ISEL
ADEETC
Área Departamental de Engenharia Electrónica e de Telecomunicações e de Computadores

**Licenciatura em Engenharia Informática e de Computadores**

# *Working Time Recorder*

Projeto
de
Laboratório de Informática e Computadores
2017 / 2018 verão

**Autores:**
Margarida Nunes n.º 44781
Rodrigo Marques n.º 44820
Rui Paças n.º 44815

# 1 Keyboard Reader

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.

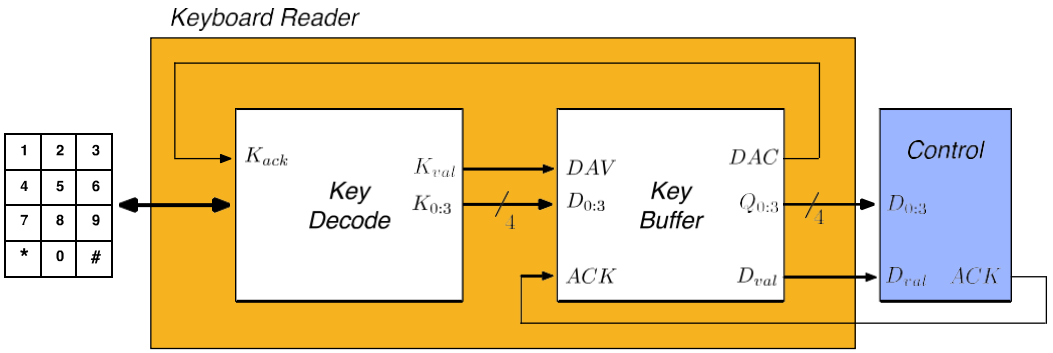
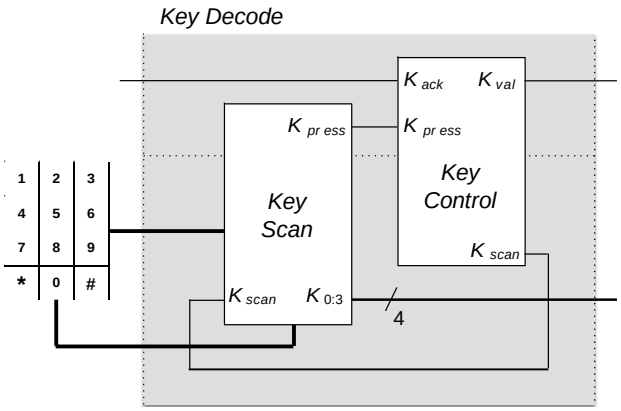



Figura 1 – Diagrama de blocos do módulo *Keyboard Reader*

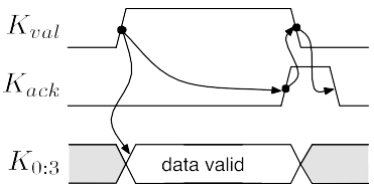
## 1.1 Key Decode

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. Para este bloco optou-se pela implementação da versão 3 proposta no enunciado do trabalho, que, em relação às restantes, necessita de menos *clocks*. Isto porque existe apenas um contador (que conta de 0 a 3) e um registo.

O bloco *Key Control* foi implementado pela máquina de estados representada em *ASM-chart* na Figura 4. No estado 00, a máquina de estados faz Kscan0 para que o CE (*count enable*) do contador esteja ativo e este possa contar. Testa *Kpress* para saber se existe uma tecla premida ou não. Em caso negativo, continua neste estado. Se existir *Kpress* passa para o próximo estado (01), onde faz *Kval* porque foi detetada pressão numa tecla e Kscan1 para que o registo possa cumprir a sua função, registar, pois já existe *data valid* para registar. Testa *Kack* (com base no diagrama temporal) e se este for *false* fica no mesmo estado. Caso contrário, passa para o estado 10, onde deixa de fazer *Kval*. Neste estado testa novamente *Kack* e se este for *true* continua neste estado. Se não testa *Kpress* e se este for *false* vai para o estado 00 se não continua no estado 10.

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

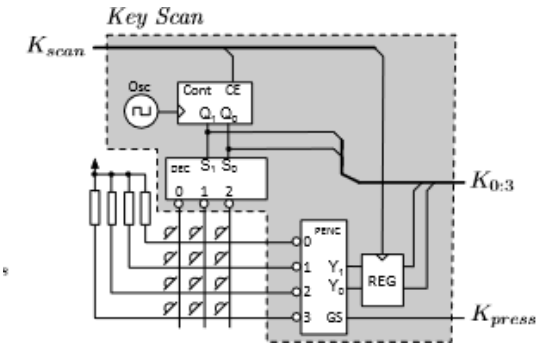



Figura 3 - Diagrama de blocos do bloco *Key Scan*

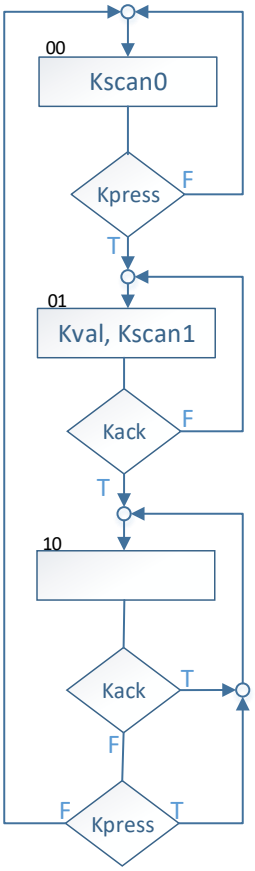



Figura 4 – Máquina de estados do bloco *Key Comtrol*

## 1.2 Key Buffer

O bloco *Key Buffer* implementa uma estrutura de armazenamento de dados, com capacidade para armazenar uma palavra de quatro bits. A escrita de dados no bloco *Key Buffer*, cujo diagrama de blocos é apresentado na Figura 5, inicia-se com a ativação do sinal *DAV* (*Data Available*) pelo sistema produtor, neste caso pelo bloco *Key Decode*, indicando que tem dados para serem armazenados. Logo que tenha disponibilidade para armazenar informação, o bloco *Key Buffer* regista 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 o sinal *DAC* seja ativado. O bloco *Key Buffer* só desativa o sinal *DAC* após o sinal *DAV* ter sido desativado. A implementação do bloco *Key Buffer* é baseada numa máquina de estados de controlo (*Key Buffer Control*) e num registo, designado por *Output Register*.
O sub-bloco *Key Buffer Control* do bloco *Key Buffer* também é responsável pela interação com o sistema consumidor, neste caso o módulo *Control*. Quando pretende ler dados do bloco *Key Buffer*, o módulo *Control* aguarda que o sinal $D_{val}$ fique ativo, recolhe os dados e ativa o sinal *ACK* para indicar que estes já foram consumidos. Logo que o sinal *ACK* fique ativo, o módulo *Key Buffer Control* invalida os dados baixando o sinal $D_{val}$. Para que uma nova palavra possa ser armazenada é necessário que o módulo *Control* desative o sinal *ACK*.

A máquina de estados do *Key Buffer Control* é representada em *ASM-chart* na Figura 6. No estado 00 testa-se *DAV*. Se este for falso, a máquina de estados continua no mesmo estado. Se este for verdadeiro passa para o próximo estado (01), onde faz *DAC* e *Wreg*; pois existem dados para serem registados. Deste estado passa-se para o próximo sem se ter de testar *DAV*, porque os *clocks* estão em oposição de transição. No estado 10, faz-se *Dval* para recolher os dados e testa-se *ACK*. Se for falso continua neste estado. Caso contrário, passa para o próximo estado (11) para invalidar os dados, baixando o *Dval*. Volta-se a testar *ACK*, para saber se já existem condições de armazenar uma nova palavra. Portanto, se *ACK* for verdadeiro continua no mesmo estado. Se for falso vai para o estado 00.

A descrição hardware do bloco *Key Buffer* em CUPL/VHDL encontra-se no Anexo B.

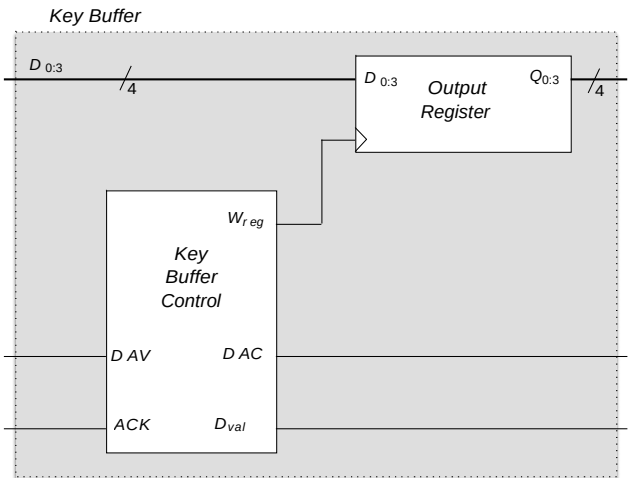



Figura 5 – Diagrama de blocos do bloco *Key Buffer*

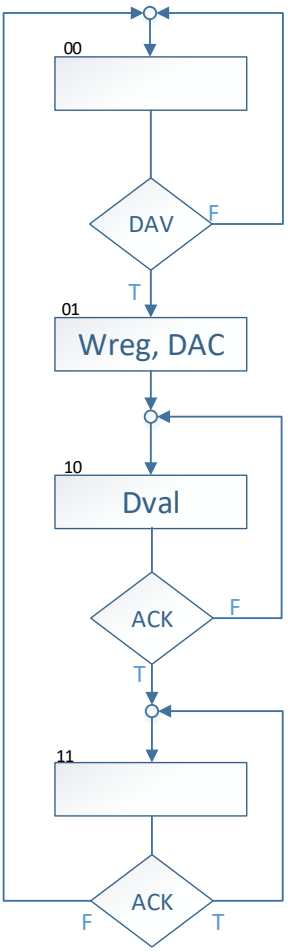



Figura 6 – Máquina de estados do bloco *Key Buffer Control*

Com base nas descrições dos blocos *Key Decode* e *Key Buffer* implementou-se o módulo *Keyboard Reader* de acordo com o esquema elétrico representado no Anexo F.
No bloco *Key Scan*, do *Key Decoder*, são utilizadas resistências porque as entradas do teclado matricial estão sempre '1' e, portanto, é necessário ter a certeza do valor lógico presente no local. Para este efeito escolheu-se resistências entre 1kΩ e 10 KΩ, devido à tensão máxima ser 5V.
Para as máquinas de estado do *Key Control* e do *Key Buffer Control*, é utilizado o mesmo *clock*. Isto é possível porque os *clocks* das máquinas estão em oposição de transição. Quando um está a '1' o outro está a '0'.

# 2 Serial Output Controller

O módulo *Serial Output Controller* (*SOC*) implementa a interface com o *LCD* e com o mecanismo da porta, fazendo a receção em série da informação enviada pelo módulo de controlo e entregando-a posteriormente ao destinatário, conforme representado na Figura 7.

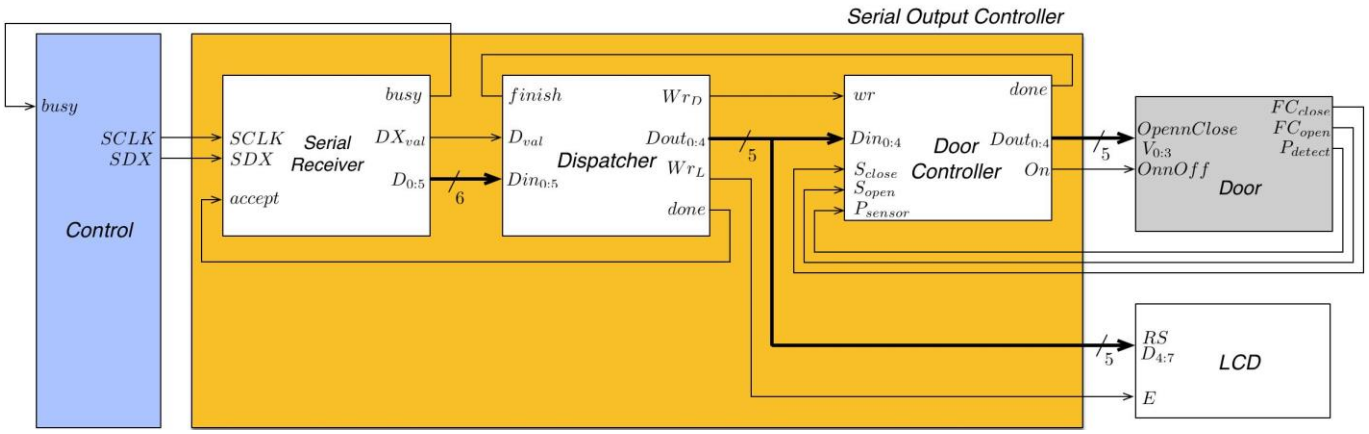



Figura 7 – Diagrama de blocos do módulo *Serial Output Controller*.

O módulo *SOC* recebe em série uma mensagem constituída por 6 bits de informação e um bit de paridade. A comunicação com este módulo realiza-se segundo o protocolo ilustrado na Figura 8, em que o bit *LnD* identifica o destinatário da mensagem. Nas mensagens para o LCD, ilustrado na Figura 9, o bit *RS* é o primeiro bit de informação e indica se a mensagem é de controlo ou dados. Os seguintes 4 bits contêm os dados a entregar ao *LCD*. O último bit contém a informação de paridade par, utilizada para detetar erros de transmissão.
As mensagens para o mecanismo da porta, ilustradas na Figura 10, contém para além do bit *LnD* e do bit paridade mais 5 bits de dados a entregar ao dispositivo, em que *OC* representa se o comando é de abrir ou fechar, e os restantes bits a velocidade do motor.

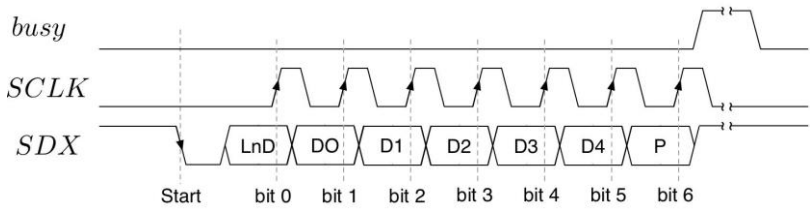



Figura 8 – Protocolo de comunicação com o módulo *Serial Output Controller*.

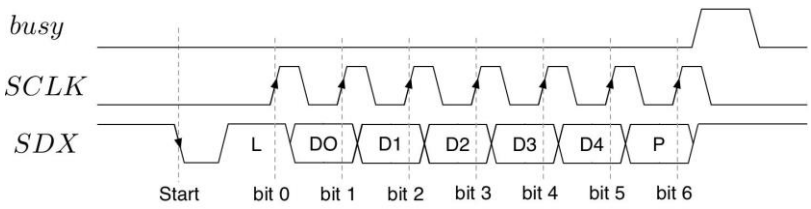



Figura 9 – Trama para o *LCD*.

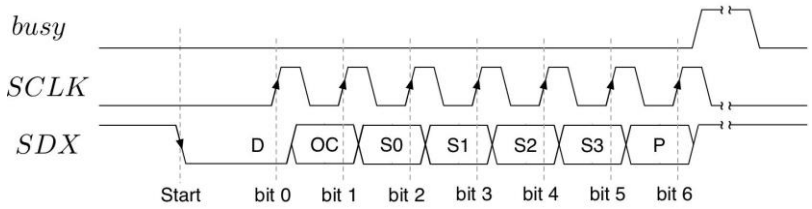



Figura 10 – Trama para o mecanismo da porta (*Door*).

O emissor, realizado em *software*, quando pretende enviar uma trama para o módulo *SOC* promove uma condição de início de trama (*Start*), que corresponde a uma transição descendente na linha *SDX* com a linha *SCLK* no valor lógico zero. Após a condição de início, o módulo *SOC* armazena os bits de dados da trama nas transições ascendentes do sinal *SCLK*.
.

## 2.1 Serial Receiver

O bloco *Serial Receiver* do módulo *SOC* é constituído por quatro blocos principais: *i*) um bloco de controlo; *ii*) um bloco conversor série paralelo; *iii*) um contador de bits recebidos; e *iv*) um bloco de validação de paridade, designados por *Serial Control*, *Shift Register*, *Counter* e *Parity Check* respetivamente. O bloco *Serial Receiver* deverá ser implementado com base no diagrama de blocos apresentado na Figura 11.

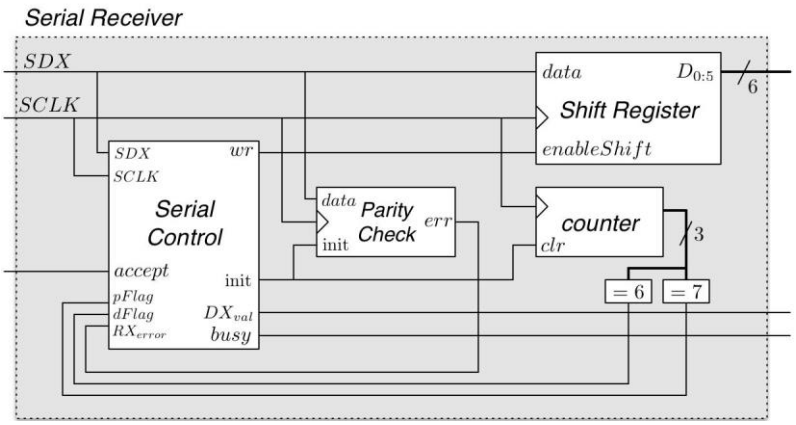



Figura 11 – Diagrama de blocos do bloco *Serial Receiver*

A máquina de estados do *Serial Control* é representada em *ASM-chart* na Figura 13 e inclui outra máquina de estados para o *Start* representada também em *ASM-chart* na Figura 12. Na máquina de estados do *Start* cumpre-se o protocolo de iniciação de acordo com os diagramas temporais nas figuras 8,9 e 10. No estado 00 testa-se *SCLK* e *SDX*, se estes forem false e true, respetivamente, vai para o estado 01. Caso contrário fica no mesmo estado. No estado 01 volta-se a testar *SCLK* e *SDX*, se forem ambos false passa para o estado 10 onde faz *Start* regressando ao estado 00.

No *ASM-chart* do *Serial Control* segue-se o protocolo do diagrama temporal da figura 8. No estado 000 testa-se *Start* se este for *true* vai para o estado 001, caso contrário permanece neste estado. No estado 01 faz *init* e testa novamente *Start* se este for *false* vai para o próximo estado (010) onde faz *Wr*. Neste estado testa uma vez mais *Start* se este for *true* regressa ao estado 001, se não testa *dFlag*. Se *dFlag* for *true* vai para o estado 011, caso contrário fica no mesmo estado (010). No estado 011 testa *Start*, *pFlag* e *RxErr*, se estes forem *false*, *true* e *false*, respetivamente, vai para o próximo estado (100) onde faz *DxVal* e *Busy*. Neste vai testar *accept* se for *true* passa para o estado 101, caso contrário permanece no mesmo estado. No estado 101 faz *Busy* e volta a testar *accept* se for *false* vai para o estado 00, se não fica neste estado.

A descrição hardware do bloco *Serial Receiver* em CUPL/VHDL encontra-se no Anexo C.

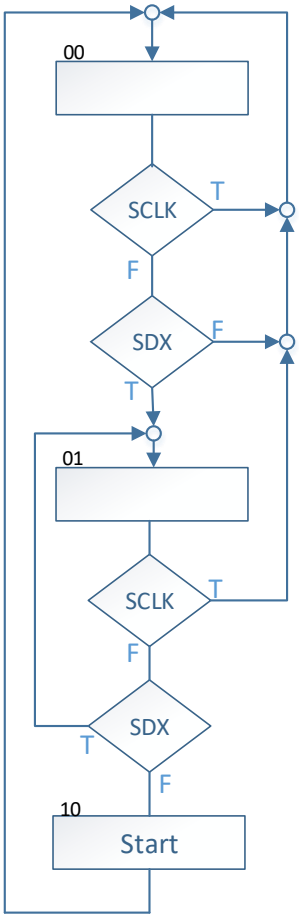



Figura 12 – Máquina de estados do *Start* da máquina de estados do bloco *Serial Contol*.

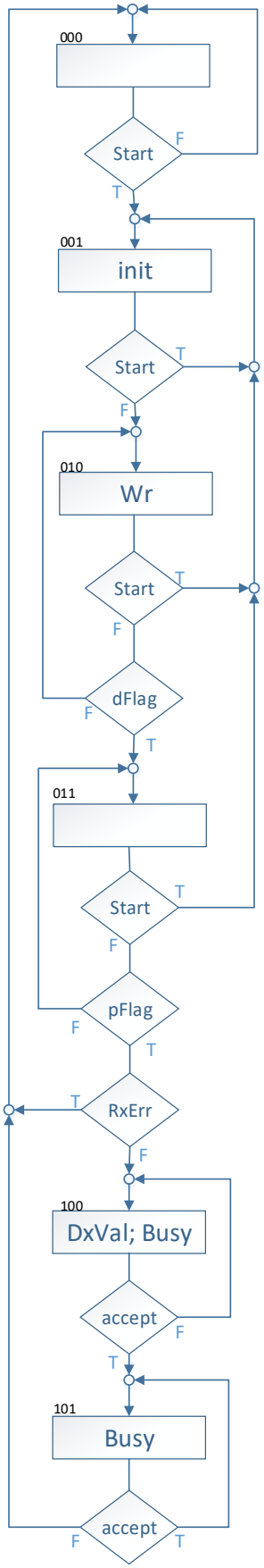



Figura 13 – Máquina de estados do bloco *Serial Contol*.

## 2.2 Dispatcher

O bloco *Dispatcher* é responsável pela entrega das tramas válidas recebidas pelo bloco *Serial Receiver* ao *LCD* e ao *Door Controller*, através da ativação do sinal $Wr_L$ e $Wr_D$. A receção de uma trama válida é sinalizada pela ativação do sinal $D_{val}$. O processamento das tramas recebidas pelo *SOC*, para o *LCD* ou para o *Door Controller*, deverá respeitar os comandos definidos pelo fabricante de cada periférico, devendo libertar o canal de receção série logo que seja possível.

A máquina de estados do *Dispatcher* é representada em *ASM-chart* na Figura 14. No estado 00 testa *Dval* e *Din0* se estes forem ambos *true* a máquina de estados vai para o estado 01 onde faz *Wrl* e passa para o estado 11 que faz *DispDone*. Se *Dval* for false vai para o estado 00 e se *Din0* for *false* vai para o estado 10. No estado 10 faz *Wrd* e testa *finish* se este for *true* passa para o estado 11, caso contrário mantem-se no mesmo estado. No estado 11 testa *Wr* se este for *false* regressa ao estado 00, se for *true* fica no mesmo estado.

A descrição hardware do bloco *dispatcher* em CUPL/VHDL encontra-se no Anexo D.

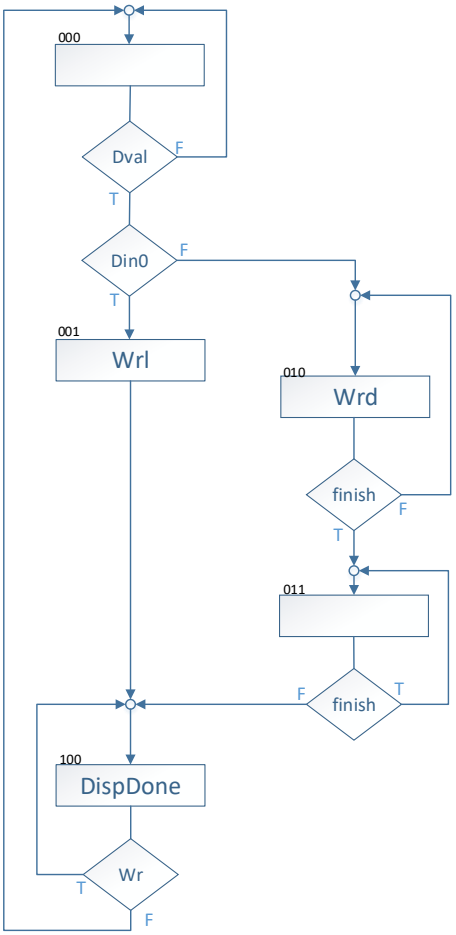



Figura 14 – Máquina de estados do bloco *dispatcher*.

## 2.3 Door Controller

O bloco *Door* Controller após ter recebido um comando válido entregue pelo *Dispatcher*, deverá proceder à atuação deste no mecanismo da porta. Se o comando recebido for de abertura da porta, o *Door Controller* deverá colocar o sinal $OnnOff$ e o sinal $OpennClose$ no valor lógico '1', até o sensor de porta aberta ($FC_{open}$) ficar ativo. No entanto se o comando for de fecho, o *Door Controller* deverá ativar o sinal $OnnOff$ e colocar o sinal $OpennClose$ no valor lógico '0', até o sensor de porta fechada ($FC_{close}$) ficar ativo. Se durante o fecho for detetada uma pessoa na zona da porta, através do sensor de presença

($P_{detect}$), o sistema deverá interromper o fecho reabrindo a porta. Após a interrupção do fecho da porta o bloco *Door Controller* deverá permitir de forma automática, ou seja, sem necessidade de envio de um novo comando, o encerramento da porta e o finalizar do comando de fecho. Após concluir qualquer um dos comandos o *Door Controller* sinaliza o *Dispatcher* que está pronto para processar um novo comando através da ativação do sinal *done*.

A máquina de estados do *Door Controller* é representada em *ASM-chart* na Figura 15. No estado 000 testa *Wr* e *Din1*, se estes forem ambos *true* vai para o estado 001 onde faz *On*; *OpennClose*. Se *Wr* for false vai para o estado 000 e se *Din1* for false vai para o estado 010 onde faz *On*. No estado 001 testa *Sopen* se for *true* vai para o estado 100 onde faz *DoorDone*, caso contrário fica no mesmo estado. No estado 010 testa *Sclose* e *Psensor* se estes forem *false* e *true*, respetivamente, vai para o estado 011 onde faz *On* e *OpennClose*. Se *Sclose* for *true* vai para o estado 100 e se *Psensor* for false fica no mesmo estado (010). No estado 011 testa *Sopen* se for *true* vai para o estado 010 e se for false fica no mesmo estado. No estado 100 testa *Wr* se for false vai para o estado 000, caso contrário fica no mesmo estado.

A descrição hardware do bloco *Door Controller* em CUPL/VHDL encontra-se no Anexo E.

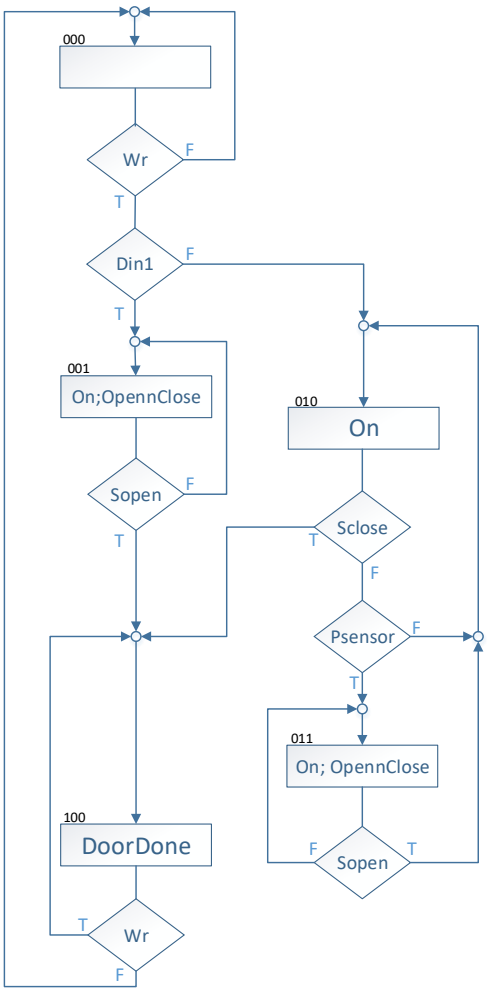



Figura 15 – Máquina de estados do bloco *Door Controller*.

Com base nas descrições dos blocos *Serial Receiver*, *dispatcher* e *Door Controller* implementou-se o módulo *Serial Output Controller* de acordo com o esquema elétrico representado no Anexo F.

Neste módulo o *clock* do bloco *Dispatcher* e do *Door Controller* são o mesmo.

# 3 Control

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem Java e seguindo a arquitetura lógica apresentada na Figura 16.

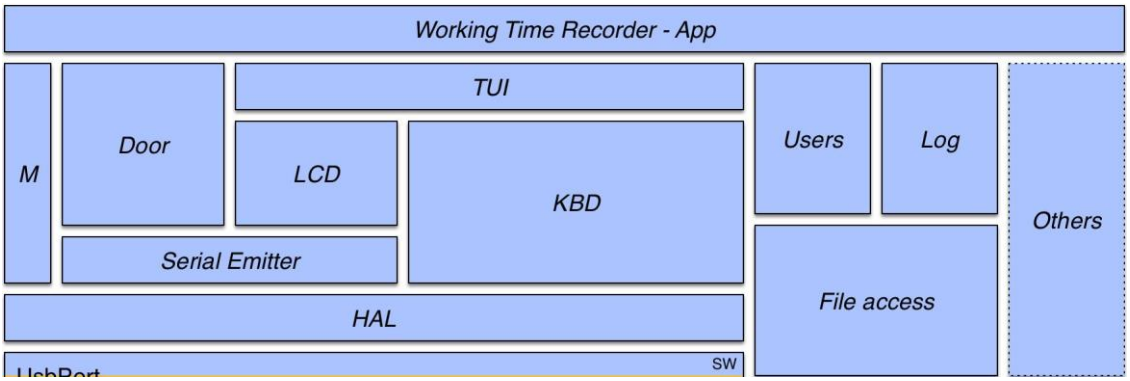



Figura 16 – Diagrama lógico do módulo *Control* de interface com o módulo *Keyboard Reader e com o módulo Serial Output Controller*.

As classes representadas na figura 16 foram desenvolvidas e são descritas nas secções 3.1 a 3.11, e o código fonte desenvolvido nos Anexos H a S, respetivamente.

## *3.1* Classe *HAL*

Classe responsável pela interação entre o *Software* e o *Hardware* através do *USBPort*. Aqui estão implementados, conforme o enunciado, os métodos init (que inicia a classe), o isBit (retorna true se o bit tiver o valor lógico '1'), readBits (retorna os valores dos bits representados por *mask* e apresenta-os no *UsbPort*), writeBits (escreve nos bits representados por *mask* o valor de *value*), setBits (coloca os bits representados por *mask* no valor lógico '1') e clrBits (coloca os bits representados por *mask* no valor lógico '0'). Ainda implementámos mais dois métodos o in e o out, que negam o resultado no input e no output respetivamente, visto que por omissão o *USBPort* tem as entradas e as saídas a '1'.

## *3.2* Classe *KBD*

Classe responsável pela leitura das teclas. Aqui estão implementados, conforme o enunciado, os métodos init (que inicia a classe), o getKey (retorna de imediato a tecla premida ou NONE se não há tecla premida) e waitKey (retorna quando a tecla for premida ou NONE após decorrido '*timeout*' milisegundos).

## *3.3* Classe LCD

Classe responsável pela escrita no display usando a interface a 4 bits. Foram implementados os métodos: init (que define o protocolo de comunicação a 4 bits com o LCD), WriteNibbleParallel e o WriteNibble ( ambos escrevem um nibble (4 bits) de comando /dados no LCD) que usa máscaras e o hardware respetivamente,WriteByte (escrever um Byte (8 bits) de comando/dados no LCD à custa de duas chamadas do WriteNibble), WriteCMD (escreve um comando no LCD, ou seja, chama o writeByte com o bit de RS a false), WriteData (escreve um dado no LCD, ou seja, chama o método writeByte com o bit de RS a true), Write (char c) (escreve um carácter na posição corrente à custa do método WriteData), Write(String s) (escreve uma string na posição corrente à custa de chamar o write (char c) para cada caracter da string s), Cursor (colocar o cursor na posição indicada pelos parâmetros passados) e Clear (limpa o display e coloca o cursor na primeira linha e primeira coluna), Off (comando para desligar o LCD), getStartingCol (retorna o número da coluna em que começa a String passada como parâmetro), clearLine (limpa a linha cujo número é passado como parâmetro).

## 3.4 Classe Serial Emitter

Classe responsável pelo envio de tramas para o LCD e *DoorMechanism*. Aqui estão implementados os métodos: init (inicia a classe), send (envia uma trama para o *Serial Receiver* identificando o destino no parâmetro addr e os bits de dados em data), isBusy (retorna *true* se o canal estiver ocupado) e start (executa as condições de iniciação necessárias). Também está implementado o enumerado que define o destino para o qual a trama é enviada.

## 3.5 Classe Door

Classe responsável por controlar o mecanismo da porta. Aqui estão os métodos: init (inicia a classe, estabelecendo os valores iniciais), open (envia comando para abrir a porta indicando a velocidade), close (envia comando para fechar a porta indicando a velocidade), isFinished (retorna *true* se o processo de fecho ou de abertura já está terminado) e writeCMD (escreve um comando para o mecanismo da porta).

## 3.6 Classe M

Classe responsável pela virtualização do botão de manutenção. Aqui estão implementados os métodos: init (inicia a classe) e isOn (retorna *true* se o botão de manutenção estiver ligado).

## 3.7 Classe TUI

Classe responsável pela interação do utilizador com o LCD. Aqui estão implementados os métodos: init (inicia a classe), readInteger (retorna o valor dos dígitos introduzidos e escreve-os na posição line e col, em escrita visível ou oculta dependendo do boolean hide), writeMessage (escreve uma mensagem no LCD), clear (limpa o que estiver escrito no LCD), turnOff (envia o comando para desligar o LCD), isPressed (retorna true se a tecla premida for igual a que é passada como parâmetro), setCol (vai buscar a coluna onde começa a String passada como parâmetro), clearLine (limpa a linha cujo número é passado como parâmetro).

## 3.8 Classe Users

Classe responsável pela lista de todos os utilizadores. Para guardar os utilizadores nesta classe optamos por um array pois permite o acesso pelo índice de uma forma eficiente e rápida e começa com uma dimensão inicial escolhida. Também constatamos alguns pontos menos positivos da utilização do array como, por exemplo, o facto de vários índices ficarem com referências a null quando existem poucos utilizadores. Ponderamos também outros contentores de dados, dos quais consideramos o hashMap um forte candidato. O hashMap também permite o acesso por indíce mas com uma key que é transformada num índice e é dinâmico, ou seja, a sua dimensão vai crescendo dependendo da necessidade, por isso não tem tantas referências a null como o array. Mas isso também pode ser uma desvantagem visto que cada vez que é criado um hashMap novo com o dobro da dimensão tem de se copiar todas as referências existentes no outro. Aqui estão implementados os métodos: init (inicia a classe), setDimension (calcula a dimensão do array que contém os utilizadores), createNewUser (retorna true se for possível adicionar um utilizador criado ao array Users), addUser (adiciona um utilizador já existente ao *array*), canAdd (verifica se é possível adicionar um utilizador ao array na posição number), load (reúne num arrayList as informações dos utilizadores lidas do ficheiro), save (cria um novo utilizador no array com a informação que reúne e passa a String que é suposto ser escrita no ficheiro de testo), userIsValid (retorna *true* se o utilizador da posição number corresponder ao pin introduzido), getUser (retorna um utilizador), miliToHours (retorna o valor em horas), miliToMinutes (retorna o valor em minutos), remove (comando que em modo de manutenção remove um utilizador do sistema), getFormat (retorna uma String formatada), create (comando que em modo de manutenção cria um novo utlizador) e listUsers (comando que em modo de manutenção lista os utilizadores que estão no momento a trabalhar).

## 3.9 Classe User

Classe responsável pela informação de cada utilizador. Contém todos os campos que cada utilizador deve ter. Aqui estão implementados alguns métodos nomeadamente: User (cria um novo utilizador), User (String s) (cria um novo utilizador com os dados passados na String s), isIn (retorna true se o utilizador fez login e se encontra a trabalhar), masckedPin (encripta o pin), isPin (retorna true se o pin introduzido for igual ao pin que está encriptado), toString (escreve a String com o número do utilizador, o pin encriptado, o nome, as horas acumuladas e no caso da pessoa se encontrar a trabalhar as horas a que foi feito o login) e fromString (separa os diferentes campos que identificam um utilizador e cria o mesmo, chamando o construtor User (String s)).

## 3.10 Classe FileAcess

Classe responsável pela escrita e leitura nos ficheiros de texto Users e Log. Aqui estão implementados os métodos: write (escreve nos ficheiro ficheiros de texto Users ou Log) e readUsers (lê dos ficheiros de texto Users ou Log).

## 3.11 Classe Log

Classe responsável pelo login feito pelo utilizador. Contém os campos com a informação de cada login e o método toString que retorna uma String de login ou logout de cada utlizador que efetua estas ações.

## 3.12 Classe APP (Working Time Recorder)

Classe responsável pela ligação de todas as outras classes, direta ou indiretamente. Aqui estão implementados os métodos init (inicia a classe), launchApp (responsável pelo normal funcionamento da aplicação), inOrOut (escreve as mensagens no LCD quando são efetuados login ou logout), maintenance (quando o modo de manutenção está ativo), changePin (quando se pretende alterar o pin de um utilizador), doorSequence (escreve no LCD os comandos da sequência da porta, ou seja, quando esta está a

abrir ou a fechar), exit (comando quando em modo de manutenção saí do mesmo), doCommand (vários comandos que podem ser executados quando em modo de manutenção), commandList (em modo de manutenção lista os comando que podem ser efetuados), off (desliga o sistema) e dateChange (muda a data e a hora no LCD quando estas são alteradas).

# Conclusões

O módulo *KeyBoard Reader* tem como função descodificar o teclado e determinar a tecla pressionada e o seu código. Para a implementação deste utilizou-se um teclado, uma *PAL ATF750C* e um registo 74 574.
Quanto à deteção das teclas pressionadas do teclado, existe um compasso de espera devido ao número de *clocks* necessários e ao tempo que a PAL e o registo levam a identificá-la e a guardá-la.
O módulo *Seria Output Controller* faz a receção em série da informação enviada pelo controlo e entregando-a posteriormente a com quem comunica, o *LCD* ou o mecanismo da porta. Para a implementação deste módulo utilizou-se um *LCD* e duas *PAL ATF750C*.
Para control foram definidas as classes: HAL, KBD, LCD, Serial Emitter, Door, M, TUI, Users, User, FileAccess, Log e App.

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

```
/*Key Scan*/

/*Decoder */

D0 = !(!T0 & !T1);
D1=  !(T0 & !T1);
D2 = !(!T0 & T1);


/* Counter */

[T0..1].sp = 'b'0 ;
[T0..1].ar = 'b'0;
[T0..1].CK = !CLK;
T0.t = Kscan0;
T1.t= T0&Kscan0;

/*PENC*/

Kpress=!E0#!E1#!E2#!E3;

Y0=(!E1&E2)#!E3;
Y1=!E3#!E2;


/*Registo*/

[T2,T3].sp='b'0;
[T2,T3].ar='b'0;
[T2,T3].CK = Kscan1;
[T2,T3].d = [Y0,Y1];


/*Key Control*/

/*Flip Flop tipo D*/

[X0..1].sp = 'b'0;
[X0..1].ar= 'b'0;
[X0..1].CK = CLK;

sequence [X0,X1]{
Present 0
       out Kscan0;
       if Kpress next 1;
       default next 0;

Present 1
       out Kval,Kscan1;
       if Kack next 2;
       default next 1;

Present 2
       if !Kack&!Kpress next 0 ;
       default next 2;
}
```

```
/* *************** INPUT PINS *********************/
PIN      1 = CLK                        ; /*                              */
PIN      [3..6]=[E0..3]                 ; /*                              */
PIN      8 = ACK                        ; /*                              */


/* *************** OUTPUT PINS *********************/
PIN          14 = Dval                              ;
PIN          15 = Kpress                            ;
PIN          16 = Wreg                              ;
PIN          [17..19] = [D0..2]                     ;
PIN          [20..23] = [T0..3]     ;
PINNODE      [31,32] = [K0,K1]                      ;
PINNODE      [33,34] = [X0,X1]                      ;
```

Figura 17 – Declaração de pinos de entrada e saída do módulo KEYBOARD READER.

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

A listagem dos Pinos de entrada e saída deste bloco estão apresentados na figura 8.

```
/*Key Buffer */

DAV = Kval ;
Kack = DAC ;

/*Key Buffer Control*/
[K0..1].sp='b'0;
[K0..1].ar ='b'0;
[K0..1].CK= !CLK;



sequence [K0,K1]{
present 0
       if DAV next 1;
       default next 0;
present 1
       out DAC,Wreg;
       next 2;
present 2
       out Dval;
       if ACK next 3;
       default next 2;
present 3
       if !ACK next 0;
       default next 3;
}
```

# C. Descrição CUPL/VHDL do bloco *Serial Receiver*

```
/* *************** INPUT PINS *********************/
PIN     1 = SCLK                    ; /*                                 */
PIN     2 = MCLK ;
PIN     3 = SDX                     ; /*                                 */
PIN     4=  accept                  ; /*                                 */

/* *************** OUTPUT PINS *********************/
PIN     [14..19] = [D0..5]          ; /*                                 */
PIN     [20..21] = [K0..1]          ; /*                                 */
PINNODE 31 = K2 ;
PINNODE 33 = Y0;
PIN     22  = DxVal;
PIN     23  = Busy;
PINNODE   [25,26] = [X0,X1]         ; /*                                 */
PINNODE   [27,28,29] = [T0,T1,T2]         ; /*
*/
```

Figura 18 – Declaração de pinos de entrada e saída do módulo Serial Receiver.

```
/*SERIAL CONTROL*/

[X0..1].ck= MCLK ;
[X0..1].ar= 'b'0;
[X0..1].sp= 'b'0;

[K0..2].ck= !MCLK ;
[K0..2].ar= 'b'0;
[K0..2].sp= 'b'0;
```

```
/*Shift Register*/

[D0..5].ckmux= SCLK;
[D0..5].ar= 'b'0;
[D0..5].sp= 'b'0;

D0.d = !Wr&D0#Wr&SDX;
D1.d = !Wr&D1#Wr&D0;
D2.d = !Wr&D2#Wr&D1;
D3.d = !Wr&D3#Wr&D2;
D4.d = !Wr&D4#Wr&D3;
D5.d = !Wr&D5#Wr&D4;

/*COUNTER*/

[T0..2].ckmux= SCLK;
[T0..2].ar= init;
[T0..2].sp= 'b'0;

T0.t= 'b'1;
T1.t= 'b'1 &T0;
T2.t= 'b'1 &T0&T1;

dFlag=!T0&T1&T2;
pFlag=T0&T1&T2;
```

```
/*PARITY CHECK*/
Y0.ckmux= SCLK ;
Y0.ar= init;
Y0.sp= 'b'0;

Y0.t= SDX;
RxErr=Y0;
```

```
Sequence [K0..2] {
Present 0
        if Start next 1;
         default next 0;
Present 1
        out init;
        if !Start next 2;
        default next 1;
Present 2
         out Wr;
        if !Start&dFlag next 3;
        if Start next 1;
        if !Start&!dFlag next 2;
Present 3
        if Start next 1;
        if !Start&!pFlag next 3;
        if !Start&pFlag&!RxErr next 4;
        if !Start&pFlag&RxErr next 0;
Present 4
        out DxVal,Busy;
        if accept next 5;
        default next 4;
Present 5
        out Busy;
         if accept next 5;
         default next 0;
}
```

```
Sequence [X0..1] {
Present 0
         if !SCLK&SDX next 1;
         default next 0;
Present 1
          if !SCLK&!SDX next 2;
          if SCLK next 0;
         if !SCLK&SDX next 1;
Present 2
        out Start;
        next 0;
}
```

# D. Descrição CUPL/VHDL do bloco *Dispacher*

```
[D0..2].ck= MCLK ;
[D0..2].ar= 'b'0;
[D0..2].sp= 'b'0;

Sequence [D0..2] {
Present 0
        if Dval & Din0 next 1;
        if Dval & !Din0 next 2;
        default next 0;
Present 1
        out Wrl;
        default next 4;
Present 2
        out Wrd;
        if Finish next 3;
        default next 2;
Present 3
        if !Finish next 4;
        default next 3;
Present 4
        out DispDone;
        if !Dval next 0;
        default next 3;
}
```

```
/* *************** INPUT PINS *********************/
PIN     1 =  MCLK                       ; /*                                 */
PIN     2 =  Dval                       ; /*                                 */
PIN    [3,4] = [Din0,Din1]             ; /*                                */
PIN     5 = Sclose;
PIN     6 = Sopen;
PIN     7 = Psensor;

/* *************** OUTPUT PINS *********************/
PIN      14=Wrd                         ; /*                                 */
PIN      15=Wrl                         ; /*                                 */
PIN      16=DispDone                        ; /*                                  */
PIN      17=DoorDone                     ;
PIN      18 = OpennClose                 ;
PIN      19 = On                         ;
PIN      [20,21,22] = [C0,C1,C2]         ;
PINNODE      [25,26]=[D0,D1]             ;
```

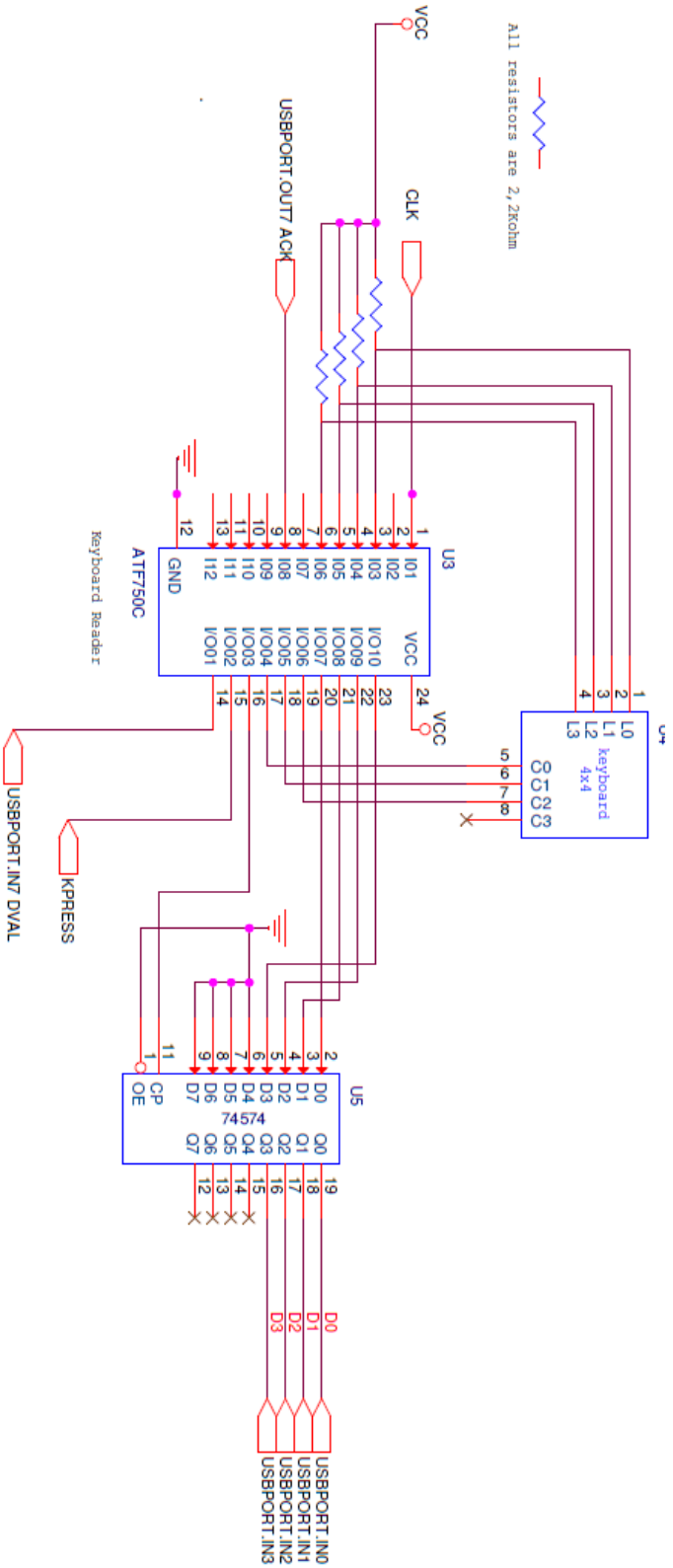
Figura 19 – Declaração de pinos de entrada e saída do módulo Dispacher e Door Controller.

# E. Descrição CUPL/VHDL do bloco *Door Controller*

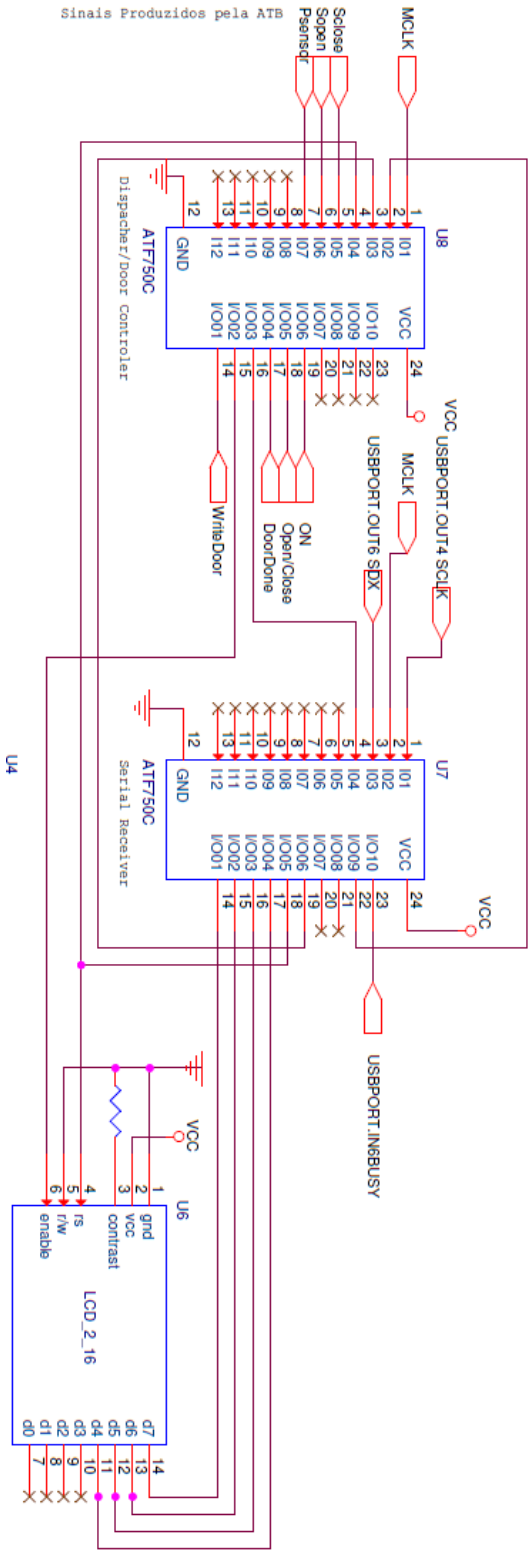
A listagem de pinos de entrada e saída deste bloco encontram-se na figura 19.

```
Wr=Wrd;
Finish=DoorDone;

[C0..2].ck= !MCLK ;
[C0..2].ar= 'b'0;
[C0..2].sp= 'b'0;

Sequence [C0..2] {
Present 0
       if Wr & Dinl next 1;
       if Wr & !Dinl next 2;
       default next 0;
Present 1
       out On,OpennClose;
       if Sopen next 4;
       default next 1;
Present 2
       out On;
       if Sclose next 4;
       if !Sclose & Psensor next 3;
       default next 2;

Present 3
       out On,OpennClose;
       if Sopen next 2;
       default next 3;

Present 4
       out DoorDone;
       if !Wr next 0;
       default next 4;
}
```

# F. Esquema elétrico do módulo *Keyboard Reader*



# G. Esquema elétrico do módulo Serial Output Controller



# H. Código Java da Classe *HAL*

```
public class HAL { // Virtualiza o acesso ao sistema UsbPort

    private static int out;

    // Inicia a Classe
    public static void init() {
        out(out = 0);
    }

    public static void out(int val) {
        UsbPort.out(~val);
    }

    public static int in() {
        return ~UsbPort.in();
    }

    // Retorna true se o bit tiver o valor lógico ‘1’
    public static boolean isBit(int mask) {
        return (readBits(mask) != 0);
    }

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

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

    // Coloca os bits representados por mask no valor lógico ‘1’
    public static void setBits(int mask) {
        out(out = mask | out);
    }

    // Coloca os bits representados por mask no valor lógico ‘0’
    public static void clrBits(int mask) {
        out(out = (out & (~mask)));
    }
}
```

# I. Código Java da Classe *KBD*

```
public class KBD { // Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.

   private static final char NONE = 0;

//Hardware table
   private static char[] table = {'1','2','3',NONE,
                                  '4','5','6',NONE,
                                  '7','8','9',NONE,
                                  '*','0','#',NONE};

//Simulator Table
  // private static char[] table = {'1','2','3','4','5','6','7','8', '9','*','0','#', NONE,NONE,NONE,NONE};

   private final static int ACK_MASK = 0x80; // out
   private final static int DVAL_MASK = 0x80; // in
   private final static int DATA_MASK = 0x0F; // in

   // 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() {
      if(HAL.isBit(DVAL_MASK)) {
        int key = HAL.readBits(DATA_MASK);
         HAL.setBits(ACK_MASK);
         while(HAL.isBit(DVAL_MASK)){
                // Espera por Dval = 0
             }
         HAL.clrBits(ACK_MASK);
         return table[key];
      }
      return NONE;
   }

   // Retorna quando a tecla for premida ou NONE após decorrido 'timeout' milisegundos.
   public static char waitKey(long timeout){
     long currTime= isel.leic.utils.Time.getTimeInMillis();
     long waitTime = currTime+timeout;
     char myChar;
     while((myChar=getKey())==NONE){
        if(isel.leic.utils.Time.getTimeInMillis()>=waitTime) break ;
     }
     return myChar;
   }

}
```

# J. Código Java da Classe LCD

```
public class LCD { // Escreve no LCD usando a interface a 4 bits.
   private static final int LINES = 2, COLS = 16; // Dimensão do display.

   private final static int ENABLE_MASK = 0x40; // out Only needed in Parallel
   private final static int DATA_MASK = 0x0F; // out
   private final static int RS_MASK = 0x20; // out Only needed in parallel
   private static final boolean SERIAL = true;

   public static void main(String[] args) {
      HAL.init();
      SerialEmitter.init();
      init();
   }

   // Escreve um nibble de comando/dados no LCD
   private static void writeNibbleParallel(boolean rs, int data) {
      if (rs)
         HAL.setBits(RS_MASK);
      else
         HAL.clrBits(RS_MASK);
      HAL.setBits(ENABLE_MASK);
      HAL.writeBits(DATA_MASK, data);
      HAL.clrBits(ENABLE_MASK);
      Time.sleep(10);
   }

   private static void writeNibbleSerial(boolean rs, int data) {
      SerialEmitter.send(SerialEmitter.Destination.LCD, (data << 1) | ((rs)? 1: 0));
   }

   private static void writeNibble(boolean rs, int data) {
      if (SERIAL)
         writeNibbleSerial(rs, data);
      else
         writeNibbleParallel(rs, data);
   }

   /* // Escreve um byte de comando/dados no LCD
    private static void writeByte(boolean rs, int data) {
        writeNibble(rs, data>>4&DATA_MASK);
        writeNibble(rs, data&DATA_MASK);
    }*/
   private static void writeByte(boolean rs, int data) {
      writeNibble(rs, data >> 4);
      writeNibble(rs, data);
   }

   // Escreve um comando no LCD
   private static void writeCMD(int data) {
      writeByte(false, data);
   }
```

```
// Escreve um dado no LCD
private static void writeDATA(int data) {
    writeByte(true, data);
}

// Envia a sequência de iniciação para comunicação a 4 bits.
public static void init() {
    //definir o protocolo de iniciacao
    Time.sleep(20);
    writeNibble(false, 0x03);
    Time.sleep(5);
    writeNibble(false, 0x03);
    Time.sleep(1);
    writeNibble(false, 0x03);
    writeNibble(false, 0x02);
    writeByte(false, 0x28);
    writeByte(false, 0x08);
    writeByte(false, 0x01);
    writeByte(false, 0x06);
    writeByte(false, 0x0F);
}

// Escreve um caráter na posição corrente.
public static void write(char c) {
    writeDATA((int) c);
}

// Escreve uma string na posição corrente.
public static void write(String txt) {
    for (int i = 0; i < txt.length(); i++) {
        write(txt.charAt(i));
    }
}

// Envia comando para posicionar cursor ('lin':0..LINES-1 , 'col':0..COLS-1)
public static void cursor(int lin, int col) {
    writeByte(false, (lin == 0 ? 128 : 192) + col);
}

// Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
public static void clear() {
    writeByte(false, 0x01);
    cursor(0, 0);
}

//Comando para desligar o LCD
public static void off(){
    writeByte(false,0x08);
}

// Retorna o número da coluna em que começa a String passada como parâmetro
public static int getStartingCol(String name){
    return (COLS-name.length())/2 ;
}
```

```
  // Limpa linha cujo número é passado como parâmetro
public static void clearLine(int line) {
        cursor(line,0);
        for (int i = 0; i < COLS; i++) {
            write(' ');
        }
    }
}
```

# K. Código Java da Classe Serial Emitter

```
public class SerialEmitter { // Envia tramas para os diferentes módulos Serial Receiver.

   private final static int BUSY_MASK = 0x40; // in
   private final static int SCLK_MASK = 0x10; //out
   private final static int SDX_MASK = 0x40; //out same pin as ENABLE_MASK(LCD Class);

   // Diferentes destinos para onde envia tramas
   public static enum Destination {
      DoorMechanism, LCD
   }

   public static void main(String[] args) {
      HAL.init();
      init();
      send(Destination.LCD, 0x41);
   }

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

   // Envia uma trama para o SerialReceiver identificado o destino em addr e os bits de dados em‘data’.
   public static void send(Destination addr, int data) {
      while (isBusy()) ;
      int parity = 0;
      start();
      if (addr == Destination.LCD) {
         ++parity;
         HAL.setBits(SDX_MASK);
      }
      else if (addr == Destination.DoorMechanism)
               HAL.clrBits(SDX_MASK);
      for (int counter = 0; counter <= 4; counter++) {
         HAL.setBits(SCLK_MASK);
         if (data % 2 == 1) { HAL.setBits(SDX_MASK); }
         else {HAL.clrBits(SDX_MASK);}
         HAL.clrBits(SCLK_MASK);
         parity += data % 2;
         data >>= 1;
      }
      HAL.setBits(SCLK_MASK);
      if (parity % 2 == 1) HAL.setBits(SDX_MASK);
      else HAL.clrBits(SDX_MASK);
      HAL.clrBits(SCLK_MASK);
      HAL.setBits(SCLK_MASK);
      HAL.clrBits(SCLK_MASK);
      HAL.setBits(SDX_MASK);
   }
```

```
    private static void start() {
        HAL.clrBits(SDX_MASK);
    }

    // Retorna true se o canal série estiver ocupado
    public static boolean isBusy() {
        return HAL.isBit(BUSY_MASK);
    }


}
```

# L. Código Java da Classe Door

```
public class Door { // Controla o mecanismo da porta.

 // Inicia a classe, estabelecendo os valores iniciais.
    public static void init() {
        HAL.init();
        SerialEmitter.init();
        close(5);
    }

    private static void writeCMD(int speed, boolean open){
        SerialEmitter.send(SerialEmitter.Destination.DoorMechanism,speed<<1|(open?1:0));
    }

    // Envia comando para abrir a porta, indicando a velocidade
    public static void open(int speed) {
        writeCMD(speed,true);
    }

    // Envia comando para fechar a porta, indicando a velocidade
    public static void close(int speed) {
        writeCMD(speed,false);
    }

    // Retorna true se o tiver terminado o comando
    public static boolean isFinished() {
        return !SerialEmitter.isBusy();
    }
}
```

# M. Código Java da Classe M

```
public class M {
    private static final int mMask = 0x20;

    // inicia a classe
    public static void init() {
        HAL.init();
    }

    // Retorna true se o botão estiver ligado
    public static boolean isOn() {
        return HAL.isBit(mMask);
    }
}
```

# N. Código Java da Classe TUI

```
public class TUI { //Text User Interface

   private static final char CLEAR_BUTTON = '*';
   private static final char HASH_BUTTON = '#';

   public static void main(String[] args) {
     init();

      LCD.cursor(0, 0);
      LCD.write("Introduzir codigo");
      System.out.println((readInteger(5000,false, 8, 1, 3)));
      }

   // Escreve uma mensagem no LCD
   public static void writeMessage(String message,int lin , int col){
      LCD.cursor(lin,col);
      LCD.write(message);
   }
   public static void clear(){
      LCD.clear();
   }

   //Retorna o valor dos dígitos introduzidos e escreve-os na posição line e col, em escrita visível ou oculta
   dependendo do boolean hide
   public static int readInteger(int timeout ,boolean hide, int numberOfDigits, int line, int col) {
      int value = 0;
      if (numberOfDigits <= 0) return -1;
      char digit;
      LCD.cursor(line, col);
      for (int i = 0; i < numberOfDigits; i++) LCD.write('?');
      LCD.cursor(line, col);
      for (int i = 0; i < numberOfDigits; i++) {
         digit = 0;
         if ((digit = KBD.waitKey(timeout))==0) {
            return -1;
         }
         if (digit == CLEAR_BUTTON) {
            if (i == 0) return -1;
            else {
               LCD.cursor(line, col);
               for (int j = 0; j < numberOfDigits; j++) LCD.write('?');
               LCD.cursor(line, col);
               i = -1;
               value = 0;
            }
         }
         else if(digit==HASH_BUTTON){
            i--;
         }
         else {
            LCD.write(hide ? '*' : digit);
            int aux = digit - '0';
            int power = numberOfDigits - 1 - i;
```

```
                while (power != 0) {
                    aux *= 10;
                    --power;
                }
                value += aux;
            }
        }
        return value;
    }

    // inicia a classe
    public static void init() {
        HAL.init();
        KBD.init();
        SerialEmitter.init();
        LCD.init();
    }

    // Desliga o LCD
    public static void turnOff(){
        clear();
        LCD.off();
    }

    // Retorna true a tecla premida é igual à passada como parâmetro
    public static boolean isPressed(char c, int time) {
        return KBD.waitKey(time)==c;
    }

    //Retorna o número da coluna onde começa a String passada como parâmetro
    public static int setCol(String name){
        return LCD.getStartingCol(name);
    }

    // Limpa a linha do LCD passada como parâmetro
    public static void clearLine(int line) {
        LCD.clearLine(line);
    }
}
```

# O. Código Java da Classe Users

```
public class Users {

    private static User[] allUsers;
    private static final String USERSFILE = "USERS.txt";
    private static int pinLength = 0;
    private static int uLength = 0;

    public static void init() {
        load();
        FileAccess.init();
    }

    public static void setDimensions(int dim, int pLength, int uL) {
        allUsers = new User[dim];
        pinLength = pLength;
        uLength = uL;

    }

    public static boolean createNewUser(String name, int number, int pin) {
        if (canAdd(number)) {
            allUsers[number] = new User(name, number, pin);
            return true;
        }
        return false;
    }

    public static void addUser(User u) {
        int number = u.getNumber();
        if (canAdd(number)) {
            allUsers[number] = u;
        }
    }

    private static boolean canAdd(int number) {
        return allUsers[number] == null;
    }

    public static boolean UserIsValid(int number, int pin) {
        User u;
        if ((u = allUsers[number]) == null) return false;
        return u.IsPin(pin);
    }

    public static User getUser(int number) {
        return allUsers[number];
    }

    public static long miliToHours(long value) {
        return ((value / 1000) / 60) / 60;
    }
```

```
public static long miliToMinutes(long value) {
        long hours = miliToHours(value);
        hours = hours * 60 * 60 * 1000;
        long a = value - hours;
        return (a / 1000) / 60;
    }

    public static void load() {
        ArrayList<String> al = FileAccess.read(USERSFILE);
        for (String s : al)
            User.fromString(s);
    }


    public static void save() {
        int counter = 0;
        for (int i = 0; i < allUsers.length; i++) {
            if (allUsers[i] != null) {
                User u = allUsers[i];
                String s = u.toString();
                FileAccess.write(USERSFILE, counter++ != 0, s);
            }
        }
    }

    public static void remove(Scanner input) {
        System.out.println("Uin? (0-" + (allUsers.length - 1) + ")");
        int i = input.nextInt();
        if (i >= 0 && i < allUsers.length && allUsers[i] != null) {
            System.out.println(allUsers[i].getName() + " : Do you want to delete this user? (Yes/No)");
            if ((input.next()).toLowerCase().equals("yes")) {
                allUsers[i] = null;
                save();
                System.out.println("User removed.");
            }
        } else System.out.println("This Uin doesn't exist.");
    }

    private static String getFormat(int i) {
        String s = "";
        for (int j = 0; j < i; j++) {
            s += "0";
        }
        return s;
    }

    public static void create(Scanner input) {
        NumberFormat form = new DecimalFormat(getFormat(uLength));
        int number;
        for (number = 0; number < allUsers.length; number++) if (allUsers[number] == null) break;
        System.out.println("Enter User name.");
        input.nextLine();
        String name = input.nextLine();
        System.out.println("Enter a " + pinLength + " digit pin");
        int pin = input.nextInt();
```

```
        if (createNewUser(name, number, pin)) {
            save();
            System.out.println("User number is " + form.format(number));
            System.out.println("New User created.");
        }
    }

    public static void listUsers() {
        DateFormat b = new SimpleDateFormat("HH:mm");
        for (int i = 0; i < allUsers.length; i++) {
            if (allUsers[i] != null && allUsers[i].isIn()) {
                User u = allUsers[i];
                System.out.println(i + " - " + u.getName() + " - " + b.format(u.getIn()));
            }
        }
    }
}
```

# P. Código Java da Classe User

```
public class User {
    private static final int NAMEPOS = 2;
    private static final int NUMBERPOS = 0;
    private static final int PINPOS = 1;
    private static final int ACUMULATEDPOS = 3;
    private static final int INPOS = 4;
    private static final String FILTER_SEPARATOR = ";";
    private static final int NUMOFPARAMETERS = 4;
    private int number;
    private int pin;
    private String name;
    private long in = 0;
    private long acumulated = 0;

    public User(String name, int number, int pin) {
        this.name = name;
        this.number = number;
        this.pin = maskedPin(pin);
    }

    public User(String[] s) {
        this.name = s[NAMEPOS];
        number = Integer.parseInt(s[NUMBERPOS]);
        pin = (Integer.parseInt(s[PINPOS]));
        acumulated = Long.parseLong(s[ACUMULATEDPOS]);
        if (s.length == NUMOFPARAMETERS + 1)
            in = Long.parseLong(s[INPOS]);
        Users.addUser(this);
    }

    public void setPin(int pin) {
        this.pin = maskedPin(pin);
    }

    public void setIn(long in) {
        this.in = in;
    }

    public void setAcumulated(long acumulated) {
        this.acumulated += acumulated;
    }

    public String getName() {
        return name;
    }

    public long getIn() {
        return in;
    }

    public int getNumber() {
        return number;
    }
```

```
    public int getPin() {
        return pin;
    }

    public long getAcumulated() {
        return acumulated;
    }

    public boolean isIn() {
        return in != 0;
    }

    public int maskedPin(int pin) {
        return pin * 5 + 128;
    }

    public boolean IsPin(int pin) {
        return this.pin == maskedPin(pin);
    }

    public String toString() {
        String s = number + FILTER_SEPARATOR + pin + FILTER_SEPARATOR + name +
FILTER_SEPARATOR + acumulated;
        if (isIn()) s+=FILTER_SEPARATOR + getIn();
        return s;
    }

    public static void fromString(String i) {
        String[] s = i.split(FILTER_SEPARATOR);
        User u = new User(s);
    }
}
```

# Q. Código Java da Classe FileAcess

```
public class FileAccess {

   public static void write(String fileName,boolean append,String s){
      try{
         PrintWriter out = new PrintWriter(new FileWriter(fileName,append));
         out.println(s);
         out.close();
      } catch (IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static ArrayList<String> read(String fileName) {
      ArrayList<String> s= new ArrayList<>();
      try {
         Scanner in = new Scanner(new FileReader(fileName));
         while(in.hasNextLine()){
            s.add(in.nextLine());
         }
         in.close();
      } catch (Exception e) {
         System.out.println(e);
      }
      finally {
         return s;
      }
   }

   public static void init() {    }
}
```

# R. Código Java da Classe Log

```
public class Log {
   private static final String LOGFILE = "LOG.txt";
   private Date in;
   private String name;
   private int number;
   private boolean action;

   public Log (Date in, User user, boolean action){
      this.in=in;
      this.name=user.getName();
      this.number=user.getNumber();
      this.action=action;
      FileAccess.write(LOGFILE,true,toString());
   }

   public String toString(){
      DateFormat a = new SimpleDateFormat("dd/MM/yyyy HH:mm ");
      String s=a.format(in);
      s+=((!action)? "-> " : "<- ");
      s+=number+":"+name;
      return s;
   }
}
```

# S. Código Java da Classe App (Working Time Recorder)

```
public class App {

    public static final int UIN_LENGTH = 3;
    public static final int PIN_LENGTH = 4;
    private static final int DOOR_SPEED = 5;
    private static final DateFormat dateAndHour = new SimpleDateFormat("dd/MM/yyyy HH:mm");
    private static final DateFormat dayAndHour = new SimpleDateFormat("EEE. HH:mm", Locale.UK);
    private static final DateFormat hour = new SimpleDateFormat("HH:mm");
    private static final int TIMEOUT = 5000;
    private static Date date = new Date();

    public static void main(String[] args) {
        init();
        LaunchApp();
    }

    private static void LaunchApp() {
        while (true) {
            if (M.isOn()) Maintenance();
            TUI.clear();
            TUI.writeMessage(dateAndHour.format(date), 0, 0);
            boolean cicle=true;
            while (cicle){
                if (M.isOn()){ Maintenance();
                TUI.writeMessage(dateAndHour.format(date), 0, 0);}
                if(dateChanged()) TUI.writeMessage(hour.format(date),0,11);
                TUI.clearLine(1);
                TUI.writeMessage("UIN:", 1, 0);
                int uin = TUI.readInteger(TIMEOUT, false, UIN_LENGTH, 1, 4);
                TUI.clearLine(1);
                TUI.writeMessage("PIN:", 1, 0);
                int pin = -1;
                if (uin != -1)
                    pin = TUI.readInteger(TIMEOUT, true, PIN_LENGTH, 1, 4);
                if (pin != -1 && Users.UserIsValid(uin, pin)) {
                    User user = Users.getUser(uin);
                    TUI.clear();
                    TUI.writeMessage("HELLO", 0, 5);
                    TUI.writeMessage(user.getName(), 1, TUI.setCol(user.getName()));
                    if (TUI.isPressed('#', TIMEOUT)) changePin(user);
                    new Log(date, user, user.isIn());
                    TUI.clear();
                    inOrOut(dayAndHour, date, user);
                    Users.save();
                    cicle=false;
                 }
            }
        }
    }

    private static void inOrOut(DateFormat b, Date date, User user) {
        NumberFormat form = new DecimalFormat("00");
```

```
        if (!user.isIn()) {
            //Entrar No Trabalho
            user.setIn(date.getTime());
            TUI.writeMessage(b.format(date), 0, 0);
            TUI.writeMessage(form.format(Users.miliToHours(user.getAcumulated())) + ":" +
form.format(Users.miliToMinutes(user.getAcumulated())), 0, 11);
            TUI.writeMessage("???. ??:?? ??:??", 1, 0);
        } else {
            //Sair do Trabalho
            user.setAcumulated(date.getTime() - user.getIn());
            TUI.writeMessage(b.format(user.getIn()), 0, 0);
            TUI.writeMessage(b.format(date), 1, 0);
            TUI.writeMessage(form.format(Users.miliToHours(user.getAcumulated())) + ":" +
form.format(Users.miliToMinutes(user.getAcumulated())), 1, 11);
            user.setIn(0);
        }
        Time.sleep(5000);
        doorSequence(user);
    }

    private static void Maintenance() {
        TUI.clear();
        TUI.writeMessage("Out of Service", 0, 1);
        TUI.writeMessage("Wait", 1, 6);
        while (M.isOn()) {
            Scanner input = new Scanner(System.in);
            System.out.println("Write 'Help' for command list");
            System.out.println("Insert command : ");
            String s = input.next();
            if (!exit(s))
                doCommand(s, input);
            else while (M.isOn()) ;
        }
        TUI.clear();
    }

    private static void changePin(User user) {
        TUI.clear();
        int firstEntry = -1;
        int secondEntry = -2;
        TUI.writeMessage("Change Pin ? ", 0, 2);
        TUI.writeMessage("(YES==*)", 1, 3);
        if (!TUI.isPressed('*', TIMEOUT)) return;
        TUI.clear();
        TUI.writeMessage("Insert New ", 0, 3);
        TUI.writeMessage("PIN:", 1, 0);
        firstEntry = TUI.readInteger(TIMEOUT, true, PIN_LENGTH, 1, 4);
        TUI.clear();
        if (firstEntry != -1) {
            TUI.writeMessage("RE-Insert New", 0, 2);
            TUI.writeMessage("PIN:", 1, 0);
            secondEntry = TUI.readInteger(TIMEOUT, true, PIN_LENGTH, 1, 4);
        }
        TUI.clear();
        TUI.writeMessage("PIN has been ", 0, 2);
```

```
        if (firstEntry == secondEntry) {
            TUI.writeMessage("Changed", 1, 5);
            user.setPin(firstEntry);
        } else {
            TUI.writeMessage("Helded", 1, 5);
            Time.sleep(2000);
        }
        Users.save();
        TUI.clear();
    }

    private static void doorSequence(User user) {
        TUI.clear();
        TUI.writeMessage(user.getName() + "", 0, TUI.setCol(user.getName()));
        TUI.writeMessage("Opening Door", 1, 2);
        Time.sleep(1000);
        Door.open(DOOR_SPEED);
        TUI.clear();
        TUI.writeMessage(user.getName() + "", 0, TUI.setCol(user.getName()));
        TUI.writeMessage("Door Opened", 1, 2);
        Time.sleep(1000);
        TUI.clear();
        TUI.writeMessage(user.getName() + "", 0, TUI.setCol(user.getName()));
        TUI.writeMessage("Closing Door...", 1, 1);
        Time.sleep(1000);
        Door.close(DOOR_SPEED);
    }

    private static boolean exit(String s) {
        return s.toLowerCase().equals("exit");
    }

    private static void doCommand(String s, Scanner input) {
        String command = s.toLowerCase();
        if (command.equals("create")) Users.create(input);
        else if (command.equals("delete")) Users.remove(input);
        else if (command.equals("list")) Users.listUsers();
        else if (command.equals("off")) off();
        else if (command.equals("help")) commandList();
        else System.out.println("Invalid command");
    }

    private static void off() {
        TUI.turnOff();
        System.exit(0);
    }

    private static void commandList() {
        System.out.println("The available  commands are : ");
        System.out.println("Create,Delete,List,Exit,OFF");

    }

    private static boolean dateChanged(){
        Date dNew = new Date();
```

```
        if((Users.miliToMinutes(dNew.getTime())-Users.miliToMinutes(date.getTime()))!=0) {
            date = dNew;
            return true;
        }
        return false;
    }
    private static int getDimension(int length) {
        int val = 10;
        for (int i = 1; i < length; ++i) {
            val *= 10;
        }
        return val;
    }

    private static void init() {
        M.init();
        TUI.init();
        Door.init();
        Users.setDimensions(getDimension(UIN_LENGTH), PIN_LENGTH, UIN_LENGTH);
        Users.init();
    }
}
```