ISEL
DEETC
Departamento de Engenharia Electrónica e de Telecomunicações e de Computadores

Licenciatura em Engenharia Informática e de Computadores

Máquina de venda
(*Vending Machine*)

Projeto
de
Laboratório de Informática e Computadores
2021 / 2022 inverno

publicado: 06 de outubro de 2021

1 Descrição

Pretende-se implementar o sistema de controlo de uma máquina de venda (*Vending Machine*), onde são armazenados diferentes tipos de produtos. A máquina permite armazenar até 16 tipos de produtos e no máximo 20 produtos de cada tipo. A seleção do produto é realizada digitando o identificador do produto ou através das teclas ↑ e ↓, sendo exibido num ecrã o identificador do produto, o nome, a quantidade existente e o preço. A ordem de dispensa é dada através da pressão da tecla de confirmação, sendo dispensada uma unidade do produto exibido no ecrã.

Para além do modo de Dispensa, o sistema tem mais um modo de funcionamento designado por Manutenção, que é ativado por uma chave de manutenção. Este modo permite a gestão dos produtos disponíveis na máquina, selecionando-se através do teclado, o tipo de produto que se pretende visualizar, carregar ou anular.

O sistema de controlo da máquina de venda é constituído por um teclado de 12 teclas, um moedeiro (designado por *Coin Acceptor*), um ecrã *Liquid Cristal Display* (LCD) de duas linhas de 16 caracteres, um mecanismo de dispensa de produtos (designado por *Dispenser*) e uma chave de manutenção (designada por *M*) que define se o dispensador está em modo de Manutenção, conforme o diagrama de blocos apresentado na Figura 1.

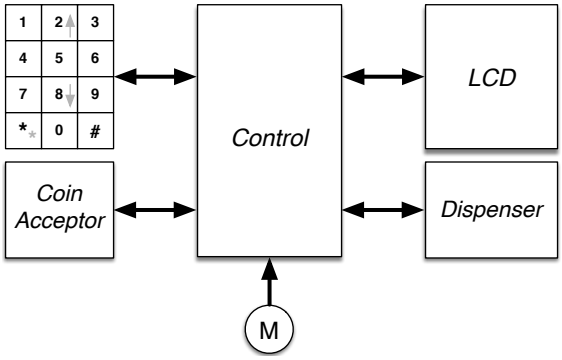


Figura 1 – Diagrama de blocos do sistema de controlo da máquina de venda (*Vending Machine*)

Sobre o sistema podem-se realizar as seguintes ações em modo Dispensa:

- **Consulta e dispensa** – A consulta de um produto, é realizada digitando o identificador do produto ou listando os produtos através das teclas ↑ e ↓, a dispensa deste é realizada após confirmação dada pela tecla '#' e a inserção do respetivo valor monetário. Durante a dispensa ficam afixados no LCD as informações referentes ao produto até que o mecanismo de dispensa confirme que a dispensa já foi efetuada. O modo de seleção ↑ e ↓ alterna com a seleção numérica por pressão da tecla '*'.

Sobre o sistema podem realizar-se as seguintes ações no modo Manutenção:

- **Carregamento** – Para cada produto é possível estabelecer qual a quantidade existente. O carregamento de cada produto é iniciado, pela seleção da opção no menu, seguido da seleção do identificador do produto e da quantidade. Após inserir o identificador prime-se a tecla de confirmação '#', para de seguida inserir a quantidade seguida da tecla '#' para concretizar o carregamento do produto.
- **Anulação** – Para anular um tipo de produto seleciona-se a operação de anulação no menu, seguindo-se a seleção do identificador de produto finalizando com a tecla de confirmação '#'.
- **Desligar** – O sistema desliga-se ao selecionar-se esta opção no menu, ou seja, o software de gestão termina armazenando as estruturas de dados de forma persistente em ficheiros de texto. O ficheiro contendo a informação dos produtos, deve estar organizado com uma linha por cada produto, em que os campos de dados são separados por ";", com o formato "*SLOT;NAME;STOCK;PRICE*" que é carregado no início do programa e reescrito no final do programa.

Nota: A inserção de informação através do teclado tem o seguinte critério: *i*) se não for premida nenhuma tecla num intervalo de cinco segundos o comando em curso é abortado; *ii*) quando o dado a introduzir é composto por mais que um dígito, são considerados apenas os últimos dígitos, a inserção realiza-se do dígito de maior peso para o de menor peso.

2 Arquitetura do sistema

O sistema será implementado numa solução híbrida de hardware e software, como apresentado no diagrama de blocos da Figura 2. A arquitetura proposta é constituída por três módulos principais: *i*) um leitor de teclado, designado por *Keyboard Reader*; *ii*) um módulo de interface com o LCD e com o mecanismo de dispensa, designado por *Integrated Output System* (*IOS*); e *iii*) um módulo de controlo, designado por *Control*. Os módulos *i*) e *ii*) deverão ser implementados em *hardware*, enquanto o módulo de controlo deverá ser implementado em *software* usando linguagem *Kotlin* executado num PC.

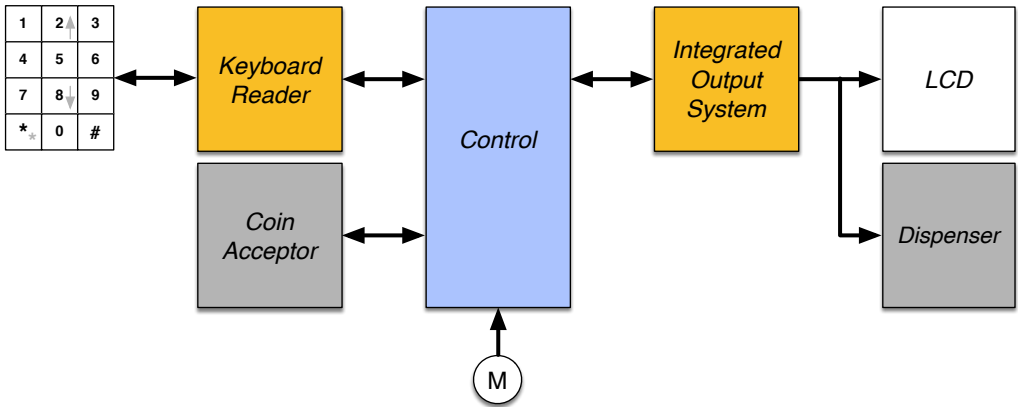


Figura 2 – Arquitetura do sistema que implementa a máquina de venda (*Working Time Recorder*)

O módulo *Keyboard Reader* é responsável pela descodificação do teclado matricial de 12 teclas, determinando qual a tecla pressionada e disponibilizando o seu código, com quatro bits, ao módulo *Control*. Caso este não esteja disponível para o receber imediatamente, o código da tecla é armazenado até ao limite de dois códigos. O módulo *Control* processa os dados e envia a informação a apresentar no *LCD* através do módulo *IOS*. O mecanismo de dispensa, designado por *Dispenser*, é atuado pelo módulo *Control*, através do módulo *IOS*. Por razões de ordem física, e por forma a minimizar o número de fios de interligação, a comunicação entre o módulo *Control* e o módulo *IOS* é realizada recorrendo a um protocolo série.

2.1 Keyboard Reader

O módulo *Keyboard Reader* 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 3. Neste caso o módulo de controlo, implementado em *software*, é a entidade consumidora.

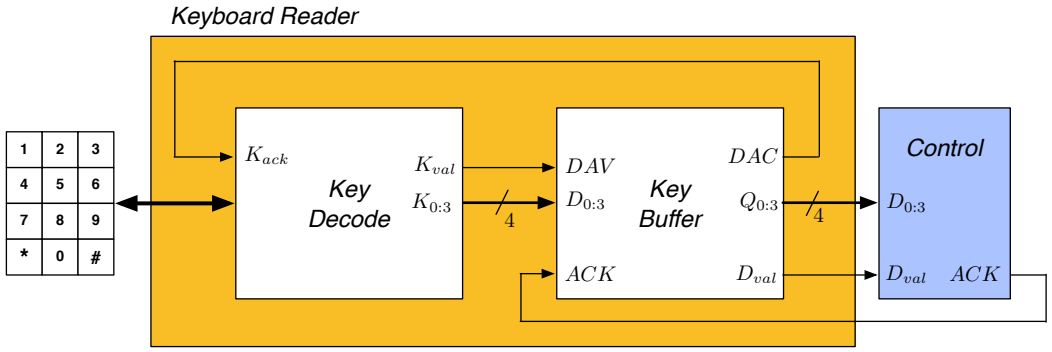
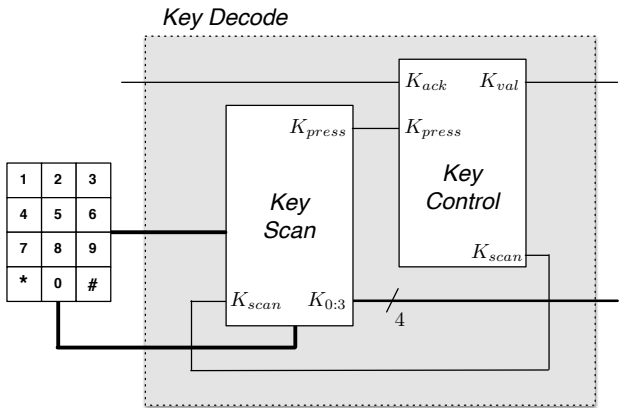


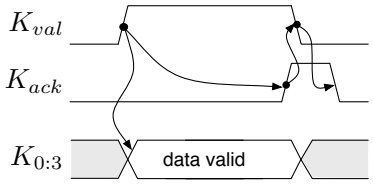
Figura 3 – Diagrama de blocos do módulo *Keyboard Reader*

2.1.1 Key Decode

O bloco *Key Decode* deverá implementar 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 4a.

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 4b.

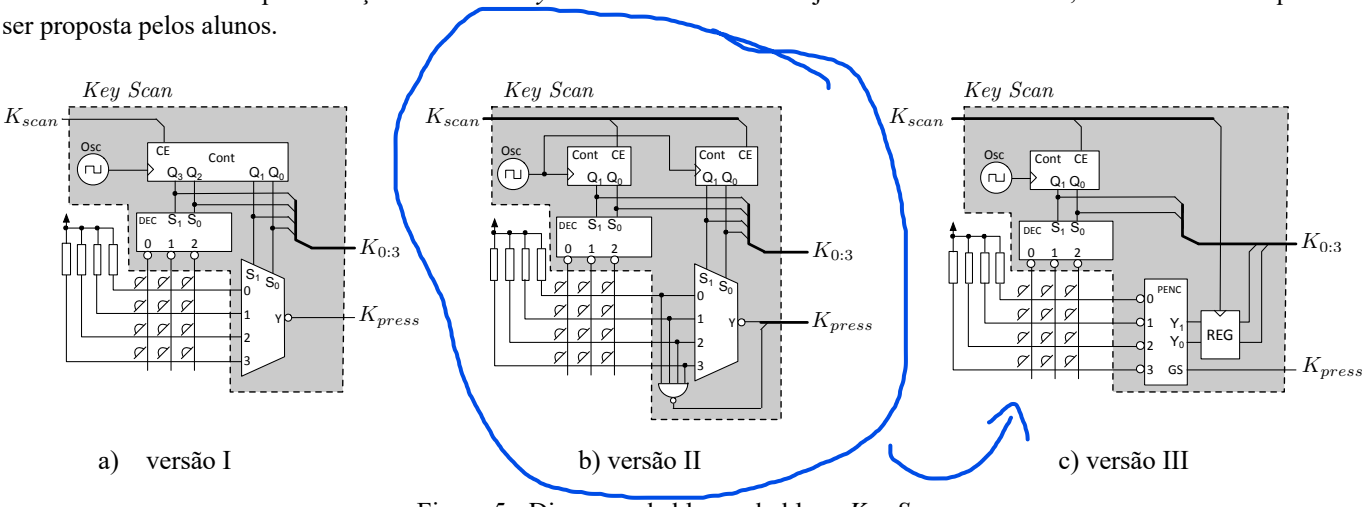




a) Diagrama de blocos b) Diagrama temporal

Figura 4 – Bloco *Key Decode*

O bloco *Key Scan* deverá ser implementado de acordo com um dos diagramas de blocos representados na Figura 5, enquanto o desenvolvimento e a implementação do bloco *Key Control* ficam como objeto de análise e estudo, devendo a sua arquitetura ser proposta pelos alunos.



a) versão I b) versão II c) versão III

Figura 5 - Diagrama de blocos do bloco *Key Scan*

2.1.2 Key Buffer

Coisa linda né

O bloco *Key Buffer* a desenvolver corresponderá a 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 6, 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* deverá ser baseada numa máquina de controlo (*Key Buffer Control*) e num registo (*Output Register*).

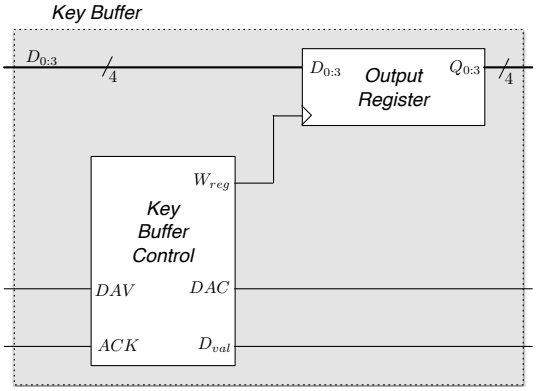


Figura 6 – Diagrama de blocos do bloco *Key Buffer*

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* deve invalidar os dados baixando o sinal D_{val}. Para que uma nova palavra possa ser armazenada será necessário que o módulo *Control* tenha desativado o sinal *ACK*.

2.2 Coin Acceptor

Este módulo implementa a interface com o moedeiro, sinalizando que este recebeu uma moeda através da ativação do sinal *Coin*. A entidade consumidora informa o *Coin Acceptor* que já contabilizou a moeda através da ativação do sinal *accept*, conforme apresentado no diagrama temporal da Figura 7. O *Control* pode devolver as moedas inseridas no moedeiro através da ativação do sinal *eject* durante um segundo, ou recolher as moedas presentes no moedeiro através da ativação do sinal *collect* durante um segundo.

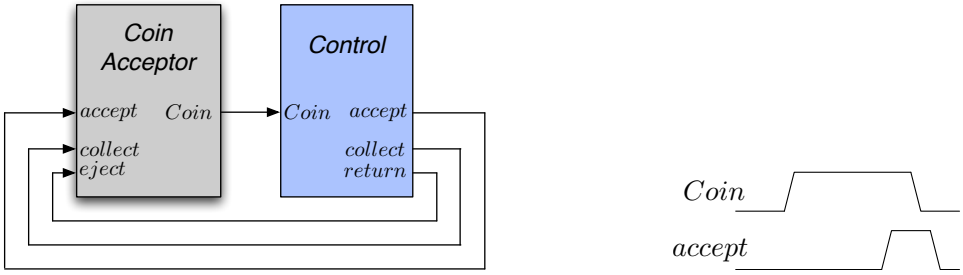


Figura 7 – Diagrama de blocos e diagrama temporal do *Coin Acceptor*

2.3 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 8.

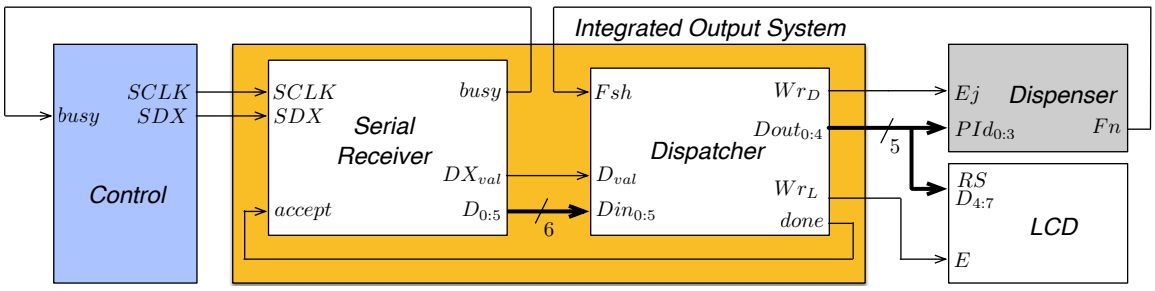


Figura 8 – Diagrama de blocos do *Integrated Output System*

O módulo *IOS* recebe em série uma mensagem constituída por 5 ou 6 bits de informação e um bit de paridade. A comunicação com este módulo realiza-se segundo o protocolo ilustrado na Figura 9, em que o bit *LnD* identifica o destinatário da mensagem e, por consequência, a sua dimensão. Nas mensagens para o LCD, ilustrado na Figura 10, o bit *RS* é o primeiro bit de informação e indica se a mensagem é de controlo ou dados. Os seguintes 5 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 11, 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.

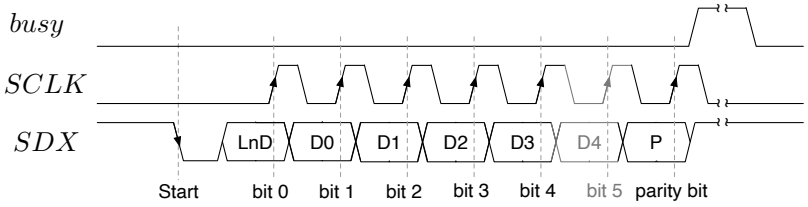


Figura 9 – Protocolo de comunicação com o módulo *Integrated Output System*

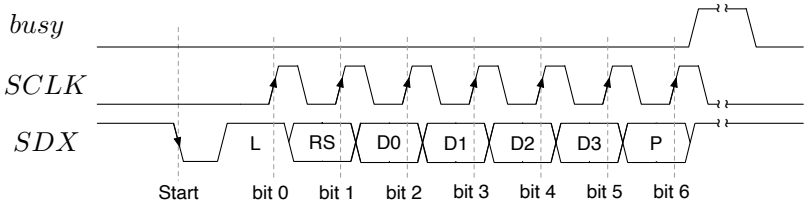


Figura 10 – Trama para o *LCD*

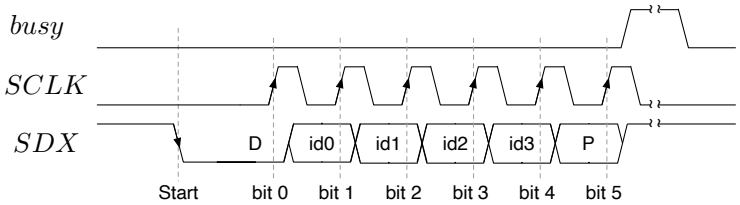


Figura 11 – Trama para o mecanismo de dispensa (*Dispenser)*

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* assinalar que processou a trama, ativando o sinal *done*.

2.3.1 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 12.

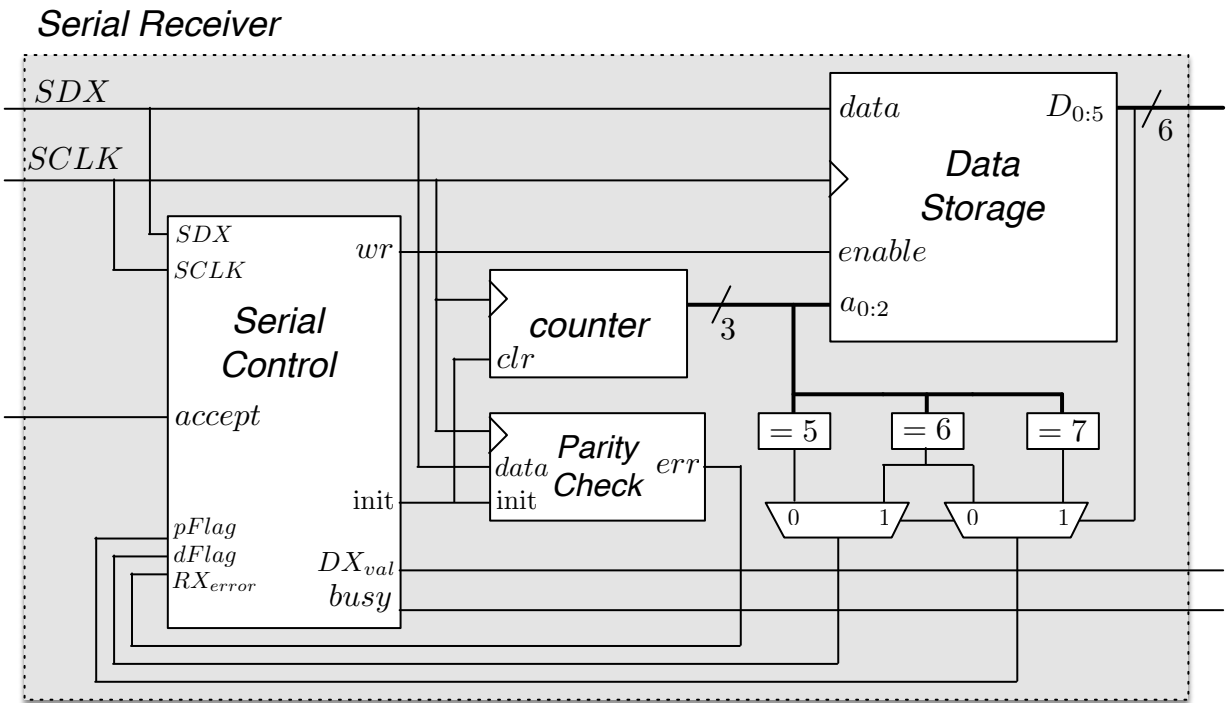


Figura 12 – Diagrama de blocos do bloco *Serial Receiver*

2.3.2 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 Wr_L e Wr_D. A receção de uma nova trama válida é sinalizada pela ativação do sinal D_{val}.
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*.

2.3.3 Dispenser

O *Dispenser* recebe em quatro bits o código do produto ($PId_{0:3}$) a dispensar. O comando de ejeção de um produto com o código presente em *PId* é realizado pela ativação do sinal de ejeção (*Ej*). Em resposta, o *Dispenser* ativa o sinal de término de execução (*Fn*) quando concluída a dispensa. Os sinais *Fn* e *Ej* têm o comportamento descrito no diagrama temporal apresentado na Figura 13.

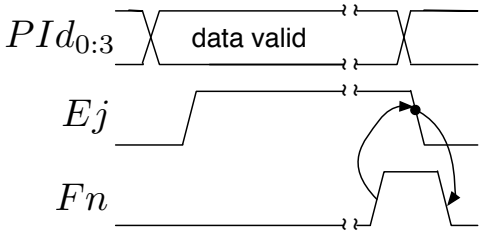


Figura 13- Diagrama temporal do mecanismo de dispensa

2.4 *Control*

A implementação do módulo *Control* deverá ser realizada em *software*, usando a linguagem *Kotlin* e seguindo a arquitetura lógica apresentada na Figura 14.

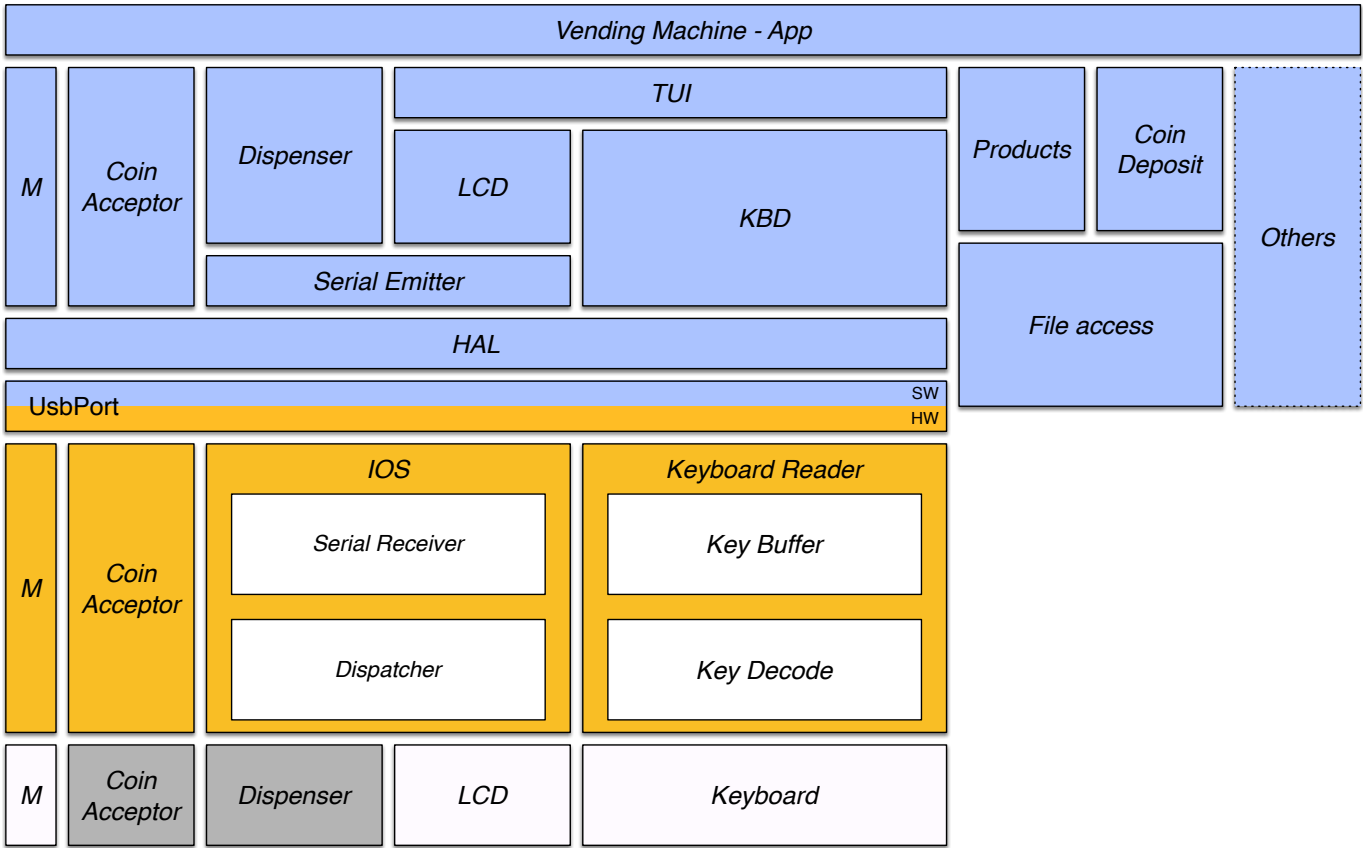


Figura 14 – Diagrama lógico do sistema de controlo da máquina de venda (*Vending Machine*)

As assinaturas das principais classes a desenvolver são apresentadas nas próximas secções. As restantes são objeto de análise e decisão livre.

2.4.1 HAL

```
object HAL {   // Virtualiza o acesso ao sistema UsbPort
   // Inicia a classe
   fun init() ...
   // Retorna true se o bit tiver o valor lógico ‘1’
   fun isBit(mask: Int): Boolean ...
   // Retorna os valores dos bits representados por mask presentes no UsbPort
   fun readBits(mask: Int): Int ...
   // Escreve nos bits representados por mask o valor de value
   fun writeBits(mask: Int, value: Int) ...
   // Coloca os bits representados por mask no valor lógico ‘1’
   fun setBits(int mask) ...
   // Coloca os bits representados por mask no valor lógico ‘0’
   fun clrBits(mask: Int) ...
}
```

2.4.2 KBD

```
object KBD {   // Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.
   const val NONE = 0;
   // Inicia a classe
   fun init() ...
   // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
   fun getKey(): Char ...
   // Retorna quando a tecla for premida ou NONE após decorrido 'timeout' milisegundos.
   fun waitKey(timeout: Long): Char ...
}
```

2.4.3 LCD

```
object LCD {   // Escreve no LCD usando a interface a 8 bits.
   private const val LINES = 2, COLS = 16; // Dimensão do display.
   // Escreve um byte de comando/dados no LCD
   private fun writeByte(rs: Boolean, data: Int) ...
   // Escreve um comando no LCD
   private fun writeCMD(data: Int) ...
   // Escreve um dado no LCD
   private fun writeDATA(data: Int) ...
   // Envia a sequência de iniciação para comunicação a 4 bits.
   fun init() ...
   // Escreve um caráter na posição corrente.
   fun write(c: Char) ...
   // Escreve uma string na posição corrente.
   fun write(text: String) ...
   // Envia comando para posicionar cursor ('line':0..LINES-1 , 'column':0..COLS-1)
   fun cursor(line: Int, column: Int) ...
   // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
   fun clear() ...
}
```

2.4.4 SerialEmitter

```
object SerialEmitter {       // Envia tramas para os diferentes módulos Serial Receiver.
   enum class Destination {DISPENSER, LCD}
   // Inicia a classe
   fun init() ...
   // Envia uma trama para o SerialReceiver identificado o destino em addr e os bits de dados em
   'data'.
   fun send(addr: Destination, data: Int) ...
   // Retorna true se o canal série estiver ocupado
   fun isBusy(): Boolean ...
}
```

2.4.5 *CoinAcceptor*

```
object CoinAcceptor { // Implementa a interface com o moedeiro.
   // Inicia a classe
   fun init() ...
   // Retorna true se foi introduzida uma nova moeda.
   fun hasCoin(): Boolean ...
   // Informa o moedeiro que a moeda foi contabilizada.
   fun acceptCoin() ...
   // Devolve as moedas que estão no moedeiro.
   fun ejectCoins() ...
   // Recolhe as moedas que estão no moedeiro.
   fun collectCoins() ...
}
```

2.4.6 *Dispenser*

```
object Dispenser {           // Controla o estado do mecanismo de dispensa.
   // Inicia a classe, estabelecendo os valores iniciais.
   fun init() ...
   // Envia comando para dispensar uma unidade de um produto
   fun dispense(productid: Int) ...
}
```

3 Calendarização do projeto

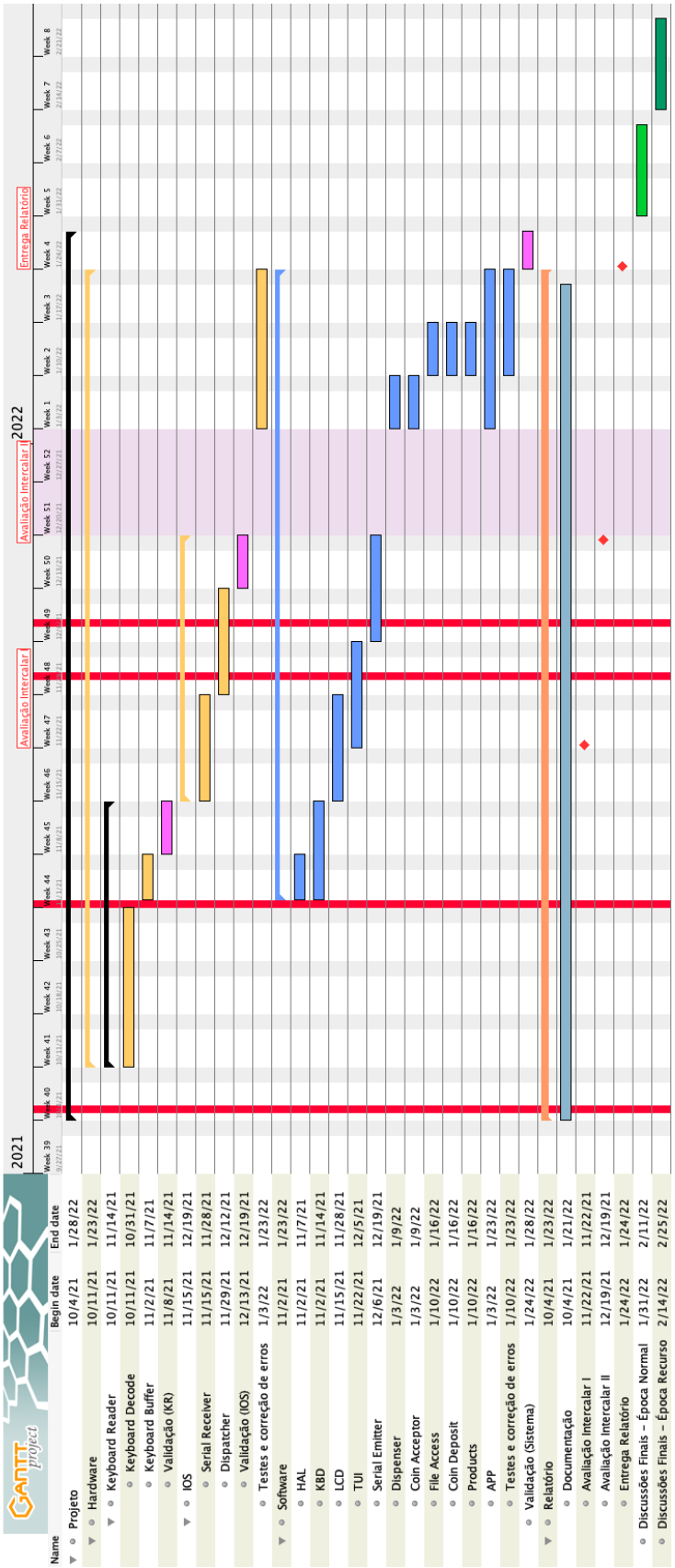


Figura 15 – Diagrama de *Gantt* relativo à calendarização do projeto