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*

# 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 *Ring Buffer*), define que o sinal *Kval* é ativado quando é detetada a pressão de uma tecla, sendo também disponibilizado o código dessa tecla no barramento *K0:3*. Apenas é iniciado um novo ciclo de varrimento ao teclado quando o sinal *Kack* 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. Escolhemos a versão I pois foi a que o professor recomendou e ao longo do desenvolvimento do trabalho consideramos ser uma versão que nos agradou.

Uma imagem com texto, diagrama, Esquema, Desenho técnico

Descrição gerada automaticamente

Figura 3 - Diagrama de blocosdo bloco *Key Scan*

O bloco *Key Control* foi implementado pela máquina de estados representada em *ASM-chart* na Figura 4.

Uma imagem com diagrama, esboço, desenho, Esquema

Descrição gerada automaticamente

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

A descrição hardware do bloco *Key Decode* em VHDL encontra-se no anexo D.

# Interface com o *Control*

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



Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

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

*HAL* e *KBD* desenvolvidos são descritos nas secções 2.1. e 2.2, e o código fonte desenvolvido nos Anexos O e P, respetivamente.

## *HAL*

É a interação entre usbPort e o software.

Desenvolvemos 6 funções:

### fun init()

Inicia o usbPort a 0 e atualiza o lastState para 0.

### fun readBits()

Lê o valor de output nos bits correspondentes a máscara enquanto restantes bits retornam a 0.

### fun isBits()

Verifica o valor de um bit em específico.

### fun setBits()

Coloca valores correspondentes a máscara a 1 e deixa os restantes como estavam.

### fun clrBits()

Coloca valores correspondentes à máscara a 0 e deixa os restantes como estavam.

### fun writeBits()

Tem como função escrever valor recebido nos bits indicados pela máscara (restantes bits mantém-se).

A utilização da linha de código:

**lastState = lastState.or(mask)**

nas funções *setBits*, *clrBits* e *writeBits* serve para manter valor de *lastState* atualizado.

## *KBD*

O desenvolvimento o software de KBD é composto por 3 funções:

### fun init()

Inicia escrita e leitura do usbPort chamando o *HAL.init* e faz o *clearBits(HAL.clrBits)* à máscara do *Kack* (para este começar a 0).

### fun getKey()

Lê o código da máscara que está no valor de cada tecla e transforma-o no devido caracter caso seja válida, é dado *ack* da tecla, espera tecla ser validada e devolve o char correspondente á tecla que leu. Isto no caso de *Kval* estar ativo. Se *Kval* estiver inativo retorna NONE.

### Uma imagem com diagrama, Desenho técnico, Esquema, esquemático Descrição gerada automaticamentefun waitKey()

Recebe período (timeout) e tem um ciclo que tenta ler teclas. Se ler uma tecla válida retorna essa tecla. Se não, continua à procura até o tempo de o ciclo exceder e retorna NONE.

## *Ring Buffer*

O bloco *Ring Buffer* desenvolvido 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.

A escrita de dados no *Ring Buffer* inicia-se com a ativação do sinal DAV (*Data Available*) pelo sistema produtor, neste caso pelo *Key Decode*, indicando que tem dados para serem armazenados. Logo que tenha disponibilidade para armazenar informação, o *Ring Buffer* escreve os dados D0:3 em memória. Concluída a escrita em memória ativa o sinal DAC (*Data Accepted*) para informar o sistema produtor que os dados foram aceites. O sistema produtor mantém o sinal DAV ativo até que DAC seja ativado. O *Ring* *Buffer* só desativa DAC depois de DAV ter sido desativado.

A implementação do *Ring Buffer é* baseada numa memória RAM (*Random Access Memory*). O endereço de

escrita/leitura, selecionado por 𝑝𝑢𝑡get, é definido pelo bloco *Memory Address Control* (MAC) composto por dois

registos, que contêm o endereço de escrita e leitura, designados por *putIndex* e *getIndex* respetivamente. O MAC suporta assim ações de *incPut* e *incGet*, gerando informação se a estrutura de dados está cheia (*full*) ou se está vazia (e*mpty*). O bloco *Ring* *Buffer* procede à entrega de dados à entidade consumidora, sempre que esta indique que está disponível para receber, através do sinal *Clear To Send* (CTS). Na Figura 6 é apresentado o diagrama de blocos para uma estrutura do bloco *Ring Buffer*.

Uma imagem com texto, captura de ecrã, diagrama, ecrã

Descrição gerada automaticamente

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

## Figura 7 – Diagrama de blocos do bloco *Memory Address Control*

Uma imagem com diagrama, Esquema, Desenho técnico, esquemático

Descrição gerada automaticamente

Figura 8 – Máquina de estados do *Ring Buffer Control*

## *Output Buffer*

O bloco *Output Buffer* do *Keyboard Reader* é responsável pela interação com o sistema consumidor, neste caso o módulo

*Control*.

O *Output Buffer* indica que está disponível para armazenar dados através do sinal *OBfree*. Assim, nesta situação o sistema produtor pode ativar o sinal *Load* para registar os dados.

O *Control* quando pretende ler dados do *Output Buffer*, aguarda que o sinal *Dval* fique ativo, recolhe os dados e pulsa o sinal *ACK* indicando que estes já foram consumidos.

O *Output Buffer*, logo que o sinal *ACK* pulse, deve invalidar os dados baixando o sinal *Dval* e sinalizar que está novamente disponível para entregar dados ao sistema consumidor, ativando o sinal *OBfree*. Na Figura 9, é apresentado o diagrama de blocos do *Output Buffer*.

Uma imagem com texto, captura de ecrã, diagrama, Retângulo

Descrição gerada automaticamente

Figura 9 - Diagrama de blocos do bloco *Output Buffer*

Uma imagem com diagrama, texto, Esquema, Desenho técnico

Descrição gerada automaticamente

Figura 10 – Máquina de estados do *Buffer Control*

# *Conclusões*

Por o clock da placa ser 50MHz, estava demasiado rápido para o propósito desejado, pelo que utilizámos

o CLK\_div com div = 1000, para que fosse mais lento e atingisse o objetivo.

1. Descrição VHDL do bloco Key Scan

library ieee;

use ieee.std\_logic\_1164.all;

entity KeyScan is

port(

KScan, CLK, Reset: in std\_logic;

I : in std\_logic\_vector(3 downto 0);

Kpress : out std\_logic;

O : out std\_logic\_vector(2 downto 0);

K : out std\_logic\_vector(3 downto 0));

end KeyScan;

architecture arc\_Keyscan of KeyScan is

component Counter

port(

PL, CE, CLK, Reset: in std\_logic;

Data\_in: in std\_logic\_vector(3 downto 0);

TC: out std\_logic;

Q: out std\_logic\_vector(3 downto 0));

end component;

component Decoder

port(

S: in std\_logic\_vector(1 downto 0);

O: out std\_logic\_vector(2 downto 0));

end component;

component MUX4x1

port(

S: in std\_logic\_vector(1 downto 0);

I: in std\_logic\_vector(3 downto 0);

Y: out std\_logic);

end component;

signal kp : std\_logic;

signal col : std\_logic\_vector(2 downto 0);

signal qcount : std\_logic\_vector(3 downto 0);

begin

count: Counter port map(

Reset => Reset,

PL => '0',

Data\_in => "0000",

CE => KScan,

CLK => CLK,

Q => qcount);

dec: Decoder port map(

S(1) => qcount(3),

S(0) => qcount(2),

O => col);

mux: MUX4x1 port map(

S(1) => qcount(1),

S(0) => qcount(0),

I => I,

Y => kp);

O <= col;

Kpress <= not kp;

K <= qcount;

end arc\_Keyscan;

1. Descrição VHDL do bloco *Key Control*

library ieee;

use ieee.std\_logic\_1164.all;

entity KeyControl is

port(

Kpress, Kack, CLK, Reset: in std\_logic;

Kval, Kscan: out std\_logic);

end KeyControl;

architecture arcKC of KeyControl is

type STATE\_TYPE is (WAIT\_FOR\_PRESS, VALIDATE, WAIT\_FOR\_AFK);

signal CurrentState, NextState: STATE\_TYPE;

begin

CurrentState <= WAIT\_FOR\_PRESS when Reset = '1' else NextState when rising\_edge(CLK);

GenerateNextState:

process (CurrentState, Kpress, Kack)

begin

case CurrentState is

when WAIT\_FOR\_PRESS =>

if (Kpress = '1') then NextState <= VALIDATE;

else NextState <= WAIT\_FOR\_PRESS;

end if;

when VALIDATE =>

if (Kack = '1' and Kpress = '0') then NextState <= WAIT\_FOR\_AFK;

else NextState <= VALIDATE;

end if;

when WAIT\_FOR\_AFK =>

if (Kack = '0') then NextState <= WAIT\_FOR\_PRESS;

else NextState <= WAIT\_FOR\_AFK;

end if;

end case;

end process;

Kval <= '1' when ( CurrentState = VALIDATE) else '0';

Kscan <= '1' when ( CurrentState = WAIT\_FOR\_PRESS and Kpress = '0') else '0';

end arcKC;

1. Descrição VHDL do bloco *CLK Div*

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;

1. Descrição VHDL do bloco *Key Decode*

library ieee;

use ieee.std\_logic\_1164.all;

entity KeyDecode is

port(

Kack , CLK, Reset: in std\_logic;

I : in std\_logic\_vector(3 downto 0);

Kval : out std\_logic;

O : out std\_logic\_vector(2 downto 0);

K : out std\_logic\_vector(3 downto 0));

end KeyDecode;

architecture arc\_kd of KeyDecode is

component CLKDIV

generic(div: natural := 50000000);

port ( clk\_in: in std\_logic;

clk\_out: out std\_logic);

end component;

component KeyScan

port(

KScan, CLK, Reset: in std\_logic;

I : in std\_logic\_vector(3 downto 0);

Kpress : out std\_logic;

O : out std\_logic\_vector(2 downto 0);

K : out std\_logic\_vector(3 downto 0));

end component;

component KeyControl

port(

Kpress, Kack, CLK, Reset: in std\_logic;

Kval, Kscan: out std\_logic);

end component;

signal kp, ks, clockm : std\_logic;

begin

clock: clkDIV generic map(1000)

port map(

clk\_in => CLK,

clk\_out => clockm);

scan : KeyScan port map(

KScan => ks,

CLK => clockm,

Reset => Reset,

I => I,

Kpress => kp,

O => O,

K => K);

control : KeyControl port map(

Kpress => kp,

Kack => Kack,

CLK => clockm,

Reset => Reset,

Kval => Kval,

KScan => ks);

end arc\_kd;

1. Descrição VHDL do sub-bloco full add

library ieee;

use ieee.std\_logic\_1164.all;

entity full\_add is

Port(A, B, Cin: in std\_logic;

Cout, S: out std\_logic);

end full\_add;

architecture arc\_fa of full\_add is

begin

S <= A xor B xor Cin;

Cout <= (A and B) or (A and Cin) or (B and Cin);

end arc\_fa;

1. Descrição VHDL do sub-bloco adder

library ieee;

use ieee.std\_logic\_1164.all;

entity adder is

Port(A, B :in std\_logic\_vector(3 downto 0);

Ci: in std\_logic;

Co: out std\_logic;

S: out std\_logic\_vector(3 downto 0));

end adder;

architecture arc\_adder of adder is

component full\_add

Port(A, B, Cin: in std\_logic;

Cout, S: out std\_logic);

end component;

signal c1, c2, c3: std\_logic;

begin

fa1: full\_add port map(

A => A(0),

B => B(0),

Cin => Ci,

S => S(0),

Cout => c1);

fa2: full\_add port map(

A => A(1),

B => B(1),

Cin => C1,

S => S(1),

Cout => c2);

fa3: full\_add port map(

A => A(2),

B => B(2),

Cin => C2,

S => S(2),

Cout => c3);

fa4: full\_add port map(

A => A(3),

B => B(3),

Cin => C3,

S => S(3),

Cout => Co);

end arc\_adder;

1. Descrição VHDL do sub-bloco Counter

library ieee;

use ieee.std\_logic\_1164.all;

-- CC -> contador crescente!

entity Counter is

port(

PL, CE, CLK, Reset: in std\_logic;

Data\_in: in std\_logic\_vector(3 downto 0);

TC: out std\_logic;

Q: out std\_logic\_vector(3 downto 0));

end Counter;

architecture arc\_cc of Counter is

component adder

Port(A, B :in std\_logic\_vector(3 downto 0);

Ci: in std\_logic;

Co: out std\_logic;

S: out std\_logic\_vector(3 downto 0));

end component;

component MUX2x1

port(A, B: in std\_logic\_vector(3 downto 0);

S: in std\_logic;

Y: out std\_logic\_vector(3 downto 0));

end component;

component Registry

port(

D:in std\_logic\_vector(3 downto 0);

CLK, E, Reset: in std\_logic;

Q: out std\_logic\_vector(3 downto 0));

end component;

component Terminal\_Count

port(

Q : in std\_logic\_vector(3 downto 0);

TC: out std\_logic);

end component;

signal outadder, outmux, outreg: std\_logic\_vector(3 downto 0);

begin

ad: adder port map(

A => outreg,

B => "0000",

Ci => CE,

S => outadder);

mux: MUX2x1 port map(

A => Data\_in,

B => outadder,

S => PL,

Y => outmux);

reg: Registry port map(

Reset => Reset,

E => '1',

D => outmux,

CLK => CLK,

Q => outreg);

Q <= outreg;

utc: Terminal\_Count port map(

Q => outreg,

TC => TC);

end arc\_cc;

1. Descrição VHDL do sub-bloco MUX 2x1

library ieee;

use ieee.std\_logic\_1164.all;

entity MUX2x1 is

port(A, B: in std\_logic\_vector(3 downto 0);

S: in std\_logic;

Y: out std\_logic\_vector(3 downto 0));

end MUX2x1;

architecture arc\_mux of MUX2x1 is

begin

Y(0)<=(S and A(0)) or (not S and B(0));

Y(1)<=(S and A(1)) or (not S and B(1));

Y(2)<=(S and A(2)) or (not S and B(2));

Y(3)<=(S and A(3)) or (not S and B(3));

end arc\_mux;

1. Descrição VHDL do sub-bloco Memory Address Control

library ieee;

use ieee.std\_logic\_1164.all;

entity MemoryAddressControl is

port(

putget, incPut, incGet, CLK, Reset : in std\_logic;

full, empty : out std\_logic;

A : out std\_logic\_vector(2 downto 0));

end MemoryAddressControl;

architecture arc of MemoryAddressControl is

component Counter

port(

PL, CE, CLK, Reset: in std\_logic;

Data\_in: in std\_logic\_vector(3 downto 0);

TC: out std\_logic;

Q: out std\_logic\_vector(3 downto 0));

end component;

component Mux2x1

port(A, B: in std\_logic\_vector(3 downto 0);

S: in std\_logic;

Y: out std\_logic\_vector(3 downto 0));

end component;

signal putIndex, getIndex, muxout : std\_logic\_vector(3 downto 0);

signal r, state\_decider, equals : std\_logic;

begin

--r <= Reset;

-- contador do indice no qual o put se encontra

incrementPut: Counter port map(

PL => '0',--r,

CE => incPut,

CLK => CLK,

Reset => Reset,

Data\_in => "0000",

Q => putIndex);

-- contador do indice no qual o get se encontra

incrementGet: Counter port map(

PL => '0',--r,

CE => incGet,

CLK => CLK,

Reset => Reset,

Data\_in => "0000",

Q => getIndex);

-- decide dependedendo do sinal putget qual indice indicar como output

idxDecider : Mux2x1 port map(

A => putIndex,

B => getIndex,

S => putget,

Y => muxout);

A <= muxout(2 downto 0);

state\_decider <= putIndex(3) xor getIndex(3);

equals <= (putIndex(2) xnor getIndex(2)) and (putIndex(1) xnor getIndex(1)) and (putIndex(0) xnor getIndex(0));

--caso os indices put e get sejam iguais, o quarto bit indica se a ram está vazia ou cheia,

--pela lógica de que se forem iguais quer dizer que está vazio e se forem diferentes, quer dizer

--que está cheio

empty <= not state\_decider and equals;

full <= state\_decider and equals;

end arc;

1. Descrição VHDL do sub-bloco Ring Buffer Control

library ieee;

use ieee.std\_logic\_1164.all;

entity RingBufferControl is

port(

DAV, CTS, full, empty, Reset, CLK : in std\_logic;

Wr, selPG, Wreg, DAC, incPut, incGet : out std\_logic);

end RingBufferControl;

architecture arc of RingBufferControl is

type STATE\_TYPE is (ZEN, SELECT\_PUT, WRITE\_RAM, INC\_PUT, ACCEPTED, WRITE\_REG, INC\_GET);

signal CurrentState, NextState: STATE\_TYPE;

begin

CurrentState <= ZEN when Reset = '1' else NextState when rising\_edge(CLK);

GenerateNextState:

process(CurrentState,DAV, full, empty, CTS)

begin

case CurrentState is

when ZEN =>

if (DAV = '1' and full = '0') then NextState <= SELECT\_PUT;

elsif(DAV = '1' and full = '1' and CTS = '1') then NextState <= WRITE\_REG;

elsif(DAV = '0' and CTS = '1' and empty = '0') then NExtState <= WRITE\_REG;

else NextState <= ZEN;

end if;

when SELECT\_PUT => NextState <= WRITE\_RAM;

when WRITE\_RAM => NextState <= INC\_PUT;

when INC\_PUT => NextState <= ACCEPTED;

when ACCEPTED =>

if (DAV = '1') then NextState <= ACCEPTED;

else NextState <= ZEN;

end if;

when WRITE\_REG =>

if (CTS = '1') then NextState <= WRITE\_REG;

else NextState <= INC\_GET;

end if;

when INC\_GET => NextState <= ZEN;

end case;

end process;

selPG <= '1' when (CurrentState = SELECT\_PUT or CurrentState = WRITE\_RAM) else '0';

Wr <= '1' when (CurrentState = WRITE\_RAM) else '0';

incPut <= '1' when (CurrentState = INC\_PUT) else '0';

DAC <= '1' when (CurrentState = ACCEPTED) else '0';

Wreg <= '1' when (CurrentState = WRITE\_REG) else '0';

incGet <= '1' when (CurrentState = INC\_GET) else '0';

end arc;

1. Descrição VHDL do bloco Ring Buffer

library ieee;

use ieee.std\_logic\_1164.all;

entity RingBuffer is

port(

DAV, CTS, CLK, Reset : in std\_logic;

D : in std\_logic\_vector(3 downto 0);

Wreg, DAC : out std\_logic;

Q : out std\_logic\_vector(3 downto 0));

end RingBuffer;

architecture arc of RingBuffer is

component MemoryAddressControl

port(

putget, incPut, incGet, CLK, Reset : in std\_logic;

full, empty : out std\_logic;

A : out std\_logic\_vector(2 downto 0));

end component;

component RAM

generic(

ADDRESS\_WIDTH : natural := 3;

DATA\_WIDTH : natural := 4

);

port(

address : in std\_logic\_vector(ADDRESS\_WIDTH - 1 downto 0);

wr: in std\_logic;

din: in std\_logic\_vector(DATA\_WIDTH - 1 downto 0);

dout: out std\_logic\_vector(DATA\_WIDTH - 1 downto 0));

end component;

component RingBufferControl

port(

DAV, CTS, full, empty, Reset, CLK : in std\_logic;

Wr, selPG, Wreg, DAC, incPut, incGet : out std\_logic);

end component;

signal putget, incp, incg, f, e, wram : std\_logic;

signal idx : std\_logic\_vector(2 downto 0);

begin

control : RingBufferControl port map(

Reset => Reset,

CLK => CLK,

DAV => DAV,

CTS => CTS,

full => f,

empty => e,

Wr => wram,

selPG => putget,

Wreg => Wreg,

DAC => DAC,

incPut => incp,

incGet => incg);

mac : MemoryAddressControl port map(

putget => putget,

incPut => incp,

incGet => incg,

CLK => CLK,

Reset => Reset,

full => f,

empty => e,

A => idx);

memory : RAM port map(

address => idx(2 downto 0),

wr => wram,

din => D,

dout => Q);

end arc;

# Descrição VHDL do sub-*bloco Buffer Control*

library ieee;

use ieee.std\_logic\_1164.all;

entity BufferControl is

port(

CLK, Reset, Load, ACK : in std\_logic;

Wreg, OBfree, Dval : out std\_logic);

end BufferControl;

architecture arc of BufferControl is

type STATE\_TYPE is (CAN\_RECEIVE, WRITE\_DATA, SEND\_DATA, WAIT\_ACK);

signal CurrentState, NextState: STATE\_TYPE;

begin

CurrentState <= CAN\_RECEIVE when Reset = '1' else NextState when rising\_edge(CLK);

GenerateNextState:

process(Load, ACK)

begin

case CurrentState is

when CAN\_RECEIVE =>

if (Load = '1') then NextState <= WRITE\_DATA;

else NextState <= CAN\_RECEIVE;

end if;

when WRITE\_DATA =>

if (Load = '0') then NextState <= SEND\_DATA;

else NextState <= WRITE\_DATA;

end if;

when SEND\_DATA =>

if (ACK = '0') then NextState <= SEND\_DATA;

else NextState <= WAIT\_ACK;

end if;

when WAIT\_ACK =>

if(ACK = '1') then NextState <= WAIT\_ACK;

else NextState <= CAN\_RECEIVE;

end if;

end case;

end procesS;

OBfree <= '1' when (CurrentState = CAN\_RECEIVE) else '0';

Dval <= '1' when (CurrentState = SEND\_DATA) else '0';

Wreg <= '1' when (CurrentState = WRITE\_DATA) else '0';

end arc;

# Descrição VHDL do sub-*bloco Output Buffer*

library ieee;

use ieee.std\_logic\_1164.all;

entity OutputBuffer is

port(

Load, ACK, CLK, Reset : in std\_logic;

D : in std\_logic\_vector(3 downto 0);

OBfree, Dval : out std\_logic;

Q : out std\_logic\_vector(3 downto 0));

end OutputBuffer;

architecture arc of OutputBuffer is

component BufferControl

port(

CLK, Reset, Load, ACK : in std\_logic;

Wreg, OBfree, Dval : out std\_logic);

end component;

component Registry

port(

E, CLK, Reset : in std\_logic;

D : in std\_logic\_vector(3 downto 0);

Q : out std\_logic\_vector(3 downto 0));

end component;

signal w : std\_logic;

begin

control : BufferControl port map(

CLK => CLK,

Reset => Reset,

Load => Load,

ACK => ACK,

Wreg => w,

OBfree => OBfree,

Dval => Dval);

outreg : Registry port map(

E => w,

CLK => CLK,

Reset => Reset,

D => D,

Q => Q);

end arc;

1. Descrição VHDL do bloco *Keyboard Reader*

library ieee;

use ieee.std\_logic\_1164.all;

entity KeyControl is

port(

Kpress, Kack, CLK, Reset: in std\_logic;

Kval, Kscan: out std\_logic);

end KeyControl;

architecture arcKC of KeyControl is

type STATE\_TYPE is (WAIT\_FOR\_PRESS, VALIDATE, WAIT\_FOR\_AFK);

signal CurrentState, NextState: STATE\_TYPE;

begin

CurrentState <= WAIT\_FOR\_PRESS when Reset = '1' else NextState when rising\_edge(CLK);

GenerateNextState:

process (CurrentState, Kpress, Kack)

begin

case CurrentState is

when WAIT\_FOR\_PRESS =>

if (Kpress = '1') then NextState <= VALIDATE;

else NextState <= WAIT\_FOR\_PRESS;

end if;

when VALIDATE =>

if (Kack = '1' and Kpress = '0') then NextState <= WAIT\_FOR\_AFK;

else NextState <= VALIDATE;

end if;

when WAIT\_FOR\_AFK =>

if (Kack = '0') then NextState <= WAIT\_FOR\_PRESS;

else NextState <= WAIT\_FOR\_AFK;

end if;

end case;

end process;

Kval <= '1' when ( CurrentState = VALIDATE) else '0';

Kscan <= '1' when ( CurrentState = WAIT\_FOR\_PRESS and Kpress = '0') else '0';

end arcKC;

1. Código *Kotlin* - *HAL*

import isel.leic.UsbPort

object HAL {

private var lastState = 0

private var initialize = false

// inicia o usbport a 0 e atualiza o lastState para 0

fun init() {

if (!initialize) {

UsbPort.write(0)

lastState = 0

initialize = true

}

}

// le o(s) bit(s) indicado(s) na máscara

fun readBits(mask: Int) = UsbPort.read().and(mask)

// verifica se o bit indicado na máscara está on ou off

fun isBit(mask: Int) = UsbPort.read().and(mask) != 0

// coloca o(s) bit(s) indicado(s9 na máscara a on

fun setBits(mask: Int) {

lastState = lastState.or(mask)

UsbPort.write(lastState)

}

// coloca o(s) bit(s) indicado(s9 na máscara a off

fun clrBits(mask: Int) {

lastState = lastState.and(mask.inv())

UsbPort.write(lastState)

}

// coloca o valor indicado em value no(s) bit(s) indicado(s) na máscara

fun writeBits(mask: Int, value: Int) {

lastState = value.or(lastState.and(mask.inv()))

UsbPort.write(lastState)

}

}

1. Código *Kotlin* - *KBD*

object KBD {

private const val Kval = 0x01

private const val Kack = 0x01

private const val K = 0x1E

const val waitTime: Long = 2000

private const val sleepTime: Long = 10

private const val CODE1 = 0

private const val CODE2 = 4

private const val CODE3 = 8

private const val CODE4 = 1

private const val CODE5 = 5

private const val CODE6 = 9

private const val CODE7 = 2

private const val CODE8 = 6

private const val CODE9 = 10

private const val CODE0 = 7

private const val CODEEXT = 3

private const val CODEHASH = 11

// inicia o HAL e coloca o bit correspondente a K Ack a 0

fun init() {

HAL.init()

HAL.clrBits(Kack)

}

const val NONE = 0.toChar()

// le o que é escrito no teclado e devolve o que leu, caso não consiga ler nada devolve NONE

private fun getKey(): Char {

if (HAL.isBit(Kval)) {

val c = when (HAL.readBits(K).shr(1)) {

CODE1 -> '1'

CODE2 -> '2'

CODE3 -> '3'

CODE4 -> '4'

CODE5 -> '5'

CODE6 -> '6'

CODE7 -> '7'

CODE8 -> '8'

CODE9 -> '9'

CODEEXT -> '\*'

CODE0 -> '0'

CODEHASH -> '#'

else -> NONE

}

HAL.setBits(Kack)

while (HAL.isBit(Kval)) {

Thread.sleep(sleepTime)

}

HAL.clrBits(Kack)

return c

}

return NONE

}

// chama o getKey() até passar o tempo de timeout ou até ler uma tecla

fun waitKey(timeout: Long): Char {

val timeInit = System.currentTimeMillis()

while (true) {

val time = System.currentTimeMillis()

val c = getKey()

if (c != NONE) return c

if (time - timeInit >= timeout) return NONE

}

}

}