
#pragma weak _write
int _write (int handle, char *buffer, int size)
{
    if (buffer == 0)
    {
        /* return -1 if error */
        return -1;
    }

    /* This function only writes to "standard out" and "standard err",*/
    /* for all other file handles it returns failure.*/
    if ((handle != 1) && (handle != 2))
    {
        return -1;
    }

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
```

```
399
400     /* Send data.*/
401     s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (uint8_t *)buffer, size);
402     return size;
403 }
404
405 #pragma weak _read
406 int _read(int handle, char *buffer, int size)
407 {
408     /* This function only reads from "standard in", for all other file*/
409     /* handles it returns failure.*/
410     if (handle != 0)
411     {
412         return -1;
413     }
414
415     /* Do nothing if the debug uart is not initialized.*/
416     if (s_debugConsole.type == kDebugConsoleNone)
417     {
418         return -1;
419     }
420
421     /* Receive data.*/
422     s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, (uint8_t *)buffer, size);
423     return size;
424 }
425 #elif defined(__CC_ARM) && !defined(MQX_STDIO)
426 struct __FILE
427 {
428     int handle;
429     /* Whatever you require here. If the only file you are using is */
430     /* standard output using printf() for debugging, no file handling */
431     /* is required. */
432 };
433
434 /* FILE is typedef in stdio.h. */
435 #pragma weak __stdout
436 FILE __stdout;
437 FILE __stdin;
438
439 #pragma weak fputc
440 int fputc(int ch, FILE *f)
441 {
442     /* Do nothing if the debug uart is not initialized.*/
443     if (s_debugConsole.type == kDebugConsoleNone)
444     {
445         return -1;
446     }
447
448     /* Send data.*/
449     s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (const uint8_t*)&ch, 1);
450     return 1;
451 }
452
453 #pragma weak fgetc
454 int fgetc(FILE *f)
455 {
456     uint8_t temp;
457     /* Do nothing if the debug uart is not initialized.*/
458     if (s_debugConsole.type == kDebugConsoleNone)
459     {
460         return -1;
461     }
462
463     /* Receive data.*/
464     s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, &temp, 1);
465
```

```
    return temp;
}

#endif

/*************Code for debug_printf/scanf/assert*******************************/
int debug_printf(const char  *fmt_s, ...)
{
    va_list  ap;
    int  result;
    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
    va_start(ap, fmt_s);
    result = _doprint(NULL, debug_putc, -1, (char *)fmt_s, ap);
    va_end(ap);

    return result;
}

static int debug_putc(int ch, void* stream)
{
    const unsigned char c = (unsigned char) ch;
    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
    s_debugConsole.ops.tx_union.Send(s_debugConsole.base, &c, 1);

    return 0;

}

int debug_putchar(int ch)
{
    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
    debug_putc(ch, NULL);

    return 1;
}

int debug_scanf(const char  *fmt_ptr, ...)
{
    char    temp_buf[IO_MAXLINE];
    va_list ap;
    uint32_t i;
    char result;

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
    va_start(ap, fmt_ptr);
    temp_buf[0] = '\0';

    for (i = 0; i < IO_MAXLINE; i++)
    {
        temp_buf[i] = result = debug_getchar();
```

```
533
534         if ((result == '\r') || (result == '\n'))
535         {
536             /* End of Line */
537             if (i == 0)
538             {
539                 i = (uint32_t)-1;
540             }
541             else
542             {
543                 break;
544             }
545         }
546
547         temp_buf[i + 1] = '\0';
548     }
549
550     result = scan_prv(temp_buf, (char *)fmt_ptr, ap);
551     va_end(ap);
552
553   return result;
554 }
555
556 int debug_getchar(void)
557 {
558     unsigned char c;
559
560     /* Do nothing if the debug uart is not initialized.*/
561     if (s_debugConsole.type == kDebugConsoleNone)
562     {
563         return -1;
564     }
565     s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, &c, 1);
566
567     return c;
568 }
    /*******************************************************************************
     * EOF
     ******************************************************************************/
```

```c
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:       lcd.c                                             */
/*                                                                            */
/*   Descricao:           Arquivo contendo as implementacoes das funcoes      */
/*                      necessarias para a interface do LCD com o kit         */
/*                                                                            */
/*   Autores:             Gustavo Lino e Giacomo Dollevedo                    */
/*   Criado em:             07/04/2020                                        */
/*   Ultima revisao em:     31/07/2020                                        */
/* ************************************************************************** */

/*Correções implementadas: Pino 3 da GPIO não setado resolvido setando  do Pino 3 como GPIO
na função lcd_initLcd;
No comando "GPIOC_PDOR |= LCD_RS_CMD;" poderia ser considerado valores anteriores, resolvido
zerando a saida da porta C responsável pelo pino RS ao entrar  na função lcd_write2Lcd
inserido na board.h a constante LCD_RS_WAINTG;
Não é alocado memória para a strings usada, resolvido dando malloc.
*/

#include "lcd.h"
#include "board.h"

/* Bibliotecas da linguagem */

#include <string.h>


/* line and columns */
#define LINE0       0U
#define LINE1       1U

#define COLUMN0     0U

#define L0C0_BASE   0x80 /* line 0, column 0 */
#define L1C0_BASE   0xC0 /* line 1, column 0 */
#define MAX_COLUMN  15U

/* ************************************************************************** */
/* Nome do metodo:         lcd_initLcd                                        */
/* Descricao:            Inicializa as funcoes do LCD                         */
/*                                                                            */
/* Parametros de entrada:   n/a                                               */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void lcd_initLcd(void)
{
    /* pins configured as outputs */

    /* un-gate port clock*/
    SIM_SCGC5 |= PORTC_CLOCK_GATE;


    /* set pin as gpio */
    PORTC_PCR0 |= uiSetPinAsGPIO;
    PORTC_PCR1 |= uiSetPinAsGPIO;
    PORTC_PCR2 |= uiSetPinAsGPIO;
    PORTC_PCR3 |= uiSetPinAsGPIO;
    PORTC_PCR4 |= uiSetPinAsGPIO;
    PORTC_PCR5 |= uiSetPinAsGPIO;
    PORTC_PCR6 |= uiSetPinAsGPIO;
    PORTC_PCR7 |= uiSetPinAsGPIO;
    PORTC_PCR8 |= uiSetPinAsGPIO;
    PORTC_PCR9 |= uiSetPinAsGPIO;

```

```
67
68      /* set pin as digital output */
69
70      GPIOC_PDDR |= uiPin0MaskEnable;
71      GPIOC_PDDR |= uiPin1MaskEnable;
72      GPIOC_PDDR |= uiPin2MaskEnable;
73      GPIOC_PDDR |= uiPin3MaskEnable;
74      GPIOC_PDDR |= uiPin4MaskEnable;
75      GPIOC_PDDR |= uiPin5MaskEnable;
76      GPIOC_PDDR |= uiPin6MaskEnable;
77      GPIOC_PDDR |= uiPin7MaskEnable;
78      GPIOC_PDDR |= uiPin8MaskEnable;
79      GPIOC_PDDR |= uiPin9MaskEnable;
80
81
82      // turn-on LCD, with no cursor and no blink
83      lcd_sendCommand(CMD_NO_CUR_NO_BLINK);
84
85      // init LCD
86      lcd_sendCommand(CMD_INIT_LCD);
87
88      // clear LCD
89      lcd_sendCommand(CMD_CLEAR);
90
91      // LCD with no cursor
92      lcd_sendCommand(CMD_NO_CURSOR);
93
94      // cursor shift to right
95      lcd_sendCommand(CMD_CURSOR2R);
96
97  }
98
99
100
101 /* ************************************************************************** */
102 /* Nome do metodo:          lcd_write2Lcd                                     */
103 /* Descricao:               Escreve caracter no LCD                           */
104 /*                                                                            */
105 /* Parametros de entrada:   ucBuffer    -> char do dado que sera enviado      */
106 /*                          cDataType   -> commando (LCD_RS_CMD) ou dado      */
107 /*                                         (LCD_RS_DATA)                      */
108 /* Parametros de saida:     n/a                                               */
109 /* ************************************************************************** */
110 void lcd_write2Lcd(unsigned char ucBuffer,  unsigned char cDataType)
111 {
112     //Vamos colocar apenas o pino RS como indefinido (em zero)
113     GPIOC_PDOR &= LCD_RS_WAITING;
114
115     /* writing data or command */
116     if(LCD_RS_CMD == cDataType)
117         /* will send a command */
118         GPIOC_PDOR |= LCD_RS_CMD;
119     else
120         /* will send data */
121         GPIOC_PDOR |= LCD_RS_DATA;
122
123     /*Zera as portas de dados que sera utilizada e insere o valor binario do caracter*/
124     GPIOC_PDOR &= 0xFFFFFF00;
125     GPIOC_PDOR |= ucBuffer;
126
127     /* enable, delay, disable LCD */
128     /* this generates a pulse in the enable pin */
129
130     GPIOA_PDOR |= LCD_ENABLED;
131     util_genDelay1ms();
132     GPIOA_PDOR &= LCD_DISABLED;
133     //util_genDelay1ms();
```

```
134     //util_genDelay1ms();
135 }
136
137
138
139 /* ********************************************************************** */
140 /* Nome do metodo:          lcd_writeData                                 */
141 /* Descricao:               Escreve um dado no LCD                        */
142 /*                                                                        */
143 /* Parametros de entrada:   Um unsigned char que serÃƒÂ¡ escrito           */
144 /*                                                                        */
145 /* Parametros de saida:     n/a                                           */
146 /* ********************************************************************** */
147 void lcd_writeData(unsigned char ucData)
148 {
149     /* just a relay to send data */
150     lcd_write2Lcd(ucData, LCD_RS_DATA);
151 }
152
153
154
155 /* ********************************************************************** */
156 /* Nome do metodo:          lcd_sendCommand                               */
157 /* Descricao:               Escreve um comando no LCD                     */
158 /*                                                                        */
159 /* Parametros de entrada:   Um unsigned char descrevendo o comando que sera feito */
160 /*                                                                        */
161 /* Parametros de saida:     n/a                                           */
162 /* ********************************************************************** */
163 void lcd_sendCommand(unsigned char ucCmd)
164 {
165     /* just a relay to send command */
166     lcd_write2Lcd(ucCmd, LCD_RS_CMD);
167 }
168
169
170
171 /* ********************************************************************** */
172 /* Nome do metodo:          lcd_setCursor                                 */
173 /* Descricao:               Move o cursor no LCD para uma posicao especifica */
174 /*                                                                        */
175 /* Parametros de entrada:   Dois unsigned char, contendo a linha (cLine) e coluna */
176 /*                          (cColumn) para onde o cursor sera movido no display */
177 /*                                                                        */
178 /* Parametros de saida:     n/a                                           */
179 /* ********************************************************************** */
180 void lcd_setCursor(unsigned char cLine, unsigned char cColumn)
181 {
182     char cCommand;
183
184     if(LINE0 == cLine)
185         /* line 0 */
186         cCommand = L0C0_BASE;
187     else
188         /* line 1 */
189         cCommand = L1C0_BASE;
190
191     /* maximum MAX_COLUMN columns */
192     cCommand += (cColumn & MAX_COLUMN);
193
194     // send the command to set the cursor
195     lcd_sendCommand(cCommand);
196 }
197
198
199 /* ********************************************************************** */
200 /* Nome do metodo:          lcd_dummyText                                 */
```

```
/* Descricao:              Escreve um texto padrao no LCD                    */
/*                                                                           */
/* Parametros de entrada:   n/a                                              */
/*                                                                           */
/* Parametros de saida:     n/a                                              */
/* ************************************************************************* */
void lcd_dummyText(void)
{
   // clear LCD
   lcd_sendCommand(CMD_CLEAR);

   // set the cursor line 0, column 1
   lcd_setCursor(LINE0,1);

   // send string
   lcd_writeString("*** ES670 ***");

   // set the cursor line 1, column 0
   lcd_setCursor(1,0);
   lcd_writeString("Prj Sis Embarcad");
}


/* ************************************************************************* */
/* Nome do metodo:         lcd_writeString                                   */
/* Descricao:              Escreve uma string no LCD                         */
/*                                                                           */
/* Parametros de entrada:   Um array dinamico de char, contendo a string que sera   */
/*                         escrita                                           */
/*                                                                           */
/* Parametros de saida:     n/a                                              */
/* ************************************************************************* */
void lcd_writeString(const char *cBuffer){
   while(*cBuffer){
      lcd_writeData(*cBuffer++);
   }
}

/* ************************************************************************* */
/* Nome do metodo:         lcd_writeText                                     */
/* Descricao:              Escreve um texto especifico em uma das duas linhas      */
/*                        do LCD                                             */
/*                                                                           */
/* Parametros de entrada:   Uma string contendo o texto a ser escrito e um inteiro  */
/*                         indicando a linha (0 ou 1) do LCD para escrita          */
/*                                                                           */
/* Parametros de saida:     n/a                                              */
/* ************************************************************************* */
void lcd_writeText(const char *cBuffer, int iLine)
{

   int ilen = strlen(cBuffer);
   char *cLine1, *cLine2;
   cLine1 = (char*)malloc(sizeof(char) * 16);
   cLine2 = (char*)malloc(sizeof(char) * 16);
   // clear LCD
   lcd_sendCommand(CMD_CLEAR);
   // identifica a linha desejada
   if(0 == iLine)
      lcd_setCursor(LINE0,1);
   else
      lcd_setCursor(LINE1,1);

   // send string


   if(ilen < 17){
```

```
268         lcd_writeString(cBuffer);
269     }
270 
271     else if(ilen < 37){
272         strncpy(cLine1, cBuffer, 16);
273         strcpy(cLine2, &cBuffer[17]);
274         lcd_writeString(cLine1);
275 
276         lcd_setCursor(LINE1,1);
277         lcd_writeString(cLine1);
278 
279     }
280 
281     else{
282         lcd_setCursor(LINE0,1);
283         lcd_writeString("Too Many Car");
284 
285     }
286 
287 
288 }
```

```
/* ********************************************************************************** */
/*                                                                                    */
/*   Nome do arquivo:       lcd.h                                                     */
/*                                                                                    */
/*   DescriÃƒÂ§ÃƒÂ£o:            Arquivo Header contendo a declaraÃƒÂ§ÃƒÂ£o                */
/*                        das funÃƒÂ§ÃƒÂµes de interface do microcontrolador          */
/*                        com o LCD do kit                                          */
/*                                                                                    */
/*   Autores:             Gustavo Lino e GiÃƒÂ¡como Dollevedo                        */
/*   Criado em:             07/04/2020                                              */
/*   Ultima revisÃƒÂ£o em:     09/04/2020                                             */
/* ********************************************************************************** */

#ifndef SOURCES_LCD_H_
#define SOURCES_LCD_H_

/* lcd basic commands list */
#define CMD_INIT_LCD        0x0F
#define CMD_CLEAR           0x01
#define CMD_NO_CURSOR       0x0C
#define CMD_CURSOR2R        0x06 /* cursor to right */
#define CMD_NO_CUR_NO_BLINK 0x38 /* no cursor, no blink */

#define LINE0       0U
#define LINE1       1U


/* ********************************************************************************** */
/* Nome do metodo:          lcd_initLcd                                               */
/* DescriÃƒÂ§ÃƒÂ£o:              Envia um comando ou dado para o LCD                   */
/*                                                                                    */
/* Parametros de entrada:   ucBuffer    -> char do dado que sera enviado             */
/*                        cDataType   -> commando (LCD_RS_CMD) ou dado                */
/*                                  (LCD_RS_DATA)                                     */
/*                                                                                    */
/* Parametros de saida:     n/a                                                       */
/* ********************************************************************************** */
void lcd_initLcd(void);


/* ********************************************************************************** */
/* Nome do metodo:          lcd_write2Lcd                                             */
/* DescriÃƒÂ§ÃƒÂ£o:              Inicializa as funcoes do LCD                          */
/*                                                                                    */
/* Parametros de entrada:   n/a                                                       */
/*                                                                                    */
/* Parametros de saida:     n/a                                                       */
/* ********************************************************************************** */
void lcd_write2Lcd(unsigned char ucBuffer,  unsigned char cDataType);


/* ********************************************************************************** */
/* Nome do metodo:          lcd_writeData                                             */
/* DescriÃƒÂ§ÃƒÂ£o:              Escreve um dado no LCD                                */
/*                                                                                    */
/* Parametros de entrada:   Um unsigned char que serÃƒÂ¡ escrito                       */
/*                                                                                    */
/* Parametros de saida:     n/a                                                       */
/* ********************************************************************************** */
void lcd_writeData(unsigned char ucData);


/* ********************************************************************************** */
/* Nome do metodo:          lcd_sendCommand                                           */
/* DescriÃƒÂ§ÃƒÂ£o:              Escreve um comando no LCD                             */
/*                                                                                    */
```

```
67  /* Parametros de entrada:   Um unsigned char descrevendo o comando que serÃƒÂ¡ feito  */
68  /*                                                                                   */
69  /* Parametros de saida:     n/a                                                      */
70  /* ********************************************************************************* */
71  void lcd_sendCommand(unsigned char ucCmd);
72
73
74  /* ********************************************************************************* */
75  /* Nome do metodo:          lcd_writeString                                          */
76  /* DescriÃƒÂ§ÃƒÂ£o:              Escreve uma string no LCD                               */
77  /*                                                                                   */
78  /* Parametros de entrada:   Um array dinamico de char, contendo a string que sera   */
79  /*                          escrita                                                  */
80  /*                                                                                   */
81  /* Parametros de saida:     n/a                                                      */
82  /* ********************************************************************************* */
83  void lcd_writeString(const char *cBuffer);
84
85
86  /* ********************************************************************************* */
87  /* Nome do metodo:          lcd_setCursor                                            */
88  /* DescriÃƒÂ§ÃƒÂ£o:              Move o cursor no LCD para uma posicao especifica        */
89  /*                                                                                   */
90  /* Parametros de entrada:   Dois unsigned char, contendo a linha (cLine) e coluna   */
91  /*                          (cColumn) para onde o cursor sera movido no display      */
92  /*                                                                                   */
93  /* Parametros de saida:     n/a                                                      */
94  /* ********************************************************************************* */
95  void lcd_setCursor(unsigned char cLine, unsigned char cColumn);
96
97
98  /* ********************************************************************************* */
99  /* Nome do metodo:          lcd_dummyText                                            */
100 /* DescriÃƒÂ§ÃƒÂ£o:              Escreve um texto padrÃƒÂ£o no LCD                        */
101 /*                                                                                   */
102 /* Parametros de entrada:   n/a                                                      */
103 /*                                                                                   */
104 /* Parametros de saida:     n/a                                                      */
105 /* ********************************************************************************* */
106 void lcd_dummyText(void);
107
108
109 /* ********************************************************************************* */
110 /* Nome do metodo:          lcd_writeText                                            */
111 /* DescriÃƒÂ§ÃƒÂ£o:              Escreve um texto especÃƒÂfico em uma das duas linhas      */
112 /*                          do LCD                                                   */
113 /*                                                                                   */
114 /* Parametros de entrada:   Uma string contendo o texto a ser escrito e um inteiro  */
115 /*                          indicando a linha (0 ou 1) do LCD para escrita           */
116 /*                                                                                   */
117 /* Parametros de saida:     n/a                                                      */
118 /* ********************************************************************************* */
119 void lcd_writeText(const char *cBuffer, int iLine);
120
121 #endif /* SOURCES_LCD_H_ */
```

```c
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:       lcdTemp.c                                         */
/*                                                                            */
/*   Descricao:             Funcoes para operar o LCD uteis no contexto do projeto  */
/*       do controlador de temperatura        */
/*                                                                            */
/*   Autores:               Gustavo Lino e Giacomo A. Dollevedo               */
/*   Criado em:             28/07/2020                                        */
/*   Ultima revisao em:     31/07/2020                                        */
/* ************************************************************************** */

/* Incluindo bibliotecas */
#include "lcd.h"


/* ************************************************************************** */
/* Nome do metodo:          lcdTemp_init                                      */
/* Descricao:               Inicializa as funcoes do LCD                      */
/*                                                                            */
/* Parametros de entrada:   n/a                                               */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void lcdTemp_init(void){

 lcd_initLcd();

}


/* ************************************************************************** */
/* Nome do metodo:          showLCDdisp                                       */
/* Descricao:               Realiza a troca de mensagem no LCD de acordo com  */
/*                      estado                                                */
/*                                                                            */
/* Parametros de entrada:   unsigned char ucFrame -> Indica o frame que sera  */
/*                      mostrado                                              */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void showLCDdisp(unsigned char ucFrame){

   switch(ucFrame){
   case 0:
      /*Nada*/
      break;

   case 1:
      lcd_setCursor(LINE0,0);
      lcd_writeString("Configure a Temp");
      lcd_setCursor(LINE1,0);
      lcd_writeString("Temp Alvo: 00C");
      break;

    case 2:
      lcd_setCursor(LINE0,0);
      lcd_writeString("UART HABILITADO");
      lcd_setCursor(LINE1,0);
      lcd_writeString("Temp Alvo:   C");
      break;

    default:
       break;
   }
```

```
66
67  }
68
69
70  /* ************************************************************************** */
71  /* Nome do metodo:          attTempAlvo                                       */
72  /* Descricao:               Atualiza o display com a temperatura desejada     */
73  /*                                                                            */
74  /* Parametros de entrada:   unsigned char ucDezena -> Indica a dezena da temp.  */
75  /*                          alvo                                              */
76  /*       unsigned char ucUnidade -> Indica a unidade da temp.*/
77  /*      alvo            */
78  /*                                                                            */
79  /* Parametros de saida:     n/a                                               */
80  /* ************************************************************************** */
81  void attTempAlvo(unsigned char ucDezena, unsigned char ucUnidade){
82
83      lcd_setCursor(LINE1,11);
84          lcd_writeData(ucDezena+48);
85          lcd_setCursor(LINE1,12);
86          lcd_writeData(ucUnidade+48);
87
88  }
```

```c
/* ********************************************************************** */
/*                                                                        */
/*   Nome do arquivo:       lcdTemp.h                                     */
/*                                                                        */
/*   Descricao:             Arquivo Header contendo as declaracoes das funcoes      */
/*                       definidas em lcdTemp.c                           */
/*                                                                        */
/*   Autores:               Gustavo Lino e Giacomo A. Dollevedo           */
/*   Criado em:             28/07/2020                                    */
/*   Ultima revisao em:     28/07/2020                                    */
/* ********************************************************************** */

#ifndef SOURCES_LCDTEMP_H_
#define SOURCES_LCDTEMP_H_


/* ********************************************************************** */
/* Nome do metodo:          lcdTemp_init                                  */
/* Descricao:               Inicializa as funcoes do LCD                  */
/*                                                                        */
/* Parametros de entrada:   n/a                                           */
/*                                                                        */
/* Parametros de saida:     n/a                                           */
/* ********************************************************************** */
void lcdTemp_init(void);


/* ********************************************************************** */
/* Nome do metodo:          showLCDdisp                                   */
/* Descricao:               Realiza a troca de mensagem no LCD de acordo com    */
/*                       estado                                           */
/*                                                                        */
/* Parametros de entrada:   unsigned char ucFrame -> Indica o frame que sera    */
/*                       mostrado                                         */
/*                                                                        */
/* Parametros de saida:     n/a                                           */
/* ********************************************************************** */
void showLCDdisp(unsigned char ucFrame);

/* ********************************************************************** */
/* Nome do metodo:          attTempAlvo                                   */
/* Descricao:               Atualiza o display com a temperatura desejada       */
/*                                                                        */
/* Parametros de entrada:   unsigned char ucDezena -> Indica a dezena da temp.  */
/*                       alvo                                             */
/*       unsigned char ucUnidade -> Indica a unidade da temp.*/
/*      alvo            */
/*                                                                        */
/* Parametros de saida:     n/a                                           */
/* ********************************************************************** */
void attTempAlvo(unsigned char ucDezena, unsigned char ucUnidade);

#endif /*SOURCES_LCDTEMP_H_*/
```

```c
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:       ledSwi.c                                          */
/*                                                                            */
/*   Descrição:             Arquivo contendo as funcoes que lidam com         */
/*                       a atuacao do microcontrolador com os LEDs e          */
/*                       botoes do kit                                        */
/*                                                                            */
/*   Autores:              Gustavo Lino e Giácomo Dollevedo                   */
/*   Criado em:             31/03/2020                                        */
/*   Ultima revisão em:     31/07/2020                                        */
/* ************************************************************************** */


/*Adequações realizadas: Não deve ser considerada as variaveis tipo booleanas, para isso foram trocadas por tipo char
outra adequação implementada foi o testar se os parametros passados estão dentro do padrão esperado*/


#include "ledSwi.h"
#include "board.h"


/* ************************************************************************** */
/* Nome do metodo:         ledSwi_init                                        */
/* Descrição:              Inicializa os clocks e pinos necessarios para utilizar */
/*      a interface de botoes/leds do kit                                     */
/*                                                                            */
/* Parametros de entrada:   5 char (0 ou 1) que indica se o pino sera configurado   */
/*      como led ou como botao                                                */
/*                      0 -> botao;  1 -> led                                 */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
char cLedSwi1 = 0, cLedSwi2 = 0, cLedSwi3 = 0, cLedSwi4 = 0;
unsigned char ucError = 0;

void ledSwi_init(char led1, char led2, char led3, char led4) {



/* ativar o clock para a porta A*/
   SIM_SCGC5 |= uiSetClockPort;

/*Configura os pinos das portas para GPIO*/

   PORTA_PCR1 |= uiSetPinAsGPIO;
   PORTA_PCR2 |= uiSetPinAsGPIO;
   PORTA_PCR4 |= uiSetPinAsGPIO;
   PORTA_PCR5 |= uiSetPinAsGPIO;

 // testa se os parametros foram passado corretamente, se nao foram ativa todas as portas como led

 if((led1 != 0 && led1 != 1) && (led2 != 0 && led2 != 1) && (led3 != 0 && led3 != 1) && (led4 != 0 && led4 != 1)) {
 GPIOA_PDDR |= uiPin1MaskEnable;
      cLedSwi1 = 1;
      GPIOA_PDDR |= uiPin2MaskEnable;
      cLedSwi2 = 1;
      GPIOA_PDDR |= uiPin3MaskEnable;
      cLedSwi3 = 1;
      GPIOA_PDDR |= uiPin4MaskEnable;
      cLedSwi4 = 1;
 }

/*Define se os pinos serao entrada (chave) ou saida (led)*/

   if(led1 == 1){
```

```
64      GPIOA_PDDR |= uiPin1MaskEnable;
65      cLedSwi1 = 1;
66    }
67    else{
68      GPIOA_PDDR &= uiPin1MaskDisable;
69      cLedSwi1 = 0;
70    }
71
72    if(led2 == 1){
73      GPIOA_PDDR |= uiPin2MaskEnable;
74      cLedSwi2 = 1;
75    }
76    else{
77      GPIOA_PDDR &= uiPin2MaskDisable;
78      cLedSwi2 = 0;
79    }
80
81    if(led3== 1){
82      GPIOA_PDDR |= uiPin4MaskEnable;
83      cLedSwi3 = 1;
84    }
85    else{
86      GPIOA_PDDR &= uiPin4MaskDisable;
87      cLedSwi3 = 0;
88    }
89
90    if(led4 == 1){
91      GPIOA_PDDR |= uiPin5MaskEnable;
92      cLedSwi4 = 1;
93    }
94    else{
95      GPIOA_PDDR &= uiPin5MaskDisable;
96      cLedSwi4 = 0;
97    }
98
99  }
100
101
102 /* ************************************************************************** */
103 /* Nome do metodo:          readSwitch                                        */
104 /* Descrição:               Le o status de um switch para saber se o mesmo    */
105 /*                          está pressionado ou não                           */
106 /*                                                                            */
107 /* Parametros de entrada:   Um inteiro (0<n<5) que indica qual botão será lido */
108 /*                          inicializado como entrada (botao) ou saida (LED)  */
109 /*                          0 -> Leitura PTA1; 1 -> Leitura PTA2,             */
110 /*                          2 -> Leitura PTA4; 3 -> Leitura PTA5;             */
111 /*                                                                            */
112 /* Parametros de saida:     Um char indicando se o botao lido está sendo      */
113 /*                          pressionado (1), se não está ou se é inválido     */
114 /*                          (0)                                               */
115 /* ************************************************************************** */
116 char readSwitch(int n){
117 //testa se houve erro na passagem dos parametros
118  if(n > 5){
119   ucError = 1;
120  }
121
122  else{
123   ucError = 0;
124  }
125
126 // se não houver erro na entrada
127  if(0 == ucError){
128
129
130
```

```
131     switch(n){
132       case 1:
133         if(0 == cLedSwi1){
134           if (uiPin1MaskEnable == (GPIOA_PDIR & uiPin1MaskEnable)){
135             return 1;
136           }
137           else {
138             return 0;
139           }
140         }
141         else{
142           return 0;
143         }
144       break;
145 
146       case 2:
147         if(0 == cLedSwi2){
148           if (uiPin2MaskEnable == (GPIOA_PDIR & uiPin2MaskEnable)){
149             return 1;
150           }
151           else {
152             return 0;
153           }
154         }
155         else{
156           return 0;
157         }
158       break;
159 
160       case 3:
161         if(0 == cLedSwi3){
162           if (uiPin4MaskEnable == (GPIOA_PDIR & uiPin4MaskEnable)){
163             return 1;
164           }
165           else {
166             return 0;
167           }
168         }
169         else{
170           return 0;
171         }
172       break;
173 
174       case 4:
175         if(0 == cLedSwi4){
176           if (uiPin5MaskEnable == (GPIOA_PDIR & uiPin5MaskEnable)){
177             return 1;
178           }
179           else {
180             return 0;
181           }
182         }
183       else{
184             return 0;
185         }
186       break;
187       }
188     }
189   return 0;
190    }
191 
192 
193 
194 /* ************************************************************************** */
195 /* Nome do metodo:          writeLED                                          */
196 
197 
```

```
198 /* Descrição:              Liga ou desliga o LED selecionado conforme as          */
199 /*                        entradas                                               */
200 /*                                                                               */
201 /* Parametros de entrada:   Um inteiro (0<n<5) indicando sobre qual LED sera      */
202 /*                        efetuado o comando;                                    */
203 /*                        Um char (status) indicando se o LED sera               */
204 /*                        aceso (1) ou apagado (0)                               */
205 /*                                                                               */
206 /* Parametros de saida:     n/a                                                  */
207 /* ****************************************************************************** */
208 void writeLED(int n, char status){
209
210 //testa se houve erro na passagem dos parametros
211
212  if((status != 0 && status != 1) && (n < 5) ){
213   ucError = 1;
214  }
215
216  else{
217   ucError = 0;
218  }
219
220 // se não houver erro na entrada
221  if (0 == ucError){
222
223     switch(n){
224
225         case 1:
226             if(1 == cLedSwi1){
227                 if (status){
228                     GPIOA_PDOR |= uiPin1MaskEnable;
229                 }
230                 else {
231                     GPIOA_PDOR |= uiPin1MaskDisable;
232                 }
233             }
234
235         break;
236
237         case 2:
238             if(1 == cLedSwi2){
239                 if (status){
240                     GPIOA_PDOR |= uiPin2MaskEnable;
241                 }
242                 else {
243                     GPIOA_PDOR |= uiPin2MaskDisable;
244                 }
245             }
246
247         break;
248
249         case 3:
250             if(1 == cLedSwi3){
251                 if (status){
252                     GPIOA_PDOR |= uiPin4MaskEnable;
253                 }
254                 else {
255                     GPIOA_PDOR |= uiPin4MaskDisable;
256                 }
257             }
258
259         break;
260
261         case 4:
262             if(1 == cLedSwi4){
263                 if (status){
264
```

```
                GPIOA_PDOR |= uiPin5MaskEnable;
            }
            else {
                GPIOA_PDOR |= uiPin5MaskDisable;
            }
          }

        break;
     }
    }
}


/* ************************************************************************** */
/* Nome do metodo:          turnOnLED                                         */
/* Descrição:               Liga um LED especificado pela entrada             */
/*                                                                            */
/* Parametros de entrada:   Um inteiro (0<n<5) indicando qual LED sera aceso  */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void turnOnLED(int n){
//testa se houve erro na passagem dos parametros
 if(n > 5){
  ucError = 1;
 }

 else{
  ucError = 0;
 }

// se não houver erro na entrada
 if(0 == ucError){
    switch(n){

        case 1:
            if(1 == cLedSwi1){
                GPIOA_PSOR |= uiPin1MaskEnable;
            }
        break;

        case 2:
            if(1 == cLedSwi2){
                GPIOA_PSOR |= uiPin2MaskEnable;
            }
        break;

        case 3:
            if(1 == cLedSwi3){
                GPIOA_PSOR |= uiPin4MaskEnable;
            }
        break;

        case 4:
            if(1 == cLedSwi4){
                GPIOA_PSOR |= uiPin5MaskEnable;
            }
        break;
     }

   }
}


/* ************************************************************************** */
```

```
/* Nome do metodo:          turnOffLED                                      */
/* Descrição:               Desliga um LED especificado pela entrada        */
/*                                                                          */
/* Parametros de entrada:   Um inteiro (0<n<5) indicando qual LED sera apagado  */
/*                                                                          */
/* Parametros de saida:     n/a                                             */
/* ************************************************************************ */
void turnOffLED(int n){
//testa se houve erro na passagem dos parametros
 if(n > 5){
  ucError = 1;
 }

 else{
  ucError = 0;
 }

// se não houver erro na entrada
 if(0 == ucError){

    switch(n){

        case 1:
            if(1 == cLedSwi1){
                GPIOA_PCOR |= uiPin1MaskDisable;
            }
        break;

        case 2:
            if(1 == cLedSwi2){
                GPIOA_PCOR |= uiPin2MaskDisable;
            }
        break;

        case 3:
            if(1 == cLedSwi3){
                GPIOA_PCOR |= uiPin4MaskDisable;
            }
        break;

        case 4:
            if(1 == cLedSwi4){
                GPIOA_PCOR |= uiPin5MaskDisable;
            }
        break;
     }

   }
}



/* ************************************************************************ */
/* Nome do metodo:          toggleLED                                       */
/* Descrição:               Inverte o status atual de um LED especificado pela  */
/*                          entrada                                         */
/*                                                                          */
/* Parametros de entrada:   Um inteiro (0<n<5) indicando qual LED tera seu      */
/*                          status invertido                                */
/*                                                                          */
/* Parametros de saida:     n/a                                             */
/* ************************************************************************ */
void toggleLED(int n){
//testa se houve erro na passagem dos parametros
 if(n > 5){
  ucError = 1;
```

```
399  }
400
401  else{
402   ucError = 0;
403  }
404
405 // se não houver erro na entrada
406  if(0 == ucError){
407
408     switch(n){
409
410         case 1:
411             if(1 == cLedSwi1){
412                 GPIOA_PTOR |= uiPin1MaskEnable;
413             }
414         break;
415
416         case 2:
417             if(1 == cLedSwi2){
418                 GPIOA_PTOR |= uiPin2MaskEnable;
419             }
420         break;
421
422         case 3:
423             if(1 == cLedSwi3){
424                 GPIOA_PTOR |= uiPin4MaskEnable;
425             }
426         break;
427
428         case 4:
429             if(1 == cLedSwi4){
430                 GPIOA_PTOR |= uiPin5MaskEnable;
431             }
432         break;
433     }

    }
}
```

```
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:       ledSwi.h                                          */
/*                                                                            */
/*   Descrição:             Arquivo Header contendo a declaração              */
/*                       das funções de atuação do microcontrolador           */
/*                       sobre os LEDs e chaves do kit                        */
/*                                                                            */
/*   Autores:              Gustavo Lino e Giácomo Dollevedo                   */
/*   Criado em:             29/03/2020                                        */
/*   Ultima revisão em:     03/04/2020                                        */
/* ************************************************************************** */
#ifndef SOURCES_LEDSWI_

#define SOURCES_LEDSWI_

/* ************************************************************************** */
/* Nome do metodo:        ledSwi_init                                         */
/* Descrição:             Inicializa os GPIO como entrada (botao) ou saida    */
/*                     (LED)                                                  */
/*                                                                            */
/* Parametros de entrada:  Quatro variaveis chareanas que indicam se sera     */
/*                     inicializado como entrada (false) ou saida (true)      */
/*                                                                            */
/* Parametros de saida:    n/a                                                */
/* ************************************************************************** */
void ledSwi_init(char led1, char led2, char led3, char led4);




/* ************************************************************************** */
/* Nome do metodo:        readSwitch                                          */
/* Descrição:             Le o status de um switch para saber se o mesmo      */
/*                     está pressionado ou não                                */
/*                                                                            */
/* Parametros de entrada:  Um inteiro (0<n<5) que indica qual botão será lido */
/*                     inicializado como entrada (botao) ou saida (LED)       */
/*                     0 -> Leitura PTA1; 1 -> Leitura PTA2,                  */
/*                     2 -> Leitura PTA4; 3 -> Leitura PTA5;                  */
/*                                                                            */
/* Parametros de saida:    Um chareano indicando se o botao lido está sendo   */
/*                     pressionado (true), se não está ou se é inválido       */
/*                     (false)                                                */
/* ************************************************************************** */
char readSwitch(int n);



/* ************************************************************************** */
/* Nome do metodo:        writeLED                                            */
/* Descrição:             Liga ou desliga o LED selecionado conforme as       */
/*                     entradas                                               */
/*                                                                            */
/* Parametros de entrada:  Um inteiro (0<n<5) indicando sobre qual LED sera   */
/*                     efetuado o comando;                                    */
/*                     Um chareano (status) indicando se o LED sera           */
/*                     aceso (true) ou apagado (false)                        */
/*                                                                            */
/* Parametros de saida:    n/a                                                */
/* ************************************************************************** */
void writeLED(int n, char status);



/* ************************************************************************** */
```

```
67  /* Nome do metodo:         turnOnLED                                   */
68  /* Descrição:              Liga um LED especificado pela entrada       */
69  /*                                                                     */
70  /* Parametros de entrada:  Um inteiro (0<n<5) indicando qual LED sera aceso  */
71  /*                                                                     */
72  /* Parametros de saida:    n/a                                         */
73  /* ********************************************************************** */
74  void turnOnLED(int n);
75
76
77
78
79  /* ********************************************************************** */
80  /* Nome do metodo:         turnOffLED                                  */
81  /* Descrição:              Desliga um LED especificado pela entrada    */
82  /*                                                                     */
83  /* Parametros de entrada:  Um inteiro (0<n<5) indicando qual LED sera apagado  */
84  /*                                                                     */
85  /* Parametros de saida:    n/a                                         */
86  /* ********************************************************************** */
87  void turnOffLED(int n);
88
89
90
91  /* ********************************************************************** */
92  /* Nome do metodo:         toggleLED                                   */
93  /* Descrição:              Inverte o status atual de um LED especificado pela  */
94  /*                         entrada                                     */
95  /*                                                                     */
96  /* Parametros de entrada:  Um inteiro (0<n<5) indicando qual LED tera seu  */
97  /*                         status invertido                            */
98  /*                                                                     */
99  /* Parametros de saida:    n/a                                         */
100 /* ********************************************************************** */
101 void toggleLED(int n);
102
103
104
105 #endif
```

```
/* ************************************************************** */
/* File name:        tc_hal.c                                     */
/* File description: This file has a couple of useful functions to   */
/*                   timer and counter hardware abstraction layer    */
/*                                                                */
/* Author name:      dloubach                                     */
/* Creation date:    23out2015                                    */
/* Revision date:    25fev2016                                    */
/* ************************************************************** */

#include "lptmr.h"

/* system includes */
#include "fsl_lptmr_driver.h"

#include "fsl_clock_manager.h"
#include "fsl_port_hal.h"
#include "fsl_gpio_hal.h"


/* LPTMR configurations */
lptmr_user_config_t lptmrConfig =
{
    .timerMode              = kLptmrTimerModeTimeCounter,
    .freeRunningEnable      = false,
    .prescalerEnable        = true,
    .prescalerClockSource   = kClockLptmrSrcLpoClk,
    .prescalerValue         = kLptmrPrescalerDivide2,
    .isInterruptEnabled     = true,
};

/* LPTMR driver state information */
lptmr_state_t lptmrState;


/* LPTMR IRQ handler that would cover the same name's APIs in startup code */
/* Do not edit this part */
void LPTMR0_IRQHandler(void)
{
    LPTMR_DRV_IRQHandler(0U);
}



/* ********************************************** */
/* Method name:        tc_installLptmr            */
/* Method description: Low power timer 0          */
/*                     initialization and start   */
/* Input params:       uiTimeInUs:                */
/*                     time in micro seconds      */
/*                     tUserCallback              */
/*                     function pointer to be called*/
/*                     when counter achieves      */
/*                     uiTimeInUs                 */
/* Output params:      n/a                        */
/* ********************************************** */
void tc_installLptmr0(uint32_t uiTimeInUs, lptmr_callback_t tUserCallback)
{
    /* Initialize LPTMR */
    LPTMR_DRV_Init(LPTMR0_IDX, &lptmrState, &lptmrConfig);

    /* Set timer period for TMR_PERIOD micro seconds */
    LPTMR_DRV_SetTimerPeriodUs(LPTMR0_IDX, uiTimeInUs);

    /* Install interrupt call back function for LPTMR */
    LPTMR_DRV_InstallCallback(LPTMR0_IDX, tUserCallback);
```

```
    /* Start LPTMR */
    LPTMR_DRV_Start(LPTMR0_IDX);
}
```

```
/* ************************************************************** */
/* File name:        lptmr.c                                      */
/* File description: Header file containing the functions/methods */
/*                   interfaces for handling timers and counter   */
/*                   from the FRDM-KL25Z board                    */
/* Author name:      dloubach                                     */
/* Creation date:    23out2015                                    */
/* Revision date:    25fev2016                                 s  */
/* ************************************************************** */

#ifndef SOURCES_LPTMR_H_
#define SOURCES_LPTMR_H_

#include "fsl_lptmr_driver.h"

/* ********************************************** */
/* Method name:        tc_installLptmr            */
/* Method description: Low power timer 0          */
/*                   initialization and start     */
/* Input params:     uiTimeInUs:                  */
/*                   time in micro seconds        */
/*                   tUserCallback                */
/*                   function pointer to be called */
/*                   when counter achieves        */
/*                   uiTimeInUs                   */
/* Output params:    n/a                          */
/* ********************************************** */
void tc_installLptmr0(uint32_t uiTimeInUs, lptmr_callback_t tUserCallback);


#endif /* SOURCES_LPTMR_H_ */
```

```
/* ****************************************************************** */
/* File name:        lut_adc_3v3.c                                    */
/* File description: This file cotains the Lookup Table that correlates */
/*                   sensor output and the Temperature in celcius     */
/* Author name:      julioalvesMS & lagoAF & dloubach                 */
/* Creation date:    07jun2018                                        */
/* Revision date:    21jun2018                                        */
/* ****************************************************************** */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                TABELA PARA USO DO SENSOR DE TEMPERATURA           *
 *                modificado para o range 0 - 3v3                    *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

extern const unsigned char tabela_temp[256] = {
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,                  //15
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,                  //31
   1, 1, 2, 2, 3, 3, 3, 3, 4, 4, 5, 5, 6, 6, 6, 6,               //47
   7, 7, 8, 8, 8, 8, 9, 9, 10, 10, 10, 10, 11, 11, 12, 12,           //63
   12, 12, 13, 13, 14, 14, 15, 15, 15, 15, 16, 16, 16, 17, 17, 17,   //79
   17, 18, 18, 19, 19, 19, 19, 20, 20, 21, 21, 21, 21, 22, 22, 23,   //95
   23, 24, 24, 24, 24, 25, 25, 26, 26, 26, 26, 27, 27, 28, 28, 28,   //111
   28, 29, 29, 30, 30, 30, 30, 31, 31, 32, 32, 32, 32, 33, 33, 34,   //127
   34, 35, 35, 35, 35, 36, 36, 37, 37, 37, 37, 38, 38, 39, 39, 39,   //143
   39, 40, 40, 41, 41, 41, 41, 42, 42, 43, 43, 44, 44, 44, 44, 45,   //159
   45, 46, 46, 46, 46, 47, 47, 48, 48, 48, 48, 49, 49, 50, 50, 50,   //175
   50, 51, 51, 52, 52, 53, 53, 53, 53, 54, 54, 55, 55, 55, 55, 56,   //191
   56, 57, 57, 57, 57, 58, 58, 59, 59, 59, 59, 60, 60, 61, 61, 62,   //207
   62, 62, 62, 63, 63, 64, 64, 64, 64, 65, 65, 66, 66, 66, 66, 67,   //223
   67, 68, 68, 68, 68, 69, 69, 70, 70, 71, 71, 71, 71, 72, 72, 72,   //239
   73, 73, 73, 73, 74, 74, 75, 75, 75, 75, 76, 76, 77, 77, 77, 77    //255
};
```

```c
/* ********************************************************************** */
/* File name:        lut_adc_3v3.h                                        */
/* File description: Header file containing the interface for handling  */
/*                   the Lookup Table that correlates sensor output and */
/*                   the Temperature in celcius                         */
/* Author name:      julioalvesMS & lagoAF & dloubach                     */
/* Creation date:    07jun2018                                          */
/* Revision date:    21jun2018                                          */
/* ********************************************************************** */

#ifndef SOURCES_ADC_LUT_ADC_3V3_H_
#define SOURCES_ADC_LUT_ADC_3V3_H_

const unsigned char tabela_temp[256];

#endif /* SOURCES_ADC_LUT_ADC_3V3_H_ */
```

```c
/* ************************************************************** */
/* File name:        pid.c                                        */
/* File description: This file has a couple of useful functions to   */
/*                   control the implemented PID controller         */
/* Author name:      julioalvesMS, lagoAF, rBacurau                 */
/* Creation date:    21jun2018                                      */
/* Revision date:    31jul2020                                      */
/* ************************************************************** */

#include "pid.h"

pid_data_type pidConfig;

/* ********************************************** */
/* Method name:        pid_init                   */
/* Method description: Initialize the PID controller*/
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void pid_init(void)
{
 pidConfig.fKp = 0.0;
 pidConfig.fKd = 0.0;
 pidConfig.fKi = 0.0;
 pidConfig.fError_previous = 0;
 pidConfig.fError_sum = 0.0;
}


/* ************************************************ */
/* Method name:        pid_setKp                    */
/* Method description: Set a new value for the PID    */
/*                     proportional constant          */
/* Input params:       fKp: New value                 */
/* Output params:      n/a                          */
/* ************************************************ */
void pid_setKp(float fKp)
{
 pidConfig.fKp = fKp;
}


/* ************************************************ */
/* Method name:        pid_getKp                    */
/* Method description: Get the value from the PID     */
/*                     proportional constant          */
/* Input params:       n/a                          */
/* Output params:      float: Value                   */
/* ************************************************ */
float pid_getKp(void)
{
 return pidConfig.fKp;
}


/* ************************************************ */
/* Method name:        pid_setKi                    */
/* Method description: Set a new value for the PID    */
/*                     integrative constant           */
/* Input params:       fKi: New value                 */
/* Output params:      n/a                          */
/* ************************************************ */
void pid_setKi(float fKi)
{
 pidConfig.fKi = fKi;
}
```

```
/* ********************************************** */
/* Method name:        pid_getKi                  */
/* Method description: Get the value from the PID */
/*                     integrative constant       */
/* Input params:       n/a                        */
/* Output params:      float: Value               */
/* ********************************************** */
float pid_getKi(void)
{
 return pidConfig.fKi;
}


/* ********************************************** */
/* Method name:        pid_setKd                  */
/* Method description: Set a new value for the PID */
/*                     derivative constant        */
/* Input params:       fKd: New value             */
/* Output params:      n/a                        */
/* ********************************************** */
void pid_setKd(float fKd)
{
 pidConfig.fKd = fKd;
}


/* ********************************************** */
/* Method name:        pid_getKd                  */
/* Method description: Get the value from the PID */
/*                     derivative constant        */
/* Input params:       n/a                        */
/* Output params:      float: Value               */
/* ********************************************** */
float pid_getKd(void)
{
 return pidConfig.fKd;
}


/* ********************************************** */
/* Method name:        pid_updateData             */
/* Method description: Update the control output  */
/*                     using the reference and sensor */
/*                     value                      */
/* Input params:       fSensorValue: Value read from  */
/*                     the sensor                 */
/*                     fReferenceValue: Value used as */
/*                     control reference          */
/*        fDutyCycleHeater: Value of the */
/*        heater duty cycle     */
/* Output params:      float: New Control effort     */
/* ********************************************** */
float pidUpdateData(unsigned char ucTempAtual, float fSetValue, float fDutyCycleHeater)
{
 float fError, fDifference, fOut;

 fError = fSetValue - ucTempAtual;
 /*Devemos incrementar o erro apenas se nÃŁo houver saturado o duty cycle evitando o wind up*/
 if(fDutyCycleHeater < 1.0|| fDutyCycleHeater > 0.0){
  pidConfig.fError_sum += fError;
 }
 fDifference = pidConfig.fError_previous - fError;

 fOut = pidConfig.fKp*fError
   + pidConfig.fKi*pidConfig.fError_sum
   + pidConfig.fKd*fDifference;
```

```
 pidConfig.fError_previous = fError;

 if (fOut>1)
  fOut = 1;

 else if (fOut<0.0)
  fOut = 0.0;

 return fOut;
}
```

```c
/* ************************************************************** */
/* File name:        pid.h                                        */
/* File description: Header file containing the functions/methods */
/*                   interfaces for handling the PID              */
/* Author name:      julioalvesMS, lagoAF, rBacurau               */
/* Creation date:    21jun2018                                    */
/* Revision date:    27mai2020                                    */
/* ************************************************************** */

#ifndef SOURCES_CONTROLLER_PID_H_
#define SOURCES_CONTROLLER_PID_H_


typedef struct pid_data_type {
 float fKp, fKi, fKd;        // PID gains
 float fError_previous;      // used in the derivative
 float fError_sum;           // integrator cumulative error
} pid_data_type;


/* ********************************************** */
/* Method name:        pid_init                   */
/* Method description: Initialize the PID controller*/
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void pid_init(void);


/* ********************************************** */
/* Method name:        pid_setKp                  */
/* Method description: Set a new value for the PID    */
/*                     proportional constant      */
/* Input params:       fKp: New value             */
/* Output params:      n/a                        */
/* ********************************************** */
void pid_setKp(float fKp);


/* ********************************************** */
/* Method name:        pid_getKp                  */
/* Method description: Get the value from the PID     */
/*                     proportional constant      */
/* Input params:       n/a                        */
/* Output params:      float: Value               */
float pid_getKp(void);


/* ********************************************** */
/* Method name:        pid_setKi                  */
/* Method description: Set a new value for the PID    */
/*                     integrative constant       */
/* Input params:       fKi: New value             */
/* Output params:      n/a                        */
/* ********************************************** */
void pid_setKi(float fKi);


/* ********************************************** */
/* Method name:        pid_getKi                  */
/* Method description: Get the value from the PID     */
/*                     integrative constant       */
/* Input params:       n/a                        */
/* Output params:      float: Value               */
/* ********************************************** */
float pid_getKi(void);
```

```
/* ********************************************** */
/* Method name:        pid_setKd                  */
/* Method description: Set a new value for the PID    */
/*                     derivative constant            */
/* Input params:       fKd: New value                 */
/* Output params:      n/a                        */
/* ********************************************** */
void pid_setKd(float fKd);


/* ********************************************** */
/* Method name:        pid_getKd                  */
/* Method description: Get the value from the PID     */
/*                     derivative constant            */
/* Input params:       n/a                        */
/* Output params:      float: Value                   */
/* ********************************************** */
float pid_getKd(void);


/* ********************************************** */
/* Method name:        pid_updateData                 */
/* Method description: Update the control output      */
/*                     using the reference and sensor */
/*                     value                          */
/* Input params:       fSensorValue: Value read from  */
/*                     the sensor                     */
/*                     fReferenceValue: Value used as */
/*                     control reference              */
/*        fDutyCycleHeater: Value of the */
/*        heater duty cycle     */
/* Output params:      float: New Control effort      */
/* ********************************************** */
float pidUpdateData(unsigned char ucTempAtual, float fSetValue, float fDutyCycleHeater);


#endif /* SOURCES_CONTROLLER_PID_H_ */
```

```
/*****************************************************************
 * File:    print_scan.c
 * Purpose: Implementation of debug_printf(), debug_scanf() functions.
 *
 * This is a modified version of the file printf.c, which was distributed
 * by Motorola as part of the M5407C3BOOT.zip package used to initialize
 * the M5407C3 evaluation board.
 *
 * Copyright:
 *      1999-2000 MOTOROLA, INC. All Rights Reserved.
 *  You are hereby granted a copyright license to use, modify, and
 *  distribute the SOFTWARE so long as this entire notice is
 *  retained without alteration in any modified and/or redistributed
 *  versions, and that such modified versions are clearly identified
 *  as such. No licenses are granted by implication, estoppel or
 *  otherwise under any patents or trademarks of Motorola, Inc. This
 *  software is provided on an "AS IS" basis and without warranty.
 *
 *  To the maximum extent permitted by applicable law, MOTOROLA
 *  DISCLAIMS ALL WARRANTIES WHETHER EXPRESS OR IMPLIED, INCLUDING
 *  IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR
 *  PURPOSE AND ANY WARRANTY AGAINST INFRINGEMENT WITH REGARD TO THE
 *  SOFTWARE (INCLUDING ANY MODIFIED VERSIONS THEREOF) AND ANY
 *  ACCOMPANYING WRITTEN MATERIALS.
 *
 *  To the maximum extent permitted by applicable law, IN NO EVENT
 *  SHALL MOTOROLA BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING
 *  WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS
 *  INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY
 *  LOSS) ARISING OF THE USE OR INABILITY TO USE THE SOFTWARE.
 *
 *  Motorola assumes no responsibility for the maintenance and support
 *  of this software
 ******************************************************************/

#include "print_scan.h"
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdint.h>
#include <stdbool.h>
// Keil: suppress ellipsis warning in va_arg usage below
#if defined(__CC_ARM)
#pragma diag_suppress 1256
#endif

#define FLAGS_MINUS     (0x01)
#define FLAGS_PLUS      (0x02)
#define FLAGS_SPACE     (0x04)
#define FLAGS_ZERO      (0x08)
#define FLAGS_POUND     (0x10)

#define IS_FLAG_MINUS(a)    (a & FLAGS_MINUS)
#define IS_FLAG_PLUS(a)     (a & FLAGS_PLUS)
#define IS_FLAG_SPACE(a)    (a & FLAGS_SPACE)
#define IS_FLAG_ZERO(a)     (a & FLAGS_ZERO)
#define IS_FLAG_POUND(a)    (a & FLAGS_POUND)

#define LENMOD_h        (0x01)
#define LENMOD_l        (0x02)
#define LENMOD_L        (0x04)
#define LENMOD_hh       (0x08)
#define LENMOD_ll       (0x10)

#define IS_LENMOD_h(a)  (a & LENMOD_h)
#define IS_LENMOD_hh(a) (a & LENMOD_hh)
#define IS_LENMOD_l(a)  (a & LENMOD_l)
```

```
#define IS_LENMOD_ll(a) (a & LENMOD_ll)
#define IS_LENMOD_L(a)  (a & LENMOD_L)

#define SCAN_SUPPRESS             0x2

#define SCAN_DEST_MASK            0x7c
#define SCAN_DEST_CHAR            0x4
#define SCAN_DEST_STRING          0x8
#define SCAN_DEST_SET             0x10
#define SCAN_DEST_INT             0x20
#define SCAN_DEST_FLOAT           0x30

#define SCAN_LENGTH_MASK          0x1f00
#define SCAN_LENGTH_CHAR          0x100
#define SCAN_LENGTH_SHORT_INT     0x200
#define SCAN_LENGTH_LONG_INT      0x400
#define SCAN_LENGTH_LONG_LONG_INT 0x800
#define SCAN_LENGTH_LONG_DOUBLE   0x1000

#define SCAN_TYPE_SIGNED          0x2000

/*!
 * @brief Scanline function which ignores white spaces.
 *
 * @param[in] s The address of the string pointer to update.
 *
 * @return String without white spaces.
 */
static uint32_t scan_ignore_white_space(const char **s);

#if defined(SCANF_FLOAT_ENABLE)
static double fnum = 0.0;
#endif

/*!
 * @brief Converts a radix number to a string and return its length.
 *
 * @param[in] numstr    Converted string of the number.
 * @param[in] nump      Pointer to the number.
 * @param[in] neg       Polarity of the number.
 * @param[in] radix     The radix to be converted to.
 * @param[in] use_caps  Used to identify %x/X output format.

 * @return Length of the converted string.
 */
static int32_t mknumstr (char *numstr, void *nump, int32_t neg, int32_t radix, bool use_caps);

#if defined(PRINTF_FLOAT_ENABLE)
/*!
 * @brief Converts a floating radix number to a string and return its length.
 *
 * @param[in] numstr          Converted string of the number.
 * @param[in] nump            Pointer to the number.
 * @param[in] radix           The radix to be converted to.
 * @param[in] precision_width   Specify the precision width.

 * @return Length of the converted string.
 */
static int32_t mkfloatnumstr (char *numstr, void *nump, int32_t radix, uint32_t precision_width);
#endif


static void fput_pad(int32_t c, int32_t curlen, int32_t field_width, int32_t *count, PUTCHAR_FUNC func_ptr, void *farg, int
*max_count);

double modf(double input_dbl, double  *intpart_ptr);
```

```
#if !defined(PRINT_MAX_COUNT)
#define n_putchar(func, chacter, p, count)      func(chacter, p)
#else
static int n_putchar(PUTCHAR_FUNC func_ptr, int chacter, void *p, int *max_count)
{
    int result = 0;
    if (*max_count)
    {
        result = func_ptr(chacter, p);
        (*max_count)--;
    }
    return result;
}
#endif

/*FUNCTION*****************************************************************
 *
 * Function Name : _doprint
 * Description   : This function outputs its parameters according to a
 * formatted string. I/O is performed by calling given function pointer
 * using following (*func_ptr)(c,farg);
 *
 *END*********************************************************************/
int _doprint(void *farg, PUTCHAR_FUNC func_ptr, int max_count, char *fmt, va_list ap)
{
    /* va_list ap; */
    char *p;
    int32_t c;

    char vstr[33];
    char *vstrp;
    int32_t vlen;

    int32_t done;
    int32_t count = 0;
    int temp_count = max_count;


    uint32_t flags_used;
    uint32_t field_width;

    int32_t ival;
    int32_t schar, dschar;
    int32_t *ivalp;
    char *sval;
    int32_t cval;
    uint32_t uval;
    bool use_caps;
    uint32_t precision_width;
    //uint32_t length_modifier = 0;
#if defined(PRINTF_FLOAT_ENABLE)
    double fval;
#endif

    if (max_count == -1)
    {
        max_count = INT32_MAX - 1;
    }

    /*
     * Start parsing apart the format string and display appropriate
     * formats and data.
     */
    for (p = (char *)fmt; (c = *p) != 0; p++)
    {
        /*
         * All formats begin with a '%' marker.  Special chars like
```

```
     * '\n' or '\t' are normally converted to the appropriate
     * character by the __compiler__.  Thus, no need for this
     * routine to account for the '\' character.
     */
    if (c != '%')
    {
        n_putchar(func_ptr, c, farg, &max_count);

        count++;

        /*
         * By using 'continue', the next iteration of the loop
         * is used, skipping the code that follows.
         */
        continue;
    }

    /*
     * First check for specification modifier flags.
     */
    use_caps = true;
    flags_used = 0;
    done = false;
    while (!done)
    {
        switch (/* c = */*++p)
        {
            case '-':
                flags_used |= FLAGS_MINUS;
                break;
            case '+':
                flags_used |= FLAGS_PLUS;
                break;
            case ' ':
                flags_used |= FLAGS_SPACE;
                break;
            case '0':
                flags_used |= FLAGS_ZERO;
                break;
            case '#':
                flags_used |= FLAGS_POUND;
                break;
            default:
                /* we've gone one char too far */
                --p;
                done = true;
                break;
        }
    }

    /*
     * Next check for minimum field width.
     */
    field_width = 0;
    done = false;
    while (!done)
    {
        switch (c = *++p)
        {
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
```

```
        case '7':
        case '8':
        case '9':
            field_width = (field_width * 10) + (c - '0');
            break;
        default:
            /* we've gone one char too far */
            --p;
            done = true;
            break;
    }
}

/*
 * Next check for the width and precision field separator.
 */
precision_width = 6;
if (/* (c = *++p) */*++p == '.')
{
    /* precision_used = true; */

    /*
     * Must get precision field width, if present.
     */
    precision_width = 0;
    done = false;
    while (!done)
    {
        switch (c = *++p)
        {
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                precision_width = (precision_width * 10) + (c - '0');
                break;
            default:
                /* we've gone one char too far */
                --p;
                done = true;
                break;
        }
    }
}
else
{
    /* we've gone one char too far */
    --p;
}

/*
 * Check for the length modifier.
 */
/* length_modifier = 0; */
switch (/* c = */*++p)
{
    case 'h':
        if (*++p != 'h')
        {
```

```
            --p;
        }
        /* length_modifier |= LENMOD_h; */
        break;
    case 'l':
        if (*++p != 'l')
        {
            --p;
        }
        /* length_modifier |= LENMOD_l; */
        break;
    case 'L':
        /* length_modifier |= LENMOD_L; */
        break;
    default:
        /* we've gone one char too far */
        --p;
        break;
}

/*
 * Now we're ready to examine the format.
 */
switch (c = *++p)
{
    case 'd':
    case 'i':
        ival = (int32_t)va_arg(ap, int32_t);
        vlen = mknumstr(vstr,&ival,true,10,use_caps);
        vstrp = &vstr[vlen];

        if (ival < 0)
        {
            schar = '-';
            ++vlen;
        }
        else
        {
            if (IS_FLAG_PLUS(flags_used))
            {
                schar = '+';
                ++vlen;
            }
            else
            {
                if (IS_FLAG_SPACE(flags_used))
                {
                    schar = ' ';
                    ++vlen;
                }
                else
                {
                    schar = 0;
                }
            }
        }
        dschar = false;

        /*
         * do the ZERO pad.
         */
        if (IS_FLAG_ZERO(flags_used))
        {
            if (schar)
            {
                n_putchar(func_ptr, schar, farg, &max_count);
```

```
                count++;
            }
            dschar = true;

            fput_pad('0', vlen, field_width, &count, func_ptr, farg, &max_count);
            vlen = field_width;
        }
        else
        {
            if (!IS_FLAG_MINUS(flags_used))
            {
                fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
                if (schar)
                {
                    n_putchar(func_ptr, schar, farg, &max_count);
                    count++;
                }
                dschar = true;
            }
        }

        /* the string was built in reverse order, now display in */
        /* correct order */
        if ((!dschar) && schar)
        {
            n_putchar(func_ptr, schar, farg, &max_count);
            count++;
        }
        goto cont_xd;
#if defined(PRINTF_FLOAT_ENABLE)
    case 'f':
    case 'F':
        fval = (double)va_arg(ap, double);
        vlen = mkfloatnumstr(vstr,&fval,10, precision_width);
        vstrp = &vstr[vlen];

        if (fval < 0)
        {
            schar = '-';
            ++vlen;
        }
        else
        {
            if (IS_FLAG_PLUS(flags_used))
            {
                schar = '+';
                ++vlen;
            }
            else
            {
                if (IS_FLAG_SPACE(flags_used))
                {
                    schar = ' ';
                    ++vlen;
                }
                else
                {
                    schar = 0;
                }
            }
        }
        dschar = false;
        if (IS_FLAG_ZERO(flags_used))
        {
            if (schar)
            {
```

```
                n_putchar(func_ptr, schar, farg, &max_count);
                count++;
            }
            dschar = true;
            fput_pad('0', vlen, field_width, &count, func_ptr, farg, &max_count);
            vlen = field_width;
        }
        else
        {
            if (!IS_FLAG_MINUS(flags_used))
            {
                fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
                if (schar)
                {
                    n_putchar(func_ptr, schar, farg, &max_count);
                    count++;
                }
                dschar = true;
            }
        }
        if (!dschar && schar)
        {
            n_putchar(func_ptr, schar, farg, &max_count);
            count++;
        }
        goto cont_xd;
#endif
    case 'x':
        use_caps = false;
    case 'X':
        uval = (uint32_t)va_arg(ap, uint32_t);
        vlen = mknumstr(vstr,&uval,false,16,use_caps);
        vstrp = &vstr[vlen];

        dschar = false;
        if (IS_FLAG_ZERO(flags_used))
        {
            if (IS_FLAG_POUND(flags_used))
            {
                n_putchar(func_ptr, '0', farg, &max_count);
                n_putchar(func_ptr, (use_caps ? 'X' : 'x'), farg, &max_count);
                count += 2;
                /*vlen += 2;*/
                dschar = true;
            }
            fput_pad('0', vlen, field_width, &count, func_ptr, farg, &max_count);
            vlen = field_width;
        }
        else
        {
            if (!IS_FLAG_MINUS(flags_used))
            {
                if (IS_FLAG_POUND(flags_used))
                {
                    vlen += 2;
                }
                fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
                if (IS_FLAG_POUND(flags_used))
                {
                    n_putchar(func_ptr, '0', farg, &max_count);
                    n_putchar(func_ptr, (use_caps ? 'X' : 'x'), farg, &max_count);
                    count += 2;

                    dschar = true;
                }
            }
```

```
    }

    if ((IS_FLAG_POUND(flags_used)) && (!dschar))
    {
        n_putchar(func_ptr, '0', farg, &max_count);
        n_putchar(func_ptr, (use_caps ? 'X' : 'x'), farg, &max_count);
        count += 2;
        vlen += 2;
    }
    goto cont_xd;

case 'o':
    uval = (uint32_t)va_arg(ap, uint32_t);
    vlen = mknumstr(vstr,&uval,false,8,use_caps);
    goto cont_u;
case 'b':
    uval = (uint32_t)va_arg(ap, uint32_t);
    vlen = mknumstr(vstr,&uval,false,2,use_caps);
    goto cont_u;
case 'p':
    uval = (uint32_t)va_arg(ap, uint32_t);
    uval = (uint32_t)va_arg(ap, void *);
    vlen = mknumstr(vstr,&uval,false,16,use_caps);
    goto cont_u;
case 'u':
    uval = (uint32_t)va_arg(ap, uint32_t);
    vlen = mknumstr(vstr,&uval,false,10,use_caps);

    cont_u:
        vstrp = &vstr[vlen];

        if (IS_FLAG_ZERO(flags_used))
        {
            fput_pad('0', vlen, field_width, &count, func_ptr, farg, &max_count);
            vlen = field_width;
        }
        else
        {
            if (!IS_FLAG_MINUS(flags_used))
            {
                fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
            }
        }

    cont_xd:
        while (*vstrp)
        {
            n_putchar(func_ptr, *vstrp--, farg, &max_count);
            count++;
        }

        if (IS_FLAG_MINUS(flags_used))
        {
            fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
        }
    break;

case 'c':
    cval = (char)va_arg(ap, uint32_t);
    n_putchar(func_ptr, cval, farg, &max_count);
    count++;
    break;
case 's':
    sval = (char *)va_arg(ap, char *);
    if (sval)
    {
```

```
                vlen = strlen(sval);
                if (!IS_FLAG_MINUS(flags_used))
                {
                    fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
                }
                while (*sval)
                {
                    n_putchar(func_ptr, *sval++, farg, &max_count);
                    count++;
                }
                if (IS_FLAG_MINUS(flags_used))
                {
                    fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
                }
            }
            break;
        case 'n':
            ivalp = (int32_t *)va_arg(ap, int32_t *);
            *ivalp = count;
            break;
        default:
            n_putchar(func_ptr, c, farg, &max_count);
            count++;
            break;
      }
   }

   if (max_count)
   {
      return count;
   }
   else
   {
      return temp_count;
   }
}

/*FUNCTION*****************************************************************
 *
 * Function Name : _sputc
 * Description   : Writes the character into the string located by the string
 * pointer and updates the string pointer.
 *
 *END*****************************************************************/
int _sputc(int c, void * input_string)
{
   char **string_ptr = (char  **)input_string;

   *(*string_ptr)++ = (char)c;
   return c;
}

/*FUNCTION*****************************************************************
 *
 * Function Name : mknumstr
 * Description   : Converts a radix number to a string and return its length.
 *
 *END*****************************************************************/
static int32_t mknumstr (char *numstr, void *nump, int32_t neg, int32_t radix, bool use_caps)
{
    int32_t a,b,c;
    uint32_t ua,ub,uc;

    int32_t nlen;
    char *nstrp;
```

```
    nlen = 0;
    nstrp = numstr;
    *nstrp++ = '\0';

    if (neg)
    {
        a = *(int32_t *)nump;
        if (a == 0)
        {
            *nstrp = '0';
            ++nlen;
            goto done;
        }
        while (a != 0)
        {
            b = (int32_t)a / (int32_t)radix;
            c = (int32_t)a - ((int32_t)b * (int32_t)radix);
            if (c < 0)
            {
                c = ~c + 1 + '0';
            }
            else
            {
                c = c + '0';
            }
            a = b;
            *nstrp++ = (char)c;
            ++nlen;
        }
    }
    else
    {
        ua = *(uint32_t *)nump;
        if (ua == 0)
        {
            *nstrp = '0';
            ++nlen;
            goto done;
        }
        while (ua != 0)
        {
            ub = (uint32_t)ua / (uint32_t)radix;
            uc = (uint32_t)ua - ((uint32_t)ub * (uint32_t)radix);
            if (uc < 10)
            {
                uc = uc + '0';
            }
            else
            {
                uc = uc - 10 + (use_caps ? 'A' : 'a');
            }
            ua = ub;
            *nstrp++ = (char)uc;
            ++nlen;
        }
    }
    done:
    return nlen;
}

#if defined(PRINTF_FLOAT_ENABLE)
/*FUNCTION**********************************************************************
 *
 * Function Name : mkfloatnumstr
 * Description   : Converts a floating radix number to a string and return
 * its length, user can specify output precision width.
```

```
 *
 *END**************************************************************************/
static int32_t mkfloatnumstr (char *numstr, void *nump, int32_t radix, uint32_t precision_width)
{
    int32_t a,b,c,i;
    double fa,fb;
    double r, fractpart, intpart;

    int32_t nlen;
    char *nstrp;
    nlen = 0;
    nstrp = numstr;
    *nstrp++ = '\0';
    r = *(double *)nump;
    if (r == 0)
    {
        *nstrp = '0';
        ++nlen;
        goto done;
    }
    fractpart = modf((double)r , (double *)&intpart);
    /* Process fractional part */
    for (i = 0; i < precision_width; i++)
    {
        fractpart *= radix;
    }
    //a = (int32_t)floor(fractpart + (double)0.5);
    fa = fractpart + (double)0.5;
    for (i = 0; i < precision_width; i++)
    {
        fb = fa / (int32_t)radix;
        c = (int32_t)(fa - (uint64_t)fb * (int32_t)radix);
        if (c < 0)
        {
            c = ~c + 1 + '0';
        }else
        {
            c = c + '0';
        }
        fa = fb;
        *nstrp++ = (char)c;
        ++nlen;
    }
    *nstrp++ = (char)'.';
    ++nlen;
    a = (int32_t)intpart;
    while (a != 0)
    {
        b = (int32_t)a / (int32_t)radix;
        c = (int32_t)a - ((int32_t)b * (int32_t)radix);
        if (c < 0)
        {
            c = ~c + 1 + '0';
        }else
        {
            c = c + '0';
        }
        a = b;
        *nstrp++ = (char)c;
        ++nlen;
    }
    done:
    return nlen;
}
#endif
```

```
static void fput_pad(int32_t c, int32_t curlen, int32_t field_width, int32_t *count, PUTCHAR_FUNC func_ptr, void *farg, int
*max_count)
{
   int32_t i;

   for (i = curlen; i < field_width; i++)
   {
      func_ptr((char)c, farg);
      (*count)++;
   }
}

/*FUNCTION*****************************************************************
 *
 * Function Name : scan_prv
 * Description   : Converts an input line of ASCII characters based upon a
 * provided string format.
 *
 *END*********************************************************************/
int scan_prv(const char *line_ptr, char *format, va_list args_ptr)
{
   uint8_t base;
   /* Identifier for the format string */
   char *c = format;
   const char *s;
   char temp;
   /* Identifier for the input string */
   const char *p = line_ptr;
   /* flag telling the conversion specification */
   uint32_t flag = 0 ;
   /* filed width for the matching input streams */
   uint32_t field_width;
   /* how many arguments are assigned except the suppress */
   uint32_t nassigned = 0;
   /* how many characters are read from the input streams */
   uint32_t n_decode = 0;

   int32_t val;
   char *buf;
   int8_t neg;

   /* return EOF error before any convernsion */
   if (*p == '\0')
   {
      return EOF;
   }

   /* decode directives */
   while ((*c) && (*p))
   {
      /* ignore all white-spaces in the format strings */
      if (scan_ignore_white_space((const char **)&c))
      {
         n_decode += scan_ignore_white_space(&p);
      }
      else if (*c != '%')
      {
         /* Ordinary characters */
         c++;
ordinary:   if (*p == *c)
         {
            n_decode++;
            p++;
            c++;
         }
         else
```

```
        {
            /* Match failure. Misalignment with C99, the unmatched
             * characters need to be pushed back to stream. HOwever
             * , it is deserted now. */
            break;
        }
    }
    else
    {
        /* convernsion specification */
        c++;
        if (*c == '%')
        {
            goto ordinary;
        }

        /* Reset */
        flag = 0;
        field_width = 0;
        base = 0;

        /* Loop to get full conversion specification */
        while ((*c) && (!(flag & SCAN_DEST_MASK)))
        {
            switch (*c)
            {
                case '*':
                    if (flag & SCAN_SUPPRESS)
                    {
                        /* Match failure*/
                        return nassigned;
                    }
                    flag |= SCAN_SUPPRESS;
                    c++;
                    break;
                case 'h':
                    if (flag & SCAN_LENGTH_MASK)
                    {
                        /* Match failure*/
                        return nassigned;
                    }
                    flag |= SCAN_LENGTH_SHORT_INT;

                    if (c[1] == 'h')
                    {
                        flag |= SCAN_LENGTH_CHAR;
                        c++;
                    }
                    c++;
                    break;
                case 'l':
                    if (flag & SCAN_LENGTH_MASK)
                    {
                        /* Match failure*/
                        return nassigned;
                    }
                    flag |= SCAN_LENGTH_LONG_INT;

                    if (c[1] == 'l')
                    {
                        flag |= SCAN_LENGTH_LONG_LONG_INT;
                        c++;
                    }
                    c++;
                    break;
#if defined(ADVANCE)
```

```
                case 'j':
                    if (flag & SCAN_LENGTH_MASK)
                    {
                        /* Match failure*/
                        return nassigned;
                    }
                    flag |= SCAN_LENGTH_INTMAX;
                    c++
                case 'z'
                    if (flag & SCAN_LENGTH_MASK)
                    {
                        /* Match failure*/
                        return nassigned;
                    }
                    flag |= SCAN_LENGTH_SIZE_T;
                    c++;
                    break;
                case 't':
                    if (flag & SCAN_LENGTH_MASK)
                    {
                        /* Match failure*/
                        return nassigned;
                    }
                    flag |= SCAN_LENGTH_PTRDIFF_T;
                    c++;
                    break;
#endif
#if defined(SCANF_FLOAT_ENABLE)
                case 'L':
                    if (flag & SCAN_LENGTH_MASK)
                    {
                        /* Match failure*/
                        return nassigned;
                    }
                    flag |= SCAN_LENGTH_LONG_DOUBLE;
                    c++;
                    break;
#endif
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                case '8':
                case '9':
                    if (field_width)
                    {
                        /* Match failure*/
                        return nassigned;
                    }
                    do {
                        field_width = field_width * 10 + *c - '0';
                        c++;
                    } while ((*c >= '0') && (*c <= '9'));
                    break;
                case 'd':
                    flag |= SCAN_TYPE_SIGNED;
                case 'u':
                    base = 10;
                    flag |= SCAN_DEST_INT;
                    c++;
                    break;
```

```
                case 'o':
                    base = 8;
                    flag |= SCAN_DEST_INT;
                    c++;
                    break;
                case 'x':
                case 'X':
                    base = 16;
                    flag |= SCAN_DEST_INT;
                    c++;
                    break;
                case 'i':
                    base = 0;
                    flag |= SCAN_DEST_INT;
                    c++;
                    break;
#if defined(SCANF_FLOAT_ENABLE)
                case 'a':
                case 'A':
                case 'e':
                case 'E':
                case 'f':
                case 'F':
                case 'g':
                case 'G':
                    flag |= SCAN_DEST_FLOAT;
                    c++;
                    break;
#endif
                case 'c':
                    flag |= SCAN_DEST_CHAR;
                    if (!field_width)
                    {
                        field_width = 1;
                    }
                    c++;
                    break;
                case 's':
                    flag |= SCAN_DEST_STRING;
                    c++;
                    break;
#if defined(ADVANCE) /* [x]*/
                case '[':
                    flag |= SCAN_DEST_SET;
                    /*Add Set functionality */
                    break;
#endif
                default:
#if defined(SCAN_DEBUG)
                    printf("Unrecognized expression specifier: %c format: %s, number is: %d\r\n", c, format, nassigned);
#endif
                    return nassigned;
            }
        }

        if (!(flag & SCAN_DEST_MASK))
        {
            /* Format strings are exausted */
            return nassigned;
        }

        if (!field_width)
        {
            /* Larget then length of a line */
            field_width = 99;
```

```
    }

    /* Matching strings in input streams and assign to argument */
    switch (flag & SCAN_DEST_MASK)
    {
        case SCAN_DEST_CHAR:
            s = (const char *)p;
            buf = va_arg(args_ptr, char *);
            while ((field_width--) && (*p))
            {
                if (!(flag & SCAN_SUPPRESS))
                {
                    *buf++ = *p++;
                }
                else
                {
                    p++;
                }
                n_decode++;
            }

            if (((!(flag)) & SCAN_SUPPRESS) && (s != p))
            {
                nassigned++;
            }
            break;
        case SCAN_DEST_STRING:
            n_decode += scan_ignore_white_space(&p);
            s = p;
            buf = va_arg(args_ptr, char *);
            while ((field_width--) && (*p != '\0') && (*p != ' ') &&
                   (*p != '\t') && (*p != '\n') && (*p != '\r') && (*p != '\v') && (*p != '\f'))
            {
                if (flag & SCAN_SUPPRESS)
                {
                    p++;
                }
                else
                {
                    *buf++ = *p++;
                }
                n_decode++;
            }

            if ((!(flag & SCAN_SUPPRESS)) && (s != p))
            {
                /* Add NULL to end of string */
                *buf = '\0';
                nassigned++;
            }
            break;
        case SCAN_DEST_INT:
            n_decode += scan_ignore_white_space(&p);
            s = p;
            val = 0;
            /*TODO: scope is not testsed */
            if ((base == 0) || (base == 16))
            {
                if ((s[0] == '0') && ((s[1] == 'x') || (s[1] == 'X')))
                {
                    base = 16;
                    if (field_width >= 1)
                    {
                        p += 2;
                        n_decode += 2;
                        field_width -= 2;
```

```
        }
    }
}

if (base == 0)
{
    if (s[0] == '0')
    {
        base = 8;
    }
    else
    {
        base = 10;
    }
}

neg = 1;
switch (*p)
{
    case '-':
        neg = -1;
        n_decode++;
        p++;
        field_width--;
        break;
    case '+':
        neg = 1;
        n_decode++;
        p++;
        field_width--;
        break;
    default:
        break;
}

while ((*p) && (field_width--))
{
    if ((*p <= '9') && (*p >= '0'))
    {
        temp = *p - '0';
    }
    else if((*p <= 'f') && (*p >= 'a'))
    {
        temp = *p - 'a' + 10;
    }
    else if((*p <= 'F') && (*p >= 'A'))
    {
        temp = *p - 'A' + 10;
    }
    else
    {
        break;
    }

    if (temp >= base)
    {
        break;
    }
    else
    {
        val = base * val + temp;
    }
    p++;
    n_decode++;
}
```

```
            val *= neg;
            if (!(flag & SCAN_SUPPRESS))
            {
               switch (flag & SCAN_LENGTH_MASK)
               {
                  case SCAN_LENGTH_CHAR:
                     if (flag & SCAN_TYPE_SIGNED)
                     {
                        *va_arg(args_ptr, signed char *) = (signed char)val;
                     }
                     else
                     {
                        *va_arg(args_ptr, unsigned char *) = (unsigned char)val;
                     }
                     break;
                  case SCAN_LENGTH_SHORT_INT:
                     if (flag & SCAN_TYPE_SIGNED)
                     {
                        *va_arg(args_ptr, signed short *) = (signed short)val;
                     }
                     else
                     {
                        *va_arg(args_ptr, unsigned short *) = (unsigned short)val;
                     }
                     break;
                  case SCAN_LENGTH_LONG_INT:
                     if (flag & SCAN_TYPE_SIGNED)
                     {
                        *va_arg(args_ptr, signed long int *) = (signed long int)val;
                     }
                     else
                     {
                        *va_arg(args_ptr, unsigned long int *) = (unsigned long int)val;
                     }
                     break;
                  case SCAN_LENGTH_LONG_LONG_INT:
                     if (flag & SCAN_TYPE_SIGNED)
                     {
                        *va_arg(args_ptr, signed long long int *) = (signed long long int)val;
                     }
                     else
                     {
                        *va_arg(args_ptr, unsigned long long int *) = (unsigned long long int)val;
                     }
                     break;
                  default:
                     /* The default type is the type int */
                     if (flag & SCAN_TYPE_SIGNED)
                     {
                        *va_arg(args_ptr, signed int *) = (signed int)val;
                     }
                     else
                     {
                        *va_arg(args_ptr, unsigned int *) = (unsigned int)val;
                     }
                     break;
               }
               nassigned++;
            }
            break;
#if defined(SCANF_FLOAT_ENABLE)
         case SCAN_DEST_FLOAT:
            n_decode += scan_ignore_white_space(&p);
            fnum = strtod(p, (char **)&s);

            if ((fnum == HUGE_VAL) || (fnum == -HUGE_VAL))
```

```
                {
                    break;
                }

                n_decode += (int)(s) - (int)(p);
                p = s;
                if (!(flag & SCAN_SUPPRESS))
                {
                    if (flag & SCAN_LENGTH_LONG_DOUBLE)
                    {
                        *va_arg(args_ptr, double *) = fnum;
                    }
                    else
                    {
                        *va_arg(args_ptr, float *) = (float)fnum;
                    }
                    nassigned++;
                }
                break;
#endif
#if defined(ADVANCE)
            case SCAN_DEST_SET:
                break;
#endif
            default:
#if defined(SCAN_DEBUG)
                printf("ERROR: File %s  line: %d\r\n", __FILE__, __LINE__);
#endif
                return nassigned;
        }
    }
  }
  return nassigned;
}

/*FUNCTION*****************************************************************
 *
 * Function Name : scan_ignore_white_space
 * Description   : Scanline function which ignores white spaces.
 *
 *END*********************************************************************/
static uint32_t scan_ignore_white_space(const char **s)
{
  uint8_t count = 0;
  uint8_t c;

  c = **s;
  while ((c == ' ') || (c == '\t') || (c == '\n') || (c == '\r') || (c == '\v') || (c == '\f'))
  {
    count++;
    (*s)++;
    c = **s;
  }
  return count;
}
```

```
/*
 * Copyright (c) 2013 - 2014, Freescale Semiconductor, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright notice, this
 *   list of conditions and the following disclaimer in the documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of Freescale Semiconductor, Inc. nor the names of its
 *   contributors may be used to endorse or promote products derived from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
#ifndef __print_scan_h__
#define __print_scan_h__

#include <stdio.h>
#include <stdarg.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>

//#define PRINTF_FLOAT_ENABLE   1
//#define PRINT_MAX_COUNT       1
//#define SCANF_FLOAT_ENABLE    1

#ifndef HUGE_VAL
#define HUGE_VAL        (99.e99)///wrong value
#endif

typedef int (*PUTCHAR_FUNC)(int a, void *b);

/*!
 * @brief This function outputs its parameters according to a formatted string.
 *
 * @note I/O is performed by calling given function pointer using following
 * (*func_ptr)(c,farg);
 *
 * @param[in] farg      Argument to func_ptr.
 * @param[in] func_ptr  Function to put character out.
 * @param[in] max_count Maximum character count for snprintf and vsnprintf.
 * Default value is 0 (unlimited size).
 * @param[in] fmt_ptr   Format string for printf.
 * @param[in] args_ptr  Arguments to printf.
 *
 * @return Number of characters
 * @return EOF (End Of File found.)
 */
int _doprint(void *farg, PUTCHAR_FUNC func_ptr, int max_count, char  *fmt, va_list ap);

/*!
```

```
 * @brief Writes the character into the string located by the string pointer and
 * updates the string pointer.
 *
 * @param[in]      c            The character to put into the string.
 * @param[in, out] input_string This is an updated pointer to a string pointer.
 *
 * @return Character written into string.
 */
int _sputc(int c, void * input_string);

/*!
 * @brief Converts an input line of ASCII characters based upon a provided
 * string format.
 *
 * @param[in] line_ptr The input line of ASCII data.
 * @param[in] format   Format first points to the format string.
 * @param[in] args_ptr The list of parameters.
 *
 * @return Number of input items converted and assigned.
 * @return IO_EOF - When line_ptr is empty string "".
 */
int scan_prv(const char *line_ptr, char *format, va_list args_ptr);

#endif
```

```c
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:       sensTemp.c                                        */
/*                                                                            */
/*   Descricao:             Funcoes para operar o sensor de temperatura, apenas     */
/*                       no contexto do projeto                               */
/*                                                                            */
/*   Autores:               Gustavo Lino e Giacomo A. Dollevedo                */
/*   Criado em:              29/07/2020                                        */
/*   Ultima revisao em:     31/07/2020                                         */
/* ************************************************************************** */

/* Incluindo bibliotecas */
#include "adc.h"
#include "variaveis_globais.h"
#include "lut_adc_3v3.h"
#include "util.h"

/* ************************************************************************** */
/* Nome do metodo:          sensTemp_init                                     */
/* Descricao:               Inicializa as funcoes do ADC para sensor de temperatura */
/*                                                                            */
/* Parametros de entrada:   n/a                                               */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void sensTemp_init(void){

 adc_initADCModule();

}


/* ************************************************************************** */
/* Nome do metodo:          readTemp                                          */
/* Descricao:               Le o ADC conectado ao sensor de temperatura e       */
/*                       converte o valor pela lookup table em graus Celsius */
/*                                                                            */
/* Parametros de entrada:   n/a                                               */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void readTemp(void){

   int iRawTempAtual = 0;

   adc_initConvertion();
   while(0 == adc_isAdcDone())
   {
      util_genDelay250us();
   }

   iRawTempAtual   = adc_getConvertionValue();
   ucTempAtual     = tabela_temp[iRawTempAtual];
   ucDezTempAtual  = ucTempAtual/10;
   ucUnTempAtual   = ucTempAtual%10;
}
```

```
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:       sensTemp.h                                        */
/*                                                                            */
/*   Descricao:             Declaracao das funcoes implementadas no arquivo   */
/*                      sensTemp.c                                            */
/*                                                                            */
/*   Autores:               Gustavo Lino e Giacomo A. Dollevedo               */
/*   Criado em:             29/07/2020                                        */
/*   Ultima revisao em:     29/07/2020                                        */
/* ************************************************************************** */

#ifndef SOURCES_SENSTEMP_
#define SOURCES_SENSTEMP_

/* ************************************************************************** */
/* Nome do metodo:          sensTemp_init                                     */
/* Descricao:               Inicializa as funcoes do ADC para sensor de temperatura */
/*                                                                            */
/* Parametros de entrada:   n/a                                               */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void sensTemp_init(void);


/* ************************************************************************** */
/* Nome do metodo:          readTemp                                          */
/* Descricao:               Le o ADC conectado ao sensor de temperatura e     */
/*                      converte o valor pela lookup table em graus Celsius */
/*                                                                            */
/* Parametros de entrada:   n/a                                               */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void readTemp(void);


#endif
```

```
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:        tacometro.c                                      */
/*                                                                            */
/*   Descricao:              Arquivo contendo as funcoes de interface do uC   */
/*                        com o encoder do kit, para leitura da rotacao do    */
/*                        cooler                                              */
/*                                                                            */
/*   Autores:                Gustavo Lino e Giacomo Dollevedo                 */
/*   Criado em:              08/05/2020                                       */
/*   Ultima revisao em:      31/07/2020                                       */
/* ************************************************************************** */

/* REVISÃƒO: */
/* ALTERADO A LIBERACAO DO CLOCK PARA PORTA "E" ["SCGC6 -> SCGC5"]*/
/* ALTERADA A MASCARA "TPM0_CLOCK_GATE" no board.h */
/* ALTERADO PARA "&=" no CLOCK DIVIDER */
/* ALTERADA A FORMA COM QUE RPM EH CALCULADA*/

#include "board.h"
#include "tacometro.h"


/* ************************************************************************** */
/* Nome do metodo:          tachometer_init                                   */
/* Descricao:               Inicializa os registradores para funcionamento do TPM0  */
/*                       como contador de pulsos                              */
/*                                                                            */
/* Parametros de entrada:    n/a                                              */
/*                                                                            */
/* Parametros de saida:      n/a                                              */
/* ************************************************************************** */
void tachometer_init(){



 /*Liberar Clock para TPM 0*/
 SIM_SCGC6 |= TPM0_CLOCK_GATE;

 /*Configurar o divisor de clock em 1*/
 TPM0_SC  &= CLOCK_DIVIDER_1;

 /*Liberar o Clock para porta E (encoder)*/
 SIM_SCGC5 |= PORTE_CLOCK_GATE;

 /*Configurar o pino PTE29 como external clock (ALT4) e o CLKIN0 como entrada*/
   PORTE_PCR29  |= MUX_ALT4;
   SIM_SOPT4  &= TPM0CLKSEL_AS_CLKIN0;

 /*Configurar contador para clock externo*/
 TPM0_SC |= TPM_EXTERNAL_CLOCK;


}


/* ************************************************************************** */
/* Nome do metodo:          tachometer_readSensor                             */
/* Descricao:               Le a velocidade do cooler (RPM) e a retorna       */
/*                                                                            */
/* Parametros de entrada:    uiPeriod -> periodo da janela de contagem (LPTMR0)    */
/*                                                                            */
/* Parametros de saida:      Um unsigned int indicando a rotacao (RPM) do cooler   */
/* ************************************************************************** */
unsigned int tachometer_readSensor(unsigned int uiPeriod){

 /*Numero de pulsos contados*/
```

```
 unsigned int uiCounted = TPM0_CNT;
 /*Reseta o contador*/
 TPM0_CNT &= CLEAR_16;

 /*7 pas => A cada 7 pulsos contados, temos 1 rotacao completa*/
 unsigned int uiRotations = uiCounted/7;

 /*Convertendo a leitura na janela para RPM*/
 unsigned int uiCoolerRps = uiRotations/(uiPeriod*1000000);
 unsigned int uiCoolerRpM = uiCoolerRps*60;

 return uiCoolerRpM;


}
```

```
/* ********************************************************************************** */
/*                                                                                    */
/*   Nome do arquivo:        tacometro.h                                              */
/*                                                                                    */
/*   Descricao:              Arquivo Header contendo a declaracao                     */
/*                         das funcoes de interface do microcontrolador               */
/*                         com o encoder do kit, para leitura da rotacao do           */
/*                         cooler                                                     */
/*                                                                                    */
/*   Autores:                Gustavo Lino e Giacomo Dollevedo                         */
/*   Criado em:              08/05/2020                                               */
/*   Ultima revisao em:      10/05/2020                                               */
/* ********************************************************************************** */

#ifndef SOURCES_TACHOMETER_
#define SOURCES_TACHOMETER_


/* ********************************************************************************** */
/* Nome do metodo:          tachometer_init                                           */
/* Descricao:               Inicializa os registradores para funcionamento do TPM0    */
/*                        como contador de pulsos                                     */
/*                                                                                    */
/* Parametros de entrada:    n/a                                                      */
/*                                                                                    */
/* Parametros de saida:      n/a                                                      */
/* ********************************************************************************** */
void tachometer_init(void);


/* ********************************************************************************** */
/* Nome do metodo:          tachometer_readSensor                                     */
/* Descricao:               Le a velocidade do cooler (RPM) e a retorna               */
/*                                                                                    */
/* Parametros de entrada:    uiPeriod -> periodo da janela de contagem (LPTMR0)       */
/*                                                                                    */
/* Parametros de saida:      Um unsigned int indicando a rotacao (RPM) do cooler      */
/* ********************************************************************************** */
unsigned int tachometer_readSensor(unsigned int uiPeriod);


#endif /* SOURCES_COOLER_HEATER_ */
```

```c
/* ************************************************************** */
/* File name:        UART.c                                       */
/* File description: Debugging through UART interface             */
/* Author name:      dloubach, rbacurau                           */
/* Creation date:    22out2015                                    */
/* Revision date:    01mai2020                                    */
/* ************************************************************** */

/* definition include */
#include "UART.h"

/* system includes */
#include "fsl_clock_manager.h"
#include "fsl_device_registers.h"
#include "fsl_port_hal.h"
#include "fsl_smc_hal.h"
#include "fsl_debug_console.h"
#include "communicationStateMachine.h"

/* UART definitions */
#ifndef BOARD_DEBUG_UART_INSTANCE
  #define BOARD_DEBUG_UART_INSTANCE   0
  #define BOARD_DEBUG_UART_BASEADDR   UART0
#endif
#ifndef BOARD_DEBUG_UART_BAUD
  #define BOARD_DEBUG_UART_BAUD       115200
#endif


/* ********************************************** */
/* Method name:        UART0_init                 */
/* Method description: Initialize the UART0 as debug*/
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void UART0_init (void)
{
  /* UART0 */
  /* UART0_RX */
  PORT_HAL_SetMuxMode(PORTA, 1u, kPortMuxAlt2);
  /* UART0_TX */
  PORT_HAL_SetMuxMode(PORTA, 2u, kPortMuxAlt2);

  /* Select the clock source for UART0 */
  SIM_SOPT2 |= 0x4000000;

  /* Init the debug console (UART) */
  DbgConsole_Init(BOARD_DEBUG_UART_INSTANCE, BOARD_DEBUG_UART_BAUD, kDebugConsoleLPSCI);
}

/* ********************************************** */
/* Method name:        UART0_enableIRQ            */
/* Method description: Enable the interruption for  */
/*                     serial port inputs and     */
/*                     prepare the buffer         */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void UART0_enableIRQ(void)
{
  /* Enable interruption in the NVIC */
  NVIC_EnableIRQ(UART0_IRQn);

  /* Enable receive interrupt (RIE) in the  UART module */
  UART0_C2 |= 0x20;
}
```

```
/* ********************************************** */
/* Method name:        UART0_IRQHandler           */
/* Method description: Serial port interruption   */
/*                     handler method. It Reads the */
/*                     new character and saves in  */
/*                     the buffer                  */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void UART0_IRQHandler(void)
{
 // Solicita o processo de tratamento de byte recebido pela UART
 processByteCommUART(debug_getchar());

}
```

```c
/* ************************************************************** */
/* File name:        UART.h                                       */
/* File description: Debugging through UART interface             */
/* Author name:      dloubach, rbacurau                           */
/* Creation date:    22out2015                                    */
/* Revision date:    01mai2020                                    */
/* ************************************************************** */

#ifndef UART_H_
#define UART_H_

/* ********************************************** */
/* Method name:        UART_init                  */
/* Method description: Initialize the UART0       */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void UART0_init(void);

/* ********************************************** */
/* Method name:        UART0_enableIRQ            */
/* Method description: Enable the interruption for  */
/*                     serial port inputs and     */
/*                     prepare the buffer         */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void UART0_enableIRQ(void);


/* ********************************************** */
/* Method name:        UART0_IRQHandler           */
/* Method description: Serial port interruption     */
/*                     handler method. It Reads the */
/*                     new character and saves in   */
/*                     the buffer                 */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void UART0_IRQHandler(void);

#endif /* UART_H_ */
```

```
/* ************************************************************** */
/* File name:        util.c                                       */
/* File description: This file has a couple of useful functions to   */
/*                   make programming more productive             */
/*                                                                */
/*                   Remarks: The soft delays consider            */
/*                   core clock @ 40MHz                           */
/*                   bus clock @ 20MHz                            */
/*                                                                */
/* Author name:      dloubach                                     */
/* Creation date:    09jan2015                                    */
/* Revision date:    31jul2020                                    */
/* ************************************************************** */

#include "util.h"

/* ********************************************** */
/* Method name:        util_genDelay088us         */
/* Method description: generates ~ 088 micro sec  */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void util_genDelay088us(void)
{
   char i;
   for(i=0; i<120; i++)
   {
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
   }
}



/* ********************************************** */
/* Method name:        util_genDelay250us         */
/* Method description: generates ~   250 micro sec  */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void util_genDelay250us(void)
{
   char i;
   for(i=0; i<120; i++)
   {
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
      __asm("NOP");
```

```
        __asm("NOP");
        __asm("NOP");
    }
    util_genDelay088us();
    util_genDelay088us();
}



/* ********************************************** */
/* Method name:        util_genDelay1ms           */
/* Method description: generates ~   1 mili sec   */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void util_genDelay1ms(void)
{
    util_genDelay250us();
    util_genDelay250us();
    util_genDelay250us();
    util_genDelay250us();
}


/* ********************************************** */
/* Method name:        util_genDelay10ms          */
/* Method description: generates ~   10 mili sec  */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void util_genDelay10ms(void)
{
    util_genDelay1ms();
    util_genDelay1ms();
    util_genDelay1ms();
    util_genDelay1ms();
    util_genDelay1ms();
    util_genDelay1ms();
    util_genDelay1ms();
    util_genDelay1ms();
    util_genDelay1ms();
    util_genDelay1ms();
}



/* ********************************************** */
/* Method name:        util_genDelay100ms         */
/* Method description: generates ~   100 mili sec */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void util_genDelay100ms(void)
{
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
}
```

```
/* ************************************************************************** */
/* Nome do metodo:          extrai_digito                                     */
/* Descricao:               Extrai digitos de um numero e armazena num vetor  */
/*                                                                            */
/*                                                                            */
/* Parametros de entrada:   numero -> numero que os digitos serao extraidos   */
/*       digitos -> vetor onde os digitos serao armazenados                   */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void extrai_digito(unsigned int numero, unsigned char* digitos){

    unsigned char i = 0;
    unsigned int x = 1;
    unsigned int sobra = 0;

    /*Ja inicializa cada digito com "0" (tabela ASCII)*/
    digitos[0] = 48;
    digitos[1] = 48;
    digitos[2] = 48;
    digitos[3] = 48;

    /*Checar quantos digitos tem no numero*/
    while(numero > x){
        x *= 10;
        i++;
    }

    x /= 10;

    /*Divide por uma potencia de 10 para sobrar um digito so*/
    /*Extrai o digito e armazena no vetor +48 (ASCII)       */
    while(i > 0){
        sobra = numero/x;
        digitos[4-i] = (sobra + 48);

        numero -= (sobra*x);
        x /= 10;



        i--;

    }

    return;

}
```

```
/* ************************************************************** */
/* File name:        util.h                                       */
/* File description: Header file containing the function/methods  */
/*                   prototypes of util.c                         */
/*                   Those delays were tested under the following:*/
/*                   core clock @ 40MHz                           */
/*                   bus clock @ 20MHz                            */
/* Author name:      dloubach                                     */
/* Creation date:    09jan2015                                    */
/* Revision date:    09jun2020                                    */
/* ************************************************************** */

#ifndef UTIL_H
#define UTIL_H

/* ********************************************** */
/* Method name:        util_genDelay088us         */
/* Method description: generates ~ 088 micro sec  */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void util_genDelay088us(void);


/* ********************************************** */
/* Method name:        util_genDelay250us         */
/* Method description: generates ~  250 micro sec */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void util_genDelay250us(void);


/* ********************************************** */
/* Method name:        util_genDelay1ms           */
/* Method description: generates ~   1 mili sec   */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void util_genDelay1ms(void);


/* ********************************************** */
/* Method name:        util_genDelay10ms          */
/* Method description: generates ~   10 mili sec  */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void util_genDelay10ms(void);

/* ********************************************** */
/* Method name:        util_genDelay10ms          */
/* Method description: generates ~   100 mili sec */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void util_genDelay100ms(void);

/* ********************************************************************** */
/* Nome do metodo:          extrai_digito                                 */
/* Descricao:               Extrai digitos de um numero e armazena num vetor  */
/*                                                                        */
/*                                                                        */
/* Parametros de entrada:   numero -> numero que os digitos serao extraidos  */
/*       digitos -> vetor onde os digitos serao armazenados  */
/*                                                                        */
/* Parametros de saida:     n/a                                           */
```

```c
/* ************************************************************************ */
void extrai_digito(unsigned int numero, unsigned char* digitos);

#endif /* UTIL_H */
```

```
/* ********************************************************************************** */
/*                                                                                    */
/*   Nome do arquivo:        variaveis_globais.c                                      */
/*                                                                                    */
/*   Descricao:            ContÃŠm as variaveis globais que sÃŁo alteradas e          */
/*                       acessadas constantemente                                     */
/*                                                                                    */
/*   Autores:                Gustavo Lino e Giacomo A. Dollevedo                      */
/*   Criado em:               27/07/2020                                              */
/*   Ultima revisao em:      31/07/2020                                               */
/* ********************************************************************************** */

unsigned char ucTempAlvo   = 30;
unsigned char ucTempAtual  = 0;
unsigned char ucDezTempAlvo    = 3;
unsigned char ucUnTempAlvo     = 0;
unsigned char ucUnTempAtual    = 0;
unsigned char ucDezTempAtual   = 0;
float fDutyCycle_Heater = 0;
float fKp = 0;
float fKi = 0;
float fKd = 0;
```

```
/* ********************************************************************************** */
/*                                                                                    */
/*   Nome do arquivo:        variaveis_globais.h                                      */
/*                                                                                    */
/*   Descricao:            declara o acesso externos das variaveis globais que sÃ£o */
/*                       acessadas constantemente                                     */
/*                                                                                    */
/*   Autores:                Gustavo Lino e Giacomo A. Dollevedo                      */
/*   Criado em:               27/07/2020                                              */
/*   Ultima revisao em:      28/07/2020                                               */
/* ********************************************************************************** */




#ifndef VARIAVEIS_GLOBAIS_H_
#define VARIAVEIS_GLOBAIS_H_

extern unsigned char ucTempAlvo;
extern unsigned char ucTempAtual;
extern unsigned char ucDezTempAlvo;
extern unsigned char ucUnTempAlvo;
extern unsigned char ucUnTempAtual;
extern unsigned char ucDezTempAtual;
extern float fDutyCycle_Heater;
extern float fKp;
extern float fKi;
extern float fKd;

#endif /* VARIAVEIS_GLOBAIS_H_ */
```

```
/* ************************************************************************** */
/*                                                                            */
/*   Nome do arquivo:       main.c                                            */
/*                                                                            */
/*   Descricao:           Projeto final do controlador de temperatura com     */
/*                      interface                                             */
/*                                                                            */
/*   Autores:             Gustavo Lino e Giacomo A. Dollevedo                 */
/*   Criado em:             27/07/2020                                        */
/*   Ultima revisao em:     28/07/2020                                        */
/* ************************************************************************** */

/* Incluindo bibliotecas */
#include "board.h"
#include "util.h"
#include "lut_adc_3v3.h"
#include "lptmr.h"
#include "lcdTemp.h"
#include "ledSwi.h"
#include "pid.h"
#include "display7Temp.h"
#include "uart.h"
#include "sensTemp.h"
#include "variaveis_globais.h"
#include "stdio.h"
#include "aquecedorECooler.h"

/*****************************************************************************/
/*******************          DECLARANDO CONSTANTES       *********************/

/*Temperatura*/
#define TEMP_DEFAULT 30

/*Tick base do timer (em microsegundos [us])*/
#define TICK_4MS     4000

/*Estados principais do sistema*/
#define CONFIG      0
#define CONTROLE    1

/*Subestados CONFIG*/
#define UNIDADE     0
#define DEZENA      1

/*Subestados UNIDADE e DEZENA*/
#define STATE_0     0
#define STATE_N     1
#define STATE_9     2

/*Constante de ganho do controlador*/
#define FKP         0.001f
#define FKI         0.002f
#define FKD         0.003f

/*******************          FIM DAS CONSTANTES          *********************/
/*****************************************************************************/

/*Variaveis para operacao do sistema*/
unsigned char ucEstado      = 0;
unsigned char ucSubestado1  = 0;
unsigned char ucSubestado2  = 0;


/*Variaveis para manter controle do tempo durante execucao*/
unsigned char ucContador1       = 0;
unsigned char ucContador2       = 0;
unsigned char ucContadorCtrl    = 0;
```

```
unsigned char ucSegundos       = 0;
unsigned char ucMinutos        = 0;
unsigned char ucIdleTime       = 0;

/*Variaveis relacionadas aos displays*/
unsigned char ucD7Flag      = 0;
unsigned char ucLCDFrame    = 1;
unsigned char ucDisableD7   = 0;

/*Variavel para disparar o controle*/
unsigned char ucAttCtrl     = 0;

/* ************************************************************************** */
/* Nome do metodo:         timerAtt                                           */
/* Descricao:              Callback da interrupcao gerada pelo timer ltpmr0   */
/*                         para controlar os displays e demais elementos      */
/*                         sensiveis ao tempo                                 */
/*                                                                            */
/* Parametros de entrada:  n/a                                                */
/*                                                                            */
/* Parametros de saida:    n/a                                                */
/* ************************************************************************** */
void timerAtt(){

   ucContador1++;
   ucContador2++;
   ucD7Flag = 1;
}


/* ************************************************************************** */
/* Nome do metodo:         checkTime                                          */
/* Descricao:              Atualiza os contadores de tempo de execucao do     */
/*                         programa                                           */
/*                                                                            */
/* Parametros de entrada:  n/a                                                */
/*                                                                            */
/* Parametros de saida:    n/a                                                */
/* ************************************************************************** */
void checkTime(){

   /*+1 a cada 100ms*/
   if(25 <= ucContador2){
      ucContador2 = 0;
      ucContadorCtrl++;
   }

   /*+1 a cada segundo*/
   if(250 <= ucContador1){
      ucContador1 = 0;
      ucSegundos++;
   }

   /*+1 a cada minuto*/
   if(60 <= ucSegundos){
      ucSegundos = 0;
      ucMinutos++;

      if(CONFIG == ucEstado){
         ucIdleTime++;
      }
   }
}


/* ************************************************************************** */
/* Nome do metodo:         boardInit                                          */
```

```
/* Descricao:              Inicializa os perifericos necessarios para o       */
/*                      projeto                                               */
/*                                                                            */
/* Parametros de entrada:   n/a                                               */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
void boardInit()
{

   /*Inicializa o primeiro LED e os 3 ultimos botoes*/
   /*Botao 2 => "-"; Botao 3 => "+"; Botao 4 => "OK/Reset"*/
 ledSwi_init(1, 0, 0, 0);

   /*Inicializa o ADC para leitura do sensor de temperatura*/
   sensTemp_init();

   /*Inicializa o Display de 7 segmentos*/
   /*O D7S sera usado para exibir a temperatura atual*/
   display7Temp_init();

   /*Inicializa o display LCD*/
   lcdTemp_init();

   /*Inicializa um timer com tick de 4ms para atualizacao dos displays*/
   /*e controle de tempo interno do sistema*/
   tc_installLptmr0(TICK_4MS, timerAtt);

   /*Inicializa o controlador PID para atuar sobre o aquecedor*/
   pid_init();
   pid_setKi(fKi);
   pid_setKp(fKp);
   pid_setKd(fKp);

   /*Inicializa a comunicaÃ§Ã£o UART*/
   UART0_init();

   /*Inicializa o PWM, o ventilador e o aquecedor*/
   PWM_init();
   coolerfan_init();
   heater_init();
}


/* ************************************************************************** */
/* Nome do metodo:         main                                               */
/* Descricao:              Executa o loop principal do programa               */
/*                                                                            */
/*                                                                            */
/* Parametros de entrada:   n/a                                               */
/*                                                                            */
/* Parametros de saida:     n/a                                               */
/* ************************************************************************** */
int main(void){

   /*Inicializa todos os perifericos do Kit*/
   boardInit();


   /*Definiciao dos estados iniciais*/
   ucEstado       = CONFIG;
   ucSubestado1   = DEZENA;
   ucSubestado2   = STATE_0;


   /*Loop infinito de operacao do sistema*/
   while(1){
```

```
/*Faz a contagem de tempo de acordo com a interrupcao programada*/
checkTime();

/*Atualiza o LCD com o frame correto*/
/*1 -> CONFIG; 2 -> CONTROLE*/
if(0 != ucLCDFrame){
   ucDisableD7 = 1;
   showLCDdisp(ucLCDFrame);
   attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
   ucLCDFrame  = 0;
   ucDisableD7 = 0;
}

/*Atualiza o D7S quando hÃ¡ interrupcao && quando o LCD nao esta sendo operado*/
if(0 != ucD7Flag && 0 == ucDisableD7){
   attDisp7Temp();
}

/*Se o sistema fica inoperado por 2 minutos (no estado CONFIG), uma temperatura padrao eh*/
/*setada, e o sistema passa para o estado de CONTROLE*/
if(2 == ucIdleTime){
   ucIdleTime      = 0;
   ucTempAlvo      = TEMP_DEFAULT;
   ucDezTempAlvo   = ucTempAlvo/10;
   ucUnTempAlvo    = ucTempAlvo%10;
   ucEstado        = CONTROLE;
   ucSubestado1    = DEZENA;
   ucSubestado2    = STATE_0;
}

if(1 <= ucContadorCtrl){
   readTemp();
   ucContadorCtrl = 0;
   ucAttCtrl++;
}

/*INICIA A MAQUINA DE ESTADOS DO SISTEMA*/
switch (ucEstado){

/*ESTADO DE CONFIGURACAO DA TEMPERATURA ALVO*/
case CONFIG:
   /*LED 1 indica que o estado atual eh de configuracao*/
   turnOnLED(1);
   ucLCDFrame = 0;

   if(0 == ucDezTempAlvo && DEZENA == ucSubestado1)
       ucSubestado2 = STATE_0;

   else if(9 == ucDezTempAlvo && DEZENA == ucSubestado1)
      ucSubestado2 = STATE_9;

   else
      if(DEZENA == ucSubestado1)
         ucSubestado2 = STATE_N;

   switch(ucSubestado1){

   /*ESTADO PARA CONFIGURACAO DO DIGITO DA DEZENA*/
   case DEZENA:
      switch(ucSubestado2){
      /*ESTADO PARA DIGITO == 0*/
      case STATE_0:
         /*Caso "OK" pressionado*/
         if(1 == readSwitch(4)){
```

```
            util_genDelay100ms();
            ucSubestado1 = UNIDADE;

            if(0 == ucUnTempAlvo)
                ucSubestado2 = STATE_0;

            else if(9 == ucUnTempAlvo)
                ucSubestado2 = STATE_9;

            else
                ucSubestado2 = STATE_N;

            break;
        }

        /*Caso "+" pressionado*/
        else if(1 == readSwitch(3)){
            util_genDelay100ms();
            ucDezTempAlvo++;
            ucDisableD7 = 1;
            attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
            ucDisableD7 = 0;

            ucSubestado2 = STATE_N;
            break;
        }

        /*Caso "-" pressionado*/
        else if(1 == readSwitch(2)){
            util_genDelay100ms();
            ucDisableD7 = 1;
            attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
            ucDisableD7 = 0;
            ucSubestado2 = STATE_0;
            break;
        }

    /*ESTADO PARA 0 < DIGITO < 9*/
    case STATE_N:
        /*Caso "OK" pressionado*/
        if(1 == readSwitch(4)){
            util_genDelay100ms();
            ucSubestado1 = UNIDADE;

            if(0 == ucUnTempAlvo)
                ucSubestado2 = STATE_0;

            else if(9 == ucUnTempAlvo)
                ucSubestado2 = STATE_9;

            else
                ucSubestado2 = STATE_N;
            break;
        }

        /*Caso "+" pressionado*/
        else if(1 == readSwitch(3)){
            util_genDelay100ms();
            if(8 == ucDezTempAlvo){
                ucDezTempAlvo++;

                ucDisableD7 = 1;
                attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
                ucDisableD7 = 0;
                ucSubestado2 = STATE_9;
            }
```

```
            else{
                ucDezTempAlvo++;

                ucDisableD7 = 1;
                attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
                ucDisableD7 = 0;
                ucSubestado2 = STATE_N;
            }
            break;
        }

        /*Caso "-" pressionado*/
        else if(1 == readSwitch(2)){
            util_genDelay100ms();

            if(1 == ucUnTempAlvo){
                ucDezTempAlvo--;
                ucDisableD7 = 1;
                attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
                ucDisableD7 = 0;
                ucSubestado2 = STATE_0;
            }

            else{
                ucDezTempAlvo--;
                ucDisableD7 = 1;
                attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
                ucDisableD7 = 0;
                ucSubestado2 = STATE_N;
            }
            break;
        }

    /*ESTADO PARA DIGITO == 9*/
    case STATE_9:
        /*Caso "OK" pressionado*/
        if(1 == readSwitch(4)){
            util_genDelay100ms();
            ucSubestado1 = UNIDADE;

            if(0 == ucUnTempAlvo)
                ucSubestado2 = STATE_0;

            else if(9 == ucUnTempAlvo)
                ucSubestado2 = STATE_9;

            else
                ucSubestado2 = STATE_N;

            break;
        }

        /*Caso "+" pressionado*/
        else if(1 == readSwitch(3)){
            /*Nada acontece*/
            util_genDelay100ms();
            ucDisableD7 = 1;
            attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
            ucDisableD7 = 0;
            ucSubestado2 = STATE_9;
            break;
        }

        /*Caso "-" pressionado*/
        else if(1 == readSwitch(2)){
```

```
            util_genDelay100ms();
            ucDezTempAlvo--;
            ucDisableD7 = 1;
            attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
            ucDisableD7 = 0;
            ucSubestado2 = STATE_N;
            break;
         }
      }

      break; /*Break do Subestado1*/


   /*ESTADO PARA CONFIGURACAO DO DIGITO DA UNIDADE*/
   case UNIDADE:
      switch(ucSubestado2){
      /*ESTADO PARA DIGITO == 0*/
      case STATE_0:
         /*Caso "OK" pressionado*/
         if(1 == readSwitch(4)){
            util_genDelay100ms();
            ucSubestado1 = DEZENA;
            ucSubestado2 = STATE_0;
            ucEstado     = CONTROLE;
            ucLCDFrame   = 2;
            ucTempAlvo = ((10*ucDezTempAlvo)+ucUnTempAlvo);
            break;
         }

         /*Caso "+" pressionado*/
         else if(1 == readSwitch(3)){
            util_genDelay100ms();
            ucUnTempAlvo++;
            ucDisableD7 = 1;
            attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
            ucDisableD7 = 0;
            ucSubestado2 = STATE_N;
            break;
         }

         /*Caso "-" pressionado*/
         else if(1 == readSwitch(2)){
            util_genDelay100ms();
            ucDisableD7 = 1;
            attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
            ucDisableD7 = 0;
            ucSubestado2 = STATE_0;
            break;
         }

      /*ESTADO PARA 0 < DIGITO < 9*/
      case STATE_N:
         /*Caso "OK" pressionado*/
         if(1 == readSwitch(4)){
            util_genDelay100ms();
            ucSubestado1 = DEZENA;
            ucSubestado2 = STATE_0;
            ucEstado     = CONTROLE;
            ucLCDFrame   = 2;

            if(9 == ucDezTempAlvo)
               ucUnTempAlvo = 0;

            ucTempAlvo = ((10*ucDezTempAlvo)+ucUnTempAlvo);

            break;
```

```
        }

        /*Caso "+" pressionado*/
        else if(1 == readSwitch(3)){
            util_genDelay100ms();
            if(8 == ucUnTempAlvo){
                ucUnTempAlvo++;
                ucDisableD7 = 1;
                attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
                ucDisableD7 = 0;
                ucSubestado2 = STATE_9;
            }

            else{
                ucUnTempAlvo++;
                ucDisableD7 = 1;
                attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
                ucDisableD7 = 0;
                ucSubestado2 = STATE_N;
            }
            break;
        }

        /*Caso "-" pressionado*/
        else if(1 == readSwitch(2)){
            util_genDelay100ms();

            if(1 == ucUnTempAlvo){
                ucUnTempAlvo--;
                ucDisableD7 = 1;
                attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
                ucDisableD7 = 0;
                ucSubestado2 = STATE_0;
            }

            else{
                ucUnTempAlvo--;
                ucDisableD7 = 1;
                attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
                ucDisableD7 = 0;
                ucSubestado2 = STATE_N;
            }
            break;
        }

    /*ESTADO PARA DIGITO == 9*/
    case STATE_9:
        /*Caso "OK" pressionado*/
        if(1 == readSwitch(4)){
            util_genDelay100ms();
            ucSubestado1 = DEZENA;
            ucSubestado2 = STATE_0;
            ucEstado     = CONTROLE;
            ucLCDFrame   = 2;

            if(9 == ucDezTempAlvo)
                ucUnTempAlvo = 0;

            ucTempAlvo = ((10*ucDezTempAlvo)+ucUnTempAlvo);
            break;
        }

        /*Caso "+" pressionado*/
        else if(1 == readSwitch(3)){
            util_genDelay100ms();
            ucDisableD7 = 1;
```

```
                attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
                ucDisableD7 = 0;
                ucSubestado2 = STATE_9;
                break;
            }

            /*Caso "-" pressionado*/
            else if(1 == readSwitch(2)){
                util_genDelay100ms();
                ucUnTempAlvo--;
                ucDisableD7 = 1;
                attTempAlvo(ucDezTempAlvo, ucUnTempAlvo);
                ucDisableD7 = 0;
                ucSubestado2 = STATE_N;
                break;
            }
        }

        break; /*Break do Subestado1*/


    }

    break; /*Break switch Estado*/

/*ESTADO DE CONTROLE DE TEMPERATURA*/
case CONTROLE:
    /*Habilita as interrupcoes da porta Serial*/
    UART0_enableIRQ();

    /*Apaga o led que indica o estado de configuracao*/
    turnOffLED(1);

    /*NOVA ATUALIZAÃ‡ÃƒO DE CONTROLE NECESSÃ�RIA*/
    if(0 != ucAttCtrl) {

     fDutyCycle_Heater = pidUpdateData(ucTempAtual, ucTempAlvo, fDutyCycle_Heater);
        heater_PWMDuty(fDutyCycle_Heater);
    }

    /*INDICACAO VISUAL SE A TEMPERATURA ESTA ACIMA DO SETPOINT*/
    if(ucTempAtual > ucTempAlvo){
        turnOffLED(2);
        turnOffLED(3);
        turnOnLED(3);
        coolerfan_PWMDuty(0.5);

    }

    /*INDICACAO VISUAL SE A TEMPERATURA ESTA ABAIXO DO SETPOINT*/
    else if(ucTempAtual < ucTempAlvo){
        turnOffLED(2);
        turnOffLED(3);
        turnOnLED(2);
        coolerfan_PWMDuty(0);
    }

    /*INDICACAO VISUAL SE A TEMPERATURA ESTA NO SETPOINT*/
    else if(ucTempAtual == ucTempAlvo){
        turnOnLED(2);
        turnOnLED(3);
        coolerfan_PWMDuty(0);
    }

    /*DEVE RETORNAR PARA O MENU DE CONFIGURACAO CASO O BOTAO OK SEJA PRESSIOANDO*/
    if(1 == readSwitch(4)){
```

```
            util_genDelay100ms();
            ucSubestado1    = DEZENA;
            ucSubestado2    = STATE_0;
            ucEstado        = CONFIG;
            ucLCDFrame      = 1;
            ledSwi_init(1, 0, 0, 0);
            break;
        }


        break; /*Break switch Estado*/


    }
  }
}
```