SLCDC (*Roulette Game*)
Laboratório de Informática e Computadores 2024 / 2025 verão
Autores: Ricardo Martins nº52326 / Bernardo Silva nº52605 / Bernardo Esteves nº52948

1 SLCDC

O bloco SLCDC faz a ligação do módulo *Control* com o LCD no qual, recebe em série 5 bits de informação e 1 bit de paridade. Dentro destes 5 bits o 1 é o bit RS no qual, indica se a mensagem é de controlo ou de dados,os próximos 4 bits entregam os dados necessários ao LCD e o último bit contém a paridade ímpar. A implementação do SLCDC foi fundamentada através da Figura 1 no qual, é possível ver o diagrama de blocos resultante da nossa implementação na Figura 2.

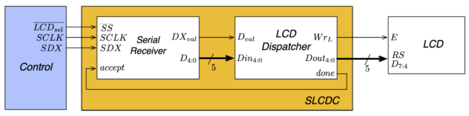

Figura 1: Diagrama de Blocos do *SLCDC*

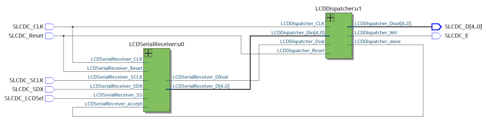

Figura 2: Diagrama de Blocos do módulo *SLCDC*

2 Serial Receiver

O *Serial Receiver* tem como função tratar do recebimento dos dados vindo do *Control*, detetanto o bit de paridade para confimar que não houve erros de transmissão, separar quando são os bits de dados ou de paridade e por fim enviar para o *LCD Dispatcher*.Assim sendo, o *Serial Receiver* contém 4 componentes sendo eles o *Serial Control*,*Parity Check*,*Shift Register* e *Counter*. É também feito no *Serial Receiver* a ligação das flags pFlag e dFlag que quando o *Counter* contar até 5 bits irá ligar a dFlag corrrespondente aos dados e quando contar até 6 bits irá ligar a pFlag corrrespondente á paridade. Os números no bloco *SRC* irão ser diferentes devido ao invés de receber 5 bits rercebe 8. Como base para a conceção do mesmo foi usado o diagrama de blocos correspondente á figura 3.O diagrama de blocos que representa o *Serial Receiver* está representado na 4.

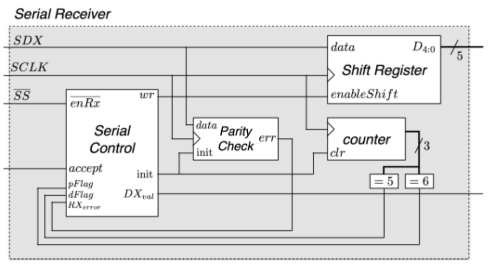

Figura 3: Diagrama de Blocos do *Serial Receiver*

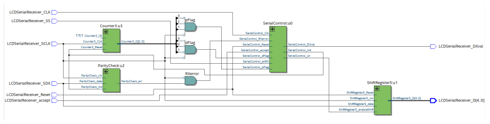

Figura 4: Diagrama de Blocos do módulo *Serial Receiver*

2.1 Serial Control

O *Serial Control* faz o controlo dos dados recebidos pelo *Control*, manuseando assim quando os outros componentes do *Serial Receiver* devem ou não ser ligados e transmitiindo se os dados são válidos ou não. O ASM-Chart produzido ao implementar o *Serial Control* está representado na Figura 5 e o diagrama de blocos correspondente na Figura 6.

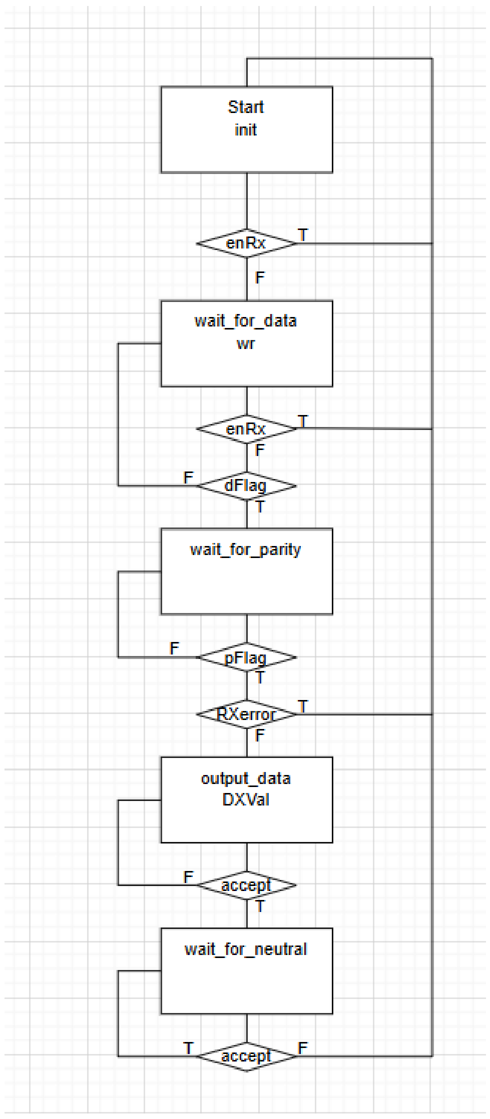

Figura 5: ASM Chart do *Serial Control*

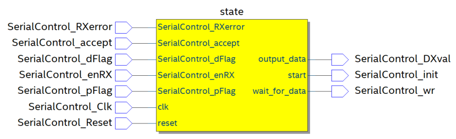

Figura 6: Diagrama de Blocos do módulo *Serial Control*

2.2 Parity Check

Parity Check é o componente responsável por validar a paridade ímpar no qual é iniciado pelo *Serial Control* e a sua saída será 0 quando o número de 1's for ímpar e será 1 quando o número de 1's for par. Assim o resultado da nossa implementação está disponível no diagrama de blocos da Figura 7.

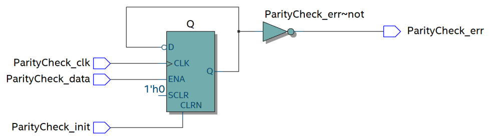

Figura 7: Diagrama de Blocos do módulo *Parity Check*

2.3 Shift Register

O bloco *Shift Register* também é ligado pelo *Serial Control* e tem como função enviar cada bit individualmente para o *LCD Dispatcher* transmitindo assim apenas os bits de dados. Com isto, o diagrama de blocos correspondente ao *Shift Register* está representado na Figura 8.

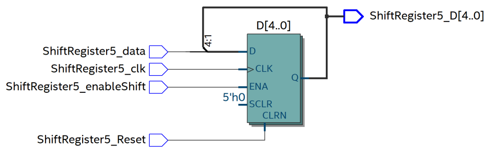

Figura 8: Diagrama de Blocos do módulo *Shift Register*

2.4 Counter

O componente *Counter* ao ser ativado pelo *Serial Control* tem a função de contar quantos bits são recebidos pelo *Serial Receiver* assim, o diagrama de bloco do *Counter* está demonstrado na Figura 9.

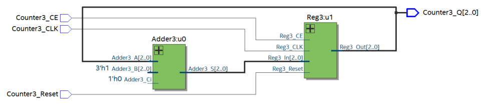

Figura 9: Diagrama de Blocos do módulo *Counter*

3 LCD Dispatcher

Por fim, temos o bloco *LCD Dispatcher* que é responsável por entregar as dados válidas do *Serial Receiver* ao *LCD* enviando assim, os bits de dados, RS e de paridade e o sinal Wrl para posteiormente ligar o LCD. Para a conceção do *LCD Dispatcher* foi preciso conceber uma máquina de estados podendo ser visualizada na 10. O ASM-Chart ficou com 12 estados de esperar devido ao clock ser 20 ns e o tempo necessário para os dados serem enviados para o LCD serem 236ns. O diagrama de blocos condizente com o *LCD Dispatcher* está representado na Figura 11.

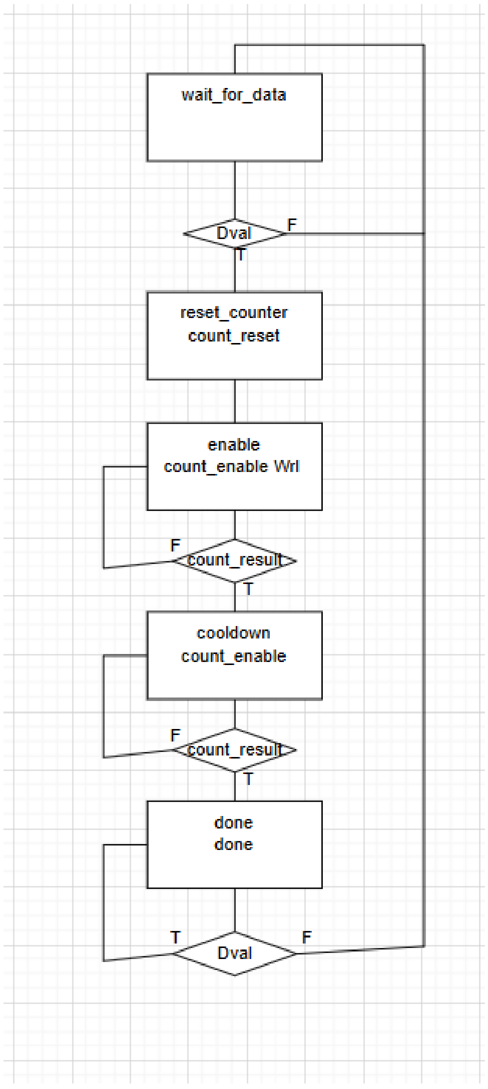

Figura 10: ASM-Chart do *LCD Dispatcher*

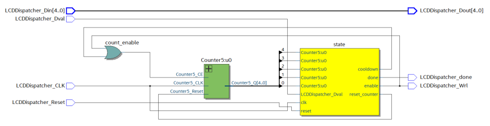

Figura 11: Diagrama de Blocos do módulo *LCD Dispatcher*

4 Conclusões

Em conclusão, o desenvolvimento do SLCDC apresentou sérias dificuldades e complexidades durante todo o seu desenvolvimento desde o ASM-Chart do bloco *Serial Control*, a implementação do *Parity Check* com a paridade ímpar e principalmente a conceção do *LCD Dispatcher* pois, o tempo necessário para transmitir os dados para o LCD causou sérios desafios e problemas.

A VHDL

A.1 Roleta

```
library ieee;
use ieee.std_logic_1164.all;

entity Roleta is
        port(

                -- input
                CLK: in std_logic;
                Reset: in std_logic;
                KBD_KeyLin: in std_logic_vector(3 downto 0);
        Coin: in std_logic;
        CoinID: in std_logic;
        Maintenance_in: in std_logic;

                -- output
                KBD_KeyCol: out std_logic_vector(3 downto 0);
                LCD_E: out std_logic;
                LCD_RS: out std_logic;
                LCD_D: out std_logic_vector(7 downto 4);
        RD_HEX0: out std_logic_vector(7 downto 0);
        RD_HEX1: out std_logic_vector(7 downto 0);
        RD_HEX2: out std_logic_vector(7 downto 0);
        RD_HEX3: out std_logic_vector(7 downto 0);
        RD_HEX4: out std_logic_vector(7 downto 0);
        RD_HEX5: out std_logic_vector(7 downto 0);
                Accept: out std_logic;

        debug_full: out std_logic;
        debug_empty: out std_logic

        );
end Roleta;

architecture logicFunction of Roleta is

        component UsbPort is
                port(

                        -- input
                        inputPort: in std_logic_vector(7 downto 0);

                        -- output
                        outputPort: out std_logic_vector(7 DOWNTO 0)

                );
        end component;

        component KeyboardReader is
        port(

            -- input
            KeyboardReader_Ack: in std_logic;
            KeyboardReader_Lin: in std_logic_vector(3 downto 0);
            KeyboardReader_CLK: in std_logic;
            KeyboardReader_Reset: in std_logic;

            -- output
            KeyboardReader_Col: out std_logic_vector(3 downto 0);
            KeyboardReader_Dval: out std_logic;
            KeyboardReader_Q: out std_logic_vector(3 downto 0);

            debug_full: out std_logic;
            debug_empty: out std_logic

        );
        end component;

    component SLCDC is
        port(

            -- input
            SLCDC_Reset: in std_logic;
            SLCDC_LCDSel: in std_logic;
            SLCDC_SCLK: in std_logic;
            SLCDC_CLK: in std_logic;
            SLCDC_SDX: in std_logic;

            -- output
            SLCDC_E: out std_logic;
            SLCDC_D: out std_logic_vector(4 downto 0)
        );
    end component;

    component SRC is
        port(

            -- input
            SRC_Reset: in std_logic;
            SRC_RDSel: in std_logic;
            SRC_SCLK: in std_logic;
            SRC_CLK: in std_logic;
            SRC_SDX: in std_logic;

            -- output
            SRC_set: out std_logic;
            SRC_D: out std_logic_vector(7 downto 0)

        );
    end component;

    component rouletteDisplay is
        port(

            -- input
            set: in std_logic;
            cmd: in std_logic_vector(2 downto 0);
            data: in std_logic_vector(4 downto 0);

            -- output
            HEX0: out std_logic_vector(7 downto 0);
            HEX1: out std_logic_vector(7 downto 0);
            HEX2: out std_logic_vector(7 downto 0);
            HEX3: out std_logic_vector(7 downto 0);
            HEX4: out std_logic_vector(7 downto 0);
            HEX5: out std_logic_vector(7 downto 0)

        );
    end component;

    component CoinAcceptor is
```

```
        port(

            -- input
            CoinAcceptor_Accept_in: in std_logic;
            CoinAcceptor_Coin_in: in std_logic;
            CoinAcceptor_CoinID_in: in std_logic;

            -- output
            CoinAcceptor_Accept_out: out std_logic;
            CoinAcceptor_Coin_out: out std_logic;
            CoinAcceptor_CoinID_out: out std_logic

        );
    end component;

    component Maintenance is
        port(

            -- input
            Maintenance_in: in std_logic;

            -- output
            Maintenance_out: out std_logic


        );
    end component;

    -- Input (Hardware → Software)
    signal usb_input: std_logic_vector(7 downto 0);

    -- Output (Software → Hardware)
    signal usb_output: std_logic_vector(7 downto 0);

    signal RD_set: std_logic;
    signal RD_cmd: std_logic_vector(2 downto 0);
    signal RD_data: std_logic_vector(4 downto 0);

    -- signal tmp: std_logic_vector(9 downto 0);

begin

    u0: UsbPort port map (
                inputPort => usb_input,
                outputPort => usb_output
        );

        u1: KeyboardReader port map (
                KeyboardReader_Ack => usb_output(7),
        KeyboardReader_Lin => KBD_KeyLin,
        KeyboardReader_CLK => CLK,
        KeyboardReader_Reset => Reset,
        KeyboardReader_Col => KBD_KeyCol,
        KeyboardReader_Dval => usb_input(4),
        KeyboardReader_Q => usb_input(3 downto 0),
        debug_empty => debug_empty,
        debug_full => debug_full
    );

    u2: SLCDC port map (
        SLCDC_Reset => Reset,
        SLCDC_LCDSel => usb_output(0),
        SLCDC_SCLK => usb_output(4),
        SLCDC_CLK => CLK,
        SLCDC_SDX => usb_output(3),
        SLCDC_E => LCD_E,
        SLCDC_D(0) => LCD_RS,
        SLCDC_D(4 downto 1) => LCD_D(7 downto 4)
    );

    u3: SRC port map (
        SRC_Reset => Reset,
        SRC_RDSel => usb_output(1),
        SRC_SCLK => usb_output(4),
        SRC_CLK => CLK,
        SRC_SDX => usb_output(3),
        SRC_set => RD_set,
        SRC_D(2 downto 0) => RD_cmd,
        SRC_D(7 downto 3) => RD_data
    );

    u4: rouletteDisplay port map (
        set => RD_set,
        cmd => RD_cmd,
        data => RD_data,
        HEX0 => RD_HEX0,
        HEX1 => RD_HEX1,
        HEX2 => RD_HEX2,
        HEX3 => RD_HEX3,
        HEX4 => RD_HEX4,
        HEX5 => RD_HEX5
    );

    u5: CoinAcceptor port map (
        CoinAcceptor_Accept_in => usb_output(6),
        CoinAcceptor_Coin_in => Coin,
        CoinAcceptor_CoinID_in => CoinID,
        CoinAcceptor_Accept_out => Accept,
        CoinAcceptor_Coin_out => usb_input(6),
        CoinAcceptor_CoinID_out => usb_input(5)
    );

    u6: Maintenance port map (
        Maintenance_in => Maintenance_in,
        Maintenance_out => usb_input(7)
    );

end logicFunction;
```

A.2 UsbPort

```
-- Copyright (C) 2020  Intel Corporation. All rights reserved.
-- Your use of Intel Corporation's design tools, logic functions
-- and other software and tools, and any partner logic
-- functions, and any output files from any of the foregoing
-- (including device programming or simulation files), and any
-- associated documentation or information are expressly subject
-- to the terms and conditions of the Intel Program License
-- Subscription Agreement, the Intel Quartus Prime License Agreement,
-- the Intel FPGA IP License Agreement, or other applicable license
-- agreement, including, without limitation, that your use is for
-- the sole purpose of programming logic devices manufactured by
-- Intel and sold by Intel or its authorized distributors.  Please
-- refer to the applicable agreement for further details, at
-- https://fpgasoftware.intel.com/eula.

-- PROGRAM                "Quartus Prime"
-- VERSION                "Version 20.1.1 Build 720 11/11/2020 SJ Lite Edition"
-- CREATED                "Tue Mar 01 09:42:31 2022"

library ieee;
use ieee.std_logic_1164.all;

library work;

entity UsbPort is
        port(
                inputPort: in std_logic_vector(7 downto 0);
                outputPort: out std_logic_vector(7 downto 0)
        );
end UsbPort;

architecture bdf_type of UsbPort is

    component sld_virtual_jtag
        GENERIC(
            lpm_type: string;
                        sld_auto_instance_index: string;
                        sld_instance_index: integer;
                        sld_ir_width: integer;
                        sld_sim_action: string;
                        sld_sim_n_scan: integer;
                        sld_sim_total_length: integer
                );
            PORT(
            tdo: in std_logic;
                    ir_out: in std_logic_vector(7 downto 0);
                    tck: out std_logic;
                    tdi: out std_logic;
                    virtual_state_cdr: out std_logic;
                    virtual_state_sdr: out std_logic;
                    virtual_state_e1dr: out std_logic;
                    virtual_state_pdr: out std_logic;
                    virtual_state_e2dr: out std_logic;
                    virtual_state_udr: out std_logic;
                    virtual_state_cir: out std_logic;
                    virtual_state_uir: out std_logic;
                    tms: out std_logic;
                    jtag_state_tlr: out std_logic;
                    jtag_state_rti: out std_logic;
                    jtag_state_sdrs: out std_logic;
                    jtag_state_cdr: out std_logic;
                    jtag_state_sdr: out std_logic;
                    jtag_state_e1dr: out std_logic;
                    jtag_state_pdr: out std_logic;
                    jtag_state_e2dr: out std_logic;
                    jtag_state_udr: out std_logic;
                    jtag_state_sirs: out std_logic;
                    jtag_state_cir: out std_logic;
                    jtag_state_sir: out std_logic;
                    jtag_state_e1ir: out std_logic;
                    jtag_state_pir: out std_logic;
                    jtag_state_e2ir: out std_logic;
                    jtag_state_uir: out std_logic;
                    ir_in: out std_logic_vector(7 downto 0)
            );
    end component;

begin

    b2v_inst: sld_virtual_jtag
        generic map (
            lpm_type => "sld_virtual_jtag",
            sld_auto_instance_index => "YES",
            sld_instance_index => 0,
            sld_ir_width => 8,
            sld_sim_action => "UNUSED",
            sld_sim_n_scan => 0,
            sld_sim_total_length => 0
        )
        port map (
            ir_out => inputPort,
            ir_in => outputPort
        );

end bdf_type;
```

A.3 SLCDC

```
library ieee;
use ieee.std_logic_1164.all;

entity SLCDC is
        port(

                -- input
        SLCDC_Reset: in std_logic;
                SLCDC_LCDSel: in std_logic;
                SLCDC_SCLK: in std_logic;
                SLCDC_CLK: in std_logic;
                SLCDC_SDX: in std_logic;

                -- output
                SLCDC_E: out std_logic;
                SLCDC_D: out std_logic_vector(4 downto 0)
        );
end SLCDC;

architecture logicFunction of SLCDC is

    component LCDSerialReceiver is
            port(

            -- input
            LCDSerialReceiver_Reset: in std_logic;
            LCDSerialReceiver_SS: in std_logic;
            LCDSerialReceiver_SCLK: in std_logic;
            LCDSerialReceiver_CLK: in std_logic;
            LCDSerialReceiver_SDX: in std_logic;
            LCDSerialReceiver_accept: in std_logic;

            -- output
            LCDSerialReceiver_DXval: out std_logic;
            LCDSerialReceiver_D: out std_logic_vector(4 downto 0)

        );
    end component;

    component LCDDispatcher is
        port(

            -- input
            LCDDispatcher_Reset: in std_logic;
            LCDDispatcher_Dval: in std_logic;
            LCDDispatcher_Din: in std_logic_vector(4 downto 0);
            LCDDispatcher_CLK: in std_logic;

            -- output
            LCDDispatcher_Wrl: out std_logic;
            LCDDispatcher_Dout: out std_logic_vector(4 downto 0);
            LCDDispatcher_done: out std_logic

        );
    end component;

    signal val: std_logic;
    signal D: std_logic_vector(4 downto 0);
    signal done_to_accept: std_logic;

begin
```

```
    u0: LCDSerialReceiver port map (
        LCDSerialReceiver_Reset => SLCDC_Reset,
        LCDSerialReceiver_SS => SLCDC_LCDSel,
        LCDSerialReceiver_SCLK => SLCDC_SCLK,
        LCDSerialReceiver_CLK => SLCDC_CLK,
        LCDSerialReceiver_SDX => SLCDC_SDX,
        LCDSerialReceiver_accept => done_to_accept,
        LCDSerialReceiver_DXval => val,
        LCDSerialReceiver_D => D
    );

    u1: LCDDispatcher port map (
        LCDDispatcher_Reset => SLCDC_Reset,
        LCDDispatcher_Dval => val,
        LCDDispatcher_Din => D,
        LCDDispatcher_CLK => SLCDC_CLK,
        LCDDispatcher_Wrl => SLCDC_E,
        LCDDispatcher_Dout => SLCDC_D,
        LCDDispatcher_done => done_to_accept
    );

end logicFunction;
```

A.4 LCDSerialReceiver

```
library ieee;
use ieee.std_logic_1164.all;

entity LCDSerialReceiver is
        port(

                -- input
                LCDSerialReceiver_SS: in std_logic;
                LCDSerialReceiver_SCLK: in std_logic;
                LCDSerialReceiver_CLK: in std_logic;
                LCDSerialReceiver_Reset: in std_logic;
                LCDSerialReceiver_SDX: in std_logic;
                LCDSerialReceiver_accept: in std_logic;

                -- output
                LCDSerialReceiver_DXval: out std_logic;
                LCDSerialReceiver_D: out std_logic_vector(4 downto 0)
        );
end LCDSerialReceiver;

architecture logicFunction of LCDSerialReceiver is

        component SerialControl is port (

                -- input
                SerialControl_Reset: in std_logic;
                SerialControl_enRX: in std_logic;
                SerialControl_accept: in std_logic;
                SerialControl_pFlag: in std_logic;
                SerialControl_dFlag: in std_logic;
                SerialControl_RXerror: in std_logic;
        SerialControl_Clk: in std_logic;

                -- output
                SerialControl_wr: out std_logic;
                SerialControl_init: out std_logic;
                SerialControl_DXval: out std_logic

        );
        end component;

        component ShiftRegister5 is
                port(

                        -- input
                        ShiftRegister5_data: in std_logic;
                        ShiftRegister5_clk: in std_logic;
                        ShiftRegister5_enableShift: in std_logic;
                        ShiftRegister5_Reset: in std_logic;

                        -- output
                        ShiftRegister5_D: out std_logic_vector(4 downto 0)

                );
        end component;

        component ParityCheck is
                port(

                        -- input
                        ParityCheck_data: in std_logic;
                        ParityCheck_init: in std_logic;
                        ParityCheck_clk: in std_logic;

                        -- output
                        ParityCheck_err: out std_logic

                );
        end component;

        component Counter3 is
                port(

                        -- input
                        Counter3_CE: in std_logic;
                        Counter3_CLK: in std_logic;
                        Counter3_Reset: in std_logic;

                        -- output
                        Counter3_Q: out std_logic_vector(2 downto 0)

                );
        end component;

        signal pFlag: std_logic;
        signal dFlag: std_logic;
        signal error: std_logic;
    signal RXerror: std_logic;
        signal wr: std_logic;
        signal init: std_logic;
        signal counter: std_logic_vector(2 downto 0);

begin

        u0: SerialControl port map (
                SerialControl_Reset => LCDSerialReceiver_Reset,
                SerialControl_enRX => LCDSerialReceiver_SS,
                SerialControl_accept => LCDSerialReceiver_Accept,
                SerialControl_pFlag => pFlag,
                SerialControl_dFlag => dFlag,
                SerialControl_RXerror => RXerror,
        SerialControl_Clk => LCDSerialReceiver_CLK,
                SerialControl_wr => wr,
                SerialControl_init => init,
                SerialControl_DXval => LCDSerialReceiver_DXval
        );

    RXerror <= error and not LCDSerialReceiver_SCLK;

        u1: ShiftRegister5 port map (
                ShiftRegister5_data => LCDSerialReceiver_SDX,
                ShiftRegister5_clk => LCDSerialReceiver_SCLK,
                ShiftRegister5_Reset => LCDSerialReceiver_Reset,
            ShiftRegister5_enableShift => wr,
                ShiftRegister5_D => LCDSerialReceiver_D
        );

        u2: ParityCheck port map (
                ParityCheck_data => LCDSerialReceiver_SDX,
                ParityCheck_init => init,
                ParityCheck_clk => LCDSerialReceiver_SCLK,
                ParityCheck_err => error
        );
        u3: Counter3 port map (
                Counter3_CE => '1',
                Counter3_CLK => LCDSerialReceiver_SCLK,
                Counter3_Reset => init,
                Counter3_Q => counter
        );

        dFlag <= counter(2) and not counter(1) and counter(0) and not LCDSerialReceiver_SCLK;
        pflag <= counter(2) and counter(1) and not counter(0) and not LCDSerialReceiver_SCLK;

end logicFunction;
```

A.5 SerialControl

```
library ieee;
use ieee.std_logic_1164.all;

entity SerialControl is
        port(

                -- input
        SerialControl_Reset: in std_logic;
                SerialControl_enRX: in std_logic;
                SerialControl_accept: in std_logic;
                SerialControl_pFlag: in std_logic;
                SerialControl_dFlag: in std_logic;
                SerialControl_RXerror: in std_logic;
                SerialControl_Clk: in std_logic;

                -- output
                SerialControl_wr: out std_logic;
                SerialControl_init: out std_logic;
                SerialControl_DXval: out std_logic

        );
end SerialControl;

architecture logicFunction of SerialControl is

    type Tstate is (start, wait_for_data, wait_for_parity, output_data, wait_for_neutral);

    signal state: Tstate;
    signal next_state: Tstate;

begin

    state <= start when SerialControl_Reset = '1' else next_state when rising_edge(SerialControl_Clk);

    process(state, SerialControl_enRX, SerialControl_accept, SerialControl_pFlag, SerialControl_dFlag, SerialControl_RXerror

        case state is

            when start =>
                if SerialControl_enRX = '0' then
                    next_state <= wait_for_data;
                else
                    next_state <= start;
                end if;

            when wait_for_data =>
                if SerialControl_enRX = '0' then
                    if SerialControl_dFlag = '1' then
                        next_state <= wait_for_parity;
                    else
                        next_state <= wait_for_data;
                    end if;
                else
                    next_state <= start;
                end if;

            when wait_for_parity =>
                --if SerialControl_enRX = '1' then
                    if SerialControl_pFlag = '1' then
                        if SerialControl_RXerror = '0' then
                            next_state <= output_data;
                        else
                            next_state <= start;
                        end if;
                    else
                        next_state <= wait_for_parity;
                    end if;
                --else
                    --next_state <= wait_for_parity;
                --end if;

            when output_data =>
                if SerialControl_accept = '1' then
                    next_state <= wait_for_neutral;
                else
                    next_state <= output_data;
                end if;

            when wait_for_neutral =>
                if SerialControl_accept = '0' then
                    next_state <= start;
                else
                    next_state <= wait_for_neutral;
                end if;

        end case;
    end process;

    SerialControl_DXval <= '1' when (state = output_data) else '0';
    SerialControl_wr <= '1' when (state = wait_for_data) else '0';
    SerialControl_init <= '1' when (state = start) else '0';

end logicFunction;
```

A.6 ParityCheck

```
library ieee;
use ieee.std_logic_1164.all;

entity ParityCheck is
        port(

                -- input
                ParityCheck_data: in std_logic;
                ParityCheck_init: in std_logic;
                ParityCheck_clk: in std_logic;

                -- output
                ParityCheck_err: out std_logic

        );
end ParityCheck;

architecture logicFunction of ParityCheck is

        signal Q: std_logic;
        signal D: std_logic;

begin

        D <= ParityCheck_data xor Q;
        Q <= '0' when ParityCheck_init = '1' else D when rising_edge(ParityCheck_clk);
        ParityCheck_err <= not Q;

end logicFunction;
```

A.7 ShiftRegister5

```
library ieee;
use ieee.std_logic_1164.all;

entity ShiftRegister5 is
        port(

                -- input
                ShiftRegister5_Reset: in std_logic;
                ShiftRegister5_data: in std_logic;
                ShiftRegister5_clk: in std_logic;
                ShiftRegister5_enableShift: in std_logic;

                -- output
                ShiftRegister5_D: out std_logic_vector(4 downto 0)

        );
end ShiftRegister5;

architecture logicFunction of ShiftRegister5 is

    signal D: std_logic_vector(4 downto 0);

begin

    D(4) <= '0' when ShiftRegister5_Reset = '1' else ShiftRegister5_data when rising_edge(ShiftRegister5_clk) and ShiftRegis
    D(3) <= '0' when ShiftRegister5_Reset = '1' else D(4) when rising_edge(ShiftRegister5_clk) and ShiftRegister5_enableShif
    D(2) <= '0' when ShiftRegister5_Reset = '1' else D(3) when rising_edge(ShiftRegister5_clk) and ShiftRegister5_enableShif
    D(1) <= '0' when ShiftRegister5_Reset = '1' else D(2) when rising_edge(ShiftRegister5_clk) and ShiftRegister5_enableShif
    D(0) <= '0' when ShiftRegister5_Reset = '1' else D(1) when rising_edge(ShiftRegister5_clk) and ShiftRegister5_enableShif

    ShiftRegister5_D <= D;

end logicFunction;
```

A.8 Counter3

```
library ieee;
use ieee.std_logic_1164.all;

entity Counter3 is
        port(

                -- input
                Counter3_CE: in std_logic;
                Counter3_CLK: in std_logic;
                Counter3_Reset: in std_logic;

                -- output
                Counter3_Q: out std_logic_vector(2 downto 0)

        );
end Counter3;

architecture logicFunction of Counter3 is

        component Adder3 is
                port(

                          -- input
                          Adder3_Ci: in std_logic;
                          Adder3_A: in std_logic_vector(2 downto 0);
                          Adder3_B: in std_logic_vector(2 downto 0);

                          -- output
                          Adder3_Co: out std_logic;
                          Adder3_S: out std_logic_vector(2 downto 0)

                );
        end component;

        component Reg3 is
                port(

                        -- input
                        Reg3_In: in std_logic_vector(2 downto 0);
                        Reg3_CE: in std_logic;
                        Reg3_CLK: in std_logic;
                        Reg3_Reset: in std_logic;

                        -- output
                        Reg3_Out: out std_logic_vector(2 downto 0)

                );
        end component;

        signal Q: std_logic_vector(2 downto 0);
        signal Adder3_S_to_Reg3_In: std_logic_vector(2 downto 0);

begin

        u0: Adder3 port map (
                Adder3_Ci => '0',
                Adder3_A => Q,
                Adder3_B => "001",
                Adder3_S => Adder3_S_to_Reg3_In
        );
        u1: Reg3 port map (
                Reg3_In => Adder3_S_to_Reg3_In,
                Reg3_CE => Counter3_CE,
                Reg3_CLK => Counter3_CLK,
                Reg3_Reset => Counter3_Reset,
                Reg3_Out => Q
        );

        Counter3_Q <= Q;

end logicFunction;
```

A.9 Adder3

```
library ieee;
use ieee.std_logic_1164.all;

entity Adder3 is
    port(

        -- input
        Adder3_Ci: in std_logic;
        Adder3_A: in std_logic_vector(2 downto 0);
        Adder3_B: in std_logic_vector(2 downto 0);

        -- output
        Adder3_Co: out std_logic;
        Adder3_S: out std_logic_vector(2 downto 0)

    );
end Adder3;

architecture logicFunction of Adder3 is

    component FullAdder is
        port(

            -- input
            FullAdder_A: in std_logic;
            FullAdder_B: in std_logic;
            FullAdder_Ci: in std_logic;

            -- output
            FullAdder_S: out std_logic;
            FullAdder_Co: out std_logic

        );
    end component;

    signal c0_to_c1: std_logic;
    signal c1_to_c2: std_logic;

begin

    c0: FullAdder port map (
        FullAdder_A => Adder3_A(0),
        FullAdder_B => Adder3_B(0),
        FullAdder_Ci => Adder3_Ci,
        FullAdder_S => Adder3_S(0),
        FullAdder_Co => c0_to_c1
    );

    c1: FullAdder port map (
        FullAdder_A => Adder3_A(1),
        FullAdder_B => Adder3_B(1),
        FullAdder_Ci => c0_to_c1,
        FullAdder_S => Adder3_S(1),
        FullAdder_Co => c1_to_c2
    );

    c2: FullAdder port map (
        FullAdder_A => Adder3_A(2),
        FullAdder_B => Adder3_B(2),
        FullAdder_Ci => c1_to_c2,
        FullAdder_S => Adder3_S(2),
        FullAdder_Co => Adder3_Co
    );

end logicFunction;
```

A.10 Reg3

```
library ieee;
use ieee.std_logic_1164.all;

entity Reg3 is
        port(

                -- input
                Reg3_In: in std_logic_vector(2 downto 0);
                Reg3_CE: in std_logic;
                Reg3_CLK: in std_logic;
                Reg3_Reset: in std_logic;

                -- output
                Reg3_Out: out std_logic_vector(2 downto 0)

        );
end Reg3;

architecture logicFunction of Reg3 is
begin

        Reg3_Out(2) <= '0' when Reg3_Reset = '1' else Reg3_In(2) when rising_edge(Reg3_CLK) and Reg3_CE = '1';
        Reg3_Out(1) <= '0' when Reg3_Reset = '1' else Reg3_In(1) when rising_edge(Reg3_CLK) and Reg3_CE = '1';
        Reg3_Out(0) <= '0' when Reg3_Reset = '1' else Reg3_In(0) when rising_edge(Reg3_CLK) and Reg3_CE = '1';

end logicFunction;
```

A.11 FullAdder

```
library ieee;
use ieee.std_logic_1164.all;

entity FullAdder is
    port(

        -- input
        FullAdder_A: in std_logic;
        FullAdder_B: in std_logic;
        FullAdder_Ci: in std_logic;

        -- output
        FullAdder_S: out std_logic;
        FullAdder_Co: out std_logic

    );
end FullAdder;

architecture logicFunction of FullAdder is
begin

    FullAdder_S <= FullAdder_A xor FullAdder_B xor FullAdder_Ci;
    FullAdder_Co <= (FullAdder_A and FullAdder_B) or (FullAdder_Ci and (FullAdder_A xor FullAdder_B));

end logicFunction;
```

A.12 LCDDispatcher

```
library ieee;
use ieee.std_logic_1164.all;

entity LCDDispatcher is
        port(

                -- input
        LCDDispatcher_Reset: in std_logic;
                LCDDispatcher_Dval: in std_logic;
        LCDDispatcher_Din: in std_logic_vector(4 downto 0);
        LCDDispatcher_CLK: in std_logic;

                -- output
                LCDDispatcher_Wrl: out std_logic;
                LCDDispatcher_Dout: out std_logic_vector(4 downto 0);
        LCDDispatcher_done: out std_logic

        );
end LCDDispatcher;

architecture logicFunction of LCDDispatcher is

    component Counter5 is
        port(

            -- input
            Counter5_CE: in std_logic;
            Counter5_CLK: in std_logic;
            Counter5_Reset: in std_logic;

            -- output
            Counter5_Q: out std_logic_vector(4 downto 0)

        );
    end component;

    type Tstate is (wait_for_data, reset_counter, enable, cooldown, done);

    signal state: Tstate;
    signal next_state: Tstate;

    signal count_enable: std_logic;
    signal count_reset: std_logic;
    signal count_result: std_logic_vector(4 downto 0);

begin

    state <= wait_for_data when LCDDispatcher_Reset = '1' else next_state when rising_edge(LCDDispatcher_CLK);

    u0: Counter5 port map (
                Counter5_CE => count_enable,
        Counter5_CLK => LCDDispatcher_CLK,
        Counter5_Reset => count_reset,
        Counter5_Q => count_result
        );

    process(state, LCDDispatcher_Dval, LCDDispatcher_Din, count_result) begin

        case state is

            when wait_for_data =>
```

```
                if LCDDispatcher_Dval = '1' then
                    next_state <= reset_counter;
                else
                    next_state <= wait_for_data;
                end if;

            when reset_counter =>
                next_state <= enable;

            when enable =>
                -- tw = 230ns  clk = 50mhz  => min 12 ciclos (0b01100)
                if count_result = "01111" then
                    next_state <= cooldown;
                else
                    next_state <= enable;
                end if;

            when cooldown =>
                -- tc = 500ns  clk = 50mhz  => min 25 ciclos (0b11001)
                -- if count_result(4) and count_result(3) and not count_result(2) and not count_result(1) and count_result(
                if count_result = "11111" then
                    next_state <= done;
                else
                    next_state <= cooldown;
                end if;

            when done =>
                if LCDDispatcher_Dval = '0' then
                    next_state <= wait_for_data;
                else
                    next_state <= done;
                end if;

        end case;
    end process;

    LCDDispatcher_Dout <= LCDDispatcher_Din;
    count_enable <= '1' when state = enable or state = cooldown else '0';
    count_reset <= '1' when state = reset_counter else '0';
    LCDDispatcher_Wrl <= '1' when state = enable else '0';
    LCDDispatcher_done <= '1' when state = done else '0';

end logicFunction;
```

B *Kotlin*

B.1 HAL

Algoritmo 1: HAL - Hardware Abstract Layer

```kotlin
import isel.leic.UsbPort

object HAL {

    /**  ltimo  _output_ enviado. */
    var lastOutput = 0

    /** Inicializa o _output_ do UsbPort. */
    fun init() = UsbPort.write(0)

    /** Verifica se os bits da `mask` est o ativos no _input_ do UsbPort. */
    fun isBit(mask: Int) = readBits(mask) == mask

    /** Retorna os bits no _input_ do UsbPort filtrados por `mask`. */
    fun readBits(mask: Int) = UsbPort.read() and mask

    /** Escreve no _output_ do UsbPort `value` filtrado por `mask`. */
    fun writeBits(mask: Int, value: Int) = write((value and mask) or (lastOutput and mask.inv()))

    /** Ativa no _output_ do UsbPort os bits da `mask`. */
    fun setBits(mask: Int) = write(lastOutput or mask)

    /** Desativa no _output_ do UsbPort os bits da `mask`. */
    fun clearBits(mask: Int) = write(lastOutput and mask.inv())

    /** Escreve no _output_ do UsbPort e guarda o que escreveu.  */
    private fun write(output: Int){
        UsbPort.write(output)
        lastOutput = output
    }

}

fun main(){

    HAL.init()

    while(true){
        println(HAL.readBits(0xF).toString(2))
        readln()
    }

}
```

B.2 SerialEmitter

Algoritmo 2: SerialEmitter

```
import isel.leic.utils.Time

/** Envia tramas para os diferentes m dulos Serial Receiver */
object SerialEmitter {

    enum class Destination { LCD, ROULETTE }

    // [ NOTA IMPORTANTE! ]
    // O bits SEL t m que ser negados
    // Ou seja, para selecionar o LCD, deve-se fazer HAL.clearBits(LCD_SEL)

    private const val LCD_SEL = 0b00000001
    private const val RD_SEL  = 0b00000010

    private const val SDX     = 0b00001000
    private const val SCLK    = 0b00010000

    /** Inicializa o SerialEmitter */
    fun init(){
        HAL.writeBits(
            LCD_SEL or RD_SEL or SDX or SCLK,
            LCD_SEL or RD_SEL // SDX e SCLK devem estar a zero
        )
    }

    /**
     * Envia uma trama para o _Serial Receiver_.
     * @property addr Destino
     * @property data Bits de dados
     * @property size N  de bits a enviar
     */
    fun send(addr: Destination, data: Int, size: Int){

        // Ativar destino
        HAL.clearBits(when(addr){
            Destination.LCD -> LCD_SEL
            Destination.ROULETTE -> RD_SEL
        })

        // Enviar dados
        repeat(size){ i ->

            // Preparar SDX
            when((data shr i) and 1){
                1 -> HAL.setBits(SDX)
                0 -> HAL.clearBits(SDX)
            }

            // CLK
            HAL.setBits(SCLK)
            HAL.clearBits(SCLK)

        }

        // Enviar bit de paridade
        when(data.countOneBits() % 2){
            1 -> HAL.clearBits(SDX)
            0 -> HAL.setBits(SDX)
        }

        // CLK
        HAL.setBits(SCLK)
        HAL.clearBits(SCLK)

        // Desativar destino
        HAL.setBits(LCD_SEL or RD_SEL)
    }

}
```

```
fun main(){

    SerialEmitter.init()

    while(true){
        SerialEmitter.send(SerialEmitter.Destination.ROULETTE, readln().toInt(2), 8)
    }

//    println("11100001 - Off")
//    SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b11100001, 8)
//    Time.sleep(1000)
//
//    println("11100000 - On")
//    SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b11100000, 8)
//    Time.sleep(1000)
//
//    // clear
//
//    println("00011111 - clear digit 0")
//    SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b00011111, 8)
//    Time.sleep(1000)
//
//    println("00111111 - clear digit 1")
//    SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b00111111, 8)
//    Time.sleep(1000)
//
//    println("01011111 - clear digit 2")
//    SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b01011111, 8)
//    Time.sleep(1000)
//
//    println("01111111 - clear digit 3")
//    SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b01111111, 8)
//    Time.sleep(1000)
//
//    println("10011111 - clear digit 4")
//    SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b10011111, 8)
//    Time.sleep(1000)
//
//    println("10111111 - clear digit 5")
//    SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b10111111, 8)
//    Time.sleep(1000)
//
//    // end clear
//
//    println("00000000 - update digit")
//    SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b00000000, 8)
//    Time.sleep(1000)
//
//    println("11000000 - update")
//    SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b11000000, 8)
//    Time.sleep(1000)

    // O DXval n o parece ser ativo na simula  o
    // Mas isso n o significa que n o est  a funcionar
}
```

B.3 LCD

Algoritmo 3: LCD

```
import isel.leic.utils.Time

/** Escreve no LCD usando a interface a 4 bits. */
object LCD {

    // Dimens o do display.
    const val LINES = 2
    const val COLS = 16

    /** Define se a interface   S rie ou Paralela */
    private const val SERIAL_INTERFACE = true

    private const val RS_MASK = 0b010000
    private const val E_MASK =  0b100000

    private const val CGRAM_MASK = 0b01000000

    /** Escreve um nibble de comando/dados no LCD em paralelo. */
    private fun writeNibbleParallel(rs: Boolean, data: Int){

        if(rs) HAL.setBits(RS_MASK)
        else HAL.clearBits(RS_MASK)

       HAL.setBits(E_MASK)
       HAL.writeBits(0xF, data)
       HAL.clearBits(E_MASK)
    }

    /** Escreve um nibble de comando/dados no LCD em s rie. */
    private fun writeNibbleSerial(rs: Boolean, data: Int){

        // Adicionar bit de RS
        var data = data shl 1
        if(rs) data = data or 1

        SerialEmitter.send(SerialEmitter.Destination.LCD, data, 5)
    }

    /** Escreve um nibble de comando/dados no LCD. */
    private fun writeNibble(rs: Boolean, data: Int){
        require(data in 0x0..0xF){ "0x${data.toString(16)}␣is␣not␣a␣valid␣nibble" }
        return when(SERIAL_INTERFACE){
            true -> writeNibbleSerial(rs, data)
            false -> writeNibbleParallel(rs, data)
        }
    }

    /** Escreve um byte de comando/dados no LCD. */
    private fun writeByte(rs: Boolean, data: Int) {
        require(data in 0x00..0xFF){ "0x${data.toString(16)}␣is␣not␣a␣valid␣byte" }
        writeNibble(rs, data shr 4)
        writeNibble(rs, data and 0xF)
    }

    /** Escreve um comando no LCD. */
    private fun writeCMD(data: Int) = writeByte(false, data)

    /** Escreve dados no LCD. */
    private fun writeDATA(data: Int) = writeByte(true, data)

    /** Envia a sequ ncia de inicia  o para comunica  o a 4 bits. */
    fun init(){

        Time.sleep(20)
        writeNibble(false, 0b0011)
        Time.sleep(5)
        writeNibble(false, 0b0011)
        Time.sleep(1)
        writeNibble(false, 0b0011)
        Time.sleep(1)
        writeNibble(false, 0b0010)

        writeCMD(0b00101000) // function set - DL=0 N=1 F=0
        Time.sleep(1)
        writeCMD(0b00001000) // display off - D=0 C=0 B=0
        Time.sleep(1)
        clear() // clear display
        Time.sleep(5)
        writeCMD(0b00000110) // entry mode set - I/D=1 S=0
        Time.sleep(1)
        writeCMD(0b00001100) // display on - D=1 C=0 B=0

    }

    /** Escreve uma string na posi   o corrente. */
    fun write(text: String){
        for(c in text) write(c)
    }

    /** Escreve um car cter na posi   o corrente. */
    fun write(char: Char) = writeDATA(char.code)

    /** Envia um comando para posicionar o cursor. */
    fun cursor(line: Int, column: Int){
        require(line in 0..LINES - 1) { "Line␣must␣be␣between␣0␣and␣${LINES␣-␣1}" }
        require(column in 0..COLS - 1) { "Line␣must␣be␣between␣0␣and␣${COLS␣-␣1}" }
        writeCMD(0b10000000 + line * 0x40 + column)
    }

    /** Envia comando para limpar o ecr   e posicionar o cursor na posi   o inicial. */
    fun clear() = writeCMD(0b00000001)

    /**
     * Regista um car ter personalizado.
     * @property char Car ter a associar.
     * @property picture Desenho do car ter 5 8 .
     */
    fun loadChar(char: Char, picture: List<String>){
        require(char.code in 0b000..0b111){ "Character␣is␣more␣than␣3␣bits" }
        repeat(8){ i ->
            writeCMD(CGRAM_MASK or (char.code shl 3) or i)
            val code = picture[i].map{ if(it == ' ') '0' else '1' }.joinToString("").toInt(2)
            writeDATA(code)
        }
        writeCMD(0b10)
    }

}

fun main(){

    HAL.init()
    SerialEmitter.init()
    LCD.init()

    LCD.loadChar(0.toChar(), listOf(
        "␣␣␣␣␣",
        "␣␣␣␣␣",
        "␣#␣#␣",
        "␣␣␣␣␣",
        "#␣␣␣#",
        "␣###␣",
        "␣␣␣␣␣",
        "␣␣␣␣␣"
    ))

    LCD.write("Hello!␣${0.toChar()}")
    LCD.cursor(1, 0)
    LCD.write("0123456789")

    Time.sleep(3000)

    LCD.clear()
    LCD.write("Goodbye...")

    Time.sleep(3000)

    LCD.clear()

}
```

B.4 TUI

Algoritmo 4: TUI

```
import isel.leic.utils.Time
import util.Alignment

object TUI {

    /** 9+ - Para indicar que h  mais de 9 cr ditos */
    const val NINE_PLUS = 0.toChar()

    /** Emoticon feliz - Para vit rias */
    const val HAPPY = 1.toChar()

    /** Emoticon triste - Para perdas */
    const val SAD = 2.toChar()

    /** S mbolo de Manuten  o */
    const val MAINTENANCE = 3.toChar()

    /** Seta para a direita */
    const val ARROW = 4.toChar()

    /** Inicializa o TUI. */
    fun init(){

        LCD.clear()

        LCD.loadChar(NINE_PLUS, listOf(
            "###␣␣",
            "#␣#␣␣",
            "###␣␣",
            "␣␣#␣␣",
            "###␣␣",
            "␣␣␣#␣",
            "␣␣###",
            "␣␣␣#␣"
        ))

        LCD.loadChar(HAPPY, listOf(
            "␣␣␣␣␣",
            "␣␣␣␣␣",
            "␣#␣#␣",
            "␣␣␣␣␣",
            "#␣␣␣#",
            "␣###␣",
            "␣␣␣␣␣",
            "␣␣␣␣␣"
        ))

        LCD.loadChar(SAD, listOf(
            "␣␣␣␣␣",
            "␣␣␣␣␣",
            "␣#␣#␣",
            "␣␣␣␣␣",
            "␣###␣",
            "#␣␣␣#",
            "␣␣␣␣␣",
            "␣␣␣␣␣"
        ))

        LCD.loadChar(MAINTENANCE, listOf(
            "#####",
            "␣␣␣␣␣",
            "#␣␣␣#",
            "##␣##",
            "#␣#␣#",
            "#␣␣␣#",
            "␣␣␣␣␣",
            "#####"
        ))
        LCD.loadChar(ARROW, listOf(
            "␣␣␣␣␣",
            "␣␣#␣␣",
            "␣␣␣#␣",
            "#####",
            "␣␣␣#␣",
            "␣␣#␣␣",
            "␣␣␣␣␣",
            "␣␣␣␣␣"
        ))

    }

    /* ---------- LCD ---------- */

    /**
     * Escreve texto.
     * @property align Alinhamento.
     * @property range Limites do texto.
     */
    fun write(text: String, line: Int, align: Alignment = Alignment.CENTER, range: IntRange = 0..LCD.COLS){
        require(line in 0 until LCD.LINES) { "Invalid␣line." }

        val rangeSize = range.last - range.start
        val startCol = range.start + when(align){
            Alignment.LEFT -> 0
            Alignment.CENTER -> (rangeSize - text.length) / 2
            Alignment.RIGHT -> rangeSize - text.length
        }

        updateLine(line, text, startCol)

    }

    /**
     * Atualiza a frame de anima  o de um texto rotativo.
     *
     *   til  para mostrar texto maior do que a largura do LCD.
     */
    fun scrollText(text: String, line: Int, speed: Int = 250, range: IntRange = 0..LCD.COLS){
        require(line in 0 until LCD.LINES) { "Invalid␣line." }

        val time = Time.getTimeInMillis() / speed
        val rangeSize = range.last - range.start

        val text = "$text␣" // Adicionar espa o como margem
        var res = text.substring((time % text.length).toInt()) // Cortar texto
        while(res.length < rangeSize) res += text // Duplicar texto para ocupar espa o inteiro
        res = res.substring(0 until rangeSize) // Manter texto dentro do range

        updateLine(line, res, range.start)

    }

    fun clear(){
        line0 = List(LCD.COLS){ "␣" }.joinToString("")
        line1 = List(LCD.COLS){ "␣" }.joinToString("")
        LCD.clear()
    }

    private var line0 = List(LCD.COLS){ "␣" }.joinToString("")
    private var line1 = List(LCD.COLS){ "␣" }.joinToString("")

    /** Atualiza as linhas do LCD evitando comandos redundantes. */
    private fun updateLine(line: Int, text: String, start: Int = 0){

        // Obter  ltimo  estado da linha
        val lastLine = when(line){
            0 -> line0
            1 -> line1
            else -> throw IllegalArgumentException("Invalid␣line.")
        }

        // Sobrescrever linha
```

```
        val prefix = if(start > 0) lastLine.substring(0 until start) else ""
        val suffix = if(start + text.length <= LCD.COLS) lastLine.substring(start + text.length) else ""
        val currLine = (prefix + text + suffix).substring(0 until LCD.COLS)

        // Ignorar se n o existir altera  es
        if(lastLine == currLine) return

        // Escrever caract res nos locais necess rios
        var col: Int? = null
        repeat(LCD.COLS){ i ->
            if(lastLine[i] == currLine[i]) return@repeat

            if(col != i) LCD.cursor(line, i)
            LCD.write(currLine[i])
            col = (col ?: 0) + 1

        }

        // Atualizar  ltimo  estado da linha
        when(line){
            0 -> line0 = currLine
            1 -> line1 = currLine
        }

    }

    /* ---------- Keyboard ---------- */

    /** Retorna de imediato a tecla premida ou `NONE` se n o h  tecla premida. */
    fun getKey() = KBD.getKey()

}

fun main(){

    HAL.init()
    SerialEmitter.init()
    LCD.init()
    TUI.init()

    while(true){

        TUI.scrollText("If␣you␣are␣seeing␣this␣text␣scrolling,␣it␣means␣that␣the␣TUI␣is␣working!", 0)

        val key = TUI.getKey()
        if(key != 0.toChar()) TUI.write(key.toString(), 1)

    }

}
```