Código Fonte- Controlador de Temperatura
Caio Villela - 168342
Hebert Wandick - 174335

adc.h

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

#ifndef SOURCES_ADC_H_
#define SOURCES_ADC_H_


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


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


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

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


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


#endif /* SOURCES_ADC_H_ */
```

adc.c

```
/* ****************************************************************** */
/* File name:        adc.c                                            */
/* File description: This file has a couple of useful functions to    */
/*                   control the ADC from the peripheral board.       */
/*                   The converter is connected to the Temperature    */
/*                   sen/*
****************************************************************** */
/* 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"
#include "fsl_port_hal.h"
#include "fsl_gpio_hal.h"
#include "fsl_clock_manager.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
}


sor.                                          */
/* Author name:      dloubach, julioalvesMS, IagoAF e rbacurau      */
/* Creation date:    07jun2018                                      */
/* Revision date:    20mai2020                                      */
/* ************************************************************** */

#include "board.h"
#include "adc.h"
#include "fsl_port_hal.h"
#include "fsl_gpio_hal.h"
#include "fsl_clock_manager.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
}
```

aquecedorECooler.h

```
/* ************************************************************ */
/* File name:        aquecedorECooler.h                         */
```

```
/* File description: Header file containing the functions needed to  */
/*                   use heater and cooler                           */
/* Author name:      Caio Villela, Hebert Wandick                    */
/* Creation date:    10/apr/2020                                     */
/* Revision date:    26/jul/2020                                     */
/* ********************************************************** */

#ifndef AQUECEDORECOOLER_H_
#define AQUECEDORECOOLER_H_

/* ********************************************* */
/* Method name:        PWM_init                  */
/* Method description: Initialize PWM            */
/* Input params:       n/a                       */
/* Output params:      n/a                       */
/* ********************************************* */
void PWM_init(void);

/* ********************************************* */
/* Method name:        coolerfan_init            */
/* Method description: Initialize coolerfan      */
/* Input params:       n/a                       */
/* Output params:      n/a                       */
/* ********************************************* */
void coolerfan_init(void);

/* ********************************************* */
/* Method name:        heater_init               */
/* Method description: Initialize heater         */
/* Input params:       n/a                       */
/* Output params:      n/a                       */
/* ********************************************* */
void heater_init(void);

/* ********************************************* */
/* Method name:        coolerfan_PWMDuty         */
/* Method description: set the coolerfan speed   */
/* Input params:       fCoolerDuty how many      */
/*                     percentage time the cooler*/
/*                     will be in high           */
```

```
/* Output params:      n/a                          */
/* ******************************************** */
void coolerfan_PWMDuty(float fCoolerDuty);


/* ******************************************** */
/* Method name:        getCoolerDuty                */
/* Method description: get the coolerfan speed      */
/* Output params:      float 0 to 1 representing    */
/*                     speed in %                   */
/* ******************************************** */
float getCoolerDuty();

/* ******************************************** */
/* Method name:        heater_PWMDuty               */
/* Method description: set the heater power         */
/* Input params:       fHeaterDuty how many         */
/*                     percentage time the cooler   */
/*                     will be in high              */
/* Output params:      n/a                          */
/* ******************************************** */
void heater_PWMDuty(float fHeaterDuty);

/* ******************************************** */
/* Method name:        getHeaterDuty                */
/* Method description: get the heater power use     */
/* Output params:      float (0 to 1) representing  */
/*                     power use in %               */
/* ******************************************** */
float getHeaterDuty();

#endif
```

aquecedorECooler.c

```
/* ************************************************************** */
/* File name:        aquecedorECooler.c                           */
/* File description: File containing the implementation of functions */
/*                   used to control the CoolerFan and Header     */
```

```
/* Author name:      Caio Villela, Hebert Wandick                    */
/* Creation date:    21/apr/2020                                     */
/* Revision date:    26/jul/2020                                     */
/* ********************************************************** */

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

/*system include*/
#include "aquecedorECooler.h"
#include "fsl_clock_manager.h"

/* ********************************************* */
/* Method name:        PWM_init                  */
/* Method description: Initialize PWM            */
/* Input params:       n/a                       */
/* Output params:      n/a                       */
/* ********************************************* */
void PWM_init(void){
    /*un-gateport A clock*/
    SET_BITS(SIM_SCGC5, CGC_CLOCK_ENABLED, 1, PORT_A_TO_SHIFT);

    /*un-gateport TPM1 clock*/
    SET_BITS(SIM_SCGC6, CGC_CLOCK_ENABLED, 1, 25);

    /*Prescale Factor Select at TPM1_SC on bytes 2-0*/
    SET_BITS(TPM1_SC, PRESCALER, 3, 0);

    /*TPM clock source select at SIM_SOPT2 on bytes 24-25*/
    SET_BITS(SIM_SOPT2, TPM_CLOCK, 2, 24);

    /*Clock Mode Selection at TPM1_SC on bytes 3-4*/
    SET_BITS(TPM1_SC, TPM_CMOD, 2, 3);

    /*Center-aligned PWM Select,0 LPTPM counter operates in up counting
mode.*/
    SET_BITS(TPM1_SC, TPM_CPWMS, 1, 5);

    /**/
    /*Counting 50 pulses*/
```

```
    SET_BITS(TPM1_MOD, TPM_MOD, 16, 0);
}

/* ********************************************** */
/* Method name:        coolerfan_init             */
/* Method description: Initialize coolerfan       */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void coolerfan_init(void){
    /*Set MSnA:MSnB and ELSnA:ELSnB*/
    SET_BITS(COOLERFAN_TPMx_CnSC, TPM_MSnx, 2, 4);
    SET_BITS(COOLERFAN_TPMx_CnSC, TPM_ELSnx, 2, 2);

    /*Config port as PWM port */
    SET_BITS(COOLERFAN_PORTx_PCRn, CONFIG_PORT_AS_PWM, 3, 8);
    coolerfan_PWMDuty(0);
}

/* ********************************************** */
/* Method name:        heater_init                */
/* Method description: Initialize heater          */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void heater_init(void){
    /*Set MSnA:MSnB and ELSnA:ELSnB*/
    SET_BITS(HEADER_TPMx_CnSC, TPM_MSnx, 2, 4);
    SET_BITS(HEADER_TPMx_CnSC, TPM_ELSnx, 2, 2);

    /*Config port as PWM port */
    SET_BITS(HEADER_PORTx_PCRn, CONFIG_PORT_AS_PWM, 3, 8);
    heater_PWMDuty(0);
}

float fSaveCoolerDuty;//aux varieble to use in getCoolerDuty

/* ********************************************** */
/* Method name:        coolerfan_PWMDuty          */
/* Method description: set the coolerfan speed    */
```

```
/* Input params:       fCoolerDuty how many         */
/*                     percentage time the cooler   */
/*                     will be in high              */
/* Output params:      n/a                          */
/* ************************************************ */
void coolerfan_PWMDuty(float fCoolerDuty){
    /*reset the TPM1 */
    SET_BITS(TPM1_CNT,0x00,16,0);

    SET_BITS(COOLERFAN_TPMx_CnV, (unsigned int)(fCoolerDuty*49), 16,
0);

    /*save state*/
    fSaveCoolerDuty= fCoolerDuty;
}

/* ************************************************ */
/* Method name:        getCoolerDuty                */
/* Method description: get the coolerfan speed      */
/* Output params:      float 0 to 1 representing    */
/*                     speed in %                   */
/* ************************************************ */
float getCoolerDuty(){
    return fSaveCoolerDuty;
}

float fSaveHeaterDuty;//aux varieble to use in getHeaterDuty

/* ************************************************ */
/* Method name:        heater_PWMDuty               */
/* Method description: set the heater power         */
/* Input params:       fHeaterDuty how many         */
/*                     percentage time the cooler   */
/*                     will be in high              */
/* Output params:      n/a                          */
/* ************************************************ */
void heater_PWMDuty(float fHeaterDuty){
    /*reset the TPM1 */
    SET_BITS(TPM1_CNT,0x00,16,0);
```

```
    SET_BITS(HEADER_TPMx_CnV, (unsigned int)(fHeaterDuty*49), 16, 0);

    /*save state*/
    fSaveHeaterDuty= fHeaterDuty;

}

/* ********************************************** */
/* Method name:        getHeaterDuty              */
/* Method description: get the heater power use   */
/* Output params:      float (0 to 1) representing */
/*                     power use in %             */
/* ********************************************** */
float getHeaterDuty(){

    return fSaveHeaterDuty;
};
```

board.h

```
/* ************************************************************** */
/* File name:        board.h                                      */
/* File description: Header file containing the peripherals mapping */
/*                     of the peripheral board for the ES670 hardware*/
/*                                                                */
/* Author name:      Hebert Wandick / Caio Villela                */
/* Creation date:    24bri2020                                   */
/* Revision date:    25abri2020                                   */
/* ************************************************************** */
#ifndef SOURCES_BOARD_H_
#define SOURCES_BOARD_H_


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

/*                 General uC definitions                 */
```

```
#define SET_BITS(OUTPUT, INPUT, NUM_BITS, TO_SHIFT)
(OUTPUT = ((OUTPUT & ( ~( (0XFFFFFFFFU  >> (32 - NUM_BITS)) <<
TO_SHIFT) ) ) | (INPUT << TO_SHIFT)) )

#define BUTTON1_PORT_BASE_PNT   PORTA /* peripheral port base pointer
*/
#define BUTTON1_GPIO_BASE_PNT   PTA   /* peripheral gpio base pointer
*/
#define BUTTON1_PIN          (uint32_t) 1u
#define BUTTON1_MASK            0x02

#define BUTTON2_PORT_BASE_PNT   PORTA /* peripheral port base pointer
*/
#define BUTTON2_GPIO_BASE_PNT   PTA   /* peripheral gpio base pointer
*/
#define BUTTON2_PIN          (uint32_t) 2u
#define BUTTON2_MASK            0x04

#define BUTTON3_PORT_BASE_PNT   PORTA /* peripheral port base pointer
*/
#define BUTTON3_GPIO_BASE_PNT   PTA   /* peripheral gpio base pointer
*/
#define BUTTON3_PIN          (uint32_t) 4u
#define BUTTON3_MASK            0x10

#define BUTTON4_PORT_BASE_PNT   PORTA /* peripheral port base pointer
*/
#define BUTTON4_GPIO_BASE_PNT   PTA   /* peripheral gpio base pointer
*/
#define BUTTON4_PIN          (uint32_t) 5u
#define BUTTON4_MASK            0x20

#define LED1_PORT_BASE_PNT   PORTA /* peripheral port base pointer */
#define LED1_GPIO_BASE_PNT   PTA   /* peripheral gpio base pointer */
#define LED1_PIN             (uint32_t) 1u
#define LED1_MASK            0x02

#define LED2_PORT_BASE_PNT   PORTA /* peripheral port base pointer */
#define LED2_GPIO_BASE_PNT   PTA   /* peripheral gpio base pointer */
#define LED2_PIN             (uint32_t) 2u
```

```
#define LED2_MASK            0x04

#define LED3_PORT_BASE_PNT   PORTA /* peripheral port base pointer */
#define LED3_GPIO_BASE_PNT   PTA   /* peripheral gpio base pointer */
#define LED3_PIN             (uint32_t) 4u
#define LED3_MASK            0x10

#define LED4_PORT_BASE_PNT   PORTA /* peripheral port base pointer */
#define LED4_GPIO_BASE_PNT   PTA   /* peripheral gpio base pointer */
#define LED4_PIN             (uint32_t) 5u
#define LED4_MASK            0x20


/* Clock gate control */
#define  CGC_CLOCK_DISABLED     0x00U       /*Disable clock*/
#define  CGC_CLOCK_ENABLED      0x01U       /*Enable clock*/
#define PORT_A_TO_SHIFT         0x09U       /*Select byte to be turn
on*/

#define PRESCALER               0b110U      /*Prescale Factor
Selection, 110 Divide by 64*/
#define TPM_CLOCK               0b11U       /*TPM clock source select,
11 MCGIRCLK clock (32KHz)*/
#define TPM_CMOD                0b01U       /*Clock Mode Selection *,01
LPTPM counter increments on every LPTPM counter clock */
#define TPM_CPWMS               0b0U        /*Center-aligned PWM
Select,0 LPTPM counter operates in up counting mode.*/
#define TPM_MOD                 49U         /*Counting 50 pulses*/
#define TPM_MSnx                0b10U       /*Edge-aligned PWM*/
#define TPM_ELSnx               0b10U       /*High-true pulses (clear
Output on match, set Output on reload)*/
#define CONFIG_PORT_AS_PWM      0b011U      /*011 Alternative 3
(chip-specific).*/

#define COOLERFAN_PORTx_PCRn    PORTA_PCR13 /*Cooler connect at portA
pin 13*/
#define COOLERFAN_TPMx_CnSC     TPM1_C0SC   /*Cooler connect at TPM1 on
CH 0*/
#define COOLERFAN_TPMx_CnV  TPM1_C0V    /*Cooler connect at TPM1 on CH
0*/
```

```
#define HEADER_PORTx_PCRn       PORTA_PCR12 /*Header connect at portA
pin 12*/
#define HEADER_TPMx_CnSC        TPM1_C1SC   /*Header connect at TPM1 on
CH 1*/
#define HEADER_TPMx_CnV         TPM1_C1V    /*Header connect at TPM1 on
CH 1*/

/*                General uC definitions                */

/* Clock gate control */
#define  CGC_CLOCK_DISABLED         0x00U
#define  CGC_CLOCK_ENABLED          0x01U

/* GPIO input / output */
#define GPIO_INPUT                  0x00U
#define GPIO_OUTPUT                 0x01U

/*                LCD 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                */

/*                General uC definitions                */

/* Clock gate control */
#define  CGC_CLOCK_DISABLED         0x00U
#define  CGC_CLOCK_ENABLED          0x01U

/* GPIO input / output */
#define GPIO_INPUT                  0x00U
```

```
#define GPIO_OUTPUT                 0x01U


/*  TEMPERATURE SENSOR DIEODE DEFINITONS */
#define THERMOMETER_PORT_BASE_PNT   PORTE       /*peripheral port base
pointer */
#define THERMOMETER_GPIO_BASE_PNT   PTE     /*peripheral gpio base
pointer */
#define THERMOMETER_PIN             21U     /*THERMOMETER PIN */
#define THERMOMETER_DIR             (GPIO_INPUT << TERMOMETER_PIN)
#define THERMOMETER_ALT             0X00u

/*      END OF TERMPERATURE SENSOR DIODE DEFINITONS */

/*state machine constants*/
#define IDLE '0'
#define READY '1'
#define GET '2'
#define SET '3'
#define PARAM '4'
#define FLOAT_VALUE '5'
#define BUTTON_VALUE '6'
#define SET_VALUE '7'
#define TARGETTEMP '8'
#define TARGETKD '9'
#define TARGETKI '10'
#define TARGETKP '11'
#define DUTYHEATER '12'
#define DUTYCOOLER '13'


#define MAX_VALUE_LENGTH 5
/*end of state machine constants*/

#endif /* SOURCES_BOARD_H_ */
```

communicationStateMachine.h

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

```
/* File name:        communicationStateMachine.h                    */
/* File description: Implements the State machine, discribe in .pdf  */
/* Author name:      Caio Villela, Hebert Wandick                   */
/* Creation date:    24/may/2020                                    */
/* Revision date:    24/may/2020                                    */
/* ************************************************************** */
#ifndef COMMUNICATIONSTATEMACHINE_H_
#define COMMUNICATIONSTATEMACHINE_H_

/* ********************************************* */
/* Method name:        processByteCommunication     */
/* Method description: Handle what to do by         */
/*                     corresponding byte and order */
/* Input params:       ucByte the byte received     */
/* Output params:      n/a                          */
/* ********************************************* */
void processByteCommunication(unsigned char ucByte);

extern bool bPidConfig;

enum state{
    IDLE,
    READY,
    GET,
    SET,
    PARAM,
    FLOAT_VALUE,
    BUTTON_VALUE,
    TARGETTEMP,
    TARGETKD,
    TARGETKI,
    TARGETKP,
    DUTYHEATER,
    DUTYCOOLER,
};

/* ********************************************* */
/* Method name:        getState                     */
/* Method description: Return in witch state the    */
/*                     state machine are            */
```

```
/* Input params:       n/a                          */
/* Output params:      enum state                   */
/* ******************************************** */
enum state getState();

#endif
```

communicationStateMachine.c

```
/* ************************************************************** */
/* File name:         communicationStateMachine.c                 */
/* File description: Implements the state machine using the util.h */
/*                    file to implements the responsive acts      */
/*                    through UART interface                      */
/* Author name:       Caio Villela/ Hebert Wandick                */
/* Creation date:     24/may/2020                                 */
/* Revision date:     24/may/2020                                 */
/* ************************************************************** */

/*my includes*/
#include "util.h"
#include "communicationStateMachine.h"
#include "board.h"

unsigned char ucUartState = IDLE;
unsigned char ucValueCount = 0;
int iCommaPos = 0, iAuxCommaPos = 0, iFlag = 0;
bool bPidConfig = false;

enum state esUartState = IDLE;
/* ******************************************** */
/* Method name:        processByteCommunication */
/* Method description: Handle what to do by     */
/*                     corresponding byte and order */
/* Input params:       ucByte the byte received */
/* Output params:      n/a                      */
/* ******************************************** */
void processByteCommunication(unsigned char ucByte)
{
```

```
    static unsigned char ucParam;
    static unsigned char ucValue[MAX_VALUE_LENGTH + 2];
    switch (esUartState)
    {
    case IDLE: /*wait for UART comand*/
        switch (ucByte)
        {
        case '#': /*UART Comand arive */
            esUartState = READY;
            break;
        case '@':/*progrming via Local interface*/
            esUartState = TARGETTEMP;
        }
        break;
    case TARGETTEMP:/*condiguring controled temperature*/
        switch (ucByte)
        {
        case '@':/*go to next menu*/
            esUartState = TARGETKP;
            break;
        case '>':/*going up*/
            setParam('T', "u");
            break;
        case '<':/*going dow*/
            setParam('T', "d");
            break;
        default:
            esUartState = IDLE;/*some one send comand via uart in same
time, wi back idle state, before acet nem orders */
        }
        break;

    case TARGETKP:/*set KP*/
        switch (ucByte)
        {
        case '@':/*next menu */
            esUartState = TARGETKD;
            break;
        case '>':/*up*/
```

```
            setParam('P', "u");
            break;
        case '<':/*dow*/
            setParam('P', "d");
            break;
        default:
            esUartState = IDLE;/*some one send comand via uart in same
time, wi back idle state, before acet nem orders */
        }
        break;

    case TARGETKD:/*same logic in here*/
        bPidConfig = true;
        switch (ucByte)
        {
        case '@':
            esUartState = TARGETKI;
            break;
        case '>':
            setParam('D', "u");
            break;
        case '<':
            setParam('D', "d");
            break;
        default:
            esUartState = IDLE;
        }
        break;
    case TARGETKI:
        switch (ucByte)
        {
        case '@':
            esUartState = DUTYHEATER;
            break;
        case '>':
            setParam('I', "u");
            break;
        case '<':
            setParam('I', "d");
            break;
```

```
        default:
            esUartState = IDLE;
        }
        break;

    case DUTYCOOLER:
        bPidConfig = false;
        switch (ucByte)
        {
        case '@':
            esUartState = DUTYHEATER;
            break;
        case '>':
            setParam('C', "u");
            break;
        case '<':
            setParam('C', "d");
            break;
        default:
            esUartState = IDLE;
        }
        break;

    case DUTYHEATER:
        esUartState = IDLE;
        break;

    case READY:
        switch (ucByte)
        {
        case 'g':/*get infamation */
            esUartState = GET;
            break;

        case 's':/*set varieble */
            esUartState = SET;
            break;
        case '@':
        case '<':
```

```
        case '>':/*case some on click on local inteface it will not
interfere*/
            break;
        default:
            esUartState = IDLE;
        }
        break;

    case GET:
        switch (ucByte)
        {
        case 't':/* temperature now*/
        case 'c':/* cooler*/
        case 'a':/* resistor*/
        case 'i': /*ki*/
        case 'd': /*kd*/
        case 'p': /*kp*/
        case 'b': /*botaão*/
        case 's': /*temp to control*/
            ucParam = ucByte;
            esUartState = PARAM;
            break;
        case '@':
        case '<':
        case '>':/*case some on click on local inteface it will not
interfere*/
            break;
        default:
            esUartState = IDLE;
            break;
        }
        break;

    case SET:
        switch (ucByte)
        {
        case 'c': /*cooler*/
        case 'i':/*ki*/
        case 'd':/*kd*/
        case 'p':/*Kp*/
```

```
        case 's':/*Temp to control*/
            ucParam = ucByte;
            ucValueCount = 0;
            esUartState = FLOAT_VALUE;
            break;
        case 'b':/*button*/
            ucParam = ucByte;
            iFlag = 0;
            esUartState = BUTTON_VALUE;
            break;
        case '@':
        case '<':
        case '>':
            break;
        default:
            esUartState = IDLE;
            break;
        }
        break;

    case PARAM:
        if (';' == ucByte)
            answerParam(ucParam);
        if (ucByte == '@' || ucByte == '<' || ucByte == '>')
            ;
        else
            esUartState = IDLE;
        break;

    case FLOAT_VALUE:
        switch (ucByte)
        {
        case '0':
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
```

```
        case '8':
        case '9':
        case ',':
            if (ucValueCount < MAX_VALUE_LENGTH)
            {
                ucValue[ucValueCount++] = ucByte;
                iAuxCommaPos++;
            }
            else
            {
                esUartState = IDLE;
            }
            break;
        case '@':
        case '<':
        case '>':
            break;
        case ';':
            ucValue[ucValueCount] = '\0';
            setParam(ucParam, ucValue);
            esUartState = IDLE;
            break;
        default:
            esUartState = IDLE;
            break;
        }
        break;

    case BUTTON_VALUE:
        if (iFlag == 0)
        {
            if ('0' == ucByte || '1' == ucByte)
            {
                iFlag++;
                ucValue[0] = ucByte;
            }
        }
        else
        {
            if (';' == ucByte)
```

```
                setParam(ucParam, ucValue);

            if (ucByte == '@' || ucByte == '<' || ucByte == '>')
                ;
            else
                esUartState = IDLE;
            break;
        }
    }
}

/* ********************************************** */
/* Method name:        getState                   */
/* Method description: Return in witch state the  */
/*                     state machine are          */
/* Input params:       n/a                        */
/* Output params:      enum state                 */
/* ********************************************** */
enum state getState(void)
{
    return esUartState;
}
```

fsl_debug_console.c

```
/*
 * Copyright (c) 2013 - 2014, Freescale Semiconductor, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright
notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright
notice, this
```

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

#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include "fsl_device_registers.h"
#include "fsl_debug_console.h"
#if defined(UART_INSTANCE_COUNT)
#include "fsl_uart_hal.h"
#endif
#if defined(LPUART_INSTANCE_COUNT)
```

```
#include "fsl_lpuart_hal.h"
#endif
#if defined(UART0_INSTANCE_COUNT)
#include "fsl_lpsci_hal.h"
#endif
#include "fsl_clock_manager.h"
#include "fsl_os_abstraction.h"
#include "print_scan.h"

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

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

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

/*************************************************************************
*********
 * Definitions

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

/*! @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
```

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

lcd.h

```
/* ************************************************************** */
/* File name:        lcd.h                                        */
/* File description: file containing the description of functions */
/*                   used to control the LCD display              */
/* Author name:      Caio Villela, Hebert Wandick                 */
/* Creation date:    09/apr/2020                                  */
/* Revision date:    10/apr/2020                                  */
/* ************************************************************** */


#ifndef SOURCES_LCD_H_
#define SOURCES_LCD_H_

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


/* ********************************************* */
/* Method name:        lcd_initLcd               */
/* Method description: Initialize the LCD function  */
/* Input params:       n/a                       */
/* Output params:      n/a                       */
/* ********************************************* */
void lcd_initLcd(void);


/* ********************************************* */
/* Method name:        lcd_writeData             */
/* Method description: Write data to be displayed   */
```

```
/* Input params:       ucData => char to be written */
/* Output params:      n/a                          */
/* ******************************************** */
void lcd_writeData(unsigned char ucData);


/* ******************************************** */
/* Method name:        lcd_sendCommand              */
/* Method description: Write command to LCD         */
/* Input params:       ucCmd=>command to be executed*/
/* Output params:      n/a                          */
/* ******************************************** */
void lcd_sendCommand(unsigned char ucCmd);


/* ******************************************** */
/* Method name:        lcd_WriteString              */
/* Method description: Write string to be displayed */
/* Input params:       cBuffer => string to be      */
/*                     written in LCD               */
/* Output params:      n/a                          */
/* ******************************************** */
void lcd_writeString(const char *cBuffer);


/* ******************************************** */
/* Method name:        lcd_setCursor                */
/* Method description: Set cursor line and column   */
/* Input params:       cLine = LINE0..LINE1         */
/*                     cColumn = COLUMN0..MAX_COLUMN*/
/* Output params:      n/a                          */
/* ******************************************** */
void lcd_setCursor(unsigned char cLine, unsigned char cColumn);


/* ******************************************** */
/* Method name:        lcd_dummyText                */
/* Method description: Write a dummy hard coded text*/
/* Input params:       n/a                          */
/* Output params:      n/a                          */
```

```
/* ********************************************** */
void lcd_dummyText(void);

/* ********************************************** */
/* Method name:        lcd_writeText              */
/* Method description: Write text sent by user, on  */
/*                     especified line            */
/* Input params:       int iL, line // chr* cString */
/* Output params:      n/a                        */
/* ********************************************** */
void lcd_writeText(int iL, char* cString );

#endif /* SOURCES_LCD_H_ */
```

lcd.c

```
/* ************************************************************** */
/* File name:        lcd.c                                        */
/* File description: file containing the implementation of functions */
/*                   used to control the LCD display              */
/* Author name:      Caio Villela, Hebert Wandick                 */
/* Creation date:    10/apr/2020                                  */
/* Revision date:    10/apr/2020                                  */
/* ************************************************************** */

#include "lcd.h"
//#include "KL25Z/es670_peripheral_board.h" //orginal
#include "board.h"
//#include "Util/util.h" //Original
#include "util.h"
//compiling with 2 warnings, by integer truncated unsigned type
/* system includes */
#include "fsl_clock_manager.h"
#include "fsl_port_hal.h"
#include "fsl_gpio_hal.h"

/* line and columns */
#define LINE0       0U
#define COLUMN0     0U
```

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

/* ********************************************** */
/* Method name:        lcd_initLcd                */
/* Method description: Initialize the LCD function  */
/* Input params:       n/a                        */
/* Output params:       n/a                       */
/* ********************************************** */
void lcd_initLcd(void)
{
    /* pins configured as outputs */

    /*un-gateport C clock*/
    SIM_SCGC5|=0x0800;

    /*set pins as gpio*/
    PORTC_PCR8|= 0x100; /*RS*/
    PORTC_PCR9|= 0x100; /*RENABLE*/
    PORTC_PCR0|= 0x100; /*RD0*/
    PORTC_PCR1|= 0x100; /*RD1*/
    PORTC_PCR2|= 0x100; /*RD2*/
    PORTC_PCR3|= 0x100; /*RD3*/
    PORTC_PCR4|= 0x100; /*RD4*/
    PORTC_PCR5|= 0x100; /*RD5*/
    PORTC_PCR6|= 0x100; /*RD6*/
    PORTC_PCR7|= 0x100; /*RD7*/


    /* set pins as digital output */
    GPIOC_PDDR |= LCD_DATA_DB0_DIR;
    GPIOC_PDDR |= LCD_DATA_DB1_DIR;
    GPIOC_PDDR |= LCD_DATA_DB2_DIR;
    GPIOC_PDDR |= LCD_DATA_DB3_DIR;
    GPIOC_PDDR |= LCD_DATA_DB4_DIR;
    GPIOC_PDDR |= LCD_DATA_DB5_DIR;
    GPIOC_PDDR |= LCD_DATA_DB6_DIR;
    GPIOC_PDDR |= LCD_DATA_DB7_DIR;
    GPIOC_PDDR |= LCD_RS_DIR;
```

```
    GPIOC_PDDR |= LCD_ENABLE_DIR;

    // 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);

}

/* ********************************************** */
/* Method name:        lcd_write2Lcd              */
/* Method description: Send command or data to LCD  */
/* Input params:       ucBuffer => char to be sent  */
/*                     cDataType=>command LCD_RS_CMD*/
/*                              or data LCD_RS_DATA */
/* Output params:      n/a                        */
/* ********************************************** */
void lcd_write2Lcd(unsigned char ucBuffer,  unsigned char cDataType)
{
    /* writing data or command */
    if(LCD_RS_CMD == cDataType)
        /* will send a command */
        GPIOC_PCOR |= LCD_RS_DIR;
    else
        /* will send data */
        GPIOC_PSOR |= LCD_RS_DIR;

    /* write in the LCD bus */
    GPIOC_PDOR |= (((ucBuffer & (1u << 0u)) & (ucBuffer & (1u << 1u)) &
                    (ucBuffer & (1u << 2u)) & (ucBuffer & (1u << 3u))) &
```

```
                  ((ucBuffer & (1u << 4u)) & (ucBuffer & (1u << 5u)) &
                  (ucBuffer & (1u << 6u)) & (ucBuffer & (1u << 7u))));

    /* enable, delay, disable LCD */
    /* this generates a pulse in the enable pin */
    GPIOC_PSOR |= LCD_ENABLE_DIR;
    util_genDelay1ms();
    GPIOC_PCOR |= LCD_ENABLE_DIR;
    util_genDelay1ms();
    util_genDelay1ms();
}

/* ********************************************** */
/* Method name:        lcd_writeData              */
/* Method description: Write data to be displayed */
/* Input params:       ucData => char to be written */
/* Output params:      n/a                        */
/* ********************************************** */
void lcd_writeData(unsigned char ucData)
{
    /* just a relay to send data */
    lcd_write2Lcd(ucData, LCD_RS_DATA);
}

/* ********************************************** */
/* Method name:        lcd_sendCommand            */
/* Method description: Write command to LCD       */
/* Input params:       ucCmd=>command to be executed*/
/* Output params:      n/a                        */
/* ********************************************** */
void lcd_sendCommand(unsigned char ucCmd)
{
    /* just a relay to send command */
    lcd_write2Lcd(ucCmd, LCD_RS_CMD);
}

/* ********************************************** */
/* Method name:        lcd_setCursor              */
/* Method description: Set cursor line and column */
/* Input params:       cLine = LINE0..LINE1       */
```

```
/*                      cColumn = COLUMN0..MAX_COLUMN*/
/* Output params:        n/a                          */
/* ******************************************** */
void lcd_setCursor(unsigned char cLine, unsigned char cColumn)
{
    char cCommand;

    if(LINE0 == cLine)
        /* line 0 */
        cCommand = L0C0_BASE;
    else
        /* line 1 */
        cCommand = L1C0_BASE;

    /* maximum MAX_COLUMN columns */
    cCommand += (cColumn & MAX_COLUMN);

    // send the command to set the cursor
    lcd_sendCommand(cCommand);
}

/* ******************************************** */
/* Method name:         lcd_writeString         */
/* Method description: Write string to be displayed */
/* Input params:        cBuffer => string to be      */
/*                      written in LCD               */
/* Output params:       n/a                          */
/* ******************************************** */
void lcd_writeString(const char *cBuffer)
{
    while(*cBuffer)
    {
        lcd_writeData(*cBuffer++);
    };
}

/* ******************************************** */
/* Method name:         lcd_dummyText           */
/* Method description: Write a dummy hard coded text*/
/* Input params:        n/a                          */
```

```
/* Output params:      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");
}

/* ******************************************** */
/* Method name:        lcd_writeText            */
/* Method description: Write text sent by user, on  */
/*                     especified line          */
/* Input params:       int iL, line // chr* cString */
/* Output params:      n/a                          */
/* ******************************************** */
void lcd_writeText(int iL, char* cString )
{
    // clear LCD
    lcd_sendCommand(CMD_CLEAR);

    //set cursor to line iL, column 0
    lcd_setCursor(iL, 0);

    // send string
    lcd_writeString(cString);

}
```

ledSwi.h

```
/* ************************************************************** */
/* File name:        ledSwi.h                                     */
/* File description: Header file containing the functions needed to  */
/*                   initialize, switch in between and operate RGB   */
/*                   leds and buttons                             */
/*                                                                */
/* Author name:      Caio Villela/ Hebert Wandick                 */
/* Creation date:    02apr2020                                    */
/* Revision date:    02apr2020                                    */
/* ************************************************************** */

#ifndef LEDSWI_H
#define LEDSWI_H

/* ********************************************************/
/* Methodname:          initLedButton                        */
/* Method description: Esse metodoso realiza a inicializacao*/
/*                     dos Led e Botoes blugados no comtrole*/
/* Inputparams:        iPinsLed- Um vetor contendo o numero */
/*                     leds e utilizados                    */
/*                     iTamLed- Quantidade de led a serem   */
/*                     incializados                         */
/*                     iPinsButton- Um vetor contendo o     */
/*                     numero botoesutilizados              */
/*                     iTamButton- Quantidade de botoes     */
/*                     a serem incializados                 */
/* Outputparams:       n/a                                  */
/********************************************************/
void initLedButton(int *iPinsLed, int iTamLed,int *iPinsButton, int
iTamButton);

/* ********************************************************/
/* Methodname:          readButton                           */
/* Method description: Esse metodo realiza a leitura de um  */
/*                     botao em especifico                  */
/* Inputparams:        iPin- Numero do botao a ser lido     */
/* Outputparams:       0 caso seja lido nivel logico baixo  */
/*                     1 caso seja lido nivel logico alto   */
/********************************************************/
int readButton(int iPin);
```

```
/* ************************************************************/
/* Methodname:          ligaLed                                */
/* Method description:  Liga un led                            */
/* Inputparams:         iPin- Numero do led a ser ligado       */
/* Outputparams:        n/a                                    */
/************************************************************/
void ligaLed(int iPin);

/* ************************************************************/
/* Methodname:          desligaLed                             */
/* Method description:  Desliga led                            */
/* Inputparams:         iPin- Numero do led a ser desligado    */
/* Outputparams:        n/a                                    */
/************************************************************/
void desligaLed(int iPin);

/* ************************************************************/
/* Methodname:          toggleLed                              */
/* Method description:  Deixa o pino puglado o led             */
/*                      desconetado do circtuto eletrico       */
/*                      externo                                */
/* Inputparams:         iPin- Numero do led a ser desconetado*/
/*                      do controlador                         */
/* Outputparams:        n/a                                    */
/************************************************************/
void toggleLed(int iPin);

/* ************************************************************/
/* Methodname:          writeLed                               */
/* Method description:  Congifura o nivel logic do pino        */
/*                      ligado o led para nivel alto ou baixo*/
/*                      externo                                */
/* Inputparams:         iPin- Numero do pino a ter o nivel     */
/*                      logico alterado                        */
/*                      iNivel- nivel logico a ser aplicado    */
/* Outputparams:        n/a                                    */
/************************************************************/
void writeLed(int iPin, int iNivel);
```

```
#endif
```

ledSwi.c

```
/* ************************************************************** */
/* File name:         ledSwi.c                                    */
/* File description:  This file implements the functions needed to */
/*                    initialize, switch in between and operate RGB */
/*                    leds and buttons                            */
/*                                                                */
/* Author name:       Caio Villela/ Hebert Wandick                */
/* Creation date:     02apr2020                                   */
/* Revision date:     03apr2020                                   */
/* ************************************************************** */

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


/*************************************************************/
/* Method name:           initLedButton                      */
/* Method description:    This method initializes given      */
/*                        LEDs and buttons, and sets them    */
/*                        as digital outputs/inputs          */
/* Input params:          *iPinsLed, defines what pins the LED*/
/*                        are set to.                        */
/*                        iTamLed, number of LEDS to be set  */
/*                        *iPinsButton, defines what pins the */
/*                        buttons are set to.                */
/*                        iTamBautton, number of buttons to be*/
/*                        set                                */
/* Output params:         n/a                                */
/*************************************************************/

void initLedButton(int *iPinsLed, int iTamLed,int *iPinsButton, int
iTamButton){
    int iI = 0; /*integer iterator*/
```

```
    /*un-gateport A clock*/
    SIM_SCGC5|=0x200;

    /*initialize selected Leds as GPIO and set them as digital output*/
    if(iTamLed != 0){
        for(iI = 0; iI < iTamLed; iI ++){
            if(1 == iPinsLed[iI]){
                PORTA_PCR1 |= 0x100;
                GPIOA_PDDR |= LED1_MASK;
            }else if(2 == iPinsLed[iI]){
                PORTA_PCR2 |= 0x100;
                GPIOA_PDDR |= LED2_MASK;
            }else if(3 == iPinsLed[iI]){
                PORTA_PCR4 |= 0x100;
                GPIOA_PDDR |= LED3_MASK;
            }else if(4 == iPinsLed[iI]){
                PORTA_PCR5 |= 0x100;
                GPIOA_PDDR |= LED4_MASK;
            }
        }
    }

    /*initialize selected buttons as GPIO and set them as digital
input*/
    if(iTamButton != 0){
        for(iI = 0; iI < iTamButton; iI ++){
            if(1 == iPinsButton[iI]){
                PORTA_PCR1 |= 0x100;
            }else if(2 == iPinsButton[iI]){
                PORTA_PCR2 |= 0x100;
            }else if(3 == iPinsButton[iI]){
                PORTA_PCR4 |= 0x100;
            }else if(4 == iPinsButton[iI]){
                PORTA_PCR5 |= 0x100;
            }
        }
    }
}
```

```
/*************************************************************/
/* Method name:           toggleLed                          */
/* Method description:    This method toggles the selected LED*/
/* Input params:          iPin, defines what pin is to be    */
/*                        toggled.                           */
/* Output params:         n/a                                */
/*************************************************************/

void toggleLed(int iPin)
{
    /* toggle PTAx*/
    /* PTOR : Port Toggle Output Register */
    switch (iPin)
    {
    case 1:
        GPIOA_PTOR |= LED1_MASK;
        break;
    case 2:
        GPIOA_PTOR |= LED2_MASK;
        break;
    case 3:
        GPIOA_PTOR |= LED3_MASK;
        break;
    case 4:
        GPIOA_PTOR |= LED4_MASK;
        break;
    };
}


/*************************************************************/
/* Method name:           writeLed                           */
/* Method description:    writes certain value to a LED pin  */
/* Input params:          iPin, defines what pin is to be    */
/*                        set.                               */
/*                        iNivel, defines what level it is to */
/*                        be set to                          */
/* Output params:         n/a                                */
/*************************************************************/
```

```
void writeLed(int iPin, int iNivel)
{
    /* set PTAx*/
    /* PSOR : Port Set Output Register */
    /* GPIOA_PSOR |= turn HIGH the correponting output pin -> OFF */
    /* GPIOA_PCOR |= turn LOW the correponting output pin -> ON   */

    if (iNivel)
    {
        switch (iPin)
        {
        case 1:
            GPIOA_PSOR|= LED1_MASK;
            break;
        case 2:
            GPIOA_PSOR|= LED2_MASK;
            break;
        case 3:
            GPIOA_PSOR|= LED3_MASK;
            break;
        case 4:
            GPIOA_PSOR|= LED4_MASK;
            break;
        };
    }
    else
    {
      switch(iPin)
      {
        case 1:
            GPIOA_PCOR|= LED1_MASK;
            break;
        case 2:
            GPIOA_PCOR|= LED2_MASK;
            break;
        case 3:
            GPIOA_PCOR|= LED3_MASK;
            break;
        case 4:
            GPIOA_PCOR|= LED4_MASK;
```

```
            break;
        };
    }
}


/*************************************************************/
/* Method name:           ligaLed                            */
/* Method description:    turns led ON                       */
/* Input params:          iPin, defines what pin is to be    */
/*                        turned on.                         */
/* Output params:         n/a                                */
/*************************************************************/

void ligaLed(int iPin){
    /*call the function responsible by set set pin output level to
HIGH*/
    writeLed(iPin,0);
};

/*************************************************************/
/* Method name:           desligaLed                         */
/* Method description:    turns led OFF                      */
/* Input params:          iPin, defines what pin is to be    */
/*                        turned off.                        */
/* Output params:         n/a                                */
/*************************************************************/


void desligaLed(int iPin){
    /*call the function responsible by set set pin output level to
LOW*/
    writeLed(iPin,1);
}


/* ***********************************************************/
/* Methodname:         readButton                           */
/* Method description: Esse metodo realiza a leitura de um  */
/*                     botao em especifico                  */
```

```
/* Inputparams:        iPin- Numero do botao a ser lido     */
/* Outputparams:       0 caso seja lido nivel logico baixo  */
/*                     1 caso seja lido nivel logico alto   */
/*********************************************************/

int readButton(int iPin){
    int iInput; /*32 bit number showing the inputs*/

    iInput = GPIOA_PDIR;

    /*configures iPin as 32 bit pinnage regarding it's button*/
    switch(iPin){
        case 1:
            iPin = BUTTON1_MASK;
        case 2:
            iPin = BUTTON2_MASK;
        case 3:
            iPin = BUTTON3_MASK;
        case 4:
            iPin = BUTTON4_MASK;
    }

    /*bit mask, returns 1 if bit is turned off, returns 0 otherwise*/
    switch(iInput & iPin){
      case 1:
        return(0);
      case 0:
        return(1);
    }
    return 0;
}
```

lptmr.h

```
/* ************************************************************** */
/* 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_ */
```

lptmr.c

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

```
#include "lptmr.h"

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

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

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

/* LPTMR driver state information */
lptmr_state_t lptmrState;

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

/* ********************************************** */
/* Method name:        tc_installLptmr            */
/* Method description: Low power timer 0          */
/*                     initialization and start   */
/* Input params:       uiTimeInUs:                */
/*                     time in micro seconds      */
/*                     tUserCallback              */
/*                     function pointer to be called*/
```

```
/*                    when counter achieves        */
/*                    uiTimeInUs                   */
/* Output params:     n/a                          */
/* ********************************************** */
void tc_installLptmr0(uint32_t uiTimeInUs, lptmr_callback_t
tUserCallback)
{
    /* Initialize LPTMR */
    LPTMR_DRV_Init(LPTMR0_IDX, &lptmrState, &lptmrConfig);

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

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

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

lut_adc_3v3.h

```
/* ********************************************************************
*/
/* 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_

const unsigned char tabela_temp[256];

#endif /* SOURCES_ADC_LUT_ADC_3V3_H_ */
```

lut_adc_3v3.c

```
/* ****************************************************************
*/
/* 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, 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
};
```

main.c

```
/* ************************************************************** */
/* File name:        main.c                                       */
/* File description: implements UART configuration and interruption  */
/*                                                                */
/* Author name:      Caio Villela/ Hebert Wandick                 */
```

```
/* Creation date:    30/may/2020                                          */
/* Revision date:    30/may/2020                                          */
/* ********************************************************************* */

/*My includes*/
#include "TempSensor.h"
#include "lcd.h"
#include "util.h"
#include "UART.h"
#include "print_scan.h"
#include "aquecedorECooler.h"
#include "pid.h"
#include "ledSwi.h"
#include "communicationStateMachine.h"
#include "lptmr.h"

bool bFlag = false;

/* ***********************************************  */
/* Method name:        main_cyclicExecuteIsr        */
/* Method description: lets program continue running */
/*                     upon timer reaching its limit */
/* Input params:       n/a                          */
/* Output params:      n/a                          */
/* ***********************************************  */
void main_cyclicExecuteIsr(void)
{
    bFlag = true;
}

int main(void)
{

    /*init GPIOS*/
    int *iLeds = (int *)NULL; /* no leds*/
    int iNumLeds = 0;
    /*using the 1°,2°,3° buttons*/
    int iButtons[3] = {1, 2, 3};
    int iNumButtons = 3;
    initLedButton(iLeds, iNumLeds, iButtons, iNumButtons);
```

```
    bool press = false;

    /*init UART0*/
    UART0_init();
    UART0_enableIRQ();

    /*cooler and fan control*/
    PWM_init();
    coolerfan_init();
    heater_init();
    coolerfan_PWMDuty(0); /*turn off fan*/
    heater_PWMDuty(0); /*turn off heater*/

    /*init Temperature sensor*/
    initTempSensor();

    /*init lcd*/
    void lcd_initLcd(void);

    /*init PID with interruptions*/
    pid_init();
    pid_setSetValue(20);
    tc_installLptmr0(100000, main_cyclicExecuteIsr);

    /*Updade Screen*/
    setScreen();

    while (true)
    {
        /*read buttons and put on statemachine*/
        if (!press)
        {
            if (readButton(1))
            {
                processByteCommunication('@');
                press = true;
            }
            else if (readButton(2))
            {
                processByteCommunication('<');
```

```
                press = true;
            }
            else if (readButton(3))
            {
                processByteCommunication('>');
                press = true;
            }
        }
        else
        {
            if (!readButton(1) && !readButton(2) && readButton(3))
                press = false;
        }

        /*PID LOOP*/
        if (bFlag)
        {
            heater_PWMDuty(pidUpdateData(getTemp())/100);
            bFlag = false;
            /*Updade Screen*/
            setScreen();
        }
    }
    /* Never leave main */
    return 0;
}
```

pid.h

```
/* ************************************************************** */
/* 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_setSetValue                */
/* Method description: set temperature target in PID  */
```

```
/* Input params:       fValue: the temperature        */
/* Output params:      n/a                            */
/* ********************************************** */
void  pid_setSetValue(float fValue);

/* ********************************************** */
/* Method name:        pid_getSetValue                */
/* Method description: get temperature target in PID  */
/* Input params:       n/a                            */
/* Output params:      float: temperature             */
/* ********************************************** */
float pid_getSetValue();

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


#endif /* SOURCES_CONTROLLER_PID_H_ */
```

pid.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, Hebert Wandick  */
/*                   Caio Villela                                 */
/* Creation date:    21jun2018                                    */
/* Revision date:    01gos2020                                    */
```

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

#include "pid.h"

pid_data_type pidConfig;
float fSetValue=0;

/* ********************************************** */
/* 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_setSetValue                  */
/* Method description: set temperature target in PID   */
/* Input params:        fValue: the temperature         */
/* Output params:       n/a                              */
/* ************************************************** */
void  pid_setSetValue(float fValue){
    fSetValue=( fValue > 77 ) ? 77 : fValue; /*max at 77 */
}

/* ************************************************** */
/* Method name:         pid_getSetValue                  */
/* Method description: get temperature target in PID   */
/* Input params:        n/a                              */
/* Output params:       float: temperature               */
/* ************************************************** */
float pid_getSetValue(){
    return fSetValue;
};
```

```
/* ************************************************ */
/* Method name:        pid_updateData               */
/* Method description: Update the control output    */
/*                     using the reference and sensor */
/*                     value                        */
/* Input params:       fSensorValue: Value read from  */
/*                     the sensor                   */
/* Output params:      float: New Control effort     */
/* ************************************************ */
float pidUpdateData(float fSensorValue)
{
    float fError, fDifference, fOut;
    static int iStaturationFlag;

    fError = fSetValue - fSensorValue;

    /* if there is no saturation */
    if( iStaturationFlag == 0)
        pidConfig.fError_sum += fError;

    fDifference = pidConfig.fError_previous - fError;

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

    pidConfig.fError_previous = fError;

    if (fOut>100.0){
        fOut = 100.0;
        iStaturationFlag = 1;
    }

    else if (fOut<0.0){
        fOut = 0.0;
        iStaturationFlag = 1;
    }

    else
        iStaturationFlag = 0;
```

```
    return fOut;
}
```

print_scan.h

```
/*
 * Copyright (c) 2013 - 2014, Freescale Semiconductor, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright
notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright
notice, this
 *   list of conditions and the following disclaimer in the
documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of Freescale Semiconductor, Inc. nor the names of
its
 *   contributors may be used to endorse or promote products derived
from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS
BE LIABLE FOR
```

```
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
#ifndef __print_scan_h__
#define __print_scan_h__

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

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

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

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

/*!
 * @brief This function outputs its parameters according to a formatted
string.
 *
 * @note I/O is performed by calling given function pointer using
following
 * (*func_ptr)(c,farg);
 *
 * @param[in] farg      Argument to func_ptr.
```

```
 * @param[in] func_ptr  Function to put character out.
 * @param[in] max_count Maximum character count for snprintf and
vsnprintf.
 * Default value is 0 (unlimited size).
 * @param[in] fmt_ptr   Format string for printf.
 * @param[in] args_ptr  Arguments to printf.
 *
 * @return Number of characters
 * @return EOF (End Of File found.)
 */
int _doprint(void *farg, PUTCHAR_FUNC func_ptr, int max_count, char
*fmt, va_list ap);

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

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

```
#endif
```

print_scan.c

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

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

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

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

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

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

#define IS_LENMOD_h(a)   (a & LENMOD_h)
#define IS_LENMOD_hh(a)  (a & LENMOD_hh)
#define IS_LENMOD_l(a)   (a & LENMOD_l)
#define IS_LENMOD_ll(a)  (a & LENMOD_ll)
#define IS_LENMOD_L(a)   (a & LENMOD_L)

#define SCAN_SUPPRESS               0x2

#define SCAN_DEST_MASK              0x7c
```

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

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

#define SCAN_TYPE_SIGNED            0x2000

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

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

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

 * @return Length of the converted string.
 */
```

```
static int32_t mknumstr (char *numstr, void *nump, int32_t neg, int32_t
radix, bool use_caps);

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

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



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

double modf(double input_dbl, double  *intpart_ptr);

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

```
/*FUNCTION*****************************************************************
*********
 *
 * Function Name : _doprint
 * Description   : This function outputs its parameters according to a
 * formatted string. I/O is performed by calling given function pointer
 * using following (*func_ptr)(c,farg);
 *

*END**********************************************************************
*******/
int _doprint(void *farg, PUTCHAR_FUNC func_ptr, int max_count, char
*fmt, va_list ap)
{
    /* va_list ap; */
    char *p;
    int32_t c;

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

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


    uint32_t flags_used;
    uint32_t field_width;

    int32_t ival;
    int32_t schar, dschar;
    int32_t *ivalp;
    char *sval;
    int32_t cval;
    uint32_t uval;
    bool use_caps;
    uint32_t precision_width;
    //uint32_t length_modifier = 0;
```

```
#if defined(PRINTF_FLOAT_ENABLE)
    double fval;
#endif

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

    /*
     * Start parsing apart the format string and display appropriate
     * formats and data.
     */
    for (p = (char *)fmt; (c = *p) != 0; p++)
    {
        /*
         * All formats begin with a '%' marker.  Special chars like
         * '\n' or '\t' are normally converted to the appropriate
         * character by the __compiler__.  Thus, no need for this
         * routine to account for the '\' character.
         */
        if (c != '%')
        {
            n_putchar(func_ptr, c, farg, &max_count);

            count++;

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

        /*
         * First check for specification modifier flags.
         */
        use_caps = true;
        flags_used = 0;
        done = false;
```

```
        while (!done)
        {
            switch (/* c = */ *++p)
            {
                case '-':
                    flags_used |= FLAGS_MINUS;
                    break;
                case '+':
                    flags_used |= FLAGS_PLUS;
                    break;
                case ' ':
                    flags_used |= FLAGS_SPACE;
                    break;
                case '0':
                    flags_used |= FLAGS_ZERO;
                    break;
                case '#':
                    flags_used |= FLAGS_POUND;
                    break;
                default:
                    /* we've gone one char too far */
                    --p;
                    done = true;
                    break;
            }
        }

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

```
                case '4':
                case '5':
                case '6':
                case '7':
                case '8':
                case '9':
                    field_width = (field_width * 10) + (c - '0');
                    break;
                default:
                    /* we've gone one char too far */
                    --p;
                    done = true;
                    break;
            }
        }

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

            /*
             * Must get precision field width, if present.
             */
            precision_width = 0;
            done = false;
            while (!done)
            {
                switch (c = *++p)
                {
                    case '0':
                    case '1':
                    case '2':
                    case '3':
                    case '4':
                    case '5':
                    case '6':
```

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

        /*
         * Check for the length modifier.
         */
        /* length_modifier = 0; */
        switch (/* c = */ *++p)
        {
            case 'h':
                if (*++p != 'h')
                {
                    --p;
                }
                /* length_modifier |= LENMOD_h; */
                break;
            case 'l':
                if (*++p != 'l')
                {
                    --p;
                }
                /* length_modifier |= LENMOD_l; */
                break;
```

```
            case 'L':
                /* length_modifier |= LENMOD_L; */
                break;
            default:
                /* we've gone one char too far */
                --p;
                break;
        }

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

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

```
                        {
                            schar = 0;
                        }
                    }
                }
                dschar = false;

                /*
                 * do the ZERO pad.
                 */
                if (IS_FLAG_ZERO(flags_used))
                {
                    if (schar)
                    {
                        n_putchar(func_ptr, schar, farg, &max_count);
                        count++;
                    }
                    dschar = true;

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

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

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

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

```
                vlen = mknumstr(vstr,&uval,false,16,use_caps);
                vstrp = &vstr[vlen];

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

                            dschar = true;
                        }
                    }
                }
```

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

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

                cont_u:
                    vstrp = &vstr[vlen];

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

```
                        {
                            fput_pad(' ', vlen, field_width, &count,
func_ptr, farg, &max_count);
                        }
                    }

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

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

            case 'c':
                cval = (char)va_arg(ap, uint32_t);
                n_putchar(func_ptr, cval, farg, &max_count);
                count++;
                break;
            case 's':
                sval = (char *)va_arg(ap, char *);
                if (sval)
                {
                    vlen = strlen(sval);
                    if (!IS_FLAG_MINUS(flags_used))
                    {
                        fput_pad(' ', vlen, field_width, &count,
func_ptr, farg, &max_count);
                    }
                    while (*sval)
                    {
                        n_putchar(func_ptr, *sval++, farg, &max_count);
                        count++;
```

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

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

/*FUNCTION*****************************************************************
*********
 *
 * Function Name : _sputc
 * Description   : Writes the character into the string located by the
string
 * pointer and updates the string pointer.
 *

*END**********************************************************************
*******/
```

```
int _sputc(int c, void * input_string)
{
   char **string_ptr = (char  **)input_string;

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

/*FUNCTION*****************************************************************
 *
 * Function Name : mknumstr
 * Description   : Converts a radix number to a string and return its
length.
 *

*END*********************************************************************
*******/
static int32_t mknumstr (char *numstr, void *nump, int32_t neg, int32_t
radix, bool use_caps)
{
    int32_t a,b,c;
    uint32_t ua,ub,uc;

    int32_t nlen;
    char *nstrp;

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

    if (neg)
    {
        a = *(int32_t *)nump;
        if (a == 0)
        {
            *nstrp = '0';
            ++nlen;
            goto done;
        }
```

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

```
            ++nlen;
        }
    }
    done:
    return nlen;
}

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

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

    int32_t nlen;
    char *nstrp;
    nlen = 0;
    nstrp = numstr;
    *nstrp++ = '\0';
    r = *(double *)nump;
    if (r == 0)
    {
        *nstrp = '0';
        ++nlen;
        goto done;
    }
    fractpart = modf((double)r , (double *)&intpart);
    /* Process fractional part */
```

```
    for (i = 0; i < precision_width; i++)
    {
        fractpart *= radix;
    }
    //a = (int32_t)floor(fractpart + (double)0.5);
    fa = fractpart + (double)0.5;
    for (i = 0; i < precision_width; i++)
    {
        fb = fa / (int32_t)radix;
        c = (int32_t)(fa - (uint64_t)fb * (int32_t)radix);
        if (c < 0)
        {
            c = ~c + 1 + '0';
        }else
        {
            c = c + '0';
        }
        fa = fb;
        *nstrp++ = (char)c;
        ++nlen;
    }
    *nstrp++ = (char)'.';
    ++nlen;
    a = (int32_t)intpart;
    while (a != 0)
    {
        b = (int32_t)a / (int32_t)radix;
        c = (int32_t)a - ((int32_t)b * (int32_t)radix);
        if (c < 0)
        {
            c = ~c + 1 + '0';
        }else
        {
            c = c + '0';
        }
        a = b;
        *nstrp++ = (char)c;
        ++nlen;
    }
    done:
```

```
    return nlen;
}
#endif

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

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

/*FUNCTION*****************************************************************
*********
 *
 * Function Name : scan_prv
 * Description   : Converts an input line of ASCII characters based
upon a
 * provided string format.
 *

*END**********************************************************************
*******/
int scan_prv(const char *line_ptr, char *format, va_list args_ptr)
{
    uint8_t base;
    /* Identifier for the format string */
    char *c = format;
    const char *s;
    char temp;
    /* Identifier for the input string */
    const char *p = line_ptr;
    /* flag telling the conversion specification */
    uint32_t flag = 0 ;
    /* filed width for the matching input streams */
    uint32_t field_width;
```

```
    /* how many arguments are assigned except the suppress */
    uint32_t nassigned = 0;
    /* how many characters are read from the input streams */
    uint32_t n_decode = 0;

    int32_t val;
    char *buf;
    int8_t neg;

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

    /* decode directives */
    while ((*c) && (*p))
    {
        /* ignore all white-spaces in the format strings */
        if (scan_ignore_white_space((const char **)&c))
        {
            n_decode += scan_ignore_white_space(&p);
        }
        else if (*c != '%')
        {
            /* Ordinary characters */
            c++;
ordinary:   if (*p == *c)
            {
                n_decode++;
                p++;
                c++;
            }
            else
            {
                /* Match failure. Misalignment with C99, the unmatched
                 * characters need to be pushed back to stream. HOwever
                 * , it is deserted now. */
                break;
            }
```

```
        }
        else
        {
            /* convernsion specification */
            c++;
            if (*c == '%')
            {
                goto ordinary;
            }

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

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

                        if (c[1] == 'h')
                        {
                            flag |= SCAN_LENGTH_CHAR;
```

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

                        if (c[1] == 'l')
                        {
                            flag |= SCAN_LENGTH_LONG_LONG_INT;
                            c++;
                        }
                        c++;
                        break;
#if defined(ADVANCE)
                    case 'j':
                        if (flag & SCAN_LENGTH_MASK)
                        {
                            /* Match failure*/
                            return nassigned;
                        }
                        flag |= SCAN_LENGTH_INTMAX;
                        c++
                    case 'z'
                        if (flag & SCAN_LENGTH_MASK)
                        {
                            /* Match failure*/
                            return nassigned;
                        }
                        flag |= SCAN_LENGTH_SIZE_T;
                        c++;
                        break;
                    case 't':
                        if (flag & SCAN_LENGTH_MASK)
                        {
```

```
                            /* Match failure*/
                            return nassigned;
                        }
                        flag |= SCAN_LENGTH_PTRDIFF_T;
                        c++;
                        break;
#endif
#if defined(SCANF_FLOAT_ENABLE)
                    case 'L':
                        if (flag & SCAN_LENGTH_MASK)
                        {
                            /* Match failure*/
                            return nassigned;
                        }
                        flag |= SCAN_LENGTH_LONG_DOUBLE;
                        c++;
                        break;
#endif
                    case '0':
                    case '1':
                    case '2':
                    case '3':
                    case '4':
                    case '5':
                    case '6':
                    case '7':
                    case '8':
                    case '9':
                        if (field_width)
                        {
                            /* Match failure*/
                            return nassigned;
                        }
                        do {
                            field_width = field_width * 10 + *c - '0';
                            c++;
                        } while ((*c >= '0') && (*c <= '9'));
                        break;
                    case 'd':
                        flag |= SCAN_TYPE_SIGNED;
```

```
                    case 'u':
                        base = 10;
                        flag |= SCAN_DEST_INT;
                        c++;
                        break;
                    case 'o':
                        base = 8;
                        flag |= SCAN_DEST_INT;
                        c++;
                        break;
                    case 'x':
                    case 'X':
                        base = 16;
                        flag |= SCAN_DEST_INT;
                        c++;
                        break;
                    case 'i':
                        base = 0;
                        flag |= SCAN_DEST_INT;
                        c++;
                        break;
#if defined(SCANF_FLOAT_ENABLE)
                    case 'a':
                    case 'A':
                    case 'e':
                    case 'E':
                    case 'f':
                    case 'F':
                    case 'g':
                    case 'G':
                        flag |= SCAN_DEST_FLOAT;
                        c++;
                        break;
#endif
                    case 'c':
                        flag |= SCAN_DEST_CHAR;
                        if (!field_width)
                        {
                            field_width = 1;
                        }
```

```
                        c++;
                        break;
                    case 's':
                        flag |= SCAN_DEST_STRING;
                        c++;
                        break;
#if defined(ADVANCE) /* [x]*/
                    case '[':
                        flag |= SCAN_DEST_SET;
                        /*Add Set functionality */
                        break;
#endif
                    default:
#if defined(SCAN_DEBUG)
                        printf("Unrecognized expression specifier: %c
format: %s, number is: %d\r\n", c, format, nassigned);
#endif
                        return nassigned;
                }
            }

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

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

            /* Matching strings in input streams and assign to argument
*/
            switch (flag & SCAN_DEST_MASK)
            {
                case SCAN_DEST_CHAR:
                    s = (const char *)p;
                    buf = va_arg(args_ptr, char *);
```

```
                    while ((field_width--) && (*p))
                    {
                        if (!(flag & SCAN_SUPPRESS))
                        {
                            *buf++ = *p++;
                        }
                        else
                        {
                            p++;
                        }
                        n_decode++;
                    }

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

                    if ((!(flag & SCAN_SUPPRESS)) && (s != p))
                    {
```

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

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

                    neg = 1;
                    switch (*p)
                    {
```

```
                    case '-':
                        neg = -1;
                        n_decode++;
                        p++;
                        field_width--;
                        break;
                    case '+':
                        neg = 1;
                        n_decode++;
                        p++;
                        field_width--;
                        break;
                    default:
                        break;
                }

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

                    if (temp >= base)
                    {
                        break;
                    }
                    else
```

```
                        {
                            val = base * val + temp;
                        }
                        p++;
                        n_decode++;
                    }

                    val *= neg;
                    if (!(flag & SCAN_SUPPRESS))
                    {
                        switch (flag & SCAN_LENGTH_MASK)
                        {
                            case SCAN_LENGTH_CHAR:
                                if (flag & SCAN_TYPE_SIGNED)
                                {
                                    *va_arg(args_ptr, signed char *) =
(signed char)val;
                                }
                                else
                                {
                                    *va_arg(args_ptr, unsigned char *)
= (unsigned char)val;
                                }
                                break;
                            case SCAN_LENGTH_SHORT_INT:
                                if (flag & SCAN_TYPE_SIGNED)
                                {
                                    *va_arg(args_ptr, signed short *) =
(signed short)val;
                                }
                                else
                                {
                                    *va_arg(args_ptr, unsigned short *)
= (unsigned short)val;
                                }
                                break;
                            case SCAN_LENGTH_LONG_INT:
                                if (flag & SCAN_TYPE_SIGNED)
                                {
```

```
                                    *va_arg(args_ptr, signed long int
*) = (signed long int)val;
                                }
                                else
                                {
                                    *va_arg(args_ptr, unsigned long int
*) = (unsigned long int)val;
                                }
                                break;
                            case SCAN_LENGTH_LONG_LONG_INT:
                                if (flag & SCAN_TYPE_SIGNED)
                                {
                                    *va_arg(args_ptr, signed long long
int *) = (signed long long int)val;
                                }
                                else
                                {
                                    *va_arg(args_ptr, unsigned long
long int *) = (unsigned long long int)val;
                                }
                                break;
                            default:
                                /* The default type is the type int */
                                if (flag & SCAN_TYPE_SIGNED)
                                {
                                    *va_arg(args_ptr, signed int *) =
(signed int)val;
                                }
                                else
                                {
                                    *va_arg(args_ptr, unsigned int *) =
(unsigned int)val;
                                }
                                break;
                        }
                        nassigned++;
                    }
                    break;
#if defined(SCANF_FLOAT_ENABLE)
                case SCAN_DEST_FLOAT:
```

```
                    n_decode += scan_ignore_white_space(&p);
                    fnum = strtod(p, (char **)&s);

                    if ((fnum == HUGE_VAL) || (fnum == -HUGE_VAL))
                    {
                        break;
                    }

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

```
/*FUNCTION*************************************************************
*********
 *
 * Function Name : scan_ignore_white_space
 * Description   : Scanline function which ignores white spaces.
 *

*END*****************************************************************
*******/
static uint32_t scan_ignore_white_space(const char **s)
{
    uint8_t count = 0;
    uint8_t c;

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

TempSensor.h

```
/* ************************************************************** */
/* File name:        TempSensor.h                                 */
/* File description: file containing the description of functions */
/*                   used to control the Temperature sensor       */
/* Author name:      Caio Villela, Hebert Wandick                 */
/* Creation date:    30/May/2020                                  */
/* Revision date:    30/May/2020                                  */
/* ************************************************************** */

#ifndef TEMP_SENSOR_H_
#define TEMP_SENSOR_H_
```

```
/* ********************************************** */
/* Method name:        initTempSensor             */
/* Method description: Init Temperature sensor    */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void initTempSensor(void);

/* ********************************************** */
/* Method name:        getTemp                    */
/* Method description: get Temperature            */
/* Input params:       n/a                        */
/* Output params:      unsiged int with the current */
/*                     temperature                */
/* ********************************************** */
unsigned int getTemp();

#endif
```

TempSensor.c

```
/* ************************************************************** */
/* File name:        TempSensor.c                                 */
/* File description: Measurement Temperature                      */
/* Author name:      Hebert Wandick, Caio Villela                 */
/* Creation date:    30/May/2020                                  */
/* Revision date:    30/May/2020                                  */
/* ************************************************************** */

#include "adc.h"
#include "lut_adc_3v3.h"

/* ********************************************** */
/* Method name:        initTempSensor             */
/* Method description: Init Temperature sensor    */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void initTempSensor(void){
```

```
    adc_initADCModule();
};

/* ********************************************** */
/* Method name:        getTemp                    */
/* Method description: get Temperature            */
/* Input params:       n/a                        */
/* Output params:      unsiged int with the current */
/*                     temperature                */
/* ********************************************** */
unsigned int getTemp(){
    adc_initConvertion();
    while(!adc_isAdcDone());
    int iAux=adc_getConvertionValue();
    return tabela_temp[255-iAux];
};
```

UART.h

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

#ifndef UART_H_
#define UART_H_

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

/* ********************************************** */
/* Method name:        UART0_enableIRQ            */
```

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


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

#endif /* UART_H_ */
```

UART.C

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

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

/* system includes */
#include "fsl_clock_manager.h"
#include "fsl_device_registers.h"
#include "fsl_port_hal.h"
```

```
#include "fsl_smc_hal.h"
#include "fsl_debug_console.h"
#include "communicationStateMachine.h"

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


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

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

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

/* ********************************************** */
/* Method name:        UART0_enableIRQ            */
/* Method description: Enable the interruption for  */
/*                     serial port inputs and       */
```

```
/*                      prepare the buffer           */
/* Input params:        n/a                          */
/* Output params:       n/a                          */
/* ******************************************** */
void UART0_enableIRQ(void)
{
    /* Enable interruption in the NVIC */
    NVIC_EnableIRQ(UART0_IRQn);

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



/* ******************************************** */
/* Method name:         UART0_IRQHandler            */
/* Method description:  Serial port interruption    */
/*                      handler method. It Reads the */
/*                      new character and saves in   */
/*                      the buffer                   */
/* Input params:        n/a                          */
/* Output params:       n/a                          */
/* ******************************************** */
void UART0_IRQHandler(void)
{
    /*Send to State Machine handle*/
    processByteCommunication(debug_getchar());
}
```

util.h

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

/*bool type*/
typedef enum {false, true} bool;

/* ********************************************** */
/* Method name:        floatToUChar                */
/* Method description: converts 6 unsiged chars to 1  */
/*                          float                 */
/* Input params:       ucValue array character to be  */
/*                          converted             */
/*                     frec high decimal unity    */
/* Output params:      1 float                    */
/* **********************************************/
float uCharToFloat(unsigned char *ucValue, float fRec);

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

/* ********************************************** */
/* Method name:        setParam                   */
/* Method description: set the temperature or led in*/
/*                     machine                    */
/*                     uint8_t                    */
/* Input params:       ucParam type of parameter  to*/
/*                     be set                     */
/*                     ucByte array of information  */
/* Output params:      n/a                        */
/* ********************************************** */
void setParam(unsigned char ucParam, unsigned char *ucByte);

/* ********************************************** */
/* Method name:        answerParam                */
/* Method description: return the corresponding   */
```

```
/*                      information                    */
/* Input params:        ucParam type of parameter  to*/
/*                      be get                         */
/* Output params:       n/a                            */
/* ********************************************** */
void answerParam(unsigned char ucParam);

/* ********************************************** */
/* Method name:         setScreen                      */
/* Method description:  print on lcd the current       */
/*                      state  of the equipament       */
/* Input params:        n/a                            */
/* Output params:       n/a                            */
/* ********************************************** */
void setScreen();

#endif /* UTIL_H */
```

util.c

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

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

/*my includes*/
#include "util.h"
```

```c
#include "fsl_debug_console.h"

#include "util.h"
#include "aquecedorECooler.h"
#include "communicationStateMachine.h"
#include "lcd.h"
#include "pid.h"

/*union variable used for converting unsigned chars/floats */
typedef union {
    unsigned char ucBytes[4];
    int iReal;
} floatUCharType;

unsigned char ucTempNow = 't';
floatUCharType varFloatUChar;

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

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

/* ********************************************** */
/* Method name:        util_genDelay250us         */
/* Method description: generates ~   250 micro sec  */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void util_genDelay250us(void)
{
    char i;
    for (i = 0; i < 120; i++)
    {
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
    }
    util_genDelay088us();
    util_genDelay088us();
}

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

```
    util_genDelay250us();
    util_genDelay250us();
    util_genDelay250us();
    util_genDelay250us();
}

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

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

```
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
}

#define HASHTAG 0b00100011
#define LETRA_A 0b01100001
#define PONTO_VIRGULA 0b00111011


/* ************************************************ */
/* Method name:        floatToUChar                 */
/* Method description: converts 6 unsiged chars to 1  */
/*                          float                   */
/* Input params:       ucValue array character to be  */
/*                          converted               */
/*                     frec high decimal unity      */
/* Output params:      1 float                      */
/* ************************************************/
float uCharToFloat(unsigned char *ucValue, float frec)
{
    if (*ucValue == '\0')
        return 0;
    if (*ucValue == ',')
        return uCharToFloat(&(ucValue[1]), frec);
    return frec * ((float)((int)*ucValue - 48)) +
uCharToFloat(&(ucValue[1]), frec / 10);
}

/* ************************************************ */
/* Method name:        floatToUChar                 */
/* Method description: converts 1 float to 4 unsigned */
/*                          chars                   */
/* Input params:       fValue character to be         */
/*                          converted               */
/* Output params:      4 unsigned char array          */
/* ************************************************ */
unsigned char *floatToUChar(float fReceived)
{
```

```
    varFloatUChar.iReal = fReceived;
    return (varFloatUChar.ucBytes);
}

bool bLock = false; /* auv varible to look the keys */

/* ************************************************** */
/* Method name:        setParam                       */
/* Method description: set the temperature or led in  */
/*                           machine                  */
/*                                                    */
/* Input params:     ucParam -> type of parameter to be set */
/*                           ucByte -> array of information */
/* Output params:      n/a                            */
/* ************************************************** */
void setParam(unsigned char ucParam, unsigned char *ucByte)
{
    static float fCooler = 0;
    static float fHeater = 0;

    switch (ucParam)
    {
    case 'b':
        if ('0' == ucByte[0])
        {
            /*turn off button*/
            bLock = false;
        }
        else
        {
            /*turn on button*/
            bLock = true;
        }
        break;
    case 'c':
        /*set resitor power*/
        fCooler = uCharToFloat(ucByte, 100);
        fCooler = (fCooler > 50) ? 0.5 : fCooler / 100;
        coolerfan_PWMDuty(fCooler);
```

```
        break;
    case 'i':
        /*set Ki in PID*/
        pid_setKi(uCharToFloat(ucByte, 100));
        break;
    case 'p':
        /*set Kp in PID*/
        pid_setKi(uCharToFloat(ucByte, 100));
        break;
    case 'd':
        /*set Kd in PID*/
        pid_setKi(uCharToFloat(ucByte, 100));
        break;
    case 's':
        /*set target temperature */
        pid_setSetValue(uCharToFloat(ucByte, 100));
    /*Upcase mean buttons entrace*/
    case 'C':
        if (!bLock)/*check if it'not lock*/
            switch (ucByte[0])
            {
            case 'u':
                coolerfan_PWMDuty(getCoolerDuty() + 0.05);/*add 5%*/
                break;
            case 'd':
                coolerfan_PWMDuty(getCoolerDuty() - 0.05);/*remove 5%*/
                break;
            }
        break;

    case 'T':
        if (!bLock)
            switch (ucByte[0])
            {
            case 'u':
                pid_setSetValue(pid_getSetValue() + 1);/*up 1 grau*/
                break;
            case 'd':
                pid_setSetValue(pid_getSetValue() - 1);/*dou 2 grau*/
                break;
```

```
            }
        break;
    case 'D':
        if (!bLock)
            switch (ucByte[0])
            {
            case 'u':
                pid_setKd(pid_getKd() + 2);/*up 2 units*/
                break;
            case 'd':
                pid_setKd(pid_getKd() - 2);/*dow 2 units*/
                break;
            }
        break;
    case 'I':
        if (!bLock)
            switch (ucByte[0])
            {
            case 'u':
                pid_setKi(pid_getKi() + 2);
                break;
            case 'd':
                pid_setKi(pid_getKi() - 2);
                break;
            }
        break;
    case 'P':
        if (!bLock)

            switch (ucByte[0])
            {
            case 'u':
                pid_setKp(pid_getKp() + 2);
                break;
            case 'd':
                pid_setKp(pid_getKp() - 2);
                break;
            }
        break;
    }
```

```
    /*sending "#a;" to corfirm*/
    debug_putchar(HASHTAG);
    debug_putchar(LETRA_A);
    debug_putchar(PONTO_VIRGULA);
}

/* ********************************************** */
/* Method name:        answerParam                */
/* Method description: return the corresponding   */
/*                             information        */
/* Input params:       ucParam -> what parameter is */
/*                             to be returned     */
/* Output params:      n/a                        */
/* ********************************************** */

void answerParam(unsigned char ucParam)
{
    static float fTempNow = 0;
    static float fTempReq = 0;
    static float fKd = 0;
    static float fKi = 0;
    static float fKp = 0;
    static float fHeater = 0;
    static float fCooler = 0;

    unsigned char *ucValue = malloc(4 * sizeof(unsigned char));
    int iI;

    debug_putchar(HASHTAG);
    debug_putchar(LETRA_A);

    switch (ucParam)
    {
    case 't':
        /* return actual temperature*/
        fTempNow = getTemp();

        ucValue = floatToUChar(fTempNow);
        for (iI = 0; iI < 4; iI++)
```

```
            debug_putchar(ucValue[iI]);
        break;

    case 'c':
        /* return the cooler duty cycle*/
        fCooler = getCoolerDuty();
        ucValue = floatToUChar(fCooler);
        for (iI = 0; iI < 4; iI++)
            debug_putchar(ucValue[iI]);

        break;

    case 'a':
        /*return  the heater duty cycle*/
        fHeater = getHeaterDuty();
        ucValue = floatToUChar(fHeater);
        for (iI = 0; iI < 4; iI++)
            debug_putchar(ucValue[iI]);
        break;
    case 'i':
        /*return  the Ki param of PID*/
        fKi = pid_getKi();
        ucValue = floatToUChar(fKi);
        for (iI = 0; iI < 4; iI++)
            debug_putchar(ucValue[iI]);
        break;
    case 'd':
        /*return  the Kd param of PID*/
        fKd = pid_getKd();
        ucValue = floatToUChar(fKd);
        for (iI = 0; iI < 4; iI++)
            debug_putchar(ucValue[iI]);
        break;
    case 'p':
        /*return  the Kp param of PID*/
        fKp = pid_getKp();
        ucValue = floatToUChar(fKp);
        for (iI = 0; iI < 4; iI++)
            debug_putchar(ucValue[iI]);
        break;
```

```
    case 's':
        /*return  target temperature*/
        fTempReq = pid_getSetValue();
        ucValue = floatToUChar(fTempReq);
        for (iI = 0; iI < 4; iI++)
            debug_putchar(ucValue[iI]);
        break;

    case 'b':
        if (bLock)
            debug_putchar('1');
        else
            debug_putchar('0');
        break;
    };

    free(ucValue);
    debug_putchar(PONTO_VIRGULA);
}

/* ********************************************** */
/* Method name:        setScreen                  */
/* Method description: print on lcd the current   */
/*                     state  of the equipament   */
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void setScreen()
{
    static int iSendTemp = 0;
    if (iSendTemp == 10)
    {
        iSendTemp = 0;
        answerParam(ucTempNow);/*send temperature information one time
per second*/
    }
    else
    {
        iSendTemp++;
    }
```

```
    enum state esState = getState();/*to now what to print, look the
colors on state machine*/
    static float fTempNow = 0;
    static float fTempReq = 0;
    static float fKd = 0;
    static float fKi = 0;
    static float fKp = 0;
    static float fHeater = 0;
    static float fCooler = 0;

    static unsigned char ucAux[16];
    if (bLock)/*print massage relate to look buttons*/
        lcd_writeText(1, "TEC. DESBLOQ");
    else
        lcd_writeText(1, "TEC. BLOQUEADO");
    switch (esState)
        {
        case TARGETTEMP:
            ucAux[0] = 'S';
            ucAux[1] = 'E';
            ucAux[2] = 'T';
            ucAux[3] = ' ';
            ucAux[4] = 'T';
            ucAux[5] = 'E';
            ucAux[6] = 'M';
            ucAux[7] = 'P';
            ucAux[8] = ':';

            fTempReq = pid_getSetValue();
            ucAux[10] = (((int)fTempReq) % 10) + 48;/*add 48 because of
asci table, where 0 is 48*/
            fTempReq = ((int)fTempReq) / 10; /*get the next number*/
            ucAux[9] = (((int)fTempReq) % 10) + 48;

            ucAux[11] = '^';/*testei no simulador online e o ° não
aparecia :( */
            ucAux[12] = 'C';
            ucAux[13] = '\0';
```

```
            lcd_writeText(0, ucAux);
            break;
        case TARGETKD:
            ucAux[0] = 'K';
            ucAux[1] = 'D';
            ucAux[2] = ':';
            ucAux[3] = ' ';

            fKd = pid_getKd();

            /*ex 123.23*/
            fKd = fKd * 1000;/*mesma logica do anterior*/
            ucAux[9] = (((int)fKd) % 10) + 48;

            fKd = ((int)fKd) / 10;
            ucAux[8] = (((int)fKd) % 10) + 48;

            ucAux[7] = '.';

            fKd = ((int)fKd) / 10;
            ucAux[6] = (((int)fKd) % 10) + 48;

            fKd = ((int)fKd) / 10;
            ucAux[5] = (((int)fKd) % 10) + 48;

            fKd = ((int)fKd) / 10;
            ucAux[4] = (((int)fKd) % 10) + 48;

            ucAux[10] = '\0';
            lcd_writeText(0, ucAux);
            break;
        case TARGETKI:/*same logic */
            ucAux[0] = 'K';
            ucAux[1] = 'I';
            ucAux[2] = ':';
            ucAux[3] = ' ';

            fKi = pid_getKi();

            fKi = fKi * 1000;
```

```
            ucAux[9] = (((int)fKi) % 10) + 48;

            fKi = ((int)fKi) / 10;
            ucAux[8] = (((int)fKi) % 10) + 48;

            ucAux[7] = '.';

            fKi = ((int)fKi) / 10;
            ucAux[6] = (((int)fKi) % 10) + 48;

            fKi = ((int)fKi) / 10;
            ucAux[5] = (((int)fKi) % 10) + 48;

            fKi = ((int)fKi) / 10;
            ucAux[4] = (((int)fKi) % 10) + 48;

            ucAux[10] = '\0';
            lcd_writeText(0, ucAux);
            break;

        case TARGETKP:
            ucAux[0] = 'K';
            ucAux[1] = 'P';
            ucAux[2] = ':';
            ucAux[3] = ' ';

            fKp = pid_getKp();
            fKp = fKp * 100;

            ucAux[8] = (((int)fKp) % 10) + 48;
            fKp = ((int)fKp) / 10;
            ucAux[7] = (((int)fKp) % 10) + 48;
            ucAux[6] = '.';
            ucAux[5] = (((int)fKp) % 10) + 48;
            fKp = ((int)fKp) / 10;
            ucAux[4] = (((int)fKp) % 10) + 48;

            ucAux[9] = '\0';
            lcd_writeText(0, ucAux);
            break;
```

```
        case DUTYHEATER:
            ucAux[0] = 'R';
            ucAux[1] = 'E';
            ucAux[2] = 'S';
            ucAux[3] = 'I';
            ucAux[4] = 'S';
            ucAux[5] = 'T';
            ucAux[6] = 'O';
            ucAux[7] = 'R';
            ucAux[8] = ':';
            ucAux[9] = ' ';

            fHeater = getHeaterDuty();
            fHeater = fHeater * 100;

            ucAux[12] = (((int)fHeater) % 10) + 48;
            fHeater = ((int)fHeater) / 10;
            ucAux[11] = (((int)fHeater) % 10) + 48;
            fHeater = ((int)fHeater) / 10;
            ucAux[10] = (((int)fHeater) % 10) + 48;

            ucAux[13] = ' ';
            ucAux[14] = '%';
            ucAux[15] = '\0';
            lcd_writeText(0, ucAux);
            break;

        case DUTYCOOLER:
            ucAux[0] = 'F';
            ucAux[1] = 'A';
            ucAux[2] = 'N';
            ucAux[3] = ' ';
            ucAux[4] = 'E';
            ucAux[5] = 'M';
            ucAux[6] = ' ';

            fCooler = getCoolerDuty();
            fCooler = fCooler * 100;

            ucAux[9] = (((int)fCooler) % 10) + 48;
```

```
            fCooler = ((int)fCooler) / 10;
            ucAux[8] = (((int)fCooler) % 10) + 48;
            fCooler = ((int)fCooler) / 10;
            ucAux[7] = (((int)fCooler) % 10) + 48;

            ucAux[10] = ' ';
            ucAux[11] = '%';
            ucAux[12] = '\0';
            lcd_writeText(0, ucAux);
            break;

        default: /*printa a tempertura em todos estado exto pelos
descritos acima */
            ucAux[0] = 'T';
            ucAux[1] = 'E';
            ucAux[2] = 'M';
            ucAux[3] = 'P';
            ucAux[4] = ' ';
            ucAux[5] = 'A';
            ucAux[6] = 'T';
            ucAux[7] = 'U';
            ucAux[8] = 'A';
            ucAux[9] = 'L';
            ucAux[10] = ':';

            /*print Temperature*/
            fTempNow = getTemp();
            ucAux[12] = (((int)fTempNow) % 10) + 48;
            fTempNow = fTempNow / 10;
            ucAux[11] = (((int)fTempNow) % 10) + 48;

            ucAux[13] = '^';
            ucAux[14] = 'C';
            ucAux[15] = '\0';

            lcd_writeText(0, ucAux);
            break;
        }
}
```