ES670B - Projeto de Sistemas Embarcados
Relatório 03 - Laboratório 3/7
Laura Marchione - RA 156169
Victor Cintra Santos - RA 157461

Objetivos

Nesse relatório temos os seguintes requisitos de projeto: enviar comandos para do *host* para o *target*; retornar status dos periféricos do *target* para o *host* por meio de comunicação assíncrona. Tal comunicação será baseada na conexão serial assíncrona fornecida pelo USB do OpenSDA. Na Tarefa 01 do projeto tivemos que fazer as devidas configurações para estabelecermos conexão com a placa e configurarmos o display LCD. Nele, escrevemos uma mensagem padrão com sigla da disciplina na primeira linha e os nomes dos integrantes na segunda linha, sendo exibidos da direita para a esquerda continuamente. Na Tarefa 02 tivemos que alterar nossa máquina de estado do Laboratório 02 para incluir os comandos de LCDON e LCDOFF para, respectivamente, ligar o LCD com a mensagem da primeira tarefa e desligar o LCD, utilizando o conceito de interrupções.

Modelagem

Na Figura 01, no apêndice, há o diagrama UML desenvolvido para tal projeto. Nele, mostramos a máquina de estados implementada. Na lógica, utilizamos uma variável estática para guardarmos o estado e fazermos as verificações em cada interação. Com o contador da máquina de estados e o caractere recebido do *target* conseguimos detectar o comando que queremos utilizar e chamamos as funções apropriadas. Nossa máquina de estado é chamada como tratamento de interrupção, que ocorre quando enviamos uma letra como comando entre a comunicação serial entre o *host* para o nosso *target*. Ao acontecer tal interrupção, nosso código chama o tratamento com a máquina de estado implementada.

Matriz de Rastreabilidade

Requisito	**Implementação**
Acionar/ Desligar LED	ledswi_hal.c - void ledswi_initLedSwitch(char cLedNum, char cSwitchNum); - void ledswi_setLed (char cLedNum); main.c - defaultACKMessage(); - defaultErrorMessage(); - dataTargetCommand(char letter);
Monitorar chave “push button”	ledswi_hal.c - void ledswi_initLedSwitch(char cLedNum, char cSwitchNum); - switch_status_type_e ledswi_getSwitchStatus(char cSwitchNum); main.c - defaultACKMessage(); - defaultErrorMessage(); - dataTargetCommand(char letter);
Apagar Display M/ Escreve no display M o número N	main.c - ungateDisplay7Seg(); - display7SegController(char displayNumChar, char numChar); - defaultACKMessage(); - defaultErrorMessage(); - dataTargetCommand(char letter);
Ligar/desligar LCD	main.c - shuffleNames(): - lcd_initLcd();

Notas

Nesse laboratório, a maior dificuldade foi de manipular o display LCD, já que para apaga-lo, devido aos tempos de interrupção, tivemos que incluir funções de delay para que desse tempo de apagarmos o display, antes do tratamento da interrupção terminar

Apêndice

Na Figura 02 e 03 temos os estados do botão, apertado ou não, respectivamente, obtidos através da inspeção da nossa variável “switchStatus” que recebe o retorno da funcao “ledswi_getSwitchStatus” que mostra o status do switch. Na Figura 01 temos o diagrama UML da nossa máquina de estado do laboratório passado e, na sequência, sua complementação para implementarmos a lógica desse laboratório. Abaixo, após as figuras do apêndice, há o código do desenvolvimento do projeto.

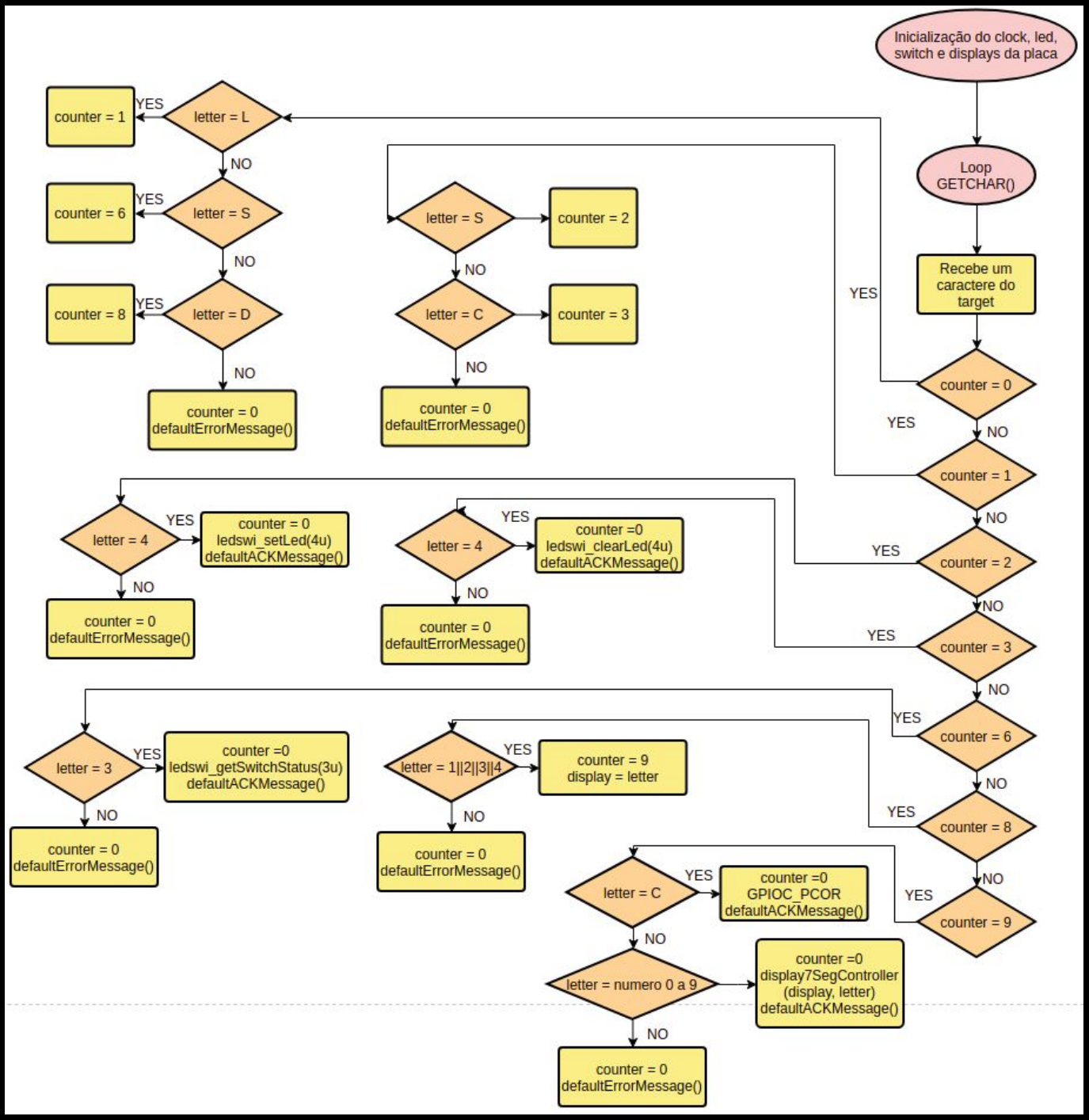

Figura 01 - Máquina de Estados

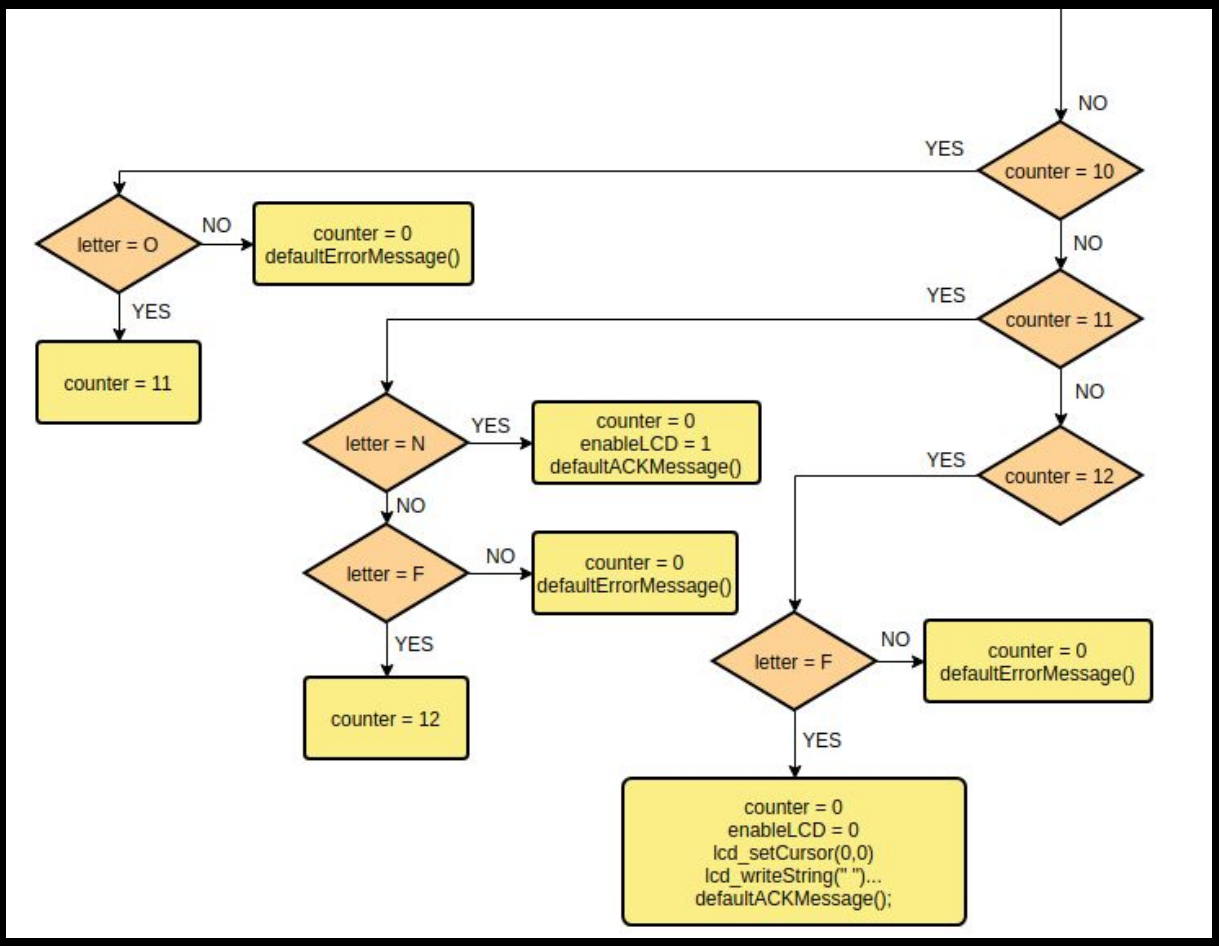

Figura 01 - Continuação Máquina de Estados

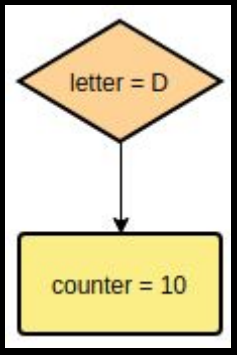

Figura 01 - Verificação adicional quando counter = 3

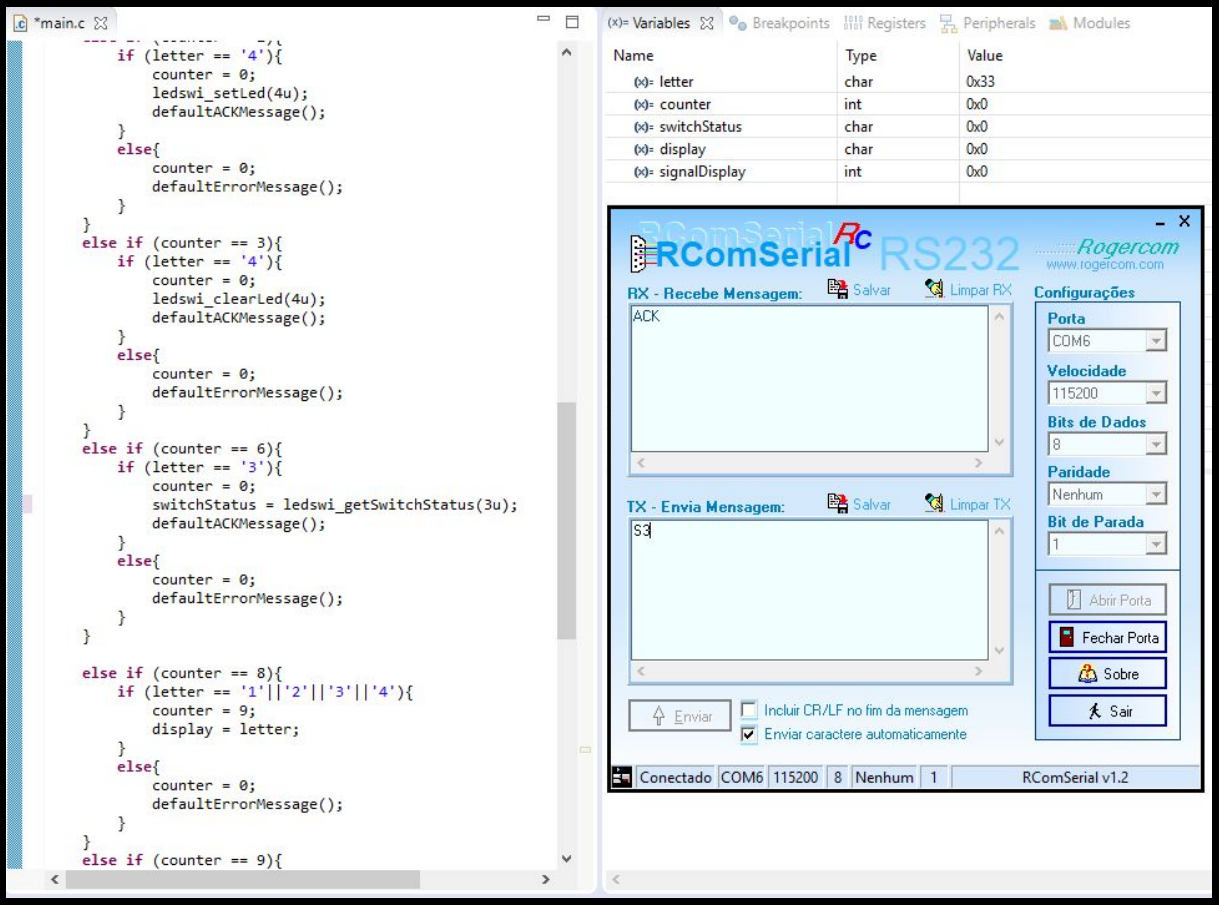

Figura 02

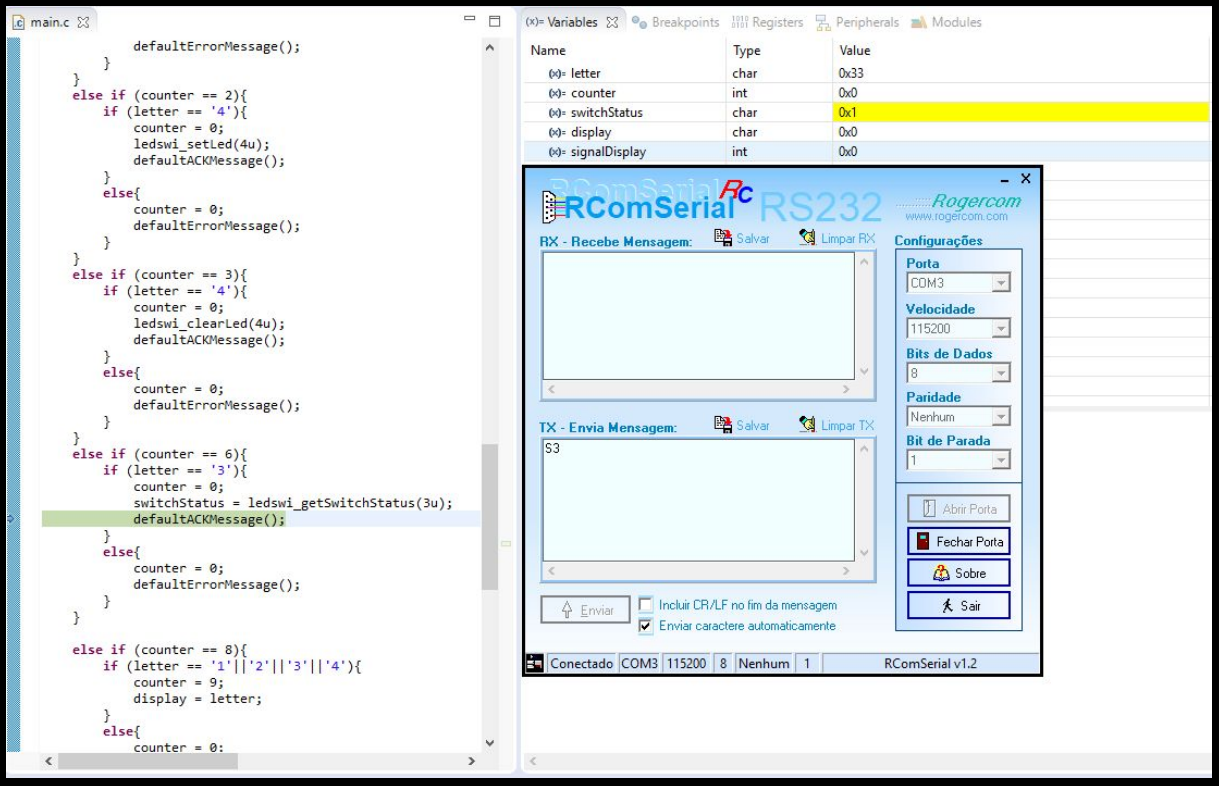

Figura 03

```c
/* ******************************************/
/* File name: main.c                        */
/* File description: This file implements the   */
/* functions needed to perform the requisits of */
/* the lab03 from ES670                     */
/* Author name: Laura Marchione RA:156169       */
/* and          Victor Cintra Santos RA:157461  */
/* Creation date: 03may2019                 */
/* Revision date: 10may2019                 */
/* ******************************************/

#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 <MKL25Z4.h>

int enableLCD = 1;

/* 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: 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{
            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;
        enableLCD = 1;
        defaultACKMessage();
    }
    else if(letter == 'F'){
        counter = 12;
    }
    else {
        counter = 0;
        defaultErrorMessage();
```

```
        }
    }
    else if (counter == 12){
        if(letter == 'F'){
            counter = 0;
            enableLCD = 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();
        }
    }
}

/* Method Name: shuffleNames            */
/* Method description: LCD 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();
        lcd_writeString(nomes);
        util_genDelay10ms();

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

```
/* ********************************************** */
/* 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);       // Configure UART0 interrupts
    NVIC_EnableIRQ(UART0_IRQn);
    UART0_C2_REG(UART0) |= UART0_C2_RIE(1); // Receive interrupt enable
    ledswi_initLedSwitch(1u, 3u);
    ungateDisplay7Seg();
    lcd_initLcd();

    for(;;){
        if(enableLCD==1){                   // Se mantermos enableLCD
verdadeiro, mantemos o chamado
            shuffleNames();                 // do metodo para imprimirmos a
mensagem padrao no LCD
        }
    }

    return(0);
}
```