Autores: Ricardo Martins nº52326 / Bernardo Silva nº52605 / Bernardo Esteves nº52948

1 SRC

O bloco SRC faz a ligação do módulo *Control* com o Roulette Display no qual, recebe em série 8 bits de informação e 1 bit de paridade. Dentro destes 8 bits os 3 primeiros bits indicam o comando a realizar na roleta e os próximos 5 bits identificam o campo de dados. A implementação do SRC foi fundamentada através da Figura 1 no qual, é possível ver o diagrama de blocos resultante da nossa implementação na Figura 2.

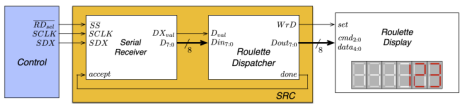

Figura 1: Diagrama de Blocos do *Buffer SRC*

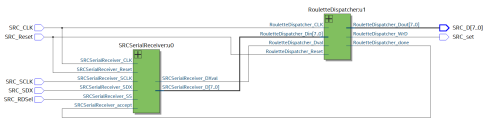

Figura 2: Diagrama de Blocos do módulo *SRC*

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 *Roulette 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 8 bits irá ligar a dFlag corrrespondente aos dados e quando contar até 9 bits irá ligar a pFlag corrrespondente á paridade. 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.

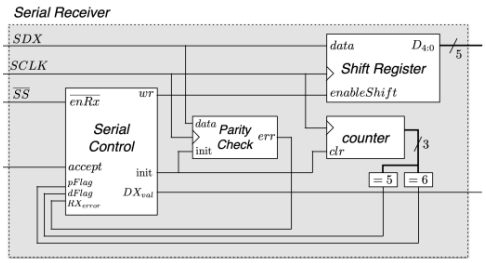

Figura 3: Diagrama de Blocos do *SLCDC*

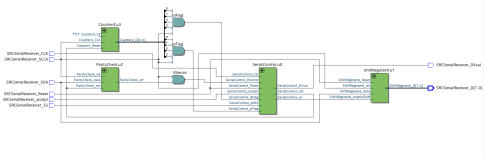

Figura 4: Diagrama de Blocos do módulo *SRC Receiver*

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.

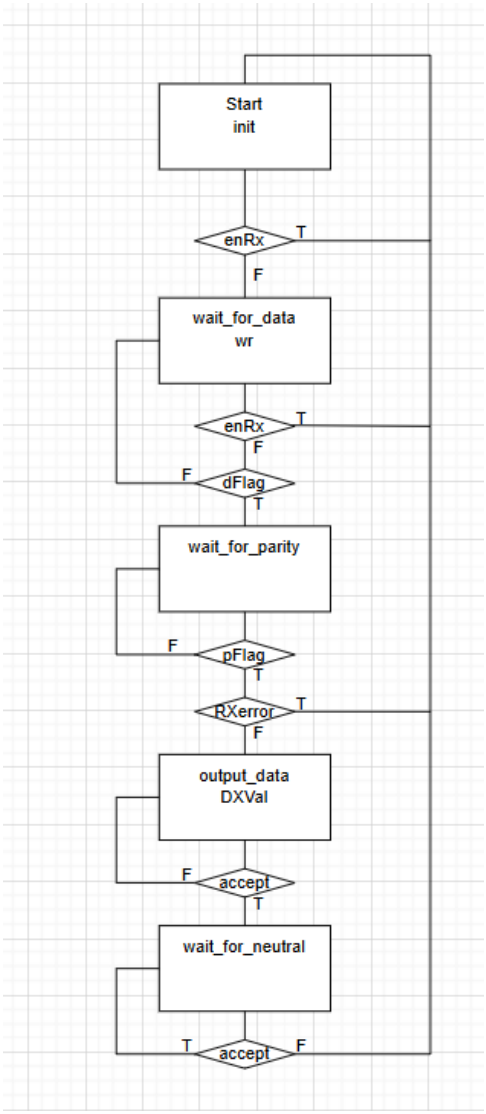

Figura 5: ASM Chart do *Serial Control*

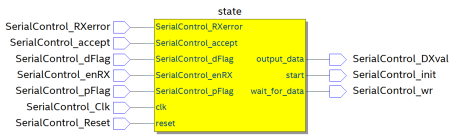

Figura 6: Diagrama de Blocos do módulo *Serial Control*

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.

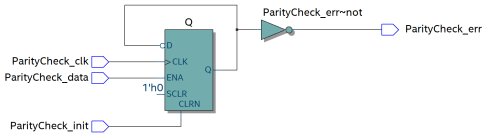

Figura 7: Diagrama de Blocos do módulo *Parity Check*

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 Roulette* transmitindo assim apenas os bits de dados. Com isto, o diagrama de blocos correspondente ao *Shift Register* está representado na Figura 8.

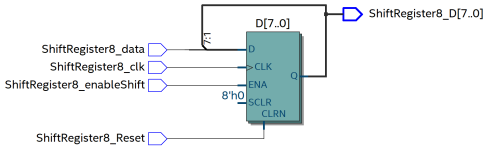

Figura 8: Diagrama de Blocos do módulo *Shift Register*

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.

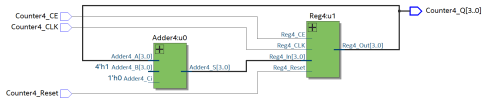

Figura 9: Diagrama de Blocos do módulo *Counter*

3 Roulette Dispatcher

or fim, temos o bloco *Roulette Dispatcher* que é responsável por entregar os dados válidos do *Serial Receiver* ao *Roullete Display* enviando assim, os bits de dados, os comandos o sinal WrD para posteiormente ligar o Roullete Display. Para a conceção do *Roulette Dispatcher* foi preciso conceber uma máquina de

estados podendo ser visualizada na 10. O diagrama de blocos condizente com o *LCD Dispatcher* está representado na Figura 11.

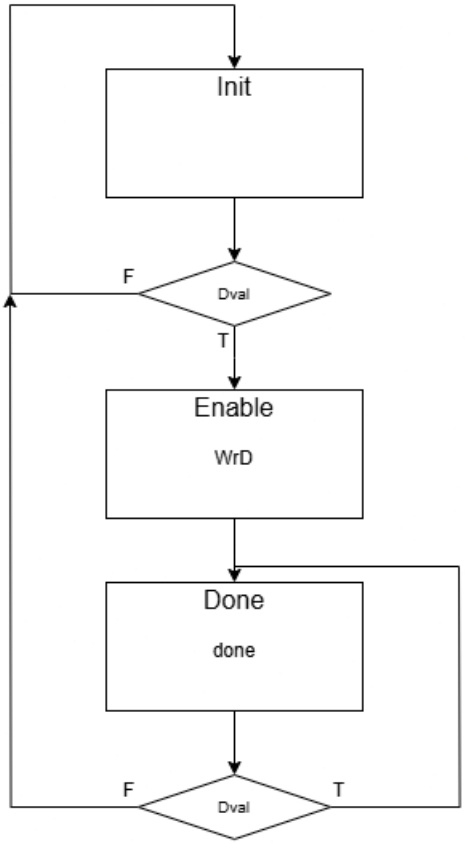

Figura 10: ASM-Chart do *Roulette Dispatcher*

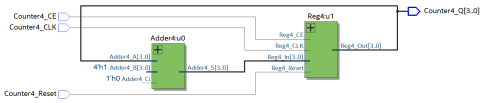

Figura 11: Diagrama de Blocos do módulo *LCD Roulette*

4 Conclusões

Em conclusão, o desenvolvimento do *SRC* apresentou sérias dificuldades e complexidades da mesma maneira que apresentou o *SLCDC*, pois ambos usam os mesmos componentes apenas diferenciando-se na quantidade de bits a ser enviado(no *SRC* 8) e ao que enviam a informação que no caso é o RRoulette Dispatcher.

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 SRC

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

entity 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 SRC;

architecture logicFunction of SRC is

    component SRCSerialReceiver is
            port(

                -- input
        SRCSerialReceiver_Reset: in std_logic;
                SRCSerialReceiver_SS: in std_logic;
                SRCSerialReceiver_SCLK: in std_logic;
                SRCSerialReceiver_CLK: in std_logic;
                SRCSerialReceiver_SDX: in std_logic;
                SRCSerialReceiver_accept: in std_logic;

                -- output
                SRCSerialReceiver_DXval: out std_logic;
                SRCSerialReceiver_D: out std_logic_vector(7 downto 0)

        );
    end component;

    component RouletteDispatcher is
        port(

                -- input
        RouletteDispatcher_Reset: in std_logic;
                RouletteDispatcher_Dval: in std_logic;
        RouletteDispatcher_Din: in std_logic_vector(7 downto 0);
                RouletteDispatcher_CLK: in std_logic;

                -- output
                RouletteDispatcher_WrD: out std_logic;
                RouletteDispatcher_Dout: out std_logic_vector(7 downto 0);
        RouletteDispatcher_done: out std_logic

        );
    end component;

    signal val: std_logic;
    signal D: std_logic_vector(7 downto 0);
    signal done_to_accept: std_logic;
```

```
begin

    u0: SRCSerialReceiver port map (
        SRCSerialReceiver_Reset => SRC_Reset,
        SRCSerialReceiver_SS => SRC_RDSel,
        SRCSerialReceiver_SCLK => SRC_SCLK,
        SRCSerialReceiver_CLK => SRC_CLK,
        SRCSerialReceiver_SDX => SRC_SDX,
        SRCSerialReceiver_accept => done_to_accept,
        SRCSerialReceiver_DXval => val,
        SRCSerialReceiver_D => D
    );

    u1: RouletteDispatcher port map (
        RouletteDispatcher_Reset => SRC_Reset,
        RouletteDispatcher_Dval => val,
        RouletteDispatcher_Din => D,
        RouletteDispatcher_CLK => SRC_CLK,
        RouletteDispatcher_WrD => SRC_set,
        RouletteDispatcher_Dout => SRC_D,
        RouletteDispatcher_done => done_to_accept
    );

end logicFunction;
```

A.4 SRCSerialReceiver

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

entity SRCSerialReceiver is
        port(

                -- input
                SRCSerialReceiver_Reset: in std_logic;
                SRCSerialReceiver_SS: in std_logic;
                SRCSerialReceiver_SCLK: in std_logic;
                SRCSerialReceiver_CLK: in std_logic;
                SRCSerialReceiver_SDX: in std_logic;
                SRCSerialReceiver_accept: in std_logic;

                -- output
                SRCSerialReceiver_DXval: out std_logic;
                SRCSerialReceiver_D: out std_logic_vector(7 downto 0)

        );
end SRCSerialReceiver;

architecture logicFunction of SRCSerialReceiver 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 ShiftRegister8 is port (

                -- input
                ShiftRegister8_Reset: in std_logic;
                ShiftRegister8_data: in std_logic;
                ShiftRegister8_clk: in std_logic;
                ShiftRegister8_enableShift: in std_logic;

                -- output
                ShiftRegister8_D: out std_logic_vector(7 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 Counter4 is port (

                -- input
                Counter4_CE: in std_logic;
                Counter4_CLK: in std_logic;
                Counter4_Reset: in std_logic;

                -- output
                Counter4_Q: out std_logic_vector(3 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(3 downto 0);

begin

        u0: SerialControl port map (
                SerialControl_Reset => SRCSerialReceiver_Reset,
                SerialControl_enRX => SRCSerialReceiver_SS,
                SerialControl_accept => SRCSerialReceiver_Accept,
                SerialControl_pFlag => pFlag,
                SerialControl_dFlag => dFlag,
                SerialControl_RXerror => RXerror,
        SerialControl_Clk => SRCSerialReceiver_CLK,
                SerialControl_wr => wr,
                SerialControl_init => init,
                SerialControl_DXval => SRCSerialReceiver_DXval
        );

        RXerror <= error and not SRCSerialReceiver_SCLK;

        u1: ShiftRegister8 port map (
                ShiftRegister8_Reset => SRCSerialReceiver_Reset,
                ShiftRegister8_data => SRCSerialReceiver_SDX,
                ShiftRegister8_clk => SRCSerialReceiver_SCLK,
                ShiftRegister8_enableShift => wr,
                ShiftRegister8_D => SRCSerialReceiver_D
        );

        u2: ParityCheck port map (
                ParityCheck_data => SRCSerialReceiver_SDX,
                ParityCheck_init => init,
                ParityCheck_clk => SRCSerialReceiver_SCLK,
                ParityCheck_err => error
        );

        u3: Counter4 port map (
                Counter4_CE => '1',
```

```
                Counter4_CLK => SRCSerialReceiver_SCLK,
                Counter4_Reset => init,
                Counter4_Q => counter
        );

        dFlag <= counter(3) and not counter(2) and not counter(1) and not counter(0) and not SRCSerialReceiver_SCLK;
        pflag <= counter(3) and not counter(2) and not counter(1) and counter(0) and not SRCSerialReceiver_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 ShiftRegister8

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

entity ShiftRegister8 is
        port(

                -- input
                ShiftRegister8_Reset: in std_logic;
                ShiftRegister8_data: in std_logic;
                ShiftRegister8_clk: in std_logic;
                ShiftRegister8_enableShift: in std_logic;

                -- output
                ShiftRegister8_D: out std_logic_vector(7 downto 0)

        );
end ShiftRegister8;

architecture logicFunction of ShiftRegister8 is

    signal D: std_logic_vector(7 downto 0);

begin

        D(7) <= '0' when ShiftRegister8_Reset = '1' else ShiftRegister8_data when rising_edge(ShiftRegister8_clk) and ShiftR
        D(6) <= '0' when ShiftRegister8_Reset = '1' else D(7) when rising_edge(ShiftRegister8_clk) and ShiftRegister8_enable
        D(5) <= '0' when ShiftRegister8_Reset = '1' else D(6) when rising_edge(ShiftRegister8_clk) and ShiftRegister8_enable
        D(4) <= '0' when ShiftRegister8_Reset = '1' else D(5) when rising_edge(ShiftRegister8_clk) and ShiftRegister8_enable
    D(3) <= '0' when ShiftRegister8_Reset = '1' else D(4) when rising_edge(ShiftRegister8_clk) and ShiftRegister8_enableShif
    D(2) <= '0' when ShiftRegister8_Reset = '1' else D(3) when rising_edge(ShiftRegister8_clk) and ShiftRegister8_enableShif
    D(1) <= '0' when ShiftRegister8_Reset = '1' else D(2) when rising_edge(ShiftRegister8_clk) and ShiftRegister8_enableShif
    D(0) <= '0' when ShiftRegister8_Reset = '1' else D(1) when rising_edge(ShiftRegister8_clk) and ShiftRegister8_enableShif

    ShiftRegister8_D <= D;

end logicFunction;
```

A.8 Counter4

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

entity Counter4 is
        port(

                -- input
                Counter4_CE: in std_logic;
                Counter4_CLK: in std_logic;
                Counter4_Reset: in std_logic;

                -- output
                Counter4_Q: out std_logic_vector(3 downto 0)

        );
end Counter4;

architecture logicFunction of Counter4 is

        component Adder4 is
                port(

                        -- input
                        Adder4_Ci: in std_logic;
                        Adder4_A: in std_logic_vector(3 downto 0);
                        Adder4_B: in std_logic_vector(3 downto 0);

                        -- output
                        Adder4_Co: out std_logic;
                        Adder4_S: out std_logic_vector(3 downto 0)

                );
        end component;

        component Reg4 is
                port(

                        -- input
                        Reg4_In: in std_logic_vector(3 downto 0);
                        Reg4_CE: in std_logic;
                        Reg4_CLK: in std_logic;
                        Reg4_Reset: in std_logic;

                        -- output
                        Reg4_Out: out std_logic_vector(3 downto 0)

                );
        end component;

        signal Q: std_logic_vector(3 downto 0);
        signal Adder4_S_to_Reg4_In: std_logic_vector(3 downto 0);

begin

        u0: Adder4 port map (
                Adder4_Ci => '0',
                Adder4_A => Q,
                Adder4_B => "0001",
                Adder4_S => Adder4_S_to_Reg4_In
        );
```

```
        u1: Reg4 port map (
                Reg4_In => Adder4_S_to_Reg4_In,
                Reg4_CE => Counter4_CE,
                Reg4_CLK => Counter4_CLK,
                Reg4_Reset => Counter4_Reset,
                Reg4_Out => Q
        );

        Counter4_Q <= Q;

end logicFunction;
```

A.9 Adder4

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

entity Adder4 is
    port(

        -- input
        Adder4_Ci: in std_logic;
        Adder4_A: in std_logic_vector(3 downto 0);
        Adder4_B: in std_logic_vector(3 downto 0);

        -- output
        Adder4_Co: out std_logic;
        Adder4_S: out std_logic_vector(3 downto 0)

    );
end Adder4;

architecture logicFunction of Adder4 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;
    signal c2_to_c3: std_logic;

begin

    c0: FullAdder port map (
        FullAdder_A => Adder4_A(0),
        FullAdder_B => Adder4_B(0),
        FullAdder_Ci => Adder4_Ci,
        FullAdder_S => Adder4_S(0),
        FullAdder_Co => c0_to_c1
    );

    c1: FullAdder port map (
        FullAdder_A => Adder4_A(1),
        FullAdder_B => Adder4_B(1),
        FullAdder_Ci => c0_to_c1,
        FullAdder_S => Adder4_S(1),
        FullAdder_Co => c1_to_c2
    );

    c2: FullAdder port map (
        FullAdder_A => Adder4_A(2),
        FullAdder_B => Adder4_B(2),
        FullAdder_Ci => c1_to_c2,
```

```
        FullAdder_S => Adder4_S(2),
        FullAdder_Co => c2_to_c3
    );

    c3: FullAdder port map (
        FullAdder_A => Adder4_A(3),
        FullAdder_B => Adder4_B(3),
        FullAdder_Ci => c2_to_c3,
        FullAdder_S => Adder4_S(3),
        FullAdder_Co => Adder4_Co
    );

end logicFunction;
```

A.10 Reg4

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

entity Reg4 is
        port(

                -- input
                Reg4_In: in std_logic_vector(3 downto 0);
                Reg4_CE: in std_logic;
                Reg4_CLK: in std_logic;
                Reg4_Reset: in std_logic;

                -- output
                Reg4_Out: out std_logic_vector(3 downto 0)

        );
end Reg4;

architecture logicFunction of Reg4 is
begin

        Reg4_Out(3) <= '0' when Reg4_Reset = '1' else Reg4_In(3) when rising_edge(Reg4_CLK) and Reg4_CE = '1';
        Reg4_Out(2) <= '0' when Reg4_Reset = '1' else Reg4_In(2) when rising_edge(Reg4_CLK) and Reg4_CE = '1';
        Reg4_Out(1) <= '0' when Reg4_Reset = '1' else Reg4_In(1) when rising_edge(Reg4_CLK) and Reg4_CE = '1';
        Reg4_Out(0) <= '0' when Reg4_Reset = '1' else Reg4_In(0) when rising_edge(Reg4_CLK) and Reg4_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 RouletteDispatcher

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

entity RouletteDispatcher is
        port(

                -- input
        RouletteDispatcher_Reset: in std_logic;
                RouletteDispatcher_Dval: in std_logic;
        RouletteDispatcher_Din: in std_logic_vector(7 downto 0);
                RouletteDispatcher_CLK: in std_logic;

                -- output
                RouletteDispatcher_WrD: out std_logic;
                RouletteDispatcher_Dout: out std_logic_vector(7 downto 0);
        RouletteDispatcher_done: out std_logic

        );
end RouletteDispatcher;

architecture logicFunction of RouletteDispatcher is

    type Tstate is (wait_for_data, enable, done);

    signal state: Tstate;
    signal next_state: Tstate;

begin

    state <= wait_for_data when RouletteDispatcher_Reset = '1' else next_state when rising_edge(RouletteDispatcher_CLK);

    process(state, RouletteDispatcher_Dval, RouletteDispatcher_Din) begin

        case state is

            when wait_for_data =>
                if RouletteDispatcher_Dval = '1' then
                    next_state <= enable;
                else
                    next_state <= wait_for_data;
                end if;

            when enable =>
                next_state <= done;

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

        end case;
    end process;

    RouletteDispatcher_Dout <= RouletteDispatcher_Din;
    RouletteDispatcher_WrD <= '1' when state = enable else '0';
    RouletteDispatcher_done <= '1' when state = done else '0';

end logicFunction;
```

B *Kotlin*

B.1 HAL

Algoritmo 1: HAL - Hardware Abstract Layer

```
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 RouletteDisplay

Algoritmo 3: RouletteDisplay

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

/** Controla o mostrador de pontua  o. */
object RouletteDiplay {

    private const val UPDATE_MASK = 0b00000_110
    private const val ON_MASK = 0b00000_111
    private const val OFF_MASK = 0b00001_111

    // Caracteres especiais
    private const val CHAR_MINUS           = 0x10
    private const val CHAR_TOP_RIGHT       = 0x11
    private const val CHAR_TOP_LEFT        = 0x12
    private const val CHAR_LEFT            = 0x13
    private const val CHAR_BOTTOM_LEFT     = 0x14
    private const val CHAR_BOTTOM_RIGHT    = 0x15
    private const val CHAR_RIGHT           = 0x16
    private const val CHAR_LEFT_RIGHT      = 0x17
    private const val CHAR_TOP_MID_BOTTOM  = 0x18
    private const val CHAR_TOP             = 0x19
    private const val CHAR_LEFT_UPPER      = 0x1a
    private const val CHAR_LEFT_LOWER      = 0x1b
    private const val CHAR_BOTTOM          = 0x1c
    private const val CHAR_RIGHT_LOWER     = 0x1d
    private const val CHAR_RIGHT_UPPER     = 0x1e
    private const val CHAR_EMPTY           = 0x1f

    /** Velocidade da anima  o */
    private const val ANIM_SPEED = 100

    /** _Frames_ de anima  o. */
    private val ANIM_FRAMES = listOf(
        listOf(CHAR_LEFT          , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY
, CHAR_EMPTY         ), // [|      ]
        listOf(CHAR_TOP_LEFT      , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY
, CHAR_EMPTY         ), // [/      ]
        listOf(CHAR_TOP           , CHAR_TOP     , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY
, CHAR_EMPTY         ), // [--     ]
        listOf(CHAR_EMPTY         , CHAR_TOP     , CHAR_TOP     , CHAR_EMPTY   , CHAR_EMPTY
, CHAR_EMPTY         ), // [ --    ]
        listOf(CHAR_EMPTY         , CHAR_EMPTY   , CHAR_TOP     , CHAR_TOP     , CHAR_EMPTY
, CHAR_EMPTY         ), // [  --   ]
        listOf(CHAR_EMPTY         , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_TOP     , CHAR_TOP
, CHAR_EMPTY         ), // [   -- ]
        listOf(CHAR_EMPTY         , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_TOP
, CHAR_TOP           ), // [     --]
        listOf(CHAR_EMPTY         , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY
, CHAR_TOP_RIGHT     ), // [      \]
        listOf(CHAR_EMPTY         , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY
, CHAR_RIGHT         ), // [      |]
        listOf(CHAR_EMPTY         , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY
, CHAR_BOTTOM_RIGHT  ), // [      /]
        listOf(CHAR_EMPTY         , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY
, CHAR_BOTTOM  , CHAR_BOTTOM       ), // [    --]
        listOf(CHAR_EMPTY         , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_BOTTOM  , CHAR_BOTTOM  , CHAR_EMPTY
), // [   -- ]
        listOf(CHAR_EMPTY         , CHAR_EMPTY   , CHAR_BOTTOM  , CHAR_BOTTOM  , CHAR_EMPTY
, CHAR_EMPTY         ), // [  --  ]
        listOf(CHAR_EMPTY         , CHAR_BOTTOM  , CHAR_BOTTOM  , CHAR_EMPTY   , CHAR_EMPTY
, CHAR_EMPTY         ), // [ --   ]
        listOf(CHAR_BOTTOM        , CHAR_BOTTOM  , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY
, CHAR_EMPTY         ), // [--    ]
        listOf(CHAR_BOTTOM_LEFT   , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY   , CHAR_EMPTY
, CHAR_EMPTY         )  // [\     ]
    )

    /** Inicializa o mostrador. */
    fun init(){
```

```
    off(true) // Desliga
    off(false) // Liga
    clear() // Limpa
}

/** Atualiza a _frame_ de anima  o. */
fun animation(){
    val time = Time.getTimeInMillis() / ANIM_SPEED
    val i = (time % ANIM_FRAMES.size).toInt()
    setAllDigits(ANIM_FRAMES[i])
}

/** Escreve um n mero no centro do mostrador. */
fun setValue(value: Int) = write(value.toString(), Alignment.CENTER)

/**
 * Envia um comando para desativar/ativar a visualiza  o do mostrador.
 *
 * - `true`: Desativar
 * - `false`: Ativar
 */
fun off(value: Boolean){
    SerialEmitter.send(
        SerialEmitter.Destination.ROULETTE,
        if(value) OFF_MASK else ON_MASK,
        8
    )
}

/** Define um d gito numa posi  o espec fica. */
private fun setDigit(pos: Int, value: Int){
    require(pos in 0..5) { "pos must be between 0 to 5" }
    require(value in 0x00..0x1f) { "value must be from 0x00 to 0x1f" }

    // Evita atualizar o d gito caso seja igual ao  ltimo  estado
    if(lastState[pos] == value) return

    // Atualizar pr ximo estado
    nextState[pos] = value

    // Enviar para o mostrar
    SerialEmitter.send(SerialEmitter.Destination.ROULETTE, (value shl 3) or (5 - pos), 8)

}

/** Define todos os d gitos a partir de uma lista e atualiza */
private fun setAllDigits(digits: List<Int>){
    repeat(6){ i -> setDigit(i, digits[i]) }
    update()
}

/** Define todos os d gitos e atualiza. */
private fun setAllDigits(d0: Int, d1: Int, d2: Int, d3: Int, d4: Int, d5: Int) =
    setAllDigits(listOf(d0, d1, d2, d3, d4, d5))

/** Escreve caracteres no mostrador com alinhamento. */
fun write(string: String, align: Alignment = Alignment.LEFT){
    require(Regex("^[-_0-9a-fA-F ]{0,6}\$").matches(string)) { "string must be up to 6 characters long and

    // Calcular  nicio  dependendo do alinhamento
    val startPos = when(align){
        Alignment.LEFT -> 0
        Alignment.CENTER -> (6 - string.length) / 2
        Alignment.RIGHT -> 6 - string.length
    }

    // Limpar tudo
    repeat(6){ i -> setDigit(i, CHAR_EMPTY) }

    // Escrever texto
    repeat(string.length){ i ->
        setDigit(startPos + i, when(string[i]){
            '-' -> CHAR_MINUS
            '_' -> CHAR_BOTTOM
```

```
                    '0' -> 0
                    '1' -> 1
                    '2' -> 2
                    '3' -> 3
                    '4' -> 4
                    '5' -> 5
                    '6' -> 6
                    '7' -> 7
                    '8' -> 8
                    '9' -> 9
                    'a', 'A' -> 0xa
                    'b', 'B' -> 0xb
                    'c', 'C' -> 0xc
                    'd', 'D' -> 0xd
                    'e', 'E' -> 0xe
                    'f', 'F' -> 0xf
                    ' ' -> CHAR_EMPTY
                    else -> throw IllegalStateException("Unexpected ${string[i]} character!")
                }
            }

            update()

        }

        private var lastState = MutableList(6) { 0 }
        private var nextState = MutableList(6) { 0 }

        /**
         * Atualiza mostrador, caso haja altera  es.
         * @property forced Atualiza mesmo n o havendo altera  es
         */
        private fun update(forced: Boolean = false){

            // Evitar atualizar caso n o seja for ado e n o exista mudan as
            if(!forced && lastState == nextState) return

            // Enviar comando para atualizar
            SerialEmitter.send(SerialEmitter.Destination.ROULETTE, UPDATE_MASK, 8)

            // Atualizar o  ltimo  estado
            repeat(6){ i-> lastState[i] = nextState[i] }

        }

        /** Limpa o mostrador. */
        fun clear() = write("      ")
}

fun main(){

    RouletteDiplay.init()

    RouletteDiplay.write("2-5", Alignment.CENTER)
    Time.sleep(1000)

    val end = Time.getTimeInMillis() + 5000
    while(Time.getTimeInMillis() < end)
        RouletteDiplay.animation()

    RouletteDiplay.setValue(-3)
}
```

B.4 RouletteGame

Algoritmo 4: RouletteGame

```
import isel.leic.utils.Time
import util.Alignment
import util.RouletteBoard
import kotlin.math.ceil

object RouletteGame {

    /** N   de moedas a gastar. */
    private var credits = 0

    /** Deteta novas moedas e adiciona ao jogo. */
    private fun checkNewCoins(){

        // Detetar nova moeda
        val coin = CoinAcceptor.get()
        if(coin == 0) return

        // Adicionar aos cr ditos
        credits += coin

    }

    fun init(){

        // Inicializa  o

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


        CoinDeposit.init()
        Statistics.init()

        // Ecr  Inicial

        while(true){

            checkNewCoins()

            // --- LCD ---

            // T tulo
            TUI.write("Roulette␣Game", 0)

            // Cr ditos
            val creditsStr = "␣${credits}$"
            TUI.write(creditsStr, 1, Alignment.RIGHT)

            // Instru  es
            TUI.scrollText(
                if(credits > 0) "Click␣*␣to␣play!" else "Insert␣coins!",
                1,
                range = 0..(LCD.COLS - creditsStr.length)
            )

            // --- KB ---

            // Come ar o jogo
            if(TUI.getKey() == '*' && credits > 0) startGame()

            // Manuten  o
            if(M.check()) M.maintenanceMode()

        }
```

```
}

fun startGame(maintenance: Boolean = false){

    // Apostas

    val bets = RouletteBoard.default(0)
    var lastBet = Time.getTimeInMillis()
    var betEnd: Long? = null

    TUI.clear()

    while(true){

        checkNewCoins()

        // --- LCD ---

        val betSum = bets.toList().sum()

        if(betSum == 14 * 9) {

            // O jogador n o pode apostar mais
            // Mostrar instru  o para come ar a jogar
            var msg = "Max bets!"
            if(betEnd == null) msg += " Click # to start."
            TUI.scrollText(msg, 0, range = 0..LCD.COLS - 2)

        }else if(Time.getTimeInMillis() - lastBet > 5e3 && credits == 0){

            // O jogador est  inativo e necessita de mais cr ditos
            val msg = if(betEnd == null) "Click # to start or insert coins!" else "Insert coins to keep be
            TUI.scrollText(msg, 0, range = 0..LCD.COLS - 2)

        }else if(Time.getTimeInMillis() - lastBet > 5e3 && betSum > 0) {

            // O jogador est  inativo
            // Mostrar instru  o para come ar a jogar
            val msg = if(betEnd == null) "Click # to start or keep betting!" else "You can still bet!"
            TUI.scrollText(msg,0, range = 0..LCD.COLS - 2)

        }else if(betSum > 0){

            // O jogador j  apostou
            // Mostrar apostas do utilizador
            TUI.write(
                bets.toList().joinToString("") { if (it == 0) " " else it.toString() },
                0, Alignment.LEFT
            )

        }else{

            // O jogador ainda n o apostou
            // Mostrar instru  o para apostar
            TUI.scrollText("Click on these keys to bet!", 0, range = 0..LCD.COLS - 2)

        }

        val cornerIcon = when {
            betEnd != null -> ceil((betEnd - Time.getTimeInMillis()) / 1e3).toInt().toString()
            !maintenance -> "$"
            else -> null
        }
        if(cornerIcon != null) TUI.write(cornerIcon, 0, Alignment.RIGHT)

        TUI.write("0123456789ABCD ${when{
                maintenance -> TUI.MAINTENANCE
                credits > 9 -> TUI.NINE_PLUS
                else -> credits
            }}", 1)

        // --- KB ---

        val key = TUI.getKey()
```

```
        // Apostas
        if(
            key in RouletteBoard.SLOTS &&
            (credits > 0 || maintenance) && // Tem cr ditos ou est  em modo manuten  o
            bets[key] < 9 // N o chegou ao limite de apostas dessa tecla
        ){

            bets[key]++
            if(!maintenance){

                // Retirar dos cr ditos
                credits--

                // Adicionar ao dep sito
                CoinDeposit.addCoin()

            }

            lastBet = Time.getTimeInMillis()

        }

        // Come ar o jogo
        if(
            key == '#' &&
            bets.toList().sum() > 0 && // Existe alguma aposta
            betEnd == null // Jogo ainda n o come ou
        ) betEnd = Time.getTimeInMillis() + 5000L // Acabar as apostas em 5 segundos

        // O jogo j  come ou
        if(betEnd != null){
            RouletteDiplay.animation()
            if(Time.getTimeInMillis() > betEnd) break // Acabar as apostas
        }

    }

    // Anima  o

    TUI.clear()

    val animEnd = Time.getTimeInMillis() + (1000L..5000L).random()
    while(Time.getTimeInMillis() < animEnd){

        RouletteDiplay.animation()

        TUI.write("Good luck!", 0)
        TUI.scrollText("Not accepting more bets.", 1)

    }

    // Fim

    TUI.clear()

    val winSlot = (0x0..0xD).random()
    val winCredits = bets[winSlot] * 2

    RouletteDiplay.write("${winSlot.toString(16)}   ${winCredits.toString().padStart(2, ' ')}")

    val winEnd = Time.getTimeInMillis() + 5000L
    while(Time.getTimeInMillis() < winEnd){
        TUI.write(if(bets[winSlot] > 0) "Congrats! ${TUI.HAPPY}" else "Oh no! ${TUI.SAD}", 0)
        TUI.scrollText(if(bets[winSlot] > 0) "You won ${winCredits} credits!" else "Better luck next time!
    }

    if(!maintenance){

        credits += winCredits

        CoinDeposit.cashOut(winCredits)

        // Atualizar info e estat sticas
```

```
                Statistics.addResult(winSlot, winCredits)
                CoinDeposit.addGamePlayed()

            }

            RouletteDiplay.clear()

        }

}

fun main() = RouletteGame.init()
```

B.5 RouletteBoard

Algoritmo 5: RouletteBoard

```
package util

/**  til  para organizar dados associados a cada op  o de aposta. */
data class RouletteBoard<E>(
    var slot0: E,
    var slot1: E,
    var slot2: E,
    var slot3: E,
    var slot4: E,
    var slot5: E,
    var slot6: E,
    var slot7: E,
    var slot8: E,
    var slot9: E,
    var slotA: E,
    var slotB: E,
    var slotC: E,
    var slotD: E
): Iterable<E> {

    companion object Factory {

        const val SLOTS = "0123456789ABCD"

        /** Cria um RouletteBoard que tenta todos os dados iguais. */
        fun <E> default(value: E) = List(14){ value }.toRouletteBoard()

    }

    /** Acesso via n meros hexadecimais. */
    operator fun get(i: Int) = when(i){
        0x0 -> slot0
        0x1 -> slot1
        0x2 -> slot2
        0x3 -> slot3
        0x4 -> slot4
        0x5 -> slot5
        0x6 -> slot6
        0x7 -> slot7
        0x8 -> slot8
        0x9 -> slot9
        0xA -> slotA
        0xB -> slotB
        0xC -> slotC
        0xD -> slotD
        else -> throw IndexOutOfBoundsException("$i is outside de 0x0..0xD range.")
    }

    /** Acesso via caracteres. */
    operator fun get(i: Char) = when(i){
        '0' -> slot0
        '1' -> slot1
        '2' -> slot2
        '3' -> slot3
        '4' -> slot4
        '5' -> slot5
        '6' -> slot6
        '7' -> slot7
        '8' -> slot8
        '9' -> slot9
        'A' -> slotA
        'B' -> slotB
        'C' -> slotC
        'D' -> slotD
        else -> throw IndexOutOfBoundsException("$i is outside de '0'..'D' range.")
    }

    /** Atribui  o via n meros hexadecimais. */
    operator fun set(i: Int, value: E) = when(i){
```

```
            0x0 -> slot0 = value
            0x1 -> slot1 = value
            0x2 -> slot2 = value
            0x3 -> slot3 = value
            0x4 -> slot4 = value
            0x5 -> slot5 = value
            0x6 -> slot6 = value
            0x7 -> slot7 = value
            0x8 -> slot8 = value
            0x9 -> slot9 = value
            0xA -> slotA = value
            0xB -> slotB = value
            0xC -> slotC = value
            0xD -> slotD = value
            else -> throw IndexOutOfBoundsException("$i is outside de 0x0..0xD range.")
        }

        /** Atribui  o via caracteres. */
        operator fun set(i: Char, value: E) = when(i){
            '0' -> slot0 = value
            '1' -> slot1 = value
            '2' -> slot2 = value
            '3' -> slot3 = value
            '4' -> slot4 = value
            '5' -> slot5 = value
            '6' -> slot6 = value
            '7' -> slot7 = value
            '8' -> slot8 = value
            '9' -> slot9 = value
            'A' -> slotA = value
            'B' -> slotB = value
            'C' -> slotC = value
            'D' -> slotD = value
            else -> throw IndexOutOfBoundsException("$i is outside de '0'..'D' range.")
        }

        /** Convers o para lista de dados. */
        fun toList() = listOf(
            slot0,
            slot1,
            slot2,
            slot3,
            slot4,
            slot5,
            slot6,
            slot7,
            slot8,
            slot9,
            slotA,
            slotB,
            slotC,
            slotD
        )

        /** Iterador */
        override fun iterator() = toList().iterator()

}

/** Convers o de lista para RouletteBoard. */
fun <E> List<E>.toRouletteBoard(): RouletteBoard<E> {
    require(size == 14) { "List must have exactly 14 elements (0x0..0xD)" }
    return RouletteBoard(
        this[0x0],
        this[0x1],
        this[0x2],
        this[0x3],
        this[0x4],
        this[0x5],
        this[0x6],
        this[0x7],
        this[0x8],
        this[0x9],
        this[0xA],
```

```
            this[0xB],
            this[0xC],
            this[0xD]
        )
}
```