```c
/* ********************************************************************** */
/* Nome do Arquivo:      main.c                                           */
/* Descricao do arquivo: Este arquivo inicializa a placa                  */
/*                       fazendo a inicializacao do clock e do PWM        */
/*                                                                        */
/*                       Caracteristicas do processador MKL25Z128VLK4     */
/*                       48 MHz ARM Cortex-M0+ core                       */
/*                       128 KB program flash memory                      */
/*                       16 KB SRAM                                       */
/*                       Voltage range: 1.71 to 3.6 V                     */
/* Nome dos autores:     Gustavo Moraes/Cassio Dezotti                    */
/* RA:                   174217/168988                                    */
/* Data de criacao:      03abril2020                                      */
/* Data da revisao:      29jul2020                                        */
/* ********************************************************************** */

/* our includes */
#include "board.h"
#include "mcg.h"
#include "aquecedorECooler.h"
#include "adc.h"
#include "UART.h"
#include "print_scan.h"
#include "communicationStateMachine.h"
#include "lcd.h"
#include "lptmr.h"
#include "pid.h"
#include "core_cm0plus.h"
#include "ledSwi.h"
#include "interruptButton.h"

#define MAX_VALUE_LENGHT 7

unsigned char ucAnswer[MAX_VALUE_LENGHT];
unsigned char ucEnable;
unsigned char ucTempAtual[4];
float fHDuty;
float fCDuty;
int iValorTempAtual = 0;
unsigned int uiSpTempertura;
unsigned char ucPeriodo = 0x64;
int iBotoesInit[4] = {0,0,0,0};
int iFlagSetTemp = 0;
int iFlagSetK = 0;

void pidTask(void)
{
    int iSensorValue,iSetPoint;
    float fActuatorValue;
```

```
    lerTemp();
    iSensorValue = iValorTempAtual;
    iSetPoint = uiSpTempertura;
    fActuatorValue = pidUpdateData(iSensorValue,iSetPoint);
    definirDutyH(fActuatorValue/100);
}

/* ********************************************** */
/* Nome da funcao:        iniciarPlaca            */
/* Descricao da funcao:   Inicia a placa e todos os */
/*                        componentes necessarios   */
/*                                                  */
/* parametros de entrada: n/a                      */
/* parametros de saida:   n/a                      */
/* ********************************************** */
void iniciarPlaca(void)
{
    /* Configuracao e inicializacao do clock */
    mcg_clockInit();

    /* Iniciando os perifericos */
    adc_initADCModule();
    adc_initConvertion();
    lcd_initLcd();
    UART0_init();
    coolerfan_init();
    heater_init();
    pid_init();
    iniciarLedSwi(iBotoesInit);
}

/* ********************************************************************** */
/* Nome da funcao:             main                                       */
/* Descricao da funcao:        Inicializa as variaveis e faz o controle   */
/*                             das chamadas das funções para a realizacao */
/*                             da tarefa proposta                         */
/* parametros de entrada:      n/a                                        */
/* parametros de saida:        n/a                                        */
/* ********************************************************************** */
int main(void)
{
    char cMensagem1[] = "KP: ";
    char cMensagem2[] = "KI: ";
    char cMensagem3[] = "KD: ";
    char cMensagem4[] = "Temperatura atual: ";
    char cMensagem5[] = "Temperatura desejada: ";
    char *c;
    char cAux[] = " ";
    float fAuxKp, fAuxKi, fAuxKd = 0;
    int iAuxTemp, iAuxSP, iAux, iAux2;
```

```
    int iCount = 1;

    /* Definimos como maior prioridade a interrupcao do
     * LPTMR, pois ela controla o controlador PID que
     * necessita de uma periodicidade de 100ms para
     * funcionar corretamente. Definimos as interrupcoes dos
     * botes e Uart como a mesma prioridade pois assumimos que
     * o usuario nao ira acionar as duas ao mesmo tempo.
     */
    NVIC_SetPriority(LPTMR0_IRQn,0);
    NVIC_SetPriority(PORTA_IRQn,1);
    NVIC_SetPriority(UART0_IRQn,1);

    /* iniciamos os componentes */
    iniciarPlaca();

    /* Valores iniciais para K, como iniciamos o controlador com os K = 0,
     * ele permanece sem executar nenhuma funcao ate receber algum valor para
K.
     * Assim primeiro passo do usuario deve ser a sintonizacao dos
controladores.
     */
    pid_setKp(1.0);
    pid_setKi(1.0);
    pid_setKd(1.0);

    /* Habilitando interrupcoes */
    tc_installLptmr0(100000,pidTask);
    UART0_enableIRQ();
    interruptButton_enableIRQ();

    while(1){

        fAuxKp = pid_getKp();
        fAuxKd = pid_getKd();
        fAuxKi = pid_getKi();
        /* clear LCD */
        lcd_sendCommand(CMD_CLEAR);
        /* set the cursor line 0, column 1 */
        lcd_setCursor(0,1);
        c = cMensagem4;
        lcd_writeText(0,c);
        lerTemp();
        /*separa dezena de unidade*/
        iAuxTemp = (iValorTempAtual/10)+48;
        cAux[0]=(char)iAuxTemp;
        iAuxTemp = (iValorTempAtual%10)+48;
        /*converte int para char*/
        cAux[1]=(char)iAuxTemp;
        /* Escreve a temperatura no LCD */
```

```
c = cAux;
lcd_writeText(1,c);

while(iFlagSetTemp){
    iAuxTemp = uiSpTempertura;
    /* Desabilitamos as interrupcoes dos botoes pois eles tem outras
     * funcoes nessa parte do codigo
     */
    interruptButton_disableIRQ();
    while(!lerChave(1)){

        /* clear LCD */
        lcd_sendCommand(CMD_CLEAR);
        /* set the cursor line 0, column 1 */
        lcd_setCursor(0,1);
        c = cMensagem5;
        lcd_writeText(0,c);
        /*separa dezena de unidade*/
        iAuxSP = (iAuxTemp/10)+48;
        cAux[0]=(char)iAuxSP;
        iAuxSP = (iAuxTemp%10)+48;
        /*converte int para char*/
        cAux[1]=(char)iAuxSP;
        /* Escreve o set point no LCD */
        c = cAux;
        lcd_writeText(1,c);

        if(lerChave(2)){
            iAuxTemp = iAuxTemp+1;
        }
        if(lerChave(3)){
           iAuxTemp = iAuxTemp+10;
        }
        if(lerChave(4)){
           iAuxTemp = iAuxTemp-1;
        }
    }
    uiSpTempertura = iAuxTemp;
    interruptButton_enableIRQ();
    iFlagSetTemp = 0;
}

while(iFlagSetK){

    /* Desabilitamos as interrupcoes dos botoes pois eles tem outras
     * funcoes nessa parte do codigo
     */
    interruptButton_disableIRQ();
    while(!lerChave(1)){
```

```
while(iCount == 1){
    /* clear LCD */
    lcd_sendCommand(CMD_CLEAR);
    /* set the cursor line 0, column 1 */
    lcd_setCursor(0,1);
    c = cMensagem1;
    lcd_writeText(0,c);
    /*separa dezena de unidade*/
    fAuxKp = pid_getKp();
    iAux  = fAuxKp;
    iAux2 = (iAux/10)+48;
    cAux[0]=(char)iAux2;
    iAux2 = (iAux%10)+48;
    /* converte int para char */
    cAux[1]=(char)iAux2;
    /* Escreve o Kp no LCD */
    cAux[2]=(char)46;
    iAux = fAuxKp*100;
    iAux2 = (iAux%100);
    iAux2 = (iAux/10)+48;
    cAux[3]=(char)iAux;
    iAux2 = (iAux%10)+48;
    cAux[4]=(char)iAux;
    c = cAux;
   /* Escreve o KP no LCD */
   lcd_writeText(1,c);

   if(lerChave(2)){
       fAuxKp = fAuxKp+0.1;
   }
   if(lerChave(3)){
       fAuxKp = fAuxKp+1;
   }
   if(lerChave(4)){
       fAuxKp = fAuxKp-0.1;
   }
}
while(iCount == 2){
    lcd_sendCommand(CMD_CLEAR);
    /* set the cursor line 0, column 1 */
    lcd_setCursor(0,1);
    c = cMensagem2;
    lcd_writeText(0,c);
    /*separa dezena de unidade*/
    fAuxKi = pid_getKi();
    iAux  = fAuxKi;
    iAux2 = (iAux/10)+48;
    cAux[0]=(char)iAux2;
    iAux2 = (iAux%10)+48;
    /* converte int para char */
```

```
        cAux[1]=(char)iAux2;
        /* Escreve o Ki no LCD */
        cAux[2]=(char)46;
        iAux = fAuxKi*100;
        iAux2 = (iAux%100);
        iAux2 = (iAux/10)+48;
        cAux[3]=(char)iAux;
        iAux2 = (iAux%10)+48;
        cAux[4]=(char)iAux;
        c = cAux;

        /* Escreve o KI no LCD */
        lcd_writeText(1,c);

        if(lerChave(2)){
            fAuxKi = fAuxKi+0.1;
        }
        if(lerChave(3)){
            fAuxKi = fAuxKi+1;
        }
        if(lerChave(4)){
            fAuxKi = fAuxKi-0.1;
        }
    }
    while(iCount == 3){
        /* clear LCD */
        lcd_sendCommand(CMD_CLEAR);
        /* set the cursor line 0, column 1 */
        lcd_setCursor(0,1);
        c = cMensagem3;
        lcd_writeText(0,c);
        /*separa dezena de unidade*/
        fAuxKd = pid_getKd();
        iAux  = fAuxKd;
        iAux2 = (iAux/10)+48;
        cAux[0]=(char)iAux2;
        iAux2 = (iAux%10)+48;
        /* converte int para char */
        cAux[1]=(char)iAux2;
        /* Escreve a temperatura no LCD */
        cAux[2]=(char)46;
        iAux = fAuxKd*100;
        iAux2 = (iAux%100);
        iAux2 = (iAux/10)+48;
        cAux[3]=(char)iAux;
        iAux2 = (iAux%10)+48;
        cAux[4]=(char)iAux;
        c = cAux;
        /* Escreve o KD no LCD */
        lcd_writeText(1,c);
```

```
                    if(lerChave(2)){
                        fAuxKd = fAuxKd+0.1;
                    }
                    if(lerChave(3)){
                        fAuxKd = fAuxKd+1;
                    }
                    if(lerChave(4)){
                        fAuxKd = fAuxKd-0.1;
                    }
                }
            }

            iCount = iCount+1;
            /* se o usuario ja setou Kp Ki e Kd, atualizamos os ganhos
             * habilitamos a interrupcao dos botoes e resetamos a flag
             * de set dos K.
             */
            if(iCount == 4){
                pid_setKp(fAuxKp);
                pid_setKi(fAuxKi);
                pid_setKd(fAuxKd);
                interruptButton_enableIRQ();
                iFlagSetK = 0;
            }
        }
    }
}
```

```
/* ************************************************************** */
/* 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, IagoAF 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

/* ************************************************ */
/* 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_CFG1_CLK_DIVIDER) | ADC_CFG1_ADLPC(ADC_CFG1_LOW_POWER));

   /*
   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(ADC_CFG2_MUX_SELECT));
}


/* ************************************************ */
/* 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 */
/* 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                                                */

#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 do arquivo: Arquivo que construi as funcoes que controlam o  */
/*                       PWM e os atuadores                      */
/*                                                               */
/* Nome dos autores:     Gustavo Moraes/Cassio Dezotti           */
/* RA:                   174217/168988                           */
/* Data de criacao:      24abril2020                             */
/* Data da revisao:      27jul2020                               */
/* ************************************************************* */

/* our includes */
#include "aquecedorECooler.h"
#include "board.h"

extern unsigned char ucPeriodo;
extern float fHDuty;
extern float fCDuty;

/* ************************************************************* */
/* Nome da funcao:        PWM_init                               */
/* Descricao da funcao:   Essa função inicializa o PWM e os parâmetros */
/*                        necessários como o clock e contador.   */
/*                                                               */
/* Parametros de entrada: n/a                                    */
/* Parametros de saida:   n/a                                    */
/* ************************************************************* */
void PWM_init()
{
    /* liberar o clock para o timer */
    SIM_SCGC6 |= 0x2000000;

    /* escolhendo divisor 32 para o clock */
    TPM1_SC |= 0x05;

    /* fonte de clock seleciona o mcgirclk (32KHz) para o TPM */
    SIM_SOPT2 |= 0x3000000;

    /*Configurar o contador do clock como up counting e a cada pulso */
    TPM1_SC &= ~(0x030);
    TPM1_SC |= 0x8;

    /*configurar o periodo do PWM para 10 Hz com o contador
     * estourando em 100 (MOD+1) MOD = 99 (0x0063)
     */

    /* Configura o estouro do contador como Periodo - 1 = 0x0063 (99) */
    TPM1_MOD |= 0x0063;

    /* Configura o TPM como Edge aligned com High-true pulses */
```

```
    TPM1_C0SC &= ~(0x14);
    TPM1_C0SC |= 0x28;
    TPM1_C1SC &= ~(0x14);
    TPM1_C1SC |= 0x28;
}

/* ********************************************************************** */
/* Nome da funcao:        coolerfan_init                                  */
/* Descricao da funcao:   Função que habilita o clock, a porta do atuador */
/*                        e configura o pino como PWM                     */
/*                                                                        */
/* Parametros de entrada: n/a                                             */
/* Parametros de saida:   n/a                                             */
/* ********************************************************************** */
void coolerfan_init()
{
    /* Habilitando o clock e selecionando como PWM o MUX de cada pino */
    SIM_SCGC5 |= 0x200;
    PORTA_PCR13 &= ~(0x400);
    PORTA_PCR13 |= 0x300;
    definirDutyC(0.5);
}

/* ********************************************************************** */
/* Nome da funcao:        heater_init                                     */
/* Descricao da função:   Função que habilita o clock, a porta do atuador */
/*                        e configura o pino 12 como PWM                  */
/*                                                                        */
/* Parametros de entrada: n/a                                             */
/* Parametros de saida:   n/a                                             */
/* ********************************************************************** */
void heater_init()
{
    /* Habilitando o clock e selecionando como PWM o MUX de cada pino */
    SIM_SCGC5 |= 0x200;
    PORTA_PCR12 &= ~(0x400);
    PORTA_PCR12 |= 0x300;
    definirDutyH(0);
}

/* ********************************************************************** */
/* Nome da funcao:        definirDutyC                                    */
/* Descricao da funcao:   Recebe um numero que corresponde ao Duty Cycle  */
/*                        desejado para o cooler e altera o Duty Cycle do PWM
/*                                                                        */
/*                                                                        */
```

```
/* Parametros de entrada: Recebe um float entre 0 e 1
*/
/* Parametros de saida:   n/a
*/
/* ************************************************************************
*/
void definirDutyC(float fCoolerDuty)
{

    fCDuty = fCoolerDuty;

    /* Define o novo valor para o Duty Cycle do cooler */
    if(fCoolerDuty > 0 && fCoolerDuty < 1)
    {
        TPM0_C0V |= convertDuty(fCoolerDuty);
    }
}

/*
************************************************************************
*/
/* Nome da funcao:        definirDutyH
*/
/* Descricao da funcao:   Recebe um numero que corresponde ao Duty Cycle
*/
/*                        desejado para o aquecedor e altera o Duty Cycle do
PWM */
/*
*/
/* Parametros de entrada: Recebe um float entre 0 e 1
*/
/* Parametros de saida:   n/a
*/
/*
************************************************************************
*/
void definirDutyH(float fHeaterDuty)
{

    if (fHeaterDuty > 0.5){
        fHeaterDuty = 0.5;
    }

    fHDuty = fHeaterDuty;

    /* Define o novo valor para o Duty Cycle do aquecedor */
    if(fHeaterDuty > 0.5 && fHeaterDuty < 1)
    {
        TPM0_C0V |= convertDuty(fHeaterDuty);
    }
```

```
}

/* ************************************************************** */
/* Nome da funcao:        convertDuty                             */
/* Descricao da funcao:   Função de apoio recebe o valor do Duty Cycle */
/*                        e converte para um numero de 16 bits    */
/*                                                                */
/* Parametros de entrada: Recebe um float entre 0 e 1             */
/* Parametros de saida:   Retorna um unsigned char com o valor do Duty */
/*                        Cycle nos primeiros 16 bits             */
/* ************************************************************** */
unsigned char convertDuty(float fDuty)
{
    unsigned char ucValor = 0;
    int iValor = fDuty*10000;

    /* Converte o float para um valor de 0 a periodo em 16 bits */
    iValor = iValor*ucPeriodo;
    iValor = iValor/10000;
    ucValor |= (0xFFFF & iValor);

    return(ucValor);
}
```

```
/* Nome do Arquivo:      aquecedorECooler.h                                  */

#ifndef SOURCES_AQUECEDORECOOLER_H_
#define SOURCES_AQUECEDORECOOLER_H_

/* ****************************************************************** */
/* Nome da funcao:        PWM_init                                    */
/* Descricao da funcao:   Essa função inicializa o PWM e os parâmetros */
/*                        necessários como o clock e contador.        */
/*                                                                    */
/* Parametros de entrada: n/a                                         */
/* Parametros de saida:   n/a                                         */
/* ****************************************************************** */
void PWM_init();

/* ****************************************************************** */
/* Nome da funcao:        coolerfan_init                              */
/* Descricao da funcao:   Função que habilita o clock, a porta do atuador */
/*                        e configura o pino como PWM                 */
/*                                                                    */
/* Parametros de entrada: n/a                                         */
/* Parametros de saida:   n/a                                         */
/* ****************************************************************** */
void coolerfan_init();

/* ****************************************************************** */
/* Nome da funcao:        heater_init                                 */
/* Descricao da função:   Função que habilita o clock, a porta do atuador */
/*                        e configura o pino 12 como PWM              */
/*                                                                    */
/* Parametros de entrada: n/a                                         */
/* Parametros de saida:   n/a                                         */
/* ****************************************************************** */
void heater_init();

/* ********************************************************************
*/
/* Nome da funcao:        definirDutyC
*/
/* Descricao da funcao:   Recebe um numero que corresponde ao Duty Cycle
*/
/*                        desejado para o cooler e altera o Duty Cycle do PWM
*/
/*
*/
/* Parametros de entrada: Recebe um float entre 0 e 1
*/
/* Parametros de saida:   n/a
*/
/* ********************************************************************
```

```
*/
void definirDutyC(float fCoolerDuty);

/*
*********************************************************************
*/
/* Nome da funcao:        definirDutyH
*/
/* Descricao da funcao:   Recebe um numero que corresponde ao Duty Cycle
*/
/*                        desejado para o aquecedor e altera o Duty Cycle do
PWM */
/*
*/
/* Parametros de entrada: Recebe um float entre 0 e 1
*/
/* Parametros de saida:   n/a
*/
/*
*********************************************************************
*/
void definirDutyH(float fHeaterDuty);

/* ************************************************************** */
/* Nome da funcao:        convertDuty                             */
/* Descricao da funcao:   Função de apoio recebe o valor do Duty Cycle */
/*                        e converte para um numero de 16 bits    */
/*                                                                */
/* Parametros de entrada: Recebe um float entre 0 e 1             */
/* Parametros de saida:   Retorna um unsigned char com o valor do Duty */
/*                        Cycle nos primeiros 16 bits             */
/* ************************************************************** */
unsigned char convertDuty(float duty);

#endif /* SOURCES_AQUECEDORECOOLER_H_ */
```

```
/* ************************************************************** */
/* File name:        board.h                                      */
/* File description: Header file containing the peripherals mapping  */
/*                   of the peripheral board for the ES670 hardware  */
/* Author name:      Rodrigo M Bacurau/Gustavo M./Cassio D.        */
/* Creation date:    26fev2020                                    */
/* Revision date:    28jul2020                                    */
/* ************************************************************** */

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

#ifndef SOURCES_BOARD_H_
#define SOURCES_BOARD_H_

/*                 General uC definitions                 */

/*                  Definicoes dos botoes                 */
#define LEDSWI1_PORT   PORTA /* peripheral port base pointer */
#define LEDSWI1_GPIO   PTA   /* peripheral gpio base pointer */
#define LEDSWI1_PIN          (uint32_t) 1u
#define LEDSWI2_PORT   PORTA /* peripheral port base pointer */
#define LEDSWI2_GPIO   PTA   /* peripheral gpio base pointer */
#define LEDSWI2_PIN          (uint32_t) 2u
#define LEDSWI3_PORT   PORTA /* peripheral port base pointer */
#define LEDSWI3_GPIO   PTA   /* peripheral gpio base pointer */
#define LEDSWI3_PIN          (uint32_t) 4u
#define LEDSWI4_PORT   PORTA /* peripheral port base pointer */
#define LEDSWI4_GPIO   PTA   /* peripheral gpio base pointer */
#define LEDSWI4_PIN          (uint32_t) 5u

/*                 ATUADORES definitions                  */
#define THERMOMETER_PORT_BASE_PNT       PORTE
#define THERMOMETER_GPIO_BASE_PNT       PTE

#define THERMOMETER_PIN                 21U
#define THERMOMETER_DIR                 (GPIO_INPUT << THERMOMETER_PIN)
#define THERMOMETER_ALT                 0x00U

#define ATUADORES_PORT_BASE_PNT          PORTA
/* peripheral port base pointer */
#define ATUADORES_PWM_BASE_PNT           PTA
/* peripheral gpio base pointer */

#define COOLER_PIN                       13U
/* register selector */

#define AQUECEDOR_PIN                    12U
/* register selector */
```

```
#define  CGC_CLOCK_DISABLED         0x00U
#define  CGC_CLOCK_ENABLED          0x01U

/*                 END OF ATUADORES definitions                 */


/* LCD Register Selector
 * Used as register selector input
 * When (LCD_RS = LCD_RS_HIGH) => DATA register is selected
 * When (LCD_RS = LCD_RS_LOW)  => INSTRUCTION register is selected
*/
#define LCD_PORT_BASE_PNT           PORTC                                   /*
peripheral port base pointer */
#define LCD_GPIO_BASE_PNT           PTC                                     /*
peripheral gpio base pointer */

#define LCD_RS_PIN                  8U                                      /*
register selector */
#define LCD_RS_DIR                  (GPIO_OUTPUT << LCD_RS_PIN)
#define LCD_RS_ALT                  kPortMuxAsGpio

#define LCD_ENABLE_PIN              9U                                      /*
enable pin */
#define LCD_ENABLE_DIR              (GPIO_OUTPUT << LCD_ENABLE_PIN)
#define LCD_ENABLE_ALT              kPortMuxAsGpio

#define LCD_RS_HIGH                 1U
#define LCD_RS_DATA                 LCD_RS_HIGH

#define LCD_RS_LOW                  0U
#define LCD_RS_CMD                  LCD_RS_LOW

#define LCD_ENABLED                 1U
#define LCD_DISABLED                0U

#define LCD_DATA_DIR                kGpioDigitalOutput                      /*
LCD data pins */
#define LCD_DATA_ALT                kPortMuxAsGpio

#define LCD_DATA_DB0_PIN            0u
#define LCD_DATA_DB1_PIN            1u
#define LCD_DATA_DB2_PIN            2u
#define LCD_DATA_DB3_PIN            3U
#define LCD_DATA_DB4_PIN            4U
#define LCD_DATA_DB5_PIN            5U
#define LCD_DATA_DB6_PIN            6U
#define LCD_DATA_DB7_PIN            7U

#define LCD_DATA_DB0_DIR            (GPIO_OUTPUT << LCD_DATA_DB0_PIN)
#define LCD_DATA_DB1_DIR            (GPIO_OUTPUT << LCD_DATA_DB1_PIN)
```

```
#define LCD_DATA_DB2_DIR            (GPIO_OUTPUT << LCD_DATA_DB2_PIN)
#define LCD_DATA_DB3_DIR            (GPIO_OUTPUT << LCD_DATA_DB3_PIN)
#define LCD_DATA_DB4_DIR            (GPIO_OUTPUT << LCD_DATA_DB4_PIN)
#define LCD_DATA_DB5_DIR            (GPIO_OUTPUT << LCD_DATA_DB5_PIN)
#define LCD_DATA_DB6_DIR            (GPIO_OUTPUT << LCD_DATA_DB6_PIN)
#define LCD_DATA_DB7_DIR            (GPIO_OUTPUT << LCD_DATA_DB7_PIN)
/*                 END OF LCD definitions                */

/*                 UART definitions                */

#define BOARD_DEBUG_UART_INSTANCE   0
#define BOARD_DEBUG_UART_BASEADDR   UART0
#define BOARD_DEBUG_UART_BAUD       115200

#define UART_PORT                   PORTA
#define UART_ALT                    kPortMuxAlt2
#define UART_PIN_1                  1u
#define UART_PIN_2                  2u

/*                 END OF UART definitions                */

#endif /* SOURCES_BOARD_H_ */
```

```c
/* ********************************************************************** */
/* Nome do Arquivo:      communicationStateMachine.c                      */
/* Descricao do arquivo: Desenvolve a maquina de estados e as outras funcoes */
/*                       que cuidam da comunicacao                        */
/* Nome dos autores:     Gustavo M./Cassio D.                             */
/* RA:                   174217/168988                                    */
/* Data de criacao:      03jun2020                                        */
/* Data da revisao:      28jul2020                                        */
/* ********************************************************************** */

/* definition include */
#include "communicationStateMachine.h"
#include "board.h"
#include "aquecedorECooler.h"
#include "adc.h"
#include "lut_adc_3v3.h"
#include "pid.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"

#define IDDLE '0'
#define READY '1'
#define GET '2'
#define SET '3'
#define PARAM '4'
#define VALUE '5'
#define MAX_VALUE_LENGHT 7

typedef union{
    unsigned char ucBytes[4];
    float fReal;
}floatUCharType;

typedef union{
    unsigned char ucBytes[4];
    int iReal;
}
intUCharType;

typedef union{
    unsigned char ucBytes[4];
    int iReal;
}
intSetPointCharType;
```

```c
unsigned char ucUARTState = IDDLE;
unsigned char ucValueCount = '0';
extern int iValorTempAtual;
extern unsigned char ucAnswer[MAX_VALUE_LENGHT];
extern unsigned char ucEnable;
extern unsigned char ucTempAtual[4];
extern float fHDuty;
extern float fCDuty;
extern unsigned int uiSpTempertura;

/* ************************************************ */
/* Nome da funcao:        processamentoByte         */
/* Descricao da funcao:   Funcao contendo maquina de */
/*                        estados para o processamento */
/*                        de bytes.                 */
/*                                                  */
/* Parametros de entrada: um char representando o byte */
/*                        recebido                  */
/* Parametros de saida:   n/a                       */
/* ************************************************ */
void processamentoByte(unsigned char ucByte)
{
    static unsigned char ucParam;
    static unsigned char ucValue[MAX_VALUE_LENGHT];

    /* Toda mensagem comeca com '#'*/
    if('#' == ucByte)
    {
        ucUARTState = READY;
    }else
    {
        if(IDDLE != ucUARTState)
        {
            switch(ucUARTState)
            {
                 case READY:
                    /* As opcoes sao get 'g' e set 's' */
                    switch(ucByte)
                    {
                        case 'g':
                            ucUARTState = GET;
                            break;
                        case 's':
                            ucUARTState = SET;
                            break;
                        default:
                            ucUARTState = IDDLE;
                    }
                    break;
                /*
```

```
                * Para get, temos a opcao de temperatura 't',
                * duty do aquecedor 'a' e duty do cooler 'c',
                * 'p', 'i' e 'd' para os valores de Kp, Ki e Kd
                * respectivamente
                */
                case GET:
                    if('t' == ucByte || 'a' == ucByte || 'c' == ucByte || 'i'
== ucByte || 'p' == ucByte || 'd' == ucByte)
                    {
                        ucParam = ucByte;
                        ucUARTState = PARAM;
                    }else
                    {
                        ucUARTState = IDDLE;
                    }
                    break;
                   /*
                    * Para set, temos a opcao de temperatura 't',
                    * 'i' para setar o Ki, 'p' para setar o Kp e 'd'
                    * para setar o Kd
                    */
                case SET:
                   if('t' == ucByte || 'i' == ucByte || 'p' == ucByte || 'd'
== ucByte)
                   {
                       ucParam = ucByte;
                       ucUARTState = VALUE;
                       ucValueCount = 0;
                       }else
                       {
                           ucUARTState = IDDLE;
                       }
                       break;
                case PARAM:
                   if(';' == ucByte)
                   {
                       returnParam(ucParam);
                   }
                   ucUARTState = IDDLE;
                   break;
                /*
                 * Verificamos se o byte recedido e um numero
                 * ou uma virgula para o float.
                */
                case VALUE:
                   if((ucByte >= '0' && ucByte <= '9') || ',' == ucByte)
                   {
                       if(ucValueCount < MAX_VALUE_LENGHT)
                       {
                           ucValue[ucValueCount++] = ucByte;
```

```
                    }
                }else
                {
                    if(';' == ucByte)
                    {
                        ucValue[ucValueCount] = '\0';
                        setParam(ucParam,ucValue);
                     }
                    ucUARTState = IDDLE;
                }
                break;
            }
        }
    }
}

/* ************************************************ */
/* Nome da funcao:       returnParam                */
/* Descricao da funcao:  Funcaco para retornar a    */
/*                       resposta solicitada por um */
/*                       comando get                */
/*                                                  */
/* Parametros de entrada: um char podendo ser 't' para */
/*                        temperatura, 'a' para duty */
/*                        cycle do aquecedor ou 'c' */
/*                        para o duty cycle do cooler */
/* Parametros de saida:   n/a                       */
/* ************************************************ */
void returnParam(unsigned char ucParam)
{
    /*
     * Colocamos o valor #a no comeco de toda mensagem de retorno
     * para identificar a mensagem como resporta, anwser
     */

    ucAnswer[0] = 0x23;
    ucAnswer[1] = 0x61;

    switch(ucParam)
    {
        case 't':
            /*Le temperatura e armazena no vetor de answer*/
            lerTemp();
            break;
        case 'a':
            /*Le duty cycle do aquecedor e armazena no vetor de answer*/
            lerHeaterDuty();
            break;
        case 'c':
            /*Le duty cycle do cooler e armazena no vetor de answer*/
```

```
            lerCoolerFanDuty();
            break;
        case 'p':
            /*Le Kp e armazena no vetor de answer*/
            lerKp();
            break;
        case 'i':
            /*Le Ki e armazena no vetor de answer*/
            lerKi();
            break;
        case 'd':
            /*Le Kd e armazena no vetor de answer*/
            lerKd();
            break;
        default:
            break;
    }

    /* Envia a resposta de volta para a UART */
    for(int i=0;i<MAX_VALUE_LENGHT;i++){
        debug_putchar(ucAnswer[i]);
    }
}

/* ************************************************ */
/* Nome da funcao:        setParam                  */
/* Descricao da funcao:   Funcao para setar parametros */
/*                                                  */
/* Parametros de entrada: recebe um vetor de char com  */
/*                        o valor para o set, e um     */
/*                        char escolhendo o parametro  */
/*                        a ser alterado, 't' para     */
/*                        temperatura e 'e' para       */
/*                        habilitar ou desabilitar a   */
/*                        interface local              */
/* Parametros de saida:   n/a                       */
/* ************************************************ */
void setParam(unsigned char ucParam,unsigned char ucValue[MAX_VALUE_LENGHT])
{
    float fAux;
    switch(ucParam){
    /*converte o valor de temperatura desejada de char para int e armazena na
     * variavel global
     */
    case 't':

        for(int i=3; i<MAX_VALUE_LENGHT; i++){
            uiSpTempertura = convertChar2Int(ucValue[i]);
        }
        break;
```

```
    /*converte o valor de Kp desejada de char para float e chama a
     * funcao de set
     */
    case 'p':
        for(int i=3; i<MAX_VALUE_LENGHT; i++){
            fAux = convertChar2Float(ucValue[i]);
        }
        pid_setKp(fAux);
        break;
    /*converte o valor de Ki desejada de char para float e chama a
     * funcao de set
     */
    case 'i':
        for(int i=3; i<MAX_VALUE_LENGHT; i++){
            fAux = convertChar2Float(ucValue[i]);
        }
        pid_setKi(fAux);
        break;
     /*converte o valor de Kd desejada de char para float e chama a
      * funcao de set
      */
    case 'd':
       for(int i=3; i<MAX_VALUE_LENGHT; i++){
            fAux = convertChar2Float(ucValue[i]);
       }
       pid_setKd(fAux);
       break;
    }
}

/* ************************************************** */
/* Nome da funcao:        lerTemp                     */
/* Descricao da funcao:   Funcao para ler temperatura */
/*                        sera reformulada nas proximas */
/*                        etapas                      */
/*                                                    */
/* Parametros de entrada: n/a                         */
/* Parametros de saida:   n/a                         */
/* ************************************************** */
void lerTemp()
{
    unsigned char ucSendChar, ucCount;
    unsigned char *ucAux;

    /* inicia a conversao AD e espera terminar */
    adc_initConvertion();
    while(!adc_isAdcDone())
    {
```

```
    }

    /* pega o valor correspondente a tabela e converte de int para char*/
    iValorTempAtual = tabela_temp[adc_getConvertionValue()];
    ucAux = convertInt2Char(iValorTempAtual);

    /* armazena o valor na variavel answer*/
    for (ucCount= 0; ucCount< 4; ucCount++)
    {
        ucSendChar= ucAux[ucCount];
        ucAnswer[ucCount+2] = ucSendChar;
    }
    ucAnswer[6] = 0x3b;
}

/* ************************************************** */
/* Nome da funcao:        heater_PWMDuty              */
/* Descricao da funcao:   Funcao para ler duty cycle do */
/*                        aquecedor, sera reformulada   */
/*                        nas proximas etapas           */
/*                                                      */
/* Parametros de entrada: n/a                         */
/* Parametros de saida:   n/a                         */
/* ************************************************** */
void lerHeaterDuty()
{
    unsigned char *ucHeaterDuty;

    /* le a variavel global de duty, converte de float para char e armazena na
anwser */
    ucHeaterDuty = convertFloat2Char(fHDuty);
    for(int i=0;i<4;i++){
        ucAnswer[i+2] = ucHeaterDuty[i];
    }
    ucAnswer[6] = 0x3b;
}

/* ************************************************** */
/* Nome da funcao:        coolerFan_PWMDuty           */
/* Descricao da funcao:   Funcao para ler duty cycle do */
/*                        cooler, sera reformulada nas  */
/*                        proximas etapas               */
/*                                                      */
/* Parametros de entrada: n/a                         */
/* Parametros de saida:   n/a                         */
/* ************************************************** */
void lerCoolerFanDuty()
{
    unsigned char *ucCoolerDuty;
```

```
    /* le a variavel global de duty, converte de float para char e armazena na
anwser */
    ucCoolerDuty = convertFloat2Char(fCDuty);

    for(int i=0;i<4;i++){
         ucAnswer[i+2] = ucCoolerDuty[i];
    }
    ucAnswer[6] = 0x3b;
}

/* ************************************************** */
/* Nome da funcao:        lerKp                       */
/* Descricao da funcao:   Funcao para ler o KP setado no */
/*                        controlador                 */
/*                                                    */
/* Parametros de entrada: n/a                         */
/* Parametros de saida:   n/a                         */
/* ************************************************** */
void lerKp()
{
    unsigned char *ucKp;

    /* le o valor de Kp, converte de float para char e armazena na anwser */
    ucKp = convertFloat2Char(pid_getKp());

    for(int i=0;i<4;i++){
        ucAnswer[i+2] = ucKp[i];
    }
    ucAnswer[6] = 0x3b;
}

/* ************************************************** */
/* Nome da funcao:        lerKi                       */
/* Descricao da funcao:   Funcao para ler o KI setado no */
/*                        controlador                 */
/*                                                    */
/* Parametros de entrada: n/a                         */
/* Parametros de saida:   n/a                         */
/* ************************************************** */
void lerKi()
{
    unsigned char *ucKi;

    /* le o valor de Ki, converte de float para char e armazena na anwser */
    ucKi = convertFloat2Char(pid_getKi());

    for(int i=0;i<4;i++){
    ucAnswer[i+2] = ucKi[i];
    }
    ucAnswer[6] = 0x3b;
```

```
}

/* ************************************************** */
/* Nome da funcao:        lerKd                       */
/* Descricao da funcao:   Funcao para ler o KD setado no */
/*                        controlador                 */
/*                                                    */
/* Parametros de entrada: n/a                         */
/* Parametros de saida:   n/a                         */
/* ************************************************** */
void lerKd()
{
    unsigned char *ucKd;

    /* le o valor de Kd, converte de float para char e armazena na anwser */
    ucKd = convertFloat2Char(pid_getKd());

    for(int i=0;i<4;i++){
        ucAnswer[i+2] = ucKd[i];
    }
    ucAnswer[6] = 0x3b;
}

/* ************************************************** */
/* Nome da funcao:        convertChar2Float           */
/* Descricao da funcao:   Funcao que converte 4 caracteres */
/*                        para um valor float         */
/* Parametros de entrada: caracter                    */
/* Parametros de saida:   valor float                 */
/* ************************************************** */
float convertChar2Float(unsigned char ucReceivedChar)
{
    floatUCharType varFloatUChar;
    static unsigned char ucCount;

    varFloatUChar.ucBytes[ucCount] = ucReceivedChar;
    if(++ucCount>= 4)
    {
        return varFloatUChar.fReal;
        ucCount= 0;
    }
    return (0);
}

/* ************************************************** */
/* Nome da funcao:        convertFloat2Char           */
/* Descricao da funcao:   Funcao que converte um valor float */
/*                        para um valor de 4 caracteres */
/* Parametros de entrada: valor float                 */
/* Parametros de saida:   caracter                    */
```

```c
/* ******************************************************** */
unsigned char* convertFloat2Char(float fReceivedFloat)
{
    floatUCharType varCharUFloat;
    unsigned char *ucAux;

    varCharUFloat.fReal= fReceivedFloat;

    ucAux = varCharUFloat.ucBytes;

    return(ucAux);
}

/* ******************************************************** */
/* Nome da funcao:          convertInt2Char                 */
/* Descricao da funcao:     Funcao que converte um valor int */
/*                          para um valor de 4 caracteres   */
/* Paramentros de entrada: valor int                        */
/* Parametros de saida:     caracter                        */
/* ******************************************************** */
unsigned char* convertInt2Char(int ucReceivedInt)
{
    intUCharType varIntUChar;
    unsigned char *ucAux;

    varIntUChar.iReal= ucReceivedInt;
    ucAux = varIntUChar.ucBytes;

    return(ucAux);
}

/* ******************************************************** */
/* Nome da funcao:         convertChar2Int                  */
/* Descricao da funcao:    Funcao que converte 4 caracteres */
/*                         para um valor int                */
/* Parametros de entrada: caracter                          */
/* Parametros de saida:    valor int                        */
/* ******************************************************** */
int convertChar2Int(unsigned char ucReceivedChar)
{
    intSetPointCharType varChar2int;
    static unsigned char ucCount;

    varChar2int.ucBytes[ucCount] = ucReceivedChar;
    if(++ucCount>= 4)
    {
        return varChar2int.iReal;
        ucCount= 0;
    }
    return (0);
```

```
}
```

```
/* ************************************************************************
*/
/* Nome do Arquivo:      communicationStateMachine.c
*/
/* Descricao do arquivo: Contêm a descrição das funções utilizadas no programa
*/
/* Nome dos autores:     Gustavo M./Cassio D.
*/
/* RA:                   174217/168988
*/
/* Data de criacao:      03jun2020
*/
/* Data da revisao:      09jun2020
*/
/* ************************************************************************
*/

#ifndef COMMUNICATION_STATE_MACHINE_H_
#define COMMUNICATION_STATE_MACHINE_H_

#define MAX_VALUE_LENGHT 7

/* ************************************************ */
/* Nome da funcao:        processamentoByte         */
/* Descricao da funcao:   Funcao contendo maquina de */
/*                        estados para o processamento */
/*                        de bytes.                 */
/*                                                  */
/* Parametros de entrada: um char representando o byte */
/*                        recebido                  */
/* Parametros de saida:   n/a                       */
/* ************************************************ */
void processamentoByte(unsigned char ucByte);

/* ************************************************ */
/* Nome da funcao:       returnParam                */
/* Descricao da funcao:  Funcaco para retornar a    */
/*                       resposta solicitada por um */
/*                       comando get                */
/*                                                  */
/* Parametros de entrada: um char podendo ser 't' para */
/*                        temperatura, 'a' para duty */
/*                        cycle do aquecedor ou 'c' */
/*                        para o duty cycle do cooler */
/* Parametros de saida:   n/a                       */
/* ************************************************ */
void returnParam(unsigned char ucParam);

/* ************************************************ */
/* Nome da funcao:        setParam                  */
```

```
/* Descricao da funcao:   Funcao para setar parametros */
/*                                                     */
/* Parametros de entrada: recebe um vetor de char com  */
/*                        o valor para o set, e um     */
/*                        char escolhendo o parametro  */
/*                        a ser alterado, 't' para     */
/*                        temperatura e 'e' para       */
/*                        habilitar ou desabilitar a   */
/*                        interface local              */
/* Parametros de saida:   n/a                          */
/* ************************************************** */
void setParam(unsigned char ucParam,unsigned char ucValue[MAX_VALUE_LENGHT]);

/* ************************************************** */
/* Nome da funcao:        lerTemp                      */
/* Descricao da funcao:   Funcao para ler temperatura  */
/*                        sera reformulada nas proximas */
/*                        etapas                       */
/*                                                     */
/* Parametros de entrada: n/a                          */
/* Parametros de saida:   n/a                          */
/* ************************************************** */
void lerTemp();

/* ************************************************** */
/* Nome da funcao:        heater_PWMDuty               */
/* Descricao da funcao:   Funcao para ler duty cycle do */
/*                        aquecedor, sera reformulada  */
/*                        nas proximas etapas          */
/*                                                     */
/* Parametros de entrada: n/a                          */
/* Parametros de saida:   n/a                          */
/* ************************************************** */
void lerHeaterDuty();

/* ************************************************** */
/* Nome da funcao:        coolerFan_PWMDuty            */
/* Descricao da funcao:   Funcao para ler duty cycle do */
/*                        cooler, sera reformulada nas */
/*                        proximas etapas              */
/*                                                     */
/* Parametros de entrada: n/a                          */
/* Parametros de saida:   n/a                          */
/* ************************************************** */
void lerCoolerFanDuty();

/* ************************************************** */
/* Nome da funcao:        convertChar2Float            */
/* Descricao da funcao:   Funcao que converte 4 caracteres */
/*                        para um valor float          */
```

```
/* Parametros de entrada: caracter                          */
/* Parametros de saida:   valor float                       */
/* ******************************************************** */
float convertChar2Float(unsigned char ucReceivedChar);

/* ******************************************************** */
/* Nome da funcao:         convertInt2Char                  */
/* Descricao da funcao:    Funcao que converte um valor int */
/*                         para um valor de 4 caracteres    */
/* Paramentros de entrada: valor int                        */
/* Parametros de saida:    caracter                         */
/* ******************************************************** */
unsigned char* convertInt2Char(int ucReceivedInt);

/* ******************************************************** */
/* Nome da funcao:        convertChar2Int                   */
/* Descricao da funcao:   Funcao que converte 4 caracteres  */
/*                        para um valor int                 */
/* Parametros de entrada: caracter                          */
/* Parametros de saida:   valor int                         */
/* ******************************************************** */
int convertChar2Int(unsigned char ucReceivedChar);

/* ******************************************************** */
/* Nome da funcao:        convertFloat2Char                 */
/* Descricao da funcao:   Funcao que converte um valor float*/
/*                        para um valor de 4 caracteres     */
/* Parametros de entrada: valor float                       */
/* Parametros de saida:   n/a                               */
/* ******************************************************** */
unsigned char* convertFloat2Char(float fReceivedFloat);

/* ******************************************************** */
/* Nome da funcao:        lerKp                             */
/* Descricao da funcao:   Funcao para ler o KP setado no    */
/*                        controlador                       */
/*                                                          */
/* Parametros de entrada: n/a                               */
/* Parametros de saida:   n/a                               */
/* ******************************************************** */
void lerKp();

/* ******************************************************** */
/* Nome da funcao:        lerKi                             */
/* Descricao da funcao:   Funcao para ler o KI setado no    */
/*                        controlador                       */
/*                                                          */
/* Parametros de entrada: n/a                               */
/* Parametros de saida:   n/a                               */
/* ******************************************************** */
```

```
void lerKi();

/* ************************************************** */
/* Nome da funcao:        lerKd                       */
/* Descricao da funcao:   Funcao para ler o KD setado no */
/*                        controlador                 */
/*                                                    */
/* Parametros de entrada: n/a                         */
/* Parametros de saida:   n/a                         */
/* ************************************************** */
void lerKd();

#endif /* COMMUNICATION_STATE_MACHINE_H_ */
```

```
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include "fsl_device_registers.h"
#include "fsl_debug_console.h"
#include "fsl_clock_manager.h"
#include "fsl_os_abstraction.h"
#include "print_scan.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

#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

*************************************************************************/

/*! @brief Operation functions definiations for debug console. */
typedef struct DebugConsoleOperationFunctions {
    union {
        void (* Send)(void *base, const uint8_t *buf, uint32_t count);
#if defined(UART_INSTANCE_COUNT)
        void (* UART_Send)(UART_Type *base, const uint8_t *buf, uint32_t
count);
#endif
#if defined(LPUART_INSTANCE_COUNT)
        void (* LPUART_Send)(LPUART_Type* base, const uint8_t *buf, uint32_t
count);
#endif
#if defined(UART0_INSTANCE_COUNT)
        void (* UART0_Send)(UART0_Type* base, const uint8_t *buf, uint32_t
count);
#endif
#if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))
        void (* USB_Send)(uint32_t base, const uint8_t *buf, uint32_t count);
#endif
    } tx_union;
```

```
    union{
        void (* Receive)(void *base, uint8_t *buf, uint32_t count);
#if defined(UART_INSTANCE_COUNT)
        uart_status_t (* UART_Receive)(UART_Type *base, uint8_t *buf, uint32_t
count);
#endif
#if defined(LPUART_INSTANCE_COUNT)
        lpuart_status_t (* LPUART_Receive)(LPUART_Type* base, uint8_t *buf,
uint32_t count);
#endif
#if defined(UART0_INSTANCE_COUNT)
        lpsci_status_t (* UART0_Receive)(UART0_Type* base, uint8_t *buf,
uint32_t count);
#endif
#if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))
        usb_status_t (* USB_Receive)(uint32_t base, uint8_t *buf, uint32_t
count);
#endif

    } rx_union;
} debug_console_ops_t;

/*! @brief State structure storing debug console. */
typedef struct DebugConsoleState {
    debug_console_device_type_t type;/*<! Indicator telling whether the debug
console is inited. */
    uint8_t instance;               /*<! Instance number indicator. */
    void*   base;                   /*<! Base of the IP register. */
    debug_console_ops_t ops;        /*<! Operation function pointers for debug
uart operations. */
} debug_console_state_t;

/*************************************************************************
*
 * Variables

*************************************************************************/
/*! @brief Debug UART state information.*/
static debug_console_state_t s_debugConsole;

/*************************************************************************
*
 * Code

*************************************************************************/
/* See fsl_debug_console.h for documentation of this function.*/
debug_console_status_t DbgConsole_Init(
        uint32_t uartInstance, uint32_t baudRate, debug_console_device_type_t
device)
{
```

```
    if (s_debugConsole.type != kDebugConsoleNone)
    {
        return kStatus_DEBUGCONSOLE_Failed;
    }

    /* Set debug console to initialized to avoid duplicated init operation.*/
    s_debugConsole.type = device;
    s_debugConsole.instance = uartInstance;

    /* 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);
                LPSCI_HAL_EnableReceiver(base);

                /* Set the funciton pointer for send and receive for this kind
of device. */
                s_debugConsole.ops.tx_union.UART0_Send =
LPSCI_HAL_SendDataPolling;
                s_debugConsole.ops.rx_union.UART0_Receive =
LPSCI_HAL_ReceiveDataPolling;
            }
            break;
#endif
#if defined(LPUART_INSTANCE_COUNT)
        case kDebugConsoleLPUART:
            {
                LPUART_Type* g_Base[LPUART_INSTANCE_COUNT] = LPUART_BASE_PTRS;
```

```
                LPUART_Type* base = g_Base[uartInstance];
                uint32_t lpuartSourceClock;

                s_debugConsole.base = base;
                CLOCK_SYS_EnableLpuartClock(uartInstance);

                /* LPUART clock source is either system or bus clock depending
on instance */
                lpuartSourceClock = CLOCK_SYS_GetLpuartFreq(uartInstance);

                /* initialize the parameters of the LPUART config structure
with desired data */
                LPUART_HAL_SetBaudRate(base, lpuartSourceClock, baudRate);
                LPUART_HAL_SetBitCountPerChar(base, kLpuart8BitsPerChar);
                LPUART_HAL_SetParityMode(base, kLpuartParityDisabled);
                LPUART_HAL_SetStopBitCount(base, kLpuartOneStopBit);

                /* finally, enable the LPUART transmitter and receiver */
                LPUART_HAL_SetTransmitterCmd(base, true);
                LPUART_HAL_SetReceiverCmd(base, true);

                /* Set the funciton pointer for send and receive for this kind
of device. */
                s_debugConsole.ops.tx_union.LPUART_Send =
LPUART_HAL_SendDataPolling;
                s_debugConsole.ops.rx_union.LPUART_Receive =
LPUART_HAL_ReceiveDataPolling;

            }
            break;
#endif
        /* If new device is requried as the low level device for debug console,
         * Add the case branch and add the preprocessor macro to judge whether
         * this kind of device exist in this SOC. */
        default:
            /* Device identified is invalid, return invalid device error code.
*/
            return kStatus_DEBUGCONSOLE_InvalidDevice;
    }

    /* Configure the s_debugConsole structure only when the inti operation is
successful. */
    s_debugConsole.instance = uartInstance;

    return kStatus_DEBUGCONSOLE_Success;
}

/* See fsl_debug_console.h for documentation of this function.*/
debug_console_status_t DbgConsole_DeInit(void)
{
```

```
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return kStatus_DEBUGCONSOLE_Success;
    }

    switch(s_debugConsole.type)
    {
#if defined(UART_INSTANCE_COUNT)
        case kDebugConsoleUART:
            CLOCK_SYS_DisableUartClock(s_debugConsole.instance);
            break;
#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;
    }

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

```
}

#pragma weak _read
int _read(int handle, char *buffer, int size)
{
    /* This function only reads from "standard in", for all other file*/
    /* handles it returns failure.*/
    if (handle != 0)
    {
        return -1;
    }

    /* 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, (uint8_t *)buffer,
size);
    return size;
}
#elif defined(__CC_ARM) && !defined(MQX_STDIO)
struct __FILE
{
    int handle;
    /* Whatever you require here. If the only file you are using is */
    /* standard output using printf() for debugging, no file handling */
    /* is required. */
};

/* FILE is typedef in stdio.h. */
#pragma weak __stdout
FILE __stdout;
FILE __stdin;

#pragma weak fputc
int fputc(int ch, FILE *f)
{
    /* 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, (const uint8_t*)&ch,
1);
    return 1;
```

```
}

#pragma weak fgetc
int fgetc(FILE *f)
{
    uint8_t temp;
    /* 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, &temp, 1);
    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();

        if ((result == '\r') || (result == '\n'))
        {
            /* End of Line */
            if (i == 0)
            {
                i = (uint32_t)-1;
            }
            else
            {
                break;
            }
        }

        temp_buf[i + 1] = '\0';
    }

    result = scan_prv(temp_buf, (char *)fmt_ptr, ap);
    va_end(ap);
```

```
   return result;
}

int debug_getchar(void)
{
    unsigned char c;

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
    s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, &c, 1);

    return c;
}
/*******************************************************************************
*
 * EOF

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

```c
/* ********************************************************************** */
/* Nome do Arquivo:      interruptButton.c                                */
/* Descricao do arquivo: Funcoes que configuram as interrupcoes dos botoes */
/* Nome dos autores:     Gustavo M./Cassio D.                             */
/* RA:                   174217/168988                                    */
/* Data de criacao:      03jun2020                                        */
/* Data da revisao:      29jul2020                                        */
/* ********************************************************************** */

/* definition include */
#include "interruptButton.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 "board.h"
#include "core_cm0plus.h"

extern int iFlagSetTemp;
extern int iFlagSetK;

/* ****************************************** */
/* Method name:        interruptButton_init   */
/* Method description: Initialize the buttons */
/* Input params:       n/a                    */
/* Output params:      n/a                    */
/* ****************************************** */
void interruptButton_init (void)
{
    PORTA_PCR1 |= (0XA0000);
    PORTA_PCR2 |= (0XA0000);
}

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

```
/* ********************************************** */
/* Method name:        interruptButton_enableIRQ  */
/* Method description: Disable the interruption for */
/*                     serial port inputs and     */
/*                     prepare the buffer         */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void interruptButton_disableIRQ(void)
{
    /* Enable interruption in the NVIC */
    NVIC_DisableIRQ(PORTA_IRQn);
}

/* ********************************************** */
/* Method name:        PORTA_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 PORTA_IRQHandler(void)
{
    /* Fazer logica para verificacao de flags de cada botao */
    unsigned char ucAux, ucLido;
    ucLido = PORTA_ISFR;

    ucAux = (0x1) & (ucLido >> 1);
    if(ucAux){
        iFlagSetTemp = 1;
        PORTA_ISFR |= 0x2;
    }
    ucAux = (0x1) & (ucLido >> 2);
    if(ucAux){
        iFlagSetK = 1;
        PORTA_ISFR |= 0x4;
    }
}
```

```c
/* ********************************************************** */
/* Nome do Arquivo:      interruptButton.h                    */
/* Descricao do arquivo: Funcoes para as interrupcoes dos botoes */
/* Nome dos autores:     Gustavo M./Cassio D.                 */
/* RA:                   174217/168988                        */
/* Data de criacao:      03jun2020                            */
/* Data da revisao:      29jul2020                            */
/* ********************************************************** */

#ifndef interruptButton_H_
#define interruptButton_H_

/* ****************************************** */
/* Method name:        interruptButton_init   */
/* Method description: Initialize the buttons */
/* Input params:       n/a                    */
/* Output params:      n/a                    */
/* ****************************************** */
void interruptButton_init (void);

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

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

/* ********************************************** */
/* Method name:        PORTA_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 PORTA_IRQHandler(void);
```

```
#endif /* interruptButton_H_ */
```

```
/* ********************************************************************** */
/* Nome do Arquivo:      lcd.c                                            */
/* Descricao do arquivo: Este arquivo e dedicado criar todas as funcoes   */
/*                       do lcd, escrita de comando, escrita de dados e   */
/*                       inicializacao                                    */
/* Nome dos autores:     Gustavo Moraes/Cassio Dezotti                    */
/* RA:                   174217/168988                                    */
/* Data de criacao:      06abril2020                                      */
/* Data da revisao:      09abril2020                                      */
/* ********************************************************************** */

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

/* includes do sistema */
#include "fsl_clock_manager.h"
#include "fsl_port_hal.h"
#include "fsl_gpio_hal.h"

/* linha e coluna zero */
#define LINE0         0U
#define COLUMN0       0U
#define L0C0_BASE     0x80
#define L1C0_BASE     0xC0
#define MAX_COLUMN  15U

/* ********************************************************************** */
/* Nome da funcao:         lcd_initLcd                                    */
/* Descricao da funcao:    Essa funcao inicializa todo o LCD e os parametros */
/*                         necessarios como o clock, a porta e os pinos.  */
/*                                                                        */
/* Parametros de entrada: n/a                                             */
/* Parametros de saida:   n/a                                             */
/* ********************************************************************** */
void lcd_initLcd(void)
{
    unsigned int uiVetorInit = 0;

    /* pins configured as outputs */

    /* un-gate port clock*/
    SIM_SCGC5 |= 0x0800;

    /*
     * set pin as gpio 0 = DB0, 1 = DB1...
     * 8 = RS, 9 = Enable.
     */
    PORTC_PCR8 |= 0X100;
    PORTC_PCR9 |= 0X100;
```

```
    PORTC_PCR0 |= 0X100;
    PORTC_PCR1 |= 0X100;
    PORTC_PCR2 |= 0X100;
    PORTC_PCR3 |= 0X100;
    PORTC_PCR4 |= 0X100;
    PORTC_PCR5 |= 0X100;
    PORTC_PCR6 |= 0X100;
    PORTC_PCR7 |= 0X100;

    /* set pin as digital output */
    for(int i=0;i<10;i++)
    {
        uiVetorInit |= (1 << i);
    }
    /*
     * Apos esse for, temos um dos primeiros 10 pinos setados com 1
     * para passar como output.
     */
    GPIOC_PDDR |= uiVetorInit;

    /* turn-on LCD, with no cursor and no blink */
    lcd_sendCommand(CMD_NO_CUR_NO_BLINK);

    /* init LCD */
    lcd_sendCommand(CMD_INIT_LCD);

    /* clear LCD */
    lcd_sendCommand(CMD_CLEAR);

    /* LCD with no cursor */
    lcd_sendCommand(CMD_NO_CURSOR);

    /* cursor shift to right */
    lcd_sendCommand(CMD_CURSOR2R);
}

/* ************************************************************** */
/* Nome da funcao:        lcd_write2Lcd                           */
/* Descricao da funcao:   funcao que faz a escrita de um dado no LCD. */
/*                                                                */
/* Parametros de entrada: Recebe um caractere de dado ou comando  */
/*                        e o tipo da acao que ela realizarao.    */
/*                        Se 0 --> LCD recebera um comando        */
/*                        Se 1 --> LCD recebera um dado           */
/* Parametros de saida:   n/a                                     */
/* ************************************************************** */
void lcd_write2Lcd(unsigned char ucBuffer,  unsigned char ucDataType)
{
    unsigned char ucAux = 0;
```

```
    /* writing data or command */
    if(LCD_RS_CMD == ucDataType){
       /* will send a command */
       GPIOC_PSOR &= ~(0x17F);
       GPIOC_PCOR |= (1 << LCD_RS_PIN);
    }
    else{
       /* will send data */
       GPIOC_PSOR &= ~(0x17F);
       GPIOC_PSOR |= (1 << LCD_RS_PIN);
    }

    /*
     * Faz um E bit a bit com o caracter armazenado no ucBuffer
     * extraindo os 8 primeiros bit do buffer para a variavel
     * ucAux.
     */
    for(int i=0;i<8;i++)
    {
        ucAux |= (ucBuffer & (1u << i));
    }

    /* Envia os 8 primeiros bits de data para as portas */
    GPIOC_PDOR |= ucAux;

    /*
     *
     *enable, delay, disable LCD
     *this generates a pulse in the enable pin this
     */
    GPIOC_PSOR |= (1 << LCD_ENABLE_PIN);
    util_genDelay1ms();

    GPIOC_PCOR |= (1 << LCD_ENABLE_PIN);

    util_genDelay1ms();
    util_genDelay1ms();
}

/* ************************************************************** */
/* Nome da funcao:        lcd_writeData                           */
/* Descricao da funcao:   funcao de apoio que faz a chamada da    */
/*                        funcao lcd_Write2Lcd a qual escreve no LCD */
/*                                                                */
/* Parametros de entrada: Recebe um valor 0 ou 1                  */
/*                        Se 0 --> LCD recebera um comando        */
/*                        Se 1 --> LCD recebera um dado           */
/* Parametros de saida:   n/a                                     */
/* ************************************************************** */
void lcd_writeData(unsigned char ucData)
```

```
{
    /* just a relay to send data */
    lcd_write2Lcd(ucData, LCD_RS_DATA);
}

/* ************************************************************** */
/* Nome da funcao:        lcd_sendoCommand                        */
/* Descricao da funcao:   funcao que manda um comando para a      */
/*                        funcao lcd_Write2Lcd a qual realiza no LCD */
/*                                                                */
/* Parametros de entrada: Recebe um valor 0 ou 1                  */
/*                        Se 0 --> mandara um comando ao LCD      */
/*                        Se 1 --> mandara um dado ao LCD         */
/* Parametros de saida:   n/a                                     */
/* ************************************************************** */
void lcd_sendCommand(unsigned char ucCmd)
{
    /* just a relay to send command */
    lcd_write2Lcd(ucCmd, LCD_RS_CMD);
}

/* ************************************************************** */
/* Nome da funcao:        lcd_setCursor                           */
/* Descricao da funcao:   Coloca o cursor na linha e coluna recebidas */
/*                        por parametro                           */
/*                                                                */
/* Parametros de entrada: Recebe um valor para a linha e coluna   */
/* Parametros de saida:   n/a                                     */
/* ************************************************************** */
void lcd_setCursor(unsigned char ucLine, unsigned char ucColumn)
{
    char ucCommand;

    if(LINE0 == ucLine)
        /* line 0 */
        ucCommand = L0C0_BASE;
    else
        /* line 1 */
        ucCommand = L1C0_BASE;

    /* maximum MAX_COLUMN columns */
    ucCommand += (ucColumn & MAX_COLUMN);

    /* send the command to set the cursor */

    lcd_sendCommand(ucCommand);
}

/* ************************************************************** */
/* Nome da funcao:        lcd_writeString                         */
```

```
/* Descricao da funcao:   Recebe a string que sera enviada ao */
/*                        o LCD                               */
/*                                                            */
/* Parametros de entrada: A string                            */
/* Parametros de saida:   n/a                                 */
/* ********************************************************** */
void lcd_writeString(const char *cBuffer)
{
    while(*cBuffer)
    {
        lcd_writeData(*cBuffer++);
    }
}

/* ********************************************************************** */
/* Nome da funcao:        lcd_dummyText                                   */
/* Descricao da funcao:   Faz a configuracao da mensagem que sera exibida */
/*                                                                        */
/* 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(0,1);

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

    /* set the cursor line 1, column 0 */

    lcd_setCursor(1,0);
    lcd_writeString("Prj Sis Embarcad");
}

/* ********************************************************************** */
/* Nome da funcao:        lcd_writeText                                   */
/* Descricao da funcao:   Recebe do programador qual mensagem serÃ¡ exibida */
/*                        e em qual linha e coluna do LCD isso ocorrera   */
/*                                                                        */
/* Parametros de entrada: A string e a linha e coluna do LCD              */
/* Parametros de saida:   n/a                                             */
/* ********************************************************************** */
void lcd_writeText(int iLinha, const char *cString)
{
    /*
     * Primeiro analisa se a linha a ser escrita e a primeira
     * ou a segunda, chamando a funcao setCursor para definir onde
```

```
         * a mensagem comeca. Em seguida enviamos a String para ser escrita
         * no LCD
         */
        if(0 == iLinha)
        {
            lcd_setCursor(LINE0,COLUMN0);
            lcd_writeString(cString);
        }
        else if(1 == iLinha)
        {
            lcd_setCursor(1,COLUMN0);
            lcd_writeString(cString);
        }
}
```

```
/* ********************************************************************** */
/* Nome do Arquivo:      lcd.h                                            */
/* Descricao do arquivo: Cabecalho contendo as atribuicoes para as funcoes  */
/*                       utilzadas na lcd.c                               */
/* Nome dos autores:     Gustavo Moraes/Cassio Dezotti                    */
/* RA:                   174217/168988                                    */
/* Data de criacao:      06abril2020                                      */
/* Data da revisao:      09abril2020                                      */
/* ********************************************************************** */

#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 */

/* ********************************************************************** */
/* Nome da funcao:        lcd_initLcd                                     */
/* Descricao da funcao:   Essa funcao inicializa todo o LCD e os parametros */
/*                        necessarios como o clock, a porta e os pinos.   */
/*                                                                        */
/* Parametros de entrada: n/a                                             */
/* Parametros de saida:   n/a                                             */
/* ********************************************************************** */
void lcd_initLcd(void);

/* ********************************************************************** */
/* Nome da funcao:        lcd_write2Lcd                                   */
/* Descricao da funcao:   funcao que faz a escrita de um dado no LCD.     */
/*                                                                        */
/* Parametros de entrada: Recebe um caractere de dado ou comando          */
/*                        e o tipo da acao que ela realizarao.            */
/*                        Se 0 --> LCD recebera um comando                */
/*                        Se 1 --> LCD recebera um dado                   */
/* Parametros de saida:   n/a                                             */
/* ********************************************************************** */
void lcd_write2Lcd(unsigned char ucBuffer,  unsigned char cDataType);

/* ********************************************************************** */
/* Nome da funcao:        lcd_writeData                                   */
/* Descricao da funcao:   funcao de apoio que faz a chamada da            */
/*                        funcao lcd_Write2Lcd a qual escreve no LCD      */
/*                                                                        */
/* Parametros de entrada: Recebe um valor 0 ou 1                          */
/*                        Se 0 --> LCD recebera um comando                */
/*                        Se 1 --> LCD recebera um dado                   */
```

```
/* Parametros de saida:   n/a                                          */
/* ******************************************************************* */
void lcd_writeData(unsigned char ucData);

/* ******************************************************************* */
/* Nome da funcao:        lcd_sendoCommand                             */
/* Descricao da funcao:   funcao que manda um comando para a           */
/*                        funcao lcd_Write2Lcd a qual realiza no LCD   */
/*                                                                     */
/* Parametros de entrada: Recebe um valor 0 ou 1                       */
/*                        Se 0 --> mandara um comando ao LCD           */
/*                        Se 1 --> mandara um dado ao LCD              */
/* Parametros de saida:   n/a                                          */
/* ******************************************************************* */
void lcd_sendCommand(unsigned char ucCmd);

/* ******************************************************************* */
/* Nome da funcao:        lcd_setCursor                                */
/* Descricao da funcao:   Coloca o cursor na linha e coluna recebidas  */
/*                        por parametro                                */
/*                                                                     */
/* Parametros de entrada: Recebe um valor para a linha e coluna        */
/* Parametros de saida:   n/a                                          */
/* ******************************************************************* */
void lcd_setCursor(unsigned char cLine, unsigned char cColumn);

/* ********************************************************** */
/* Nome da funcao:        lcd_writeString                     */
/* Descricao da funcao:   Recebe a string que sera enviada ao */
/*                        o LCD                               */
/*                                                            */
/* Parametros de entrada: A string                            */
/* Parametros de saida:   n/a                                 */
/* ********************************************************** */
void lcd_writeString(const char *cBuffer);

/* ******************************************************************* */
/* Nome da funcao:        lcd_dummyText                                */
/* Descricao da funcao:   Faz a configuracao da mensagem que sera exibida */
/*                                                                     */
/* Parametros de entrada: n/a                                          */
/* Parametros de saida:   n/a                                          */
/* ******************************************************************* */
void lcd_dummyText(void);

/* ******************************************************************* */
/* Nome da funcao:        lcd_writeText                                */
/* Descricao da funcao:   Recebe do programador qual mensagem serÃ¡ exibida */
/*                        e em qual linha e coluna do LCD isso ocorrera */
/*                                                                     */
```

```
/* Parametros de entrada: A string e a linha e coluna do LCD                */
/* Parametros de saida:   n/a                                               */
/* ************************************************************************ */
void lcd_writeText(int iLinha, const char *cString);

#endif /* SOURCES_LCD_H_ */
```

```
/* ********************************************************************** */
/* Nome do Arquivo:        ledSwi.h                                       */
/* Descrição do arquivo:   Este arquivo contem funcoes para inicializacao */
/*                         e utilizacao dos leds e botoes do kit.         */
/* Nome dos autores:       Gustavo Moraes/Cassio Dezotti                  */
/* RA:                     174217/168988                                  */
/* Data de criacao:        26mar2020                                      */
/* Data da revisao:        28jul2020                                      */
/* ********************************************************************** */

/* our includes */
#include "board.h"

/* system includes */
#include "fsl_clock_manager.h"
#include "fsl_port_hal.h"
#include "fsl_gpio_hal.h"

/*
************************************************************************************
** */
/* Nome da função:         iniciarLedSwi
*/
/* Descrição da função:    Inicializa os LEDS e os botões conforme
especificado.     */
/*                         Se 0 --> configura pino como botão(input)
*/
/*                         Se 1 --> configura pino como LED (output)
*/
/*
*/
/* parâmetros de entrada:  Vetor de entrada com quais LEDS ou botões serão
usados.   */
/* parâmetros de saída:    n/a
*/
/*
************************************************************************************
** */
void iniciarLedSwi(int iEstados[4])
{
    unsigned char ucEstado = 0;

    /* ligar o clock da porta A*/
    SIM_SCGC5 |= 0x0200;

    /* seta os pinos 1 2 4 5 como gpio */
    PORTA_PCR1 &= ~(0X600);
    PORTA_PCR1 |= 0X100;
    PORTA_PCR2 &= ~(0X600);
    PORTA_PCR2 |= 0X100;
```

```
    PORTA_PCR4 &= ~(0X600);
    PORTA_PCR4 |= 0X100;
    PORTA_PCR5 &= ~(0X600);
    PORTA_PCR5 |= 0X100;
    /*
     * Percorre o vetor, como as 2 opcoes de entrada sao 0 ou 1
     * shifta a entrada para o lugar que ele deveria corresponder
     * no vetor final que sera atribuido a função PDDR para definir
     * o pino como entrada ou saida
     */
    ucEstado |= (iEstados[0] << 1);
    ucEstado |= (iEstados[1] << 2);
    ucEstado |= (iEstados[2] << 4);
    ucEstado |= (iEstados[3] << 5);

    GPIOA_PDDR |= ucEstado;

}

/*
**************************************************************************
****  */
/* Nome da função:        mapearEntrada
*/
/* Descrição da função:   Recebe um número de 1 a 4 e mapeia ele para o
respectivo LED  */
/*                        Se 1 --> retorna 1
*/
/*                        Se 2 --> retorna 2
*/
/*                        Se 3 --> retorna 4
*/
/*                        Se 3 --> retorna 5
*/
/*                        Se 4 --> LED 5 alterado
*/
/*
*/
/* Parâmetros de entrada: Valor de 1 a 4
*/
/* Parâmetros de saída:   Retorna 1,2,4 ou 5
*/
/*
**************************************************************************
****  */
int mapearEntrada(int iValor)
{
    /*
     * Queremos que os numeros 1234 representem 1245
     * Nessa funcao, se o numero eh maior que 2 acrescentamos 1
```

```
     */
    if(iValor > 2){
        iValor += 1;
    }

    return iValor;
}

/* *****************************************************************************
*/
/* Nome da função:        lerChave
*/
/* Descrição da função:   Recebe do programador um valor de 1 a 4 que
*/
/*                        determina qual botão deverão ser lido e retorna
*/
/*                        o status atual do botão.
*/
/*                        Se 0 --> botão pressionado
*/
/*                        Se 1 --> botão solto
*/
/*
*/
/* Parâmetros de entrada: Valor de de 1 a 4
*/
/* Parâmetros de saída:   Retorna 0 ou 1
*/
/*                        0 --> botão pressionado
*/
/*                        1 --> botão solto
*/
/* *****************************************************************************
*/
int lerChave(int iChave)
{
    unsigned char ucChaveLida = 0;

    /* Mapeamento da entrada para 1245*/
    int iValorChave = mapearEntrada(iChave);
    /*
     * Shifta o retorno da funcao de leitura da porta A, o numero
     * de vezes necessario para extrairmos o bit de interesse, para isso
     * fazendo um E com o número 1.
     */
    ucChaveLida = (GPIOA_PDIR >> iValorChave) & 1;

    /*
     * Se o valor lido for 1, o botao esta liberado entao,
     * retornamos 0.
```

```
     */
    if( '1' == ucChaveLida){
        return 0;
    }
    /*
     *  Se o valor lido for 0, o botão esta pressionado,
     * retornamos 1.
     */
   if( '0' == ucChaveLida){
       return 1;
   }
   return 0;
}

/* ************************************************************************
*/
/* Nome da função:        escreverLED
*/
/* Descrição da função:   Recebe do programador um valor de 1 a 4 que
*/
/*                        determina em qual LED deve-se escrever e qual serão
*/
/*                        o status final do LED.
*/
/*                        Se 1 --> LED irá apagar
*/
/*                        Se 0 --> LED irá acender
*/
/*
*/
/* Parâmetros de entrada: Valor de 1 a 4 que indica qual o LED serão usado
*/
/*                        SetClear --> 0 ou 1 para indicar status futuro do LED
*/
/* Parâmetros de saída:   n/a
*/
/* ************************************************************************
*/
void escreverLED(int iWriteLed, int iSetClear)
{
    /* Mapeamento da entrada para 1245*/
    int iLedWrite = mapearEntrada(iWriteLed);
    unsigned char ucNumeroDeComando = 1;

    /*
     * Analisa se o parâmetro de entrada eh 1, se for
     * fazemos um shift do número 1 para a posicao desejada e entao um
     * E com o valor que ja estava na porta.
     */
    if(0 == iSetClear){
```

```
        GPIOA_PDOR |= (ucNumeroDeComando << iLedWrite);//se for set dou OU com
a mascara do bit q eu quero
    }
        /*
         * Analisa se o parametro de entrada eh 0, se for
         * fazemos um shift do número 1 negado, para nao perder o conteudo
         * anterior, para a posicao desejada e entao um
         * E com o valor que ja estava na porta.
         */
    else if(1 == iSetClear){
       /* se for clear damos E com a mascara de bits negada */
       GPIOA_PDOR &= ~(ucNumeroDeComando << iLedWrite);
    }
}

/* ********************************************************************
*/
/* Nome da função:        setarLED
*/
/* Descrição da função:   Recebe do programador um valor de 1 a 4 que
*/
/*                        determina qual LED serão alterado.
*/
/*                        Se 1 --> LED 1 acende
*/
/*                        Se 2 --> LED 2 acende
*/
/*                        Se 3 --> LED 4 acende
*/
/*                        Se 4 --> LED 5 acende
*/
/*
*/
/* Parâmetros de entrada: Valor de 1 a 4 que indica qual o LED serão usado
*/
/* Parâmetros de saída:   n/a
*/
/* ********************************************************************
*/
void setarLED(int iSetLed)
{
    /* Mapeamento da entrada para 1245*/
    int iLedSetado = mapearEntrada(iSetLed);
    unsigned char ucNumeroDeComando = 1;
    /*
     * Fazemos um shift do numero 1 ate a posicao do bit desejada
     * entao chamamos a funcao para apagar uma porta
     */
    GPIOA_PCOR |= (ucNumeroDeComando << iLedSetado);
}
```

```
/* ************************************************************************
*/
/* Nome da função:        apagarLED
*/
/* Descrição da função:   Recebe do programador um valor de 1 a 4 que
*/
/*                        determina qual LED serão alterado.
*/
/*                        Se 1 --> LED 1 apaga
*/
/*                        Se 2 --> LED 2 apaga
*/
/*                        Se 3 --> LED 4 apaga
*/
/*                        Se 4 --> LED 5 apaga
*/
/*
*/
/* Parâmetros de entrada: Valor de 1 a 4 que indica qual o LED serão usado
*/
/* Parâmetros de saída:   n/a
*/
/* ************************************************************************
*/
void apagarLED(int iClearLed)
{
    /* Mapeamento da entrada para 1245*/
    int iLedClear = mapearEntrada(iClearLed);
    unsigned char ucNumeroDeComando = 1;
    /*
     * Fazemos um shift do numero 1 ate a posicao do bit desejada
     * entao chamamos a funcao para setar uma porta
     */
    GPIOA_PSOR |= (ucNumeroDeComando << iLedClear);
}

/*
************************************************************************
*** */
/* Nome da função:        alternarLED
*/
/* Descrição da função:   Recebe do programador um valor de 1 a 4 que
*/
/*                        determina qual LED mudarão de status (0 para 1 ou 1
para 0) */
/*                        Se 1 --> LED 1 alterado
*/
/*                        Se 2 --> LED 2 alterado
*/
```

```
/*                          Se 3 --> LED 4 alterado
*/
/*                          Se 4 --> LED 5 alterado
*/
/*
*/
/* Parâmetros de entrada: Valor de 1 a 4 que indica qual o LED serão usado
*/
/* Parâmetros de saída:   n/a
*/
/*
*****************************************************************************
*** */
void alternarLED(int iToggleLed)
{
    /* Mapeamento da entrada para 1245*/
    int iLedToggled = mapearEntrada(iToggleLed);
    unsigned char ucNumeroDeComando = 1;
    /*
     * Fazemos um shift do numero 1 ate a posicao do bit desejada
     * entao chamamos a funcao para apagar uma porta
     */
    GPIOA_PTOR |= (ucNumeroDeComando << iLedToggled);
}
```

```
/* ********************************************************************* */
/* Nome do Arquivo:      ledSwi.h                                        */
/* Descricao do arquivo: Este arquivo contem funcoes para inicializacao  */
/*                       e utilizacao dos leds e botoes do kit.          */
/* Nome dos autores:     Gustavo Moraes/Cassio Dezotti                   */
/* RA:                   174217/168988                                   */
/* Data de criacao:      26mar2020                                       */
/* Data da revisao:      04abril2020                                     */
/* ********************************************************************* */

#ifndef SOURCES_LEDSWI_H_
#define SOURCES_LEDSWI_H_

/*
*************************************************************************
** */
/* Nome da função:         iniciarLedSwi
*/
/* Descrição da função:    Inicializa os LEDS e os botões conforme
especificado.     */
/*                         Se 0 --> configura pino como botão(input)
*/
/*                         Se 1 --> configura pino como LED (output)
*/
/*
*/
/* parâmetros de entrada:  Vetor de entrada com quais LEDS ou botões serão
usados.   */
/* parâmetros de saída:    n/a
*/
/*
*************************************************************************
** */
void iniciarLedSwi(int iEstados[4]);

/* ********************************************************************
*/
/* Nome da função:         lerChave
*/
/* Descrição da função:    Recebe do programador um valor de 1 a 4 que
*/
/*                         determina qual botão deverão ser lido e retorna
*/
/*                         o status atual do botão.
*/
/*                         Se 0 --> botão pressionado
*/
/*                         Se 1 --> botão solto
*/
/*
```

```
*/
/* Parâmetros de entrada: Valor de de 1 a 4
*/
/* Parâmetros de saída:   Retorna 0 ou 1
*/
/*                        0 --> botão pressionado
*/
/*                        1 --> botão solto
*/
/* ************************************************************************
*/
int lerChave(int iChave);

/* ************************************************************************
*/
/* Nome da função:        escreverLED
*/
/* Descrição da função:   Recebe do programador um valor de 1 a 4 que
*/
/*                        determina em qual LED deve-se escrever e qual serão
*/
/*                        o status final do LED.
*/
/*                        Se 1 --> LED irá apagar
*/
/*                        Se 0 --> LED irá acender
*/
/*
*/
/* Parâmetros de entrada: Valor de 1 a 4 que indica qual o LED serão usado
*/
/*                        SetClear --> 0 ou 1 para indicar status futuro do LED
*/
/* Parâmetros de saída:   n/a
*/
/* ************************************************************************
*/
void escreverLED(int iWriteLed, int iSetClear);

/* ************************************************************************
*/
/* Nome da função:        setarLED
*/
/* Descrição da função:   Recebe do programador um valor de 1 a 4 que
*/
/*                        determina qual LED serão alterado.
*/
/*                        Se 1 --> LED 1 acende
*/
/*                        Se 2 --> LED 2 acende
```

```
*/
/*                      Se 3 --> LED 4 acende
*/
/*                      Se 4 --> LED 5 acende
*/
/*
*/
/* Parâmetros de entrada: Valor de 1 a 4 que indica qual o LED serão usado
*/
/* Parâmetros de saída:   n/a
*/
/* ******************************************************************************
*/
void setarLED(int setLed);

/* ******************************************************************************
*/
/* Nome da função:        apagarLED
*/
/* Descrição da função:   Recebe do programador um valor de 1 a 4 que
*/
/*                      determina qual LED serão alterado.
*/
/*                      Se 1 --> LED 1 apaga
*/
/*                      Se 2 --> LED 2 apaga
*/
/*                      Se 3 --> LED 4 apaga
*/
/*                      Se 4 --> LED 5 apaga
*/
/*
*/
/* Parâmetros de entrada: Valor de 1 a 4 que indica qual o LED serão usado
*/
/* Parâmetros de saída:   n/a
*/
/* ******************************************************************************
*/
void apagarLED(int clearLed);

/*
*****************************************************************************
*** */
/* Nome da função:        alternarLED
*/
/* Descrição da função:   Recebe do programador um valor de 1 a 4 que
*/
/*                      determina qual LED mudarão de status (0 para 1 ou 1
para 0) */
```

```
/*                          Se 1 --> LED 1 alterado
*/
/*                          Se 2 --> LED 2 alterado
*/
/*                          Se 3 --> LED 4 alterado
*/
/*                          Se 4 --> LED 5 alterado
*/
/*
*/
/* Parâmetros de entrada: Valor de 1 a 4 que indica qual o LED serão usado
*/
/* Parâmetros de saída:   n/a
*/
/*
*****************************************************************************
*** */
void alternarLED(int iToggleLed);

/*
*****************************************************************************
****  */
/* Nome da função:         mapearEntrada
*/
/* Descrição da função:   Recebe um número de 1 a 4 e mapeia ele para o
respectivo LED  */
/*                          Se 1 --> retorna 1
*/
/*                          Se 2 --> retorna 2
*/
/*                          Se 3 --> retorna 4
*/
/*                          Se 3 --> retorna 5
*/
/*                          Se 4 --> LED 5 alterado
*/
/*
*/
/* Parâmetros de entrada: Valor de 1 a 4
*/
/* Parâmetros de saída:   Retorna 1,2,4 ou 5
*/
/*
*****************************************************************************
****  */
int mapearEntrada(int iValor);

#endif /* SOURCES_LEDSWI_H_ */
```

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

```c
/* ********************************************** */
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 & IagoAF & dloubach             */
/* Creation date:    07jun2018                                    */
/* Revision date:    21jun2018                                    */
/* ************************************************************** */

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

const unsigned char tabela_temp[256] = {
    0, 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, 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
};
```

```
/* ************************************************************** */
/* 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 & IagoAF & dloubach             */
/* Creation date:    07jun2018                                    */
/* Revision date:    21jun2018                                    */
/* ************************************************************** */

#ifndef SOURCES_ADC_LUT_ADC_3V3_H_
#define SOURCES_ADC_LUT_ADC_3V3_H_

extern const unsigned char tabela_temp[256];

#endif /* SOURCES_ADC_LUT_ADC_3V3_H_ */
```

```
/* ************************************************************************ */
/* File name:        mgc.c                                                  */
/* File description: Multipurpose clk generator hardware abstraction        */
/*                   layer. Enables the clock configuration                 */
/*                                                                          */
/*                   Modes of Operation                                     */
/*                   FLL Engaged Internal (FEI)    = DEFAULT                */
/*                   FLL Engaged External (FEE)                             */
/*                   FLL Bypassed Internal (FBI)                            */
/*                   FLL Bypassed External (FBE)                            */
/*                   PLL Engaged External (PEE)                             */
/*                   PLL Bypassed External (PBE)                            */
/*                   Bypassed Low Power Internal (BLPI)                     */
/*                   Bypassed Low Power External (BLPE)                     */
/*                   Stop                                                   */
/*                                                                          */
/*                   For clock definitions, check the chapter               */
/*                   5.4 Clock definitions from                             */
/*                   KL25 Sub-Family Reference Manual                       */
/* Author name:      dloubach                                               */
/* Creation date:    21out2015                                              */
/* Revision date:    21mar2016                                              */
/* ************************************************************************ */

#include "mcg.h"

/* systems include */
#include "fsl_smc_hal.h"
#include "fsl_port_hal.h"
#include "fsl_clock_manager.h"

/* EXTAL0 PTA18 */
#define EXTAL0_PORT                 PORTA
#define EXTAL0_PIN                  18U
#define EXTAL0_PINMUX               kPortPinDisabled

/* XTAL0 PTA19 */
#define XTAL0_PORT                  PORTA
#define XTAL0_PIN                   19U
#define XTAL0_PINMUX                kPortPinDisabled

/* OSC0 configuration */
#define OSC0_INSTANCE               0U
#define OSC0_XTAL_FREQ              8000000U /* 08 MHz*/
#define OSC0_SC2P_ENABLE_CONFIG     false
#define OSC0_SC4P_ENABLE_CONFIG     false
#define OSC0_SC8P_ENABLE_CONFIG     false
#define OSC0_SC16P_ENABLE_CONFIG    false
#define MCG_HGO0                    kOscGainLow
#define MCG_RANGE0                  kOscRangeVeryHigh
```

```
#define MCG_EREFS0                   kOscSrcOsc

/* RTC external clock configuration. */
#define RTC_XTAL_FREQ                0U
#define RTC_SC2P_ENABLE_CONFIG       false
#define RTC_SC4P_ENABLE_CONFIG       false
#define RTC_SC8P_ENABLE_CONFIG       false
#define RTC_SC16P_ENABLE_CONFIG      false
#define RTC_OSC_ENABLE_CONFIG        false
#define RTC_CLK_OUTPUT_ENABLE_CONFIG false

/* RTC_CLKIN PTC1 */
#define RTC_CLKIN_PORT               PORTC
#define RTC_CLKIN_PIN                1U
#define RTC_CLKIN_PINMUX             kPortMuxAsGpio


#define CLOCK_VLPR                   1U /* very low power run mode */
#define CLOCK_RUN                    2U /* run mode */

#ifndef CLOCK_INIT_CONFIG
#define CLOCK_INIT_CONFIG CLOCK_RUN
#endif


/* Configuration for enter VLPR mode, Core clock = 4MHz */
const clock_manager_user_config_t g_defaultClockConfigVlpr =
{
    .mcgConfig =
    {
        .mcg_mode           = kMcgModeBLPI,       // Work in BLPI mode
        .irclkEnable        = true,               // MCGIRCLK enable
        .irclkEnableInStop  = false,              // MCGIRCLK disable in STOP
mode
        .ircs               = kMcgIrcFast,        // Select IRC4M
        .fcrdiv             = 0U,                 // FCRDIV is 0

        .frdiv   = 0U,
        .drs     = kMcgDcoRangeSelLow,            // Low frequency range
        .dmx32   = kMcgDmx32Default,              // DCO has a default range of
25%

        .pll0EnableInFllMode = false,             // PLL0 disable
        .pll0EnableInStop  = false,               // PLL0 disalbe in STOP mode
        .prdiv0            = 0U,
        .vdiv0             = 0U,
    },
    .simConfig =
    {
        .pllFllSel = kClockPllFllSelFll,          // PLLFLLSEL select FLL
```

```
        .er32kSrc  = kClockEr32kSrcLpo,          // ERCLK32K selection, use
LPO
        .outdiv1   = 0U,
        .outdiv4   = 4U,
    },
    .oscerConfig =
    {
        .enable       = true,                    // OSCERCLK enable
        .enableInStop = false,                   // OSCERCLK disable in STOP
mode
    }
};

/* Configuration for enter RUN mode, Core clock = 40 MHz */
/*
 * 24.5.1.1 Initializing the MCG
 * KL25 Sub-Family Reference Manual, Rev. 3, September 2012
 *
 * Refer also to
 * Table 24-18. MCG modes of operation
 *
 * On L-series devices the MCGFLLCLK frequency is limited to 48 MHz max
 * The DCO is limited to the two lowest range settings (MCG_C4[DRST_DRS] must
be set to either 0b00 or 0b01).
 *
 * FEE (FLL engaged external)
 * fext / FLL_R must be in the range of 31.25 kHz to 39.0625 kHz
 * FLL_R is the reference divider selected by the C1[FRDIV] bits
 * F is the FLL factor selected by C4[DRST_DRS] and C4[DMX32] bits
 *
 * (fext / FLL_R) * F = (8 MHz / 256 ) * 1280 = 40 MHz
 *
 * */
const clock_manager_user_config_t g_defaultClockConfigRun =
{
    /* ----------- multipurpose clock generator configurations --------------
*/
    .mcgConfig =
    {
        .mcg_mode           = kMcgModeFEE,       // Work in FEE mode

        /* ------------------ MCGIRCCLK settings ---------------------- */
        .irclkEnable        = true,              // MCGIRCLK enable
        .irclkEnableInStop  = false,             // MCGIRCLK disable in STOP
mode
        .ircs               = kMcgIrcSlow,       // Select IRC 32kHz
        .fcrdiv             = 0U,                // FCRDIV is 0

        /* -------------------- MCG FLL settings ---------------------- */
        .frdiv   = 0b011,                        // Divide Factor is 256 (EXT
```

```
OSC 8 MHz / 256 = 31.250 kHz)
                                                    // The resulting frequency
must be in the range 31.25 kHz to 39.0625 kHz
        .drs     = kMcgDcoRangeSelMid,              // frequency range
        .dmx32   = kMcgDmx32Default,                // DCO has a default range of
25%

        /* -------------------- MCG PLL settings ---------------------- */
        .pll0EnableInFllMode = false,               // PLL0 disable
        .pll0EnableInStop  = false,                 // PLL0 disable in STOP mode
        .prdiv0            = 0x0U,
        .vdiv0             = 0x0U,
    },
    /* ----------- system integration module configurations    --------------
*/
    .simConfig =
    {
        .pllFllSel = kClockPllFllSelFll,            // PLLFLLSEL select PLL
        .er32kSrc  = kClockEr32kSrcLpo,             // ERCLK32K selection, use LPO
        .outdiv1   = 0U,                            // core/system clock, as well
as the bus/flash clocks.
        .outdiv4   = 1U,                            // bus and flash clock and is
in addition to the System clock divide ratio
    },
    /* ----------- system oscillator output configurations     --------------
*/
    .oscerConfig =
    {
        .enable       = true,                       // OSCERCLK enable
        .enableInStop = false,                      // OSCERCLK disable in STOP
mode
    }
};


/* ************************************************ */
/* Method name:        mcg_initOsc0                 */
/* Method description: Oscillator configuration     */
/* Input params:       n/a                          */
/* Output params:      n/a                          */
/* ************************************************ */
void mcg_initOsc0(void)
{
    /* OSC0 configuration */
    osc_user_config_t osc0Config =
    {
        .freq                = OSC0_XTAL_FREQ,
        .hgo                 = MCG_HGO0,
        .range               = MCG_RANGE0,
        .erefs               = MCG_EREFS0,
```

```
        .enableCapacitor2p   = OSC0_SC2P_ENABLE_CONFIG,
        .enableCapacitor4p   = OSC0_SC4P_ENABLE_CONFIG,
        .enableCapacitor8p   = OSC0_SC8P_ENABLE_CONFIG,
        .enableCapacitor16p  = OSC0_SC16P_ENABLE_CONFIG,
    };

    /* oscillator initialization */
    CLOCK_SYS_OscInit(OSC0_INSTANCE, &osc0Config);
}



/* ********************************************** */
/* Method name:        mcg_initRtcOsc             */
/* Method description: Function to initialize RTC */
/*                     external clock base on     */
/*                     board configuration        */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void mcg_initRtcOsc(void)
{

#if RTC_XTAL_FREQ
    // If RTC_CLKIN is connected, need to set pin mux. Another way for
    // RTC clock is set RTC_OSC_ENABLE_CONFIG to use OSC0, please check
    // reference manual for details
    PORT_HAL_SetMuxMode(RTC_CLKIN_PORT, RTC_CLKIN_PIN, RTC_CLKIN_PINMUX);
#endif

#if ((OSC0_XTAL_FREQ != 32768U) && (RTC_OSC_ENABLE_CONFIG))
#error Set RTC_OSC_ENABLE_CONFIG will override OSC0 configuration and OSC0 must
be 32k.
#endif

    rtc_osc_user_config_t rtcOscConfig =
    {
        .freq                = RTC_XTAL_FREQ,
        .enableCapacitor2p   = RTC_SC2P_ENABLE_CONFIG,
        .enableCapacitor4p   = RTC_SC4P_ENABLE_CONFIG,
        .enableCapacitor8p   = RTC_SC8P_ENABLE_CONFIG,
        .enableCapacitor16p  = RTC_SC16P_ENABLE_CONFIG,
        .enableOsc           = RTC_OSC_ENABLE_CONFIG,
        .enableClockOutput   = RTC_CLK_OUTPUT_ENABLE_CONFIG,
    };

    /* OSC RTC initialization */
    CLOCK_SYS_RtcOscInit(0U, &rtcOscConfig);
}
```

```
/* ********************************************** */
/* Method name:        mcg_initSystemClock        */
/* Method description: System clock configuration */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void mcg_initSystemClock(void)
{
    /* Set system clock configuration. */
    #if (CLOCK_INIT_CONFIG == CLOCK_VLPR)
        CLOCK_SYS_SetConfiguration(&g_defaultClockConfigVlpr);
    #else
        CLOCK_SYS_SetConfiguration(&g_defaultClockConfigRun);
    #endif
}



/* ********************************************** */
/* Method name:        mcg_clockInit              */
/* Method description: main board clk configuration */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void mcg_clockInit(void)
{
    /* enable clock for PORTs */
    CLOCK_SYS_EnablePortClock(PORTA_IDX);
    CLOCK_SYS_EnablePortClock(PORTC_IDX);
    CLOCK_SYS_EnablePortClock(PORTE_IDX);

    /* set allowed power mode to allow all */
    SMC_HAL_SetProtection(SMC, kAllowPowerModeAll);

    /* configure OSC0 pin mux */
    PORT_HAL_SetMuxMode(EXTAL0_PORT, EXTAL0_PIN, EXTAL0_PINMUX);
    PORT_HAL_SetMuxMode(XTAL0_PORT, XTAL0_PIN, XTAL0_PINMUX);

    /* setup OSC0 */
    mcg_initOsc0();

    /* setup OSC RTC */
    mcg_initRtcOsc();

    /* setup system clock */
    mcg_initSystemClock();
}
```

```
/* ************************************************************** */
/* File name:        mcg.h                                        */
/* File description: Header file containing the functions/methods */
/*                   interfaces for handling the Multipurpose clock */
/*                   generator module                             */
/* Author name:      dloubach                                     */
/* Creation date:    21out2015                                    */
/* Revision date:    25fev2016                                    */
/* ************************************************************** */

#ifndef SOURCES_MCG_H_
#define SOURCES_MCG_H_

/* ********************************************* */
/* Method name:        mcg_clockInit             */
/* Method description: main board clk configuration */
/* Input params:       n/a                       */
/* Output params:      n/a                       */
/* ********************************************* */
void mcg_clockInit(void);

#endif /* SOURCES_MCG_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, IagoAF, rBacurau               */
/* Creation date:    21jun2018                                    */
/* Revision date:    27mai2020                                    */
/* ************************************************************** */

#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            */
/* Output params:      float: New Control effort     */
/* ************************************************ */
float pidUpdateData(float fSensorValue, float fSetValue)
{
    float fError, fDifference, fOut;

    fError = fSetValue - fSensorValue;
    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>100.0)
        fOut = 100.0;

    else if (fOut<0.0)
        fOut = 0.0;

    return fOut;
}
```

```
/* ************************************************************** */
/* File name:        pid.h                                        */
/* File description: Header file containing the functions/methods */
/*                   interfaces for handling the PID              */
/* Author name:      julioalvesMS, IagoAF, 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            */
/* Output params:      float: New Control effort     */
/* ************************************************ */
float pidUpdateData(float fSensorValue, float fReferenceValue);

#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 conversion */
    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.
 */

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

#ifndef __PRINT_SCAN_H__
#define __PRINT_SCAN_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 /* __PRINT_SCAN_H__ */
```

```
/* ************************************************************** */
/* 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"
#include "board.h"

/* ********************************************** */
/* 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(UART_PORT, UART_PIN_1, UART_ALT);
    /* UART0_TX */
    PORT_HAL_SetMuxMode(UART_PORT, UART_PIN_2, UART_ALT);

    /* 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)
{
    // Echo received character
    processamentoByte(debug_getchar());
}
```

```
/* ************************************************************** */
/* 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:        UART0_init                 */
/* Method description: Initialize the UART0 as debug */
/* 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:    21mar2016                                    */
/* ************************************************************** */

#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");
    }
}



/* ********************************************* */
/* 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();
}
```

```
/* ************************************************************** */
/* 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:    09mar2016                                    */
/* ************************************************************** */

#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);


#endif /* UTIL_H */
```