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

Licenciatura em Engenharia Informática e de Computadores

Máquina de Venda de Bilhetes
(*Ticket Machine*)

Projeto
de
Laboratório de Informática e Computadores
2021 / 2022 verão

publicado: 05 de março de 2022

1 Descrição

Pretende-se implementar uma máquina de venda de bilhetes (*Ticket Machine*), que permite a aquisição de bilhetes de comboio. O percurso é definido pela estação de origem, local de compra do bilhete e pela seleção do destino digitando o identificador da estação ou através das teclas ↑ e ↓, sendo exibido no ecrã além do identificador da estação de destino o preço e o tipo de bilhete (ida ou ida/volta). A ordem de aquisição é dada através da pressão da tecla de confirmação, sendo impressa uma unidade do bilhete exibido no ecrã. A máquina não realiza trocos e só aceita moedas de: 0,05€; 0,10€; 0,20€; 0,50€; 1,00€; e 2,00€. 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 o teste da máquina de venda de bilhetes, além disso permite iniciar e consultar os contadores de bilhetes e moedas.

A máquina de venda de bilhetes é constituída pelo sistema de gestão (designado por *Control* na Figura 1) e pelos seguintes periféricos: 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 impressão de bilhetes (designado por *Ticket Dispenser*) e uma chave de manutenção (designada por *M*) que define se a máquina de venda de bilhetes está em modo de Manutenção, conforme o diagrama de blocos apresentado na Figura 1.

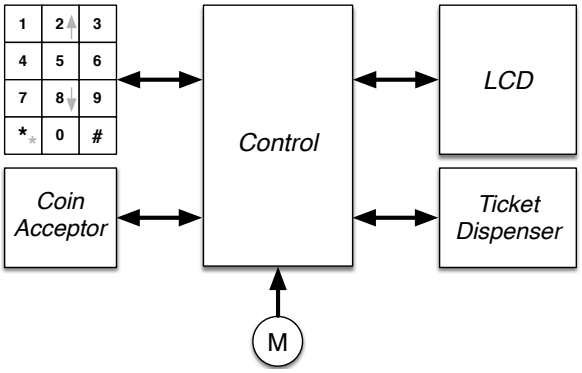


Figura 1 – Máquina de venda de bilhetes (*Ticket Machine*)

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

- **Consulta e venda** – A consulta de um bilhete é realizada digitando o identificador da estação de destino ou listando-a através das teclas ↑ e ↓. O processo de compra do bilhete inicia-se premindo a tecla '#'. Durante a inserção do respetivo valor monetário, é possível alterar o tipo de bilhete (ida ou ida/volta) premindo a tecla '0', com a consequente alteração do preço da viagem afixado no *LCD*, duplicando o valor no caso de ida/volta. Durante a compra ficam afixados no *LCD* as informações referentes ao bilhete pretendido até que o mecanismo de impressão de bilhetes confirme que a impressão já foi realizada e a recolha do bilhete efetuada. O modo de seleção ↑ e ↓ alterna com a seleção numérica por pressão da tecla '*'. A compra pode ser cancelada premindo a tecla '#', devolvendo as moedas inseridas.

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

- **Teste** – Esta opção do menu permite realizar um procedimento de consulta e venda de um bilhete, sem introdução de moedas e sem esta operação ser contabilizada como uma aquisição.
- **Consulta** – Para visualizar os contadores de moedas e bilhetes seleciona-se a operação de consulta no menu, e permite-se a listagem dos contadores de moedas e bilhetes, através das teclas ↑ e ↓.
- **Iniciar** – Esta opção do menu inicia os contadores de moedas e bilhetes a zero, iniciando um novo ciclo de contagem.
- **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. A informação do número de moedas no cofre do moedeiro e dos bilhetes vendidos deve ser armazenada em ficheiros separados. A informação em cada ficheiros deve estar organizada por linha, em que os campos de dados são separados por ";", com o respetivo formato: "*COIN;NUMBER*" (moedas) e "*PRICE;NUMBER;STATION_NAME*" (bilhetes vendidos). Estes ficheiros são lidos e carregados no início do programa e reescritos 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 de bilhetes, 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.

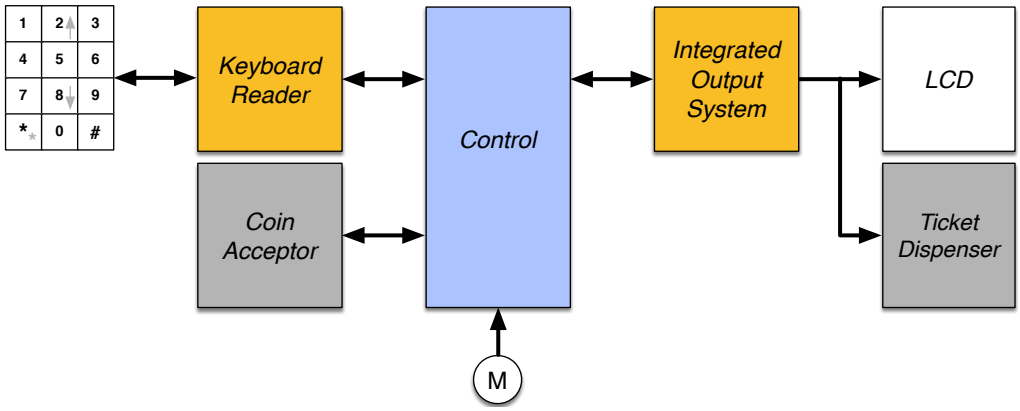


Figura 2 – Arquitetura do sistema que implementa a Máquina de Venda de Bilhetes (*Ticket Machine*)

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

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 Transmitter*), conforme ilustrado na Figura 3. Neste caso o módulo de controlo, implementado em *software*, é a entidade consumidora.

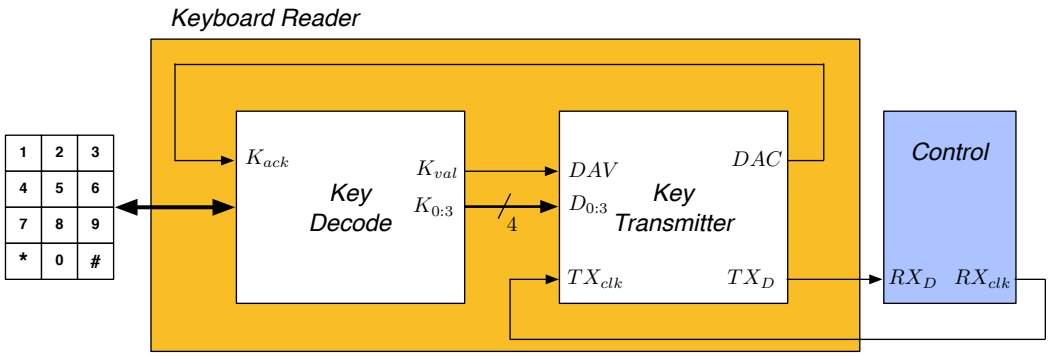
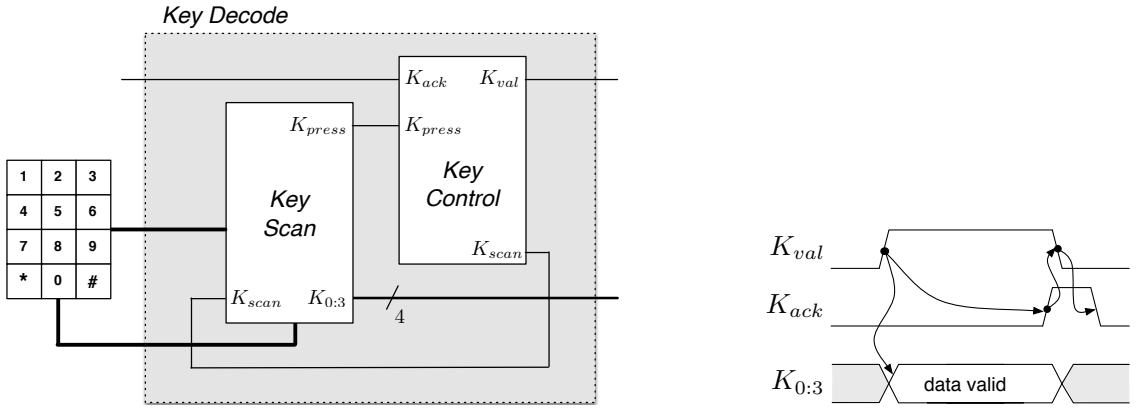


Figura 3 – Diagrama de blocos do módulo *Keyboard Reader*

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.

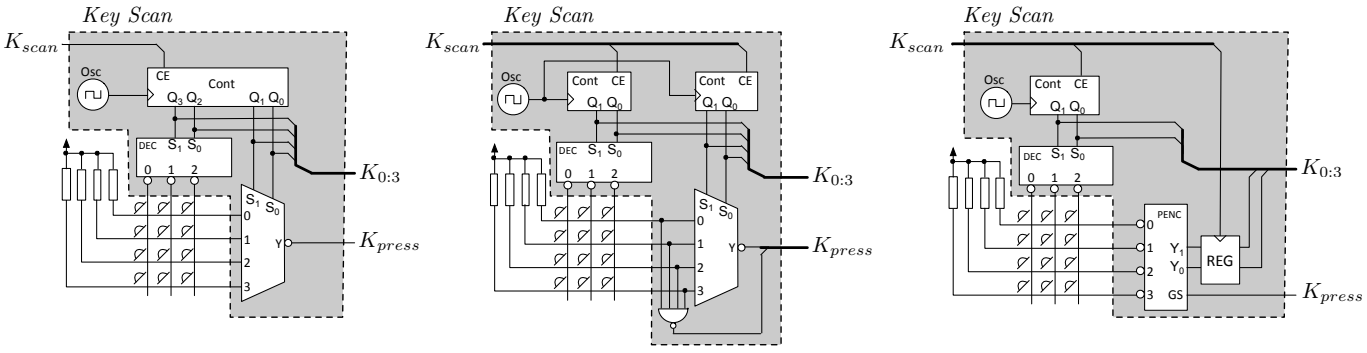
O controlo de fluxo de saída do bloco *Key Decode* (para o módulo *Key Transmitter*), define que o sinal K_{val} é ativado quando é detetada a pressão de uma tecla, sendo também disponibilizado o código dessa tecla no barramento $K_{0:3}$. Apenas é iniciado um novo ciclo de varrimento ao teclado quando 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 Transmitter

O bloco *Key Transmitter* a desenvolver corresponderá a uma estrutura de transmissão série, com capacidade para armazenar e transmitir uma palavra de quatro bits. A escrita de dados no bloco *Key Transmitter* 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 Transmitter* regista os dados $D_{0:3}$ na sua memória interna. Concluída a escrita na memória interna, 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 Transmitter* só desativa o sinal *DAC* após o sinal *DAV* ter sido desativado.
O bloco *Key Transmitter* também é responsável pela interação com o sistema consumidor, neste caso o módulo *Control*.
Este bloco quando tem dados para transmitir, sinaliza o módulo *Control* através duma transição descendente do sinal TX_D, enquanto o sinal TX_{clk} está no valor lógico zero. A cada transição ascendente do sinal TX_{clk} o bloco *Key Transmitter* coloca o sinal TX_D com o valor lógico necessário em cada instante, de acordo com o protocolo representado na Figura 6. Para que uma nova palavra possa ser armazenada neste bloco será necessário que o módulo *Control* tenha recebido todos os bits. A implementação do bloco *Key Transmitter* fica como objeto de análise e estudo, devendo a sua arquitetura ser proposta pelos alunos.

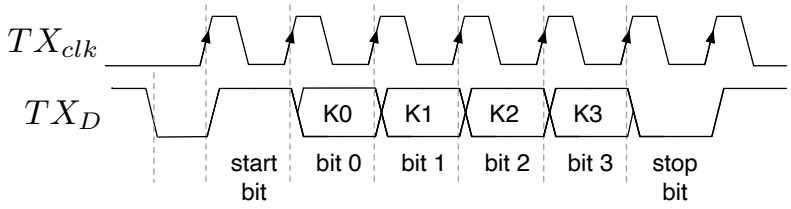


Figura 6 – Protocolo de comunicação com o módulo *Key Transmitter*

2.2 Coin Acceptor

Este módulo implementa a interface com o moedeiro, sinalizando que este recebeu uma moeda através da ativação do sinal *Coin*. O valor monetário da moeda inserida é codificado em $Cid_{0:2}$ conforme a codificação apresentada na Tabela 1. 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 dois segundos, ou recolher as moedas presentes no moedeiro através da ativação do sinal *collect* durante dois segundos.

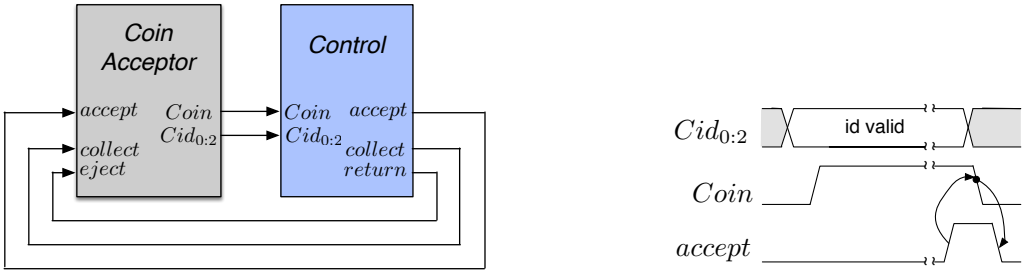


Figura 7 – Diagrama de blocos e diagrama temporal do *Coin Acceptor*

Moeda [€]	**Codificação $[Cid_2 Cid_1 Cid_0]$**
0,05	000
0,10	001
0,20	010
0,50	011
1,00	100
2,00	101

Tabela 1 – Codificação do valor facial das moedas

2.3 Integrated Output System

O módulo *Integrated Output System (IOS)* implementa a interface com o mecanismo de dispensa de bilhetes e com o *LCD*, fazendo a receção em série da informação enviada pelo módulo de controlo e entregando-a posteriormente ao destinatário, conforme representado na Figura 8.

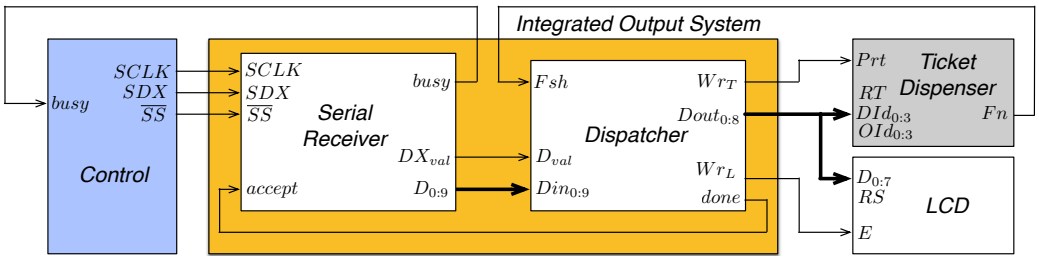


Figura 8 – Diagrama de blocos do *Integrated Output System*

O módulo *IOS* recebe, em série, uma mensagem constituída por 10 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 *TnL* identifica o destinatário da mensagem. Nas mensagens para o mecanismo de dispensa de bilhetes, ilustrado na Figura 10, o bit *RT* é o primeiro bit de informação e indica o tipo de bilhete (ida ou ida/volta), os seguintes 4 bits contêm o código de identificação da estação de destino, e os restantes 4 bits contêm o código de identificação da estação de origem. O último bit contém a informação de paridade par, utilizada para detetar erros de transmissão. As mensagens para o *LCD*, ilustrado na Figura 10, contêm para além do bit *TnL* e do bit paridade outros 9 bits de dados a entregar ao dispositivo: o bit *RS*, que é o primeiro bit de informação e indica se a mensagem é de controlo ou dados, e os restantes 8 bits que contêm os dados a entregar ao *LCD*.

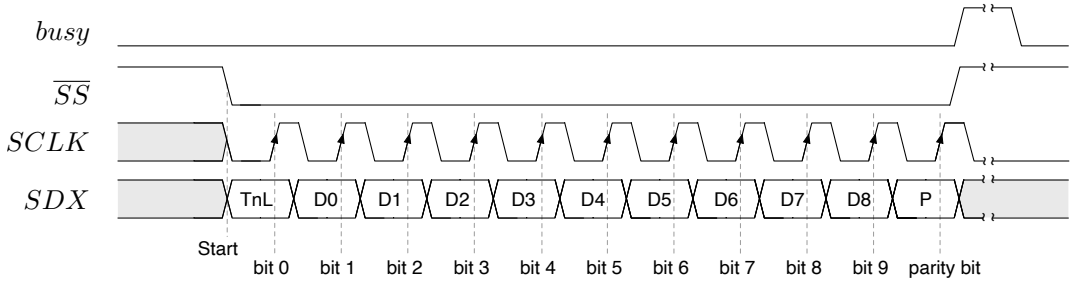


Figura 9 – Protocolo de comunicação com o módulo *Integrated Output System*

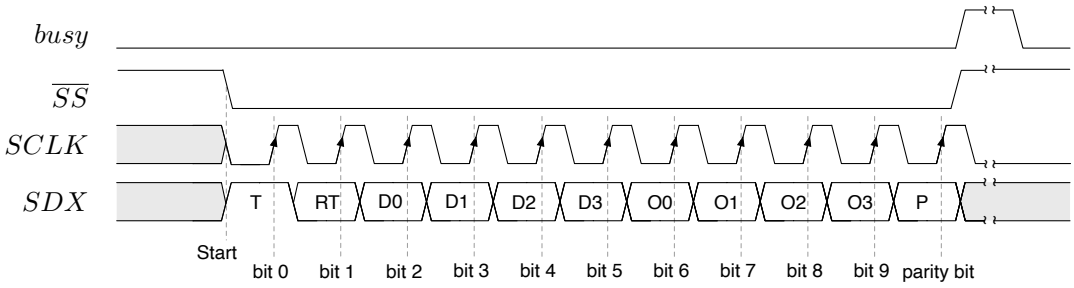


Figura 10 – Trama para o mecanismo de dispensa de bilhetes (*Ticket Dispenser)*

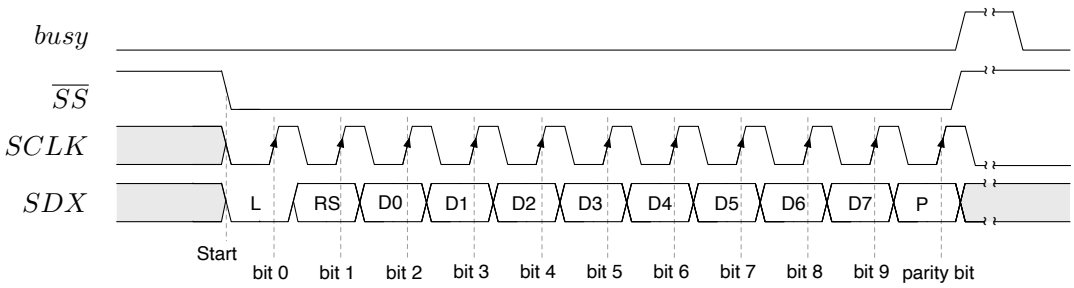


Figura 11 – Trama para o *LCD*

O emissor, realizado em *software,* quando pretende enviar uma trama para o módulo *IOS* aguarda que este esteja disponível para receção, ou seja, que o sinal *busy* esteja desativo. Em seguida, promove uma condição de início de trama (*Start*), que corresponde a uma transição descendente na linha $\overline{SS}$, mantendo-se esta no valor lógico zero até ao fim da transmissão. 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.

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, implementado através de um registo de deslocamento; *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 12.

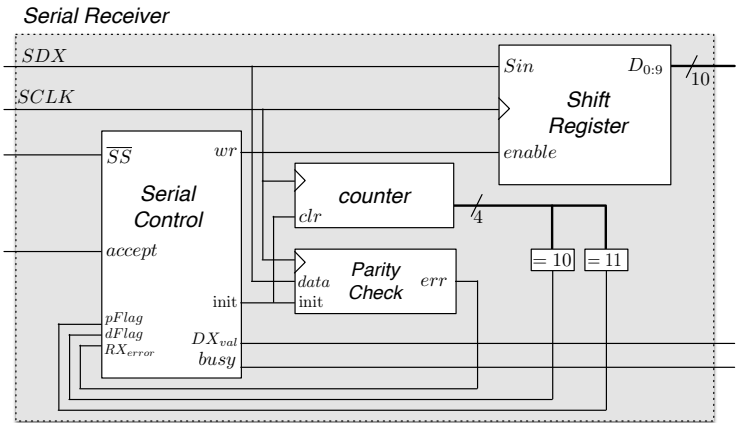


Figura 12 – Diagrama de blocos do bloco *Serial Receiver*

2.3.2 Dispatcher

O bloco *Dispatcher* é responsável pela entrega das tramas válidas recebidas pelo bloco *Serial Receiver* ao *Ticket Dispenser* e ao *LCD*, através da ativação do sinal Wr_T e Wr_L. A receção de uma nova trama válida é sinalizada pela ativação do sinal D_{val}.
O processamento das tramas recebidas pelo *IOS,* para o *Ticket Dispenser* ou para o *LCD,* 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 Ticket Dispenser

O *Ticket Dispenser* recebe em 4 bits o código da estação de destino ($DId_{0:3}$), noutros 4 bits o código da estação de origem ($OId_{0:3}$) a imprimir no bilhete e ainda o bit *RT* que define o tipo de bilhete (ida ou ida/volta). O comando de impressão do bilhete com os códigos presentes em *DId* e *RT* é realizado pela ativação do sinal de impressão (*Prt*). Em resposta, o *Ticket Dispenser* ativa o sinal de término de execução (*Fn*) quando concluída a dispensa do bilhete. Os sinais *Fn* e *Prt* têm o comportamento descrito no diagrama temporal apresentado na Figura 13.

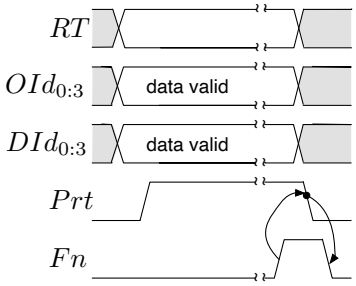


Figura 13- Diagrama temporal do mecanismo de dispensa de bilhetes

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.

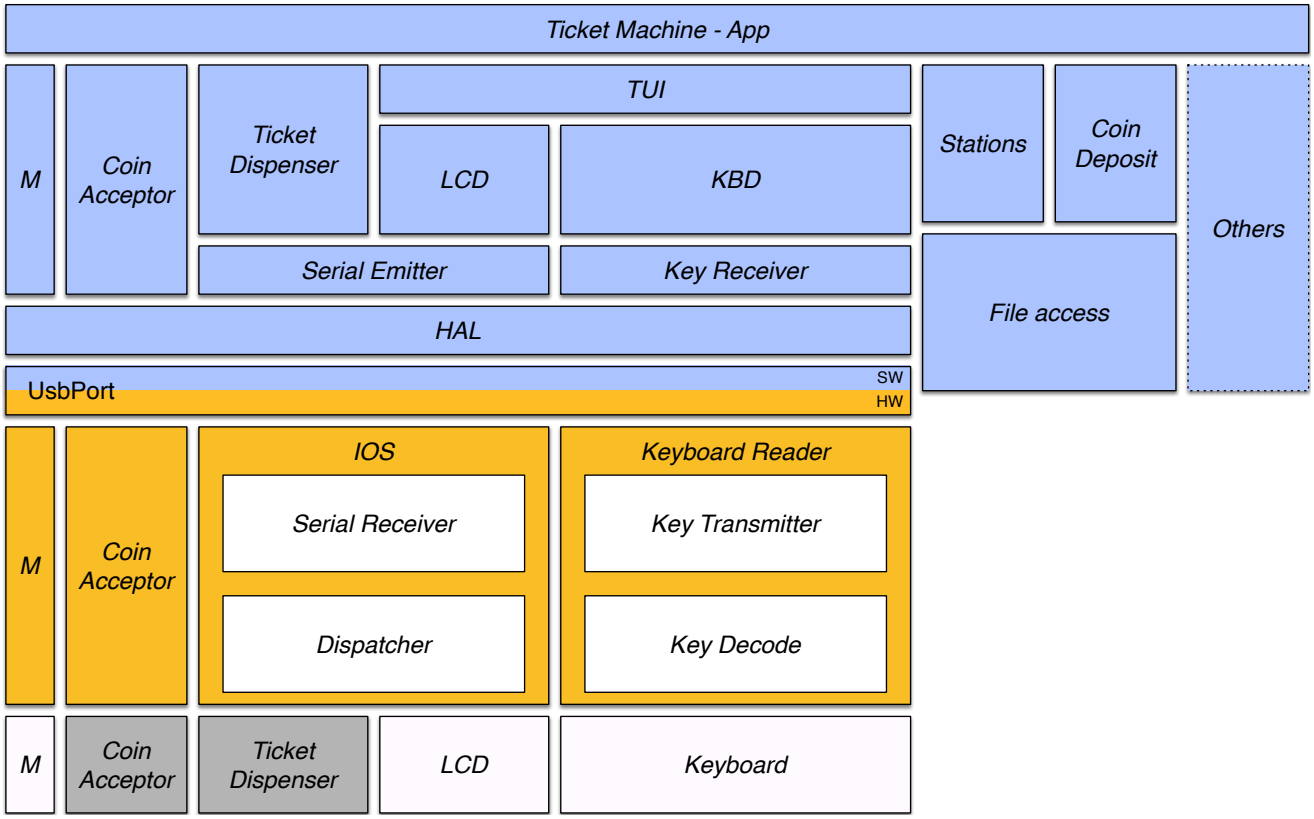


Figura 14 – Diagrama lógico do sistema de controlo da Máquina de Venda de Bilhetes (*Ticket Machine*)

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(mask: Int) ...
   // Coloca os bits representados por mask no valor lógico '0'
   fun clrBits(mask: Int) ...
}
```

2.4.2 KBD

```
KBD {  // Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.
   const val NONE = 0;
   // Inicia a classe
   fun init() ...
   // Implementa a interação paralela com o Key Decode, utilizado na 1ªfase do projeto
   private fun getKeyParallel(): Char ...
   // Implementa a interação série com o Key Transmitter, utilizado na 2ªfase do projeto
   private fun getKeySerial(): Char ...
   // 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 KeyReceiver

```
KeyReceiver {  // Recebe trama do Keyboard Reader.
   // Inicia a classe
   fun init() ...
   // Recebe uma trama e retorna o código de uma tecla caso exista
   fun rcv(): Int ...
}
```

2.4.4 LCD

```
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 em paralelo
   private fun writeByteParallel(rs: Boolean, data: Int) ...
   // Escreve um byte de comando/dados no LCD em série
   private fun writeByteSerial(rs: Boolean, data: Int) ...
   // 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 8 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() ...
}
```

2.4.5 *SerialEmitter*

```
SerialEmitter {       // Envia tramas para os diferentes módulos Serial Receiver.
   enum class Destination {LCD, TICKET_DISPENSER}
   // 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 ...
}
```

2.4.6 *CoinAcceptor*

```
CoinAcceptor { // Implementa a interface com o moedeiro.
   // Inicia a classe
   fun init() ...
   // Retorna true se foi introduzida uma nova moeda.
   fun hasCoin(): Boolean ...
   // Retorna o valor facial da moeda introduzida.
   fun getCoinValue(): Int ...
   // 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() ...
}
```

2.4.7 *TicketDispenser*

```
TicketDispenser {// Controla o estado do mecanismo de dispensa de bilhetes.
   // Inicia a classe, estabelecendo os valores iniciais.
   fun init() ...
   // Envia comando para imprimir e dispensar um bilhete
   fun print(destinyId: Int, originId: Int, roundTrip: Boolean) ...
}
```

3 Calendarização do projeto

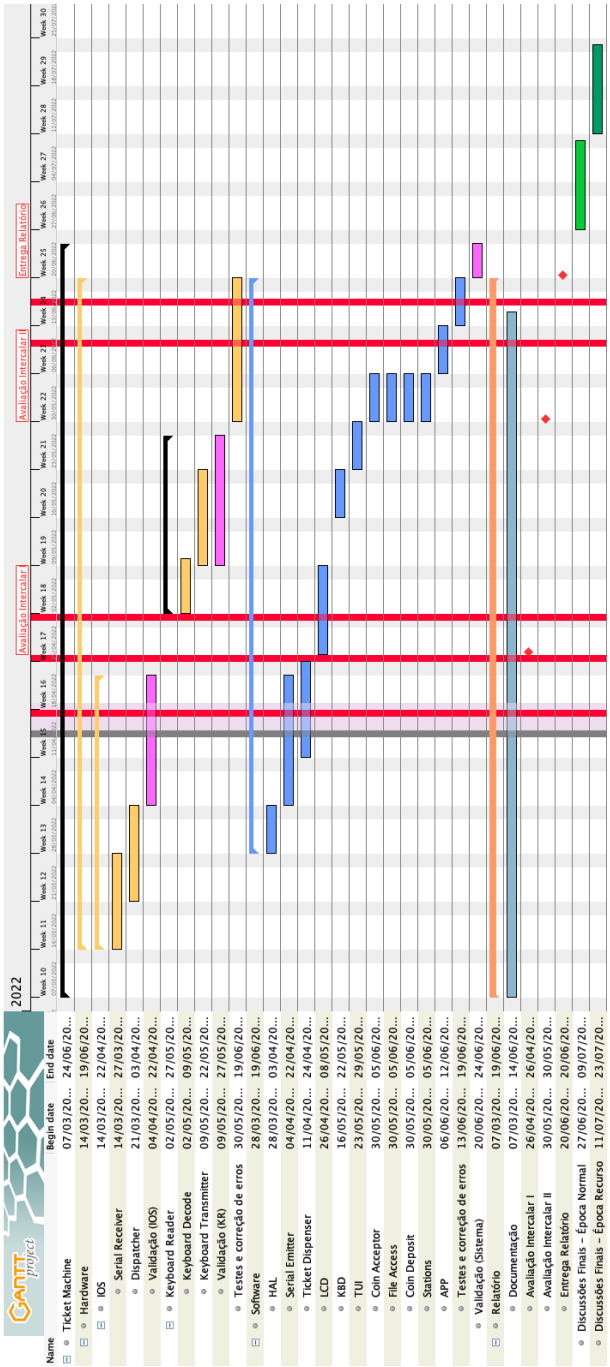


Figura 15 – Diagrama de *Gantt* relativo à calendarização do projeto