ES670B - Projeto de Sistemas Embarcados
Relatório 04 - Laboratório 4/7
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 realizar medições de velocidade de rotação do cooler que há na placa. Para isso fizemos que que o *target* trabalhasse com um executivo cíclico de período pré-definido, acionamos as hélices do cooler e implementamos o código para realizar as medições. Nele, atualizamos nossa máquina de estado para incluir o estado do comando *CS* na comunicação do *hoste* e *target*.

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**
Ler velocidade do cooler	main.c - turnOnFan(); - cooler_taco_init() - defaultACKMessage(); - defaultErrorMessage(); - dataTargetCommand(char letter); - shuffleSpeed(); - cooler_getRPS(CYCLIC_EXECUTIVE_PERIOD);

Notas

Nesse laboratório, a maior dificuldade foi de manipular o valor de velocidade do cooler retornada pela função *cooler_getRPS* , um inteiro, que tivemos que passar para um char para podermos manipular

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.

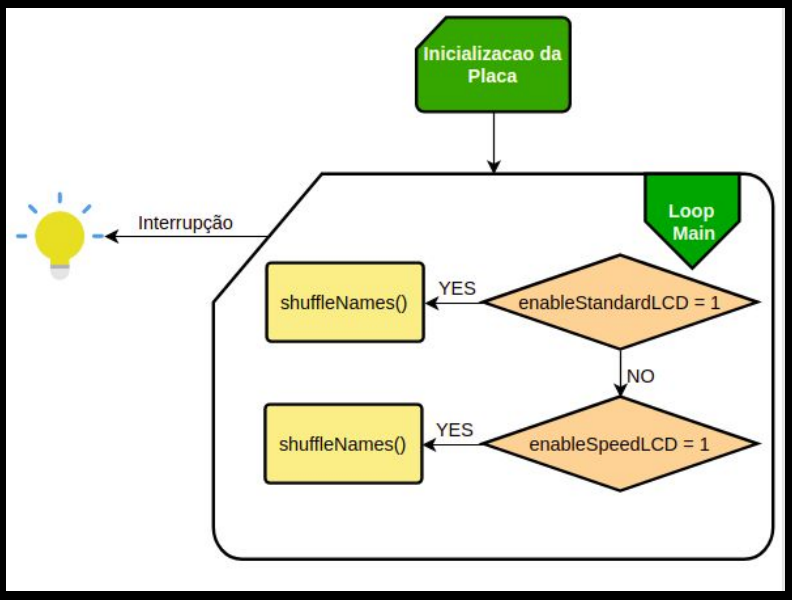

Figura 01 - Main

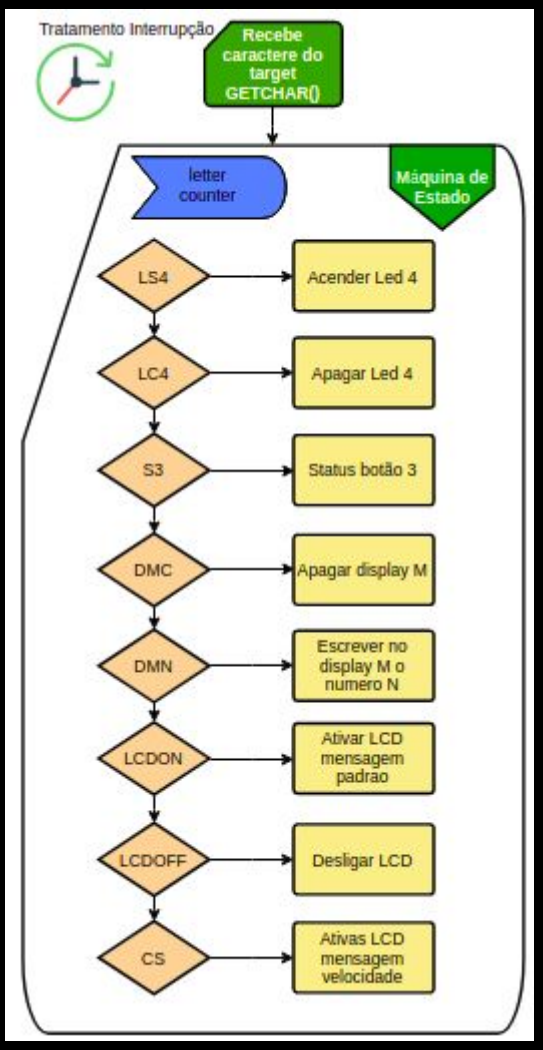

Figura 02 - Máquina de Estado

```
/* ******************************************/
/* 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: 10mai2019                 */
/* Revision date: 19may2019                 */
/* ******************************************/

#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 velocity;                               // Fan velocity
char velocityChar[2];                       // Auxiliary vector for fan velocity


/* ********************************************** */
/* 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: 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(){
    /* 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: 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(0,1);
    lcd_writeData(velocityChar[0]);
    util_genDelay1ms();
    lcd_setCursor(0,2);
    lcd_writeData(velocityChar[1]);
}

/* ********************************************** */
/* 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;
    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{
        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{
        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;
            enableStandardLCD = 1;
            defaultACKMessage();
        }
        else if(letter == 'F'){
            counter = 12;
        }
```

```
        else {
            counter = 0;
            defaultErrorMessage();
        }
    }
    else if (counter == 12){
        if(letter == 'F'){
            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("
");
            defaultACKMessage();
        }
        else {
            counter = 0;
            defaultErrorMessage();
        }
    }
    else if (counter == 13){
        if (letter == 'S'){
            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;
            defaultACKMessage();

        }
        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);
}

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);               // ...
    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

    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);
}
```