O módulo Serial LCD Controller (SLCDC) implementa a interface com o LCD, fazendo a receção em série da informação enviada pelo módulo de controlo e entregando-a posteriormente ao LCD

1 Modulo SLCDC

O módulo SLCDC recebe em série uma mensagem constituída por nove (9) bits de informação e um (1) bit de paridade. A comunicação com este módulo realiza-se segundo o protocolo ilustrado na Figura 1, em que o bit RS é o primeiro bit de informação e indica se a mensagem é de controlo ou dados. Os seguintes oito (8) bits contêm os dados a entregar ao LCD. O último bit contém a informação de paridade par, utilizada para detetar erros de transmissão.

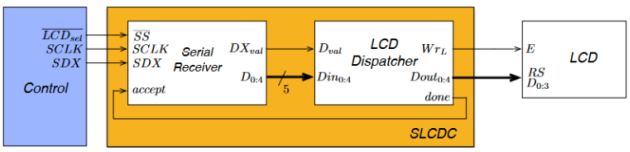

Figura 1 – Diagrama de blocos do módulo SLCDC

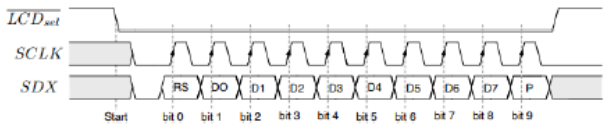

Figura 2 – Protocolo de comunicação com o módulo Serial LCD Controller

2 Interface com o *Control*

Implementou-se os módulos TUI e LCD em *software*, recorrendo a linguagem Kotlin, o bloco Serial Receiver e o bloco LCD Dispatcher em hardware, recorrendo a linguagem VHDL.

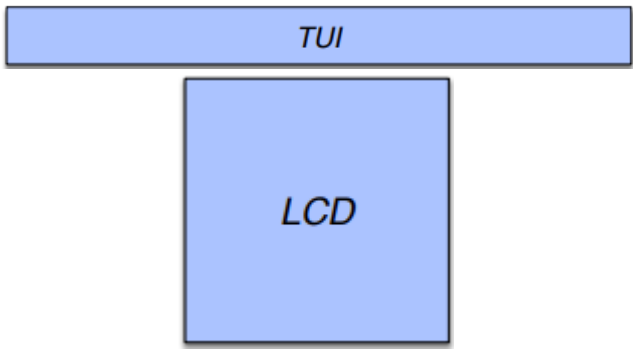

Figura 3 Classe LCD e TUI

As classes LCD e TUI desenvolvidas são descritas nas secções 2.1. e 2.2, e o código fonte desenvolvido nos Anexos C e B, respetivamente.

2.1 Classe *LCD*

- Init, que inicia a classe
- writeByteSerial, escreve um byte (comando ou dados) no LCD em série
- writeByte, chama a função wrteByteParallel e escreve um byte de comando/dados no LCD
- writeCMD, chama a função writeByte e escreve um comando no LCD
- writeDATA, chama a função writeByte e escreve um dado no LCD
- write, recebe um char e chama a função writeDATA e escreve um carácter na posição corrente
- write, recebe uma string e chama a função write e escreve uma string na posição corrente
- cursor, recebe uma linha e uma coluna e envia um comando para posicionar o cursor nessa posição
- clear, envia um comando para limpar o ecrã e posicionar o cursor em (0,0)
- Invader, spaceShip e loadCustomChars, são responsáveis pelo desenho da nave e do invader e, pela customização dos mesmos

2.2 Classe TUI

- Escrever, é responsável pela escrita no LCD na parte direita do ecrã e ir decrementando as colunas.
- Align, é responsável por escrever o texto que é passado na linha e na coluna como parâmetro, usando funções do LCD
- AlignLeft, é responsável por escrever o texto à esquerda do LCD que é passado na linha como parâmetro, usando funções do LCD

- AlignRight, é responsável por escrever o texto à direita do LCD que é passado na linha como parâmetro, usando funções do LCD
- AlignMiddle, é responsável por escrever o texto alinhado ao centro do LCD que é passado na linha como parâmetro, usando funções do LCD
- forWaitKey, é responsável por chamar a função do KBD que espera por uma tecla passado como timeout
- clrLCD, é responsável por limpar o LCD
- cursor, é responsável por colocar o cursor na linha e coluna que são passadas como parâmetro usando função do LCD
- .init, inicia o KBD e o LCD

3 Serial Receiver

O bloco Serial Receiver do módulo SLCDC é constituído por quatro blocos principais: i) um bloco de controlo; ii) um bloco conversor série paralelo; iii) um contador de bits recebidos; e iv) um bloco de validação de paridade, designados por Serial Control, Shift Register, Counter e Parity Check respetivamente.

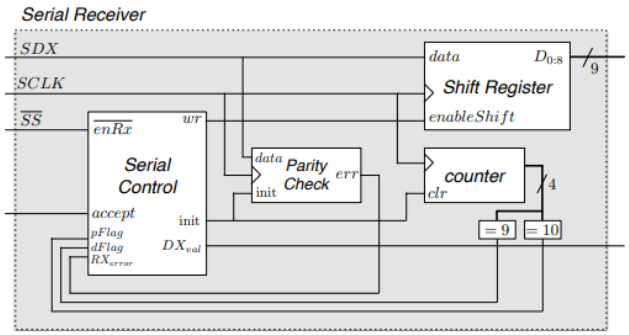

Figura 4 – Diagrama de blocos do bloco Serial Receiver

- O Serial Control é uma máquina de estados com 5 estados, Initializing, Starting, Waiting, Acepted e ItsDone. Começa por ativar o init no primeiro estado, dando assim init ao Parity Check (reset) e clear ao Counter, quando o sinal enRX tiver a ‘0’, a máquina transita para o próximo estado Starting ativando o sinal wr, dando assim enableShift ao Shift Register.

  Para transitar para o próximo estado Waiting, é feita novamente a verificação do sinal enRX, se este tiver o valor ´1´retornará para o estado inicial Initializing, se o sinal enRX tiver a ‘0’, e o sinal dFlag tiver a ‘1’ será feita a transição.

  Após isto, é feita novamente a verificação do sinal enRX, caso este esteja ativo, retornará para o estado incial, Initializing se o sinal estiver a ‘0’, é feita a verificação do sinal pFlag, se este tiver a ‘1’, a máquina transita para o estado Acepted.

  Neste estado, é feita uma verificação do sinal RXerror onde, caso este esteja ativo, retornará para o estado incial Initializing, se o sinal estiver a ‘0’, passará para o próximo e último estado ItsDone.

  Neste último estado o sinal DXval é ativo e, é feita uma verificação do sinal Accept, caso este esteja a ‘1’ retornará ao estado inicial.

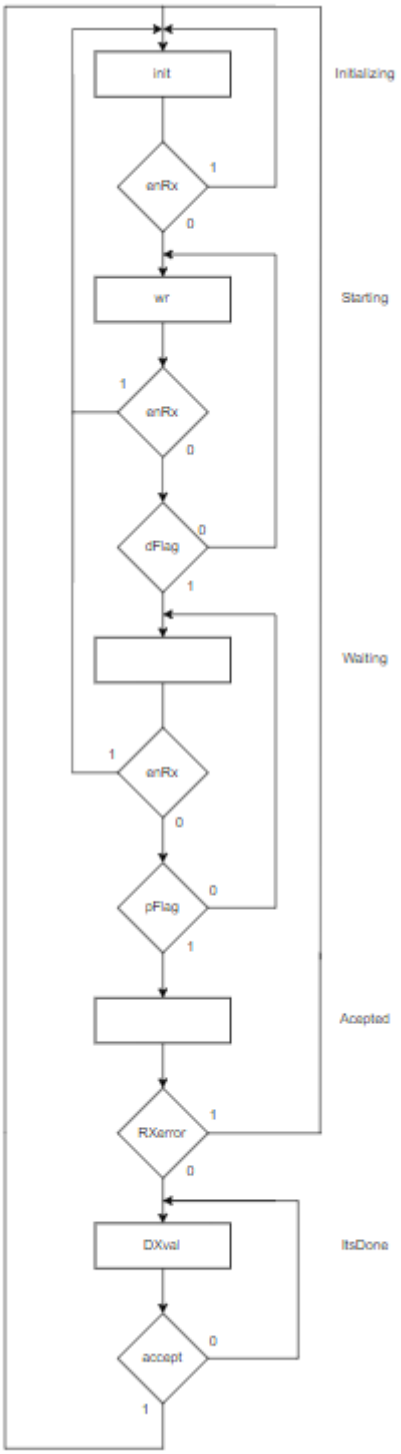

Figura 5 – Máquina de estados do Serial Control

LCD, através da ativação do sinal W*rL*. Este bloco é apenas constituído pelo Control.

- O LCDDispacherControl é uma máquina de estados com três estados, WaitingDval, ReceivingDval e DoneReceived. No primeiro estado WaitingDval, começa por verificar se o sinal Dval encontra-se a '1', se sim, significa que a trama foi recebida e a máquina transita para o próximo estado sendo este o ReceivingDval.
  Neste estado, o sinal Wrl é ativo e transita para o estado DoneReceived onde o sinal Wrl deixa de estar ativo, deixando ativo o sinal done ativo, neste estado, o Dval é novamente verificado, porém neste caso se este tiver a '0' retornará para o estado inicial WaitingDval, significando que a trama já foi processada e a máquina poderá esperar por uma nova trama.

4 LCD Dispatcher

O bloco LCD Dispatcher é responsável pela entrega das tramas válidas recebidas pelo bloco Serial Receiver ao

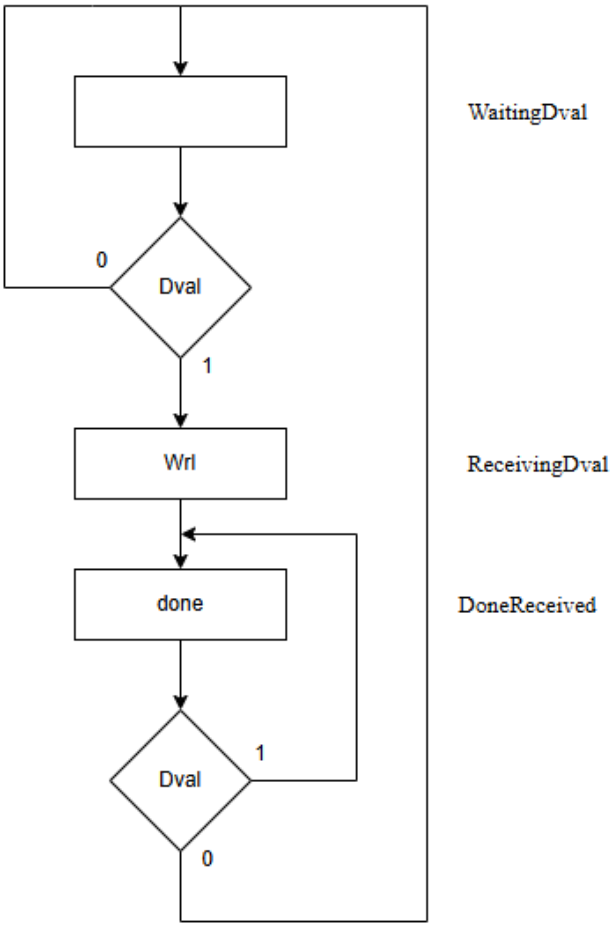

Figura 6 – Máquina de estados do LCD Dispatcher Control

5 Conclusão

Com a implementação deste módulo foi possível a interação com o LCD a partir de dois blocos: Serial Receiver e LCD Dispatcher. Estes dois blocos têm a função de ler e transmitir a mensagem enviada pelo bloco de controlo até ao LCD.

A. Descrição VHDL do bloco *SLCDC*

Código VHDL do SLCDC:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity SLCDC is
    port(
            SS : in std_logic;
            Reset : in std_logic;
            SCLK : in std_logic;
            Clk : in std_logic;
            SDX : in std_logic;
            WrL : out std_logic;
            DoutSLCDC: out std_logic_vector(8 downto 0)
            );
end SLCDC;
architecture structural of SLCDC is
component SerialReceiver is
```

```
        port(
                Reset : in std_logic;
                SDX : in std_logic;
                Clk : in std_logic;
                SCLK : in std_logic;
                SS : in std_logic;
                accept : in std_logic;
                D : out std_logic_vector(8 downto 0);
                DXval : out std_logic
    );
    end component;
    component LCDDispatcher is
        port(
                Dval : in std_logic;
                Din : in std_logic_vector(8 downto 0);
                WrL : out std_logic;
                Dout : out std_logic_vector(8 downto 0);
                clk : in std_logic;
                reset : in std_logic;
                done : out std_logic
    );
    end component;
    signal done_exit : std_logic;
    signal D_exit : std_logic_vector(8 downto 0);
    signal DXval_exit : std_logic;

    begin

    LCD: LCDDispatcher port map (Dval => DXval_exit, Din => D_exit, Wrl => Wrl, Dout => DoutSLCDC, done => done_exit,clk => Clk, reset => Reset );
    SR : SerialReceiver port map (Reset => Reset, SDX => SDX, Clk => Clk, SCLK => SCLK, SS => SS, accept => done_exit, D => D_exit, DXval => DXval_exit);

    end structural;
```

Código VHDL do SerialReceiver:

```
    LIBRARY ieee;
```

```
USE ieee.std_logic_1164.all;


entity SerialReceiver is
	port(
		Reset : in std_logic;
		SDX : in std_logic;
		Clk : in std_logic;
		SCLK : in std_logic;
		SS : in std_logic;
		accept : in std_logic;
		D : out std_logic_vector(8 downto 0);
		DXval : out std_logic
);
end SerialReceiver;
architecture structural of SerialReceiver is
component ShiftRegisterSerialReceiver is
	port(
		reset : in std_logic;
		data: in std_logic;
		SCLK: in std_logic;
		E: in std_logic;
		D: out std_logic_vector(8 downto 0)
	);
end component;
component Counter is
	port(
		PL:in std_logic;
		CE:in std_logic;
		CLK:in std_logic;
		Data_in: in std_logic_vector(3 downto 0);
		RESET: in std_logic;
		TC: out std_logic;
		Q:out std_logic_vector(3 downto 0)
		);
end component;
component ParityCheck is
	port(
		data : in std_logic;
```

```
                Sclk : in std_logic;
                init : in std_logic;
                err : out std_logic
);
end component;
component SerialControl is
    port(
                enRx : in std_logic;
                accept : in std_logic;
                pFlag : in std_logic;
                dFlag : in std_logic;
                RXerror : in std_logic;
                wr : out std_logic;
                init : out std_logic;
                Reset : in std_logic;
                Clk : in std_logic;
                DXval : out std_logic
);
end component;
component Equal9 IS
port(
    Q: in STD_LOGIC_VECTOR(3 downto 0);
    TC: out STD_LOGIC
);
end component;
component Equal10 IS
port(
    Q: in STD_LOGIC_VECTOR(3 downto 0);
    TC: out STD_LOGIC
);
end component;

signal ParitCheck_err_exit : std_logic;
signal SerialControl_init_exit : std_logic;
signal SerialControl_wr_exit : std_logic;
signal CounterSerialReceiver_exit : std_logic_vector(3 downto 0);
signal dflag_9: std_logic;
signal pflag_10: std_logic;
```

```
begin

Shift: ShiftRegisterSerialReceiver port map (data=>SDX ,SCLK=>SCLK ,E => SerialControl_wr_exit,D=>D, reset => Reset);

Count: Counter port map (PL => '0', CE => '1', CLK => SCLK, Data_in => "0000", RESET => SerialControl_init_exit, Q => CounterSerialReceiver_exit);

Parity: ParityCheck port map (data => SDX, Sclk => SCLK, init => SerialControl_init_exit, err => ParitCheck_err_exit);

Controle: SerialControl port map (enRX => SS, accept => accept, pFlag => pflag_10 , dFlag => dflag_9, RXerror => ParitCheck_err_exit, wr => SerialControl_wr_exit, init => SerialControl_init_exit, DXval => DXval, Clk => Clk, Reset => Reset);

EQ1: Equal9 port map(Q => CounterSerialReceiver_exit,TC => dflag_9);

EQ2: Equal10 port map(Q => CounterSerialReceiver_exit,TC => pflag_10);

end structural;
```

Código VHDL do ShiftRegisterSerialReceiver:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity ShiftRegisterSerialReceiver is
    port(
        reset : in std_logic;
        data: in std_logic;
        SCLK: in std_logic;
        E: in std_logic;
        D: out std_logic_vector(8 downto 0)
    );
end ShiftRegisterSerialReceiver;
architecture structural of ShiftRegisterSerialReceiver is
component FFD IS
    PORT(
        CLK : in std_logic;
        RESET : in STD_LOGIC;
        SET : in std_logic;
```

```
            D : IN STD_LOGIC;
            EN : IN STD_LOGIC;
            Q : out std_logic
            );
end component;

signal saidaFFD1: std_logic;
signal saidaFFD2: std_logic;
signal saidaFFD3: std_logic;
signal saidaFFD4: std_logic;
signal saidaFFD5: std_logic;
signal saidaFFD6: std_logic;
signal saidaFFD7: std_logic;
signal saidaFFD8: std_logic;
signal saidaFFD9: std_logic;

begin

FFD1: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>data ,EN=>E ,Q=>saidaFFD1);
FFD2: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD1,EN=>E,Q=>saidaFFD2);
FFD3: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD2,EN=>E,Q=>saidaFFD3);
FFD4: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD3,EN=>E,Q=>saidaFFD4);
FFD5: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD4,EN=>E,Q=>saidaFFD5);
FFD6: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD5,EN=>E,Q=>saidaFFD6);
FFD7: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD6,EN=>E,Q=>saidaFFD7);
FFD8: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD7,EN=>E,Q=>saidaFFD8);
FFD9: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD8,EN=>E,Q=>saidaFFD9);

D(8)<= saidaFFD1;
D(7)<= saidaFFD2;
D(6)<= saidaFFD3;
D(5)<= saidaFFD4;
D(4)<= saidaFFD5;
D(3)<= saidaFFD6;
D(2)<= saidaFFD7;
D(1)<= saidaFFD8;
D(0)<= saidaFFD9;
```

```
end structural;
```

Código VHDL do ParityCheck:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity ParityCheck is
        port(
                data : in std_logic;
                Sclk : in std_logic;
                init : in std_logic;
                err : out std_logic
);
end ParityCheck;
architecture structural of ParityCheck is

component Counter is
port(
        PL:in std_logic;
        CE:in std_logic;
        CLK:in std_logic;
        Data_in: in std_logic_vector(3 downto 0);
        RESET: in std_logic;
        TC: out std_logic;
        Q:out std_logic_vector(3 downto 0)
        );
end component;

signal SaidaB : std_logic_vector(3 downto 0);

begin

CounterUp : Counter port map (PL => '0', CE => data, CLK => Sclk, Data_in => "0000", Q => SaidaB, RESET => init);
err <= SaidaB(0);

end structural;
```

Código VHDL do SerialControl:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity SerialControl is
        port(
                enRx : in std_logic;
                RESET : in std_logic;
                accept : in std_logic;
                pFlag : in std_logic;
                dFlag : in std_logic;
                RXerror : in std_logic;
                clk : in std_logic;
    wr : out std_logic;
                init: out std_logic;
                DXval : out std_logic
);
end SerialControl;

architecture behavioral of SerialControl is

type STATE_TYPE is (Intializing, Starting, Waiting, Acepted, ItsDone);

signal currentState, NextState: STATE_TYPE;

begin

currentState <= Intializing when RESET = '1' else NextState when rising_edge(clk);

GenerateNextState:
process (currentState, enRx, accept, pFlag, RXerror, dFlag)
        begin
                case currentState is
                        when Intializing  => if (enRx = '0') then
                                                                NextState <=
Starting;
                                                                else
```

```
                                                    NextState       <=
Intializing;
                                                end if;

                when Starting           => if (enRx = '1') then
                                                    NextState       <=
Intializing;
                                                elsif (dFlag = '1') then
                                                    NextState       <=
Waiting ;
                                                else
                                                    NextState       <=
Starting;
                                                end if;

                when Waiting            => if (enRx = '1') then
                                                    NextState       <=
Intializing;
                                                elsif(pFlag = '1') then
                                                    NextState       <=
Acepted;
                                                else
                                                    NextState       <=
Waiting;
                                                end if;

                when Acepted                    => if(RXerror = '0') then
                                                    NextState       <=
ItsDone;
                                                else
                                                    NextState       <=
Intializing;
                                                end if;

                when ItsDone            => if(accept = '1') then
                                                    NextState       <=
Intializing;
                                                else
```

```
                                                                NextState <=
ItsDone;
                                                                end if;


        end case;
end process;

wr <= '1' when(currentState = Starting) else '0';
init <= '1' when(currentState = Intializing) else '0';
DXval <= '1' when (currentState = ItsDone) else '0';

end behavioral;
```

Código VHDL do LCDDispacher:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity LCDDispatcher is
        port(
                Dval : in std_logic;
                reset : in std_logic;
                clk : in std_logic;
                Din : in std_logic_vector(8 downto 0);
                WrL : out std_logic;
                Dout : out std_logic_vector(8 downto 0);
                done : out std_logic
);
end LCDDispatcher;
architecture structural of LCDDispatcher is
        component LCDDispacherControl is
                port(
                        clk : in std_logic;
                        reset : in std_logic;
                        Dval : in std_logic;
                        Wrl : out std_logic;
                        done : out std_logic
);
end component;
begin
```

```
LCD: LCDDispacherControl port map (Dval => Dval, clk => clk , Wrl => Wrl, done => done , reset => reset);
Dout <= Din;
end structural;
```

Código VHDL do LCDDispacherControl:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity LCDDispacherControl is
	port(
		clk : in std_logic;
		reset : in std_logic;
		Dval : in std_logic;
		Wrl : out std_logic;
		done : out std_logic
);
end LCDDispacherControl;
architecture behavior of LCDDispacherControl is

type STATE_TYPE is (WaitingDval, ReceivingDval, DoneReceived);

signal currentState, NextState: STATE_TYPE;

begin

currentState <= WaitingDval when reset = '1' else NextState when rising_edge(clk);

GenerateNextState:
process(currentState, Dval)
	begin
		case currentState is
			when WaitingDval => if (Dval = '1') then
NextState <=
ReceivingDval;
else
NextState <=
WaitingDval;
end if;
```

```
                    when ReceivingDval => NextState <= DoneReceived;

                    when DoneReceived =>        if (Dval = '1') then
                                                        NextState <=
DoneReceived;
                                                else
                                                        NextState <= WaitingDval;
                                                end if;
                    end case;
                    end process;

Wrl <= '1' when (CurrentState = ReceivingDval)
else '0';

done <= '1' when (CurrentState = DoneReceived)
else '0';

end behavior;
```

B. Código *Kotlin – TUI*

```kotlin
import isel.leic.utils.Time

object TUI {

    var collumL0 = 16
    var collumL1 = 16

    fun escrever(str: String, left: Boolean, lines: Int, coll: Int) {
        if (left) {
            LCD.cursor(lines, coll)
            LCD.write(str)
            if (lines == 0) {
                collumL0 += 1
            } else {
                collumL1 += 1
            }
        } else {
            LCD.cursor(lines, coll)
            LCD.write(str)
```

```
            if (lines == 0) {
                collumL0 -= 1
            } else {
                collumL1 -= 1
            }
        }
    }

    fun align(line : Int, col : Int, c : Char){
        align(line,col,c.toString())
    }
    fun align(line : Int, col : Int, txt : String){
        LCD.cursor(line,col)
        LCD.write(txt)
    }

    fun alignLeft(line : Int, c : Char) {
        alignLeft(line,c.toString())
    }

    fun alignLeft(line : Int, txt : String) {
        if(txt.length <= 15) {
            LCD.cursor(line, 0)
            LCD.write(txt)
        }
    }     fun alignRigth(line : Int, str : String) {
        if(str.length < LCD.COLS){
            LCD.cursor(line, LCD.COLS - str.length)
            LCD.write(str)
        }
    }


    fun alignMiddle(line : Int, c : Char) {
        alignMiddle(line,c.toString())
    }

    fun alignMiddle(line : Int, txt : String) {

        val halfTxt = LCD.COLS - txt.length
        if (halfTxt >= 0) {
            LCD.cursor(line, halfTxt / 2)
            LCD.write(txt)
        }
    }

    fun forWaitKey(timeout: Long) : Char{
        return KBD.waitKey(timeout)
    }

    fun clrLCD(){
        LCD.clear()
    }

    fun cursor(line:Int,col:Int){
        LCD.cursor(line,col)
    }

    fun init(){
        KBD.init()
```

```
        LCD.init()
    }
}
```

B Código *Kotlin – LCD*

```
object LCD { // Escreve no LCD usando a interface a 4 bits.
    //    const val LINES = 2
    val COLS = 16// Dimensão do display.
    const val E_MASK = 0X20
    const val RS_MASK = 0x10
    const val CLK_MASK = 0X40
    const val DATA_MASK = 0X0F
    // Envia a sequência de iniciação para comunicação a 4 bits.
    fun init() {
        SerialEmitter.init()
        writeCMD(0b00110000)
        Time.sleep(15)
        writeCMD(0b00110000)
        Time.sleep(1)
        writeCMD(0b00110000)
        writeCMD(0b00111000)
        writeCMD(0b00001000)
        writeCMD(0b00000001)
        writeCMD(0b00000110)
        writeCMD(0b00001111)
        loadCustomChars()
    }
    fun writeByteSerial(rs: Boolean, data: Int) {
        val rS = if (rs) 1 else 0
        SerialEmitter.send(addr = SerialEmitter.Destination.LCD, data.shl(1) + rS,9)
    }
    // Escreve um byte de comando/dados no LCD
    fun writeByte(rs: Boolean, data: Int) {
        writeByteSerial(rs, data)
        //writeByteParallel(rs,data)
    }


    // Escreve um comando no LCD
    fun writeCMD(data: Int) {
        writeByte(false, data)
    }
    // Escreve um dado no LCD
    fun writeDATA(data: Int) {
        writeByte(true, data)
    }
    // Escreve um caráter na posição corrente.
    fun write(c: Char) {
        writeDATA(c.code)
    }
    // Escreve uma string na posição corrente.
    fun write(text: String) {
        for (c in text) {
            write(c)
        }
    }
    // Envia comando para posicionar cursor ('line':0..LINES-1 , 'column':0..COLS-1)
```

```
    fun cursor(line: Int, column: Int) {
        writeCMD((line * 0X40 + column) or 0x80)
    }
    // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
    fun clear() {
        writeCMD(0b00000001)
        cursor(0, 0)
    }
    fun invader() {
        writeCMD (0x40)
        writeDATA(0x1F)
        writeDATA(0x1F)
        writeDATA(0x15)
        writeDATA(0x1F)
        writeDATA(0x1F)
        writeDATA(0x11)
        writeDATA(0x11)
        writeDATA(0x00)
    }
    fun spaceShip() {
        writeCMD (0x48)
        writeDATA(0x1E)
        writeDATA(0x18)
        writeDATA(0x1C)
        writeDATA(0x1F)
        writeDATA(0x1C)
        writeDATA(0x18)
        writeDATA(0x1E)
        writeDATA(0x00)
    }
    fun loadCustomChars() {
        invader()
        spaceShip()
    }
}
```