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.

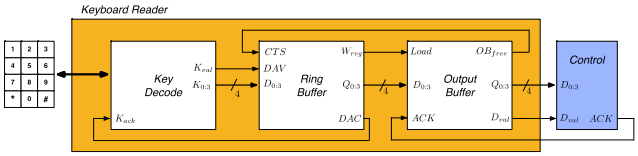
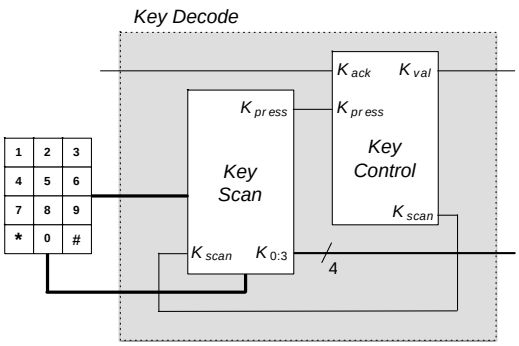


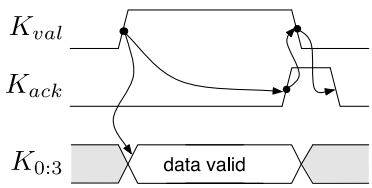
Figura 1 – Diagrama de blocos do módulo *Keyboard Reader*

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 foi baseada em uma arquitetura mais simples, compreensível e, por consequência, mais fácil de implementar, tendo em conta o tempo que nos foi disponibilizado para realizar esta tarefa.

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

A implementação possui três estados sendo eles o estado Scanning, Waiting e Ready. O estado Scanning representa o estado em que ocorre a varredura do teclado e quando é detetado uma tecla premida o Key Control avança para o estado Waiting que representa a espera de confirmação de leitura por parte do software, avançando assim para o estado Ready que espera até que não exista nenhuma tecla premida para avançar para o estado Scanning novamente.

A descrição hardware do bloco Key Decode em VHDL encontra-se no Anexo A.

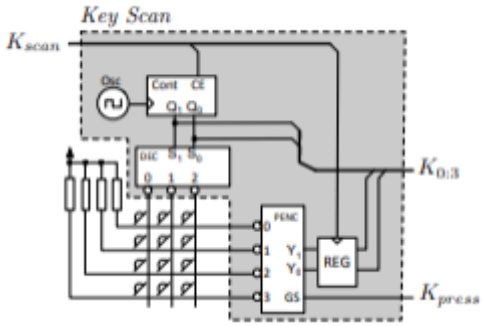


Figura 3 - Diagrama de blocos do bloco *Key Scan*

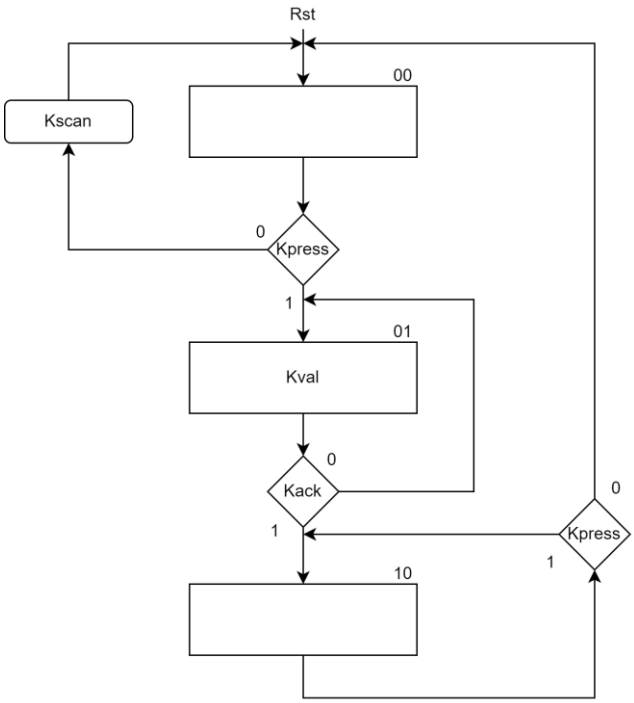


Figura 4 – Máquina de estados do bloco *Key Control*

Com base nas descrições do bloco Key Decode implementou-se parcialmente o módulo Keyboard Reader de acordo com o esquema elétrico representado no Anexo D que possui um clockDiv que gera uma frequência de clock que permite realizar a varredura da matriz e a comunicação com os restantes componentes do hardware.

2 Ring Buffer

O bloco Ring Buffer é uma estrutura de dados capaz de armazenar teclas com disciplina FIFO (First In First Out), com a capacidade de armazenar até oito palavras de 4 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 (Empty). 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 5 é apresentado o diagrama de blocos para uma estrutura do bloco Ring Buffer.

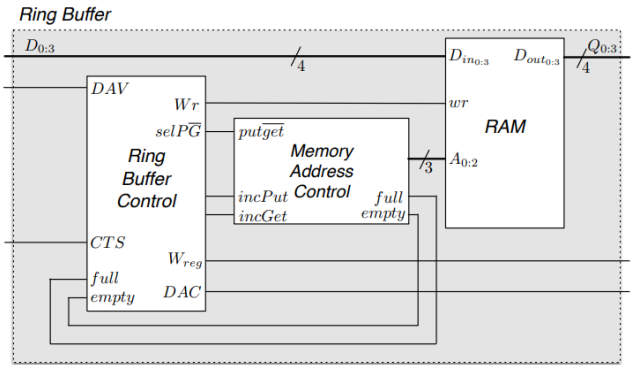


Figura 5 - Diagrama de blocos do bloco Ring Buffer

O bloco

2.1 Ring Buffer Control

Inicialmente o Ring Buffer Control se encontra no estado waiting e quando recebe o sinal DAV a 1 passa para o estado PutnotGet caso o Ring Buffer não esteja full, realizando a escrita e apontando o idx para o próximo endereço, caso não tenha DAV a 1 ou o full esteja a 1 é verificado o sinal CTS que indica o pedido de leitura pelo OutPut Buffer e caso não esteja empty é realizado uma leitura e então o índice é apontado para o próximo endereço, retornando para o estado waiting no final de ambas as execuções. O bloco Ring Buffer Control foi implementado pela máquina de estados representada em ASM-chart na Figura 6.

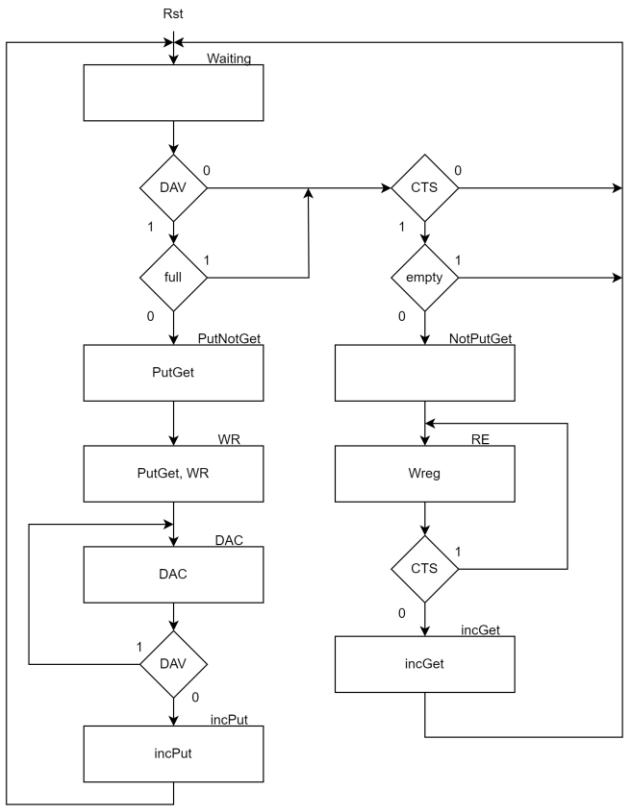


Figura 6 - Máquina de estados do bloco Ring Buffer Control

2.2 Memory Adress Control(MAC)

O Memory Adress Control(MAC) é implementado usando três contadores. Dois desses contadores são usados para armazenar o índice de leitura e o índice de escrita, enquanto o terceiro contador, auxiliado por uma unidade lógica e aritmética (ALU), armazena a diferença entre os valores dos índices de leitura e escrita.

Para determinar se a estrutura de dados está cheia ou vazia, é realizado um operador lógico "and" na saída do contador que armazena a diferença dos índices. Quando esse valor for "000", significa que ambos os índices estão na mesma posição, indicando que o sistema está vazio. Por outro lado, quando a diferença for "111", o índice de escrita deu uma volta completa, o que indica que o sistema se encontra cheio. O bloco Memory Adress Control é implementado conforme o diagrama de blocos representado na figura 7.

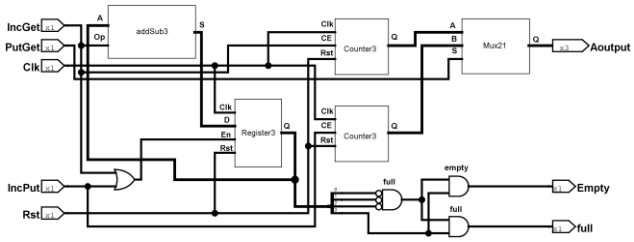


Figura 7 – Diagrama de blocos do Memory Adress Control

3 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 7, é apresentado o diagrama de blocos do Output Buffer.

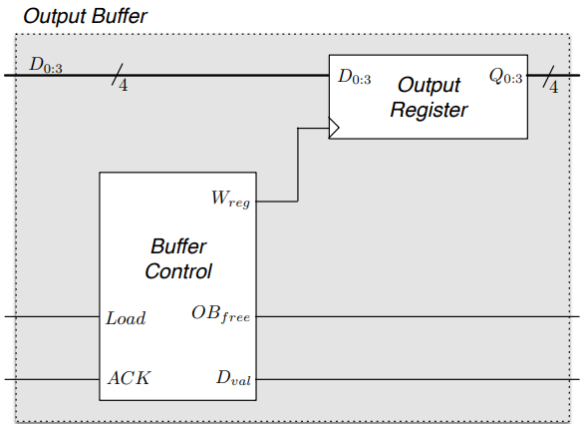


Figura 8 – Diagrama de blocos do Output Buffer

Sempre que o bloco emissor Ring Buffer tenha dados disponíveis e o bloco de entrega Output Buffer esteja disponível (OBfree ativo), o Ring Buffer realiza uma leitura da memória e entrega os dados ao Output Buffer ativando o sinal Wreg. O Output Buffer indica que já registou os dados desativando o sinal OBfree como é representado pela máquina de estados em ASM-chart na Figura 9.

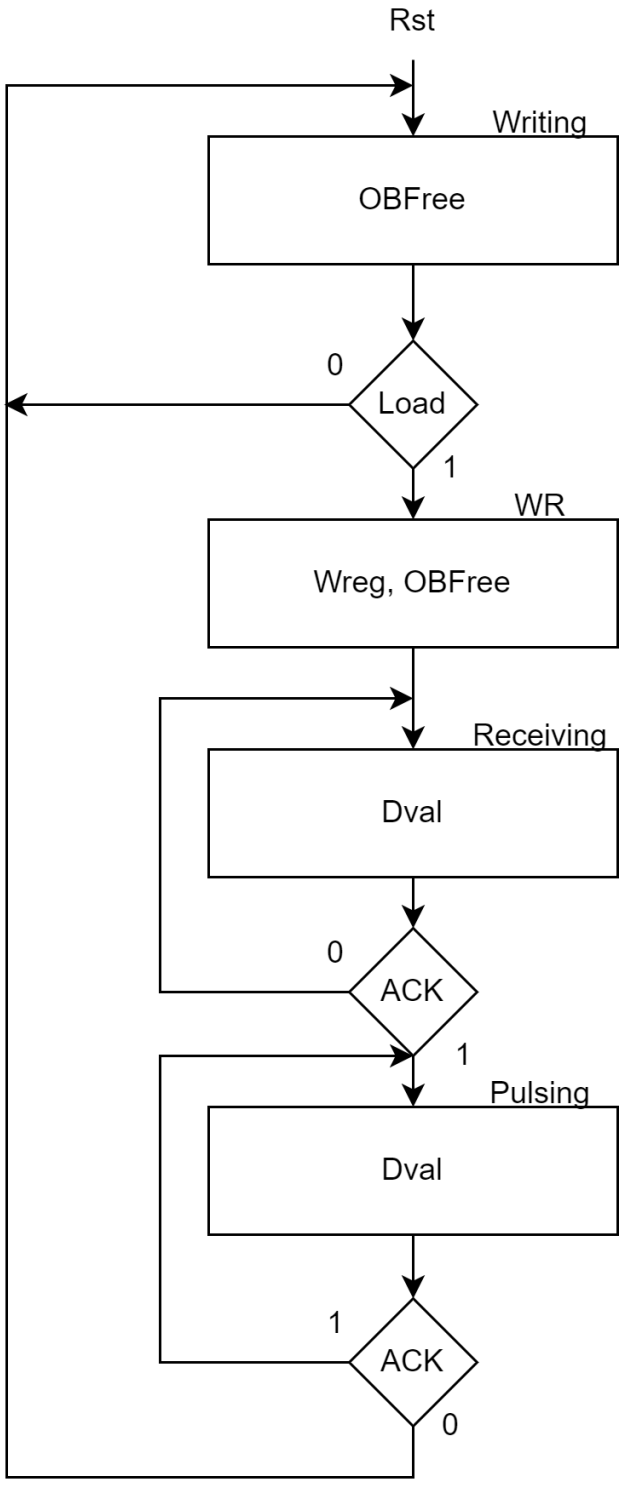


Figura 9 – Máquina de estados do Buffer Control.

4 Interface com o *Control*

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem *Kotlin* e seguindo a arquitetura lógica apresentada na Figura 10.

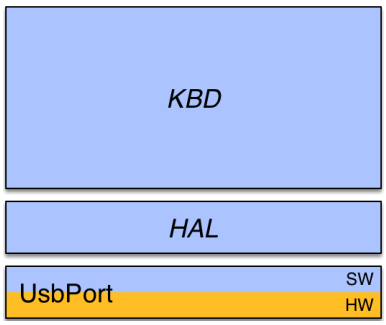


Figura 10 – Diagrama lógico do módulo *Control* de interface com o módulo *Keyboard Reader*

HAL e KBD desenvolvidos são descritos nas secções 2.1. e 2.2, e o código fonte desenvolvido nos Anexos E e F, respetivamente.

4.1 HAL

HAL (Hardware Abstraction Layer) é uma camada de abstração de hardware em um sistema de controle de acesso que permite que o software interaja com o hardware de forma mais simplificada e independente da plataforma. Em outras palavras, o HAL é uma interface entre o software e o hardware que abstrai as características específicas do hardware e as apresenta como uma interface mais genérica para o software. O objeto HAL nos permite a leitura e escrita através do UsbPort. Permitindo assim uma manipulação mais abrangente de dados.

4.2 KBD

O KBD (Keyboard Interface) é responsável por receber as entradas do usuário e convertê-las em códigos que possam ser interpretados pelo software. Sendo assim o KBD é capaz de analisar se trama recebida é valida e caso a mesma seja, também é capaz comparar os dados com uma lista de valores pré-estabelecidos para realizar a interpretação.

4.3 TUI

O TUI tem a função de fazer a interligação entre o KBD (*keyboad* Interface) e o LCD. Este fá-lo, de momento, espelhando os dados recebidos pelo KBD no LCD, sendo responsável pela interpretação dos dados gerados pelo KBD e tratamento das mensagens a serem enviadas para o LCD.
Tendo como as principais funções.

- getKey = retorna a key atual do teclado e caso não seja um valor valido retorna NONE.
- waitKey = retorna uma key lida do keyBoard, caso não seja lida nenhuma tecla dentro do tempo estabelecido, retorna NONE.
- waitInput = realiza a leitura de um UIN ou um PIN do keyboard.
- emptyScreen = realiza um clear do LCD e então tira o cursor do LCD.
- showMsg = realiza a escrita da mensagem passada como parâmetro no LCD.

5 Conclusões

O módulo Keyboard Reader é composto por três blocos principais: Key Decode, Ring Buffer e Output Buffer. O bloco Key Decode implementa um descodificador de um teclado matricial 4x3 por hardware, com sub-blocos de Key Scan e Key Control. O bloco Key Scan realiza o varrimento do teclado, enquanto o bloco Key Control controla o varrimento e o fluxo de teclas pressionadas.
O Ring Buffer é uma estrutura de dados com capacidade FIFO para armazenar até oito palavras de 4 bits. Ele recebe os dados do Key Decode e os armazena, indicando ao produtor que os dados foram aceitos. O Memory Address Control (MAC) controla os endereços de escrita e leitura no Ring Buffer, verificando se está cheio ou vazio.
O Output Buffer interage com o sistema consumidor, no caso, o módulo Control. Ele indica quando está disponível para armazenar dados e o Control lê os dados quando o sinal Dval está ativo. O Output Buffer invalida os dados após o Control consumi-los.
A implementação do módulo Keyboard Reader utiliza recursos de hardware, como um teclado matricial 4x3 e memória RAM para o Ring Buffer. O desempenho e a latência na deteção de tecla dependem da velocidade do clock do sistema, que permite a varredura da matriz e a comunicação com os outros componentes do hardware. O tempo de resposta na deteção de tecla pode ser afetado por fatores como a velocidade do clock, a implementação do Key Decode e o desempenho geral do sistema.
Além disso, foi implementado o módulo Control em software, utilizando a linguagem Kotlin. O Control atua como o consumidor dos dados do Keyboard Reader, interagindo com ele por meio da HAL (Hardware Abstraction Layer) e do KBD (Keyboard Interface). A HAL fornece uma interface abstrata para o hardware, simplificando a interação com ele, enquanto o KBD recebe as entradas do usuário e as converte em códigos interpretáveis pelo software.
O módulo TUI (Terminal User Interface) faz a interligação entre o KBD e o LCD, espelhando os dados recebidos pelo

KBD no LCD, exceto quando é recebido um valor "None" da função getKey do KBD.
Em resumo, a implementação do módulo Keyboard Reader envolveu a combinação de componentes de hardware e software para realizar a deteção e o armazenamento de teclas pressionadas.

A. Descrição VHDL do KeyBoard Reader

```
LIBRARY IEEE;
use IEEE.std_logic_1164.all;
entity KeyBoardReader is
   port(
                    Clk: in std_logic;
                    rst: in std_logic;
                    ack: in std_logic;
                    Ln: in std_logic_vector(3 downto 0); -- line
                    Cl: out std_logic_vector(2 downto 0); -- column
                    Q: out std_logic_vector(3 downto 0);
                    Dval: out std_logic
                    );
end KeyBoardReader;
architecture logic of KeyBoardReader is
component KeyDecoder is
   port(
                     Kact: in std_logic;
      clk: in std_logic;
      rst: in std_logic;
      Kval: out std_logic;
      K: out std_logic_vector(3 downto 0);
      Ln: in std_logic_vector(3 downto 0);
                     Cl: out std_logic_vector(2 downto 0)
   );
end component;
component OutputBuffer is
         port (
                    load : in std_logic;
                    D : in std_logic_vector(3 downto 0);
                    ACK : in std_logic;
                    OBfree : out std_logic;
                    Dval : out std_logic;
                    clk: in std_logic;
                    rst: in std_logic;
                    Q : out std_logic_vector(3 downto 0)
         );
end component;
```

```
component ringBuffer is
        port (
                D: in std_logic_vector(3 downto 0);
    Q: out std_logic_vector(3 downto 0);
                CTS: in std_logic;
    clk: in std_logic;
                rst: in std_logic;
                DAV: in std_logic;
                Wreg: out std_logic;
                DAC: out std_logic
                );
end component;
        component CLKDIV is
                generic(
                        div: natural := 50000000
                );
                port (
                        clk_in: in std_logic;
                        clk_out: out std_logic
                );
        end component;
signal sclk, sCTS, sDAV, sDAC, sOBfree, sWreg: std_logic;
signal sK_KeyDecode, sQ_RingBuffer : std_logic_vector(3 downto 0);
begin
UKeyDecoder: KeyDecoder
   port map(
      Kact => sDAC,
      clk => sclk,
      rst => rst,
      Kval => sDAV,
      K => sK_KeyDecode,
      Ln => Ln,
          Cl => Cl
                );
URingBuffer: ringBuffer
                port map(
                        D => sK_KeyDecode,
                        Q => sQ_RingBuffer,
```

```
                CTS => sOBfree,
                clk => Clk,
                rst => rst,
                DAV => sDAV,
                Wreg => sWreg,
                DAC => sDAC
                );
UOutputBuffer: OutputBuffer
        port map(
                load => sWreg,
                D => sQ_RingBuffer,
                ACK => ack,
                OBfree => sOBfree,
                Dval => Dval,
                clk => Clk,
                rst => rst,
                Q => Q
        );
--Clock Div
        UCLKDIV: CLKDIV
                generic map(500000)
                port map(
                        clk_in => clk,
                        clk_out => sclk
                );
end architecture;
```

B. Descrição VHDL do bloco *Key Decode*

```
LIBRARY IEEE;
use IEEE.std_logic_1164.all;
entity KeyDecoder is
port(
Kact: in std_logic;
clk: in std_logic;
rst: in std_logic;
Kval: out std_logic;
K: out std_logic_vector(3 downto 0);
Ln: in std_logic_vector(3 downto 0);
Cl: out std_logic_vector(2 downto 0)
);
end KeyDecoder;
```

```
architecture logic of KeyDecoder is
component KeyScan is
      port (
            Clk: in std_logic;
Kscan: in std_logic;
Ln: in std_logic_vector(3 downto 0); -- line
rst: in std_logic;
Cl: out std_logic_vector(2 downto 0); -- column
K: out std_logic_vector(3 downto 0);
Kpress: out std_logic
);
  end component;
  component KeyControl is
      port(
kack: in std_logic;
kpress: in std_logic;
kval: out std_logic;
clk: in std_logic;
rst: in std_logic;
kscan: out std_logic
);
end component;
signal sKpress: std_logic;
signal sKscan: std_logic;
signal sclk: std_logic;
begin
--                    KeyScan
UKeyScan: KeyScan
port map (
Clk => clk,
Kscan => sKscan,
Ln => Ln,
rst => rst,
Cl => Cl,
K => K,
Kpress => sKpress
);
--                    KeyControl
UKeyControl: KeyControl
port map (
kack => Kact,
kpress => sKpress,
kval => Kval,
clk => clk,
rst => rst,
kscan => sKscan
);
end architecture;
```

C. Descrição VHDL do bloco Ring Buffer

```
LIBRARY IEEE;
use IEEE.std_logic_1164.all;

entity ringBuffer is
            port (
                        D: in std_logic_vector(3 downto 0);
      Q: out std_logic_vector(3 downto 0);
                        CTS: in std_logic;
      clk: in std_logic;
                        rst: in std_logic;
                        DAV: in std_logic;
                        Wreg: out std_logic;
                        DAC: out std_logic
                        );
end ringBuffer;
architecture logic of ringBuffer is
    component RingBufferControl
        port(
            Dav: In std_logic;
            CTS: In std_logic;
            full: In std_logic;
            clk: in std_logic;
            empty: In std_logic;
            rst: in std_logic;
            incPut: out std_logic;
            incGet: out std_logic;
            Dac: out std_logic;
            Wr: out std_logic;
            selPG: out std_logic;
            Wreg: out std_logic
        );
    end component;
    component memoryAdressControl is
        port (
            putget: in std_logic;
            Clk: in std_logic;
            Rst: in std_logic;
            incPut: in std_logic;
            incGet: in std_logic;
            full: out std_logic;
            empty: out std_logic;
            Aoutput: out std_logic_vector(2 downto 0)
        );
    end component;
    component RAM is
        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;
    signal sincPut, sincGet, sselPG, sWr, sfull, sempty: std_logic;
    signal sAddr: std_logic_vector(2 downto 0);
```

```
    begin
        URingBufferControl: RingBufferControl
            port map(
                Dav => DAV,
                CTS => CTS,
                full => sfull,
                clk => clk,
                empty => sempty,
                rst => rst,
                incPut => sincPut,
                incGet => sincGet,
                Dac => DAC,
                Wr => sWr,
                selPG => sselPG,
                Wreg => Wreg
            );
        UmemoryAdressControl: memoryAdressControl
            port map(
                putget => sselPG,
                Clk => clk,
                Rst => rst,
                incPut => sincPut,
                incGet => sincGet,
                full => sfull,
                empty => sempty,
                Aoutput => sAddr
            );
        Uram: Ram
            port map(
                address => sAddr,
                wr => sWr,
                din => D,
                dout => Q
            );
end architecture;
```

D. Descrição VHDL do bloco Output Buffer

```
LIBRARY IEEE;
use IEEE.std_logic_1164.all;

entity OutputBuffer is
	port (
		load : in std_logic;
		D : in std_logic_vector(3 downto 0);
		ACK : in std_logic;
		OBfree : out std_logic;
		Dval : out std_logic;
		clk: in std_logic;
		rst: in std_logic;
		Q : out std_logic_vector(3 downto 0)
	);
end OutputBuffer;
architecture OutputBuffer_arq of OutputBuffer is
component BufferControl is
	port (
		load : in std_logic;
		ACK : in std_logic;
		OBfree : out std_logic;
		Dval : out std_logic;
		clk: in std_logic;
		Wreg: out std_logic;
		rst: in std_logic
	);
end component;
component Registo4 is
	port (
		D : in std_logic_vector(3 downto 0);
		En : in std_logic;
		Rst : in std_logic;
		Clk : in std_logic;
		Q : out std_logic_vector(3 downto 0)
	);
end component;
signal sWreg : std_logic;
begin
	UBufferControl : BufferControl
	port map (
		load => load,
		ACK => ACK,
		OBfree => OBfree,
		Dval => Dval,
		clk => clk,
		rst => rst,
		Wreg => sWreg
		);

		UOutputRegister: Registo4
		port map(
			D => D,
			En => '1',
			Rst => rst,
			Clk => sWreg,
			Q => Q
			);
end architecture;
```

E. Atribuição de pinos do módulo Keyboard Reader

```
#============================================================
# Altera DE10-Lite board settings
#============================================================
set_global_assignment -name FAMILY "MAX 10 FPGA"
set_global_assignment -name DEVICE 10M50DAF484C6GES
set_global_assignment -name TOP_LEVEL_ENTITY "DE10_Lite"
set_global_assignment -name DEVICE_FILTER_PACKAGE FBGA
set_global_assignment -name SDC_FILE DE10_Lite.sdc
set_global_assignment -name INTERNAL_FLASH_UPDATE_MODE "SINGLE IMAGE WITH ERAM"

#============================================================
# CLOCK
#============================================================
set_location_assignment PIN_P11 -to clk

#============================================================
# SW
#============================================================
set_location_assignment PIN_C10 -to Kact
set_location_assignment PIN_F15 -to rst

#============================================================
# LED
#============================================================
set_location_assignment PIN_A8 -to Q[0]
set_location_assignment PIN_A9 -to Q[1]
set_location_assignment PIN_A10 -to Q[2]
set_location_assignment PIN_B10 -to Q[3]
set_location_assignment PIN_A11 -to Dval
set_location_assignment PIN_B11 -to ack


#============================================================
# GPIO, GPIO connect to GPIO Default
#============================================================
set_location_assignment PIN_W5 -to Ln[0]
set_location_assignment PIN_AA14 -to Ln[1]
set_location_assignment PIN_W12 -to Ln[2]
set_location_assignment PIN_AB12 -to Ln[3]
set_location_assignment PIN_AB11 -to Cl[0]
set_location_assignment PIN_AB10 -to Cl[1]
set_location_assignment PIN_AA9 -to Cl[2]
```

F. Código *Kotlin - HAL*

```kotlin
import isel.leic.UsbPort

object HAL {
    private var currentBit = 0x00
    fun init(){
        UsbPort.write(currentBit)
    }
    fun isBit(mask: Int): Boolean =
        mask and UsbPort.read() != 0
    fun readBits(mask: Int): Int =
        mask and UsbPort.read()
    fun writeBits(mask: Int, value: Int) {
        currentBit = (mask and value) or (currentBit and mask.inv())
        UsbPort.write(currentBit)
    }
    fun setBits(mask: Int) {
        currentBit = mask or currentBit
        UsbPort.write(currentBit)
    }
    fun clrBits(mask: Int) {
        currentBit = mask.inv() and  currentBit
        UsbPort.write(currentBit)
    }
}
```

G. Código *Kotlin - KBD*

```
import isel.leic.utils.*

const val NONE = 0x00
const val D_VAL = 0x10
const val K_ACK = 0x80

private val intToChar =  listOf('1', '4', '7', '*', '2', '5', '8',
'0', '3', '6', '9', '#')

object KBD {
    fun init(){
    }
    fun getKey():Char{
        val isValid = HAL.isBit(D_VAL)
        val key = HAL.readBits(NIBBLE)
        return if (isValid) {
            HAL.setBits(K_ACK)
            while (HAL.isBit(D_VAL));
            HAL.clrBits(K_ACK)
            if (key < intToChar.size)  intToChar[key] else
NONE.toChar()
        }
        else {
            NONE.toChar()
        }
    }
    fun waitKey(timeout: Long): Char{
        var key: Char = NONE.toChar()
        val begTime = Time.getTimeInMillis()+timeout
        while (begTime >= Time.getTimeInMillis()) {
            key = getKey()
            if (key != NONE.toChar())
                break
        }
        return key
    }
}
```

H. Código Kotlin – TUI

```
import java.time.LocalDate
import java.time.LocalTime

const val INPUT_WIDTH = 4
const val USER_HEIGHT = 1
const val UIN_SIZE = 3
const val PIN_SIZE = 4

enum class DisplayMode {
    LEFT,
    CENTER,
    RIGHT
}

const val TIME_OUT = 5000

private const val DEL_CHAR = '?'

object TUI {

    private const val COLS = LCD.COLS
    fun init() {}
    private fun clearCursor() = LCD.cursor(USER_HEIGHT, COLS+1)
    private fun showTime() {
        val date = LocalDate.now().toString()
        val time = LocalTime.now()
        displayWrite(date, DisplayMode.LEFT)
        displayWrite("$time", DisplayMode.LEFT, column =
date.length+1)
    }
    fun getKey() = KBD.getKey()
    fun waitKey() = KBD.waitKey((TIME_OUT/2).toLong())
    private fun getKeys(length: Int, time: Boolean): String {
        var word = ""
        while (word.length < length){
            val key = KBD.waitKey(TIME_OUT.toLong())
            if (key != NONE.toChar()){
                if (key == '*' && word.isNotEmpty()) return
key.toString()
                if (key == '*' && word.isEmpty()) return
NONE.toString()
                if (key != '#') word += key
                if (time) showTime()
                var showWord = ""
                if (length == UIN_SIZE)
                    showWord = word
                else
                    word.forEach { _ ->
```

```
                        showWord += '*'
                    }
                displayWrite(showWord, DisplayMode.LEFT, USER_HEIGHT,
INPUT_WIDTH)
            }
            else
                return NONE.toString()
        }
        return word
    }
    fun waitInput(type: Int, text: String? = null): String {
        clear()
        if (text == null) showTime() else displayWrite(text,
DisplayMode.CENTER)
        val msg = if (type == PIN_SIZE) "PIN:" else "UID:"
        displayWrite(msg, DisplayMode.LEFT, USER_HEIGHT)
        fill(USER_HEIGHT, INPUT_WIDTH, DEL_CHAR, type)
        while (true){
            when (val keys = getKeys(type, text == null)) {
                "*" -> {
                    fill(USER_HEIGHT, INPUT_WIDTH, DEL_CHAR, type)
                }
                NONE.toString() -> return "NONE"
                else -> return keys
            }
        }
    }
    private fun clear() = LCD.clear()
    fun emptyScreen(){
        clear()
        clearCursor()
    }
    private fun fill(line: Int, column: Int, char: Char, n: Int) {
        LCD.cursor(line, column)
        LCD.write(char.toString().repeat(n))
        LCD.cursor(line, column)
    }
    fun showMsg(msg: String, split: Boolean = true){
        clear()
        if (split) {
            val list = msg.split(" ")
            if (list.size == 2) {
                displayWrite(list.first(), DisplayMode.CENTER)
                displayWrite(list.last(), DisplayMode.CENTER,
USER_HEIGHT)
            } else {
                val mid = list.size / 2
                var up = ""
                for (c in 0..mid)
                    up += "${list[c]} "
```

```
                var down = ""
                for (c in mid + 1..list.lastIndex)
                    down += "${list[c]} "
                if (up == down)
                    displayWrite(up, DisplayMode.CENTER)
                else {
                    displayWrite(up, DisplayMode.CENTER)
                    displayWrite(down, DisplayMode.CENTER,
USER_HEIGHT)
                }
            }
        }else
            displayWrite(msg, DisplayMode.CENTER, USER_HEIGHT)
        clearCursor()
    }
    private fun displayWrite(
        data: String, displayMode: DisplayMode,
        line: Int? = null, column: Int? = null
    ) =
        when (displayMode) {
            DisplayMode.LEFT -> {
                val cl = column ?: 0
                val ln = line   ?: 0
                LCD.cursor(ln, cl)
                LCD.write(data)
            }
            DisplayMode.CENTER -> {
                val cl = column ?: 0
                val ln = line   ?: 0
                LCD.cursor(ln, (COLS-data.length)/2+cl)
                LCD.write(data)
            }
            DisplayMode.RIGHT -> {
                val cl = column ?: 0
                val ln = line   ?: 0
                LCD.cursor(ln, COLS-data.length-cl)
                LCD.write(data)
            }
        }
}
```