ES670B - Projeto de Sistemas Embarcados
Relatório 05 - Laboratório 6/7
Laura Marchione - RA 156169
Victor Cintra Santos - RA 157461

Objetivos

Neste relatório temos os seguintes requisitos de projeto: atualizar nosso codigo do laboratório anterior para que o sistema seja capaz de gerar um sinal PWM para acionar o aquecedor da placa e um diodo térmico que, com o auxílio de eletrônica analógica, produz um sinal analógico a ser adquirido por uma entrada analógica do KL25Z, que deve ser lida e interpretada.

Modelagem

Na Figura 01 e 02, no apêndice, há o diagrama UML desenvolvido para tal projeto. Nelas, mostramos o funcionamento da *main* do código e o tratamento de interrupções (no caso, recebimento de algum caractere na comunicação *host/target*). Como ilustrado, há o loop principal da *main* que mostra a mensagem padrão ou de velocidade no LCD, de acordo com as variáveis *enable* para tais funções. Ao receber algum caractere na comunicação, a máquina de estado irá avaliar qual é o caractere e a sequência correta, dando um sinal *ACK* e executando o comando quando requisitado; qualquer outra sequência irá dar um *ERR*. A variável *counter* guarda o último estado da máquina de estados e a *letter* a letra atual lida.

Matriz de Rastreabilidade

Requisito	**Implementação**
Acionar o aquecedor com sinal PWN	main.c - turnOnHeat(); - heat_sensor_init(); - defaultACKMessage(); - defaultErrorMessage(); - dataTargetCommand(char letter);

Notas

Nesse laboratório sentimos muita dificuldade em implementar a interrupção para controlarmos o sinal PWN que iria controlar o aquecedor. Passamos mais de duas horas tentando debugar o código pois no momento que adicionamos as linhas de códigos para habilitar ou desabilitar as interrupções do aquecedor, o software travava e tínhamos que reiniciar toda a configuração.

Apêndice

Na Figura 01 e 02 temos a main e máquina de estado implementadas. Abaixo, após as figuras do apêndice, há o código do desenvolvimento do projeto.

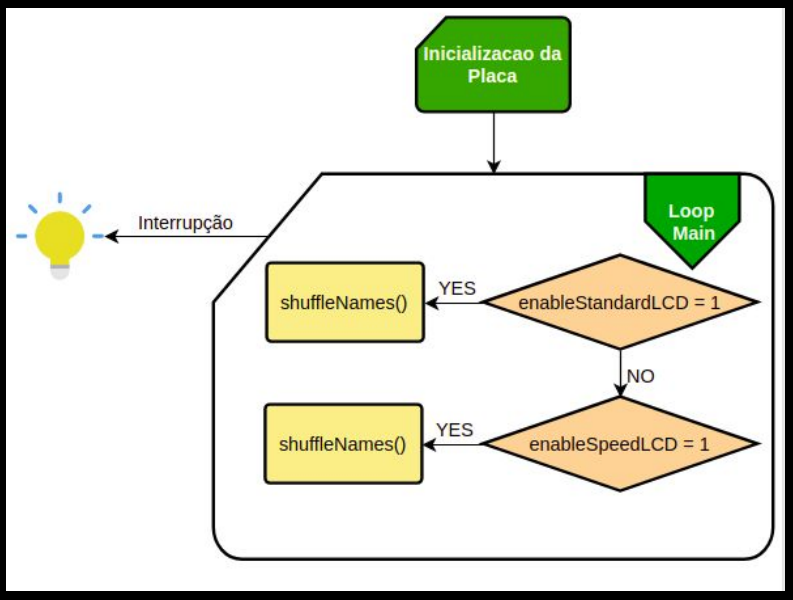

Figura 01 - Main

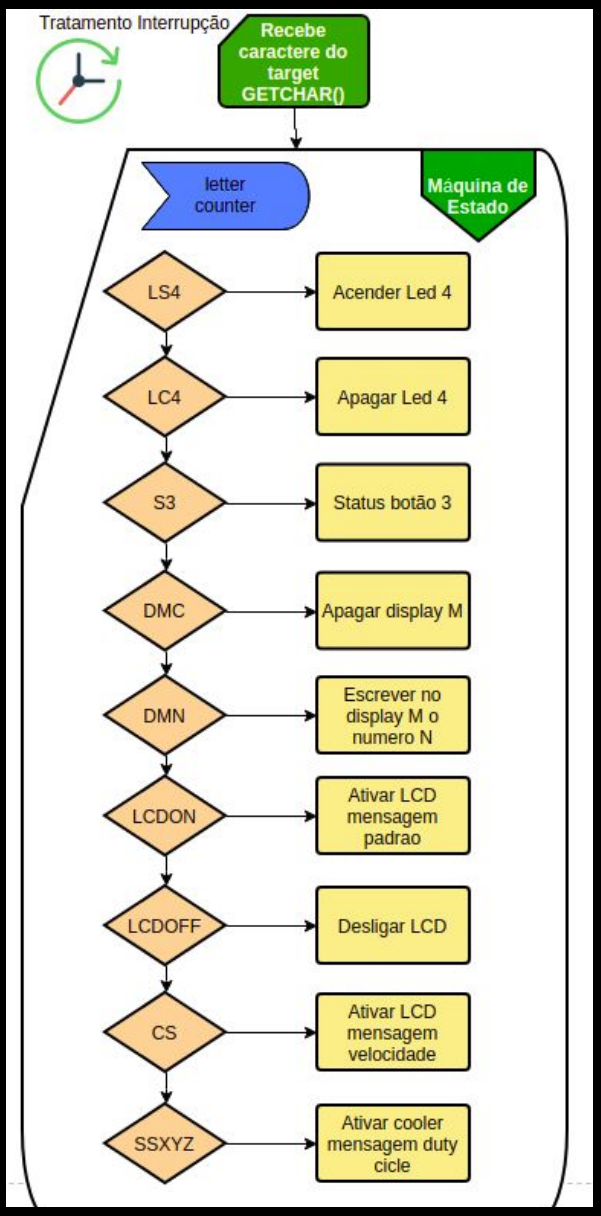

Figura 02 - Máquina de Estado

```c
/* ****************************************/
/* File name: main.c                      */
/* File description: This file implements the   */
/* functions needed to perform the requisits of */
/* the lab04 from ES670                   */
/* Author name: Laura Marchione RA:156169       */
/* and          Victor Cintra Santos RA:157461  */
/* Creation date: 04june2019              */
/* Revision date: 11mjune2019             */
/* ****************************************/

#include "buzzer_hal.h"
#include "es670_peripheral_board.h"
#include "ledswi_hal.h"
#include "mcg_hal.h"
#include "util.h"
#include "debugUart.h"
#include "fsl_debug_console.h"
#include "print_scan.h"
#include "lcd_hal.h"
#include "tc_hal.h"
#include <MKL25Z4.h>

#define CYCLIC_EXECUTIVE_PERIOD 1000 * 1000        // Micro seconds

/**********Definition of globals variables**********/
volatile unsigned int uiFlagNextPeriod = 0;        // Cyclic executive flag
int enableStandardLCD = 1;                         // Enable LCD standard message
int enableSpeedLCD = 0;                            // Enable LCD speed message
int enableTempLCD = 0;
int velocity;                                      // Fan velocity
char velocityChar[2];                              // Auxiliary vector for fan velocity
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
};

/* ********************************************** */
/* 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: cooler_taco_init                   */
/* Method description: Initialize the cooler taco. */
/* 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: heat_sensor_init                */
/* Method description: Initialize the heat sensor.  */
/* thermometer                                  */
/* Input params: n/a                            */
/* Output params: n/a                           */
/* ******************************************** */
void heat_sensor_init(void){
    SIM_SCGC5 |= SIM_SCGC5_PORTE(CGC_CLOCK_ENABLED);// Un-gate PORTE clock
    SIM_SCGC6 |= SIM_SCGC6_TPM0(CGC_CLOCK_ENABLED); // Un-gate TPM0 clock
    PORTE_PCR21 &= ~PORT_PCR_MUX(0b111);            // Configure PTE21 as analog


    ADC0_SC1A &= ~ADC_SC1_DIFF(0b1);                    // Ajusta sinal anal�gico para
single ended
    ADC0_CFG1 |= ADC_CFG1_MODE(0b00);               // Zera o mode para indicar
convers�o de 8 bits
    ADC0_SC1A &= ~ADC_SC1_ADCH(0b11011);                // Reseta o resgistrador ADCH
antes de cada requisi��o de convers�o

    ADC0_SC2 &= ~ADC_SC2_ADTRG(0b1);                    // Indica convers�o por software
    ADC0_SC3 |= ADC_SC3_AVGE(0b1);                  // Ajusta a convers�o para calcular
a m�dia entre os valores anal�gicos lidos
    ADC0_SC3 &= ~ADC_SC3_AVGS(0b11);                     // Ajusta a conversao para
fazer media entre 4 amostras
    ADC0_SC3 &= ~ADC_SC3_ADCO(0b1);                   // Ajusta para conversao produzir
uma amostra por requisi��o
    ADC0_SC1A |= ADC_SC1_AIEN(0b1);
}

/* ******************************************** */
/* 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: turnOnFan                       */
/* Method description: start the fan comunication   */
/* Input params: n/a                            */
/* Output params: n/a                           */
/* ******************************************** */
void turnOnFan(){
    /* 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%
}

/* ******************************************** */
/* Method name: turnOnHeat                      */
/* Method description: start the heater             */
/* Input params: n/a                            */
/* Output params: n/a                           */
/* ******************************************** */
void turnOnHeat(){
    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 PTA12 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: defaultErrorMessage               */
/* Method description: print the default          */
/* error message on the target                    */
/* Input params: n/a                              */
/* Output params: n/a                             */
/* ********************************************** */
void defaultErrorMessage(){
    PUTCHAR('E');
    PUTCHAR('R');
    PUTCHAR('R');
}

/* ********************************************** */
/* Method Name: defaultACKMessage                 */
/* Method description: print the default          */
/* acknowledge message on the target              */
/* Input params: n/a                              */
/* Output params: n/a                             */
/* ********************************************** */
void defaultACKMessage(){
    PUTCHAR('A');
    PUTCHAR('C');
    PUTCHAR('K');
```

```
}

/* ******************************************** */
/* Method Name: ungateDisplay7Seg                */
/* Method description: enable the port,          */
/* clock and recorders of our target             */
/* Input params: n/a                             */
/* Output params: n/a                            */
/* ******************************************** */
void ungateDisplay7Seg(){
    SIM_SCGC5 = SIM_SCGC5_PORTC(0b11110011111111); // Liberacao do clock (ungate) da
porta C e seus respectivos registradores
    PORTC_PCR0 = PORT_PCR_MUX(0x01); // Configurando registradores do display de 7seg da
porta C como GPIO
    PORTC_PCR1 = PORT_PCR_MUX(0x01); //.
    PORTC_PCR2 = PORT_PCR_MUX(0x01); //.
    PORTC_PCR3 = PORT_PCR_MUX(0x01); //.
    PORTC_PCR4 = PORT_PCR_MUX(0x01); //.
    PORTC_PCR5 = PORT_PCR_MUX(0x01); //.
    PORTC_PCR6 = PORT_PCR_MUX(0x01); //.
    PORTC_PCR7 = PORT_PCR_MUX(0x01); //.
    PORTC_PCR13 = PORT_PCR_MUX(0x01); // Configurando enable display 01
    PORTC_PCR12 = PORT_PCR_MUX(0x01); // Configurando enable display 02
    PORTC_PCR11 = PORT_PCR_MUX(0x01); // Configurando enable display 03
    PORTC_PCR10 = PORT_PCR_MUX(0x01); // Configurando enable display 04
    GPIOC_PDDR = GPIO_PDDR_PDD(0b11110011111111); // Configurando registradores da porta
C como outputs
}

/* ******************************************** */
/* Method Name: display7SegController            */
/* Method description: receives the              */
/* number display and the number that            */
/* will be written on it; returns the            */
/* binary to control the display                 */
/* Input params: displayNumChar, numChar         */
/* Output params: displaySignal                  */
/* ******************************************** */
int display7SegController (char displayNumChar, char numChar){
```

```
    int displayNum = displayNumChar - '0'; //Passamos os valores de char para int e para
os indices dos vetores
    int num = numChar - '0';
   int displaySignal;                    //Retorno da funcao

   char displayOptions [5];               // Vetor com os valores em binario dos enables
dos displays
    displayOptions [1] = 0b00100000;
    displayOptions [2] = 0b00010000;
    displayOptions [3] = 0b00001000;
    displayOptions [4] = 0b00000100;
   char displayNumbers [10];              // Vetor com os valores em binario dos numeros
em segmentos
    displayNumbers [0] = 0b00111111;
    displayNumbers [1] = 0b00000110;
    displayNumbers [2] = 0b01011011;
    displayNumbers [3] = 0b01001111;
    displayNumbers [4] = 0b01100110;
    displayNumbers [5] = 0b01101101;
    displayNumbers [6] = 0b01111101;
    displayNumbers [7] = 0b00100111;
    displayNumbers [8] = 0b01111111;
    displayNumbers [9] = 0b01101111;

    int disp = displayOptions[displayNum];
    disp =  disp  << 8;
    int numInt = displayNumbers[num];
    displaySignal = disp | numInt;          // Criamos o binário com o enable e número a
ser escrito

   return displaySignal;
}

/* ********************************************** */
/* Method Name: shuffleNames                      */
/* Method description: standard message routine   */
/* Input params: n/a                              */
/* Output params: n/a                             */
/* ********************************************** */
void shuffleNames(){
```

```
char nomes[] = " Laura e Victor";
char nomesAux[16];

lcd_setCursor(0,1);
lcd_writeString("ES670");
util_genDelay10ms();

char aux;
for(;;){
    util_genDelay10ms();
    lcd_setCursor(1,0);
    util_genDelay10ms();
    int i = 0;
    aux = nomes[i];
    for (i; i < 14; i++){
        nomesAux[i] = nomes[i+1];
    }
    nomesAux[i] = aux;

    for(int j = 0; j<16; j++){
        nomes[j] = nomesAux[j];
    }
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    util_genDelay10ms();
    lcd_writeString(nomes);
    util_genDelay10ms();

    if(enableStandardLCD==0){
        break;
    }
```

```
    }
}

/* ********************************************** */
/* Method Name: shuffleNames                      */
/* Method description: speed message routine      */
/* Input params: n/a                              */
/* Output params: n/a                             */
/* ********************************************** */
void shuffleSpeed(){
    velocity = cooler_getRPS(CYCLIC_EXECUTIVE_PERIOD);
    velocityChar[2] = "  ";
    sprintf(velocityChar, "%d", velocity);
    util_genDelay1ms();
    lcd_setCursor(0,0);
    lcd_writeString("Velocidade [RPS]:");
    lcd_setCursor(1,0);
    lcd_writeData(velocityChar[0]);
    util_genDelay1ms();
    lcd_setCursor(1,1);
    lcd_writeData(velocityChar[1]);
}

/* ********************************************** */
/* Method Name: controlSpeed                      */
/* Method description: duty cicle message routine */
/* Input params: speed[]                          */
/* Output params: n/a                             */
/* ********************************************** */
void controlSpeed(char speed[]){
    int speedInt;
    speedInt = (speed[0] - '0')*100;
    speedInt = speedInt + ((speed[1] - '0')*10);
    speedInt = speedInt + (speed[2] - '0');
    speedInt = (speedInt*255)/100;
    TPM1_C1V = TPM_CnV_VAL(speedInt);
    lcd_setCursor(0,0);
    lcd_writeString("Duty Cicle:");
    lcd_setCursor(1,0);
    lcd_writeString(speed);
```

```
}

/* ********************************************** */
/* Method Name: controlTemperature                */
/* Method description: duty cicle message routine */
/* Input params: temperature[]                    */
/* Output params: n/a                             */
/* ********************************************** */
void controlTemperature(char temperature[]){
    int tempInt;
    tempInt = (temperature[0] - '0')*100;
    tempInt = tempInt + ((temperature[1] - '0')*10);
    tempInt = tempInt + (temperature[2] - '0');
    tempInt = (tempInt*255)/100;
    if (tempInt <= 50){
    TPM1_C0V = TPM_CnV_VAL(tempInt);
    }
    lcd_setCursor(0,0);
    lcd_writeString("Temperatura - Duty Cicle:");
    lcd_setCursor(1,0);
    lcd_writeString(temperature);
}

/* ********************************************** */
/* Method Name: dataTargetCommand                 */
/* Method description: receives the letter wich is  */
/* read on the program that communicates with       */
/* the target and controls our state machine        */
/* Input params: letter                           */
/* Output params: n/a                             */
/* ********************************************** */
void dataTargetCommand(char letter){
    static int counter = 0;
    static char switchStatus;
    static char display;
    static char speed[3];
    static char temperature[3];
    int signalDisplay;

    if (counter == 0){
```

```
    if (letter == 'L'){
        counter = 1;
    }
    else if (letter == 'S'){
        counter = 6;
    }
    else if (letter == 'D'){
        counter = 8;
    }
    else if (letter == 'C'){
        counter = 13;
    }
    else if (letter == 'H'){
        counter = 17;
    }
    else{
        counter = 0;
        defaultErrorMessage();
    }
}
else if (counter == 1){
    if (letter == 'S'){
        counter = 2;
    }
    else if (letter == 'C'){
        counter = 3;
    }
    else{
        counter = 0;
        defaultErrorMessage();
    }
}
else if (counter == 2){
    if (letter == '4'){
        counter = 0;
        ledswi_setLed(4u);
        defaultACKMessage();
    }
    else{
        counter = 0;
```

```
        defaultErrorMessage();
    }
}
else if (counter == 3){
    if (letter == '4'){
        counter = 0;
        ledswi_clearLed(4u);
        defaultACKMessage();
    }
    else if (letter == 'D'){
        counter = 10;
    }
    else{
        counter = 0;
        defaultErrorMessage();
    }
}
else if (counter == 6){
    if (letter == '3'){
        counter = 0;
        switchStatus = ledswi_getSwitchStatus(3u);
        defaultACKMessage();
    }
    else if (letter == 'S'){
        counter = 14;
    }
    else{
        counter = 0;
        defaultErrorMessage();
    }
}

else if (counter == 8){
    if (letter == '1'||'2'||'3'||'4'){
        counter = 9;
        display = letter;
    }
    else{
        counter = 0;
        defaultErrorMessage();
```

```
        }
    }
    else if (counter == 9){
        if(letter == 'C'){
            counter = 0;
            GPIOC_PCOR = GPIO_PCOR_PTCO(0b11111111111111);
            defaultACKMessage();
        }
        else if (letter == '0'||'1'||'2'||'3'||'4'||'5'||'6'||'7'||'8'||'9'){
            counter = 0;
            signalDisplay = display7SegController (display, letter);
            util_genDelay088us();
            GPIOC_PCOR = GPIO_PCOR_PTCO(0b11111111111111);
            GPIOC_PSOR = GPIO_PSOR_PTSO(signalDisplay);
            defaultACKMessage();
        }

        else {
            counter = 0;
            defaultErrorMessage();
        }
    }
    else if (counter == 10){
        if(letter == 'O'){
            counter = 11;
        }
        else {
            counter = 0;
            defaultErrorMessage();
        }
    }
    else if (counter == 11){
        if(letter == 'N'){
            counter = 0;
            lcd_setCursor(0,0);
            lcd_writeString("
");
            lcd_setCursor(1,0);
            lcd_writeString("
");
```

```
            lcd_setCursor(0,0);
            lcd_writeString("
");
            lcd_setCursor(1,0);
            lcd_writeString("
");
            enableSpeedLCD = 0;
            enableTempLCD = 0;
            enableStandardLCD = 1;
            defaultACKMessage();
        }
        else if(letter == 'F'){
            counter = 12;
        }
        else {
            counter = 0;
            defaultErrorMessage();
        }
    }
    else if (counter == 12){
        if(letter == 'F'){
            counter = 0;
            enableStandardLCD = 0;
            enableSpeedLCD = 0;
            enableTempLCD = 0;
            lcd_setCursor(0,0);
            lcd_writeString("
");
            lcd_setCursor(1,0);
            lcd_writeString("
");
            lcd_setCursor(0,0);
            lcd_writeString("
");
            lcd_setCursor(1,0);
            lcd_writeString("
");
            defaultACKMessage();
        }
        else {
```

```
            counter = 0;
            defaultErrorMessage();
        }
    }
    else if (counter == 13){
        if (letter == 'S'){
            counter = 0;
            enableStandardLCD = 0;
            enableTempLCD = 0;
            lcd_setCursor(0,0);
            lcd_writeString("
");
            lcd_setCursor(1,0);
            lcd_writeString("
");
            lcd_setCursor(0,0);
            lcd_writeString("
");
            lcd_setCursor(1,0);
            lcd_writeString("
");
            lcd_setCursor(0,0);
            lcd_writeString("
");
            lcd_setCursor(1,0);
            lcd_writeString("
");
            lcd_setCursor(0,0);
            lcd_writeString("
");
            lcd_setCursor(1,0);
            lcd_writeString("
");
            enableSpeedLCD = 1;
            defaultACKMessage();

        }
        else {
            counter = 0;
            defaultErrorMessage();
```

```
        }
    }
    else if (counter == 14){
        if (letter == '0'||'1'){
            counter = 15;
            speed[0] = letter;
        }
        else {
            counter = 0;
            defaultErrorMessage();
        }
    }
    else if (counter == 15){
        if (letter == '0'||'1'||'2'||'3'||'4'||'5'||'6'||'7'||'8'||'9'){
            counter = 16;
            speed[1] = letter;
        }
        else {
            counter = 0;
            defaultErrorMessage();
        }
    }
    else if (counter == 16){
        if (letter == '0'||'1'||'2'||'3'||'4'||'5'||'6'||'7'||'8'||'9'){
            counter = 0;
            speed[2] = letter;
            defaultACKMessage();
            enableStandardLCD = 0;
            enableSpeedLCD = 0;
            enableTempLCD = 0;
            lcd_setCursor(0,0);
            lcd_writeString("
");
            lcd_setCursor(1,0);
            lcd_writeString("
");
            lcd_setCursor(0,0);
            lcd_writeString("
");
            lcd_setCursor(1,0);
```

```
            lcd_writeString("
");
            lcd_setCursor(0,0);
            lcd_writeString("
");
            lcd_setCursor(1,0);
            lcd_writeString("
");
            lcd_setCursor(0,0);
            lcd_writeString("
");
            lcd_setCursor(1,0);
            lcd_writeString("
");
            controlSpeed(speed);
        }
        else {
            counter = 0;
            defaultErrorMessage();
        }
    }
    else if (counter == 17){
            if (letter == 'S'){
                counter = 18;
            }
            else if (letter == 'C'){
                counter = 0;
                enableStandardLCD = 0;
                lcd_setCursor(0,0);
                lcd_writeString("
");
                lcd_setCursor(1,0);
                lcd_writeString("
");
                lcd_setCursor(0,0);
                lcd_writeString("
");
                lcd_setCursor(1,0);
                lcd_writeString("
");
```

```
                lcd_setCursor(0,0);
                lcd_writeString("
");
                lcd_setCursor(1,0);
                lcd_writeString("
");
                lcd_setCursor(0,0);
                lcd_writeString("
");
                lcd_setCursor(1,0);
                lcd_writeString("
");
                enableSpeedLCD = 1;
                enableTempLCD = 1;
                defaultACKMessage();

            }
            else {
                counter = 0;
                defaultErrorMessage();
            }
        }
    else if (counter == 18){
            if (letter == '0'){
                counter = 19;
                temperature[0] = letter;
            }
            else {
                counter = 0;
                defaultErrorMessage();
            }
        }
        else if (counter == 19){
            if (letter == '0'||'1'||'2'||'3'||'4'||'5'){
                counter = 20;
                temperature[1] = letter;
            }
            else {
                counter = 0;
                defaultErrorMessage();
```

```
            }
        }
        else if (counter == 20){
            if (letter == '0'||'1'||'2'||'3'||'4'||'5'||'6'||'7'||'8'||'9'){
                counter = 0;
                temperature[2] = letter;
                defaultACKMessage();
                enableStandardLCD = 0;
                enableSpeedLCD = 0;
                enableTempLCD = 0;
                lcd_setCursor(0,0);
                lcd_writeString("
");
                lcd_setCursor(1,0);
                lcd_writeString("
");
                lcd_setCursor(0,0);
                lcd_writeString("
");
                lcd_setCursor(1,0);
                lcd_writeString("
");
                lcd_setCursor(0,0);
                lcd_writeString("
");
                lcd_setCursor(1,0);
                lcd_writeString("
");
                lcd_setCursor(0,0);
                lcd_writeString("
");
                lcd_setCursor(1,0);
                lcd_writeString("
");
                controlTemperature(temperature);
            }
            else {
                counter = 0;
                defaultErrorMessage();
            }
```

```
        }

}

/* ********************************************** */
/* 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);
    int dataTarget;

    dataTarget = GETCHAR();
    dataTargetCommand(dataTarget);

    NVIC_EnableIRQ(UART0_IRQn);
}

void ADC0_IRQHandler(void){
    NVIC_DisableIRQ(ADC0_IRQn);
    int temperature;

    temperature = ADC0_RA;



    NVIC_EnableIRQ(ADC0_IRQn);
/* Obter resultado da convers□o */
/* atribuindo valor de ADC0_RA */
/* a alguma vari□vel. Acrescente */
/* tudo que julgar pertinente aqui */
}

int main(void){

    mcg_clockInit();                            // Inicializamos o clock da placa
    debugUart_init();                           // Inicializamos o debug UART
    NVIC_ClearPendingIRQ(UART0_IRQn);           // Configurando interrupcoes UART
```

```
    NVIC_EnableIRQ(UART0_IRQn);                     // ...
    NVIC_EnableIRQ(ADC0_IRQn);                      // Habilitamos interrup��es ADC0
    UART0_C2_REG(UART0) |= UART0_C2_RIE(1);         // Enable do recebimento de
interrupcoes
    ledswi_initLedSwitch(1u, 3u);                   // Inicializamos o LED e o Switch
    ungateDisplay7Seg();                            // Inicializamos o display 7seg
    lcd_initLcd();                                  // Inicializamos o LCD

    /* configure cyclic executive interruption */
    tc_installLptmr0(CYCLIC_EXECUTIVE_PERIOD, main_cyclicExecuteIsr);
    turnOnFan();                                    // Inicializamos o fan
    cooler_taco_init();                             // Inicializamos o tacometro

    turnOnHeat();                                   // Inicializamos o aquecedor
    heat_sensor_init();

    for(;;){

        if(enableStandardLCD==1){
            shuffleNames();
        }
        else if(enableSpeedLCD==1){
            shuffleSpeed();
        }

        /* Enquanto a interrupcaoo nao e atingida uiFlagNextPeriod==0 */
        /* Ao atingir a interrupcao uiFlagNextPeriod==1 e laco termina*/
        while(!uiFlagNextPeriod);
        /* Reinicializacao do uiFlagNextPeriod */
        uiFlagNextPeriod = 0;
    }

    return(0);
}
```