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

**Licenciatura em Engenharia Informática e de Computadores**

# Módulo *Keyboard Reader* (*Vending Machine*)

Projeto de
Laboratório de Informática e Computadores
2018 / 2019 inverno

**Autores:**
Diogo Leandro nº44868 LI21D
Paulo Rosa nº44873 LI21D
Tiago Cardoso nº44846 LI21D

# Índice

# 1 Hardware

## 1.1 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.

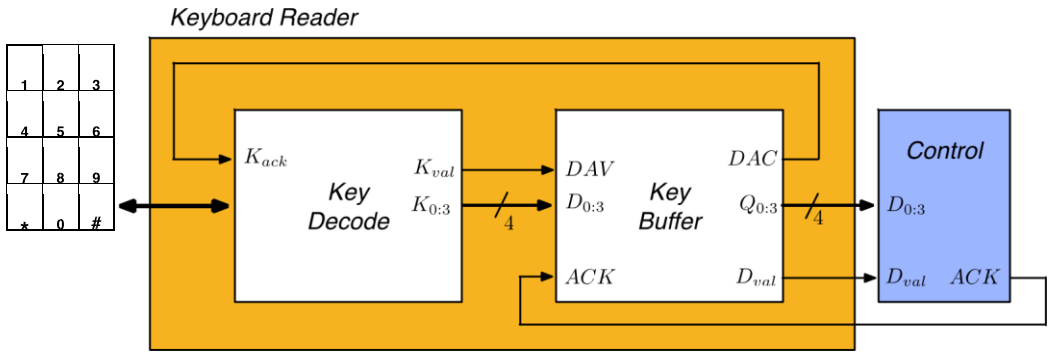
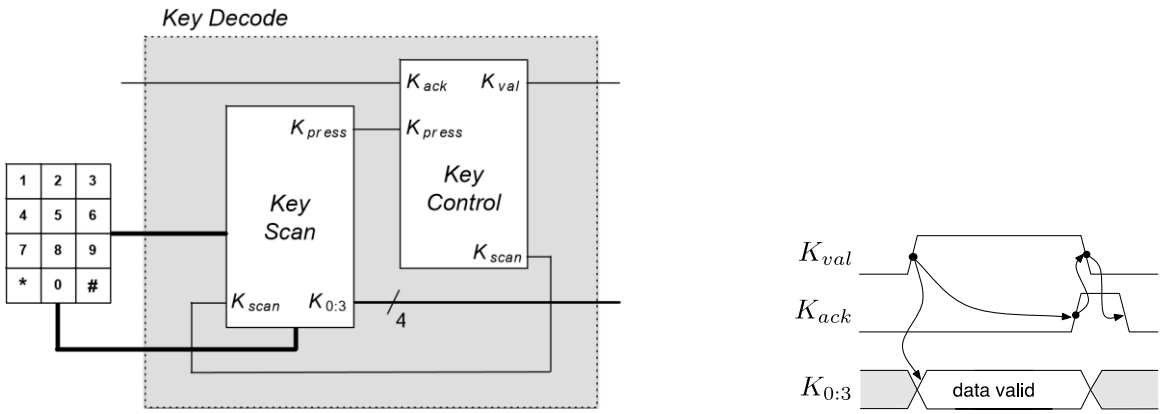



Figura 1 – Diagrama de blocos do módulo *Keyboard Reader*

### *1.1.1 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.

Nesta figura está a versão I do keyscan. Foi escolhida esta versão por ser a mais adequada e mais eficiente para o nosso projeto.

A versão I é a que utiliza o nº de clocks correto e necessário para a execução mais eficiente do projeto.

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

Este *ASM-chart* é responsável pela interação direta com o teclado pois é este que controla quando se deve fazer o varrimento das teclas, e é também este *ASM-chart* que sabe se foi premida uma tecla ou não.

A máquina de estados desenvolvida inicia-se indicando ao contador para fazer o varrimento do teclado (Kscan) esperando que seja devolvida a informação de que foi premida uma tecla (Kpress). Assim que este fenómeno se verifique é enviado um sinal que indica a existência de um novo valor (Kval). Este sinal é continuamente emitido até que o valor da tecla pressionada seja consumido (Kack). Por fim certificamo-nos que só é possivel um novo varrimento de teclado quando Kpress e Kack tiverem o valor lógico 0.

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

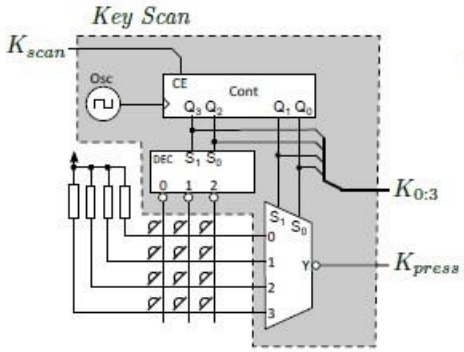



Figura 3 - Diagrama de blocos do bloco *Key Scan*

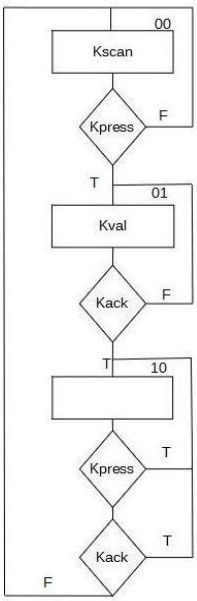



Figura 4 – Máquina de estados do bloco *Key Control*

### *1.1.2 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 tem 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.
Inicia-se com a verificação de informação de uma tecla premida (DAV). Se isto se verificar é enviado um sinal para o Output Register a informar que pode armazenar os dados dessa tecla (Wreg). Imediatamente após este estado comunica-se ao bloco KeyDecode que a informação foi recebida (DAC) e envia-se o sinal que indica ao Control a presença de nova informação disponível (Dval). Estes últimos dois sinais ficam ativos até que o Control indique que a informação foi consumida (ACK). Por fim é necessário verificar se o Control está disponível para receber uma nova tecla, ou seja, que ACK fique com sinal lógico 0.

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

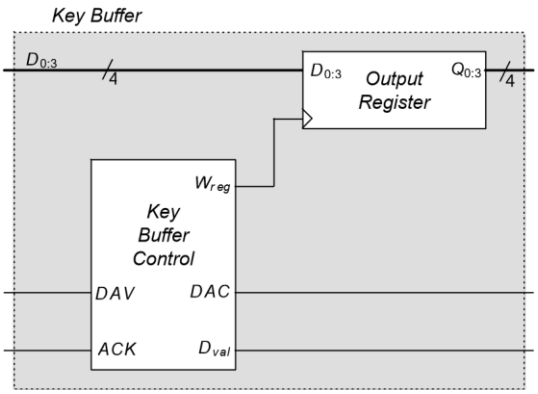



Figura 5 – Diagrama de blocos do bloco *Key Buffer*

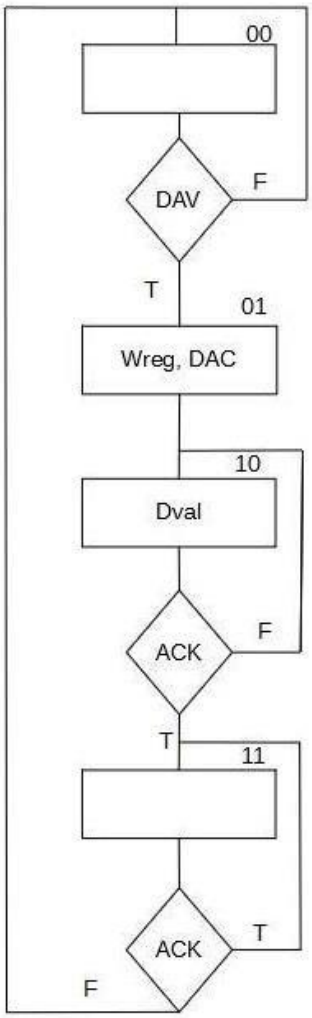



Figura 6 – Máquina de estados do bloco *Key Buffer Control*

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 E.
O valor escolhido das resistências foi de 1,5k ohm devido à corrente máxima da PAL especificada no data sheet. Implementámos dois clocks, o CLK e o MCLK para realizarmos testes, mas devido ao sistema de encadeamento isto nao será necessário. Na execução do projeto tivemos de usar uma frequência baixa para podermos contornar o fenómeno de bounce que acontece quando o utilizador pressiona uma tecla e o teclado recebe ruído aleatório.

## 1.2 Integrated Output System

O módulo Integrated Output System (IOS) implementa a interface com o LCD e com o mecanismo de dispensa, fazendo a receção em série da informação enviada pelo módulo de controlo e entregando-a posteriormente ao destinatário, conforme representado na Figura 7.

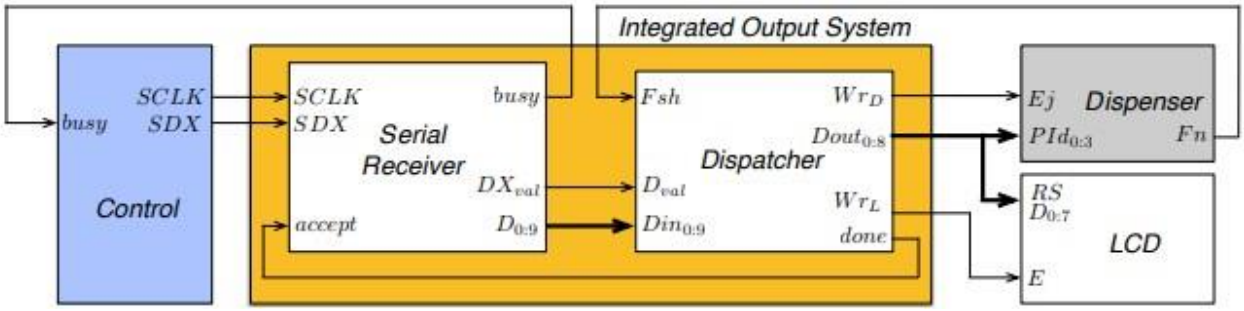



Figura 7 – Diagrama de blocos do *Integrated Output System*

O módulo IOS recebe em série uma mensagem constituída por 5 ou 10 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 e, por consequência, a sua dimensão. 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 8 bits contêm os dados a entregar ao LCD. O último bit contém a informação de paridade ímpar, utilizada para detetar erros de transmissão. As mensagens para o mecanismo de dispensa, ilustradas na Figura 10, contêm para além do bit LnD e do bit paridade mais 4 bits de dados a entregar ao dispositivo, que identificam o produto a dispensar.

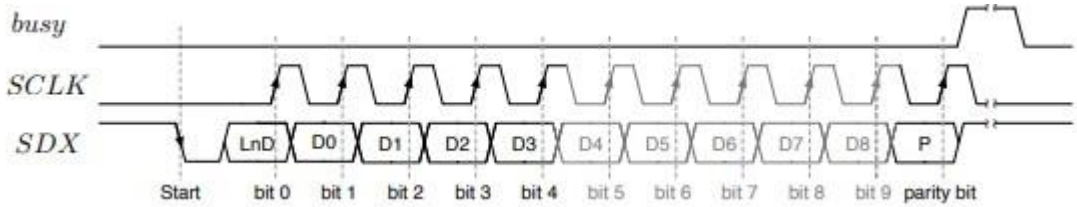



Figura 8 – Protocolo de comunicação com o módulo IOS

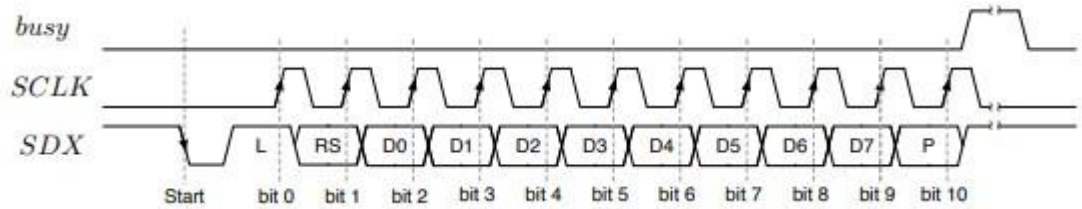



Figura 9 – Trama para o LCD

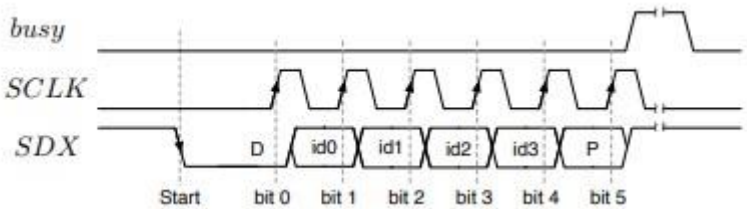



Figura 10 – Trama para o Dispenser

O emissor, realizado em software, quando pretende enviar uma trama para o módulo IOS aguarda que este esteja disponível para receção, ou seja, sinal busy desativo. Em seguida, 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 IOS armazena os bits de dados da trama nas transições ascendentes do sinal SCLK. O sinal busy é ativado, pelo módulo IOS,

quando termina a receção de uma trama válida, ou seja, quando recebe a totalidade dos bits de dados e o bit de paridade correto. O sinal busy é desativado após o Dispatcher informar o IOS que já processou a trama.

### *1.2.1 Serial Receiver*

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

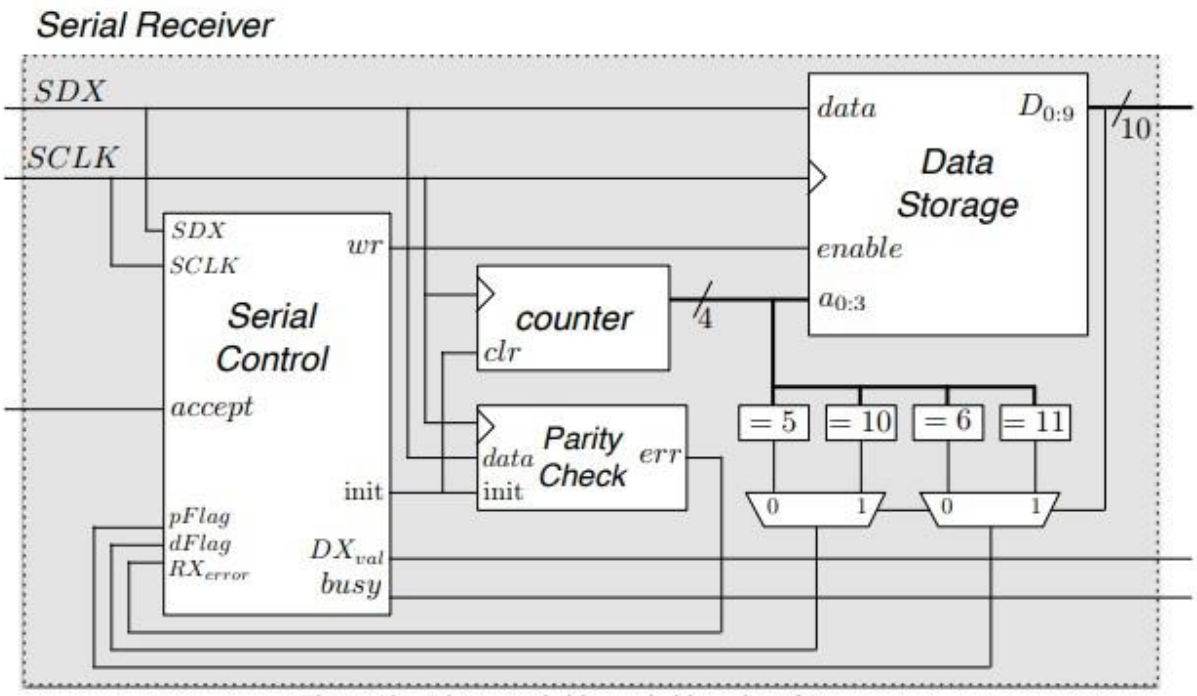



Figura 11 – Diagrama de blocos do Serial Receiver

Dentro do Serial Receiver, temos o Serial Control, que vai gerir o funcionamento dos outros componentes que são: um Counter de 4 bits, um Data Storage, um ParityCheck e sistemas de igualdade (= 5, = 10, = 6 e = 11).

A construção do Counter é idêntica à do KeyScan. O Data Storage, armazena as tramas em Registers (FlipFlops do tipo D) em posições consecutivas com o auxílio do contador. Dos bits do Data Storage, o primeiro bit (o bit 0, LnD) vai ser usado nos Multiplexers 2X1 para ver, consoante o destinatário, quando se quer acabar de ler os bits para esse destinatário e assinalar que está pronto para enviar (DXval, busy), isto usando os pequenos blocos de igualdade.

A máquina de estados Serial Control, implementa uma máquina de estados mais pequena que chamámos de Protocolo de Início de Comunicação (Start). Em que são avaliados os sinais SDX e SCLK, de modo a haver Start segundo a figura 12 do enunciado.

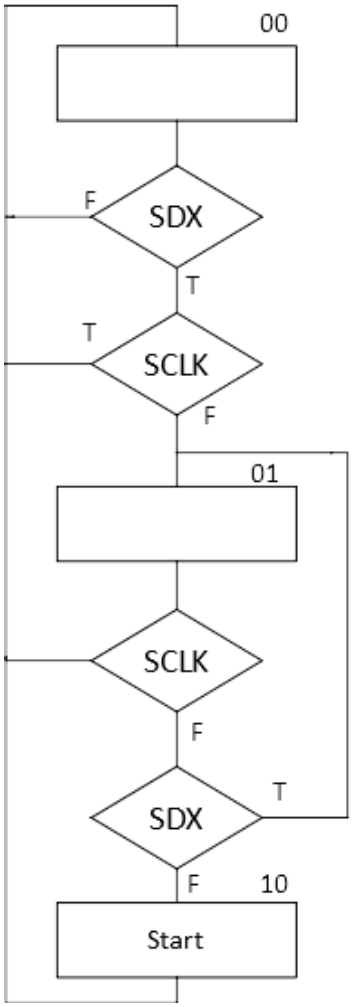



Figura 12 – Máquina de estados do bloco *Serial Receiver* (condição de *Start*)

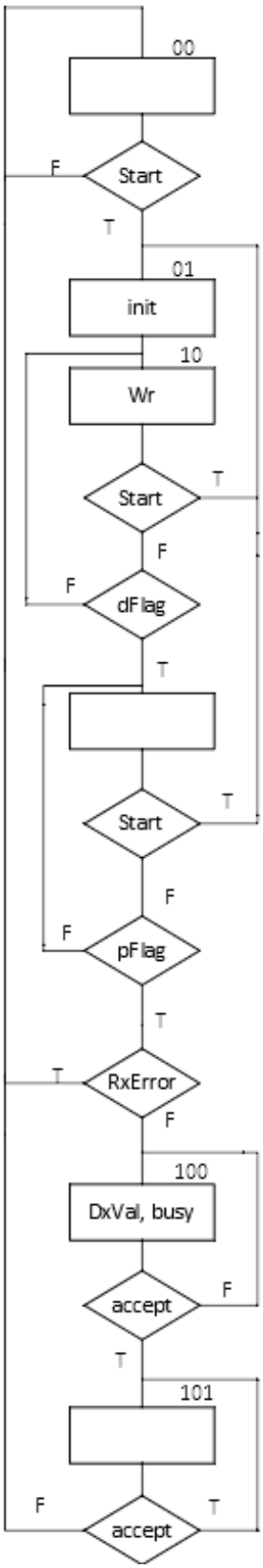



Figura 13 – Máquina de estados do bloco *Serial Receiver*

Portanto, ao haver start, segue-se para o próximo estado que faz init (clear do Counter e ParityCheck). De seguida, damos enable ao Data Storage, verifica-se que não há outro Start que interrompa esta leitura de dados e se já foram lidos os bits necessários para o destinatário, avançamos para o estado seguinte. Verificamos novamente que não há outro Start, e caso já tenha acabado a leitura e não haja sinal de erro, seguimos para o estado 4, que ativa DxVal e Busy e esperamos que o sinal Accept que vem do Done do Dispatcher, tome o valor lógico 1 para indicar que já foram recebidos os dados. E, finalmente, no próximo estado, esperamos que este sinal Accept volte a ficar no valor lógico 0 para voltarmos para o inicio da máquina de estados.

### *1.2.2 Dispatcher*

O bloco Dispatcher é responsável pela entrega das tramas válidas recebidas pelo bloco Serial Receiver ao LCD e ao Dispenser, através da ativação do sinal WrL e WrD. A receção de uma nova trama válida é sinalizada pela ativação do sinal Dval. O processamento das tramas recebidas pelo IOS, para o LCD ou para o Dispenser, deverá respeitar os comandos definidos pelo fabricante de cada periférico, devendo sinalizar o término da execução logo que seja possível ao Serial Receiver.

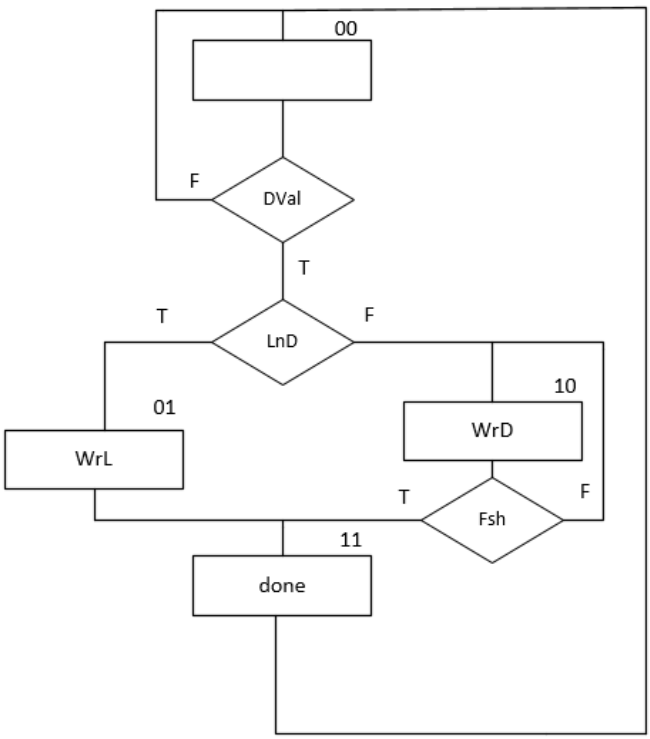



Figura 10 – Máquina de estados do bloco *Dispatcher*

Quanto ao bloco Dispatcher, é uma máquina de estados que se comporta da seguinte forma: Ao haver o sinal DXval, significa que há dados para serem recebidos, e consoante o valor do bit LnD, ou vai para um estado que faz *out* do sinal que efetua *enable* para o LCD ou para o Dispenser. Feito isto, é enviado o sinal Done para o Serial Receiver, no caso dos dados terem sido enviados para o Dispenser.

# 2 Control

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

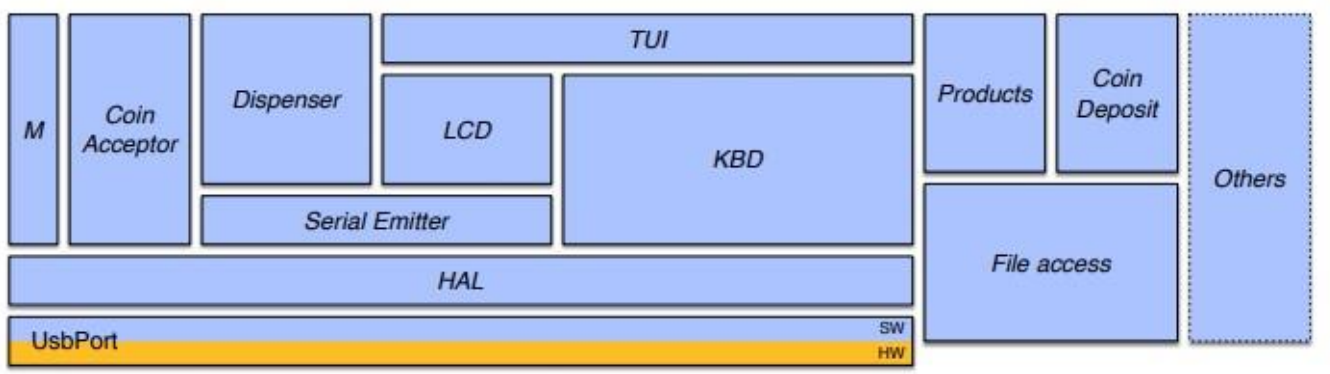



Figura 11 – Diagrama lógico do módulo *Control* de interface com o hardware

## 2.1 Classe *HAL*

Esta classe é responsável pela interação entre o software e o UsbPort, é uma classe genérica que permite às outras classes realizar facilmente funcionalidades de escrita e leitura de bits.

Esta classe dispõe de vários métodos listados no anexo D, sendo estes responsáveis pela leitura, manipulação e escrita da informação dada ao Control.

De uma forma muito sucinta, os métodos desta classe funcionam através de máscaras que sao enviadas como variáveis de entrada destes métodos. Estas máscaras indicam sempre os bits ou bit aos quais pretendemos realizar operações.

Foi necessário criar mais 2 métodos pois a informação que chegava ao Control e a que se apresenta no output estavam negadas. Desta forma, criámos o método in() que é responsável pela leitura dos dados e inversão dos mesmos e o método out() onde são colocados os dados negados e onde é realizado o envio para o output.

## 2.2 Classe *KBD*

A classe KBD tem um propósito fundamental para a arquitetura do sistema. É esta classe que realiza a interação entre o teclado e o software, pois permite ler as teclas '0'..'9','#' e '*' do teclado. E caso não haja tecla, imprime NONE.

De tal modo, temos como parâmetros da classe, um array de caracteres chamado "KEYS" que contém os caracteres das teclas segundo a ordem em que é feito o varrimento no keyscan (de cima para baixo e esqueda para a direita), o caracter NONE e as variavéis keyVal e KeyAck para controlar o ritmo do circuito.

Em primeiro lugar, há método "init" que fundamentalmente faz "reset" de variavéis caso elas tenham outro valor e para permitir um começo organizado para se complementar com o hardware.

Em segundo lugar, o método "getKey" faz return de uma tecla que foi premida ou faz return de NONE caso contrário. Primeiro há um out de keyAck do USBPort que depois entra na PAL para permitir varrimento do teclado; depois é lido o valor ("keyVal") da PAL, que ocorre quando é detetado o uso de uma tecla; de seguida, caso "keyVal" seja verdadeiro, são lidos os 4 bits que indicam a tecla premida, esta tecla é depois retornada. Caso keyVal seja falso, esta variável fica com o valor de NONE;

depois com a expressão HAL.setBits(keyAck), indica-se ao keydecode que se leu a tecla; depois faz-se reset das variáveis deste processo e finalmente é retornada a tecla premida.

Em terceiro lugar, o método waitKey tem as mesmas funções que o getKey, só que fica a tentar ler alguma tecla premida num intervalo de tempo que corresponde ao parâmetro "timeout". Para tal efeito fomos buscar o valor atual do tempo em milissegundos e somamos-lhe o timeout pretendido em milissegundos também, guardando este valor numa variável, de seguida certificamo-nos se essa variável é maior que o tempo atual em milissegundos, enquanto for maior significa que ainda não passou o timeout, logo podemos ir buscar uma tecla premida. Se for premida uma tecla durante este ciclo while esta é retornada.

É importante referir que para garantir o funcionamento do programa tivemos de implementar um sistema de encadeamento que se certifica que certas variáveis apenas mudam o seu valor lógico se outras determinadas variáveis ja o tiverem feito, é o exemplo de Kack que só toma o valor lógico 0 quando sabe que Kval já baixou e ,posteriormente, Kval só volta a tomar o valor lógico 1 quando Kack já baixou. Este sistema foi também implementado em hardware, de modo a termos uma interação software-hardware coerente e um encadeamento entre máquinas.

## 2.3 Classe *LCD*

A classe LCD usa a classe HAL e outros métodos para permitir o uso do LCD(MC1602C-SERIES), tendo em conta a documentação deste para podermos utilizar as suas funcionalidades de leitura e escrita, neste projeto apenas se utilizará a escrita.

Em primeiro lugar, temos de ligar o LCD. E fazemos isso com o método init() e seguindo os comandos e os tempos de atraso em hexadecimal a dar ao LCD segundo a página 3 da sua documentação. Mas ainda antes disso, precisamos de um método para escrever no LCD, que é o writeByte().

Este método é genérico e vai ser chamado pelos outros métodos, recebe como parâmetros uma variável chamada "rs" do tipo boolean e uma do tipo int chamada "data". Quando o rs for verdadeiro, significa que queremos escrever dados para o LCD, quando for falso, significa que queremos dar um input ao LCD e que o interprete como sendo um comando e fazer o que se pretende. O writeByte() utiliza a classe SerialEmitter para enviar os dados, quando chama esta classe tem o cuidado de enviar o rs no inicio da trama de dados.

Em segundo lugar, temos os métodos write(char) e write(String) para escrever texto(dados) no LCD. Sendo que este último chama várias vezes o primeiro com um ciclo for. O write(char) recebe um char que é passado para um valor do tipo int através de um cast e que é usado como parâmetro ao chamar o método writeDATA(), que é o que pretendemos ao escrever texto. Finalmente, temos o método cursor(lin, col) que chama o método writeCMD com o parâmetro (0x40*lin + col | 0x80), porque a última linha começa no endereço 40 (por isso quando lin= 1 o cursor vai para a posição 40, mais a soma da col que se quer, sendo lin a linha e col a coluna) e mais “| 0x80” esta instrução ter o bit de peso 7 a 1 porque a instrução setDDRAM adress necessita do bit DB7 a 1 para isto ser executado. E ainda temos o método clear() que executa a instrução "Clear display" que limpa o ecrã e põe o cursor na posição (0,0).

## 2.4 Classe *SerialEmitter*

O Serial Emitter é a classe que controla os sinais de SCLK e SDX, interagindo desta forma com o bloco Serial Receiver, pois esta é a classe que envia as tramas de dados para o Serial Receiver. Tem como *outputs* o SDX e SCLK e *inputs* o sinal *busy* que vem do Serial Receiver. Tem um enumerado chamado Destination e um parâmetro do tipo inteiro chamado parity, tem também 7 métodos: init(), send(Destination addr, int data), startSignal(), sendLnD(Destination addr), sendData(int nBits, int data), sendParity() e isBusy().

No método init(), são inicializados os *outputs* do USBPort a 0 chamando o HAL.init(). No send(), espera-se caso haja *busy*, no caso de não haver busy, é chamado o método startSignal() que efetua as condições para haver Start no SerialControl e põe o sinal *parity* a 0.

De seguida, é afixada na variável ndataBits a quantidade de bits que se vai trabalhar segundo o destinatário. Isto foi usando o método ordinal() da classe Enum e feito de tal forma que, e como no nosso enumerado, primeiro vem o Dispenser e depois o LCD, se for escolhido o Dispenser, como ele se encontra na posição 0, é efetuado 5*0 +4 = 4, que são os bits precisos para ocorrer comunicação e interpretação de dados para o Dispenser, e caso seja o LCD, o destinatário escolhido, a conta já dará 10 porque 5*1 +4 = 9. Depois é usado o método sendLnD() que faz HAL.write() do bit segundo o valor de lnD..

Depois, temos o método sendData(), que percorre a *data* bit a bit até ao número de bits definido em nBits e envia pelo SDX a informação desse determinado bit. Aqui, também modificamos o valor de *parity* para cada bit de data igual a 1. Depois dos dados, finalmente faz-se sendparity que envia pelo SDX o valor calculado de paridade.

## 2.5 Classe *Coin Acceptor*

Esta classe realiza a interação entre o controlo e o moedeiro, querendo isto dizer que a classe Coin Acceptor realiza a ejeção das moedas, a sua coleção no caso de a compra ser concretizada e guarda o numero de moedas colecionadas durante a compra. Tudo isto é feito através da execução do diagrama temporal do Coin Acceptor.

A classe tem também um método que indica se foi inserida uma nova moeda.

## 2.6 Classe *TUI*

A classe TUI é responsável pela avaliação dos inputs, procura de produtos e representação no LCD.

Nesta classe existem 2 teclas do teclado que tem funções mais especiais. Essas são o '*', que é responsável pela a alteração do modo de procura dos produtos, e o '#' que tem a funcionalidade de confirmação.

Os modos de procura de produtos são dois. O modo definido como padrão é o KeyBoardMode, este recebe valores numéricos de 0-9 do teclado e verifica se o produto nessa posição existe. O outro modo é o ScrollMode em que apenas recebe 4 inputs (2, 8, * e #), sendo que a tecla '2' efetua uma procura do produto seguinte e o 8 efetua uma procura do produto anterior. O # indica que é para fazer a compra do produto indicado no ecrã e durante a compra realiza o cancelamento do mesmo. Melhor descrição em comentário no código fonte desta classe.

## 2.7 Classe *Coin Deposit*

A classe CoinDeposit é responsável pela contagem e atualização das moedas existentes no deposito. Este deposito neste projeto é um ficheiro de texto, CoinDeposit.txt, onde contém o número de moedas que a máquina tem.

## 2.8 Classe *Product*

A classe Product é uma subclasse de Products, e é desta classe que são feitos os objetos do array, "itens" na classe Products, em que são armazenados todos os produtos. O construtor de Product contém o que é preciso para definir um objeto do tipo produto que é: o nome do produto, a sua quantidade, a posição em que vai ser colocado na máquina e o seu preço. A verificação dos limites de posição e quantidade é feita no método addProduct(String name,...float price) na classe Products antes de serem instânciados e adicionados. Logo, aqui não é necessário.

Esta classe contem o método toString(), em que passamos os valores do produto e fazemos "append" do carater ';' para depois guardarmos cada produto com uma estrutura consistente e metódica. E tem getters, setters das características do produto e um método que remove a quantidade do produto.

## 2.9 Classe *Products*

A classe Products tem como parâmetros duas variáveis de tipo inteiro e finais que são o máximo de produtos (16) e o máximo de quantidade permitido para cada produto(20) e uma variável do tipo String e final que é o nome do ficheiro de tipo txt em que vamos armazenar os nossos produtos segundo as propriedades do enunciado.

Nesta classe temos dois métodos chamados getItem que são overloaded que retornam um objeto do tipo Product. Um passando um inteiro como index para o array "items", e outro passando uma string, que é usada para procurar, com um ciclo for, o objeto do tipo Product correspondente ao nome passado. E caso não seja encontrado, é retornado null.

Depois, temos quatro métodos que retornam um booleano. Temos o addProduct, em que se adiciona um produto dado as suas características, e verificando se há espaço para esse. Caso haja, este método adiciona o produto ao arrey e retorna true, caso não haja, não há adição e retorna false.

E o removeProduct faz o contrário, remove e retorna true. Caso o produto não existe, não altera o array items e retorna false. Depois, temos dois métodos (removeItem) e overloaded que removem um item por vez dado um produto, um faz pelo nome do produto(em String) e um faz pela posição do produto. E estes dois últimos comportam-se de forma semelhante como os anteriores em termos de valor de retorno e processo.

E finalmente, temos um método para adicionar um produto e um para ler um produto (usando a classe FileAccess) do ficheiro "Products.txt".

## 2.10 Classe *File Access*

A classe FileAccess tem como função ler ou escrever ficheiros de texto consoante o pretendido. Esta classe é bastante útil para guardarmos informações acerca dos produtos presentes na máquina, tais como a sua identificação, o seu preço e o seu nome, também utilizamos esta classe para guardar o número de moedas presente no moedeiro, o que nos permite saber se temos moedas suficientes para concluir a compra dos produtos, por exemplo.

### 2.11 Classe *Dispenser*

Classe responsável pelo envio do sinal para dispensa de determinado produto consoante o seu identificador.

É constituída por dois sinais, eject e finish, sendo o eject, o que indica que deve ser executada a ejeção do produto e o finish, o sinal que indica o término de execução, esta classe foi realizada seguindo o diagrama temporal do mecanismo de dispensa

### 2.12 Classe *App*

Esta classe é a base de tudo, querendo isto dizer que interage com todas as outras classes direta ou indiretamente. É responsável pelo fluxo normal da máquina de venda, ou seja, é nesta classe que são definidos os processos a serem realizados de modo à máquina saber quando realizar as suas diferentes funções, como por exemplo, quando deve entrar em modo de manutenção, quando deve realizar a compra de um produto. Apesar desta classe saber quando devem ser realizadas os diferentes processos, não é ela que os executa, apenas "manda" as outras classes executarem.

### 2.13 Classe *M*

A classe M é utilizada única e exclusivamente para certificarmos se o utilizador pretende entrar em modo de manutenção, para fazermos essa verificação utilizamos o método da classe HAL denominado isBit.

# 3 Conclusões

Através do desenvolvimento que foi feito até agora deste projeto, pudemos consolidar conhecimentos adquiridos na unidade curricular de Lógica e Sistemas Digitais e na unidade curricular de Programação, tal como aprender a interligar software com hardware através do Kit USBPort que nos foi disponibilisado e da ATB.

Portanto, até este ponto, podemos explorar e desenvolver sistemas em hardware e software construídos com um nível de integridade e complexidade de construção superior, através desta exploração e deste desenvolvimento conseguimos concluir a construção de um sistema (uma máquina de vendas).

Aprendemos também a importância do sistema de encadeamento implementado e os problemas do fenómeno de bounce, que foram resolvidos através da manipulação do valor do clock. Estes problemas do fenómeno de bounce ocorrem devido ao contacto mecânico do teclado que pode receber ruído aleatório durante o toque e fazer com que o valor de Kpress, que verifica se existe toque nas teclas, oscile entre o valor lógico 1 e 0 várias vezes. Como tal, diminuimos o valor da frequência para nos certificarmos que Kpress não oscila tantas vezes. Esta diminuição foi feita através de um divisor de frequência implementado através de flip-flops do tipo t, o divisor de frequência é utilizado para o clock do varrimento e para as máquinas de estados do keyboard reader. Este divisor de frequência serviu para garantir que contornávamos o fenómeno de bounce e garantiu também um teclado mais responsivo ao toque.

Para o funcionamento deste projeto é necessário garantir que o clock do serial receiver é mais rápido que o do keyboard reader.

A latência no nosso projeto é de 12*(1/10) ms que corresponde ao tempo que demora a fazer um varrimento do teclado e a receber a tecla pressionada. Este valor é obtido através do número de teclas que vão ser testadas a multiplicar pelo valor de clock utilizado.

# A. Descrição CUPL do bloco *Key Decode*

```
/* ** INPUT PINS **/

PIN 1 = CLK;

/*MUX*/

PIN [2..5] = [I0..I3];

/* decoder */

PIN 6 = ACK;

/* ** OUTPUT PINS **/


PIN [21..23] = [DEC2..0];
PIN [17..20]= [D0..3];
PIN 16 = Dval;
PIN 15 = Wreg;

/* ** PINNODES **/

PINNODE [27,28] = [KBC0,KBC1]; /*16 e 17*/
PINNODE [29,30] = [KC0,KC1]; /*18 e 19*/
Pin 14 = DCK;

/*Divisor de Clock */
DCK.CK = CLK;
DCK.sp = 'b' 0;
DCK.t = 'b' 1;

/*MUX*/
MUX0 = I0 & (!D1 & !D0);
MUX1 = I1 & (!D1 & D0);
MUX2 = I2 & (D1 & !D0);
MUX3 = I3 & (D1 & D0);

Kpress = !(MUX0 # MUX1 # MUX2 # MUX3);

/*DECODER*/
!DEC0 = !D3 & !D2;
!DEC1 = !D3 & D2;
!DEC2 = D3 & !D2;

/*somador*/
Y = Kscan;
[N0..3] = [D0..3];
```

```
SUM0 = N0 $ Y;
CY0 = N0 & Y;

SUM1 = N1 $ CY0;
CY1 = N1 & CY0;

SUM2 = N2 $ CY1;
CY2 = N2 & CY1;

SUM3 = N3 $ CY2;

/*Register MUX*/
C12 = !N0 & !N1 & N2 & N3;
[M0..3] = [SUM0..3] & !C12;

[D0..3].CK = DCK;
[D0..3].sp = 'b' 0;
[D0..3].D = [M0..3];

/* Key Control  */

[KC0..1].sp = 'b'0;
[KC0..1].ar = 'b'0;
[KC0..1].CK = !DCK;



sequence [KC0,KC1]{
Present 0
    out Kscan;
    if Kpress next 1;
    default next 0;

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

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

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

```
/* ** INPUT PINS **/

PIN 1 = CLK;
PIN [2..5] = [I0..I3];
PIN 6 = ACK;
PIN 7 = MCLK;

/* ** OUTPUT PINS **/



PIN [21..23] = [DEC2..0];
PIN [17..20]= [D0..3];
PIN 16 = Dval;
PIN 15 = Wreg;


/* ** PINNODES **/
```

```
PINNODE [27,28] = [KBC0,KBC1]; /*16 e 17*/
PINNODE [29,30] = [KC0,KC1]; /*18 e 19*/



Kack = DAC;
DAV = Kval;

[KBC0..1].sp = 'b'0;
[KBC0..1].ar = 'b'0;
[KBC0..1].CK = !DCK;

sequence [KBC0, KBC1]{
Present 0
       if DAV next 1;
       default next 0;

Present 1
       out Wreg, DAC;
       next 2;

Present 2
       out Dval;
       if ACK next 3;
       default next 2;

Present 3
       if ACK next 3;
       default next 0;

}
```

# C. Descrição CUPL do bloco *Serial Receiver e Dispatcher*

```
/* *************** INPUT PINS *********************/
PIN  1  = SCLK                      ; /*                                  */
PIN  2  = SDX                       ; /*                              */
PIN  3  = CLK                      ;
PIN  4  = LnD;
PIN  5  = Fsh;


/* *************** OUTPUT PINS *********************/
PIN  [14..17]  = [C0..3];/*PIN[14..17]Q1*/
PINNODE   [25,26]  =  [D0,D1];/*PIN[18,19]Q1*/
PINNODE [27,28]  = [SR0, SR1]; /*Pin 23 Q1&Q0*/
PINNODE    34      = PC;    /*PIN 15 Q1*/

PIN   [18,19]  = [SC1,SC0];
PINNODE 31 = SC2;
PIN  20= Wr;
PIN    21=  WrD            ;
PIN   22  =  WrL ;
PIN   23  =  busy    ;


/*somador*/

[N0..3] = [C0..3];

SUM0 = !N0;
CY0 = N0;
```

```
SUM1 = N1 $ CY0;
CY1 = N1 & CY0;

SUM2 = N2 $ CY1;
CY2 = N2 & CY1;

SUM3 = N3 $ CY2;


/*Register MUX*/

[C0..3].CKMUX = SCLK;
[C0..3].sp = 'b' 0;
[C0..3].D = [SUM0..3];
[C0..3].ar=init;

dFlag = (C0 & !C1 & C2 & !C3 & !LnD) # (!C0 & C1 & !C2 & C3 & LnD);
pFlag = (!C0 & C1 & C2 & !C3 & !LnD) # (C0 & C1 & !C2 & C3 & LnD);

[SR0..SR1].CK = CLK;
[SR0..SR1].sp = 'b' 0;
[SR0..SR1].d = 'b' 0;

PC.CKMUX = SCLK;
PC.sp = 'b' 0;
PC.t = SDX;
PC.ar=init;

RxError = !PC;

Sequence [SR0, SR1]{ /*protocolo de comunicacao*/
Present 0
       if SDX & !SCLK next 1;
       default next 0;

Present 1
       if !SCLK & !SDX next 2;
       if SCLK next 0;
       default next 1;

Present 2
       out Start;
       next 0;
}

accept = done;

/*serial control*/
[SC0..2].CK = !CLK;
[SC0..2].sp = 'b' 0;
[SC0..2].d='b'0;

Sequence [SC0..2]{
Present 0
    if Start next 1;
    default next 0;

Present 1
       out init;
       next 2;

Present 2
    out Wr;
    if Start next 1;
    if !Start & dFlag next 3;
    default next 2;

Present 3
```

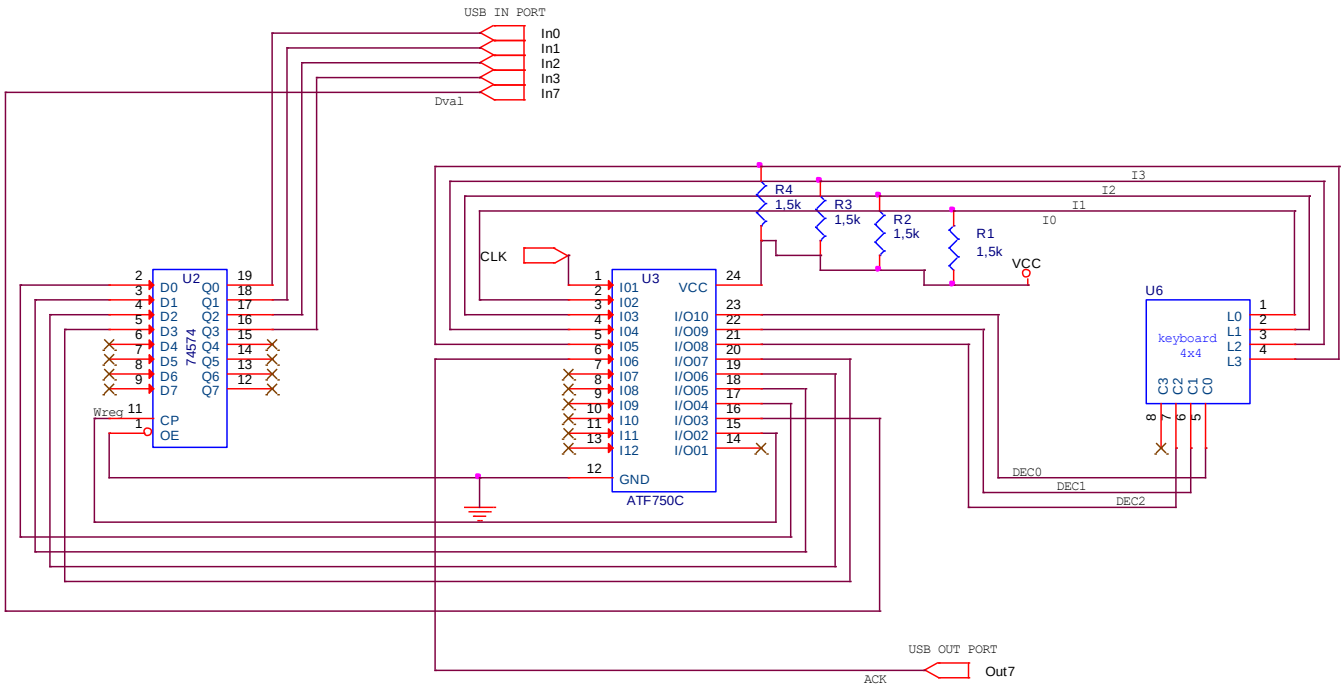
```
       if Start next 1;
    if !Start & !pFlag next 3;
    if !Start & pFlag & !RxError next 4;
    default next 0;

Present 4
    out DxVal, busy;
    if accept next 5;
    default next 4;

Present 5
       if accept next 5;
       default next 0;
}
DVal=DxVal;
/*Dispatcher*/
[D0..1].CK = !CLK;
[D0..1].sp = 'b' 0;
[D0..1].d='b'0;

Sequence [D0..1]{
Present 0
    if DVal & LnD next 1;
    if DVal & !LnD next 2;
    default next 0;

Present 1
       out WrL;
       default next 3;

Present 2
    out WrD;
       if Fsh next 3;
       default next 2;

Present 3
       out done;
       default next 0;

}
```

# D. Descrição CUPL do bloco *Data Storage*

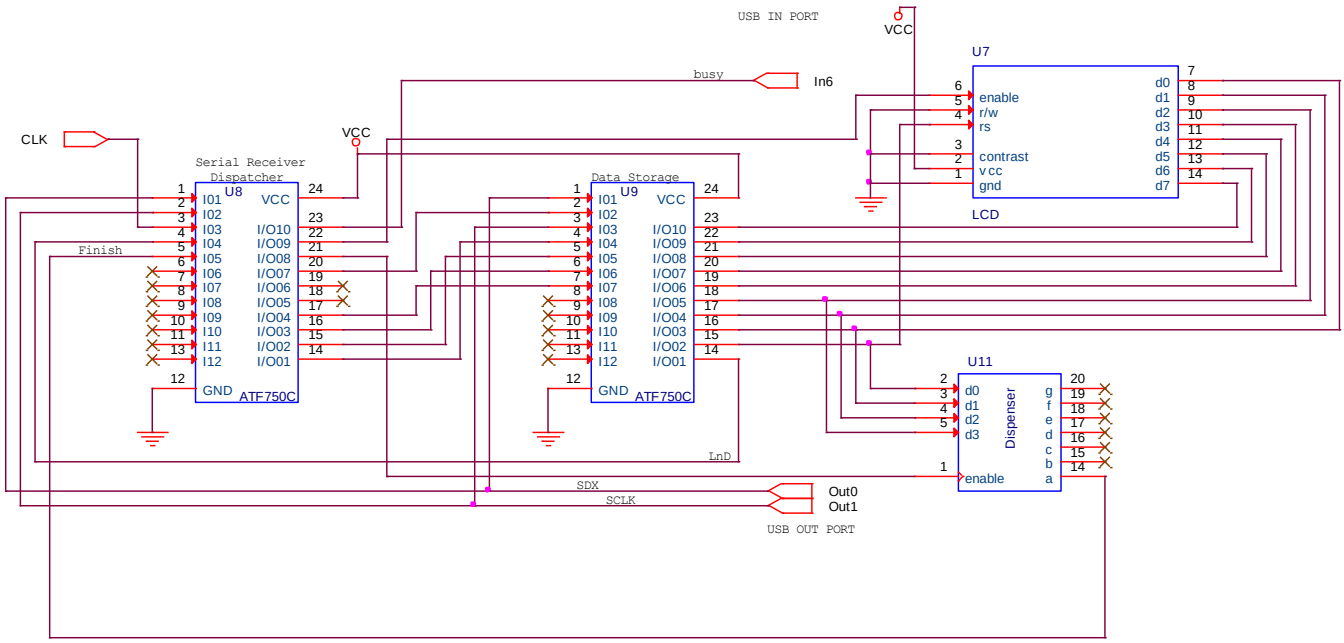
```
/* *************** INPUT PINS *********************/
PIN  1  = SCLK                          ; /*                                  */
PIN  2  = E                          ; /*                                  */
PIN  3  = Data                       ; /*
*/
PIN [4..7] = [A0..3]                          ; /*                                  */

/* *************** OUTPUT PINS *********************/
PIN   [14..23]  = [D0..9]    ; /*                                */

/*     Decoder*/

Dc0 = (!A0 & !A1 & !A2 & !A3) ;
Dc1 = (A0 & !A1 & !A2 & !A3) ;
Dc2 = (!A0 & A1 & !A2 & !A3) ;
Dc3 = (A0 & A1 & !A2 & !A3) ;
Dc4 = (!A0 & !A1 & A2 & !A3) ;
Dc5 = (A0 & !A1 & A2 & !A3) ;
Dc6 = (!A0 & A1 & A2 & !A3);
```

```
Dc7 = (A0 & A1 & A2 & !A3);
Dc8 = (!A0 & !A1 & !A2 & A3);
Dc9 = (A0 & !A1 & !A2 & A3);
[V0..9]=[D0..9];
[D0..9].D = Data & [Dc0..9] & E # ![Dc0..9] & [V0..9];
[D0..9].CKMUX = SCLK;
[D0..9].SP = 'b'0;
```

## E. Esquema elétrico do módulo *Keyboard Reader*




## F. Esquema elétrico do módulo *Integrated Output System*




## G. Código Java da classe *HAL*

```
import isel.leic.UsbPort;
public class HAL {
private static int out;
public static void init(){
out = 0
out();
   }
   public static void setBits(int mask){ // Coloca os bits representados por mask no valor lógico ‘1’
out |=mask;         out();
   }
   public static void clrBits(int mask){ // Coloca os bits representados por mask no valor lógico ‘0’
out &=~mask;
       out();
   }

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

```
    private static void out(){
        UsbPort.out(~out);
    }
}
```

## H. Código Java da classe *KBD*

```
import isel.leic.utils.Time;

public class KBD {
    // Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.
    public static final char NONE = '.';
    private static final char[] KEYS = {'1', '4', '7', '*', '2', '5', '8', '0', '3', '6', '9', '#'};//hardware
    private static final int DVAL_PIN = 1 << 7;
    private static final int ACK_PIN = 1 << 7;
    private static final int DATA_PINS = 0xf;

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

    }

    // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
    public static char getKey() {
        char c = NONE;
        if (HAL.isBit(DVAL_PIN)) {
            int charPos = HAL.readBits(DATA_PINS);
            c = KEYS[charPos];
            HAL.setBits(ACK_PIN);
            while (HAL.isBit(DVAL_PIN)) ;
        }
        init();
        return c;
```

```
        }
        // Retorna quando a tecla for premida ou NONE após decorrido 'timeout' milisegundos.

        public static char waitKey(long timeout) {
            long i = Time.getTimeInMillis() + timeout;
            while (i > Time.getTimeInMillis()) {
                char c = getKey();
                if (c != NONE) return c;
            }
            return NONE;

        }
    }
```

# I. Código Java da classe *LCD*

```
import isel.leic.utils.Time;

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

    // Escreve um byte de comando/dados no LCD
    private static void writeByte(boolean rs, int data) {
        Time.sleep(4);
        SerialEmitter.send(SerialEmitter.Destination.LCD, (data << 1) | (rs ? 1 : 0));
    }

    // 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() {
   Time.sleep(50);
   writeCMD(0x30);
   Time.sleep(5);
   writeCMD(0x30);
   Time.sleep(1);
   writeCMD(0x30);
   writeCMD(0x38);
   writeCMD(0x8);
   writeCMD(0x1);
   writeCMD(0x6);
   writeCMD(0xF);

}

// 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) {
        writeCMD(((0x40 * lin) + col) | 0x80);

    }

    // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
    public static void clear() {
        writeCMD(0x1);

    }
}
```

## J. Código Java da classe *SerialEmitter*

```
public class SerialEmitter {
    private static final int SDX_PIN = 0b1, SCLK_PIN = 0b10, BUSY_SIGNAL = 0b1000000;
    private static int parity;

    // Inicia a classe
    public static void init() {
        HAL.init();
        parity = 0;
    }

    // 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) {
        int lnd = addr.ordinal();
        while (isBusy()) ;

        startSignal();
        sendLnD(lnd);

        int nDataBits = lnd * 5 + 4;//algoritmo para calcular o numero de bits a enviar dependendo do destinatario
        sendData(nDataBits, data);

        sendParity();

    }

    // Envia o sinal de iniciação de envio de trama
    private static void startSignal() {
```

```
        HAL.clrBits(SCLK_PIN);
        HAL.setBits(SDX_PIN);
        HAL.clrBits(SDX_PIN);
        parity = 0;
    }

    //Envia o bit LnD
    private static void sendLnD(int lnd) {
        HAL.writeBits(SDX_PIN, lnd);
        HAL.setBits(SCLK_PIN);

        parity += lnd;
    }

    private static void sendData(int nBits, int data) {
        for (int i = 0; i < nBits; i++) {
            int val = (data >> i) & 1;
            HAL.writeBits(SDX_PIN, val);
            HAL.clrBits(SCLK_PIN);
            HAL.setBits(SCLK_PIN);

            parity += val;
        }
    }

    private static void sendParity() {
        HAL.writeBits(SDX_PIN, ~parity);
        HAL.clrBits(SCLK_PIN);

        HAL.setBits(SCLK_PIN);
        HAL.setBits(SDX_PIN);
        HAL.clrBits(SCLK_PIN);
    }

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

    // Envia tramas para os diferentes módulos Serial Receiver.
    public enum Destination {
        Dispenser, LCD
    }
}
```

## K. Código Java da classe *TUI*

```
public class TUI {

    public static final int SCROLL_MODE = 1;

    public static final int KEYBOARD_MODE = 0;

    public static int indexProduct;
```

```
public static int searchMode;
private static boolean off;
private static Product onDisplay;

/**
 * Method responsible for the initiation of the classes needed.
 * This method must be used before using any of the other methods of this class
 */
public static void init() {
    indexProduct = 0;
    off = false;
    onDisplay = null;
    searchMode = KEYBOARD_MODE;
}

/**
 * Display in the LCD the initial screen information
 */
public static void startScreen() {
    onDisplay = null;
    searchMode = KEYBOARD_MODE;
    clearAllScreen();
    showLeft("Vending Machine", 0);
    indexProduct = 0;
}


/**
 * Verifies if the key pressed is the confirmation
 *
```

```
     * @param key code pressed in the keyboard
     * @return {@code boolean} that indicates if the key pressed is #
     */
    public static boolean isConfirmation(char key) {
        return key == '#';
    }

    /**
     * Changes the value of indexProduct for the next of previous product depending of code putted in the keyboard, if this one is valid.
     * input 2 -gives the next product
     * input 8 -gives the previous product
     * input * our # gets out of scroll mode
     *
     * @param key key pressed in the keyboard
     * @return {@code boolean} that inform if still inside the scroll mode
     */
    public static boolean scrollMode(char key) {
        if (Character.isDigit(key)) {
            switch (key) {
                case '2':
                    indexProduct = searchNextProduct(indexProduct);
                    break;
                case '8':
                    indexProduct = searchPreviousProduct(indexProduct);
                    break;
            }
            productDisplay(indexProduct, SCROLL_MODE);
            return true;
        }
```

```
    return false;
}

/**
 * Changes the value of indexProduct for the code putted in the keyboard, if this one is valid.
 *
 * @param key key pressed in the keyboard
 * @return {@code boolean} that inform if still inside the keyboard mode
 */
public static boolean keyBoardMode(char key) {
    if (Character.isDigit(key)) {
        int value = Character.digit(key, 10);
        int tenValue = 10 * indexProduct + value;
       /* System.out.println("value: "+Integer.toString(value));
        System.out.println("tenValue: "+Integer.toString(tenValue));*/
        if (verifyIndex(value)) {
            indexProduct = value;
        }
        if (verifyIndex(tenValue)) {
            indexProduct = tenValue;
        }
        if (!verifyIndex(indexProduct)) {
            indexProduct = searchNextProduct(indexProduct);
        }
        productDisplay(indexProduct, KEYBOARD_MODE);
        return true;
    }
    return false;
}
```

```
public static boolean verifyIndex(int value) {
   if (value < Product.MAX_PRODUCTS) {

      if (value == 0)
         return true;//case product 0 is unavailable is need to reach values with number 0
      else
         for (int i = value; i != 0; i /= 10)
            if (i % 10 == 1)
               return true;//case product 1 is unavailable is need to reach values with number 1

      Product p = Product.getItem(value);
      return p != null && p.getQuantity() > 0 && p.getQuantity() < Product.MAX_QUANTITY;
   }
   return false;
}

/**
 * Displays the product information in LCD.
 *
 * @param index of the product to be displayed
 * @param mode
 */
public static void productDisplay(int index, int mode) {
   Product product = Product.getItem(index);
   productDisplay(product, mode);
}

public static void productDisplay(Product product, int mode) {
   if (product != null)
      if (product != onDisplay) {
```

```
            refreshScreen(product, mode);
            onDisplay = product;
        } else if (searchMode != mode) {
            showProductId(mode);
            searchMode = mode;
        }
}

/**
 * Responsible for the update of the information on the screen
 *
 * @param product product that should be displaying on the screen
 * @param mode    of search of the product (KEYBOARD_MODE or SCROLL_MODE)
 */
public static void refreshScreen(Product product, int mode) {
    clearAllScreen();
    showCenter(product.getPRODUCT(), 0);
    int quantity = product.getQuantity();
    if (quantity >= 1 && quantity <= Product.MAX_QUANTITY)
        showCenter("#" + String.format("%02d", quantity), 1);
    else showCenter("#--", 1);
    int price = product.getPrice();
    showRight(String.format("%.1f", price / 10f), 1);
    showProductId(mode);
}

/**
 * Show the product location in the mode passed as param
 *
 * @param mode of apresentation of the location of the prodiuct
```

```
 */
public static void showProductId(int mode) {
   showLeft(String.format("%02d", indexProduct) + (mode == TUI.KEYBOARD_MODE ? ":" : "*"), 1);
}

/**
 * Display a message on LCD with the alignment on the center.
 *
 * @param txt  Text to show on LCD. User has to make sure the text can fit in the LCD line.
 * @param line Which line the text is supposed to appear.Lines start at 0.
 *             User has to make sure that the number passed is coherent with the LCD.
 */
public static void showCenter(String txt, int line) {
   int center = (LCD.COLS - txt.length()) / 2;
   if (center >= 0 && line < LCD.LINES) {
      LCD.cursor(line, center);
      LCD.write(txt);
   }
}

/**
 * Display a message on LCD with the alignment on the right.
 *
 * @param txt  Text to show on LCD. User has to make sure the text can fit in the LCD line.
 * @param line Which line the text is supposed to appear.Lines start at 0.
 *             User has to make sure that the number passed is coherent with the LCD.
 */
public static void showRight(String txt, int line) {
   int left = LCD.COLS - txt.length();
   if (left >= 0 && line < LCD.LINES) {
```

```
        LCD.cursor(line, left);
        LCD.write(txt);
    }
}

/**
 * Display a message on LCD with the alignment on the left
 *
 * @param txt  Text to show on LCD. User has to make sure the text can fit in the LCD line.
 * @param line Which line the text is supposed to appear.Lines start at 0.
 *           User has to make sure that the number passed is coherent with the LCD.
 */
public static void showLeft(String txt, int line) {
    LCD.cursor(line, 0);
    LCD.write(txt);
}

/**
 * Search for a new product starting the searched in product above the actual one
 *
 * @param index the number where the product is located in the machine.
 * @return the product that is before the actual one.
 * Case doesn't exist returns the product with the code passed as param.
 */
public static int searchNextProduct(int index) {
    Product p;
    int start = index;
    do {
        index = Math.floorMod(index + 1, Product.MAX_PRODUCTS);
        p = Product.getItem(index);
```

```
        } while (p == null || p.getQuantity() <= 0 || p.getQuantity() > Product.MAX_QUANTITY || start == index);

        return index;
    }

    /**
     * Search for a new product starting the searched in product bellow the actual one
     *
     * @param indexProduct the code where the product is located in the machine.
     * @return the product that is before the actual one.
     * Case doesn't exist returns the product with the code passed as param.
     */
    public static int searchPreviousProduct(int indexProduct) {
        Product p;
        int start = indexProduct;
        do {
            indexProduct = Math.floorMod(indexProduct - 1, Product.MAX_PRODUCTS);
            p = Product.getItem(indexProduct);
        } while (p == null || p.getQuantity() <= 0 || p.getQuantity() > Product.MAX_QUANTITY || start ==
indexProduct);

        return indexProduct;
    }

    /**
     * Clears all information displayed in LCD screen
     */
    public static void clearAllScreen() {
        LCD.clear();
    }
```

```
/**
 * Indicates system status(on or off)
 *
 * @return boolean off
 **/
public static boolean isOff() {
   return off;
}

public static void collectScreen() {
   clearAllScreen();
   showCenter("Collect Product", 1);
}

/**
 * Display of when the machines turns off
 */
public static void turnOff() {
   off = true;
   clearAllScreen();
   showCenter("IS", 0);
   showCenter("OFF", 1);
}


public static Product getProductOnDisplay() {
   return onDisplay;
}
```

```
    /**
     * Start screen display when maintenance mode is on
     */
    public static void maintStartScreen() {
        clearAllScreen();
        showLeft("Maintenance Mode", 0);
        showLeft("1-Ld 2-Rm 3-Off", 1);
        onDisplay = null;
    }
}
```

## L. Código Java da classe *CoinAcceptor*

```
import isel.leic.utils.Time;

public class CoinAcceptor { // Implementa a interface com o moedeiro.
    private static final int COIN_SIGNAL = 1 << 5;
    private static final int ACCEPT_SIGNAL = 0b10000;
    private static final int COLLECT_SIGNAL = 0b100000;
    private static final int EJECT_SIGNAL = 0b1000000;

    public static int coins;
    public static int coinsStored;

    public static void init() {
        coins = 0;
    }

    // Retorna true se foi introduzida uma nova moeda.
    public static boolean hasCoin() {
        return HAL.isBit(COIN_SIGNAL);
    }

    // Informa o moedeiro que a moeda foi aceite.
    public static void acceptCoin() {
        while (hasCoin()) HAL.setBits(ACCEPT_SIGNAL);
        coins += 1;
        HAL.clrBits(ACCEPT_SIGNAL);
    }

    // Devolve as moedas que estão no moedeiro.
    public static void ejectCoins() {
        HAL.setBits(EJECT_SIGNAL);
        coins = 0;
```

```
        Time.sleep(1000);
        HAL.clrBits(EJECT_SIGNAL);
    }

    // Recolhe as moedas que estão no moedeiro.
    public static void collectCoin() {
        HAL.setBits(COLLECT_SIGNAL);
        coinsStored = coins;
        Time.sleep(1000);
        HAL.clrBits(COLLECT_SIGNAL);
        coins = 0;
    }
}
```

## M. Código Java da classe CoinDeposit

```
public class CoinDeposit {
    private static final String FILENAME = "CoinDeposit.txt";
    private static int coinsAvailable;

    /**
     * Method responsible for initiation of the classes needed.
     * This method must be used before using any of the other methods of this class
     */
    public static void init() {
        coinsAvailable = 0;
    }

    /**
     * Inserts a coin in the coin deposit
     */
    public static void insertedCoins(int coins) {
        coinsAvailable += coins;
    }


    /**
     * Writes on the file FILENAME the number of coins
     */
    public static void updateFile() {
        FileAccess.writeFile(FILENAME, Integer.toString(coinsAvailable));
    }

    /**
     * Reads from the file FILENAME the number os coins and stores in coinsAvailable
     */
    public static void updateCoins() {
        String coins = FileAccess.readFile(FILENAME);
        if (coins != null)
            if (coins.endsWith(";"))
                coinsAvailable = Integer.valueOf(coins.split(";")[0]);
    }
}
```

## N. Código Java da classe Dispenser

```
import isel.leic.utils.Time;

public class Dispenser { // Controla o estado do mecanismo de dispensa.

   private static final int EJECT = 0b10000;
   private static final int FINISH = 0b1000;

   // Inicia a classe, estabelecendo os valores iniciais.
   public static void init() {
      HAL.clrBits(EJECT);

   }
   // Envia comando para dispensar uma unidade de um produto
   public static void dispense(int productId) {
      SerialEmitter.send(SerialEmitter.Destination.Dispenser, productId);
      while (!HAL.isBit(FINISH))
         HAL.setBits(EJECT);
      Time.sleep(1000);
      HAL.clrBits(EJECT);
   }
}
```

## O. Código Java da classe *FileAccess*

```
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;


public class FileAccess {


   public static void writeFile(String FILENAME, String info) {
      try (BufferedWriter bw = new BufferedWriter(new FileWriter(FILENAME))) {
         bw.write(info);
      } catch (IOException e) {
         e.printStackTrace();
      }
```

```
        }

        public static String readFile(String FILENAME) {
            try (Scanner in = new Scanner(new FileReader(FILENAME))) {
                StringBuilder s = new StringBuilder();
                while (in.hasNext())
                    s.append(in.next());
                return s.toString();
            } catch (IOException e) {
                e.printStackTrace();
            }
            return null;
        }
    }
```

## P. Código Java da classe Product

```
public class Product extends Products {
    private final String PRODUCT;
    private int quantity;
    private int position;
    //prince in cents
    private int price;

    Product(int position, String name, int quantity, int price) {
        this.quantity = quantity;
        PRODUCT = name;
        this.position = position;
        this.price = price;
    }

    @Override
    public String toString() {
        return Integer.valueOf(position).toString() + ';' +
                PRODUCT + ';' +
                Integer.valueOf(quantity).toString() + ';' +
                Integer.valueOf(price).toString();
    }

    public int getPrice() {
        return price;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int value) {
        quantity = value;
    }

    public String getPRODUCT() {
        return PRODUCT;
```

```
    }

    public void removeQuantity(int number) {
        quantity -= number;
    }
}
```

## Q. Código Java da classe *Products*

```
import java.io.FileReader;
import java.util.Scanner;


public class Products {
    public static final int MAX_PRODUCTS = 16, MAX_QUANTITY = 20;
    private static final String FILENAME = "Products.txt";
    //Represents the stock of products
    private static Product[] items = new Product[MAX_PRODUCTS];

    /**
     * Returns the product located in the location passed as param
     *
     * @param index position of the product
     * @return the product located in the position index
     */
    public static Product getItem(int index) {
        return items[index];
    }


    /**
     * Adds one product to the system if possible
     *
     * @param name     name of the product
     * @param quantity how many products of this type exist
     * @param position code that represents where is located this product
     * @param price    price of the product
     * @return if the product was added
     */
    public static void addProduct(int position, String name, int quantity, int price) {
        if (items.length <= MAX_PRODUCTS)
            if (quantity <= MAX_QUANTITY)
                if (position < items.length)
                    items[position] = new Product(position, name, quantity, price);
    }
```

```
/**
 * Removes one product to the system if possible
 *
 * @param product that will be removed
 * @return if the product was removed
 */
public static boolean removeProduct(Product product) {
    for (int i = 0; i < items.length; i++) {
        if (items[i] == product) {
            items[i] = null;
            return true;
        }
    }
    return false;
}

/**
 * Removes on item from the product of the indicated name
 *
 * @param item of the product
 * @return if the item was removed or not
 */
public static boolean removeItem(Product item) {
    for (Product p : items) {
        if (p == item)
            if (p.getQuantity() > 0) {
                p.removeQuantity(1);
                return true;
            }
    }
    return false;
}

/**
 * Removes on item from the product of the indicated position
 *
 * @param position code where the product is located in the machine
 * @return if the item was removed or not
 */
public static boolean removeItem(int position) {
    Product p = items[position];
    if (p != null)
        if (p.getQuantity() > 0) {
            p.removeQuantity(1);
            return true;
        }
    return false;
}

/**
 * Writes in the file FILENAME all the products
 */
```

```
    public static void writeProducts() {
        StringBuilder s = new StringBuilder();
        for (Product p : items)
            if (p != null)
                s.append(p.toString()).append("\n");
        FileAccess.writeFile(FILENAME, s.toString());

    }

    /**
     * Reads all products of the file FILENAME
     */
    public static void readProducts() {
        try (Scanner in = new Scanner(new FileReader(FILENAME))) {
            String line;
            while (in.hasNextLine()) {
                String[] info;
                line = in.nextLine();
                info = line.split(";");
                Product.addProduct(Integer.parseInt(info[0]), info[1],
                        Integer.parseInt(info[2]), Integer.parseInt(info[3]));
            }

        } catch (Exception e) {
            System.out.println("Load Error");
            e.printStackTrace();
        }
    }
}
```

## R. Código Java da classe M

```
public class M {


    public static boolean isPressed() {
        return HAL.isBit(0b10000);
    }
}
```

# S. Código Java da classe App

```
import isel.leic.utils.Time;

import java.util.Calendar;
import java.util.TimeZone;

public class MyApp {
   //is the wait time until gets back to the start screen
   private static final int TIMEOUT = 10 * 1000;
   //information wich search mode is being used
   private static boolean isKeyBoardMode;

   //Initialization of all the classes needed in this project
   private static void init() {
      HAL.init();
      KBD.init();
      LCD.init();
      TUI.init();
      CoinDeposit.init();
      CoinAcceptor.init();
      Dispenser.init();
      isKeyBoardMode = true;
   }

   /**
    * updates the classes with the information of their files
    */
   private static void load() {
      Product.readProducts();
      CoinDeposit.updateCoins();
   }
```

```
public static void main(String[] args) {
   init();
   load();
   while (!TUI.isOff()) {//verifies if the machine is still on
      start();
      if (M.isPressed()) {
         maintenance();
         continue;
      }
      run();
   }
   save();
}

/**
 * Responsible for the search of the product and display that is selected in the moment
 *
 * @return they last key pressed
 */
private static char searchProduct() {
   char key;
   key = isKeyBoardMode ? keyBoardSearch() : scrollSearch();
   switch (key) {
      case '#':
      case KBD.NONE:
         return key;
      case '*':
         isKeyBoardMode = !isKeyBoardMode;
         if (isKeyBoardMode)
            TUI.productDisplay(TUI.indexProduct, TUI.KEYBOARD_MODE);
         else
            TUI.productDisplay(TUI.indexProduct, TUI.KEYBOARD_MODE);
   }
```

```
    return key;
}

/**
 * Executes the search for a product in scroll mode as well the display of the product selected
 *
 * @return they las key pressed
 */
private static char scrollSearch() {
    char key = '0';
    while (TUI.scrollMode(key)) key = KBD.waitKey(TIMEOUT);
    return key;
}

/**
 * Executes the search for a product in keyboard mode as well the display of the product selected
 *
 * @return the last key pressed
 */
private static char keyBoardSearch() {
    char key = (char) (TUI.indexProduct % 10 + 48);//%10 is in case of number with 2 digits gets only the unity
    while (TUI.keyBoardMode(key)) key = KBD.waitKey(TIMEOUT);
    return key;
}

/**
 * routine that executes when maintenance signal is active
 */
private static void maintenance() {
    TUI.maintStartScreen();
    while (M.isPressed())
        switch (KBD.getKey()) {
            case '1':
```

```
            maintAdd();
            TUI.maintStartScreen();
            break;
         case '2':
            maintRem();
            TUI.maintStartScreen();
            break;
         case '3':
            if (confChoice("ShutDown")) {
               TUI.turnOff();
               return;
            } else TUI.maintStartScreen();
      }
}

/**
 * Routine used to remove an item from the machine
 */
private static void maintRem() {
   char key;
   firstDisplay();
   //search for the product
   //Case the input takes more then TIMEOUT milliseconds machines goes back to the start mode
   while (!TUI.isConfirmation(key = searchProduct())) {
      if (key == KBD.NONE) return;
   }
   //p holds the information of the product that is on display
   Product p = TUI.getProductOnDisplay();
   if (confChoice("Remove " + p.getPRODUCT()))
      if (Product.removeProduct(p)) {
         TUI.clearAllScreen();
         TUI.showCenter("Product", 0);
         TUI.showCenter("Removed", 1);
```

```
            Time.sleep(1000);
            TUI.clearAllScreen();
        }
    }

    /**
     * Routine used to add a new quantity to an item from the machine
     */
    private static void maintAdd() {
        char key;
        //search for the product
        //Case the input takes more then TIMEOUT milliseconds machines goes back to the start mode
        firstDisplay();
        while (!TUI.isConfirmation(key = searchProduct())) {
            if (key == KBD.NONE)
                return;
        }
        int value = getQuantity();
        String s = value + " " + TUI.getProductOnDisplay().getPRODUCT();
        if (confChoice(s))
            TUI.getProductOnDisplay().setQuantity(value);//sets the new quantity of the product that is in display
    }

    /**
     * Waits until a valid quantity is inserted
     *
     * @return the quantity inserted
     */
    private static int getQuantity() {
        int value;
        do {
            TUI.showCenter("#--", 1);
            int q1 = getNumInput();
```

```
        TUI.showCenter("#-" + q1, 1);
        int q0 = getNumInput();
        TUI.showCenter(String.format("#%d%d", q1, q0), 1);
        value = q1 * 10 + q0;
    } while (value >= Product.MAX_QUANTITY);
    return value;
}

/**
 * Waits until it get a numerical input
 *
 * @return the number of the input
 */
private static int getNumInput() {
    int q;
    do {
        q = KBD.getKey();
    }
    while (!Character.isDigit(q));
    return Character.digit(q, 10);
}

/**
 * Prompts the message of confirmation of the action and waits for the choice
 *
 * @param choice is the message to be displayed on the prompt
 * @return the choice of the user
 */
private static boolean confChoice(String choice) {
    TUI.clearAllScreen();
    TUI.showCenter(choice, 0);
    TUI.showCenter("Yes-5   other-No", 1);
```

```
    char c;
    for (c = KBD.getKey(); c == KBD.NONE; c = KBD.getKey()) ;
    return c == '5';
}

private static void start() {
    TUI.startScreen();
    boolean wait = false;//variable used so the screen doesn't refresh more then one time per second
    Calendar time = Calendar.getInstance(TimeZone.getTimeZone("UTC"));
    showCurrTime(time);
    //cicle that will get the current time while is the start screen
    for (; !TUI.isConfirmation(KBD.getKey()) && !M.isPressed(); time =
        Calendar.getInstance(TimeZone.getTimeZone("UTC"))) {
      //when passes one second the hours in the screen will be updated with actual time
      if (time.get(Calendar.SECOND) == 0) {//passa quando os segundos terem valor 0
        if (!wait) {
          //when passes one day the date in the screen will be updated with actual day
          if (time.get(Calendar.HOUR) == 0)
            showCurrDate(time);
          showCurrHours(time);
          wait = true;
        }
      } else wait = false;
    }
}

/**
 * Show on the display the date and the hours
 *
 * @param time holds the information of the day selected
 */
private static void showCurrTime(Calendar time) {
    showCurrDate(time);
```

```
    showCurrHours(time);
}

//Shows on display the hours
private static void showCurrHours(Calendar time) {
    TUI.showRight(String.format("%02d:%02d", time.get(Calendar.HOUR_OF_DAY), time.get(Calendar.MINUTE)),
        1);
}

//Shows on display the date
private static void showCurrDate(Calendar time) {
    TUI.showLeft(String.format("%02d/%02d/%02d", time.get(Calendar.DAY_OF_MONTH),
        time.get(Calendar.MONTH) + 1, time.get(Calendar.YEAR) % 100), 1);
}

/**
 * Saves all the information in the respective files
 */
private static void save() {
    Products.writeProducts();
    System.out.println("coins Stored" + CoinAcceptor.coinsStored);
    CoinDeposit.insertedCoins(CoinAcceptor.coinsStored);
    CoinDeposit.updateFile();
}

private static void run() {
    char key;//holds the information of the key pressed
    firstDisplay();
    while ((key = searchProduct()) != KBD.NONE)
        //checks if is to execute the purchase of the product
        if (TUI.isConfirmation(key)) {
            if (purchase(TUI.getProductOnDisplay())) {
                productCollection();
```

```
            updateFiles();
        } else if (CoinAcceptor.coins > 0) cancel();
        return;
      }
   System.out.println("end");
}

/**
 * this method guaranties that indexProduct variable holds a valid information and displays it
 */
private static void firstDisplay() {
   if (Product.getItem(TUI.indexProduct) == null)
      TUI.indexProduct = TUI.searchNextProduct(TUI.indexProduct);
   TUI.productDisplay(TUI.indexProduct, TUI.KEYBOARD_MODE);
}

/**
 * Is called when the purchased was canceled
 */
private static void cancel() {
   TUI.clearAllScreen();
   TUI.showCenter("Returning", 0);
   TUI.showCenter(CoinAcceptor.coins + " coins", 1);
   CoinAcceptor.ejectCoins();
}

/**
 * gets all the information from the text files to the respective classes
 */
private static void updateFiles() {
   CoinDeposit.insertedCoins(CoinAcceptor.coinsStored);
   CoinDeposit.updateFile();
   Products.writeProducts();
```

```
}

/**
 * Shows to the costumer when to get the product and waits until the collection of the same
 * showing the greetings in the end.
 */
private static void productCollection() {
   TUI.clearAllScreen();
   TUI.showCenter("Collect", 0);
   TUI.showCenter("Product", 1);
   Dispenser.dispense(TUI.indexProduct);
   Products.removeItem(TUI.indexProduct);
   TUI.clearAllScreen();
   TUI.showCenter("Thank you", 0);
   TUI.showCenter("for buying", 1);
}

/**
 * Executes the purchase routine if the product given is valid
 *
 * @param p is the product that will be bought
 * @return whether the purchase was successful or not
 */
private static boolean purchase(Product p) {
   if (p != null && p.getQuantity() > 0) {
      TUI.clearAllScreen();
      TUI.showCenter(p.getPRODUCT(), 0);
      TUI.showLeft("Insert", 1);
      int price = p.getPrice();
      TUI.showCenter(String.format("%02d", price), 1);
      if (checkBuy(price)) {
         CoinAcceptor.collectCoin();
         return true;
```

```
            }
        }
        return false;
    }

    /**
     * Verifies if the coins inserted are enough to buy the product
     *
     * @param price of the product
     * @return true if all coins where inserted
     */
    private static boolean checkBuy(int price) {
        do {
            if (TUI.isConfirmation(KBD.getKey())) return false;
            if (CoinAcceptor.hasCoin()) {
                CoinAcceptor.acceptCoin();
                TUI.showCenter(String.format("%02d", price - CoinAcceptor.coins), 1);
            }
        } while (CoinAcceptor.coins < price);
        return true;
    }
}
```