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*). Neste caso o módulo *Control*, implementado em *software*, é a entidade consumidora.

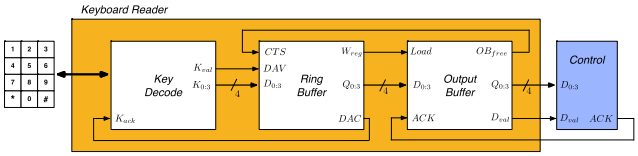
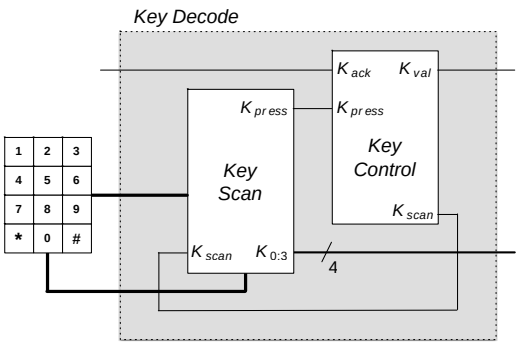


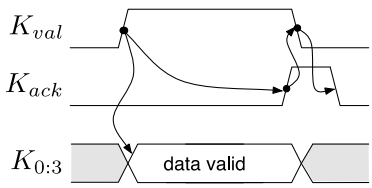
Figura 1 – Diagrama de blocos do módulo *Keyboard Reader*

1 Key Decode

O bloco *Key Decode* implementa um descodificador de um teclado matricial 4x3 por *hardware*, sendo constituído por três sub-blocos: *i)* um teclado matricial de 4x3; *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 2a. 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 2b.



a) Diagrama de blocos



b) Diagrama temporal

Figura 2 – Bloco *Key Decode*

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

A implementação 1 foi selecionada por ser a mais fácil de colocar em prática, já que tem menos componentes individuais em comparação com as outras opções.

O módulo Key Control foi projetado com base em uma máquina de estados, conforme ilustrado na máquina de estados apresentada na Figura 4. Esta máquina opera em três estados principais: Scanning, Pressing e Waiting. O sinal Kpress é ativado quando uma tecla é pressionada, o sinal Kack é '1' quando a tecla pressionada é reconhecida e armazenada. O sinal Kscan é '1' durante o processo de varredura do teclado e, o sinal Kpressa a '0'. O sinal Kval indica que há um conjunto de dados pronto para ser transmitido e, o seu sinal passa a '1' quando encontra-se no estado pressing. A descrição hardware do bloco *Key Decode* em VHDL encontra-se no Anexo A.

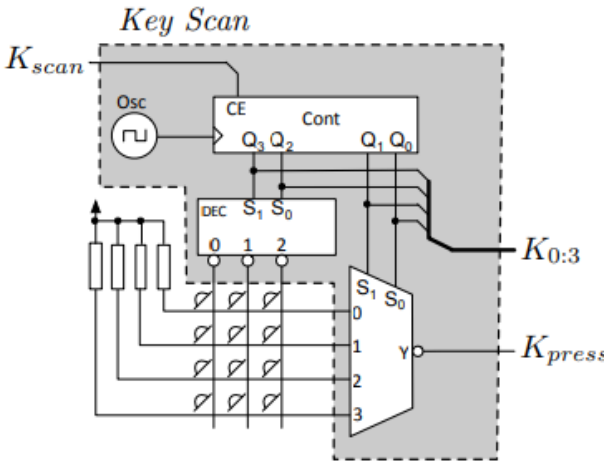


Figura 3 - Diagrama de blocos do bloco *Key Scan*

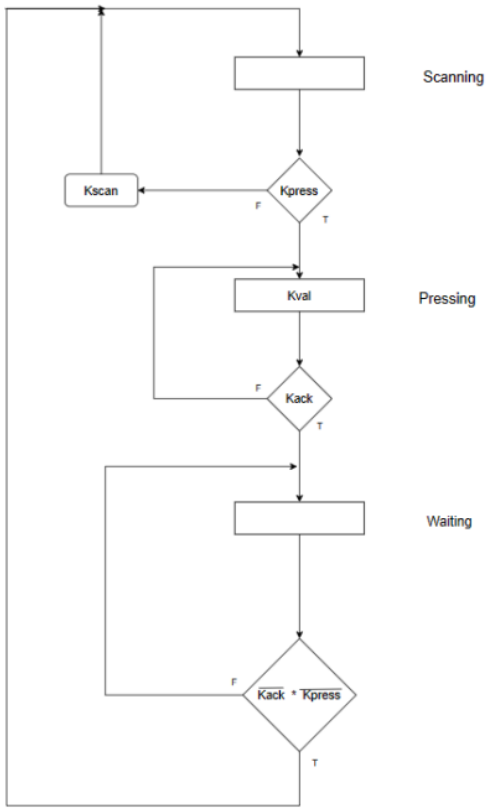
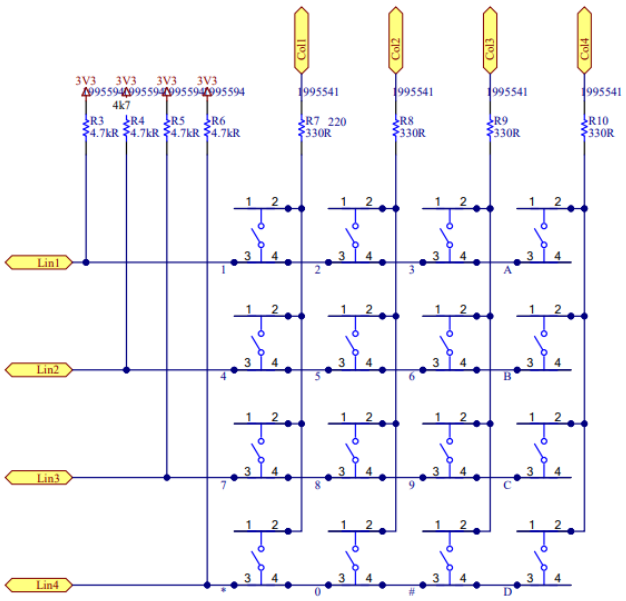


Figura 4 – Máquina de estados do bloco *Key Control*

Com base nas descrições do bloco *Key Decode* implementou-se parcialmente o módulo *Keyboard Reader* de acordo com o esquema elétrico representado no anexo abaixo.



2 Interface com o *Control*

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem Kotlin e seguindo a arquitetura lógica apresentada na figura 5.

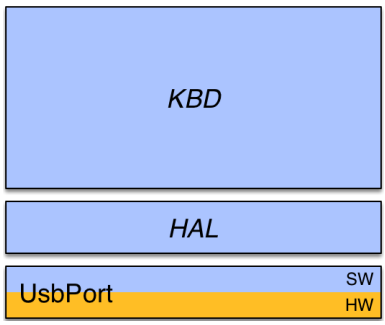


Figura 5 – Diagrama lógico do módulo *Control* de interface com o módulo *Keyboard Reader*

As classes *HAL* e *KBD* desenvolvidas são descritas nas secções 2.1. e 2.2, e o código fonte desenvolvido nos Anexos C e D, respetivamente.

2.1 Classe *HAL*

O HAL(Hardware Abstraction Layer) fornece uma interface simplificada para acessar o sistema UbsPort, permitindo que outros módulos de software interajam com o hardware de forma mais fácil e independentemente de detalhes específicos do dispositivo.

• init, que inicia a classe

• isBit, que recebe uma máscara como parâmetro do tipo inteiro e retorna true (boolean) se o bit analisado pela máscara tiver o valor '1'

• readBits, que recebe também uma máscara do tipo inteiro, que retorna os valores dos bits do UsbPort segundo a máscara

• writeBits, que recebe como parâmetros uma máscara e um valor, ambos inteiros, e que escreve no output os bits representados por mask e pelo valor

• setBits, que recebe uma máscara e coloca os bits representados pela mesma a '1' e escreve-os no output

• clrBits, que recebe também uma máscara e que faz o mesmo que a setBits, só que em vez de colocar os bits a '1', coloca '0'

2.2 Classe *KBD*

O KBD (Keyboard Reader) é encarregue de ler as teclas que são pressionadas e, em seguida, retornar o caractere correspondente à tecla pressionada. Os caracteres podem variar de '0' a '9','#','*' ou "NONE" se nenhuma tecla for pressionada.

• init, para iniciar a classe, que usa o HAL.init

• getKey, que retorna a tecla premida em formato char ou, caso nenhuma tecla esteja a ser premida, retorna NONE (valor 0, visto que NONE foi definido como um valor 0)

• waitKey, que recebe como parâmetro um timeout, do tipo Long, e retorna a tecla premida dentro do tempo do timeout

3 Ring Buffer

O bloco Ring Buffer é uma estrutura de dados para armazenamento de teclas com disciplina FIFO (First In First Out), com capacidade de armazenar até oito palavras de quatro bits.

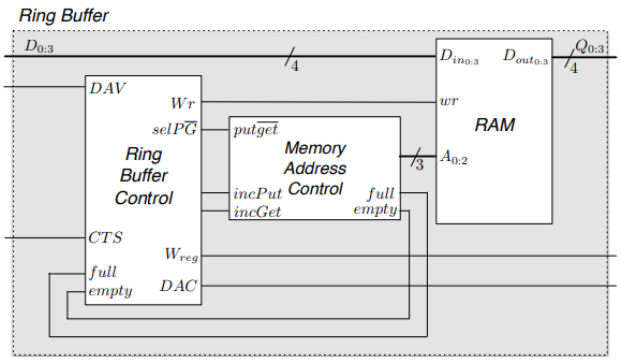


Figura 6 – Diagrama de blocos do bloco *Ring Buffer*

Contém três blocos: o Ring Buffer Control, o Memory Address Control (MAC) e a RAM:

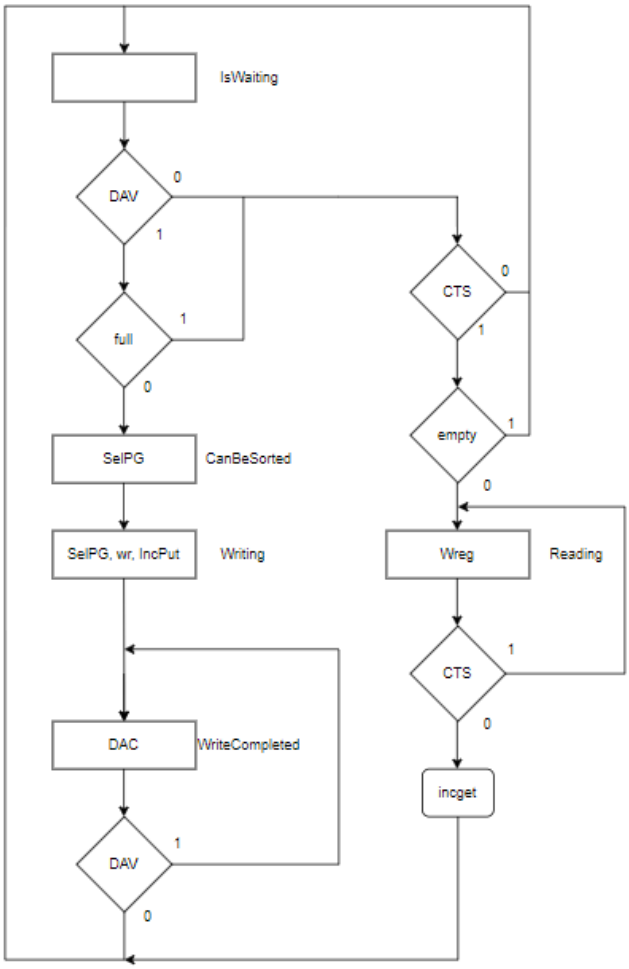


Figura 7 – Máquina de estados do bloco *Ring Buffer Control*

- O Ring Buffer Control é uma máquina de cinco estados, sendo quarto de escrita e um de leitura, (isWaiting,CanbeStored Writing, WriteCompleted e Reading) e está implementado de maneira a dar prioridade à escrita invés da leitura, o sinal que vai indicar se é para escrever é o DAV(Data Available) após verificar que o mesmo esta ativo, é verificado se o MAC está cheio (full), a máquina vai passar por três estados, no primeiro estado ativa-se o sinal selPG, no segundo o sinal selPG mantém-se ativo e os sinais wr, e incPut são ativos. No terceiro, ultimo estado da escrita, apenas o sinal DAC é ativo, os sinais ativos nos três estados anteriores, vão indicar que está a ocorrer a inserção de uma palavra na memória e

assim, a MAC vai fazer os seus ajustes no sinais full e empty, passando assim para o estado inicial isWaiting se o sinal DAV estiver inativo.
No estado IsWaiting, caso o DAV seja '0' ou o full seja '1', vai se verificar se o CTS(Clear to Send) está ativo e se o sinal empty está a '0', senão retornará para o próprio estado isWaiting, isto vai indicar que vamos fazer um "get" da RAM e vai ser lido uma palavra da memória sendo ativado o sinal Wreg e, se o sinal CTS já estiver inativo é ativado o sinal incGet, senão retornará ao próprio estado isWaiting.

- O MAC define o endereço de escrita/leitura, selecionado por $put\overline{get}$ e é composto por 1 mux 2x1 e 3 counters, sendo o primeiro responsável pela contagem do sinal incPut, o segundo responsável pela contagem do sinal incGet e por fim, o terceiro sendo responsável pela incrementação ou decrementação.. Suporta assim ações de incPut e incGet, gerando informação se a estrutura de dados está cheia (Full) ou se está vazia (Empty).

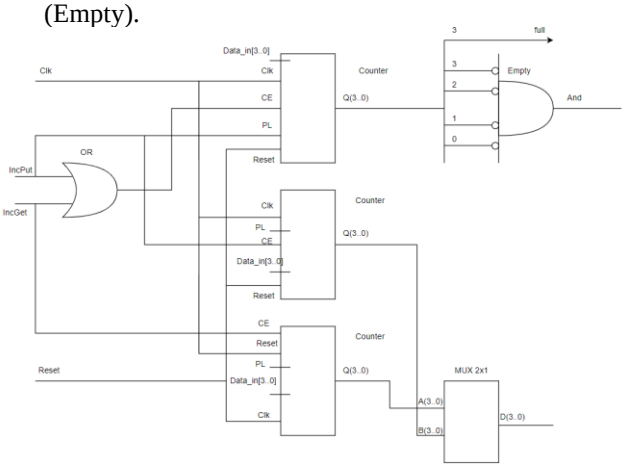


Figura 8- Diagrama de blocos do bloco MAC

- A RAM é uma memória com capacidade para oito palavras de quatro bits e tem como inputs um D*in* com quatro bits (valor que é escrito na memória), *wr* e um *A* com três bits que vai indicar qual o endereço da memória a ler e tem como outputs um D*out* com quatro bits.

4 Output Buffer

O bloco Output Buffer do Keyboard Reader é responsável pela interação com o sistema consumidor, neste caso o módulo Control, e indica se está disponível para armazenar dados através do sinal OB*free*. Assim, nesta situação o sistema produtor pode ativar o sinal *Load* para registar os dados

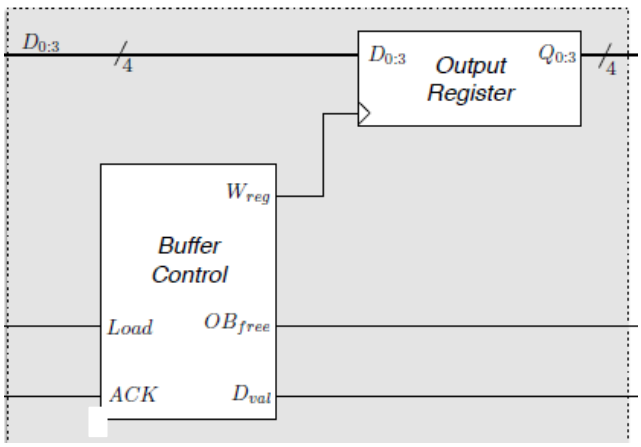


Figura 9- Diagrama de blocos do bloco Output Buffer

Contém dois blocos: o Buffer Control e o Output Register:

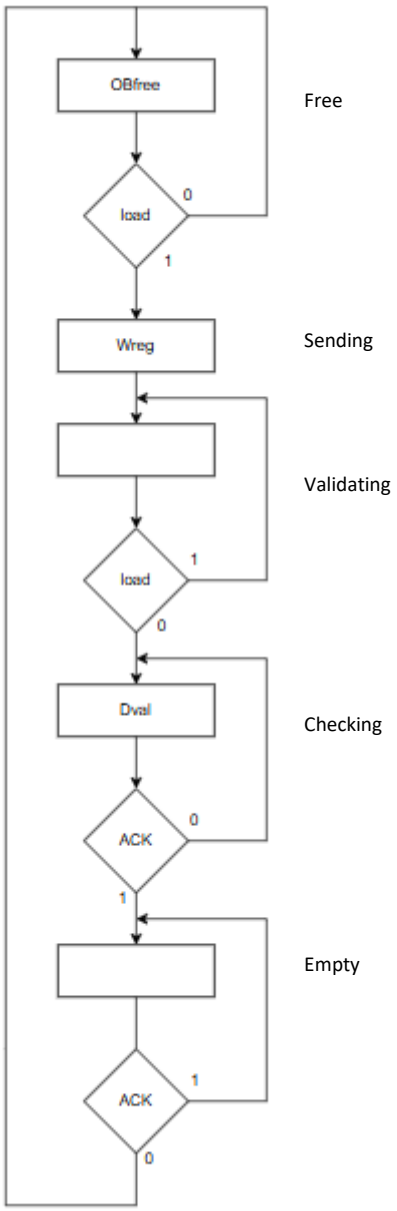


Figura 10– Máquina de estados do bloco *Buffer Control*

- O Buffer Control é uma máquina de estados com cinco estados, o Free, o Sending, o Validating, o Checking e o Empty. No primeiro estado, o sinal Obfree é ativo para armazenar dados, se o load estiver ativo, a máquina irá transitar de estado para Sending onde o W*reg* será ativado para ativar o clock do Output Register, transitando assim para o próximo estado Validating, sendo este responsável por verificar se o sinal load ainda se encontra a '1', caso este esteja a '0', passa ao próximo estado, Cheking.

  Neste estado o D*val* será ativado para indicar que há um valor válido a ser recebido, verificando se o sinal ACK está a '1', sendo este ativo quando a informação chegar.

  Por fim prossegue para o último estado Empty, sendo este responsável por verificar se a informação saiu, a partir do sinal ACK que caso esteja a '0' indica que a informação já partiu e retornando assim para o estado inicial

- O Output Register gerencia a transferência de dados do Output Buffer para o Control, garantindo que os dados sejam armazenados de forma segura até que possam ser utilizados.

5 Conclusão

Com este trabalho foi possível interligar a placa com o código em VHDL do Quartus, ao adicionarmos os blocos Ring Buffer e o *OutPut Buffer* ao *KeyBoard Reader* já é possível armazenar várias teclas num *Buffer* e enviar para o bloco de controlo no momento certo. O teclado 4x3 da placa foi fácil de utilizar e tornou o trabalho bastante interativo.

A. Descrição VHDL do bloco *Key Decode:*

Código VHDL do KeyDecode:

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

entity KeyDecode is
	port(
		Kack: in std_logic;
		Kval: out std_logic;
		KDecode : out std_logic_vector(3 downto 0);
		Lines : in std_logic_vector(3 downto 0);
		Cols : out std_logic_vector(2 downto 0);
		Clk : in std_logic;
		Reset : in std_logic
);
end KeyDecode;
architecture structural of KeyDecode is
component KeyScan is
	port(
		Kscan : in std_logic;
		Kpress : out std_logic;
		K : out std_logic_vector(3 downto 0);
		Lines : in std_logic_vector(3 downto 0);
		Cols : out std_logic_vector(2 downto 0);
		Osc : in std_logic;
		Reset : in std_logic
);
end component;
component KeyControl is
	port(
		Kack : in std_logic;
		Kpress : in std_logic;
		clk: in std_logic;
		rst: in std_logic;
		Kval: out std_logic;
		Kscan: out std_logic
);
```

```
end component;
component ClkDiv is
        port(
                clk_in: in std_logic;
                clk_out: out std_logic
);
end component;
signal SKscan : std_logic;
signal SKpress : std_logic;
signal Clock: std_logic;

begin
Clocke : ClkDiv port map (clk_in => Clk, clk_out => Clock);
Scan :  KeyScan port map (Kscan => SKscan , K => KDecode, Kpress => SKpress , Lines => Lines, Cols => Cols ,
Osc => Clock, Reset => Reset);
Control : KeyControl port map (Kack => Kack,Kpress => SKpress,clk=>Clock ,rst => Reset ,Kval => Kval  ,Kscan =>
SKscan);

end structural;
```

Código VHDL do KeyScan:

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

entity KeyScan is
        port(
                Kscan : in std_logic;
                Kpress : out std_logic;
                K : out std_logic_vector(3 downto 0);
                Lines : in std_logic_vector(3 downto 0);
                Cols : out std_logic_vector(2 downto 0);
                Osc : in std_logic;
                Reset : in std_logic
);
end KeyScan;
architecture structural of KeyScan is
component Counter is
```

```
        port
        (
        PL:in std_logic;
        CE:in std_logic;
        CLK:in std_logic;
        Data_in: in std_logic_vector(3 downto 0);
        RESET: in std_logic;
        TC: out std_logic;
        Q:out std_logic_vector(3 downto 0)
        );
end component;
component Decoder2x3 is
        port(
                Y : out std_logic_vector(2 downto 0);
                S0, S1 : in std_logic
);
end component;
component mux4x1 is
        port(
                A : in std_logic_vector(3 downto 0);
                S0, S1 : in std_logic;
                Y : out std_logic
);
end component;
signal interm_Q : std_logic_vector(3 downto 0);
signal Col : std_logic_vector(2 downto 0);
signal Yn : std_logic;

begin
Somador : Counter port map (Data_in(0) => '0', Data_in(1) => '0', Data_in(2) => '0',Data_in(3) => '0',PL=>'0',Clk => Osc , CE => Kscan, Reset => Reset, Q(0) => interm_Q(0) , Q(1) => interm_Q(1) , Q(2) => interm_Q(2) , Q(3) => interm_Q(3));
Decoder : Decoder2x3 port map (Y(0) => Col(0), Y(1) =>  Col(1), Y(2) =>  Col(2),S0 => interm_Q(2), S1 => interm_Q(3) );
Mux : mux4x1 port map (A=> Lines, S0 => interm_Q(0), S1 => interm_Q(1), Y => Yn);
K(0) <= interm_Q(0);
K(1) <= interm_Q(1);
K(2) <= interm_Q(2);
```

```
K(3) <= interm_Q(3);
Cols <= not Col(2 downto 0);
Kpress <= not Yn;

end structural;
```

Código VHDL do KeyControl:

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

entity KeyControl is
	port (
		Kack : in std_logic;
		Kpress : in std_logic;
		clk: in std_logic;
		rst: in std_logic;
		Kval: out std_logic;
		Kscan : out std_logic
);
end KeyControl;
architecture behavior of KeyControl is

type STATE_TYPE is (SCANNING, PRESSING, WAITING);

signal CurrentState, NextState: STATE_TYPE;

begin

-- Flip-Flop's
CurrentState <= SCANNING when rst = '1' else NextState when rising_edge(clk);

-- Generate next state
Generatenextstate:
process (CurrentState, Kpress, Kack)
	begin
		case CurrentState is
			when SCANNING		=> if (Kpress = '1') then
```

```
                                                    NextState <= PRESSING;
                                                else
                                                    NextState <= SCANNING;
                                                end if;

                        when PRESSING=> if (Kack = '0') then
                                                    NextState <= PRESSING;
                                                else
                                                    NextState <= WAITING;
                                                end if;

                        when WAITING  => if (Kack = '0' and Kpress = '0') then
                                                    NextState <= SCANNING;
                                                else
                                                    NextState <= WAITING;
                                                end if;

            end case;
end process;

-- Generate outputs
Kval <= '1' when (CurrentState = PRESSING)
else '0';

Kscan <= '1' when (CurrentState = SCANNING and Kpress = '0')
else '0';

end behavior;
```

Código VHDL do ClkDiv:

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity ClkDiv is
generic(div: natural := 50000000);
port ( clk_in: in std_logic;
```

```
                clk_out: out std_logic);
end ClkDiv;

architecture bhv of ClkDiv is

signal count: integer:=1;
signal tmp : std_logic := '0';

begin

process(clk_in)
begin

        if(clk_in'event and clk_in='1') then
                count <=count+1;
                if (count = div/2) then
                        tmp <= NOT tmp;
                        count <= 1;
                end if;
        end if;
end process;

clk_out <= tmp;

End bhv;
```

Código VHDL do Counter:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity Counter is
        port
        (
        PL:in std_logic;
        CE:in std_logic;
        CLK:in std_logic;
        Data_in: in std_logic_vector(3 downto 0);
        RESET: in std_logic;
```

```
        TC: out std_logic;
        Q:out std_logic_vector(3 downto 0)
        );
end counter;
architecture structural of counter is
component Mux2_1 is
        port(
        A: in std_logic_vector(3 downto 0);
        B: in std_logic_vector(3 downto 0);
        PL:in std_logic;
        D: out std_logic_vector(3 downto 0)
);
end component;
component Registo_KeyScan is
        port(
        D:in std_logic_vector(3 downto 0);
        CE: in std_logic;
        CLK:in std_logic;
        RST:in std_logic;
        Q:out std_logic_vector(3 downto 0)
);
end component;
component ADDER is
        port(
        A : in std_logic_vector(3 downto 0);
        B : in std_logic_vector(3 downto 0);
        Ci : in std_logic;
        S : out std_logic_vector(3 downto 0);
        Co : out std_logic
);
end component;

signal saidaMux: std_logic_vector(3 downto 0);
signal saidaReg: std_logic_vector(3 downto 0);
signal saidaSomador :std_logic_vector(3 downto 0);

begin
U1: Mux2_1 port map (A=>saidaSomador,B=>Data_in,PL=>PL,D=>saidaMux);
```

```
U2: Registo_KeyScan port map (CE=>CE, CLK=>CLK, D=>saidaMux, Q=>saidaReg,RST=>RESET);
U3: ADDER port map(A=>saidaReg, B(0)=>'1',B(1)=>'0',B(2)=>'0',B(3)=>'0',
S=>saidaSomador,Ci=>'0');
TC<=not Data_in(0) and not Data_in(1) and not Data_in(2) and not Data_in(3);
Q<=saidaReg;
end structural;
```

Código VHDL do Decoder2x3:

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

ENTITY Decoder2x3 IS
        port(
                Y : out std_logic_vector(2 downto 0);
                S0, S1 : in std_logic
);
END Decoder2x3;

ARCHITECTURE arch_Decoder of Decoder2x3 is
begin
Y(0) <= (not S0 and not S1);
Y(1) <= (S0 and not S1);
Y(2) <= (not S0 and S1);

end arch_Decoder;
```

Código VHDL do mux4x1:

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

ENTITY mux4x1 IS
        port(
                A : in std_logic_vector(3 downto 0);
                S0, S1 : in std_logic;
                Y : out std_logic
);
```

```
END mux4x1;

ARCHITECTURE arch_mux4x1 of mux4x1 is
begin

Y <= (not S0 and not S1 and A(0)) or (S0 and not S1 and A(1)) or (not S0 and S1 and A(2)) or (S0 and S1 and A(3));

end arch_mux4x1;
```

Código VHDL do Mux2_1:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity Mux2_1 is
port
(
A: in std_logic_vector(3 downto 0);
B: in std_logic_vector(3 downto 0);
PL:in std_logic;
D: out std_logic_vector(3 downto 0)
);
end Mux2_1;
architecture structural of Mux2_1 is
begin
D(0)<= (A(0) and not PL) or (B(0) and PL);
D(1)<= (A(1) and not PL) or (B(1) and PL);
D(2)<= (A(2) and not PL) or (B(2) and PL);
D(3)<= (A(3) and not PL) or (B(3) and PL);
end structural;
```

Código VHDL do FFD_KeyScan:

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

ENTITY FFD_KeyScan 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 FFD_KeyScan;

architecture logicFunction OF FFD_KeyScan IS

begin
Q <= '0' when RESET = '1' else '1' when SET = '1' else D WHEN rising_edge(clk) and EN = '1';
end logicFunction;
```

Código VHDL do Registo_KeyScan:

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

Entity Registo_KeyScan is
port(
        CLK : in std_logic;
        CE : in std_logic;
        RST : in std_logic;
        D : in std_logic_vector(3 downto 0);
        Q : out std_logic_vector(3 downto 0)
        );
end Registo_KeyScan;
ARCHITECTURE arch_Registo of Registo_KeyScan is
component FFD_KeyScan is
port(
        CLK : in std_logic;
        RESET : in std_logic;
        SET : in std_logic;
        EN : in std_logic;
        D : in std_logic;
        Q : out std_logic
        );
end component;
```

```
begin

FF1 : FFD_KeyScan port map ( SET => '0', CLK => CLK, EN => CE, D => D(0), Q =>Q(0), RESET => RST);

FF2 : FFD_KeyScan port map ( SET => '0', CLK => CLK, EN => CE, D => D(1), Q =>Q(1), RESET => RST);

FF3 : FFD_KeyScan port map ( SET => '0', CLK => CLK, EN => CE, D => D(2), Q =>Q(2), RESET => RST);

FF4 : FFD_KeyScan port map ( SET => '0', CLK => CLK, EN => CE, D => D(3), Q =>Q(3), RESET => RST);

end arch_registo;
```

Código VHDL do ADDER:

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

Entity ADDER is
        port(
                A  :in std_logic_vector(3 downto 0);
                B  :in std_logic_vector(3 downto 0);
                S  :out std_logic_vector(3 downto 0);
                Ci :in std_logic;
                Co :out std_logic
);
end ADDER;

architecture structural of ADDER is
component FADDER is
        port(
                A : in std_logic;
                B : in std_logic;
                Ci : in std_logic;
                S : out std_logic;
                CO : out std_logic
);
end component;
```

```
signal carry : std_logic_vector(3 downto 1);

begin

U1 : FADDER port map ( A => A(0), B => B(0), Ci => Ci, S => S(0), CO => carry(1));

U2 : FADDER port map ( A => A(1), B => B(1), Ci => carry(1), S => S(1), CO => carry(2));

u3 : FADDER port map ( A => A(2), B => B(2), Ci => carry(2), S => S(2), CO => carry(3));

u4 : FADDER port map ( A => A(3), B => B(3), Ci => carry(3), S => S(3), CO => CO);

end structural;
```

Código VHDL do HADDER:

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

Entity HADDER is
        port (
                        A :in std_logic;
                        B :in std_logic;
                        R :out std_logic;
                        Co :out std_logic

);
end HADDER;

architecture STRUCTUHADDER of HADDER is
begin

Co <= B and A;
R <= A xor B;

end STRUCTUHADDER;
```

Código VHDL do FADDER:

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

Entity FADDER is
port
(
        A  :in std_logic;
        B  :in std_logic;
        Ci :in std_logic;
        S  :out std_logic;
        CO :out std_logic
);
end FADDER;

architecture structural of FADDER is
component HADDER is
port
(
        A : in std_logic;
        B : in std_logic;
        R : out std_logic;
        Co: out std_logic
);

end component;

signal adder: std_logic;
signal carry: std_logic_vector(1 downto 0);

begin
U1: HADDER port map ( A => A, B => B, R => adder, Co => carry(0));
U2: HADDER port map ( A => adder, B => Ci, R => S, Co => carry(1));

Co <= carry(0) or carry (1);

end structural;
```

Código VHDL do KeyDecode_tb:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.all;
entity KeyDecode_tb is
end KeyDecode_tb;
    architecture behavior of KeyDecode_tb is
    component KeyDecode is
    port(
    Lines:in std_logic_vector(3 downto 0);
    Kack:in std_logic;
    Clk:in std_logic;
    Reset:in std_logic;
    Cols: out std_logic_vector(2 downto 0);
    KDecode: out std_logic_vector(3 downto 0);
    Kval: out std_logic
    );
    end component;
    signal Kack,Clk,Reset : std_logic := '0';
    signal Lines :std_logic_vector(3 downto 0) := "1111";
    signal Cols : std_logic_vector(2 downto 0) := "000" ;
    signal KDecode: std_logic_vector(3 downto 0) :="0000" ;
    signal Kval: std_logic := '0';
    begin
    Decode: KeyDecode port map (
    Lines=>Lines,
    Kack=>Kack,
    Clk=>Clk,
    Reset=>Reset,
    Cols=>Cols,
    KDecode=>KDecode,
    Kval=>Kval
    );
    Clk <= not Clk after 5 ns;
    tb:Process
    begin
```

```
    wait for 10 ns;
    Reset<='1';
    Lines<="0111";
    wait for 10 ns;
    Reset<='0';
    wait for 30 ns;
    Kack<='1';
    wait for 10 ns;
    Kack<= '0';
end process;
end;
```

Código VHDL do KeyScan_tb:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.all;
entity KeyScan_tb is
end KeyScan_tb;
architecture behavior of KeyScan_tb is
component KeyScan is
    port(
        Reset: in std_logic;
        Osc: in std_logic;
        Kscan: in std_logic;
        Lines: in std_logic_vector(3 downto 0);
        Cols: out std_logic_vector(2 downto 0);
        K:out std_logic_vector(3 downto 0);
        Kpress : out std_logic
    );
end component;
signal Reset,Osc, Kscan: std_logic := '0';
signal Lines: std_logic_vector(3 downto 0) := "0000";
signal Kpress: std_logic := '0';
signal Cols: std_logic_vector(2 downto 0) := "000";
signal K: std_logic_vector(3 downto 0) := "0000";
begin
```

```
Scan: KeyScan port map(
Reset => Reset,
Osc=>Osc,
Kscan => Kscan,
Lines => Lines,
Cols => Cols,
K => K,
Kpress => Kpress
);
Osc <= not Osc after 5 ns;
tb:Process
begin --110ns
wait for 10 ns;
Reset <= '1';
Lines <= "1101";
wait for 10 ns;
Reset <= '0';
Kscan <= '1';
wait for 30 ns;
Kscan <= '0';
wait for 10 ns;
Lines <= "1011";
Kscan <= '1';
wait for 50 ns;
end process;
end;
```

Código VHDL do KeyControl_tb:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.all;
entity KeyControl_tb is
end KeyControl_tb;
architecture behavior of KeyControl_tb is
component Key_control is
 port(
```

```
 Kack: in std_logic;
Kpress: in std_logic;
Kval: out std_logic;
Kscan: out std_logic;
clk: in std_logic;
rst: in std_logic
 );
end component;
signal Kack,Kpress,clk,rst: std_logic := '0';
signal Kval: std_logic := '0';
signal Kscan: std_logic := '0';
begin
Control: Key_control port map(
 Kack=>Kack,
Kpress=>Kpress,
Kval=>Kval,
Kscan=>Kscan,
 clk=>clk,
 rst=>rst
);
clk <= not clk after 5 ns;
tb:Process
begin
 wait for 10 ns;
 rst<='1';
 wait for 10 ns;
 rst<='0';
 wait for 10 ns;
 Kpress<= '1';
wait for 10ns;
Kack<= '1';
wait for 10ns;
Kack<= '0';
Kpress<= '1';
wait for 10ns;
Kpress<= '0';
end process;
end;
```

Código VHDL do ADDER_tb:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.all;
entity ADDER_tb is
end ADDER_tb;
architecture behavior of ADDER_tb is
component ADDER is
   port(
      A : in std_logic_vector(3 downto 0);
      B : in std_logic_vector(3 downto 0);
      Ci : in std_logic;
      S : out std_logic_vector(3 downto 0);
      Co : out std_logic
      );
end component;
signal A,B: std_logic_vector(3 downto 0) := "0000";
signal Ci: std_logic :='0';
signal Co: std_logic := '0';
signal S: std_logic_vector(3 downto 0) := "0000";
begin
ADD: ADDER port map(
A=>A,
B=>B,
Ci=>Ci,
S=>S,
Co=>Co
);
tb:Process
begin
wait for 10 ns;
A<="0010";
B<= "0101";
Ci <= '0';
wait for 10 ns;
A<="1000";
```

```
B<= "0101";
Ci <= '1';
wait for 10 ns;
A<="1100";
B<= "0101";
Ci <= '0';
end process;
end;
```

Código VHDL do Counter_tb:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity Counter_tb is
end Counter_tb;

architecture behavior of Counter_tb is

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

   signal PL, CE, CLK, RESET: std_logic := '0';
   signal Data_in: std_logic_vector(3 downto 0) := "0000";
   signal TC: std_logic;
   signal Q: std_logic_vector(3 downto 0);

begin
```

```
UUT: Counter port map(
    PL => PL,
    CE => CE,
    CLK => CLK,
    Data_in => Data_in,
    RESET => RESET,
    TC => TC,
    Q => Q
);

CLK <= not CLK after 5 ns;

tb: process
begin
    wait for 10 ns;
    RESET <= '1';
    wait for 10 ns;
    RESET <= '0';
    wait for 10 ns;
    PL <= '1';
    wait for 10 ns;
    CE <= '1';
    wait for 10 ns;
    Data_in <= "0001";
    wait for 10 ns;
    CE <= '0';
    wait for 10 ns;
    PL <= '0';
    wait for 10 ns;
    Data_in <= "0010";
    wait for 10 ns;
    Data_in <= "0100";
    wait for 10 ns;
    Data_in <= "1000";
    wait for 10 ns;
    CE <= '1';
    wait for 10 ns;
```

```
        CE <= '0';
        wait for 10 ns;
        RESET <= '1';
        wait for 10 ns;
        RESET <= '0';
    end process;

end behavior;
```

Código VHDL do Decoder2x3_tb:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Decoder2x3_tb IS
END Decoder2x3_tb;

ARCHITECTURE behavior OF Decoder2x3_tb IS

    COMPONENT Decoder2x3 IS
        PORT(
            Y : out std_logic_vector(2 downto 0);
            S0, S1 : in std_logic
        );
    END COMPONENT;

    SIGNAL S0, S1 : std_logic := '0';
    SIGNAL Y : std_logic_vector(2 downto 0);

BEGIN

    UUT: Decoder2x3 PORT MAP(
        Y => Y,
        S0 => S0,
        S1 => S1
    );

    tb: PROCESS
```

```
    BEGIN
        wait for 10 ns;
        S0 <= '0';
        S1 <= '0';
        wait for 10 ns;
        S0 <= '1';
        S1 <= '0';
        wait for 10 ns;
        S0 <= '0';
        S1 <= '1';
        wait for 10 ns;
        S0 <= '1';
        S1 <= '1';
        wait for 10 ns;
        S0 <= '0';
        S1 <= '0';
        wait for 10 ns;
        S0 <= '1';
        S1 <= '0';
        wait for 10 ns;
        S0 <= '0';
        S1 <= '1';
        wait for 10 ns;
        S0 <= '1';
        S1 <= '1';
    END PROCESS;

END behavior;
```

Código VHDL do mux4x1_tb:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY mux4x1_tb IS
END mux4x1_tb;

ARCHITECTURE behavior OF mux4x1_tb IS
```

```
    COMPONENT mux4x1 IS
        PORT(
            A : in std_logic_vector(3 downto 0);
            S0, S1 : in std_logic;
            Y : out std_logic
        );
    END COMPONENT;

    SIGNAL A : std_logic_vector(3 downto 0) := "0000";
    SIGNAL S0, S1 : std_logic := '0';
    SIGNAL Y : std_logic;

BEGIN

    UUT: mux4x1 PORT MAP(
        A => A,
        S0 => S0,
        S1 => S1,
        Y => Y
    );

    tb: PROCESS
    BEGIN
        wait for 10 ns;
        S0 <= '0';
        S1 <= '0';
        wait for 10 ns;
        S0 <= '1';
        S1 <= '0';
        wait for 10 ns;
        S0 <= '0';
        S1 <= '1';
        wait for 10 ns;
        S0 <= '1';
        S1 <= '1';
        wait for 10 ns;
        A <= "0001";
```

```
        wait for 10 ns;
        A <= "0010";
        wait for 10 ns;
        A <= "0100";
        wait for 10 ns;
        A <= "1000";
        wait for 10 ns;
        S0 <= '0';
        S1 <= '0';
        wait for 10 ns;
        S0 <= '1';
        S1 <= '0';
        wait for 10 ns;
        S0 <= '0';
        S1 <= '1';
        wait for 10 ns;
        S0 <= '1';
        S1 <= '1';
    END PROCESS;

END behavior;
```

B. Atribuição de pinos do módulo *Keyboard Reader*

```
set_global_assignment -name FAMILY "MAX 10 FPGA"
set_global_assignment -name DEVICE 10M50DAF484C6GES
set_global_assignment -name TOP_LEVEL_ENTITY "DE10_Lite"

#linhas
set_location_assignment PIN_W5 -to Lines[0]
set_location_assignment PIN_AA14 -to Lines[1]
set_location_assignment PIN_W12 -to Lines[2]
set_location_assignment PIN_AB12 -to Lines[3]

#colunas
set_location_assignment PIN_AB11 -to Cols[0]
set_location_assignment PIN_AB10 -to Cols[1]
set_location_assignment PIN_AA9 -to Cols[2]

#Leds
set_location_assignment PIN_A8 -to QValue[0]
set_location_assignment PIN_A9 -to QValue[1]
set_location_assignment PIN_A10 -to QValue[2]
set_location_assignment PIN_B10 -to QValue[3]
set_location_assignment PIN_D13 -to Dval

#sw
set_location_assignment PIN_C11 -to Reset
set_location_assignment PIN_D12 -to ACK
```

C. Código *Kotlin - HAL*

```kotlin
import isel.leic.UsbPort
// Virtualiza o acesso ao sistema UsbPort
object HAL {
    var lastOutput = 0
    // Inicia a classe
    fun init () {
        lastOutput = 0
        UsbPort.write(lastOutput)
    }
    // Retorna true se o bit tiver o valor lógico '1'
    fun isBit (mask:Int) : Boolean {
        return UsbPort.read().and(mask) != 0
    }
    // Retorna os valores dos bits representados por mask presentes no UsbPort
    fun readBits(mask: Int): Int{
        return UsbPort.read().and(mask)
    }
    // Escreve nos bits representados por mask os valores dos bits correspondentes
em value
    fun writeBits(mask: Int, value: Int) {
        clrBits(mask)
        setBits(value and mask)
    }
    // Coloca os bits representados por mask no valor lógico '1'
    fun setBits (mask: Int) {
        lastOutput = mask.or(lastOutput)
        UsbPort.write(lastOutput)
    }
    // Coloca os bits representados por mask no valor lógico '0'
    fun clrBits(mask:Int) {
        lastOutput = lastOutput.and(0xFF - mask)
        UsbPort.write(lastOutput)
    }
}
```

D. Código *Kotlin - KBD*

```kotlin
import isel.leic.utils.Time
// Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.
object KBD {
    const val KEY = 0X0F
    const val NONE = ' '
    const val KACK_MASK = 0X80
    const val KVAL_MASK = 0X10
    val keys: CharArray = charArrayOf('1', '4', '7', '*', '2', '5', '8', '0', '3',
'6', '9', '#')

    fun init() {//Inicia a classe
        HAL.init()
    }
    //Retorna de imediato a tecla premida ou NONE se não há tecla premida.
    fun getKey(): Char {
        if (HAL.isBit(KVAL_MASK)) {
            val tecla = HAL.readBits(KEY)
            HAL.setBits(KACK_MASK)
            if(!HAL.isBit(KVAL_MASK)) {
                HAL.clrBits(KACK_MASK)
                return keys[tecla]
            }
        }
        return NONE
    }
    //Retorna a tecla premida, caso ocorra antes do 'timeout' (representado em
milissegundos), ou NONE caso contrário.
    fun waitKey(timeout: Long): Char {
        val timeFinal = timeout + Time.getTimeInMillis() //tempo atual em
milisegundos + timeout
        var key = NONE
        do {
            key = getKey()
        } while (Time.getTimeInMillis() < timeFinal && key == NONE)
        return key
    }
}
```