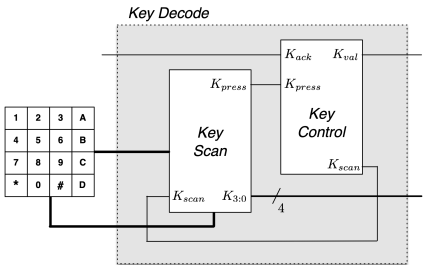
Autores: Ricardo Martins nº52326 / Bernardo Silva nº52605 / Bernardo Esteves nº52948

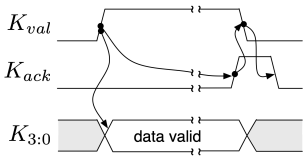
# 1 Key Decode

O bloco *Key Decode* implementa um descodificador de um teclado matricial 4x4 por hardware, sendo constituído por três sub-blocos: *i*) um teclado matricial de 4x4; *ii*) o bloco Key Scan, responsável pelo varrimento do teclado; e *iii*) o bloco Key Control, que realiza o controlo do varrimento e o controlo de fluxo, conforme o diagrama de blocos representado na Figura 1a.

O controlo de fluxo de saída do bloco *Key Decode* (para o módulo *Key Buffer*), define que o sinal $K_{val}$ é ativado quando é detetada a pressão de uma tecla, sendo também disponibilizado o código dessa tecla no barramento $K_{0:3}$. Apenas é iniciado um novo ciclo de varrimento ao teclado quando o sinal $K_{ack}$ for ativado e a tecla premida for libertada. O diagrama temporal do controlo de fluxo está representado na Figura 1b.




**(a)** Diagrama de blocos




**(b)** Diagrama temporal

**Figura 1:** Bloco *Key Decode*

## 1.1 Key Scan

O bloco *Key Scan* foi implementado de acordo com o diagrama de blocos representado na Figura 3.

Escolhemos este sistema, pois ao utilizar apenas um contador, temos um sistema muito mais sincronizado, o que facilita o processo de decodificação da tecla pressionada. Através desta implementação conseguimos um sistema mais coeso e simples de utilizar. Para a concepção do contador foi utilizado um *Adder* e um registo de 4 *bits* onde o *Adder* internamente é constituído por 4 *Full Adders*.Para o resto da implementação do *Key Scan* também foram utilizados um decoder e um mux.Após o desenvolvimento do *Key Scan* em *VHDL* foi obtido o Diagrama de Blocos do mesmo podendo ser visualizado na 2.

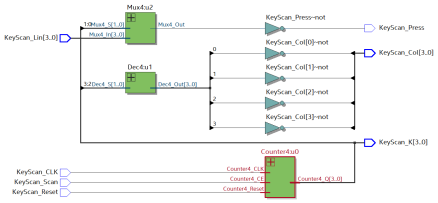


**Figura 2:** Diagrama de blocos do módulo *Key Scan*

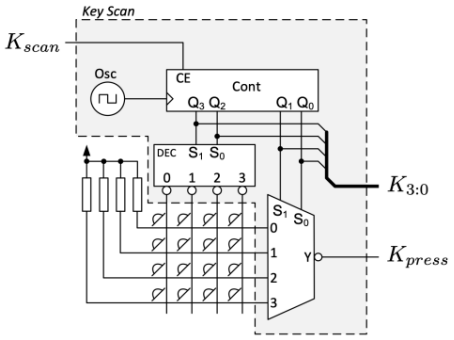



**Figura 3:** Diagrama do *Key Scan*

## 1.2 Key Control

O bloco *Key Control* foi implementado pela máquina de estados representada em *ASM-chart* na Figura 4. Para desenvolver o *ASM-chart* do *Key Control* foi preciso uma análise sobre o diagrama de blocos da figura 1a. Com isso e após perceber o funcionamento do mesmo construímos o *ASM-chart*, a tabela de estados e a expressão lógico para depois desenvolvermos em *VHDL* representado na figura 4.Ao finalizar a implementação do mesmo chegamos ao resultado referente á figura 5.

A descrição hardware do bloco *Key Control* em *VHDL* encontra-se no Anexo **??**.

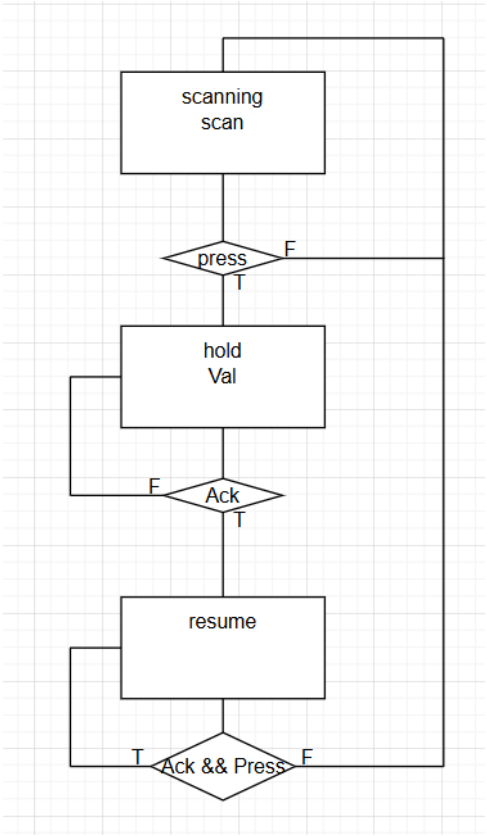



**Figura 4:** Máquina de estados do *Key Control*

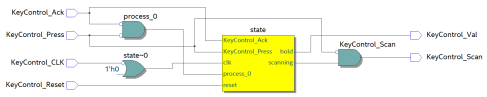



**Figura 5:** Diagrama de Blocos do *Key Control*

# 2 Ring Buffer

O bloco *Ring Buffer* foi implementado de acordo com o diagrama de blocos representado na Figura 6. O *Ring Buffer* tem como objetivo controlar a entrada de teclas vinda do *Key Decoder* através do *Ring Buffer Control* e com o *Memory Address Contro* gerenciar o acesso á RAM para poder guardar as mesmas. Com isso, usando como referência o diagrama de blocos da Figura 6, a nossa implementação está representada na Figura 7.

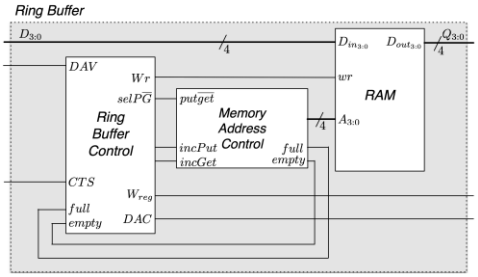



**Figura 6:** Diagrama de Blocos do *Ring Buffer*

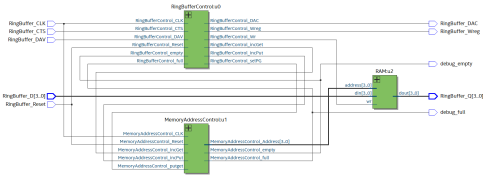


**Figura 7:** Diagrama de Blocos do módulo *Ring Buffer*

## 2.1 Ring Buffer Control

O Ring Buffer Control como dito anteriormente, tem a função de controlar as entrada das teclas vindas do *Key Decoder* e ao mesmo tempo gerenciar se a *RAM* tem espaço para armazenar essas teclas ou não. Com isso para a conceção do mesmo foi implementado pela máquina de estados representada pelo ASM-chart da figura 8. A máquina de estados consiste em 2 fases onde quando houver dados(DAV) e a *RAM* não estiver cheia, as teclas serão escritas e quando os dados estiverem prontos para serem enviados para o *Output Buffer*(CTS) e a *RAM* não estiver vazia irá ler ler os mesmos. O diagrama de blocos que representa o *Ring Buffer Control* está representado na 9.

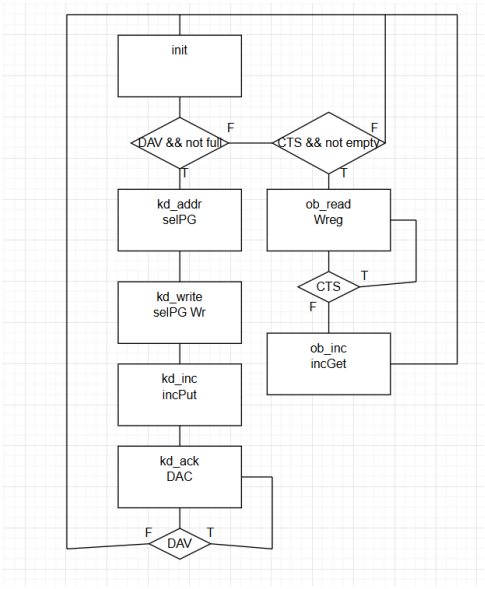



**Figura 8:** ASM-Chart do *Ring Buffer Control*

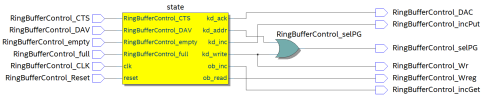



**Figura 9:** Diagrama de Blocos do módulo *Ring Buffer Control*

## 2.2 Memory Address Control

O bloco Memory Address Control é o responsável por gerenciar o acesso de escrita e leitura á *RAM* assim, manipulando o endereço onde irá escrever e de onde irá ler. Além, disso ainda informa ao *Ring Buffer Control* quando a *RAM* está vazia ou cheia. Com isso para a implementação do *Memory Address Control* foram usados 2 *Counters* de 4 bits onde a ativação dos mesmos é controlado pelos sinais incPut e incGet no qual, a saída de cada um destes *Counters* resulta como entrada do *Mu4-MAC* que é ativado pelo sinal putget e a saída é o address da *RAM*. Por fim, ainda temos um *Registo* que incrementa o putget sempre que houver incGet ou incPut e a saída gerencia quando que a *RAM* está cheia ou vazia. O diagrama de blocos que representa o *Memory Address Control* está representado na 10.

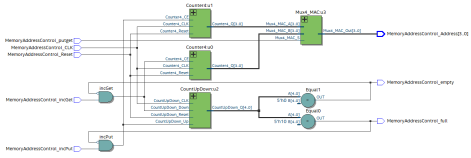


**Figura 10:** Diagrama de Blocos do módulo *Memory Address Control*

## 2.3 RAM

O bloco *RAM* já é fornecido pelos professores no qual, através dele é possível armazenar as teclas e consultar as mesmas quando necessário. O diagrama de blocos que representa o *RAM* está representado na 11.

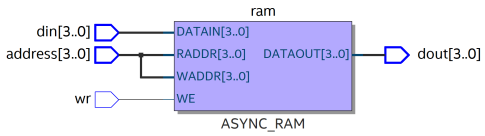



**Figura 11:** Diagrama de Blocos do módulo *RAM*

# 3 Output Buffer

O *Output Buffer* é o último bloco no Hardware antes de enviar os dados para o *Control*. Assim, o *Output Buffer* é constituído por outros 2 blocos *Buffer Control* e *Output Register* e a função do bloco como um todo é perceber quando que os dados estão disponíveis para serem recebidos do bloco *Ring Buffer* para os enviar para o bloco *Control*. Então com base no diagrama de bloco da Figura 12 implementámos a nossa versão do mesmo estando disponível o diagrama de blocos na Figura 13.

## 3.1 Buffer Control

O *Buffer Control* faz o manusemaneto de quando os dados são enviados para o controlo assim, para a implementação do mesmo recorremos a uma máquina de estados representada por um ASM-char na Figura 14. Quando o *Buffer Control* estiver

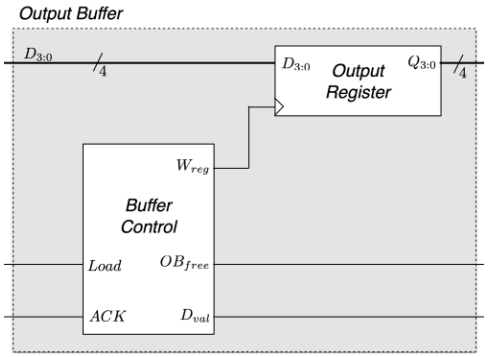



**Figura 12:** Diagrama de Blocos do *Output Buffer*

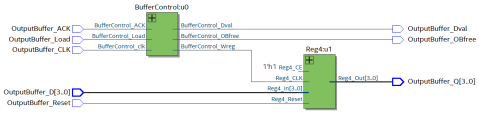


**Figura 13:** Diagrama de Blocos do módulo *Output Buffer*

disponível para receber dados(OBfree) e o *Ring Buffer* informar que tem dados(Load) irá ativar o cloak do *Output Register* para receber e enviar os dados e de seguida irá informar ao *Control* que os dados são válidos(Dval) e depois espera pela confirmação do *Control* que recebeu os dados(Ack) para voltar a estar disponível para receber dados. Assim o resultado da nossa implementação está disponível no diagrama de blocos da Figura 15.

## 3.2 Output Register

O Output Register é apenas responsável por guardar os dados provenientes do *Ring Buffer* e enviar para o *Control*. Sendo assim o *Output Register* é apenas um componente que detém de 4 *FFD's* onde cada um guarda o seu bit e envia. Com isto o diagrama de de blocos do *Output Register* está disponível na Figura 16. O componente usado para o *Output Register* foi o A.11.

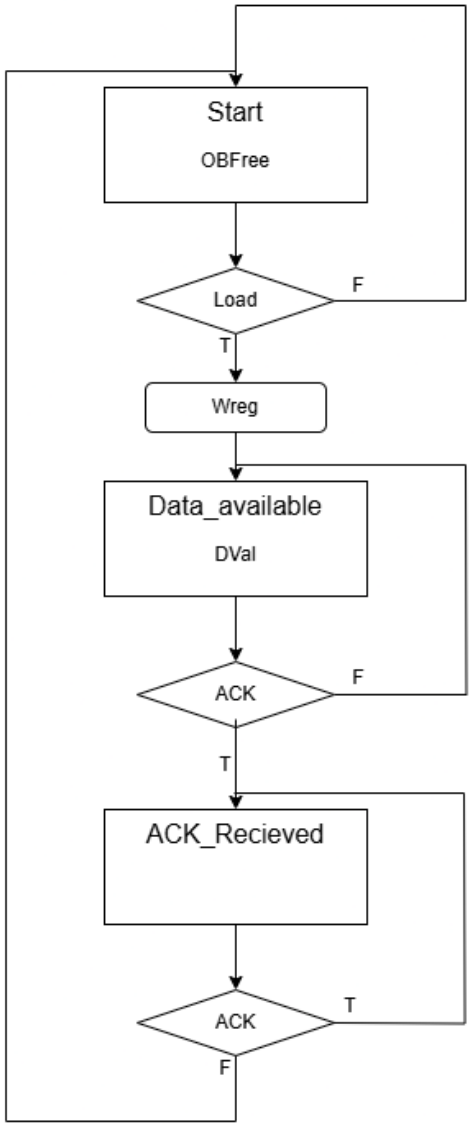



**Figura 14:** ASM do *Buffer Control*

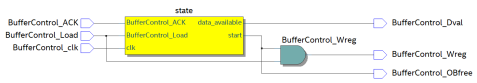



**Figura 15:** Diagrama de Blocos do módulo *Buffer Control*

# 4 Keyboard Reader

O *Keyboard Reader* é o o módulo que recebe as teclas, processa as mesmas e eventualmente envia para o *Control*. Dentro do *Keyboard Reader* temos o *Key Decode* que controla a deteção de teclas pres-

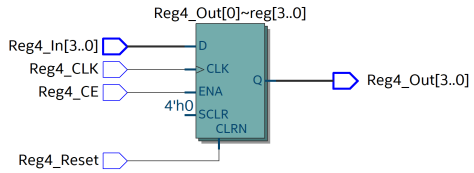



**Figura 16:** Diagrama de Blocos do módulo *Output Register*

sionadas e as envia para o *Ring Buffer* que guarda estas mesmas teclas numa *Ram* e envia para o *Output Buffer* que este é responsável por fazer a ligação com o *Control* enviando as teclas para o mesmo. Assim, sendo o diagrama de blocos do *Keyboard Reader* está disponível na Figura 17.

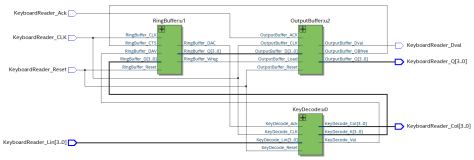


**Figura 17:** Diagrama de Blocos do módulo *Keyboard Reader*

# 5 Conclusões

Na conclusão deste módulo do trabalho, conseguimos processar e descodificar *inputs* do teclado, e enviar para o *software* através dos módulos *Key Decode*,*Ring Buffer* e *Output Buffer*. Apesar de em geral ter o funcionamento esperado, deparamonos com um erro durante o processo de leitura das teclas. O erro em questão causava que, durante o processo de leitura da tecla, os inputs eram processados muito mais do que uma vez, causando a ilusão de que a tecla estava a ser carregada repetidas vezes, ao invés de apenas uma.

# 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 KeyBoard Reader

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

entity 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 KeyboardReader;

architecture logicFunction of KeyboardReader is

        component KeyDecode is
                port(

                        -- input
                        KeyDecode_Ack: in std_logic;
                        KeyDecode_Lin: in std_logic_vector(3 downto 0);
                        KeyDecode_CLK: in std_logic;
                        KeyDecode_Reset: in std_logic;

                        -- output
                        KeyDecode_Col: out std_logic_vector(3 downto 0);
                        KeyDecode_Val: out std_logic;
                        KeyDecode_K: out std_logic_vector(3 downto 0)

                );
        end component;

        component RingBuffer is
                port(

                        -- input
                RingBuffer_D: in std_logic_vector(3 downto 0);
                RingBuffer_DAV: in std_logic;
                RingBuffer_CLK: in std_logic;
        RingBuffer_Reset: in std_logic;
                RingBuffer_CTS: in std_logic;

                -- output
                RingBuffer_Q: out std_logic_vector(3 downto 0);
                RingBuffer_Wreg: out std_logic;
                RingBuffer_DAC: out std_logic;

        debug_full: out std_logic;
        debug_empty: out std_logic

                );
```

```
        end component;

        component OutputBuffer is
                port(

                -- input
                OutputBuffer_D: in std_logic_vector(3 downto 0);
                OutputBuffer_Load: in std_logic;
                OutputBuffer_ACK: in std_logic;
        OutputBuffer_Reset: in std_logic;
        OutputBuffer_CLK: in std_logic;

                -- output
                OutputBuffer_Q: out std_logic_vector(3 downto 0);
                OutputBuffer_OBfree: out std_logic;
                OutputBuffer_Dval: out std_logic

                );
        end component;

        signal K_to_D: std_logic_vector(3 downto 0);
        signal Q_to_D: std_logic_vector(3 downto 0);
        signal DAC_to_Kack: std_logic;
        signal Kval_to_DAV: std_logic;
        signal OBfree_to_CTS: std_logic;
        signal Wreg_to_Load: std_logic;

begin

        u0: KeyDecode port map (
                KeyDecode_CLK => KeyboardReader_CLK,
                KeyDecode_Reset => KeyboardReader_Reset,
                KeyDecode_Lin => KeyboardReader_Lin,
                KeyDecode_Ack => DAC_to_Kack,
                KeyDecode_Col => KeyboardReader_Col,
                KeyDecode_Val => Kval_to_DAV,
                KeyDecode_K => K_to_D
        );

        u1: RingBuffer port map (
                RingBuffer_D => K_to_D,
                RingBuffer_DAV => Kval_to_DAV,
                RingBuffer_CLK => KeyboardReader_CLK,
        RingBuffer_Reset => KeyboardReader_Reset,
                RingBuffer_CTS => OBfree_to_CTS,
                RingBuffer_Q => Q_to_D,
                RingBuffer_Wreg => Wreg_to_Load,
                RingBuffer_DAC => DAC_to_Kack,

        debug_empty => debug_empty,
        debug_full => debug_full
        );

        u2: OutputBuffer port map (
                OutputBuffer_D => Q_to_D,
                OutputBuffer_Load => Wreg_to_Load,
                OutputBuffer_ACK => KeyboardReader_Ack,
        OutputBuffer_Reset => KeyboardReader_Reset,
        OutputBuffer_CLK => KeyboardReader_CLK,
                OutputBuffer_Q => KeyboardReader_Q,
                OutputBuffer_OBfree => OBfree_to_CTS,
                OutputBuffer_Dval => KeyboardReader_Dval
```

```
        );

end logicFunction;
```

## A.4 KeyBoard Decoder

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

entity KeyDecode is
        port(

                -- input
                KeyDecode_Ack: in std_logic;
                KeyDecode_Lin: in std_logic_vector(3 downto 0);
                KeyDecode_CLK: in std_logic;
                KeyDecode_Reset: in std_logic;

                -- output
                KeyDecode_Col: out std_logic_vector(3 downto 0);
                KeyDecode_Val: out std_logic;
                KeyDecode_K: out std_logic_vector(3 downto 0)
        );
end KeyDecode;

architecture logicFunction of KeyDecode is

        component KeyScan is
                port(

                        -- input
                        KeyScan_Lin: in std_logic_vector(3 downto 0);
                        KeyScan_Scan: in std_logic;
                        KeyScan_CLK: in std_logic;
                        KeyScan_Reset: in std_logic;

                        -- output
                        KeyScan_Col: out std_logic_vector(3 downto 0);
                        KeyScan_K: out std_logic_vector(3 downto 0);
                        KeyScan_Press: out std_logic

                );
        end component;

        component KeyControl is
                port(

                        -- input
                        KeyControl_Ack: in std_logic;
                        KeyControl_Press: in std_logic;
                        KeyControl_CLK: in std_logic;
                        KeyControl_Reset: in std_logic;

                        -- output
                        KeyControl_Val: out std_logic;
                        KeyControl_Scan: out std_logic

                );
        end component;

        component ClkDiv is
                port(
                        -- input
                clk_in: in std_logic;

                -- output
                clk_out: out std_logic
```

```
            );
    end component;

        signal Scan: std_logic;
        signal Press: std_logic;
        signal clk: std_logic;

begin

        u0: KeyScan port map (
                KeyScan_Lin => KeyDecode_Lin,
                KeyScan_Scan => Scan,
                KeyScan_CLK => clk,
                KeyScan_Reset => KeyDecode_Reset,
                KeyScan_Col => KeyDecode_Col,
                KeyScan_K => KeyDecode_K,
                KeyScan_Press => Press
        );

        u1: KeyControl port map (
                KeyControl_Ack => KeyDecode_Ack,
                KeyControl_Press => Press,
                KeyControl_CLK => clk,
                KeyControl_Reset => KeyDecode_Reset,
                KeyControl_Val => KeyDecode_Val,
                KeyControl_Scan => Scan
        );

        u2: ClkDiv port map (
                clk_in => KeyDecode_CLK,
                clk_out => clk
        );

end logicFunction;
```

## A.5 KeyBoard Scan

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

entity KeyScan is
        port(

                -- input
                KeyScan_Lin: in std_logic_vector(3 downto 0);
                KeyScan_Scan: in std_logic;
                KeyScan_CLK: in std_logic;
                KeyScan_Reset: in std_logic;

                -- output
                KeyScan_Col: out std_logic_vector(3 downto 0);
                KeyScan_K: out std_logic_vector(3 downto 0);
                KeyScan_Press: out std_logic

        );
end KeyScan;

architecture logicFunction of KeyScan is

        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;

        component Dec4 is
                port(

                        -- input
                        Dec4_S: in std_logic_vector(1 downto 0);

                        -- output
                        Dec4_Out: out std_logic_vector(3 downto 0)

                );
        end component;

        component Mux4 is
                port(

                        -- input
                        Mux4_In: in std_logic_vector(3 downto 0);
                        Mux4_S: in std_logic_vector(1 downto 0);

                        -- output
                        Mux4_Out: out std_logic

                );
        end component;
```

```
        signal Q: std_logic_vector(3 downto 0);
        signal Not_KeyScan_Col: std_logic_vector(3 downto 0);
        signal Not_KeyScan_Press: std_logic;

begin

        u0: Counter4 port map (
                Counter4_CE => KeyScan_Scan,
                Counter4_CLK => KeyScan_CLK,
                Counter4_Reset => KeyScan_Reset,
                Counter4_Q => Q
        );

        KeyScan_K <= Q;

        u1: Dec4 port map (
                Dec4_S(1) => Q(3),
                Dec4_S(0) => Q(2),
                Dec4_Out => Not_KeyScan_Col
        );

        KeyScan_Col(3) <= not Not_KeyScan_Col(3);
        KeyScan_Col(2) <= not Not_KeyScan_Col(2);
        KeyScan_Col(1) <= not Not_KeyScan_Col(1);
        KeyScan_Col(0) <= not Not_KeyScan_Col(0);

        u2: Mux4 port map (
                Mux4_In => KeyScan_Lin,
                Mux4_S(1) => Q(1),
                Mux4_S(0) => Q(0),
                Mux4_Out => Not_KeyScan_Press
        );

        KeyScan_Press <= not Not_KeyScan_Press;

end logicFunction;
```

## A.6 KeyBoard Control

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

entity KeyControl is
        port(

                -- input
                KeyControl_Ack: in std_logic;
                KeyControl_Press: in std_logic;
                KeyControl_CLK: in std_logic;
                KeyControl_Reset: in std_logic;

                -- output
                KeyControl_Val: out std_logic;
                KeyControl_Scan: out std_logic

        );
end KeyControl;

architecture logicFunction of KeyControl is

        type Tstate is (scanning, hold, resume);

        signal state: Tstate;
        signal next_state: Tstate;

begin

        state <= scanning when KeyControl_Reset = '1' else next_state when falling_edge(KeyControl_CLK);

        process(state, KeyControl_Ack, KeyControl_Press) begin
                case state is

                        when scanning =>
                                if KeyControl_Press = '1' then
                                        next_state <= hold;
                                else
                                        next_state <= scanning;
                                end if;

                        when hold =>
                                if KeyControl_Ack = '1' then
                                        next_state <= resume;
                                else
                                        next_state <= hold;
                                end if;

                        when resume =>
                                if KeyControl_Ack = '0' and KeyControl_Press = '0' then
                                        next_state <= scanning;
                                else
                                        next_state <= resume;
                                end if;

                end case;
        end process;

        KeyControl_Val <= '1' when state = hold else '0';
        KeyControl_Scan <= '1' when state = scanning and KeyControl_Press = '0' else '0';

end logicFunction;
```

## A.7 Decoder

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

entity Dec4 is
        port(

                -- input
                Dec4_S: in std_logic_vector(1 downto 0);

                -- output
                Dec4_Out: out std_logic_vector(3 downto 0)

        );
end Dec4;

architecture logicFunction of Dec4 is
begin

        Dec4_Out(0) <= not Dec4_S(1) and not Dec4_S(0);
        Dec4_Out(1) <= not Dec4_S(1) and     Dec4_S(0);
        Dec4_Out(2) <=     Dec4_S(1) and not Dec4_S(0);
        Dec4_Out(3) <=     Dec4_S(1) and     Dec4_S(0);

end logicFunction;
```

## A.8 Mux4

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

entity Mux4 is
        port(

                -- input
                Mux4_In: in std_logic_vector(3 downto 0);
                Mux4_S: in std_logic_vector(1 downto 0);

                -- output
                Mux4_Out: out std_logic

        );
end Mux4;

architecture logicFunction of Mux4 is
begin

        Mux4_Out <= (Mux4_In(0) and not Mux4_S(1) and not Mux4_S(0)) or
                                (Mux4_In(1) and not Mux4_S(1) and     Mux4_S(0)) OR
                                (Mux4_In(2) and     Mux4_S(1) and not Mux4_S(0)) OR
                                (Mux4_In(3) and     Mux4_S(1) and     Mux4_S(0));

end logicFunction;
```

## A.9 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.10 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.11 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.12 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.13 Ring Buffer

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

entity RingBuffer is
        port(

                -- input
                RingBuffer_D: in std_logic_vector(3 downto 0);
                RingBuffer_DAV: in std_logic;
                RingBuffer_CLK: in std_logic;
        RingBuffer_Reset: in std_logic;
                RingBuffer_CTS: in std_logic;

                -- output
                RingBuffer_Q: out std_logic_vector(3 downto 0);
                RingBuffer_Wreg: out std_logic;
                RingBuffer_DAC: out std_logic;

        debug_full: out std_logic;
        debug_empty: out std_logic

        );
end RingBuffer;

architecture logicFunction of RingBuffer is

        component RingBufferControl is
                port(

                -- input
                RingBufferControl_Reset: in std_logic;
                RingBufferControl_DAV: in std_logic;
                RingBufferControl_CTS: in std_logic;
                RingBufferControl_CLK: in std_logic;
                RingBufferControl_full: in std_logic;
        RingBufferControl_empty: in std_logic;

                -- output
                RingBufferControl_Wr: out std_logic;
        RingBufferControl_selPG: out std_logic;
        RingBufferControl_Wreg: out std_logic;
        RingBufferControl_DAC: out std_logic;
        RingBufferControl_incPut: out std_logic;
        RingBufferControl_incGet: out std_logic

                );
        end component;

        component MemoryAddressControl is
                port(

                -- input
                MemoryAddressControl_putget: in std_logic;
                MemoryAddressControl_CLK: in std_logic;
                MemoryAddressControl_incPut: in std_logic;
                MemoryAddressControl_incGet: in std_logic;
        MemoryAddressControl_Reset: in std_logic;

                -- output
                MemoryAddressControl_Address: out std_logic_vector(3 downto 0);
                MemoryAddressControl_full: out std_logic;
```

```
                MemoryAddressControl_empty: out std_logic

                );
        end component;

        component RAM is
                generic(
                ADDRESS_WIDTH : natural := 4;
                DATA_WIDTH : natural := 4
        );
        port(
                address : in std_logic_vector(ADDRESS_WIDTH - 1 downto 0);
                wr: in std_logic;
                din: in std_logic_vector(DATA_WIDTH - 1 downto 0);
                dout: out std_logic_vector(DATA_WIDTH - 1 downto 0)
        );
        end component;

        signal full: std_logic;
    signal empty: std_logic;
    signal Wr: std_logic;
    signal selPG: std_logic;
    signal incPut: std_logic;
    signal incGet: std_logic;
        SIGNAL Address: std_logic_vector(3 downto 0);

begin

        u0: RingBufferControl port map (

                RingBufferControl_Reset => RingBuffer_Reset,
                RingBufferControl_DAV => RingBuffer_DAV,
                RingBufferControl_CTS => RingBuffer_CTS,
                RingBufferControl_CLK => RingBuffer_CLK,
                RingBufferControl_full => full,
        RingBufferControl_empty => empty,
                RingBufferControl_Wr => Wr,
        RingBufferControl_selPG => selPG,
        RingBufferControl_Wreg => RingBuffer_Wreg,
        RingBufferControl_DAC => RingBuffer_DAC,
        RingBufferControl_incPut => incPut,
        RingBufferControl_incGet => incGet

        );

        u1: MemoryAddressControl port map (

                MemoryAddressControl_putget => selPG,
                MemoryAddressControl_CLK => RingBuffer_CLK,
                MemoryAddressControl_incPut => incPut,
                MemoryAddressControl_incGet => incGet,
        MemoryAddressControl_Reset => RingBuffer_Reset,
                MemoryAddressControl_Address => Address,
                MemoryAddressControl_full => full,
                MemoryAddressControl_empty => empty

        );

        u2: RAM port map (

                address => Address,
                wr => Wr,
```

```
                din => RingBuffer_D,
                dout => RingBuffer_Q

        );

    debug_full <= full;
    debug_empty <= empty;

end logicFunction;
```

## A.14 Ring Buffer Control

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

entity RingBufferControl is
        port(

                -- input
        RingBufferControl_Reset: in std_logic;
                RingBufferControl_DAV: in std_logic;
                RingBufferControl_CTS: in std_logic;
                RingBufferControl_CLK: in std_logic;
                RingBufferControl_full: in std_logic;
        RingBufferControl_empty: in std_logic;

                -- output
                RingBufferControl_Wr: out std_logic;
        RingBufferControl_selPG: out std_logic;
        RingBufferControl_Wreg: out std_logic;
        RingBufferControl_DAC: out std_logic;
        RingBufferControl_incPut: out std_logic;
        RingBufferControl_incGet: out std_logic

        );
end RingBufferControl;

architecture logicFunction of RingBufferControl is

    type Tstate is (init, kd_addr, kd_write,kd_inc, kd_ack, ob_inc, ob_read);
    -- type Tstate is (start, writing, Wr_activate, incPut_activate, accepted_data, reading,incGet_activate);

    signal state: Tstate;
    signal next_state: Tstate;

begin

    state <= init when RingBufferControl_Reset = '1' else next_state when rising_edge(RingBufferControl_CLK);

    process(state, RingBufferControl_DAV, RingBufferControl_CTS, RingBufferControl_full, RingBufferControl_empty) begin
        case state is

            -- KD

            when init =>
                if RingBufferControl_DAV = '1' and RingBufferControl_full = '0' then
                    next_state <= kd_addr;
                elsif RingBufferControl_CTS = '1' and RingBufferControl_empty = '0' then
                    next_state <= ob_read;
                else
                    next_state <= init;
                end if;

            when kd_addr => next_state <= kd_write;
            when kd_write => next_state <= kd_inc;
            when kd_inc => next_state <= kd_ack;

            when kd_ack =>
                if RingBufferControl_DAV = '0' then
                   next_state <= init;
                else
                    next_state <= kd_ack;
                end if;
```

```
            -- OB

            when ob_read =>
                if RingBufferControl_CTS = '1' then
                    next_state <= ob_read;
                else
                    next_state <= ob_inc;
                end if;

            when ob_inc => next_state <= init;

        end case;
    end process;

    RingBufferControl_Wr <= '1' when state = kd_write else '0';
    RingBufferControl_selPG <= '1' when state = kd_addr or state = kd_write else '0';
    RingBufferControl_Wreg <= '1' when state = ob_read else '0';
    RingBufferControl_DAC <= '1' when state = kd_ack else '0';
    RingBufferControl_incPut <= '1' when state = kd_inc else '0';
    RingBufferControl_incGet <= '1' when state = ob_inc else '0';


end logicFunction;
```

## A.15 Memory Address Control

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

entity MemoryAddressControl is
        port(

                -- input
                MemoryAddressControl_putget: in std_logic;
                MemoryAddressControl_CLK: in std_logic;
                MemoryAddressControl_incPut: in std_logic;
                MemoryAddressControl_incGet: in std_logic;
        MemoryAddressControl_Reset: in std_logic;

                -- output
                MemoryAddressControl_Address: out std_logic_vector(3 downto 0);
                MemoryAddressControl_full: out std_logic;
                MemoryAddressControl_empty: out std_logic

        );
end MemoryAddressControl;

architecture logicFunction of MemoryAddressControl is

        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;

        component CountUpDown is
                port(

                -- input
                CountUpDown_Up: in std_logic;
                CountUpDown_Down: in std_logic;
                CountUpDown_CLK: in std_logic;
                CountUpDown_Reset: in std_logic;

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

                );
        end component;

        component Mux4_MAC is
                port(

        -- input
                Mux4_MAC_A: in std_logic_vector(3 downto 0);
        Mux4_MAC_B: in std_logic_vector(3 downto 0);
                Mux4_MAC_S: in std_logic;

                -- output
```

```
                Mux4_MAC_Out: out std_logic_vector(3 downto 0)

                );
        end component;

        signal incPut: std_logic;
        signal incGet: std_logic;

    signal get_addr: std_logic_vector(3 downto 0);
    signal put_addr: std_logic_vector(3 downto 0);
    signal size: std_logic_vector(4 downto 0);

        signal full: std_logic;
        signal empty: std_logic;

begin

        incPut <= '1' when MemoryAddressControl_incPut = '1' and full = '0' else '0';
        incGet <= '1' when MemoryAddressControl_incGet = '1' and empty = '0' else '0';

        u0: Counter4 port map (
                Counter4_CE => incGet,
                Counter4_CLK => MemoryAddressControl_CLK,
                Counter4_Reset => MemoryAddressControl_Reset,
                Counter4_Q => get_addr
        );

        u1: Counter4 port map (
                Counter4_CE => incPut,
                Counter4_CLK => MemoryAddressControl_CLK,
                Counter4_Reset => MemoryAddressControl_Reset,
                Counter4_Q => put_addr
        );

        u2: CountUpDown port map (
                CountUpDown_Up => incPut,
                CountUpDown_Down => incGet,
                CountUpDown_CLK => MemoryAddressControl_CLK,
                CountUpDown_Reset => MemoryAddressControl_Reset,
                CountUpDown_Q => size
        );

        u3: Mux4_MAC port map (
                Mux4_MAC_A => put_addr,
        Mux4_MAC_B => get_addr,
                Mux4_MAC_S => MemoryAddressControl_putget,
                Mux4_MAC_Out => MemoryAddressControl_Address
        );

    full <= '1' when size = "10000" else '0';
        MemoryAddressControl_full <= full;

    empty <= '1' when size = "00000" else '0';
        MemoryAddressControl_empty <= empty;

end logicFunction;
```

## A.16 Mux4-MAC

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

entity Mux4_MAC is
        port(

                -- input
                Mux4_MAC_A: in std_logic_vector(3 downto 0);
        Mux4_MAC_B: in std_logic_vector(3 downto 0);
                Mux4_MAC_S: in std_logic;

                -- output
                Mux4_MAC_Out: out std_logic_vector(3 downto 0)

        );
end Mux4_MAC;

architecture logicFunction of Mux4_MAC is
begin

        Mux4_MAC_Out(0) <= (Mux4_MAC_A(0) and Mux4_MAC_S) or (Mux4_MAC_B(0) and not Mux4_MAC_S);
    Mux4_MAC_Out(1) <= (Mux4_MAC_A(1) and Mux4_MAC_S) or (Mux4_MAC_B(1) and not Mux4_MAC_S);
    Mux4_MAC_Out(2) <= (Mux4_MAC_A(2) and Mux4_MAC_S) or (Mux4_MAC_B(2) and not Mux4_MAC_S);
    Mux4_MAC_Out(3) <= (Mux4_MAC_A(3) and Mux4_MAC_S) or (Mux4_MAC_B(3) and not Mux4_MAC_S);

end logicFunction;
```

## A.17 RAM

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- Project     : DE10-Lite
-- Affiliations: DEETC, ISEL - IPL
-- Funding     : -
--------------------------------------------------------------------------------
-- File        : RAM.vhd
-- Author(s)   : Pedro Miguens Matutino
-- Date        : 2023/01/03
--------------------------------------------------------------------------------
-- Copyright (c) 2023 Pedro Miguens Matutino
--------------------------------------------------------------------------------
-- Description :
-- .
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity RAM is
        generic(
                ADDRESS_WIDTH : natural := 4;
                DATA_WIDTH : natural := 4
        );
        port(
                address : in std_logic_vector(ADDRESS_WIDTH - 1 downto 0);
                wr: in std_logic;
                din: in std_logic_vector(DATA_WIDTH - 1 downto 0);
                dout: out std_logic_vector(DATA_WIDTH - 1 downto 0)
        );
end RAM;

architecture behavioral of RAM is

        type RAM_TYPE is array (0 to 2**ADDRESS_WIDTH - 1) of std_logic_vector(DATA_WIDTH - 1 downto 0);
        signal ram : RAM_TYPE;

begin

        process(address, wr)
        begin
                dout <= ram(conv_integer(unsigned(address)));

                if (wr='1') then
                        ram(conv_integer(unsigned(address))) <= din;
                end if;
        end process;

end behavioral;
```

## A.18 Output Buffer

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

entity OutputBuffer is
        port(

                -- input
                OutputBuffer_D: in std_logic_vector(3 downto 0);
                OutputBuffer_Load: in std_logic;
                OutputBuffer_ACK: in std_logic;
        OutputBuffer_Reset: in std_logic;
        OutputBuffer_CLK: in std_logic;

                -- output
                OutputBuffer_Q: out std_logic_vector(3 downto 0);
                OutputBuffer_OBfree: out std_logic;
                OutputBuffer_Dval: out std_logic

        );
end OutputBuffer;

architecture logicFunction of OutputBuffer is

        component BufferControl is
                port(

                        -- input
                        BufferControl_Reset: in std_logic;
                    BufferControl_clk: in std_logic;
                    BufferControl_Load: in std_logic;
            BufferControl_ACK: in std_logic;

                    -- output
                    BufferControl_OBfree: out std_logic;
                    BufferControl_Wreg: out std_logic;
            BufferControl_Dval: out std_logic

                );
        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 Wreg: std_logic;

begin

        u0: BufferControl port map (
                        BufferControl_Reset => OutputBuffer_Reset,
```

```
                BufferControl_clk => OutputBuffer_CLK,
                BufferControl_Load => OutputBuffer_Load,
            BufferControl_ACK => OutputBuffer_ACK,
                BufferControl_OBfree => OutputBuffer_OBfree,
                BufferControl_Wreg => Wreg,
            BufferControl_Dval => OutputBuffer_Dval
        );

        u1: Reg4 port map (
                Reg4_In => OutputBuffer_D,
                Reg4_CE => '1',
                Reg4_CLK => Wreg,
                Reg4_Reset => OutputBuffer_Reset,
                Reg4_Out => OutputBuffer_Q
        );


end logicFunction;
```

## A.19 BufferControl

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

entity BufferControl is
        port(

                -- input
                BufferControl_Reset: in std_logic;
                BufferControl_clk: in std_logic;
                BufferControl_Load: in std_logic;
        BufferControl_ACK: in std_logic;

                -- output
                BufferControl_OBfree: out std_logic;
                BufferControl_Wreg: out std_logic;
        BufferControl_Dval: out std_logic

        );
end BufferControl;

architecture logicFunction of BufferControl is

        type Tstate is (start, data_available, ack_received);

    signal state: Tstate := start;
    signal next_state: Tstate := start;

begin

        state <= start when BufferControl_Reset = '1' else next_state when rising_edge(BufferControl_clk);

        process(state, BufferControl_Load, BufferControl_ACK) begin

        case state is

            when start =>
                if BufferControl_Load = '1'then
                    next_state <= data_available;
                else
                    next_state <= start;
                end if;

            when data_available =>
                                if BufferControl_ACK = '1' then
                                        next_state <= ack_received;
                                        else
                                        next_state <= data_available;
                                        end if;

            when ack_received =>
                                if BufferControl_ACK = '1' then
                                        next_state <= ack_received;
                                        else
                                        next_state <= start;
                                end if;
            end case;

        end process;

        BufferControl_Wreg <= '1' when (state = start and BufferControl_Load = '1') else '0';
        BufferControl_OBfree <= '1' when (state = start) else '0';
```

```
    BufferControl_Dval <= '1' when (state = data_available) 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 KBD

**Algoritmo 2:** KBD

```
import isel.leic.utils.Time

// Ler teclas. Fun   es retornam '0'..'9','A'..'D','#','*' ou NONE
object KBD {

    private const val NONE = 0.toChar()

    private const val KEYPAD = "147*2580369#ABCD"

    private const val KEY_MASK = 0b00001111 // input
    private const val VAL_MASK = 0b00010000 // input

    private const val ACK_MASK = 0b10000000 // output

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

        // Garantir que nenhuma tecla    acknowledged no   nicio
        HAL.clearBits(ACK_MASK)

    }

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

        // Verificar se existe alguma tecla premida
        if(!HAL.isBit(VAL_MASK)) return NONE

        // Receber tecla premida
        val i = HAL.readBits(KEY_MASK)
        check(i in 0 until KEYPAD.length) { "Invalid key code (0x${Integer.toHexString(i)})" }

        // Acknowledge
        HAL.setBits(ACK_MASK)
        while(HAL.isBit(VAL_MASK)) {} // Esperar que acknowledge seja recebido
        HAL.clearBits(ACK_MASK)

        return KEYPAD[i]
    }

    /**
     * Retorna a tecla premida ou `NONE` caso o `timeout` passe.
     * @property timeout Milissegundos at   deixar de esperar.
     */
    fun waitKey(timeout: Long): Char {

        val end = Time.getTimeInMillis() + timeout

        while(Time.getTimeInMillis() < end){
            val key = getKey()
            if(key != NONE) return key
        }

        return NONE
    }

}

fun main(){

    KBD.init()
//    println("start")
//    Time.sleep(10000)
//    println("print")
    while(true)
        print(KBD.waitKey(60_000L))

}
```

## B.3 TUI

**Algoritmo 3:** KBD

```
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)

    }

}
```