Universidade Estadual de Campinas

ES670 - Projeto de Sistemas Embarcados

Relatório Laboratório 7/7

Nome	*RA*
Matheus G. A. Sasso	158257
Pedro Jairo N. P. Neto	156992
Breno V. de Cerqueira	154817

August 6, 2019

Contents

1 Tarefas : Controle da temperatura do diodo com o cooler

1.1 Objetivos

O objetivo principal deste laboratório é implementar um controle da temperatura do diodo usando um controlador PID. Esse controlador atua no diodo por meio do duty cycle do cooler.

Assim foram implementados dois novos comandos, um para enviar um "set point" de temperatura (temperatura de referência), que também mostra a temperatura atual e um outro comando que envia as contantes K_p, K_i e K_d. O primeiro comando é do formato TEMPXX, onde XX é uma temperatura de dois dígitos e servirá de alvo para o controle do PID. Já o segundo comando tem o formato PXXIYYDZZ, onde XX, YY e ZZ são valores de dois dígitos usados nas variáveis K_p, K_i e K_d respectivamente.

Após alguns testes nós vimos que os valores $K_p = 10$, $K_i = 1$ e $K_d = 10$ são bons valores para o funciomanto deste controlador. Quando usamos o $K_p = 10$ e os outros 0, temos um controlador que liga o cooler quando a temperatura está acima da desejada e desliga caso contrário. Quando usamos $K_i = 1$ e os outros 0, temos um controlador que deixa o cooler ligado mesmo depois de atingir a temperatura desejada tanto para valores maiores quanto menores por causa do erro aculmulado e assim ele fica oscilando. Quando usamos $K_d = 10$ e os outros 0, temos um controle que liga o cooler rapidamente a cada vez que ocorre uma mudança de um grau.

Assim pudemos ver o funcionamento do controlador PID e de cada parte desse controlador separadamente e entender seu funcionamento.

1.2 Modelagem

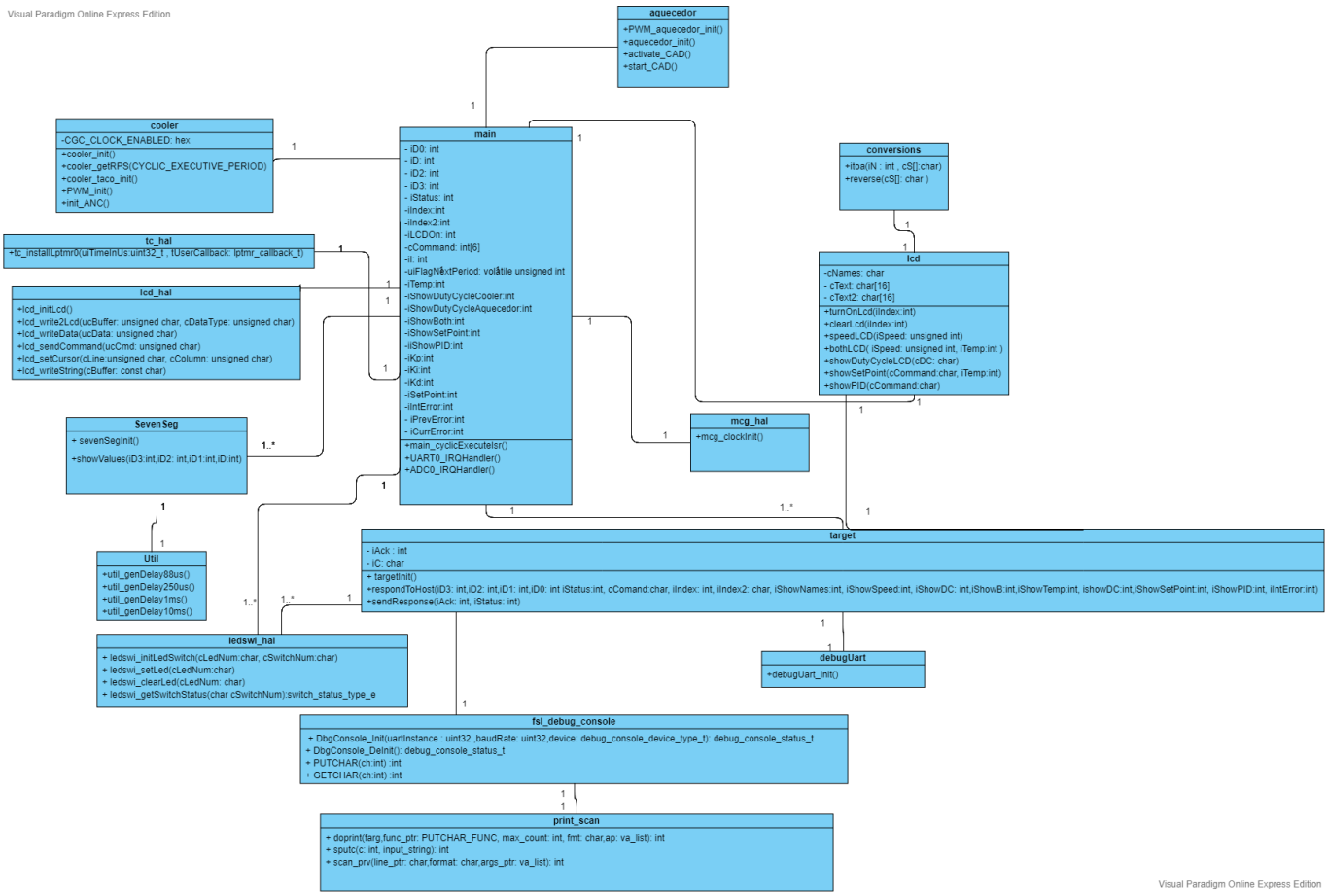

Figure 1: Diagrama de Classe Tarefa 1.

1.3 Matriz de rastreabilidade de “requisitos” X “implementações”

Table 1: Matriz de rastreabilidade Tarefa 1.

Requisito	**Implementação**
Leitura da velocidade de rotação explicado na seção 1.1	lcd_hal.c - void lcd_initLcd(void) lcd.c - void turnOnLcd(int *iIndex) - void clearLCD(int *iIndex) - void speedLCD(unsigned int iSpeed) - void bothLCD(unsigned int iSpeed, int iTemp) - void showDutyCycleLCD(char *cCommand) - void showSetPoint(char *cCommand, int iTemp) - void showPID(char *cCommand) conversions.c - void itoa(int n, char s[]) - void reverse(char s[]) cooler.c - void cooler_taco_init(void); - unsigned int cooler_getRPS(unsigned int uiPeriod) - void cooler_init(void); - void PWM_init(void); - init_ANC(void); target.c - void respondToHost(int *iD3, int *iD2, int *iD1, int *iD0, int *iStatus, char *cCommand, int *iIndex, int *iIndex2, int *iShowNames, int *iShowSpeed, int *iShowBoth, int *iShowDutyCycleAquecedor, int *iShowDutyCycleCooler, int *iShowSetPoint, int *iShowPID, int *iIntError) aquecedor.c - void aquecedor_init(void) - void PWM_aquecedor_init(void) - void activate_CAD(void) - void start_CAD(void)

1.4 Notas

- Dificuldades:

Uma das dificuldades neste laboratório foi visualizar como implementar o controlador PID nesse tipo de sistema. Isso acontece porque para usá-lo é preciso retornar à definição desse tipo de controlador e implementá-lo de forma, digamos ”rústica”, ou seja, uma integral pela soma dos erros anteriores e uma derivada como a variação entre dois erros dividida pelo tempo. A priori ficou difícil entender que a implementação poderia ser dessa forma.

Outra parte complicada é encontrar os valores bons dos ”K”, pois é difícil saber quando esse valor é bom o suficiente (os critérios de bom não estão bem definidos no roteiro).

- Sugestões:

Seria interessante acrescentar a ideia de implementação do controlador no roteiro, como por exemplo explicar a implementação do controle proporcional, integral ou derivativo para que os alunos possam deduzir os demais.

Além disso, seria interessante ter algum critério para saber se os valores de ”K” ja estão ajustados bem o suficiente.

A Códigos

A.1 main.c

```
/*
 * Copyright (c) 2015, 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 "fsl_device_registers.h"
#include "es670_peripheral_board.h"
#include <MKL25Z4.h>
#include "util.h"
#include "mcg_hal.h"
#include "SevenSeg.h"
#include "target.h"
#include "ledswi_hal.h"
#include "debugUart.h"
#include "print_scan.h"
#include "fsl_debug_console.h"
#include "lcd.h"
#include "lcd_hal.h"
#include "tc_hal.h"
#include "cooler.h"
#include "aquecedor.h"
#include "conversions.h"

#define CYCLIC_EXECUTIVE_PERIOD 1000 * 1000 //micro seconds

char cCommand[9];
/* Global Variables */
int iIndex = 0 , iIndex2 = 0, iShowNames = 0, iShowSpeed = 0, iShowBoth = 0, iShowDutyCycleCooler
    = 0,iShowDutyCycleAquecedor = 0, iTemp = 25, iShowSetPoint = 0, iShowPID = 0;;
volatile unsigned int uiFlagNextPeriod = 0; /*cyclic executive flag*/
int iKp = 0, iKi = 0, iKd = 0, iSetPoint = 0, iIntError = 0, iPrevError = 0, iCurrError = 0;

/* Handler for temperature acquisition */
void ADC0_IRQHandler(void){
  unsigned char tabela_temp[256] = {
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //15
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, //31
  1, 1, 2, 2, 3, 3, 3, 3, 4, 4, 5, 5, 6, 6, 6, 6, //47
  7, 7, 8, 8, 8, 8, 9, 9, 10, 10, 10, 10, 11, 11, 12, 12, //63
  12, 12, 13, 13, 14, 14, 15, 15, 15, 15, 16, 16, 16, 17, 17, 17, //79
  17, 18, 18, 19, 19, 19, 19, 20, 20, 21, 21, 21, 21, 22, 22, 23, //95
```

```
  23, 24, 24, 24, 24, 25, 25, 26, 26, 26, 26, 27, 27, 28, 28, 28, //111
  28, 29, 29, 30, 30, 30, 30, 31, 31, 32, 32, 32, 32, 33, 33, 34, //127
  34, 35, 35, 35, 35, 36, 36, 37, 37, 37, 37, 38, 38, 39, 39, 39, //143
  39, 40, 40, 41, 41, 41, 41, 42, 42, 43, 43, 44, 44, 44, 44, 45, //159
  45, 46, 46, 46, 46, 47, 47, 48, 48, 48, 48, 49, 49, 50, 50, 50, //175
  50, 51, 51, 52, 52, 53, 53, 53, 53, 54, 54, 55, 55, 55, 55, 56, //191
  56, 57, 57, 57, 57, 58, 58, 59, 59, 59, 59, 60, 60, 61, 61, 62, //207
  62, 62, 62, 63, 63, 64, 64, 64, 64, 65, 65, 66, 66, 66, 66, 67, //223
  67, 68, 68, 68, 68, 69, 69, 70, 70, 71, 71, 71, 71, 72, 72, 72, //239
  73, 73, 73, 73, 74, 74, 75, 75, 75, 75, 76, 76, 77, 77, 77, 77 //255
  };

  int iAux = ADC0_RA;
  iTemp = tabela_temp[iAux];
}



/* ********************************************** */
/* Method name: main_cyclicExecuteIsr         */
/* Method description: cyclic executive interrupt   */
/* service routine                    */
/* Input params: n/a                */
/* Output params: n/a                 */
/* ********************************************** */
void main_cyclicExecuteIsr(void){
  /* set the cyclic executive flag */
  uiFlagNextPeriod = 1;
}

/* ********************************************** */
/* Method name: UART0_IRQHandler          */
/* Method description: UART0 interrupt routine    */
/* Input params: n/a                */
/* Output params: n/a                 */
/* ********************************************** */
void UART0_IRQHandler(void){
  NVIC_DisableIRQ(UART0_IRQn);
  cCommand[iIndex] = GETCHAR();
  iIndex++;
  NVIC_EnableIRQ(UART0_IRQn);
}

int main(void)
{
  /* Call functions that initialize the clock, target, 7 segments displays, LED 4, LCD, coooler
    and the push button 3. */
  mcg_clockInit();
  cooler_init();
  PWM_init();
  aquecedor_init();
  PWM_aquecedor_init();
  targetInit();
  sevenSegInit();
  ledswi_initLedSwitch(1u, 3u);
  lcd_initLcd();
  activate_CAD();

  /* configure cyclic executive interruption */
  tc_installLptmr0(CYCLIC_EXECUTIVE_PERIOD, main_cyclicExecuteIsr);
  /* Variables that keep the values shown in the 7 segments displays and the push button 3 status.
     */
  int iD3 = 10, iD2 = 10, iD1 = 10, iD0 = 10, iStatus = 2;

  /* configure UART0 interrupts */
  NVIC_ClearPendingIRQ(UART0_IRQn);
  NVIC_EnableIRQ(UART0_IRQn);
  /* receive interrupt enable */
  UART0_C2_REG(UART0) |= UART0_C2_RIE(1);

  /* Initializes the interruption ADC0*/
```

```
135  NVIC_EnableIRQ(ADC0_IRQn);
136
137  int iI = 0;
138    /* This for loop should be replaced. By default this loop allows a single stepping. */
139    for (;;) {
140
141      /* Checks if the cooler speed must be displayed on the LCD */
142      if (iShowSpeed){
143        speedLCD(cooler_getRPS(CYCLIC_EXECUTIVE_PERIOD));
144      }
145      /* Checks if one of the Duty Cycles must be displayed on the LCD  */
146      else if(iShowDutyCycleCooler || iShowDutyCycleAquecedor){
147      showDutyCycleLCD(cCommand);
148      }
149      /* Checks if  both the cooler speed and the temperature must be displayed on the LCD */
150      else if(iShowBoth){
151        start_CAD();
152        bothLCD(cooler_getRPS(CYCLIC_EXECUTIVE_PERIOD), iTemp);
153      }
154      else if(iShowSetPoint){
155
156        TPM1_C0V = TPM_CnV_VAL(255*10/100);
157        iPrevError = (iSetPoint - iTemp);/*gets the error of the previuous ireaction*/
158        start_CAD();
159        char cSetPoint[2] = {cCommand[4], cCommand[5]};/*Gets the SetPoint based on the command*/
160        iSetPoint = atoi(cSetPoint);
161        showSetPoint(cCommand, iTemp);
162
163        iCurrError = (iSetPoint - iTemp);/*Current Error*/
164        iIntError += (iSetPoint - iTemp);/*Integration Error*/
165
166      /*Gets the exit value based on all the portions of the PID control*/
167        int iProp = iKp*iCurrError;
168        int iInt = iKi*iIntError;
169        int iDeriv = iKd*(iCurrError - iPrevError);
170
171      /*This is a simmulation of the control loop*/
172        if ((iProp + iInt + iDeriv) < 0){
173          TPM1_C1V = TPM_CnV_VAL(iProp + iInt + iDeriv);
174        } else {
175          TPM1_C1V = TPM_CnV_VAL(0);
176        }
177      }
178    /*Used to get the values of the PID gains*/
179      else if(iShowPID){
180        char cKp[2] = {cCommand[1], cCommand[2]};
181        char cKi[2] = {cCommand[4], cCommand[5]};
182        char cKd[2] = {cCommand[7], cCommand[8]};
183        iKp = atoi(cKp);
184        iKi = atoi(cKi);
185        iKd = atoi(cKd);
186        showPID(cCommand);
187      }
188
189      /* While the interruption is not reached, uiFlagNextPeriod==0 */
190      /* Once the interruption is reached, uiFlagNextPeriod==1 and the loop ends */
191      while(!uiFlagNextPeriod){
192        /* Communication between o Host e o Target.*/
193        respondToHost(&iD3, &iD2, &iD1, &iD0, &iStatus, cCommand, &iIndex, &iIndex2, &iShowNames,
    &iShowSpeed, &iShowBoth , &iShowDutyCycleAquecedor,&iShowDutyCycleCooler, &iShowSetPoint, &
    iShowPID, &iIntError);
194        showValues (iD3, iD2, iD1, iD0);
195          /* Checks if the group members names must be displayed on the LCD */
196            if (iShowNames && (iI%5 == 0)){
197              turnOnLcd(&iIndex2);
198              iI = 0;
199            }
200
201        iI++;
202      }
203
```

```
204         /* Resets uiFlagNextPeriod */
205         uiFlagNextPeriod = 0;
206       }
207 
208       /* Never leave main */
209       return 0;
210 }
211 ////////////////////////////////////////////////////////////////////////////
212 // EOF
213 ////////////////////////////////////////////////////////////////////////////
```

A.2 aquecedor.c

```
/* ************************************************************ */
/* File name:        aquecedor.c                                */
/* File description: This file has a couple of useful functions to   */
/*                   use the heater                             */
/*                                                              */
/*                   Remarks:                                */
/*                                                              */
/*                                                              */
/* Author name:      Breno Vicente de Cerqueira                 */
/*                   Pedro Jairo Nogueira Pinheiro Neto         */
/*                   Matheus Gustavo Alves Sasso                */
/* Creation date:    11may2019                                  */
/* Revision date:    05jun2019                                  */
/* ************************************************************ */

#include "tc_hal.h"
#include <MKL25Z4.h>

#define CGC_CLOCK_ENABLED 0x01U


/* ***************************************************** */
/* Method name: aquecedor_init                           */
/* Method description: Initialize the heater             */
/* Input params: n/a                                     */
/* Output params: n/a                                    */
/* ***************************************************** */
void aquecedor_init(void){
  /* turn on fan */
  /* un-gate port clock */
  SIM_SCGC5 |= SIM_SCGC5_PORTA(0x01);
  /* set pin as gpio */
  PORTA_PCR12 |= PORT_PCR_MUX(0x01);
  /* set pin as digital output */
  GPIOA_PDDR |= GPIO_PDDR_PDD(0b01 << 13);
  /* set desired pin */
  GPIOA_PSOR = GPIO_PSOR_PTSO(0b01 << 13);
}


/* ***************************************************** */
/* Method name: PWM_aquecedor_init                       */
/* Method description: Initialize the PWM pulse for the     */
/* heater.                          */
/* Input params: n/a                                     */
/* Output params: n/a                                    */
/* ***************************************************** */
void PWM_aquecedor_init(void){
  /* turn on fan */
  SIM_SCGC6 |= SIM_SCGC6_TPM1(CGC_CLOCK_ENABLED); // Un-gate TPM1 clock
  SIM_SCGC5 |= SIM_SCGC5_PORTA(CGC_CLOCK_ENABLED); // Un-gate PORTA clock
  SIM_SOPT2 |= SIM_SOPT2_TPMSRC(0b10); // Select TPM Source OSCERCLK clock
  PORTA_PCR12 |= PORT_PCR_MUX(0b011); // Configure PTA13 as TPM1_CH1
  TPM1_SC &= ~TPM_SC_CPWMS(1); // Increase counting
  TPM1_SC |= TPM_SC_CMOD(0b01) | TPM_SC_PS(0b000); // LPTPM increments on every clock & preescaler
     = 1
  TPM1_CNT = 0; // Reset counter
  TPM1_MOD = TPM_MOD_MOD(0xFF); // Cycle period

  TPM1_C0SC |= TPM_CnSC_MSB(1) | TPM_CnSC_ELSB(1); // Edge-aligned PWM
  TPM1_C0SC &= ~(TPM_CnSC_MSA(1) | TPM_CnSC_ELSA(1)); // Edge-aligned PWM
  TPM1_C0V = TPM_CnV_VAL(0x00); // Duty cycle 0%
}

/* ***************************************************** */
/* Method name: activate_CAD                             */
/* Method description: Set registers for Analog to digital  */
/* convertion                          */
/* Input params: n/a                                     */
```

```
/* Output params: n/a                                          */
/* ********************************************************** */
void activate_CAD(void){
  SIM_SCGC6 |= SIM_SCGC6_ADC0(CGC_CLOCK_ENABLED); // Un-gate CAD conversor
  SIM_SCGC5 |= SIM_SCGC5_PORTE(CGC_CLOCK_ENABLED); // Un-gate PORTE clock
  PORTE_PCR21 &= ~PORT_PCR_MUX(0b111); // Configure pin 21 as analog input.
  ADC0_CFG1 &= ~ADC_CFG1_MODE(0b11);
  ADC0_SC2 &= ~ADC_SC2_ADTRG(0b1);
  ADC0_SC3 |= ADC_SC3_AVGE(0b1);
  ADC0_SC3 &= ~ADC_SC3_AVGS(0b11);
  ADC0_SC3 &= ~ADC_SC3_ADCO(0b1);
}

/* ********************************************************** */
/* Method name: start_CAD                                     */
/* Method description: Start one analog digital conversion    */
/* Input params: n/a                                          */
/* Output params: n/a                                         */
/* ********************************************************** */
void start_CAD(void){
  ADC0_SC1A &= ~ADC_SC1_DIFF(0b1);
  ADC0_SC1A &= ~ADC_SC1_ADCH(0b11011);
  ADC0_SC1A |= ADC_SC1_AIEN(0b1);
}
```

A.3 cooler.c

```
/* ************************************************************ */
/* File name:        cooler.c                                   */
/* File description: This file has a couple of useful functions to   */
/*                   use the cooler                             */
/*                                                              */
/*                   Remarks:                                */
/*                                                              */
/*                                                              */
/* Author name:      Breno Vicente de Cerqueira                 */
/*                   Pedro Jairo Nogueira Pinheiro Neto         */
/*                   Matheus Gustavo Alves Sasso                */
/* Creation date:    11may2019                                  */
/* Revision date:    19may2019                                  */
/* ************************************************************ */

#include "tc_hal.h"

#define CGC_CLOCK_ENABLED 0x01U

/* ************************************************* */
/* Method name: cooler_taco_init                     */
/* Method description: Initialize the cooler tachometer     */
/* Input params: n/a                                 */
/* Output params: n/a                                */
/* ************************************************* */
void cooler_taco_init(void){

  SIM_SCGC6 |= SIM_SCGC6_TPM0(CGC_CLOCK_ENABLED); // Un-gate TPM0 clock
  SIM_SCGC5 |= SIM_SCGC5_PORTE(CGC_CLOCK_ENABLED); // Un-gate PORTE clock
  SIM_SOPT2 |= SIM_SOPT2_TPMSRC(0b10); // Select TPM Source OSCERCLK

  // clock
  SIM_SOPT4 &= ~SIM_SOPT4_TPM0CLKSEL(1); // Select TPM0 external clock as

  // TPM_CLKIN0
  PORTE_PCR29 |= PORT_PCR_MUX(0b100); // Configure PTE29 as TPM_CLKIN0
  TPM0_SC &= ~TPM_SC_CPWMS(1); // Increase counting

  /* LPTPM counter increments on rising edge & preescaler = 1 */
  TPM0_SC |= TPM_SC_CMOD(0b10) | TPM_SC_PS(0b000);
  TPM0_CNT = 0; // Reset counter
}

/* ************************************************* */
/* Method name: cooler_getRPS                        */
/* Method description: Get the cooler speed in RPS          */
/* Input params: uiPeriod in microseconds                   */
/* Output params: uiCount                            */
/* ************************************************* */
unsigned int cooler_getRPS(unsigned int uiPeriod){
  unsigned int uiCount = 0;

  uiCount = TPM0_CNT; // get counter value

  TPM0_CNT = 0; // restart TPM0 counter

  uiCount = uiCount*1000000 / (uiPeriod * 7); // compute speed

  return uiCount;
}

/* ************************************************* */
/* Method name: cooler_init                          */
/* Method description: Initialize the cooler                */
/* Input params: n/a                                 */
/* Output params: n/a                                */
/* ************************************************* */
void cooler_init(void){

```

```
  cooler_taco_init();

  /* turn on fan */
  /* un-gate port clock */
  SIM_SCGC5 |= SIM_SCGC5_PORTA(0x01);
  /* set pin as gpio */
  PORTA_PCR13 |= PORT_PCR_MUX(0x01);
  /* set pin as digital output */
  GPIOA_PDDR |= GPIO_PDDR_PDD(0b01 << 13);
  /* set desired pin */
  GPIOA_PSOR = GPIO_PSOR_PTSO(0b01 << 13);
}


/* ************************************************** */
/* Method name: PWM_init                              */
/* Method description: Initialize the PWM pulse.      */
/* Input params: n/a                                  */
/* Output params: n/a                                 */
/* ************************************************** */
void PWM_init(void){
  /* turn on fan */
  SIM_SCGC6 |= SIM_SCGC6_TPM1(CGC_CLOCK_ENABLED); // Un-gate TPM1 clock
  SIM_SCGC5 |= SIM_SCGC5_PORTA(CGC_CLOCK_ENABLED); // Un-gate PORTA clock
  SIM_SOPT2 |= SIM_SOPT2_TPMSRC(0b10); // Select TPM Source OSCERCLK clock
  PORTA_PCR13 |= PORT_PCR_MUX(0b011); // Configure PTA13 as TPM1_CH1
  TPM1_SC &= ~TPM_SC_CPWMS(1); // Increase counting
  TPM1_SC |= TPM_SC_CMOD(0b01) | TPM_SC_PS(0b000); // LPTPM increments on every clock & preescaler
     = 1
  TPM1_CNT = 0; // Reset counter
  TPM1_MOD = TPM_MOD_MOD(0xFF); // Cycle period
  TPM1_C1SC |= TPM_CnSC_MSB(1) | TPM_CnSC_ELSB(1); // Edge-aligned PWM
  TPM1_C1SC &= ~(TPM_CnSC_MSA(1) | TPM_CnSC_ELSA(1)); // Edge-aligned PWM
  TPM1_C1V = TPM_CnV_VAL(0x00); // Duty cycle 0%
}

void init_ANC(void){
  SIM_SCGC5 |= SIM_SCGC5_PORTE(CGC_CLOCK_ENABLED); // Un-gate PORTE clock
  SIM_SCGC6 |= SIM_SCGC6_ADC0(CGC_CLOCK_ENABLED); // Un-gate ANC clock
  PORTE_PCR21 &= ~PORT_PCR_MUX(0b111);
}
```

A.4 lcd.c

```
/* ************************************************************ */
/* File name:        lcd.c                                      */
/* File description: This file has some useful functions to     */
/*                   write the initial information on the ldc   */
/*                   and to clear it                            */
/*                                                              */
/*                   Remarks:                                   */
/*                                                              */
/*                                                              */
/* Author name:      Breno Vicente de Cerqueira                 */
/*                   Pedro Jairo Nogueira Pinheiro Neto         */
/*                   Matheus Gustavo Alves Sasso                */
/* Creation date:    09may2019                                  */
/* Revision date:    01jul2019                                  */
/* ************************************************************ */

#include <MKL25Z4.h>
#include "lcd_hal.h"
#include "lcd.h"
#include "conversions.h"

char *cNames = "Breno Matheus Pedro               ";
char cText[16];
char cText2[16] = "Breno Matheus Pe";

/* ************************************************/
/* Method name: turnOnLcd                         */
/* Method description: This method writes the     */
/* initial screen on the Lcd board.               */
/* Input params: iIndex used to set correctly the */
/* cursor position                                */
/* Output params: n/a                             */
/* ************************************************/
void turnOnLcd(int *iIndex){

  // Writes ES670 on the first line
  lcd_setCursor(0,0);
  lcd_writeString("ES670");

  // Writes students names on the right position
  // This position changes each time that turnOnLcd function is called
  if (*iIndex <= 15){
    lcd_setCursor(1,(15-*iIndex));
    lcd_writeString(cText2);
  }

  if (*iIndex >= 16){
    for(int iJ = 0; iJ < 16; iJ++){
      cText[iJ] = cNames[*iIndex + iJ - 15];
    }
    lcd_setCursor(1,0);
    lcd_writeString(cText);
  }

  *iIndex = *iIndex + 1;

  // When it finishes the loop the index is restarted
  if(*iIndex == 35){
    *iIndex=0;
  }
}

/* ************************************************/
/* Method name: speedLCD                          */
/* Method description: This method writes the     */
/* cooler velocity.                               */
/* Input params: iSpeed is the cooler speed in RPS*/
/* Output params: n/a                             */
/* ************************************************/
```

```
70 void speedLCD(unsigned int iSpeed){
71   lcd_setCursor(1,0);
72     lcd_writeString("                ");
73 
74   // Writes ES670 on the first line
75   lcd_setCursor(0,0);
76   lcd_writeString("Cooler Speed:");
77 
78   char cSpeed[20];
79   itoa(iSpeed,cSpeed);
80   lcd_setCursor(1,0);
81 
82   lcd_writeString(cSpeed);
83 }
84 
85 /* ************************************************/
86 /* Method name: bothLCD                           */
87 /* Method description: This method writes the     */
88 /* cooler velocity and the heater temperature     */
89 /* Input params: iSpeed is the cooler speed in RPS*/
90 /* Input params: iTemp is the heater temperature  */
91 /* Output params: n/a                             */
92 /* ************************************************/
93 void bothLCD(unsigned int iSpeed, int iTemp){
94   lcd_setCursor(1,0);
95     lcd_writeString("                ");
96 
97   // Writes ES670 on the first line
98   lcd_setCursor(0,0);
99   lcd_writeString("Speed: | Temp:");
100 
101   char cTemp[20];
102   itoa(iTemp,cTemp);
103   char cSpeed[20];
104   itoa(iSpeed,cSpeed);
105   lcd_setCursor(1,0);
106   lcd_writeString(cSpeed);
107   lcd_setCursor(1,6);
108   lcd_writeString(" | ");
109   lcd_setCursor(1,9);
110   lcd_writeString(cTemp);
111 }
112 
113 /* ************************************************/
114 /* Method name: showDutyCycleLCD                  */
115 /* Method description: This method writes the     */
116 /* current duty cycle.                            */
117 /* Input params: cCommand given by the user       */
118 /* Output params: n/a                             */
119 /* ************************************************/
120 void showDutyCycleLCD(char *cCommand){
121   lcd_setCursor(1,0);
122     lcd_writeString("                ");
123 
124 
125   lcd_setCursor(0,0);
126   // Writes one of the duty Cycles depending by the command
127   if(cCommand[0] == 'H'){
128     lcd_writeString("Duty Cycle AQ:");
129   }
130   else{
131     lcd_writeString("Duty Cycle CL:");
132   }
133 
134   lcd_setCursor(1,0);
135 
136   char cDC[3] = {cCommand[2], cCommand[3], cCommand[4]};
137   lcd_writeString(cDC);
138 }
139 
140 /* ************************************************/
```

```
/* Method name: clearLcd                                    */
/* Method description: This method clear the Lcd.           */
/*                                                          */
/* Input params: It receives the iIndex to be reseted       */
/* Output params:      n/a                                  */
/* *********************************************************/
void clearLcd(int *iIndex){
  *iIndex = 0;
  lcd_setCursor(0,0);
  lcd_writeString("                ");
  lcd_setCursor(1,0);
    lcd_writeString("                ");
}

/* *********************************************************/
/* Method name: showSetPoint                                */
/* Method description: This method writes the               */
/* setpoint                                          */
/* Input params: cCommand given by the user          */
/* iTemp                                */
/* Output params: n/a                                       */
/* *********************************************************/
void showSetPoint(char *cCommand, int iTemp){
  lcd_setCursor(1,0);
    lcd_writeString("                ");

  // Writes ES670 on the first line
  lcd_setCursor(0,0);
  lcd_writeString("SPoint: | Temp:");

  lcd_setCursor(1,0);

  char cDC[2] = {cCommand[4], cCommand[5]};
  lcd_writeString(cDC);
  char cTemp[20];
  itoa(iTemp,cTemp);
  lcd_setCursor(1,7);
  lcd_writeString(" | ");
  lcd_setCursor(1,10);
  lcd_writeString(cTemp);
}

/* *********************************************************/
/* Method name: showPID                                     */
/* Method description: This method writes on the            */
/* Lcd the values of Kp, Kd and Ki                          */
/* Input params: cCommand given by the user                 */
/* Output params: n/a                                       */
/* *********************************************************/
void showPID(char *cCommand){
  lcd_setCursor(1,0);
    lcd_writeString("                ");

  // Writes ES670 on the first line
  lcd_setCursor(0,0);
  lcd_writeString("Kp: | Ki: | Kd:");

  char cKp[2] = {cCommand[1], cCommand[2]};
  char cKi[2] = {cCommand[4], cCommand[5]};
  char cKd[2] = {cCommand[7], cCommand[8]};
  lcd_setCursor(1,0);
  lcd_writeString(cKp);
  lcd_setCursor(1,3);
  lcd_writeString(" | ");
  lcd_setCursor(1,6);
  lcd_writeString(cKi);
  lcd_setCursor(1,9);
  lcd_writeString(" | ");
  lcd_setCursor(1,12);
  lcd_writeString(cKd);
}
```

A.5 $lcd_hal.c$

```
/* ************************************************************ */
/* File name:        lcd_hal.c                                  */
/* File description: File dedicated to the hardware abstraction layer*/
/*                   related to the LCD HARDWARE based on the KS006U */
/*                   controller                                 */
/* Author name:      dloubach                                   */
/* Creation date:    16out2015                                  */
/* Revision date:    25fev2016                                  */
/* ************************************************************ */

#include "lcd_hal.h"
#include "es670_peripheral_board.h"
#include "util.h"

/* 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-gate port clock*/
    CLOCK_SYS_EnablePortClock(PORTC_IDX);

    /* set pin as gpio */
    PORT_HAL_SetMuxMode(LCD_PORT_BASE_PNT, LCD_RS_PIN, LCD_RS_ALT);
    PORT_HAL_SetMuxMode(LCD_PORT_BASE_PNT, LCD_ENABLE_PIN, LCD_ENABLE_ALT);
    PORT_HAL_SetMuxMode(LCD_PORT_BASE_PNT, LCD_DATA_DB0_PIN, LCD_DATA_ALT);
    PORT_HAL_SetMuxMode(LCD_PORT_BASE_PNT, LCD_DATA_DB1_PIN, LCD_DATA_ALT);
    PORT_HAL_SetMuxMode(LCD_PORT_BASE_PNT, LCD_DATA_DB2_PIN, LCD_DATA_ALT);
    PORT_HAL_SetMuxMode(LCD_PORT_BASE_PNT, LCD_DATA_DB3_PIN, LCD_DATA_ALT);
    PORT_HAL_SetMuxMode(LCD_PORT_BASE_PNT, LCD_DATA_DB4_PIN, LCD_DATA_ALT);
    PORT_HAL_SetMuxMode(LCD_PORT_BASE_PNT, LCD_DATA_DB5_PIN, LCD_DATA_ALT);
    PORT_HAL_SetMuxMode(LCD_PORT_BASE_PNT, LCD_DATA_DB6_PIN, LCD_DATA_ALT);
    PORT_HAL_SetMuxMode(LCD_PORT_BASE_PNT, LCD_DATA_DB7_PIN, LCD_DATA_ALT);

    /* set pin as digital output */
    GPIO_HAL_SetPinDir(LCD_GPIO_BASE_PNT, LCD_RS_PIN, LCD_RS_DIR);
    GPIO_HAL_SetPinDir(LCD_GPIO_BASE_PNT, LCD_ENABLE_PIN, LCD_ENABLE_DIR);
    GPIO_HAL_SetPinDir(LCD_GPIO_BASE_PNT, LCD_DATA_DB0_PIN, LCD_DATA_DIR);
    GPIO_HAL_SetPinDir(LCD_GPIO_BASE_PNT, LCD_DATA_DB1_PIN, LCD_DATA_DIR);
    GPIO_HAL_SetPinDir(LCD_GPIO_BASE_PNT, LCD_DATA_DB2_PIN, LCD_DATA_DIR);
    GPIO_HAL_SetPinDir(LCD_GPIO_BASE_PNT, LCD_DATA_DB3_PIN, LCD_DATA_DIR);
    GPIO_HAL_SetPinDir(LCD_GPIO_BASE_PNT, LCD_DATA_DB4_PIN, LCD_DATA_DIR);
    GPIO_HAL_SetPinDir(LCD_GPIO_BASE_PNT, LCD_DATA_DB5_PIN, LCD_DATA_DIR);
    GPIO_HAL_SetPinDir(LCD_GPIO_BASE_PNT, LCD_DATA_DB6_PIN, LCD_DATA_DIR);
    GPIO_HAL_SetPinDir(LCD_GPIO_BASE_PNT, LCD_DATA_DB7_PIN, LCD_DATA_DIR);

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

    // init LCD
    lcd_sendCommand(CMD_INIT_LCD);
```

```
70
71      // clear LCD
72      lcd_sendCommand(CMD_CLEAR);
73
74      // LCD with no cursor
75      lcd_sendCommand(CMD_NO_CURSOR);
76
77      // cursor shift to right
78      lcd_sendCommand(CMD_CURSOR2R);
79
80  }
81
82
83
84  /* ******************************************** */
85  /* Method name:        lcd_write2Lcd            */
86  /* Method description: Send command or data to LCD  */
87  /* Input params:       ucBuffer => char to be send  */
88  /*                     cDataType=>command LCD_RS_CMD*/
89  /*                     or data LCD_RS_DATA      */
90  /* Output params:      n/a                      */
91  /* ******************************************** */
92  void lcd_write2Lcd(unsigned char ucBuffer,  unsigned char cDataType)
93  {
94      /* writing data or command */
95      if(LCD_RS_CMD == cDataType)
96          /* will send a command */
97          GPIO_HAL_WritePinOutput(LCD_GPIO_BASE_PNT, LCD_RS_PIN, LCD_RS_CMD);
98      else
99          /* will send data */
100         GPIO_HAL_WritePinOutput(LCD_GPIO_BASE_PNT, LCD_RS_PIN, LCD_RS_DATA);
101
102     /* write in the LCD bus */
103     GPIO_HAL_WritePinOutput(LCD_GPIO_BASE_PNT, LCD_DATA_DB0_PIN, ((ucBuffer & (1u << 0u)) >> 0u));
104     GPIO_HAL_WritePinOutput(LCD_GPIO_BASE_PNT, LCD_DATA_DB1_PIN, ((ucBuffer & (1u << 1u)) >> 1u));
105     GPIO_HAL_WritePinOutput(LCD_GPIO_BASE_PNT, LCD_DATA_DB2_PIN, ((ucBuffer & (1u << 2u)) >> 2u));
106     GPIO_HAL_WritePinOutput(LCD_GPIO_BASE_PNT, LCD_DATA_DB3_PIN, ((ucBuffer & (1u << 3u)) >> 3u));
107     GPIO_HAL_WritePinOutput(LCD_GPIO_BASE_PNT, LCD_DATA_DB4_PIN, ((ucBuffer & (1u << 4u)) >> 4u));
108     GPIO_HAL_WritePinOutput(LCD_GPIO_BASE_PNT, LCD_DATA_DB5_PIN, ((ucBuffer & (1u << 5u)) >> 5u));
109     GPIO_HAL_WritePinOutput(LCD_GPIO_BASE_PNT, LCD_DATA_DB6_PIN, ((ucBuffer & (1u << 6u)) >> 6u));
110     GPIO_HAL_WritePinOutput(LCD_GPIO_BASE_PNT, LCD_DATA_DB7_PIN, ((ucBuffer & (1u << 7u)) >> 7u));
111
112     /* enable, delay, disable LCD */
113     /* this generates a pulse in the enable pin */
114     GPIO_HAL_WritePinOutput(LCD_GPIO_BASE_PNT, LCD_ENABLE_PIN, LCD_ENABLED);
115     util_genDelay1ms();
116     GPIO_HAL_WritePinOutput(LCD_GPIO_BASE_PNT, LCD_ENABLE_PIN, LCD_DISABLED);
117     util_genDelay1ms();
118     util_genDelay1ms();
119 }
120
121
122
123 /* ******************************************** */
124 /* Method name:        lcd_writeData            */
125 /* Method description: Write data to be displayed   */
126 /* Input params:       ucData => char to be written */
127 /* Output params:      n/a                      */
128 /* ******************************************** */
129 void lcd_writeData(unsigned char ucData)
130 {
131     /* just a relay to send data */
132     lcd_write2Lcd(ucData, LCD_RS_DATA);
133 }
134
135
136
137 /* ******************************************** */
138 /* Method name:        lcd_sendCommand          */
139 /* Method description: Write command to LCD     */
140 /* 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 ***");

```

```
212     // set the cursor line 1, column 0
213     lcd_setCursor(1,0);
214     lcd_writeString("Prj Sis Embarcad");
215 }
```

A.6 target.c

```
/* ************************************************************ */
/* File name:        target.c                                   */
/* File description: This file has some useful functions to     */
/*                   treat the information received by target   */
/*                   and respond correctly to the Host.         */
/*                                                              */
/*                   Remarks:                                   */
/*                                                              */
/*                                                              */
/* Author name:      Breno Vicente de Cerqueira                 */
/*                   Pedro Jairo Nogueira Pinheiro Neto         */
/*                   Matheus Gustavo Alves Sasso                */
/* Creation date:    05apr2019                                  */
/* Revision date:    05jun2019                                  */
/* ************************************************************ */

#include <MKL25Z4.h>
#include "util.h"
#include "debugUart.h"
#include "print_scan.h"
#include "fsl_debug_console.h"
#include "ledswi_hal.h"
#include "target.h"
#include "lcd.h"
#include <stdlib.h>

/* ******************************************** */
/* Method name:        targetInit                */
/* Method description: This method initialize the   */
/* target.                                      */
/*                                              */
/* Input params:       n/a                      */
/* Output params:      n/a                      */
/* ******************************************** */
void targetInit(void){
  debugUart_init();
}

/* ************************************************************/
/* Method name:        respondToHost                        */
/* Method description: This method treats the infor-        */
/* mation sent to the Target and performs the               */
/* actions related to the LED 4 and  to the push            */
/* button 3, LCD and and 7 segments.                        */
/*                                                          */
/* Input params: Four integers.                             */
/* iD3 = Value to be shown in display 1 (DS1)               */
/* iD2 = Value to be shown in display 2 (DS2)               */
/* iD1 = Value to be shown in display 3 (DS3)               */
/* iD0 = Value to be shown in display 4 (DS4)               */
/* iStatus = Push button 3 status                           */
/* cCommand = command to be checked                         */
/* iIndex = Instant size of the command                     */
/* iIndex2 = Position to be written in the LCD              */
/* iShowNames = Turn on LCD with names                        */
/* iShowSpeed = Turn on the LCD with speed                  */
/* iShowB = Turn on LCD with with speed and temp            */
/* iShowTemp = Turn on LCD with heater duty cycle             */
/* iShowDC = Turn on the LCD with duty cycle              */
/*iShowSetPoint = Turn on LCD with setpoindt                */
/*iShowPID = Turn on LCD with PID gains                     */
/*iIntError = Set the integrative error to zero when temp is set */
/* Output params:      n/a                                  */
/* ************************************************************ */
void respondToHost(int *iD3, int *iD2, int *iD1, int *iD0, int *iStatus, char *cCommand, int *
    iIndex, int *iIndex2, int *iShowNames, int *iShowSpeed,int *iShowB,int *iShowTemp,int *iShowDC
    , int *iShowSetPoint, int *iShowPID, int *iIntError){

  int iAck = 2;
```

```
68  char iC;
69
70  // This sequence of if and else is used to find out which command was written
71  if (cCommand[0] == 'L' && *iIndex>0){
72      if (cCommand[1] =='S' && *iIndex>1){
73        if (cCommand[2] =='4'  && *iIndex>2){ // If it found LS4
74          iAck = 1;
75          ledswi_setLed(4u); // Turn on the led 4
76        }
77        else if(*iIndex>2)
78        {
79          iAck = 0;
80        }
81      }
82      else if (cCommand[1] =='C' && *iIndex>1){
83        if (cCommand[2] =='4'  && *iIndex>2){ // If it found LC4
84          iAck = 1;
85          ledswi_clearLed(4u); // Turn off the led 4
86        }
87        else if(cCommand[2] =='D'  && *iIndex>2)
88        {
89          if (cCommand[3] =='O'  && *iIndex>3)
90          {
91            if (cCommand[4] =='N'  && *iIndex>4) // If it found LCDON
92            {
93              iAck =1;
94              *iShowSpeed = 0; //Indicates that the cooler velocity should no longer be displayed.
95              *iShowDC = 0;
96              *iShowTemp = 0;
97              *iShowB = 0;
98              *iShowSetPoint = 0;
99              *iShowPID = 0;
100             clearLcd(iIndex2);
101             *iShowNames = 1; // Indicates that it should write on the LCD - This will be updated
    on main
102           }
103           else if(cCommand[4] =='F'  && *iIndex>4)
104           {
105             if (cCommand[5] =='F'  && *iIndex>5) // If it found LCDOFF
106             {
107               iAck =1;
108               *iShowSpeed = 0; //Indicates that the cooler velocity should no longer be
    displayed.
109               *iShowDC = 0;
110               *iShowNames = 0; // Indicates that it should clean the lcd
111               *iShowTemp = 0;
112               *iShowB = 0;
113               *iShowSetPoint = 0;
114               *iShowPID = 0;
115               clearLcd(iIndex2);
116             }
117             else if(*iIndex>5)
118             {
119               iAck = 0;
120             }
121           }
122           else if(*iIndex>4)
123           {
124             iAck = 0;
125           }
126         }
127         else if(*iIndex>3)
128         {
129           iAck = 0;
130         }
131       }
132       else if(*iIndex>2)
133       {
134         iAck = 0;
135       }
136     }
```

```
137       else if(*iIndex>1)
138       {
139         iAck = 0;
140       }
141   }
142   else if(cCommand[0] == 'S' && *iIndex>0){
143       if(cCommand[1] == '3' && *iIndex>1){ // If it found S3
144         iAck = 1;
145         *iStatus = ledswi_getSwitchStatus(3u); // Take the status of switch 3
146       }
147       else if(cCommand[1] == 'S' && *iIndex>1){
148         if(cCommand[2] >= '0' &&  cCommand[2] <= '1' && *iIndex>2){
149           if(cCommand[3] == '0' ||  (cCommand[3] >= '0' && cCommand[3] <= '9' && *iIndex>3 &&
    cCommand[2] == '0')){
150             if(cCommand[4] == '0' ||  (cCommand[4] >= '0' && cCommand[4] <= '9' && *iIndex>4 &&
    cCommand[2] == '0')){ //Changes the duty cycle of the cooler.
151               iAck = 1;
152               char cDuty_cycle[3] = {cCommand[2], cCommand[3], cCommand[4]};
153               int iDuty_cycle = atoi(cDuty_cycle);
154               iDuty_cycle = 255*iDuty_cycle/100;
155               TPM1_C1V = TPM_CnV_VAL(iDuty_cycle);
156
157               *iShowNames = 0; // Indicates that it should clean the lcd
158               *iShowSpeed = 0;
159               *iShowTemp = 0;
160               *iShowB = 0;
161               *iShowSetPoint = 0;
162               *iShowPID = 0;
163               clearLcd(iIndex2);
164               *iShowDC = 1; //Indicates that the cooler duty cycle must be displayed.
165             }
166             else if(*iIndex>4)
167             {
168               iAck = 0;
169             }
170           }
171           else if(*iIndex>3)
172           {
173             iAck = 0;
174           }
175         }
176         else if(*iIndex>2)
177         {
178           iAck = 0;
179         }
180       }
181       else if(*iIndex>1)
182       {
183         iAck = 0;
184       }
185   }
186   else if(cCommand[0] == 'H' && *iIndex>0){
187     if(cCommand[1] == 'C' && *iIndex>1){
188       iAck = 1;
189       *iShowNames = 0; // Indicates that it should clean the lcd
190       *iShowSpeed = 0;
191       *iShowDC = 0;
192       *iShowTemp = 0;
193       *iShowSetPoint = 0;
194       *iShowPID = 0;
195       clearLcd(iIndex2);
196       *iShowB = 1; // Indicates that both speed and temperature must be displayed on the LCD.
197     }
198
199     else if(cCommand[1] == 'S' && *iIndex>1){
200         if(cCommand[2] >= '0' &&  cCommand[2] <= '1' && *iIndex>2){
201           if(cCommand[3] == '0' ||  (cCommand[3] >= '0' && cCommand[3] <= '9' && *iIndex>3 &&
    cCommand[2] == '0')){
202             if(cCommand[4] == '0' ||  (cCommand[4] >= '0' && cCommand[4] <= '9' && *iIndex>4 &&
    cCommand[2] == '0')){
203               iAck = 1;
```

```
204             char cDuty_cycle[3] = {cCommand[2], cCommand[3], cCommand[4]};
205             int iDuty_cycle = atoi(cDuty_cycle);
206             iDuty_cycle = 255*iDuty_cycle/100;
207             TPM1_C0V = TPM_CnV_VAL(iDuty_cycle);
208 
209             *iShowNames = 0; // Indicates that it should clean the lcd
210             *iShowSpeed = 0;
211           *iShowDC = 0;
212             *iShowB = 0;
213             *iShowSetPoint = 0;
214             *iShowPID = 0;
215             clearLcd(iIndex2);
216             *iShowTemp = 1; //Indicates that the heater duty cycle must be displayed.
217 
218           }
219           else if(*iIndex>4)
220           {
221             iAck = 0;
222           }
223         }
224         else if(*iIndex>3)
225         {
226           iAck = 0;
227         }
228       }
229       else if(*iIndex>2)
230       {
231         iAck = 0;
232       }
233     }
234     else if(*iIndex>1)
235     {
236       iAck = 0;
237     }
238 }
239 else if(cCommand[0] == 'D' && *iIndex>0){
240     iC = cCommand[1];
241     if((iC == '1' || iC == '2' || iC == '3' || iC == '4') && *iIndex>1){
242       char M = iC;
243       iC = cCommand[2];
244       switch(M){ // If it found D[1-4][0-9], it should write the value on the correct display
245         case '1':
246           if(iC == 'C' &&  *iIndex>2){
247             iAck = 1;
248             *iD3 = 10;
249           }
250           else if((iC == '0' || iC == '1' || iC == '2' || iC == '3' || iC == '4' || iC == '5' ||
    iC == '6' || iC == '7' || iC == '8' || iC == '9')&&  *iIndex>2){
251             iAck = 1;
252             *iD3 = (iC -'0');
253           }
254           else if(*iIndex>2)
255             {
256               iAck = 0;
257             }
258           break;
259 
260         case '2':
261           if(iC == 'C' &&  *iIndex>2){
262             iAck = 1;
263             *iD2 = 10;
264           }
265           else if((iC == '0' || iC == '1' || iC == '2' || iC == '3' || iC == '4' || iC == '5' ||
    iC == '6' || iC == '7' || iC == '8' || iC == '9')&&  *iIndex>2){
266             iAck = 1;
267             *iD2 = (iC -'0');
268           }
269           else if(*iIndex>2)
270             {
271               iAck = 0;
272             }
```

```
273             break;
274 
275           case '3':
276             if(iC == 'C' &&  *iIndex>2){
277               iAck = 1;
278               *iD1 = 10;
279             }
280             else if((iC == '0' || iC == '1' || iC == '2' || iC == '3' || iC == '4' || iC == '5' ||
    iC == '6' || iC == '7' || iC == '8' || iC == '9')&&  *iIndex>2){
281               iAck = 1;
282               *iD1 = (iC -'0');
283             }
284             else if(*iIndex>2)
285               {
286                 iAck = 0;
287               }
288             break;
289 
290           case '4':
291             if(iC == 'C' &&  *iIndex>2){
292               iAck = 1;
293               *iD0 = 10;
294             }
295             else if((iC == '0' || iC == '1' || iC == '2' || iC == '3' || iC == '4' || iC == '5' ||
    iC == '6' || iC == '7' || iC == '8' || iC == '9')&&  *iIndex>2){
296               iAck = 1;
297               *iD0 = (iC -'0');
298             }
299             else if(*iIndex>2)
300               {
301                 iAck = 0;
302               }
303             break;
304         }
305       }
306       else if(*iIndex>1)
307       {
308         iAck = 0;
309       }
310   }
311   else if(cCommand[0] == 'C' && *iIndex>0){
312     if(cCommand[1] == 'S' && *iIndex>1){ // If it found S3
313       iAck = 1;
314       *iShowNames = 0; // Indicates that it should clean the lcd
315       *iShowDC = 0;
316       *iShowB = 0;
317       *iShowTemp = 0;
318       *iShowSetPoint = 0;
319       *iShowPID = 0;
320       clearLcd(iIndex2);
321       *iShowSpeed = 1; //Indicates that the cooler velocity must be displayed.
322     }
323     else if(*iIndex>1)
324     {
325       iAck = 0;
326       }
327   }
328   else if(cCommand[0] == 'T' && *iIndex>0){
329     if(cCommand[1] == 'E' && *iIndex>1){
330       if(cCommand[2] == 'M' && *iIndex>2){
331         if(cCommand[3] == 'P' && *iIndex>3){
332           if(cCommand[4] >= '3' &&  cCommand[4] <= '4' && *iIndex>4){
333             if(cCommand[5] == '0' ||   (cCommand[5] >= '0' && cCommand[5] <= '9' && *iIndex>5 &&
    cCommand[4] == '3')){
334               iAck = 1;
335 
336               *iShowNames = 0; // Indicates that it should clean the lcd
337               *iShowSpeed = 0;
338               *iShowDC = 0;
339               *iShowB = 0;
340               *iShowTemp = 0; //Indicates that the heater duty cycle must be displayed.
```

```
341               *iShowPID = 0;
342               clearLcd(iIndex2);
343               *iShowSetPoint = 1;
344               *iIntError = 0; //Sets the int error to zero when the temparature is set
345 
346             }
347             else if(*iIndex>5)
348             {
349               iAck = 0;
350             }
351           }
352           else if(*iIndex>4)
353           {
354             iAck = 0;
355           }
356         }
357         else if(*iIndex>3)
358         {
359           iAck = 0;
360         }
361       }
362       else if(*iIndex>2)
363       {
364         iAck = 0;
365       }
366     }
367     else if(*iIndex>1)
368     {
369       iAck = 0;
370       }
371   }
372   else if(cCommand[0] == 'P' && *iIndex>0){
373     if(cCommand[1] >= '0' && cCommand[1] <= '9' && *iIndex>1){
374       if(cCommand[2] >= '0' && cCommand[2] <= '9' && *iIndex>2){
375         if(cCommand[3] == 'I' && *iIndex>3){
376           if(cCommand[4] >= '0' && cCommand[4] <= '9' && *iIndex>4){
377             if(cCommand[5] >= '0' && cCommand[5] <= '9' && *iIndex>5){
378               if(cCommand[6] == 'D' && *iIndex>6){
379                 if(cCommand[7] >= '0' && cCommand[7] <= '9' && *iIndex>7){
380                   if(cCommand[8] >= '0' && cCommand[8] <= '9' && *iIndex>8){
381                     iAck = 1;
382 
383                     *iShowNames = 0; // Indicates that it should clean the lcd
384                     *iShowSpeed = 0;
385                     *iShowDC = 0;
386                     *iShowB = 0;
387                     *iShowTemp = 0; //Indicates that the heater duty cycle must be displayed.
388                     *iShowSetPoint = 0;
389                     clearLcd(iIndex2);
390                     *iShowPID = 1;//Shows the message to get the PID gains
391 
392                   }
393                   else if(*iIndex>8)
394                   {
395                     iAck = 0;
396                   }
397                 }
398                 else if(*iIndex>7)
399                 {
400                   iAck = 0;
401                 }
402               }
403               else if(*iIndex>6)
404               {
405                 iAck = 0;
406               }
407             }
408             else if(*iIndex>5)
409             {
410               iAck = 0;
411             }
```

```
412           }
413           else if(*iIndex>4)
414           {
415             iAck = 0;
416           }
417         }
418         else if(*iIndex>3)
419         {
420           iAck = 0;
421         }
422       }
423       else if(*iIndex>2)
424       {
425         iAck = 0;
426       }
427     }
428     else if(*iIndex>1)
429     {
430       iAck = 0;
431     }
432   }
433   else if(*iIndex>0) // If it found an error
434   {
435     iAck = 0; // Show ERR message
436   }
437
438   if (iAck == 0 || iAck == 1){
439     *iIndex = 0;
440     sendResponse(iAck, iStatus);
441   }
442 }
443
444 /* ********************************************** */
445 /* Method name:        sendResponse               */
446 /* Method description: This method sends the appro- */
447 /* priate response to the Host.                   */
448 /*                                                */
449 /* Input params: 2 integers                       */
450 /* iAck = If the command is valid                 */
451 /* iStatus = Status of the push button 3          */
452 /* Output params:      n/a                        */
453 /* ********************************************** */
454 void sendResponse(int iAck, int *iStatus){
455   if (iAck == 1){
456       PUTCHAR('A');
457       PUTCHAR('C');
458       PUTCHAR('K');
459
460       if (*iStatus == 0){
461         PUTCHAR('O');
462         PUTCHAR('F');
463         PUTCHAR('F');
464         *iStatus = 2;
465       }
466       else if (*iStatus == 1){
467         PUTCHAR('O');
468         PUTCHAR('N');
469         *iStatus = 2;
470       }
471     }
472     else{
473       PUTCHAR('E');
474       PUTCHAR('R');
475       PUTCHAR('R');
476     }
477   PUTCHAR(' ');
478 }
```

A.7 conversions.c

```
/* ************************************************************ */
/* File name:        converstions.c                             */
/* File description: This file has some useful functions to     */
/*                   convert an integer into a string           */
/*                                                              */
/*                   Remarks:                                   */
/*                                                              */
/*                                                              */
/* Author name:      Breno Vicente de Cerqueira                 */
/*                   Pedro Jairo Nogueira Pinheiro Neto         */
/*                   Matheus Gustavo Alves Sasso                */
/* Creation date:    16may2019                                  */
/* Revision date:    16may2019                                  */
/* ************************************************************ */

#include <string.h>

/* *********************************************/
/* Method name: reverse                        */
/* Method description: This method  is used to calcu-*/
/* late reverse string from a given initial string.  */
/* Input params: char cS[] - the string that has to  */
/* be reverted                                 */
/*                                             */
/* Output params: n/a                          */
/* *********************************************/

/* reverse:  reverse string s in place */
void reverse(char cS[])
{
    int iI, jJ;
    char cC;

    for (iI = 0, jJ = strlen(cS)-1; iI<jJ; iI++, jJ--) {
      cC = cS[iI];
        cS[iI] = cS[jJ];
        cS[jJ] = cC;
    }
}

/* *********************************************/
/* Method name: itoa                           */
/* Method description: This method  is used to calcu-*/
/* late the convertion from integer to string.       */
/* Input params: int iN - The integer to be converted*/
/* char cS[] - the string where the function should  */
/* save the result                             */
/* Output params: n/a                          */
/* *********************************************/
void itoa(int iN, char cS[])
{
    int iI, iSign;

    if ((iSign = iN) < 0)  /* record sign */
        iN = -iN;          /* make n positive */
    iI = 0;
    do {       /* generate digits in reverse order */
        cS[iI++] = iN % 10 + '0';   /* get next digit */
    } while ((iN /= 10) > 0);     /* delete it */
    if (iSign < 0)
        cS[iI++] = '-';
    cS[iI] = '\0';
    reverse(cS);
}
```