O módulo *Keyboard Reader* é constituído por três blocos principais: *i*) o descodificador de teclado (*Key Decode*); *ii*) o bloco de armazenamento (designado por *Ring Buffer*); e *iii*) o bloco de entrega ao consumidor (designado por *Output Buffer*), de acordo com o diagrama representado na Figura 1. Neste caso, o módulo *Control*, implementado em software, é a entidade consumidora. Para esta primeira fase, o *Keyboard Reader* tem apenas o bloco de descodificação implementado (*Key Decode*) Figura 2a, fazendo diretamente a ligação com a entidade consumidora através do *UsbPort*.

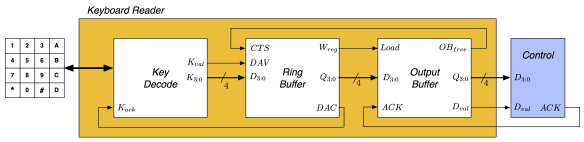


Figura 1: Diagrama de blocos do módulo *Keyboard Reader*

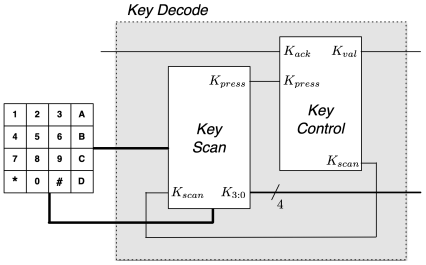
1 Keyboard Reader

1.1 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 Scanner*, 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 2a.

O controlo de fluxo de saída do bloco *Key Decode*, (para o módulo *Control*), 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 2b.

O bloco *Key Scanner* foi implementado de acordo com o diagrama de blocos representado na Figura 3. Para este bloco, foi escolhida a 3ª versão, devido ao menor número de clocks (4) necessários para fazer o varrimento das 16 teclas presentes no teclado.



(a) Diagrama de blocos do módulo *Key Decode*

(b) Diagrama temporal

Figura 2: Bloco *Key Decode*

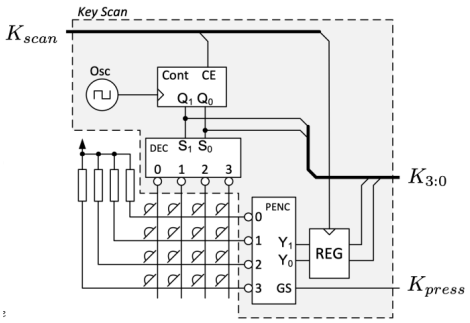


Figura 3: Diagrama de blocos do módulo *Key Scanner*

O bloco *Key Control* foi implementado pela máquina de estados representada em *ASM-chart* na Figura 4, de acordo com o funcionamento esperado do KeyDecode, acima mencionado.

A descrição hardware dos blocos *Key Reader, Key Decode, Key Scanner e Key Control* em *VHDL* encontra-se nos anexos VHDL.

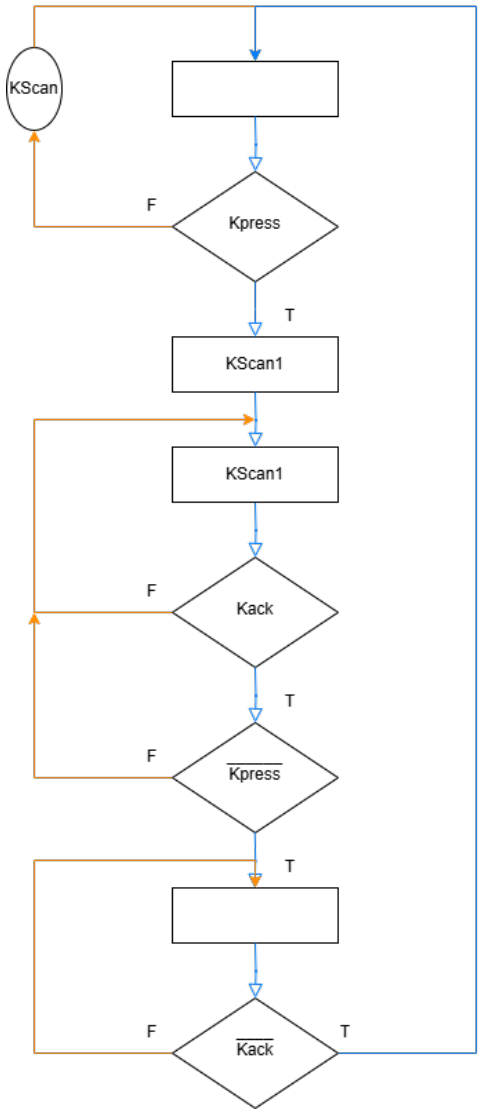


Figura 4: Máquina de estados do *Key Control*.
As linhas a laranja e azul representam o valor *false* e *true* respetivamente.

2 Control

Os módulos de Controlo são responsáveis pela lógica de alto nível, gerindo o estado do sistema, processando os dados e enviando informações para apresentar no LCD. Este tem como entradas um sinal de *input* válido do teclado, assim como a tecla correspondente a esse *input*, tendo por sua vez o dever de atualizar o estado do jogo (ack) de forma apropriada e comunicar com outros módulos a informação devida. Atua essencialmente como o "cérebro" do sistema, implementando o comportamento específico do jogo.

2.1 UsbPort

O *UsbPort* é a nossa forma de comunicação entre o teclado matricial e o módulo de *Control*. Esta ligação teve as entradas e saídas definidas de acordo com o ficheiro de texto *hardware.simul* Figura5 presente no projeto em *kotlin*.

```
#Generic modules to activate from Simulator
UsbPort = UsbPort
kbd = Keyboard("123A456B789C*0#D",4,4,0)
lcd     = LCD

#-------------------------------------
#KeyBoard Reader
#-------------------------------------
1 -> kbd.oe
kbd.K[0-3] -> UsbPort.I[1-4]
kbd.val -> UsbPort.I0
UsbPort.O0 -> kbd.ack

#-------------------------------------
#LCD
#-------------------------------------
#Serial Receiver
UsbPort.O[1-4] -> lcd.D[4-7]
UsbPort.O6 -> lcd.rs
UsbPort.O7 -> lcd.e
```

Figura 5: Configurações para a Simulação

2.2 HAL

O *HAL* é uma camada de abstração que permite separar as camadas superiores do software dos detalhes específicos do hardware usado. É no *HAL* que temos as ferramentas para usar os *bits* que entram e saem do *UsbPort*, ferramentas estas como *writeBits()*, *readBits()* fazer ref para o código do *HAL*. Ao encapsular estas funcionalidades, a *HAL* permite que os restantes módulos (como o *Scanner* do teclado ou o *Control* do jogo) interajam com o hardware de forma simplificada, sem depender dos detalhes técnicos do hardware específico utilizado. Esta abordagem promove a portabilidade do código e facilita a manutenção e evolução do sistema.

2.3 KBD

O *KBD* é a correspondência em *software* ao teclado matricial que queremos implementar. É aqui que decidimos os caracteres que queremos reproduzir no *display* e como é que chegamos ao código representado por estes. Dentro do KBD temos as funções *getKey()* e *waitKey()*, responsáveis por este processo, fazendo uso do HAL para reconhecer o código

da tecla premida no teclado *hardware*, simulado ou não, e fazendo uma correspondência ao seu caracter.

2.4 LCD

Este ficheiro define um objeto que controla o LCD através de várias funções para o iniciar, escrever dados ou comandos e manipular o cursor. A função *init()* inicializa o display, configurando-o primeiro no modo de 8 bits e depois no modo de 4 bits. As função *write(c:Char)* escreve um único caracter convertendo-o para inteiro e chamando a função *writeDATA()*. A *write(text:String)* escreve uma sequência de caracteres através da função anterior. A função *writeDATA()* chama a função *writeBYTE()* com o argumento rs a true, tal como a função writeCMD chama a função *writeBYTE()* com o argumento rs a false, distinguindo-se um do outro através do valor de rs. A função writeByte() escreve Bytes dividindo-os em Nibbles de 4 bits, escrevendo cada nibble separadamente. A função *writeNibble()* escreve um nibble de comando ou dados no LCD chamando a função correta dependendo do SERIALINTERFACE, seja em paralelo ou em série. Ainda não foi implementada a escrita em série. A função *writeNibbleParallel()* começa por definir o tipo de instrução que vai escrever, comando ou dados, ativa o enable e depois escreve o suposto comando ou dado de 4 bits, desativando o enable após a escrita da instrução. A função *cursor()* manda uma linha e um comando para posicionar o cursor no display. A função *clear()* envia um comando definido nas configurações do LCD que limpa o display.

2.5 TUI

Neste momento, o TUI é a nossa entidade que gere o LCD e o KBD, de forma a escrever com uma maior facilidade o que for pretendido no display. Por enquanto, foi feito uso dele de forma a testar o bom funcionamento dos outros módulos acima referidos e fazer o chamamento de instruções de uma forma mais abstracta, facilitando o uso dos outros módulos.

3 Conclusões

Concluindo, até ao dado momento, foi realizado um dado número de módulos de forma a termos uma componente em hardware, correspondente ao descodificamento teclado matricial 4x4, que pretendíamos implementar e fez-se a sua ligação ao *Control* capaz de com a informação descodificada, apresentar a tecla premida no teclado no *display*. Isto mostra que tendo um *Key Decode* funcional, mesmo não tendo os outros módulos pertencentes ao *KeyBoard Reader*, podemos fazer uso dele de forma a ter o comportamento pretendido. No futuro, prevê-se a implementação dos restantes módulos *Hardware* apresentados na introdução, assim como os restantes módulos *Software* de forma a apresentar um melhor funcionamento do sistema.

A VHDL

A.1 roulette.vhd

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


-- This is the top-level entity responsible for running the actual program
entity roulette is

        port (
                Liness: in std_logic_vector(3 downto 0);
                CLK   : in std_logic;
                Reset : in std_logic;

                LCD_RS  :       out std_logic;
                LCD_EN  :       out std_logic;
                LCD_DATA:   out std_logic_vector(7 downto 4);

                Colss   :       out std_logic_vector(3 downto 0);
                Kout    :       out std_logic_vector(3 downto 0);
                Kval    :       out std_logic
        );
end roulette;

architecture structural of roulette is

        component Key_decode is
            port (
                        Kack    :       in std_logic;
                        Liness  :       in std_logic_vector(3 downto 0);
                        CLK         :   in std_logic;
                        Reset   :       in std_logic;

                        Colss   :       out std_logic_vector(3 downto 0);
                        Kout    :       out std_logic_vector(3 downto 0);
                        Kval    :       out std_logic
                );
        end component;

        component UsbPort is
            port (
                        inputPort       :  IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
                        outputPort      :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0)
                );
        end component;

        signal sig_k3_0     : std_logic_vector(3 downto 0);
        signal sig_kscan        : std_logic_vector(1 downto 0);
```

```
signal sig_cols          : std_logic_vector(3 downto 0);
signal sig_kval          : std_logic;

-- Valores n o atribuidos correspondentes aos bits 5-7 do input do UsbPort
-- Este sinal foi criado para n o haver erros associados ao Usbport ao abrir o
signal inputSignal : std_logic := '0';

signal sig_d7_4      : std_logic_vector(3 downto 0);
signal sig_enable        : std_logic;
signal sig_rs            : std_logic;

signal sig_kack_outusbport : std_logic;


begin

keydecode:       Key_decode port map (
        Kack             => sig_kack_outusbport,
        Liness   => Liness,
        CLK      =>      CLK,
        Reset    =>      Reset,
        Colss    => sig_cols,
        Kout     => sig_k3_0,
        Kval     => sig_kval
);


usbPortVHD:              UsbPort port map (
        inputPort(0)     => sig_kval,
        inputPort(1)     => sig_k3_0(0),
        inputPort(2)     => sig_k3_0(1),
        inputPort(3)     => sig_k3_0(2),
        inputPort(4)     => sig_k3_0(3),

        -- Valores n o atribuidos
        inputPort(5)     => inputSignal,
        inputPort(6)     => inputSignal,
        inputPort(7)     => inputSignal,

        outputPort(0)    => sig_kack_outusbport,
        outputPort(1)    => sig_d7_4(0),
        outputPort(2)    => sig_d7_4(1),
        outputPort(3)    => sig_d7_4(2),
        outputPort(4)    => sig_d7_4(3),

        -- Valores n o atribuidos
        outputPort(5)  => inputSignal,

        outputPort(6)    => sig_rs,
        outputPort(7)    => sig_enable
```

```
        );

        LCD_RS   <= sig_rs;
        LCD_EN   <= sig_enable;
        LCD_DATA <= sig_d7_4;

        Kval  <= sig_kval;
        Kout  <= sig_k3_0;
        Colss <= sig_cols;

end structural;
```

A.2 Key_reader.vhd

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

entity Key_reader is
        port (
                Kack    : in std_logic;
                Liness  : in std_logic_vector(3 downto 0);
                CLK             : in std_logic;
                Reset   : in std_logic;

                Colss   : out std_logic_vector(3 downto 0);
                Kout    : out std_logic_vector(3 downto 0);
                Kval    : out std_logic
        );
end Key_reader;

architecture structural of Key_reader is

        component key_decode is
                port (
                        Kack    : in std_logic;
                        Liness  : in std_logic_vector(3 downto 0);
                        CLK             : in std_logic;
                        Reset   : in std_logic;

                        Colss   : out std_logic_vector(3 downto 0);
                        Kout    : out std_logic_vector(3 downto 0);
                        Kval    : out std_logic
                );
        end component;

        -- Sinais internos
        signal sig_kpress : std_logic;
        signal sig_k3     : std_logic_vector(3 downto 0);
        signal sig_cols   : std_logic_vector(3 downto 0);
        signal sig_kval   : std_logic;

        begin

        keydecode: key_decode port map (
                Kack    => Kack,
                Liness => Liness,
                CLK     => CLK,
                Reset   => Reset,
                Colss   => sig_cols,
                Kout    => sig_k3,
                Kval    => sig_kval
        );
```

```
        Kval  <= sig_kval;
        Kout  <= sig_k3;
        Colss <= sig_cols;

end structural;
```

A.3 KEY_DECODE/Key_decode.vhd

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

entity Key_decode is
        port (
                Kack      : in std_logic;
                Liness    : in std_logic_vector(3 downto 0);
                CLK           : in std_logic;
                Reset     :     in std_logic;

                Colss     : out std_logic_vector(3 downto 0);
                Kout      : out std_logic_vector(3 downto 0);
                Kval      : out std_logic
        );
end Key_decode;

architecture structural of Key_decode is

        component key_scanner is
            port (
                        KScan : in std_logic_vector(1 downto 0);
                        lines : in std_logic_vector(3 downto 0);
                        CLK   : in std_logic;
                        Reset : in std_logic;

                        columns : out std_logic_vector(3 downto 0);
                        KPress  : out std_logic;
                        K               : out std_logic_vector(3 downto 0)
                );
        end component;

        component KeyControl is
                port (
                   reset          : in std_logic;
                        clk             : in std_logic;
                        Kpress  : in std_logic;
                        Kack    : in std_logic;

                        KScan   : out std_logic_vector(1 downto 0);
                        Kval    : out std_logic
                );
        end component;


        signal sig_kpress   : std_logic;
        signal sig_k3           : std_logic_vector(3 downto 0);
        signal sig_kscan        : std_logic_vector(1 downto 0);
        signal sig_cols         : std_logic_vector(3 downto 0);
```

```
        signal sig_kval          : std_logic;


        begin

        keyscan: key_scanner port map (
                KScan   => sig_kscan,
                lines   => Liness,
                CLK     =>      CLK,
                Reset   =>      Reset,
                columns => sig_cols,
                KPress  => sig_kpress,
                K               => sig_k3
        );

        keycontrols: KeyControl port map (
                reset   => Reset,
                clk     => CLK,
                Kpress  => sig_kpress,
                Kack    => Kack,
                KScan   => sig_kscan,
                Kval    => sig_kval
        );


        Kval  <= sig_kval;
        Kout  <= sig_k3;
        Colss <= sig_cols;

end structural;
```

A.4 KEY_DECODE/KeyControl/KeyControl.vhd

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

-- This is the entity is the state machine that controls the keydecode file
entity KeyControl is

        port (
                reset                   : in std_logic;
                    clk                 : in std_logic;
                    Kpress      : in std_logic;
                    Kack            : in std_logic;

                    KScan           : out std_logic_vector(1 downto 0);
                    Kval            : out std_logic
        );

end KeyControl;

architecture behavioral of KeyControl is

type STATE_TYPE is (State_Scan_Columns, State_Scan_Lines, State_Val, State_Ack_Waiting);

signal CurrentState, NextState: STATE_TYPE;

begin

-- Flip-Flop's
CurrentState <= State_Scan_Columns when RESET = '1' else NextState when rising_edge(clk)

-- Generate Next State
GenerateNextState:
process (CurrentState, Kpress, Kack)
    begin
    case CurrentState is

        when State_Scan_Columns =>      if (Kpress = '1') then
                                            NextState <= State_Scan_Lines;
                                    else
                                            NextState <= State_Scan_Columns;
                                    end if;

        when State_Scan_Lines   =>  NextState <= State_Val;

            when State_Val              =>      if (Kack = '1' and Kpress = '0') then
                                            NextState <= State_Ack_Waiting;
                                    else
                                            NextState <= State_Val;
                                    end if;
```

```
            when State_Ack_Waiting      =>
                                        if (Kack = '0') then
                                                    NextState <= State_Scan_Columns;
                                        else
                                                    NextState <= State_Ack_Waiting;
                                        end if;
    end case;
end process;

-- Generate outputs


KScan(0)        <= '1' when (CurrentState = State_Scan_Columns and Kpress = '0') else
KScan(1)        <= '1' when (CurrentState = State_Scan_Lines) else '0';
Kval            <= '1' when (CurrentState = State_Val) else '0';

end behavioral;
```

A.5 KEY_DECODE/KEY_SCANNER/key_scanner.vhd

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

-- This entity defines the key scanner circuit
entity key_scanner is

    port (
                KScan   : in std_logic_vector(1 downto 0);
                lines   : in std_logic_vector(3 downto 0);
                CLK     : in std_logic;
                Reset   : in std_logic;

                columns : out std_logic_vector(3 downto 0);
                KPress  : out std_logic;
                K               : out std_logic_vector(3 downto 0)
    );

end key_scanner;


architecture structural of key_scanner is

        component counter is
                port (
                        CE              : in std_logic;
                        CLK     : in std_logic;
                        reset   : in std_logic;

                        parallel_load_flag  : in std_logic;
                        parallel_load_value : in std_logic_vector(1 downto 0);

                        count : out std_logic_vector(1 downto 0)
                );
        end component;

        component decoder_2x4 is
                port (
                        S : in std_logic_vector (1 downto 0);
                        Y : out std_logic_vector (3 downto 0)
                );
        end component;

        component priority_encoder is
                port (
                        A       : in std_logic_vector(3 downto 0);

                        Y       : out std_logic_vector(1 downto 0);
                        GS      : out std_logic
```

```
        );
end component;

component register_2bits is
        port (
                CLK    : in std_logic;
                RESET  : in std_logic;
                SET    : in std_logic;
                D      : in std_logic_vector(1 downto 0);
                EN     : in std_logic;
                Q      : out std_logic_vector(1 downto 0)
        );
end component;


signal counter_to_decoder: std_logic_vector(1 downto 0);
signal penc_to_reg               : std_logic_vector(1 downto 0);
signal cols                                 : std_logic_vector(3 downto 0);
signal penc_to_kpress     : std_logic;
signal K_low              : std_logic_vector(1 downto 0);
signal lines_signal       : std_logic_vector(3 downto 0);


begin


Contador: counter port map (
        CE => KScan(0),
        CLK => CLK,
        reset => Reset,
        parallel_load_flag => '0',
        parallel_load_value => "01",
        count => counter_to_decoder
);

Decoder: decoder_2x4 port map (
        S => counter_to_decoder,
        Y(0) => cols(0),
        Y(1) => cols(1),
        Y(2) => cols(2),
        Y(3) => cols(3)
);

lines_signal <= not lines;

Penc: priority_encoder port map (
        A(0)    => lines_signal(0),
        A(1)    => lines_signal(1),
        A(2)    => lines_signal(2),
        A(3)    => lines_signal(3),
```

```
            Y => penc_to_reg,
            GS => penc_to_kpress
        );

        Reg: register_2bits port map (
            CLK     => KScan(1),
            RESET => Reset,
            SET     =>      '0',
            D       => penc_to_reg,
            EN      => '1',
            Q       => K_low
        );

        K(3) <= counter_to_decoder(1);
        K(2) <= counter_to_decoder(0);
        K(1) <= K_low(1);
        K(0) <= K_low(0);
        columns <= not cols;
        KPress  <= penc_to_kpress;

end structural;
```

A.6 KEY_DECODE/KEY_SCANNER/COUNTER/counter.vhd

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

entity counter is

    port (
            CE    : in std_logic;
                  CLK   : in std_logic;
                  reset : in std_logic;

                  parallel_load_flag  : in std_logic;
                  parallel_load_value : in std_logic_vector(1 downto 0);

                  count : out std_logic_vector(1 downto 0)
    );

end counter;


architecture behavioral of counter is


        component adder_2bits is

                port (
                        A : in std_logic_vector(1 downto 0);
                        B : in std_logic_vector(1 downto 0);

                        carry_in  : in std_logic;
                        carry_out : out std_logic;

                        result : out std_logic_vector(1 downto 0)
                );

        end component;


        component mux_2x1 is

                port (
                        A : in std_logic_vector(1 downto 0);
                        B : in std_logic_vector(1 downto 0);

                        selector : in std_logic;

                        result : out std_logic_vector(1 downto 0)
                );
```

```
        end component;


        -- Imports the 2bit registry
        component register_2bits is

                port (
        CLK     : in std_logic;
        RESET   : in std_logic;
        SET     : in std_logic;
        D       : in std_logic_vector(1 downto 0);
        EN      : in std_logic;
        Q       : out std_logic_vector(1 downto 0)
                );

        end component;


        -- The carry out signal for the 2 bit adder
        signal carry_out_adder_2bits : std_logic_vector(1 downto 0);
        signal result_adder_2bits    : std_logic_vector(1 downto 0);
        signal result_mux                       : std_logic_vector(1 downto 0);
        signal result_register_2bits : std_logic_vector(1 downto 0);

begin

        -- Instantiates the 2 bit adder that always adds 1
        instance_adder_2bits : adder_2bits

                port map (
                        A => result_register_2bits,
                        B => "01",
                        carry_in => '0',
                        carry_out => open,

                        result => result_adder_2bits
                );


        -- Instantiates the MUX
        instance_mux_2x1 : mux_2x1

                port map (
                        A => result_adder_2bits,
                        B => parallel_load_value,

                        selector => parallel_load_flag,
                        result   => result_mux
                );
```

```
        -- Instantiates the 4 bit register
        instance_registry_2bits : register_2bits

                port map (
                        CLK    => CLK,
                        RESET  => reset,
                        SET    => '1',
                        D      => result_mux,
                        EN     => CE,
                        Q      => result_register_2bits
                );

                count <= result_register_2bits;

end behavioral;
```

A.7 KEY_DECODE/KEY_SCANNER/COUNTER/adder_2bits.vhd

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

entity adder_2bits is

        port (
                A : in std_logic_vector(1 downto 0);
                B : in std_logic_vector(1 downto 0);

                carry_in  : in std_logic;
                carry_out : out std_logic;

                result : out std_logic_vector(1 downto 0)
        );

end adder_2bits;


architecture behavioral of adder_2bits is

        -- Imports the full adder from the specification in the full_adder entity
        component full_adder

                port (
                        A : in std_logic;
                        B : in std_logic;

                        carry_in  : in std_logic;
                        carry_out : out std_logic;

                        result : out std_logic
                );

        end component;


        -- Declares the output and carry variables for the 4 full adders
        signal out_full_adder_1 : std_logic;
        signal out_full_adder_2 : std_logic;

        signal carry_full_adder_1 : std_logic;
        signal carry_full_adder_2 : std_logic;

begin

        -- Instantiates a full adder with all the default values
        instance_full_adder_1 : full_adder
```

```
                port map (
                        A => A(0),
                        B => B(0),

                        carry_in  => carry_in,
                        carry_out => carry_full_adder_1,

                        result => out_full_adder_1
                );


        -- Instantiates three more full adders with the previous carries as inputs
        -- and with incremental bit placements for the A and B inputs
        instance_full_adder_2 : full_adder

                port map (
                        A => A(1),
                        B => B(1),

                        carry_in  => carry_full_adder_1,
                        carry_out => carry_full_adder_2,

                        result => out_full_adder_2
                );


        -- Defines the adder/subtractor's carry out as the last full adder's carry
        carry_out <= carry_full_adder_2;

        -- Sets the result bits in order
        result(0) <= out_full_adder_1;
        result(1) <= out_full_adder_2;

end behavioral;
```

A.8 KEY_DECODE/KEY_SCANNER/COUNTER/FFD.vhd

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

-- Entity declaration for D Flip-Flop with enable, reset, and set functionalities
ENTITY FFD IS
    PORT (
        CLK   : IN  std_logic;   -- Clock input
        RESET : IN  std_logic;   -- Asynchronous reset input (active high)
        SET   : IN  std_logic;   -- Asynchronous set input (active high)
        D     : IN  std_logic;   -- Data input
        EN    : IN  std_logic;   -- Enable input (active high)
        Q     : OUT std_logic    -- Data output
    );
END FFD;

-- Architecture describing the behavior of the Flip-Flop
ARCHITECTURE logicFunction OF FFD IS
BEGIN

    -- Conditional signal assignment with priority: RESET > SET > ENABLE
    Q <= '0' WHEN RESET = '1' ELSE        -- Priority 1: Reset is active, set Q to '0'
         '1' WHEN SET = '1' ELSE          -- Priority 2: Set is active, set Q to '1'
          D  WHEN rising_edge(CLK) AND EN = '1';  -- Priority 3: Enable is active, updat

END logicFunction;
```

A.9 KEY_DECODE/KEY_SCANNER/COUNTER/full_adder.vhd

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

-- This entity is responsible for acting as a full adder, using two half-adders
-- in succession to account for first bit carries
entity full_adder is

        port (
                A : in std_logic;
                B : in std_logic;

                carry_in  : in std_logic;
                carry_out : out std_logic;

                result : out std_logic
        );

end full_adder;


architecture behavioral of full_adder is


        -- Imports the half adder from the specification in the half_adder entity
        component half_adder

                port (
                        A : in std_logic;
                        B : in std_logic;

                        result    : out std_logic;
                        carry_out  : out std_logic
                );

        end component;

        -- Initialises the result carriers for the half-adders
        signal out_half_adder_1 : std_logic;
        signal out_half_adder_2 : std_logic;

        signal carry_half_adder_1 : std_logic;
        signal carry_half_adder_2 : std_logic;


begin

        -- Instantiate the first half adder with the full adder's A and B
        instance_half_adder_1 : half_adder
```

```
            port map (
                    A => A,
                    B => B,

                    result => out_half_adder_1,
                    carry_out => carry_half_adder_1
            );


        -- Instantiate the second half adder with the first half adder's result and
        -- carry, to account for the bit carry
        instance_half_adder_2: half_adder

            port map (
                    A => out_half_adder_1,
                    B => carry_in,

                    result => out_half_adder_2,
                    carry_out => carry_half_adder_2
            );


        result <= out_half_adder_2;
        carry_out <= (carry_half_adder_1 or carry_half_adder_2);


end behavioral;
```

A.10 KEY_DECODE/KEY_SCANNER/COUNTER/half_adder.vhd

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

-- This entity is responsible for acting as a half adder, simply XOR'ing two
-- to sum them returning their value and carry.
entity half_adder is

        port (
                A : in std_logic;
                B : in std_logic;

                result      : out std_logic;
                carry_out   : out std_logic
        );

end half_adder;


architecture structural of half_adder is
begin

        result    <= A xor B;
        carry_out  <= A and B;

end structural;
```

A.11 KEY_DECODE/KEY_SCANNER/COUNTER/MUX2x1.vhd

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

-- This entity is responsible for acting like a MUX, and giving the output based
-- on the selected inputs
entity mux_2x1 is

        port (
                A : in std_logic_vector(1 downto 0);
                B : in std_logic_vector(1 downto 0);

                selector : in std_logic;

                result : out std_logic_vector(1 downto 0)
        );

end mux_2x1;


-- Implements the logic of a MUX with two inputs and one selector
architecture structural of mux_2x1 is
begin

        result(1) <= (A(1) and not selector) or (B(1) and selector);
        result(0) <= (A(0) and not selector) or (B(1) and selector);

end structural;
```

A.12 KEY_DECODE/KEY_SCANNER/COUNTER/registry_2bits.vhd

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

entity register_2bits is

    port (
        CLK   : in std_logic;
        RESET : in std_logic;
        SET   : in std_logic;
        D     : in std_logic_vector(1 downto 0);
        EN    : in std_logic;
        Q     : out std_logic_vector(1 downto 0)
    );

end register_2bits;


architecture behavioral of register_2bits is

    -- Declare the component for the flip-flop (FFD)
    component FFD is

        port (
            CLK    : in std_logic;
            RESET  : in std_logic;
            SET    : in std_logic;
            D      : in std_logic;
            EN     : in std_logic;
            Q      : out std_logic
        );

    end component;

begin

    -- Instantiate the flip-flops (FFD) for each bit of the data input
    FFD1: FFD
        port map (
            CLK   => CLK,
            RESET => RESET,
            SET   => '0',
            EN    => EN,
            D     => D(0),
            Q     => Q(0)
        );

    FFD2: FFD
        port map (
```

```
            CLK     => CLK,
            RESET   => RESET,
            SET     => '0',
            EN      => EN,
            D       => D(1),
            Q       => Q(1)
        );

end behavioral;
```

A.13 KEY_DECODE/KEY_SCANNER/COUNTER/registry_2bits_counter.vhd

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

entity register_2bits_counter is

    port (
        CLK   : in std_logic;
        RESET : in std_logic;
        SET   : in std_logic;
        D     : in std_logic_vector(1 downto 0);
        EN    : in std_logic;
        Q     : out std_logic_vector(1 downto 0)
    );

end register_2bits_counter;


architecture behavioral of register_2bits_counter is

    -- Declare the component for the flip-flop (FFD)
    component FFD is

        port (
            CLK    : in std_logic;
            RESET  : in std_logic;
            SET    : in std_logic;
            D      : in std_logic;
            EN     : in std_logic;
            Q      : out std_logic
        );

    end component;

begin

    -- Instantiate the flip-flops (FFD) for each bit of the data input
    FFD1: FFD
        port map (
            CLK   => CLK,
            RESET => RESET,
            SET   => '0',
            EN    => EN,
            D     => D(0),
            Q     => Q(0)
        );

    FFD2: FFD
        port map (
```

```
            CLK     => CLK,
            RESET   => RESET,
            SET     => '0',
            EN      => EN,
            D       => D(1),
            Q       => Q(1)
        );

end behavioral;
```

A.14 KEY_DECODE/KEY_SCANNER/DEC/decoder_2x4.vhd

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

-- This entity is responsible for acting as a 2x4 decoder
entity decoder_2x4 is

        port(
                S : in std_logic_vector (1 downto 0);
                Y : out std_logic_vector (3 downto 0)
        );

end decoder_2x4;


architecture structural of decoder_2x4 is
begin

        Y(0) <= (not S(1) and not S(0));
        Y(1) <= (not S(1) and S(0));
        Y(2) <= (S(1) and not S(0));
        Y(3) <= (S(1) and S(0));

end structural;
```

A.15 KEY_DECODE/KEY_SCANNER/PENC/mux_2x1_penc.vhd

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

-- This entity is responsible for acting like a MUX, and giving the output based
-- on the selected inputs
entity mux_2x1_penc is

        port (
                A : in std_logic;
                B : in std_logic;

                selector : in std_logic;

                result : out std_logic
        );

end mux_2x1_penc;


-- Implements the logic of a MUX with two inputs and one selector
architecture structural of mux_2x1_penc is
begin

        result <= (A and not selector) or (B and selector);

end structural;
```

A.16 KEY_DECODE/KEY_SCANNER/PENC/partial_priority_encoder.vhd

```
LIBRARY ieee;
Use ieee.std_logic_1164.all;

-- This entity is responsable for corresponding the 4 bits of the inputs to 2 bits
-- with Y(0) being A(0) and A(1) and Y(1) being the A(2) and A(3)
-- Gs is a signal that will be on when one of the inputs is active
entity partial_priority_encoder is
        port(
                A       : in std_logic_vector(1 downto 0);

                Y       : out std_logic;
                GS      : out std_logic
        );
end partial_priority_encoder;

architecture structural of partial_priority_encoder is

begin

        Y  <= not A(0);
        GS <= A(0) or A(1);

end structural;
```

A.17 KEY_DECODE/KEY_SCANNER/PENC/priority_encoder.vhd

```
LIBRARY ieee;
Use ieee.std_logic_1164.all;

-- This entity is responsable for corresponding the 4 bits of the inputs to 2 bits
-- with Y(0) being A(0) and A(1) and Y(1) being the A(2) and A(3)
-- Gs is a signal that will be on when one of the inputs is active
entity priority_encoder is
        port(
                A       : in std_logic_vector(3 downto 0);

                Y       : out std_logic_vector(1 downto 0);
                GS      : out std_logic
        );
end priority_encoder;

architecture structural of priority_encoder is

        component partial_priority_encoder is
        port(
                A       : in std_logic_vector(1 downto 0);

                Y       : out std_logic;
                GS      : out std_logic
        );
        end component;

        component mux_2x1_penc is

        port (
                A : in std_logic;
                B : in std_logic;

                selector : in std_logic;

                result : out std_logic
        );

        end component;

        signal Y_PPenc1 :  std_logic;
        signal GS_PPenc1        :  std_logic;

        signal Y_PPenc2 :  std_logic;
        signal GS_PPenc2        :  std_logic;

        signal Y_PPenc3 :  std_logic;
        signal GS_PPenc3        :  std_logic;
```

```
        signal out_mux2x1 :  std_logic;

begin


        PencPartial1: partial_priority_encoder
        port map(
                A(0) => A(0) ,
                A(1) => A(1),

                Y       => Y_PPenc1,
                GS      => GS_PPenc1
        );

        PencPartial2: partial_priority_encoder
        port map(
                A(0) => A(2) ,
                A(1) => A(3),

                Y       => Y_PPenc2,
                GS      => GS_PPenc2
        );

        PencPartial3: partial_priority_encoder
        port map(
                A(0) => GS_PPenc1,
                A(1) => GS_PPenc2,

                Y       => Y_PPenc3,
                GS      => GS_PPenc3
        );

        Mux2x1: mux_2x1_penc
        port map(
               A                => Y_PPenc1,
               B                =>      Y_PPenc2,

                selector        => Y_PPenc3,
                result  => out_mux2x1
        );

-- Sa das do Penc

        Y(0)    <= out_mux2x1;
        Y(1)    <= Y_PPenc3;

        GS      <= GS_PPenc3;

end structural;
```

A.18 UsbPort.vhd

```
-- 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;
```

B TestBench

B.1 testbench/Key_reader_tb.vhd

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

entity Key_reader_tb is
end Key_reader_tb;

architecture behavioral of Key_reader_tb is

    component Key_reader is
        port (
            Kack   : in std_logic;
            Liness : in std_logic_vector(3 downto 0);
            CLK    : in std_logic;
            Reset  : in std_logic;
            Colss  : out std_logic_vector(3 downto 0);
            Kout   : out std_logic_vector(3 downto 0);
            Kval   : out std_logic
        );
    end component;

    constant clk_period       : TIME := 10 ns;
    constant half_clk_period  : TIME := clk_period / 2;

    signal Kack_tb              : std_logic;
    signal lines_tb             : std_logic_vector(3 downto 0);
    signal clk_tb           : std_logic;
    signal reset_tb             : std_logic;
    signal columns_tb       : std_logic_vector(3 downto 0);
    signal K_tb                     : std_logic_vector(3 downto 0);
    signal Kval_tb          : std_logic;

begin

    UUT : Key_reader
        port map (
            Kack   => Kack_tb,
            Liness => lines_tb,
            CLK    => clk_tb,
            Reset  => reset_tb,
            Colss  => columns_tb,
            Kout   => K_tb,
            Kval   => Kval_tb
        );

    clk_gen: process
    begin
        reset_tb <= '1';
        lines_tb <= "1111";
```

```
Kack_tb  <= '0';
wait for clk_period;

lines_tb <= "1111";  -- not Kpress
reset_tb <= '0';
  Kack_tb  <= '1';
wait for clk_period * 2;

-- Test of first line
lines_tb <= "0111";
wait for clk_period * 2;

Kack_tb  <= '1';
wait for clk_period * 2;

lines_tb <= "1111";  -- not Kpress
wait for clk_period * 2;

Kack_tb  <= '0';
wait for clk_period * 2;

-- Test of second line
lines_tb <= "1011";
wait for clk_period * 2;

Kack_tb  <= '1';
wait for clk_period * 2;

lines_tb <= "1111";  -- not Kpress
wait for clk_period * 2;

Kack_tb  <= '0';
wait for clk_period * 2;

-- Test of third line
lines_tb <= "1101";
wait for clk_period * 2;

Kack_tb  <= '1';
wait for clk_period * 2;

lines_tb <= "1111";  -- not Kpress
wait for clk_period * 2;

Kack_tb  <= '0';
wait for clk_period * 2;

-- Test of fourth line
lines_tb <= "1110";
wait for clk_period * 2;
```

```
        Kack_tb <= '1';
        wait for clk_period * 2;

        lines_tb <= "1111"; -- not Kpress
        wait for clk_period * 2;

        Kack_tb <= '0';
        wait for clk_period * 2;

        wait;
    end process;

end behavioral;
```

B.2 testbench/Key_decode_tb.vhd

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

entity Key_decode_tb is
end Key_decode_tb;

architecture behavioral of Key_decode_tb is

    component Key_decode is
        port (
            Kack   : in std_logic;
            Liness : in std_logic_vector(3 downto 0);
            CLK    : in std_logic;
            Reset  : in std_logic;

            Colss  : out std_logic_vector(3 downto 0);
            Kout   : out std_logic_vector(3 downto 0);
            Kval   : out std_logic
        );
    end component;

    constant clk_period       : TIME := 20 ns;
    constant half_clk_period  : TIME := clk_period / 2;

    signal Kack_tb      : std_logic;
    signal lines_tb     : std_logic_vector(3 downto 0);
    signal clk_tb       : std_logic;
    signal reset_tb     : std_logic;
    signal columns_tb   : std_logic_vector(3 downto 0);
    signal K_tb         : std_logic_vector(3 downto 0);
    signal Kval_tb      : std_logic;

begin

    UUT : Key_decode
        port map (
            Kack   => Kack_tb,
            Liness => lines_tb,
            CLK    => clk_tb,
            Reset  => reset_tb,
            Colss  => columns_tb,
            Kval   => Kval_tb,
            Kout   => K_tb
        );

    clk_gen: process
    begin
        clk_tb <= '0';
```

```
        wait for half_clk_period;
        clk_tb <= '1';
        wait for half_clk_period;
    end process;

    stimulus: process
    begin
        reset_tb <= '1';
        lines_tb <= "1111";
        Kack_tb  <= '0';
        wait for clk_period;

        lines_tb <= "1111";  -- not Kpress
        reset_tb <= '0';
        Kack_tb  <= '1';
        wait for clk_period * 2;

        -- Test of first line
        lines_tb <= "0111";
        wait for clk_period * 2;

        Kack_tb  <= '1';
        wait for clk_period * 2;

        lines_tb <= "1111";  -- not Kpress
        wait for clk_period * 2;

        Kack_tb  <= '0';
        wait for clk_period * 2;

        -- Test of second line
        lines_tb <= "1011";
        wait for clk_period * 2;

        Kack_tb  <= '1';
        wait for clk_period * 2;

        lines_tb <= "1111";  -- not Kpress
        wait for clk_period * 2;

        Kack_tb  <= '0';
        wait for clk_period * 2;

        -- Test of third line
        lines_tb <= "1101";
        wait for clk_period * 2;

        Kack_tb  <= '1';
        wait for clk_period * 2;
```

```
        lines_tb <= "1111";  -- not Kpress
        wait for clk_period * 2;

        Kack_tb  <= '0';
        wait for clk_period * 2;

        -- Test of fourth line
        lines_tb <= "1110";
        wait for clk_period * 2;

        Kack_tb  <= '1';
        wait for clk_period * 2;

        lines_tb <= "1111";  -- not Kpress
        wait for clk_period * 2;

        Kack_tb  <= '0';
        wait for clk_period * 2;

        wait;
    end process;

end behavioral;
```

B.3 testbench/KeyControl_tb.vhd

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

entity KeyControl_tb is
end KeyControl_tb;

architecture behavioral of KeyControl_tb is

    component KeyControl is
        port (
            reset   : in std_logic;
            clk     : in std_logic;
            Kpress  : in std_logic;
            Kack    : in std_logic;

            KScan   : out std_logic_vector(1 downto 0);
            Kval    : out std_logic
        );
    end component;

    constant clk_period       : TIME := 10 ns;
    constant half_clk_period  : TIME := clk_period / 2;

    signal reset_tb   : std_logic;
    signal clk_tb     : std_logic;
    signal Kpress_tb  : std_logic;
    signal Kack_tb    : std_logic;
    signal KScan_tb   : std_logic_vector(1 downto 0);
    signal Kval_tb    : std_logic;

begin

    UUT : KeyControl
        port map (
            reset   => reset_tb,
            clk     => clk_tb,
            Kpress  => Kpress_tb,
            Kack    => Kack_tb,
            KScan   => KScan_tb,
            Kval    => Kval_tb
        );

    clk_gen: process
    begin
        clk_tb <= '0';
        wait for half_clk_period;
        clk_tb <= '1';
        wait for half_clk_period;
```

```
    end process;

    stimulus: process
    begin
        reset_tb    <= '1';
        wait for clk_period;

        Kpress_tb   <= '0';
        Kack_tb     <= '0';
        reset_tb    <= '0';
        wait for clk_period;

        Kpress_tb   <= '1';
        Kack_tb     <= '0';
        wait for clk_period * 2;

        Kpress_tb   <= '0';
        Kack_tb     <= '0';
        wait for clk_period;

        Kpress_tb   <= '1';
        Kack_tb     <= '1';
        wait for clk_period;

        Kpress_tb   <= '1';
        Kack_tb     <= '0';
        wait for clk_period;

        Kpress_tb   <= '0';
        Kack_tb     <= '1';
        wait for clk_period;

        Kpress_tb   <= '1';
        Kack_tb     <= '1';
        wait for clk_period;

        Kpress_tb   <= '0';
        Kack_tb     <= '0';
        wait for clk_period * 2;

        wait;
    end process;

end behavioral;
```

B.4 testbench/KeyScannerTb.vhd

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

entity Key_Scanner_tb is
end Key_Scanner_tb;

architecture behavioral of Key_Scanner_tb is

    component key_scanner is
        port(
            KScan   : in std_logic_vector(1 downto 0);
            lines   : in std_logic_vector(3 downto 0);
            CLK     : in std_logic;
            reset   : in std_logic;

            columns : out std_logic_vector(3 downto 0);
            Kpress  : out std_logic;
            K       : out std_logic_vector(3 downto 0)
        );
    end component;

    constant clk_period       : TIME := 10 ns;
    constant half_clk_period  : TIME := clk_period / 2;

    signal Kscan_tb   : std_logic_vector(1 downto 0);
    signal lines_tb   : std_logic_vector(3 downto 0);
    signal clk_tb     : std_logic;
    signal reset_tb   : std_logic;
    signal columns_tb : std_logic_vector(3 downto 0);
    signal Kpress_tb  : std_logic;
    signal K_tb       : std_logic_vector(3 downto 0);

begin

    UUT : key_scanner
        port map (
            Kscan   => Kscan_tb,
            lines   => lines_tb,
            CLK     => clk_tb,
            reset   => reset_tb,
            columns => columns_tb,
            Kpress  => Kpress_tb,
            K       => K_tb
        );

    clk_gen: process
    begin
        clk_tb <= '0';
```

```
        wait for half_clk_period;
        clk_tb <= '1';
        wait for half_clk_period;
    end process;

    stimulus: process
    begin
        reset_tb <= '1';
        Kscan_tb <= "11";
        lines_tb <= "0000";
        wait for clk_period*2;

        reset_tb <= '0';
        Kscan_tb <= "11";
        lines_tb <= "0001";
        wait for clk_period*2;

        Kscan_tb <= "10";
        lines_tb <= "0001";
        wait for clk_period*2;

        Kscan_tb <= "00";
        lines_tb <= "0001";
        wait for clk_period*2;


        Kscan_tb <= "11";
        lines_tb <= "0010";
        wait for clk_period*2;

        Kscan_tb <= "10";
        lines_tb <= "0010";
        wait for clk_period*2;

        Kscan_tb <= "00";
        lines_tb <= "0010";
        wait for clk_period*2;


        Kscan_tb <= "11";
        lines_tb <= "0100";
        wait for clk_period*2;

        Kscan_tb <= "10";
        lines_tb <= "0100";
        wait for clk_period*2;

        Kscan_tb <= "00";
        lines_tb <= "0100";
        wait for clk_period*2;
```

```
            Kscan_tb <= "11";
        lines_tb <= "1000";
        wait for clk_period*2;

        Kscan_tb <= "10";
        lines_tb <= "1000";
        wait for clk_period*2;

        Kscan_tb <= "00";
        lines_tb <= "1000";
        wait for clk_period*2;

            Kscan_tb <= "11";
        lines_tb <= "0000";
        wait for clk_period*2;

        Kscan_tb <= "10";
        lines_tb <= "0000";
        wait for clk_period*2;

        Kscan_tb <= "00";
        lines_tb <= "0000";
        wait for clk_period*2;

        wait;
    end process;

end behavioral;
```

B.5 testbench/CounterTb.vhd

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

entity counter_tb is
end counter_tb;

architecture behavioral of counter_tb is

    component counter is
        port (
            CE                  : in std_logic;
            CLK                 : in std_logic;
            reset               : in std_logic;
            parallel_load_flag  : in std_logic;
            parallel_load_value : in std_logic_vector(1 downto 0);
            count               : out std_logic_vector(1 downto 0)
        );
    end component;

    constant clk_period       : TIME := 10 ns;
    constant half_clk_period  : TIME := clk_period / 2;

    signal CE_tb                  : std_logic;
    signal CLK_tb                 : std_logic;
    signal reset_tb               : std_logic;
    signal parallel_load_flag_tb  : std_logic;
    signal parallel_load_value_tb : std_logic_vector(1 downto 0);
    signal count_tb               : std_logic_vector(1 downto 0);

begin

    UUT: counter
        port map (
            CE                  => CE_tb,
            CLK                 => CLK_tb,
            reset               => reset_tb,
            parallel_load_flag  => parallel_load_flag_tb,
            parallel_load_value => parallel_load_value_tb,
            count               => count_tb
        );

    -- Clock Generation
    clk_gen: process
    begin
        while true loop
            CLK_tb <= '0';
            wait for half_clk_period;
```

```
            CLK_tb <= '1';
            wait for half_clk_period;
        end loop;
    end process;

    -- Stimulus Process
    stimulus: process
    begin
        -- Reset the counter and enable counting
            reset_tb  <= '1';
                  CE_tb         <= '1';
                  -- Signal to use on mux that selects the parallel load
                  parallel_load_flag_tb <= '0';
                  -- Value that will be input in the register if the selector is 1
            parallel_load_value_tb <= "01";
            wait for clk_period;

            reset_tb <= '0';
                  wait for clk_period*8;


            parallel_load_flag_tb <= '1';
            wait for clk_period*2;

                  -- Stop the increment of the counter by changing the mux signal
            parallel_load_flag_tb <= '1';
                  parallel_load_value_tb <= "10";

            wait for clk_period;
            wait for clk_period;

                  -- Change it to the default form of the counter, increment +1
                  parallel_load_flag_tb <= '0';
                  parallel_load_value_tb <= "01";

                  wait for clk_period*8;

           -- Disable counting
            CE_tb <= '0';
            wait for clk_period;

            wait;
        end process;

end behavioral;
```

B.6 testbench/decoder_tb.vhd

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

entity decoder_tb is
end decoder_tb;

architecture behavioral of decoder_tb is

    component decoder_2x4 is
        port (
            S : in std_logic_vector (1 downto 0);
                                Y : out std_logic_vector (3 downto 0)
        );
    end component;

    signal s_tb           : std_logic_vector(1 downto 0);
    signal y_tb     : std_logic_vector(3 downto 0);

begin

    UUT : decoder_2x4
        port map (
            S   => s_tb,
            Y   => y_tb
        );

    stimulus: process
    begin
        s_tb    <= "00";
        wait for 20 ns;

        s_tb    <= "01";
          wait for 20 ns;

        s_tb    <= "10";
        wait for 20 ns;

          s_tb    <= "11";
        wait for 20 ns;


        wait;
    end process;

end behavioral;
```

B.7 testbench/priority_encoder_tb.vhd

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

entity priority_encoder_tb is
end priority_encoder_tb;

architecture behavioral of priority_encoder_tb is

    component priority_encoder is
        port (
            A   : in std_logic_vector(3 downto 0);

                Y       : out std_logic_vector(1 downto 0);
            GS  : out std_logic
        );
    end component;

    signal a_tb      : std_logic_vector(3 downto 0);
    signal y_tb      : std_logic_vector(1 downto 0);
    signal gs_tb     : std_logic;

begin

    UUT : priority_encoder
        port map (
                                        A       => a_tb,

                                        Y  => y_tb,
                                        GS      => gs_tb
        );

    stimulus: process
    begin

        a_tb    <= "0000";
        wait for 20 ns;

        a_tb    <= "0001";
        wait for 20 ns;

        a_tb    <= "0010";
        wait for 20 ns;

        a_tb    <= "0100";
          wait for 20 ns;

        a_tb    <= "1000";
        wait for 20 ns;
```

```
        wait;
    end process;

end behavioral;
```

C Kotlin

C.1 roulette_software/src/main/kotlin/com/github/iselgt/roulette/UsbPort.kt

```
package com.github.iselgt.roulette
import isel.leic.UsbPort

fun main() {

    HAL.init()
    var counter = 0

    while (true) {

        val value = UsbPort.read()
        println("${counter++}: ${Integer.toBinaryString(value)}")

        Thread.sleep(500)
    }
}
```

C.2 roulette_software/src/main/kotlin/com/github/iselgt/roulette/HAL.kt

```
package com.github.iselgt.roulette

import isel.leic.UsbPort

object HAL {
    private var resetOutput = 0

    fun init(){
        resetOutput = 0
        UsbPort.write(resetOutput)
    }

    //Checks if the bit chosen by the mask is on
    fun isBit(mask: Int):Boolean {
        require(mask.countOneBits()==1){ "mask must be one bit only" }
        return UsbPort.read().and(mask) != 0
    }

    //Read the bits chosen by the mask and puts 0 on the bits that you aren't checking
    fun readBits(mask:Int):Int{
        return UsbPort.read().and(mask)
    }

    //Force 0 on the bits chosen by the mask
    fun clrBits(mask:Int) {
        resetOutput = resetOutput.and(mask.inv())
        UsbPort.write(resetOutput)
    }

    //Force 1 on the bits chosen by the mask
    fun setBits(mask:Int){
        resetOutput = mask.or(resetOutput)
        UsbPort.write(resetOutput)
    }

    //Chose the bits that u want to rewrite and force them with the input value
    fun writeBits(mask:Int, value:Int){
        clrBits(mask)
        setBits(value.and(mask))
    }
}
```

C.3 roulette_software/src/main/kotlin/com/github/iselgt/roulette/KBD.kt

```
package com.github.iselgt.roulette
import isel.leic.utils.Time


object KBD {

    const val EMPTY_CHAR = 0x00.toChar()      // and empty char
    private const val KVAL = 0x01             // Mask for Kval
    private const val K = 0x1E                // Mask for K
    private const val KACK = 0x01             // Mask for Kack

    private val keys =
    charArrayOf('1', '4', '7', '*', '2', '5', '8', '0',
                '3', '6', '9', '#', 'A', 'B', 'C', 'D')

    fun init () {
        HAL.init()
    }

    private fun getKey(): Char {
        if (HAL.isBit(KVAL)) {
            val key = HAL.readBits(K)
            HAL.setBits(KACK)

            while(HAL.isBit(KVAL)) {}
                HAL.clrBits(KACK)
                return keys[key.shr(1)]
                // Need to use shr because of the K being in InputUsbPort(4-1)
            }
        return EMPTY_CHAR
    }

    fun waitKey(timeout : Long): Char {
        val endTime = timeout + Time.getTimeInMillis()
        var key: Char
        do {
            key = getKey()
        } while (Time.getTimeInMillis() < endTime && key == EMPTY_CHAR)
        return key
    }
}
```

C.4 roulette_software/src/main/kotlin/com/github/iselgt/roulette/LCD.kt

```
package com.github.iselgt.roulette

import isel.leic.utils.Time

object LCD {
    private const val LINES = 2
    private const val COLS = 16

    private const val SERIAL_INTERFACE = false

    // Useful Constants to use with LCD
    private const val NONE_VALUE = 0x00             // Null-Terminator value for when no k
    private const val DATA_MASK = 0x1E              // A useful mask that correspond to th
    private const val ENABLE_MASK = 0x80            // A useful mask that correspond to en
    private const val REGISTER_SELECTOR_MASK= 0x40// A useful mask that correspond to th

    private const val WAIT_FIRST_TIME = 15L
    private const val WAIT_TIME = 5L

    // LCD Command Constants
    private const val LCD_CLEAR = 0x01              // Clear display & reset cursor to (0,
    const val LCD_HOME = 0x02                       // Return cursor to (0,0) without clea
    private const val LCD_ENTRY_MODE = 0x06         // Cursor moves right, no display shif
    private const val LCD_DISPLAY_OFF = 0x08        // Turn off display
    const val LCD_DISPLAY_ON = 0x0C                 // Display ON, Cursor OFF, Blink OFF
    const val LCD_DISPLAY_ON_CURSOR = 0x0E          // Display ON, Cursor ON, No Blink
    private const val LCD_DISPLAY_ON_BLINK = 0x0F // Display ON, Cursor ON, Blink ON
    private const val LCD_FUNCTION_SET = 0x28       // 4-bit mode, 2 lines, 5x8 font

    // Cursor Positioning Constants
    private const val LCD_LINE_1 = 0x80             // First line
    private const val LCD_LINE_2 = 0xC0             // Second line

    // Mode Sets Bits Constants
    private const val SET4BITS = 0x2                // Entry Mode Set 4 bits long
    private const val SET8BITS = 0x3                // Entry Mode Set 8 bits long


    private fun writeDATA(data: Int) {
        writeByte(true, data)
    }

    private fun writeCMD(data: Int) {
        writeByte(false, data)
    }

    // Writes a Byte command/data on LCD
    private fun writeByte(rs: Boolean, data: Int) {
```

```
        // The 4 most significant bits
        val topdata = data shr 4
        // The 4 less significant bits
        val botdata = data and 0x0F

        writeNibble(rs, topdata)  // High
        writeNibble(rs, botdata)  // Low

    }

    // Writes a nibble command/data on LCD
    private fun writeNibble(rs: Boolean, data: Int) {
        if (SERIAL_INTERFACE) writeNibbleSerial(rs, data.shl(1))
        else writeNibbleParallel(rs, data.shl(1))
    }


    // Writes a nibble (4 bits) of command/data to the LCD in Parallel Mode
    private fun writeNibbleParallel(rs: Boolean, data: Int) {
        // Set or clear the Register Select (RS) pin depending on whether sending data c
        if (rs) {
            HAL.setBits(REGISTER_SELECTOR_MASK)      // RS = 1 for data
        }
        else {
            HAL.clrBits(REGISTER_SELECTOR_MASK)      // RS = 0 for command
        }
        // Set the Enable (E) pin to high to latch the data
        HAL.setBits(ENABLE_MASK)

        // Send the 4-bit data by writing it to the data lines
        HAL.writeBits(DATA_MASK, data)

        // Hold Enable high for a short period to ensure the data is latched
        Time.sleep(WAIT_TIME)

        // Set the Enable (E) pin to low to complete the data transfer
        HAL.clrBits(ENABLE_MASK)

         // Wait again before the next operation
        Time.sleep(WAIT_TIME)
    }


        // Writes a nibble (4 bits) of command/data to the LCD in Serial Mode
    private fun writeNibbleSerial(rs: Boolean, data: Int) {
        TODO()
    }

    fun init() {
```

```
        // Initiate LCD with 8-bit mode before switching to 4-bit mode
        Time.sleep(WAIT_FIRST_TIME)                 // Longer wait time for power-on
        writeNibble(false, SET8BITS)
        Time.sleep(WAIT_TIME)
        writeNibble(false, SET8BITS)
        Time.sleep(WAIT_TIME)
        writeNibble(false, SET8BITS)

        // Now switch to 4-bit mode
        writeNibble(false, SET4BITS)

        // Configure LCD settings using named constants
        writeCMD(LCD_FUNCTION_SET)                  // 4-bit mode, 2-line, 5x8 font
        writeCMD(LCD_DISPLAY_OFF)                   // Display OFF
        writeCMD(LCD_CLEAR)                         // Clear display
        Time.sleep(WAIT_TIME)                       // Extra delay needed for clearing
        writeCMD(LCD_ENTRY_MODE)                    // Cursor moves right
        writeCMD(LCD_DISPLAY_ON_BLINK)              // Display ON, Cursor ON, Blink ON
        Time.sleep(WAIT_TIME)                       // Short delay for stability
    }


    fun write(c: Char) {
        if (c != NONE_VALUE.toChar()) writeDATA(c.code)
//.code => .toInt()
    }

    fun write(text: String) {
        for (c in text) {
            write(c)
        }
    }

    fun cursor(line: Int, column: Int) {
        val address = when (line) {
            0 -> LCD_LINE_1 + column                // First line
            1 -> LCD_LINE_2 + column                // Second line
            else -> LCD_LINE_1                      // Default to first line if invalid
        }
        writeCMD(address)
    }

    fun clear() {
        writeCMD(LCD_CLEAR)
        Time.sleep(WAIT_TIME)
    }
}

fun main() {
    HAL.init()
```

```
    LCD.init()
    while (true) {
        LCD.write(KBD.waitKey(500))
    }
}
```

C.5 roulette_software/src/main/kotlin/com/github/iselgt/roulette/TUI.kt

```
package com.github.iselgt.roulette

import isel.leic.utils.Time
import java.util.concurrent.TimeUnit

const val WAIT_TIME = 2L
const val WAIT_KEY = 200L

object TUI {
    fun init() {
        KBD.init()
        LCD.init()
    }

    fun writeMessage(msg: String){
        LCD.clear()
        LCD.write(msg)
    }
}

fun main() {

    TUI.init()                                  // Initialize the TUI (Text User Interface) sy
    TUI.writeMessage("Hello World :)")
    Time.sleep(TimeUnit.SECONDS.toMillis(WAIT_TIME))// Pause the program for WAIT_TIME s
    LCD.clear()
    TUI.writeMessage("Grupo 11 Display")
    Time.sleep(TimeUnit.SECONDS.toMillis(WAIT_TIME))
    LCD.clear()
    TUI.writeMessage("Use '*' to Clear")

    var isCleared = false // Create a var that give us the information if the display wa

    while (true) {
        val key = KBD.waitKey(WAIT_KEY)
        if (key == '*') {     // Chose the '*' char to be a shortcut for clearing the dis
            LCD.clear()
            isCleared = true
        }
        else {       // Write the key after clearing and reset the flag
            if (isCleared) {
                LCD.write(key)
                isCleared = false     // Reset the flag after writing the first key
            }
            else {
                LCD.write(key)
            }
        }
```

```
    }
}
```

D Kotlin_Tests

D.1 roulette_software/src/test/kotlin/HALTest.kt

```
package com.github.iselgt.roulette

import isel.leic.UsbPort
import kotlin.test.Test

class HALTest {

    @Test
    fun halGlobalTest() {

        HAL.init()
        var counter = 0x00

        while (true) {

            val value = UsbPort.read()
            HAL.setBits(7)
            println("${counter++}: ${Integer.toBinaryString(value)} | ${Integer.toBinary
            Thread.sleep(500)
        }
    }
}
```

D.2 roulette_software/src/test/kotlin/KBDTest.kt

```
package com.github.iselgt.roulette

import com.github.iselgt.roulette.KBD.waitKey
import kotlin.test.Test

class KBDTest {

    @Test
    fun kbdTest() {
        KBD.init()

        while (true) {
            val keyPressed = waitKey(1000)
            if (keyPressed == KBD.EMPTY_CHAR) continue

            print(keyPressed)
        }
    }
}
```

D.3 roulette_software/src/test/kotlin/LCDTest.kt

```
package com.github.iselgt.roulette

import kotlin.test.Test

class LCDTest {

    @Test

    fun lcdTest(){
        LCD.init()
        while (true){
            val Key = KBD.waitKey(200)
            LCD.write(Key)
            if (Key =='*') {
                break
            }
        }
    }
}
```

E Pinos

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✔		Lin...[0]	Location	PIN_W5	Yes			
2	✔		Lin...[1]	Location	PIN_AA14	Yes			
3	✔		Lin...[2]	Location	PIN_W12	Yes			
4	✔		Lin...[3]	Location	PIN_AB12	Yes			
5	✔		Colss[0]	Location	PIN_AB11	Yes			
6	✔		Colss[1]	Location	PIN_AB10	Yes			
7	✔		Colss[2]	Location	PIN_AA9	Yes			
8	✔		Colss[3]	Location	PIN_AA8	Yes			
9	✔		Kout[0]	Location	PIN_A8	Yes			
10	✔		Kout[1]	Location	PIN_A9	Yes			
11	✔		Kout[2]	Location	PIN_A10	Yes			
12	✔		Kout[3]	Location	PIN_B10	Yes			
13	✔		Reset	Location	PIN_F15	Yes			
14	✔		CLK	Location	PIN_P11	Yes			
15	✔		LCD_RS	Location	PIN_W8	Yes			
16	✔		LCD_EN	Location	PIN_V5	Yes			
17	✔		LCD...[4]	Location	PIN_W11	Yes			
18	✔		LCD...[5]	Location	PIN_AA10	Yes			
19	✔		LCD...[6]	Location	PIN_Y8	Yes			
20	✔		LCD...[7]	Location	PIN_Y7	Yes			

Figura 6: Atribuição dos pinos do roulette