O módulo *Keyboard Reader* é constituído por três blocos principais:

1. 0 descodificador de teclado (*Key Decode*);
2. o bloco de armazenamento (designado por *Ring Buffer*);
3. o bloco de entrega ao consumidor (designado por *Output Buffer*). Neste caso o módulo *Control*, implementado em *software*, é a entidade consumidora.

O relatório presente apenas se refere ao módulo Key Decode e a componente de Key control, parate do diagrama de blocos que se encontra plasmado na informação que se segue.

****

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:

1. um teclado matricial virtualmente de 4x3, uma vez que está implementado em 4x4 no hardware, e cujo software garante que seja apenas avaliadas as teclas conforme solicitado(4x3);
2. o bloco *Key Scan,* responsável pelo varrimento do teclado;
3. o bloco *Key Control*, que realiza o controlo do varrimento e o controlo de fluxo, conforme o diagrama de blocos representado na Figura . O controlo de fluxo de saída do bloco *Key Decode* (para o módulo *Key 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 .



Figura 2 – Diagrama de blocos Key Decode



Figura 3 – Diagrama temporal

O bloco *Key Scan*, por agora implementado, integra-se no processo de evolução de implementação de código VHDL, por ser o que nos pareceu mais simples e, que se apresenta no diagrama de blocos na Figura 4.

A diagram of a key scan

Description automatically generatedFigura 4 - Diagrama de blocosdo bloco *Key Scan*

O bloco *Key Control* foi implementado pela máquina de estados representada em *ASM-chart* na 5. O processo de varrimento é constante para sempre que seja premida uma tecla (Kpress) esta seja registada caso ou exista Kval, esta será processada, através do módulo Key scan, lendo a linha e a coluna da matriz é transmitido o valor binário sendo transformada em informação será processada e avaliada de acordo com os módulos seguintes e de acordo com o que descodificações possibilitam visualização ou ações decorrentes de processos previstos e projetados.

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

A diagram of a block diagram

Description automatically generated

Figura 5 – Máquina de estados do bloco *Key Control*

Com base nas descrições do bloco *Key Decode* implementou-se parcialmente o módulo *Keyboard Reader*.

# Interface com o *Control*

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem Java e seguindo a arquitetura lógica apresentada na figura 6.



Figura 6 – Diagrama lógico do módulo *Control* de interface com o módulo *Keyboard Reader*

As classes *HAL* e *KBD* desenvolvidas são descritas em secções 2.1. e 2.2, e o código fonte desenvolvido em C e D no anexo, respetivamente.

## Classe *HAL*

Sendo a interface entre hardware e software virtualiza o acesso ao sistema UsbPort, que no caso presente garante que ocorram leituras e escritas corretas entre os diversos módulos que compões uma solução.

## Classe *KBD*

A classe KBD garante que após uma tecla ser premida, que pressupõe que o deixe de ser, será descodificada por esta classe, de binário para valores decimais ou outros símbolos, e apresentada para ser apresentadas, apenas durante determinado período na ordem dos milésimos de segundo. Garante ainda que durante o processo de leitura de uma tecla não sejam processadas outas, que inadvertida ou propositadamente sejam ativadas, nesta fase, em que não estão implementados buffers.

# Conclusões

Nesta fase do projeto é possível garantir a leitura correta de teclas da matriz 4x3 e visualizar no LCD os seus valores ou ativar ações, nomeadamente através do código implementao em TUI.

2. Descrição VHDL do bloco *Key Decode*

**use ieee.std\_logic\_1164.all;**

**entity keyDecode is**

**port(**

**rst: in std\_logic;**

**clk: in std\_logic;**

**kack: in std\_logic;**

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

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

**kval: out std\_logic;**

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

**);**

**end keyDecode;**

**architecture structure of keyDecode is**

**component keyScan is**

**port(**

**Kscan: in std\_logic;**

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

**clk: in std\_logic;**

**Rst: in std\_logic;**

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

**Kpress: out std\_logic;**

**O: out std\_logic\_vector(3 downto 0)**

**);**

**end component;**

**component keyControl is**

**port(**

**clk: in std\_logic;**

**rst: in std\_logic;**

**kpress: in std\_logic;**

**kack: in std\_logic;**

**kscan: out std\_logic;**

**kval: out std\_logic**

**);**

**end component;**

**signal kp: std\_logic;**

**signal ks: std\_logic;**

**begin**

**kscan: keyScan port map(**

**Kscan => ks,**

**I => I,**

**clk => clk,**

**Rst => rst,**

**K => K,**

**Kpress => kp,**

**O => O**

**);**

**kctrl: keyControl port map(**

**clk => clk,**

**rst => rst,**

**kpress => kp,**

**kack => kack,**

**kscan => ks,**

**kval => kval**

**);**

**end structure;**

1. Atribuição de pinos do módulo Keyboard Reader

**set\_global\_assignment -name BOARD "MAX 10 DE10 - Lite"**

**set\_global\_assignment -name DEVICE 10M50DAF484C6GES**

**set\_global\_assignment -name FAMILY "MAX 10"**

**set\_location\_assignment PIN\_C10 -to rst**

**set\_location\_assignment PIN\_C11 -to kack**

**set\_location\_assignment PIN\_P11 -to clk**

**set\_location\_assignment PIN\_A10 -to kval**

**set\_location\_assignment PIN\_E14 -to K[0]**

**set\_location\_assignment PIN\_D14 -to K[1]**

**set\_location\_assignment PIN\_A11 -to K[2]**

**set\_location\_assignment PIN\_B11 -to K[3]**

**set\_location\_assignment PIN\_W5 -to I[0]**

**set\_location\_assignment PIN\_AA14 -to I[1]**

**set\_location\_assignment PIN\_W12 -to I[2]**

**set\_location\_assignment PIN\_AB12 -to I[3]**

**set\_location\_assignment PIN\_AB11 -to O[0]**

**set\_location\_assignment PIN\_AB10 -to O[1]**

**set\_location\_assignment PIN\_AA9 -to O[2]**

**set\_location\_assignment PIN\_AA8 -to O[3]**

1. Código *Kotlin* - *HAL*

**import isel.leic.UsbPort**

**object HAL { // Virtualiza o acesso ao sistema UsbPort**

**var prev\_state: Int = 0**

**// Inicia a classe**

**fun init() {**

**prev\_state = 0**

**UsbPort.write(prev\_state)**

**}**

**// Retorna true se o bit tiver o valor lógico ‘1’**

**fun isBit(mask: Int): Boolean {**

**var value = UsbPort.read()**

**value = value and mask**

**if ( value == mask) {**

**return true**

**}**

**return false**

**}**

**// Retorna os valores dos bits representados por mask presentes no UsbPort**

**fun readBits(mask: Int): Int {**

**var value = UsbPort.read()**

**value = value and mask**

**return value**

**}**

**// Escreve nos bits representados por mask os valores dos bits correspondentes em value**

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

**prev\_state = (prev\_state and mask.inv()) or (mask and value)**

**UsbPort.write(prev\_state)**

**}**

**// Coloca os bits representados por mask no valor lógico ‘1’**

**fun setBits(mask: Int) {**

**prev\_state = prev\_state or mask**

**UsbPort.write(prev\_state)**

**}**

**// Coloca os bits representados por mask no valor lógico ‘0’**

**fun clrBits(mask: Int) {**

**prev\_state = prev\_state and (mask.inv())**

**UsbPort.write(prev\_state)**

**}**

**}**

**fun main() {**

**HAL.init()**

**while(true){**

**HAL.setBits(0xCC)**

**Thread.sleep(2000)**

**HAL.writeBits(0x66, 0x33)**

**Thread.sleep(2000)**

**HAL.setBits(0x3C)**

**Thread.sleep(2000)**

**HAL.clrBits(0x99)**

**Thread.sleep(2000)**

**}**

**}**

1. Código *Kotlin* - *KBD*

**import java.time.LocalDateTime**

**import java.time.LocalTime**

**object KBD { // Ler teclas. Métodos retornam ‘0’..’9’,’#’,’\*’ ou NONE.**

**private const val NONE = 0;**

**private const val keyboardMask = 0x0F**

**private const val kvalMask = 0x10**

**private const val kackMask = 0x10**

**private var count = 0**

**// Inicia a classe**

**fun init() {**

**HAL.clrBits(kackMask)**

**}**

**// Retorna de imediato a tecla premida ou NONE se não há tecla premida.**

**fun getKey(): Char {**

**if(HAL.isBit(kvalMask)){**

**val key = HAL.readBits(keyboardMask)**

**val c = when(key){**

**0x00 -> '1'**

**0x01 -> '4'**

**0x02 -> '7'**

**0x03 -> '\*'**

**0x04 -> '2'**

**0x05 -> '5'**

**0x06 -> '8'**

**0x07 -> '0'**

**0x08 -> '3'**

**0x09 -> '6'**

**0x0A -> '9'**

**0x0B -> '#'**

**else -> NONE.toChar()**

**}**

**if (c != NONE.toChar()){**

**if(HAL.isBit(kvalMask)){**

**HAL.setBits(kackMask)**

**val prevTime = LocalTime.now()**

**while(HAL.isBit(kvalMask));**

**val postTime = LocalTime.now()**

**HAL.clrBits(kackMask)**

**}**

**return c**

**}**

**}**

**return NONE.toChar()**

**}**

**// Retorna a tecla premida, caso ocorra antes do ‘timeout’ (representado em milissegundos), ou**

**//NONE caso contrário.**

**fun waitKey(timeout: Long): Char{**

**val prevTime = LocalDateTime.now()**

**while (true){**

**val currTime = LocalDateTime.now()**

**if((currTime - prevTime) == timeout){**

**return**

**}**

**}**

**return '0'**

**}**

**}**

**fun main(){**

**HAL.init()**

**KBD.init()**

**while (true){**

**Thread.sleep(10000)**

**println(KBD.getKey())**

**}**

**}** **library ieee;**

**use ieee.std\_logic\_1164.all;**

**entity keyboardReader is**

**port(**

**rst: in std\_logic;**

**clk: in std\_logic;**

**kack: in std\_logic;**

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

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

**dval: out std\_logic;**

**D: out std\_logic\_vector(3 downto 0)**

**);**

**end keyboardReader;**

**architecture structure of keyboardReader is**

**component keyDecode is**

**port(**

**rst: in std\_logic;**

**clk: in std\_logic;**

**kack: in std\_logic;**

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

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

**kval: out std\_logic;**

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

**);**

**end component;**

**begin**

**kdecode: keyDecode port map(**

**rst => rst,**

**clk => clk,**

**kack => kack,**

**I => I,**

**O => O,**

**kval => dval,**

**K => D**

**);**

**end structure;**