Autores: João Nunes (A47220@alunos.isel.pt)
Alexandre Luís (A47222@alunos.isel.pt)
Miguel Marques (A47204@alunos.isel.pt)

O módulo *Keyboard Reader* implementado é constituído por dois blocos principais: *i*) o descodificador de teclado (*Key Decode*); e *ii*) o bloco de armazenamento e de entrega ao consumidor (designado por *Key Buffer*), conforme ilustrado na Figura 1. Neste caso o módulo de controlo, implementado em *software*, é a entidade consumidora.

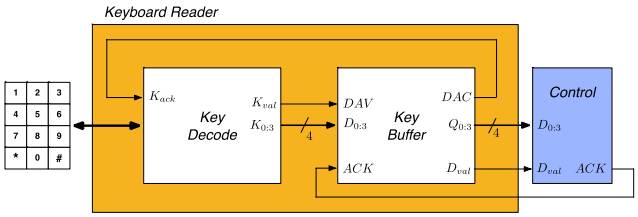
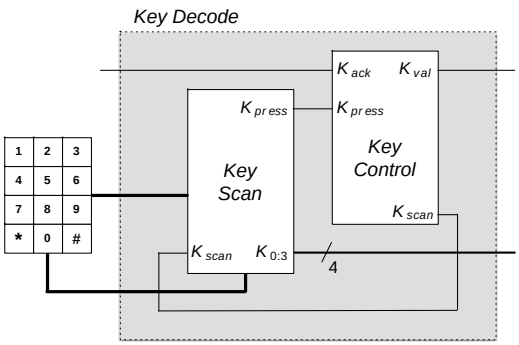


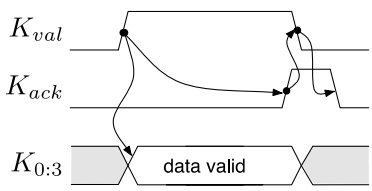
Figura 1 – Diagrama de blocos do módulo *Keyboard Reader*

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.

Após analisar as três versões disponíveis, a preferida foi a versão III, por necessitar de menos *clocks* para o varrimento do teclado matricial, usando só um contador para o fazer, recebendo imediatamente o sinal da tecla premida pelo *priority Encoder*, mesmo tendo em consideração que usa dois bits de entrada em *Kscan*, um para count enable e outro como *clock* para o *register*.

O bloco *Key Control* foi implementado pela máquina de estados representada em *ASM-chart* na Figura 4. Foi contruida esta solução tendo recurso a dois *flip-flops*, sendo usados 3 estados, no primeiro verifica-se se é premida alguma tecla do *keyboard*, mantendo *Kscan0* ligado para manter o circuito a varrer o mesmo. No segundo ativa *Kscan1* para registar o codigo da tecla premida, no terceiro ativa *Kval*, enquando este não receber *Kack* e a tecla premida não for largada, este não continua o varrimento.

A descrição hardware do bloco *Key Decode* em CUPL/VHDL encontra-se no Anexo 0 e o código fonte no anexo C.

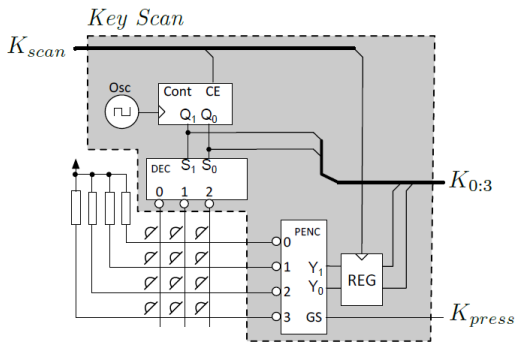


Figura 3 - Diagrama de blocos do bloco *Key Scan*

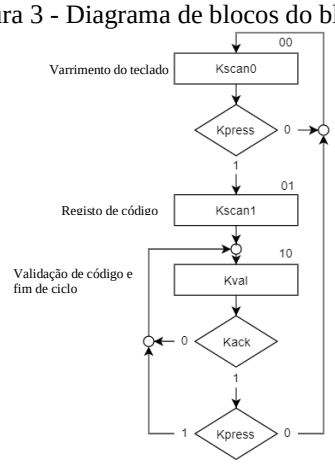


Figura 4 – Máquina de estados do bloco *Key Control*

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.

Autores: João Nunes (A47220@alunos.isel.pt)
Alexandre Luís (A47222@alunos.isel.pt)
Miguel Marques (A47204@alunos.isel.pt)

2 Key Buffer

O módulo *Key Buffer* implementa uma estrutura de armazenamento de dados, com capacidade de uma palavra de quatro bits. A escrita de dados no *Key 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 *Key Buffer* escreve os dados $D_{0:3}$ 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 *Key Buffer* só desativa *DAC* depois de *DAV* ter sido desativado.

A implementação do *key Buffer* deverá ser baseada numa máquina de controlo (*Key Buffer Control*) e num registo externo (*Output Register*), conforme o diagrama de blocos apresentado na Figura 5.

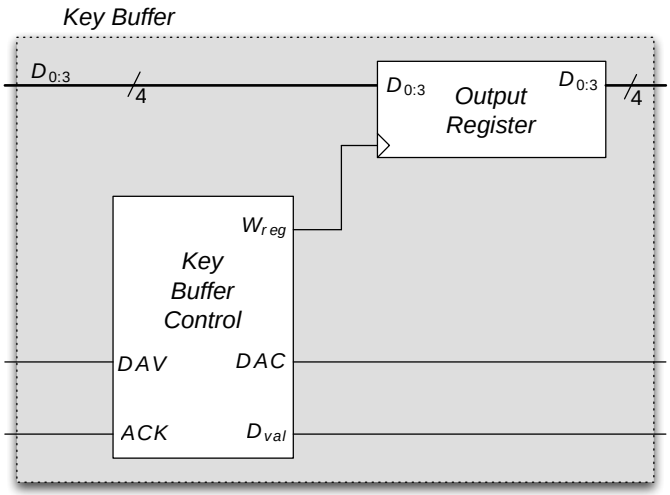


Figura 5 – Diagrama de blocos do *Key Buffer*

O bloco *Key Buffer Control* do *Key Buffer* é também responsável pela interação com o sistema consumidor, neste caso o módulo *Control*. O *Control* quando pretende ler dados do *Key Buffer*, aguarda que o sinal D_{val} fique ativo, recolhe os dados e ativa o sinal *ACK* indicando que estes já foram consumidos.

O *Key Buffer Control*, logo que o sinal *ACK* fique ativo, deve invalidar os dados baixando o sinal D_{val}, só deverá voltar a armazenar uma nova palavra depois do *Control* ter desativado o sinal *ACK*.

O bloco *Key Buffer Control* foi implementado de acordo com a máquina de estados representado na Figura 6. A solução foi implementada com dois 2 *flip-flops*, haverá soluções mais compactas, mas tendo 4 estados é mais facil de garantir o funcionamento do circuito, por exemplo, tendo em conta que o *key buffer* comunica com o control fora do *KeyboardReader*, é necessário ter estados que comuniquem com varios modulos, como será explicado no ponto B.

A descrição hardware do bloco *Key Buffer Control* em CUPL/VHDL encontra-se no Anexo e o codigo fonte no C.

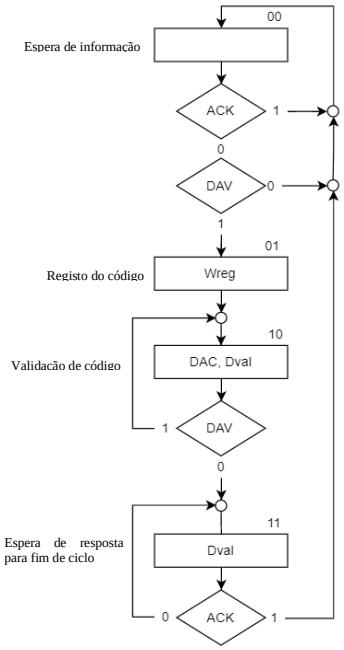


Figura 6 - Máquina de estados do bloco *Key Buffer Control*

Com base nas descrições do bloco *Key Decode* e do bloco *Key Buffer Control* implementou-se o módulo *Keyboard Reader* de acordo com o esquema elétrico representado no Anexo D.

3 Interface com o *Control*

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem Java e seguindo a arquitetura lógica apresentada na Figura 7.

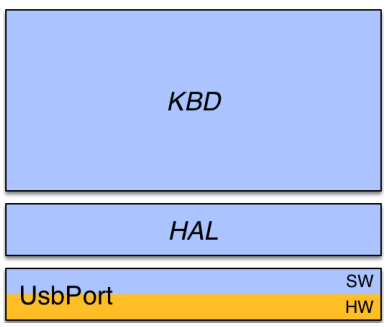


Figura 7 – Diagrama lógico do módulo *Control* de interface com o módulo *Keyboard Reader*

As classes *HAL* e *KBD* desenvolvidas são descritas nas secções 3.1. e 3.2, e o código fonte desenvolvido nos Anexos E e F, respetivamente.

Autores: João Nunes (A47220@alunos.isel.pt)
Alexandre Luís (A47222@alunos.isel.pt)
Miguel Marques (A47204@alunos.isel.pt)

3.1 Classe *HAL*

Esta classe, resumidamente, traduz a entrada(*UsbPort.in()*) do UsbPort para fácil manuseamento dentro das outras classes desenvolvidas em *software*. Faz também a conversão devida para a saída (*UsbPort.out()*).
É composta por métodos como *readBits()* e *isBit()* que leem a entrada do UsbPort e retornam, respetivamente, o valor dos bits a 1 e se o bit estiver a 1 ou a 0.
Pertencem também os métodos: *writeBits()*, *setBits()* e *clrBits()*; que servem para manipulação dos bits de saída do *UsbPort*. Estes métodos estão devidamente comentados na secção onde o código-fonte desta classe se encontra.

Por fim, *updateOutput()* atualiza UsbPort.out() para o valor atual da saída do dispositivo a ser utilizado (*UsbPort* ou *Uclix*) e *getInput()* atualiza a variável *input* com a entrada do dispositivo a ser utilizado (*UsbPort* ou *Uclix*).

3.2 Classe *KBD*

Esta classe processa e realiza a leitura do teclado. Tem dois métodos principais: *getKey()* e *waitKey()*.
getKey() lê a tecla premida ou o último valor guardado em *KeyScan*.
waitKey() espera por uma tecla dado um determindo tempo (*timeout)*
Ambos retornam o caracter correspondente ou *NONE* (0) caso nenhuma tecla tenha sido premida.
No método *getKey()* é realizado um *clock* à variável *ACK* que faz parte do *Control* do módulo *Keyboard Reader*. Isto serve para informar o módulo *Key Buffer* que o valor da tecla foi registado com sucesso.
A única alteração feita nesta classe desde a primeira versão foi a mudança da utilização de *Time.sleep()* para *Time.getTimeInMillis()*: deste modo conseguimos obter o valor real do tempo que passa, pois *Time.getTimeInMillis()* retorna valor do relógio do computador atual (em milissegundos), sendo que no fim fazemos a subtração do tempo atual com o tempo de início.

4 Conclusões

Em *software:* na primeira avaliação realizámos a classe HAL de forma incorreta. Não estávamos a ler a entrada do *UsbPort* e só usávamos a saída para fazer os comandos necessários para as outras classes. Nesta versão atualizada conseguimos corrigir vários *bugs* de longa data.

Em *hardware*: a utilização do 3º diagrama do *KeyScan* gerou problemas em relação ao sinal *Kscan*. Inicialmente pensávamos que era o mesmo, mas na verdade são dois sinais diferentes, originando *Kscan0* e *Kscan1*. Outro aspeto a ter em conta são os pulsos dos diferentes *clocks* em cada máquina de estados e módulo. Como exemplo temos o *clock* da máquina de estados do *Key Control* e o *clock* do contador que realiza o scan ao teclado. Estes *clocks* têm que ter tempos de pulso diferentes. Neste caso, o *clock* da máquina de estados deve ser maior que o que lê o teclado para diminuir a probabilidade de falhar a leitura de uma tecla. Após a validação de laboratório foi concluído que em *clock speeds* muito altas, a informação do *Keyboard* começou a sofrer de um problema de *hardware* chamado *bounce*. Isto consiste na falsa recolha de informações causadas pela repetição repentina na mudança do estado do *clock*. Para resolver este problema, foi diminuído o *clock* da máquina de estados para 1k*Hz* e o *clock* do *Key Scan* (o que faz a procura para uma tecla premida) para 10 *Hz*.

Autores: João Nunes (A47220@alunos.isel.pt)
Alexandre Luís (A47222@alunos.isel.pt)
Miguel Marques (A47204@alunos.isel.pt)

A. Descrição CUPL/VHDL do bloco *Key Decode*

Tendo em conta o diagrama de blocos da figura 3 e o seu proprio diagrama de blocos, o bloco *Key Decode* é composto pelos seguintes modulos:
- *KeyScan*:
    - *Counter*;
    - *Decoder*;
    - *PriorityEncoder*;
    - *Register*;
- Máquina de estados *KeyControl*.

- **KeyScan:**
- Counter:

Contador de dois bits, usando dois *flip-flops*, Q0 e Q1, usa a variável *Clk*, presente no pin 2 da pal, como *clock*, independente dos *clocks* dos modulos adjacentes, tem um *count enable*, na forma de um dos bits de *Kscan*, *Kscan0*, que é permanece ativo, segundo o Key control, sempre que se quiser continuar a fazer varrimentos no teclado.

- Decoder:

Decoder de dois bits de seleção e 4 saidas, das quais, só 3 é que são usadas para o varrimento do teclado matricial.
Nos dois bits de seleção estão as saidas do contador, para poder indicar qual a coluna do teclado é que se está a verificar, selecionando uma das saidas deste, *Dec0*, *Dec1* ou *Dec2*.
As saídas do *decoder* saiem negadas, e portanto a 0, para, ao varrer uma coluna, se uma tecla for premida, deixa passar a ligação ao *vcc* para o modulo seguinte, o priority encoder.

- PriorityEncoder:

Encoder com 4 entradas(Enc0..3) e consequentemente 2 saídas(Y0..1), tendo a entrada ativa sempre prioridade sobre as que são de menor peso que esta.
Tem um saida adicional, *GS*, que é ativo sempre que uma das entradas for ativada.
Este recebe o sinal da linha do teclado onde uma tecla foi premida e forma um codigo de dois bits, as saidas Y0 e Y1, e transfere-o para um *register* para ser armazenado.

- Register:

Register de 2 bits, usando 2 *flip-flops*, R0..1, as suas entradas são as saidas do *priority encoder*, e usa o segundo bit de *Kscan*, *Kscan1*, como o seu *clock*, e assim poder guardar os bits de Y0..1.

- **Key Control:**

Como ja foi referido, esta máquina de estados usa dois *flip-flops*(K0 e K1), proporciona três estados para controlar o módulo *Key decode*, usando como *clock* o *Mclk* do pin 1da PAL, no primeiro, permanece em modo de varrimento, mantendo *Ksan0*/*CE* ativo, enquando não houver uma tecla premida, sendo esta informação retirada de *Kpress* que corresponde à saida *GS* do *Pencoder*.
No segundo Kscan1 é ativo para registar o código da tecla premida.
No terceiro, ativa *Kval* para indicar de que tem uma tecla valida, e mantém-se nesse estado enquanto o modulo *Key buffer* não informar de que a tecla foi recebida, por *Kack*(sinal *DAC* vindo do *Key buffer*), e enquanto a tecla premida não for largada.
Desta forma, com esta solução é possivel verificar-se as condições do diagrama temporal da figura 2 b).

Autores: João Nunes (A47220@alunos.isel.pt)
Alexandre Luís (A47222@alunos.isel.pt)
Miguel Marques (A47204@alunos.isel.pt)

B. Descrição CUPL/VHDL do bloco *Key Buffer*

Tendo em conta o diagrama de blocos da figura 5, o bloco *Key Buffer* é composto pelos seguintes módulos:
- *KeyBufferControl*;
- *Output Register*.

- **KeyBufferControl:**

Nesta máquina de estados usa-se 2 *flip-flops*(X0 e X1), obtendo-se 4 estados para o controle do *key buffer*.
Semelhantemente a máquina de estados anterior, esta tambem usa *Mclk* do pin 1 da PAL como *clock*.
Mantem-se no primeiro estado enquanto o sinal *ACK* dum varrimento anterior não se desligar e também até receber sinal de que há data para poder armazenar, *DAV*(*Kval do key decode*), no estado dois, ativa *Wreg* para armazenar o código da tecla no *output register*. No estado seguinte ativa *DAC* para informar o key decode de que o código foi aceite e tambem ativa *Dval* para informar o control externo de que tem informação para ser enviada, e mantem-se neste estado até que o sinal *DAV* e portanto *Kval*, seja desativado.
No último estado, mantem ainda *Dval* ativo, pois ainda está em comunicação com o control externo, e só o desliga se receber do control que o código foi aceite, pelo sinal *ACK*.
Voltando ao inicio do ciclo onde é mantido no primeiro estado até o sinal *ACK* do *control* se desativar como dito em cima.

- **Output Register:**

Registo de 4 bits, usando 4 *flip-flops*(O0..3), tendo como *clock*, o *Wreg* enviado pela máquina de estados anterior.
Sendo as suas entradas, na metade, o contador do *key decode*, e na outra o registo do codigo do *priority encoder* tambem do *key decode*.

Autores: João Nunes (A47220@alunos.isel.pt)
Alexandre Luís (A47222@alunos.isel.pt)
Miguel Marques (A47204@alunos.isel.pt)

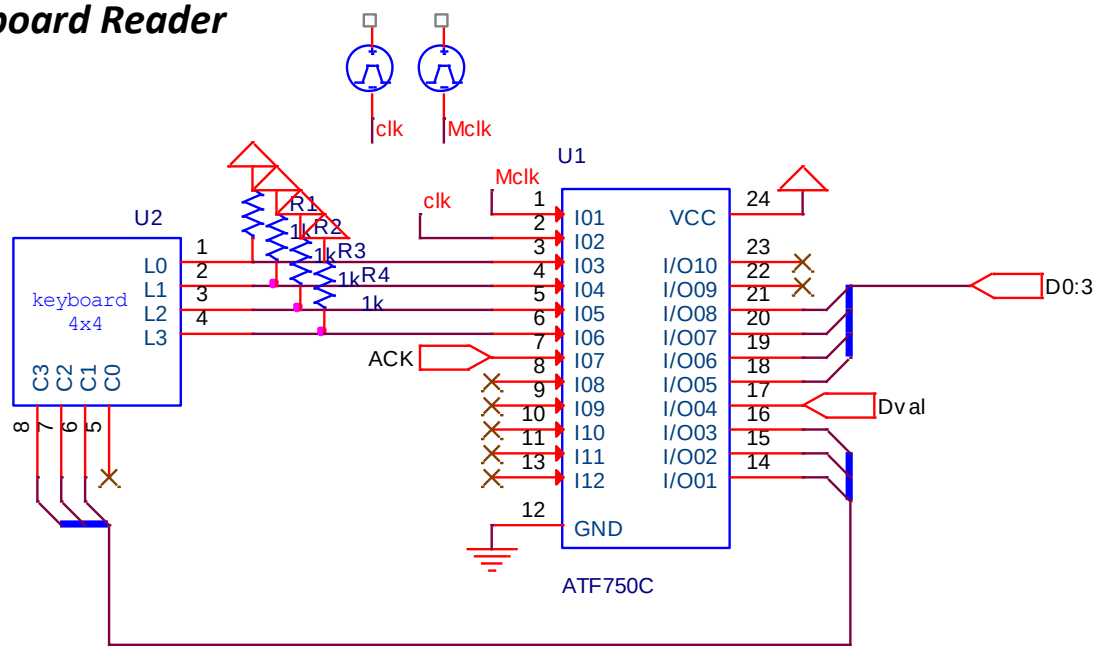
C. Código CUPL do módulo *KeyBoardReader*

```
/* *************** OUTPUT PINS *********************/
PIN 14 = !Dec0 ;
PIN 15 = !Dec1 ;
PIN 16 = !Dec2 ;
PIN 17 = Dval ;
PIN 18 = O0 ;   /*      Counter */
PIN 19 = O1 ;
PIN 20 = O2 ;   /*      Register */
PIN 21 = O3 ;
PIN 22 = X1 ;
PIN 23 = X0 ;


/* ************* PINNODES ************ */
PINNODE 27 = K0 ;
PINNODE 28 = K1 ;
PINNODE 29 = Q1 ;
PINNODE 30 = Q0 ;
PINNODE 31 = R1 ;
PINNODE 32 = R0 ;

/* ************** BODY ************** */

/* ************** COUNTER ********** */
CE = Kscan0 ;

[Q0..1].ck = Clk ;
[Q0..1].sp = 'b'0 ;
[Q0..1].ar = 'b'0 ;

Q0.T = CE ;
Q1.T = CE & Q0 ;


/* ************ DECODER ******** */
DecS0 = Q0 ; /* Para efeitos de teste */
DecS1 = Q1 ;

Dec0 = !DecS0 & !DecS1 ;

Dec1 = DecS0 & !DecS1 ;

Dec2 = !DecS0 & DecS1 ;


/* *********** PRIORITY ENCODER ******** */
GS = !Enc0 # !Enc1 # !Enc2 # !Enc3 ;

Y0 = Enc2 & !Enc1 # !Enc3 ;

Y1 = !Enc2 # !Enc3 ;


/* ********* REGISTER ********** */
[R0..1].d = [Y0..1] ;
[R0..1].sp = 'b'0 ;
[R0..1].ar = 'b'0 ;
[R0..1].ck = Kscan1 ;


/* ************* KEY CONTROL ******************* */
Kpress = GS ;
Kack = DAC ;

[K0..1].ar = 'b'0 ;
[K0..1].sp = 'b'0 ;
[K0..1].ckmux = Mclk ;

SEQUENCE [K1, K0] {
        PRESENT 0
                OUT Kscan0 ;
                IF Kpress NEXT 1 ;
                DEFAULT NEXT 0 ;
        PRESENT 1
                OUT Kscan1 ;
                DEFAULT NEXT 2 ;
        PRESENT 2
                OUT Kval ;
                IF Kack & !Kpress NEXT 0 ;
                DEFAULT NEXT 2 ;
}

/* *************** KEY BUFFER ***************** */

/* ******* OUTPUT REGISTER ******** */

[O0..1].d = [R0..1] ;
[O2..3].d = [Q0..1] ;
[O0..3].sp = 'b'0 ;
[O0..3].ar = 'b'0 ;
O0.ck = Wreg ;
O1.ck = Wreg ;
O2.ck = Wreg ;
O3.ck = Wreg ;

/* *********** KEY BUFFER CONTROL *********** */
DAV = Kval ;

[X0..1].ar = 'b'0 ;
[X0..1].sp = 'b'0 ;
[X0..1].ckmux = Mclk ;

SEQUENCE [X1, X0] {
        PRESENT 0
                IF DAV & !ack NEXT 1 ;
                DEFAULT NEXT 0 ;
        PRESENT 1
                OUT Wreg;
                DEFAULT NEXT 2 ;
        PRESENT 2
                OUT Dval, DAC ;
                IF !DAV NEXT 3 ;
                DEFAULT NEXT 2 ;
        PRESENT 3
                OUT Dval ;
                IF ack NEXT 0 ;
                DEFAULT NEXT 3 ;

}
```

Acerca dos pins de *input* e *output*, nos pins de *input* residem os dois *clocks*, o *Mclk* para as máquinas de estados e um *Clk* para o contador, as entradas do Pencoder e ainda o sinal *ACK* que vem o control.
Nos pins de *output* residem as saídas negadas dos do decoder, que vão para o teclado matricial, *Dval* para o control, o *output* do *OutputRegister*, e portanto o codigo da tecla, e nos dois ultimos que os *flip-flops* do *key buffer control* que sao usados internamente. Ao mesmo tempo nos pinnodes residem os restantes *flip-flops* que são usados internamente.

Autores: João Nunes (A47220@alunos.isel.pt)
Alexandre Luís (A47222@alunos.isel.pt)
Miguel Marques (A47204@alunos.isel.pt)

D. Esquema elétrico do módulo *Keyboard Reader*



Title: KeyBoard Reader		
Size: A	Document Number: 1	Rev: 2
Date: Monday, June 01, 2020	Sheet 1 of 1	

Para o circuito apresentado, tendo em conta o hardware disponivel para a realização dos vários módulos, aqui em especial, o teclado matricial 4x4, sabendo que este tem um maximo de 20mA de corrente de contacto é possivel calcular o valor necessário para as resistencias a utilizar, neste esquema elétrico são usadas 4 resistencias de 1k cada, que equivale a 5mA quando ligadas a 5V, estando dentro do valor máximo para o teclado.
O clock *clk*, que irá servir para o contador do módulo *Key Decode* deverá ser mais lento que o *Mclk*, portanto, o dobro do tempo de *Mclk* deverá ser suficiente para impedir que as máquinas se destabilizem e provoquem avarias.

Autores: João Nunes (A47220@alunos.isel.pt)
Alexandre Luís (A47222@alunos.isel.pt)
Miguel Marques (A47204@alunos.isel.pt)

E. Código Java da classe *HAL*

```
package edu.isel.lic.link;

import isel.leic.*;
import isel.leic.utils.Time;

import java.util.Scanner;

public class HAL // Virtualiza o acesso ao sistema UsbPort
{
  public static int input, output;
  public static final int MAX_BITS = 0xFF;
  private static final boolean ULICX = false; // mudar para true caso estiver a ser usado uLICx

  // Inicia a classe
  public static void init() {
    clrBits(MAX_BITS);
  }

  // Retorna true se o bit tiver o valor lógico '1'
  public static boolean isBit(int mask) {
    return (mask == readBits(mask));
  }

  // Retorna os valores dos bits representados por mask presentes no UsbPort
  public static int readBits(int mask) {
    getInput();
    return mask & input;
  }

  // Escreve nos bits representados por mask o valor de value
  public static void writeBits(int mask, int value) {
    output = mask & value | ~mask & output;
    updateOutput();
  }

  // Coloca os bits representados por mask no valor lógico '1'
  public static void setBits(int mask) {
    output = output | mask;
    updateOutput();
  }

  // Coloca os bits representados por mask no valor lógico '0'
  public static void clrBits(int mask) {
    output = output & ~mask;
    updateOutput();
  }

  // Atualiza a saída no UsbPort com o valor da variável output
  private static void updateOutput() {
    UsbPort.out(ULICX ? output : ~output);
  }

  // Atualiza a variável input com a entrada do UsbPort
  private static void getInput() {
    input = (ULICX) ? UsbPort.in() : ~UsbPort.in();
  }
}
```

Autores: João Nunes (A47220@alunos.isel.pt)
Alexandre Luís (A47222@alunos.isel.pt)
Miguel Marques (A47204@alunos.isel.pt)

F. Código Java da classe *KBD*

```
package edu.isel.lic.peripherals;

import edu.isel.lic.link.HAL;
import isel.leic.utils.Time;

public class KBD // Ler teclas. Métodos retornam '0'..'9','A'..'F' ou NONE.
{
  public static final char NONE = 0x20; // Para podermos usar as posições iniciais da CGRAM
  public static final String kbd="147*2580369#";
  public static final int Dval_MASK = 0x10, DATA_MASK = 0x0F, ACK_MASK = 0x20;

  // Inicia a classe
  public static void init() {
    HAL.clrBits(ACK_MASK);
  }

  // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
  public static char getKey() {
    char key = NONE;

    if (HAL.isBit(Dval_MASK)) {
      key = kbd.charAt(HAL.readBits(DATA_MASK));
      HAL.setBits(ACK_MASK);
      //Time.sleep(1);
      HAL.clrBits(ACK_MASK);
    }

    return key;
  }

  // Retorna quando a tecla for premida ou NONE após decorrido 'timeout' milisegundos.
  public static char waitKey(int timeout) {
    char key;
    long starting_time = Time.getTimeInMillis(), current_time;
    boolean keyPress = false;

    do {
      current_time = Time.getTimeInMillis();
      key = getKey();
      if (key != NONE)
        keyPress = true;
    } while((current_time - starting_time < timeout) && !keyPress);

    return key;
  }
}
```