ISEL
Departamento de Engenharia
Eletrónica e Telecomunicações
e de Computadores

Licenciatura em Engenharia Informática e de Computadores

Sistema de Controlo de Acessos
(*Access Control System*)

Projeto
de
Laboratório de Informática e Computadores
2022 / 2023
verão

publicado: 27 de fevereiro de 2023

1 Descrição

Pretende-se implementar um sistema de controlo de acessos (*Access Control System*), que permita controlar o acesso a zonas restritas através de um número de identificação de utilizador (*User Identification Number – UIN*) e um código de acesso (*Personal Identification Number - PIN*). O sistema permite o acesso à zona restrita após a inserção correta de um par *UIN* e *PIN*. Após o acesso válido o sistema permite a entrega de uma mensagem de texto ao utilizador.

O sistema de controlo de acessos é constituído por: um teclado de 12 teclas; um ecrã *Liquid Cristal Display* (LCD) de duas linhas de 16 caracteres; um mecanismo de abertura e fecho da porta (designado por *Door Mechanism*); uma chave de manutenção (designada por M) que define se o sistema de controlo de acessos está em modo de Manutenção; e um PC responsável pelo controlo dos outros componentes e gestão do sistema. O diagrama de blocos do sistema de controlo de acessos é apresentado na Figura 1.

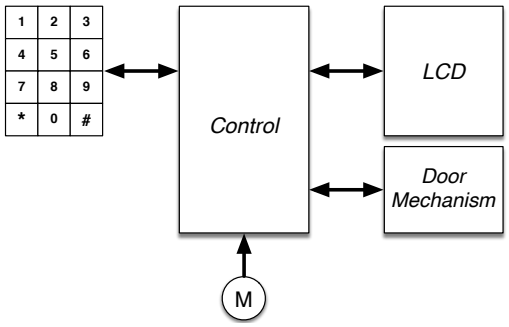


Figura 1 – Sistema de controlo de acessos (*Access Control System*)

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

- **Acesso -** Para acesso às instalações, o utilizador deverá inserir os três dígitos correspondentes ao *UIN* seguido da inserção dos quatro dígitos numéricos do *PIN*. Se o par *UIN* e *PIN* estiver correto o sistema apresenta no LCD o nome do utilizador e a mensagem armazenada no sistema se existir, acionando a abertura da porta. A mensagem é removida do sistema caso seja premida a tecla '*' durante a apresentação desta. Todas os acessos deverão ser registados com a informação de data/hora e *UIN* num ficheiro de registos (um registo de entrada por linha), designado por *Log File*.
- **Alteração do PIN –** Esta ação é realizada se após o processo de autenticação for premida a tecla '#'. O sistema solicita ao utilizador o novo *PIN*, este deverá ser novamente introduzido de modo a ser confirmado. O novo *PIN* só é registado no sistema se as duas inserções forem idênticas.

Nota: A inserção de informação através do teclado tem o seguinte critério: se não for premida nenhuma tecla num intervalo de cinco segundos, o comando em curso é abortado; se for premida a tecla '*' e o sistema contiver dígitos, elimina todos os dígitos, se não contiver dígitos, aborta o comando em curso.

Sobre o sistema, podem-se realizar também as seguintes ações em modo Manutenção. Ao contrário das ações em modo Acesso, as ações em modo Manutenção são realizadas através do teclado e ecrã do PC. As ações disponíveis neste modo são:

- **Inserção de utilizador -** Tem como objetivo inserir um novo utilizador no sistema. O sistema atribui o primeiro *UIN* disponível, e espera que seja introduzido pelo gestor do sistema o nome e o *PIN* do utilizador. O nome tem no máximo 16 caracteres.
- **Remoção de utilizador -** Tem como objetivo remover um utilizador do sistema. O sistema espera que o gestor do sistema introduza o *UIN* e pede confirmação depois de apresentar o nome.
- **Inserir mensagem -** Permite associar uma mensagem de informação dirigida a um utilizador específico a ser exibida ao utilizador no processo de autenticação de acesso às instalações.
- **Desligar –** Permite desligar o sistema de controlo de acessos. Este termina após a confirmação do utilizador e reescreve o ficheiro com a informação dos utilizadores. Esta informação deverá ser armazenada num ficheiro de texto (com um utilizador por linha) que é carregado no início do programa e reescrito no final do programa. O sistema armazena até 1000 utilizadores, que são inseridos e suprimidos através do teclado do PC pelo gestor do sistema.

Nota: Durante a execução das ações em modo manutenção, não podem ser realizadas ações no teclado do utilizador e no LCD deve constar a mensagem "*Out of Service*".

2 Arquitetura do sistema

O controlo (designado por *Control*) do sistema de acessos será implementado numa solução híbrida de *hardware* e *software*, como apresentado no diagrama de blocos da Figura 2. A arquitetura proposta é constituída por quatro módulos principais: *i*) um leitor de teclado, designado por *Keyboard Reader*; *ii*) um módulo de interface com o *LCD*, designado por *Serial LCD Controller* (*SLCDC*); *iii*) um módulo de interface com o mecanismo da porta (*Door Mechanism)*, designado por *Serial Door Controller* (*SDC*);; e *iv*) um módulo de controlo, designado por *Control*. Os módulos *i*), *ii*) e *iii*) deverão ser implementados em *hardware* e o módulo de controlo deverá ser implementado em *software* a executar num PC.

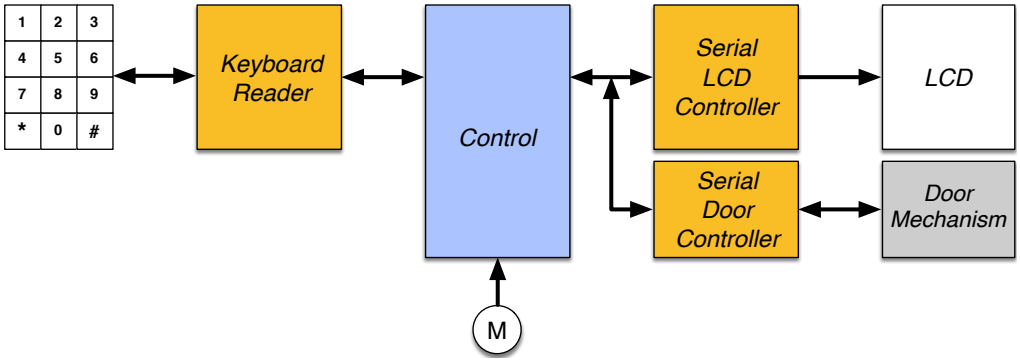


Figura 2 – Arquitetura do sistema que implementa o Sistema de Controlo de Acessos (*Access Control System*)

O módulo *Keyboard Reader* é responsável pela descodificação do teclado matricial de 12 teclas, determinando qual a tecla pressionada e disponibilizando o código desta em quatro bits ao *Control*, caso este esteja disponível para o receber. Caso este não esteja disponível para o receber imediatamente, o código da tecla é armazenado até ao limite de nove códigos. O *Control* processa e envia para o *SLCDC* a informação contendo os dados a apresentar no *LCD*. A informação para o mecanismo da porta é enviada através do *SDC*. Por razões de ordem física, e por forma a minimizar o número de sinais de interligação, a comunicação entre o módulo *Control* e os módulos *SLCDC e SDC* é realizada através de um protocolo série.

2.1 Keyboard Reader

O módulo *Keyboard Reader* é constituído por três blocos principais: *i*) o descodificador de teclado (*Key Decode*); *ii*) o bloco de armazenamento (designado por *Ring Buffer*); e *iii*) o bloco de entrega ao consumidor (designado por *Output Buffer*). Neste caso o módulo *Control*, implementado em *software*, é a entidade consumidora.

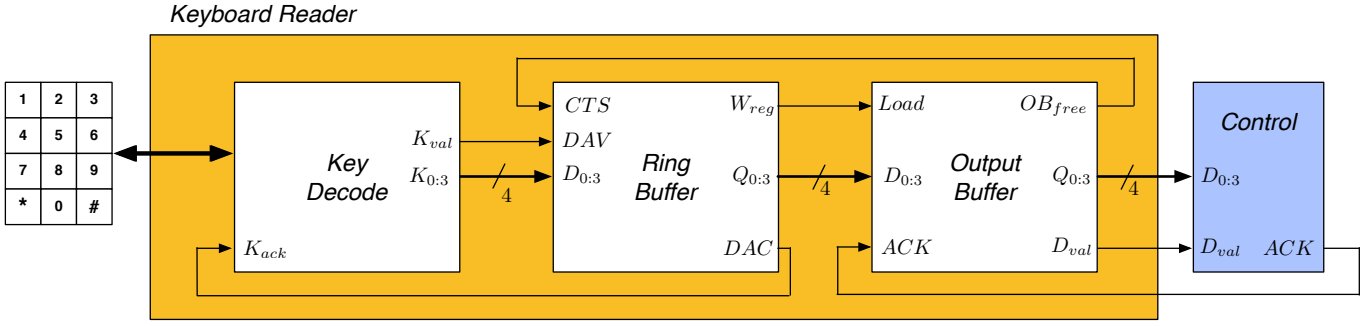


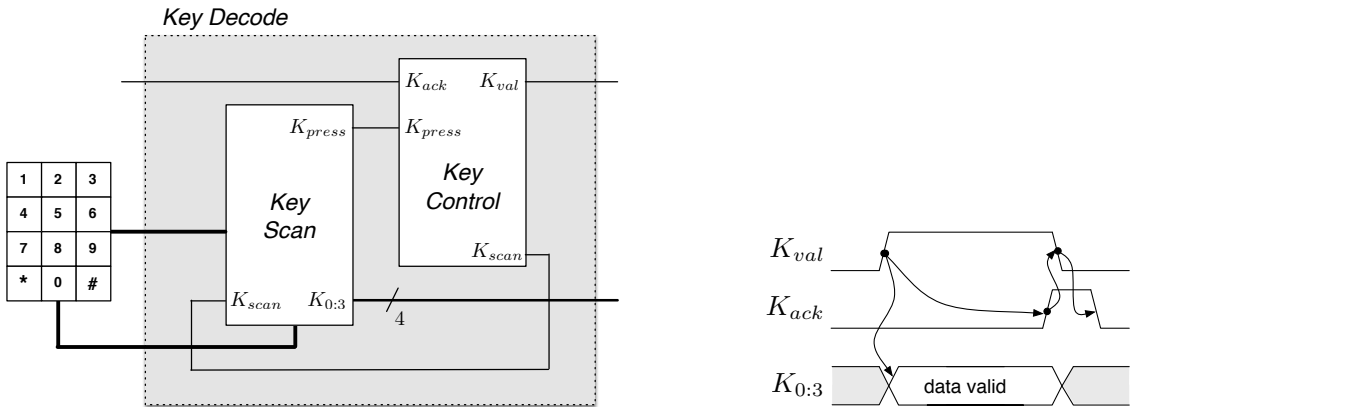
Figura 3 – Diagrama de blocos do *Keyboard Reader*

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 bloco *Ring 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

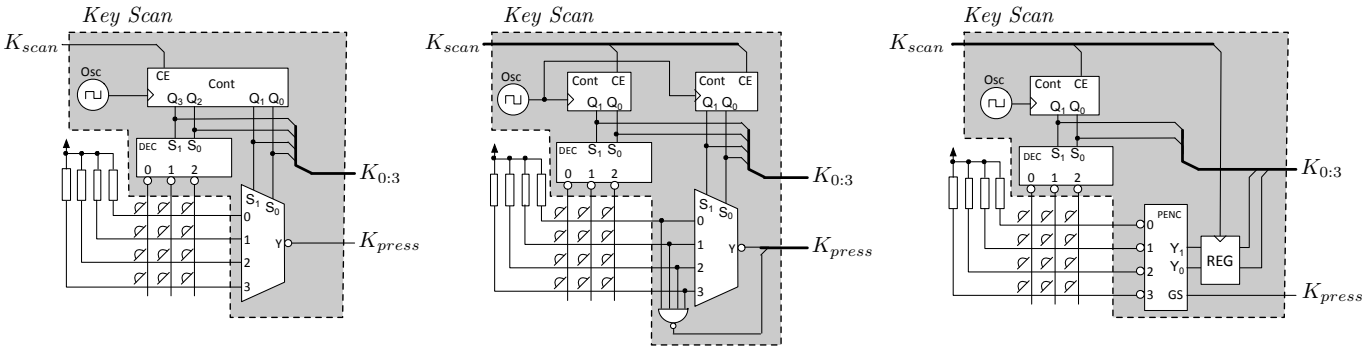
de varrimento ao teclado quando o sinal K_{ack} for ativado e a tecla premida for libertada. O diagrama temporal do controlo de fluxo está representado na Figura 4b.



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 Ring Buffer

O bloco *Ring Buffer* a desenvolver deverá ser uma estrutura de dados para armazenamento de teclas com disciplina FIFO (*First In First Out*), com capacidade de armazenar até oito palavras de quatro bits.

A escrita de dados no *Ring Buffer* inicia-se com a ativação do sinal DAV (*Data Available*) pelo sistema produtor, neste caso pelo *Key Decode*, indicando que tem dados para serem armazenados. Logo que tenha disponibilidade para armazenar informação, o *Ring Buffer* escreve os dados $D_{0:3}$ em memória. Concluída a escrita em memória ativa o sinal DAC (*Data Accepted*) para informar o sistema produtor que os dados foram aceites. O sistema produtor mantém o sinal DAV ativo até que DAC seja ativado. O *Ring Buffer* só desativa DAC depois de DAV ter sido desativado.

A implementação do *Ring Buffer* deverá ser baseada numa memória RAM (*Random Access Memory*). O endereço de escrita/leitura, selecionado por $put\overline{get}$, deverá ser definido pelo bloco *Memory Address Control* (MAC) composto por dois registos, que contêm o endereço de escrita e leitura, designados por *putIndex* e *getIndex* respetivamente. O MAC suporta assim ações de *incPut* e *incGet*, gerando informação se a estrutura de dados está cheia (*Full*) ou se está vazia (*Empty*). O bloco *Ring Buffer* procede à entrega de dados à entidade consumidora, sempre que esta indique que está disponível para receber, através do sinal *Clear To Send* (CTS). Na Figura 6 é apresentado o diagrama de blocos para uma estrutura do bloco *Ring Buffer*.

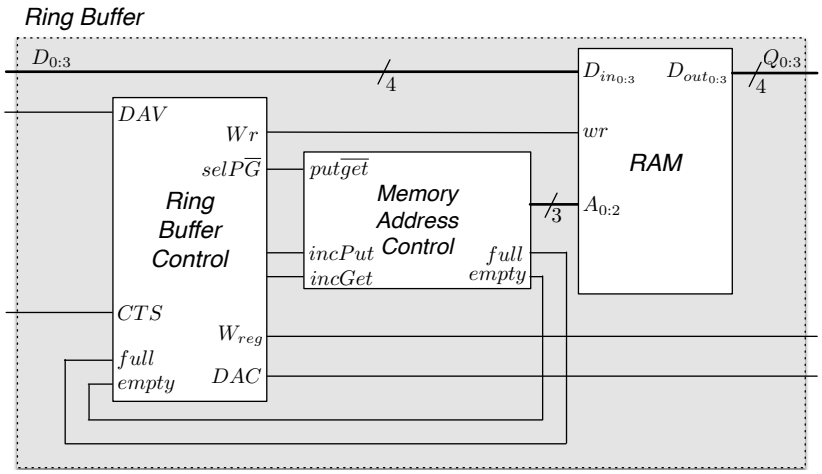


Figura 6 - Diagrama de blocos do bloco *Ring Buffer*

2.1.3 Output Buffer

O bloco *Output Buffer* do *Keyboard Reader* é responsável pela interação com o sistema consumidor, neste caso o módulo *Control*.
O *Output Buffer* indica que está disponível para armazenar dados através do sinal OB_{free}. Assim, nesta situação o sistema produtor pode ativar o sinal *Load* para registar os dados.
O *Control* quando pretende ler dados do *Output Buffer*, aguarda que o sinal D_{val} fique ativo, recolhe os dados e pulsa o sinal *ACK* indicando que estes já foram consumidos.
O *Output Buffer*, logo que o sinal *ACK* pulse, deve invalidar os dados baixando o sinal D_{val} e sinalizar que está novamente disponível para entregar dados ao sistema consumidor, ativando o sinal OB_{free}. Na Figura 7, é apresentado o diagrama de blocos do *Output Buffer*.

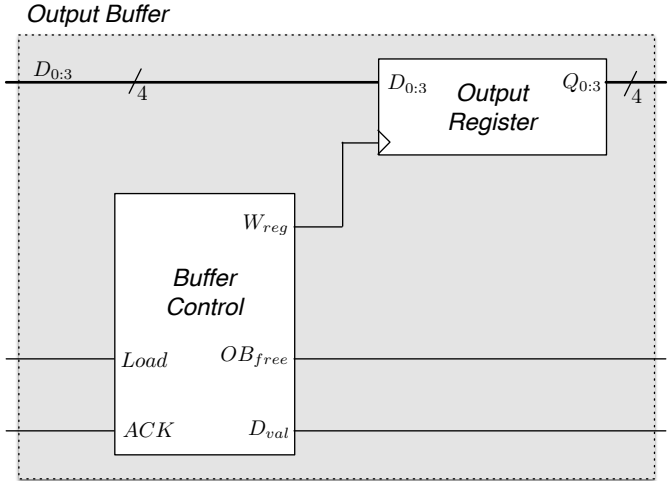


Figura 7 - Diagrama de blocos do bloco *Output Buffer*

Sempre que o bloco emissor *Ring Buffer* tenha dados disponíveis e o bloco de entrega *Output Buffer* esteja disponível (OB_{free} ativo), o *Ring Buffer* realiza uma leitura da memória e entrega os dados ao *Output Buffer* ativando o sinal W_{reg}. O *Output Buffer* indica que já registou os dados desativando o sinal OB_{free}.

2.2 Serial LCD Controller

O módulo de interface com o LCD (*Serial LCD Interface, SLCDC)* implementa a receção em série da informação enviada pelo módulo de controlo, entregando-a posteriormente ao *LCD*, conforme representado na Figura 8.

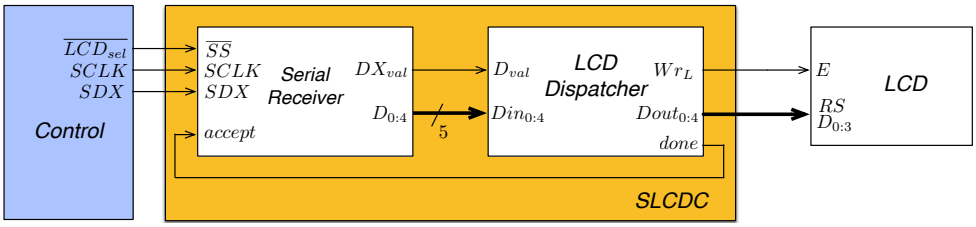


Figura 8 – Diagrama de blocos do *Serial LCD Controller*

O *SLCDC* recebe em série uma mensagem constituída por cinco bits de informação. A comunicação com o *SLCDC* realiza-se segundo o protocolo ilustrado na Figura 9, tendo como primeiro bit de informação, o bit *RS* que indica se a mensagem é de controlo ou dados, os restantes bits contêm os dados a transmitir ao *LCD*.

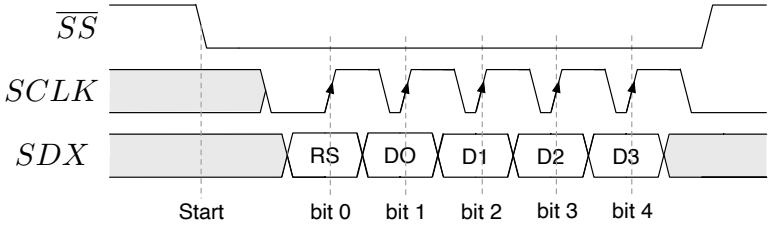


Figura 9 – Protocolo de comunicação com *Serial LCD Controller*

O emissor, realizado em *software,* quando pretende enviar uma trama para o S*LCDC* promove uma condição de início de trama (*Start*), que corresponde a uma transição descendente na linha de $\overline{LCD_{sel}}$. Após a condição de início, o S*LCDC* armazena os bits da trama nas transições ascendentes do sinal *SCLK*.

2.2.1 Serial Receiver

O bloco *Serial Receiver* do S*LCDC* é constituído por três blocos principais: *i*) um bloco de controlo; *ii*) um contador de bits recebidos; e *iii*) um bloco conversor série paralelo, designados respetivamente por *Serial Control, Counter*, e *Shift Register*. O *Serial Receiver* deverá ser implementado com base no diagrama de blocos apresentado na Figura 10.

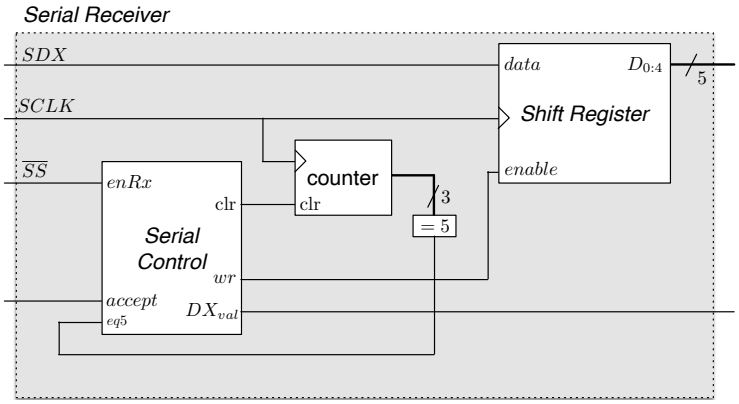


Figura 10 – Diagrama de blocos do *Serial Receiver*

2.2.2 Dispatcher

O bloco *Dispatcher* entrega a trama recebida pelo *Serial Receiver* ao *LCD* através da ativação do sinal Wr_L, após este ter recebido uma trama válida, indicado pela ativação do sinal DX_{val}.
O *LCD* processa as tramas recebidas de acordo com os comandos definidos pelo fabricante, não sendo necessário esperar pela sua execução para libertar o canal de receção série. Assim, o *Dispatcher* pode sinalizar ao *Serial Receiver* que a trama foi processada, ativando o sinal *done*.

2.3 Serial Door Controller

O módulo de interface com o mecanismo da porta (*Serial Door Controller, SDC)* implementa a receção em série da informação enviada pelo módulo de controlo, entregando-a posteriormente ao mecanismo da porta, conforme representado na Figura 11.

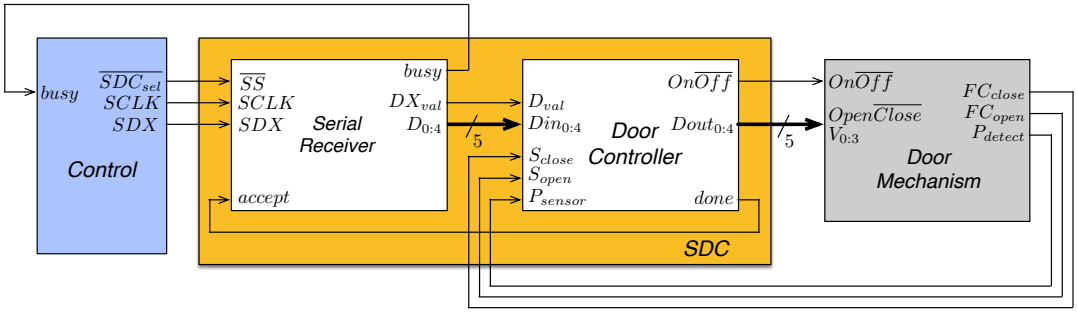


Figura 11 – Diagrama de blocos do *Serial Door Controller*

O *SDC* recebe em série uma mensagem constituída por cinco bits de informação. A comunicação com o *SDC* realiza-se segundo o protocolo ilustrado na Figura 12, tendo como primeiro bit de informação, o bit $Open\overline{Close}$ (*OC*) que indica se o comando é para abrir ou fechar a porta. Os restantes bits contêm a informação da velocidade de abertura ou fecho. O *SDC* indica que está disponível para a receção de uma nova trama após ter processado a trama anterior, colocando o *busy* no nível lógico "0".

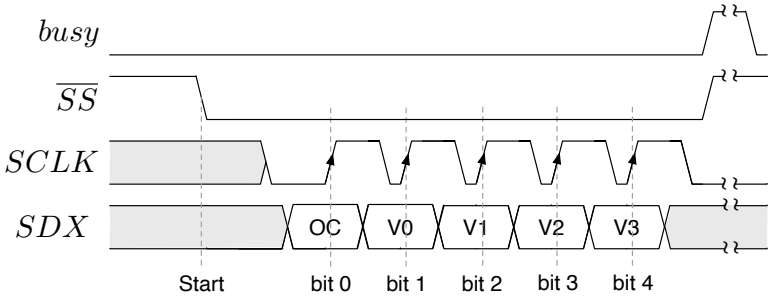


Figura 12 – Protocolo de comunicação do *Serial Door Controller*

2.3.1 Serial Receiver

O bloco *Serial Receiver* do *SDC* deve ser implementado com uma estrutura similar ao bloco *Serial Receiver* do *SLCDC*.

2.3.2 Door Controller

O bloco *Door Controller*, após este ter recebido uma trama válida recebida pelo *Serial Receiver*, deverá proceder à atuação do comando recebido no mecanismo da porta. Se o comando recebido for de abertura, o *Door Controller* deverá colocar o sinal $On\overline{Off}$ e sinal $Open\overline{Close}$ no valor lógico '1', até o sensor de porta aberta (FC_{open}) ficar ativo. No entanto, se o comando for de fecho, o *Door Controller* deverá ativar o sinal $On\overline{Off}$ e colocar o sinal $Open\overline{Close}$ no valor lógico '0', até o sensor de porta fechada (FC_{close}) ficar ativo. Se durante o fecho for detetada uma pessoa na zona da porta, através do sensor de presença (P_{detect}), o sistema deverá interromper o fecho reabrindo a porta. Após a interrupção do fecho da porta, o bloco *Door Controller* deverá permitir de forma automática, ou seja, sem necessidade de envio de uma nova trama, o encerramento da porta e o finalizar do comando de fecho. Após concluir qualquer um dos comandos, o *Door Controller* sinaliza o *Serial Receiver* que está pronto para processar uma nova trama através da ativação do sinal *done*.

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 13.

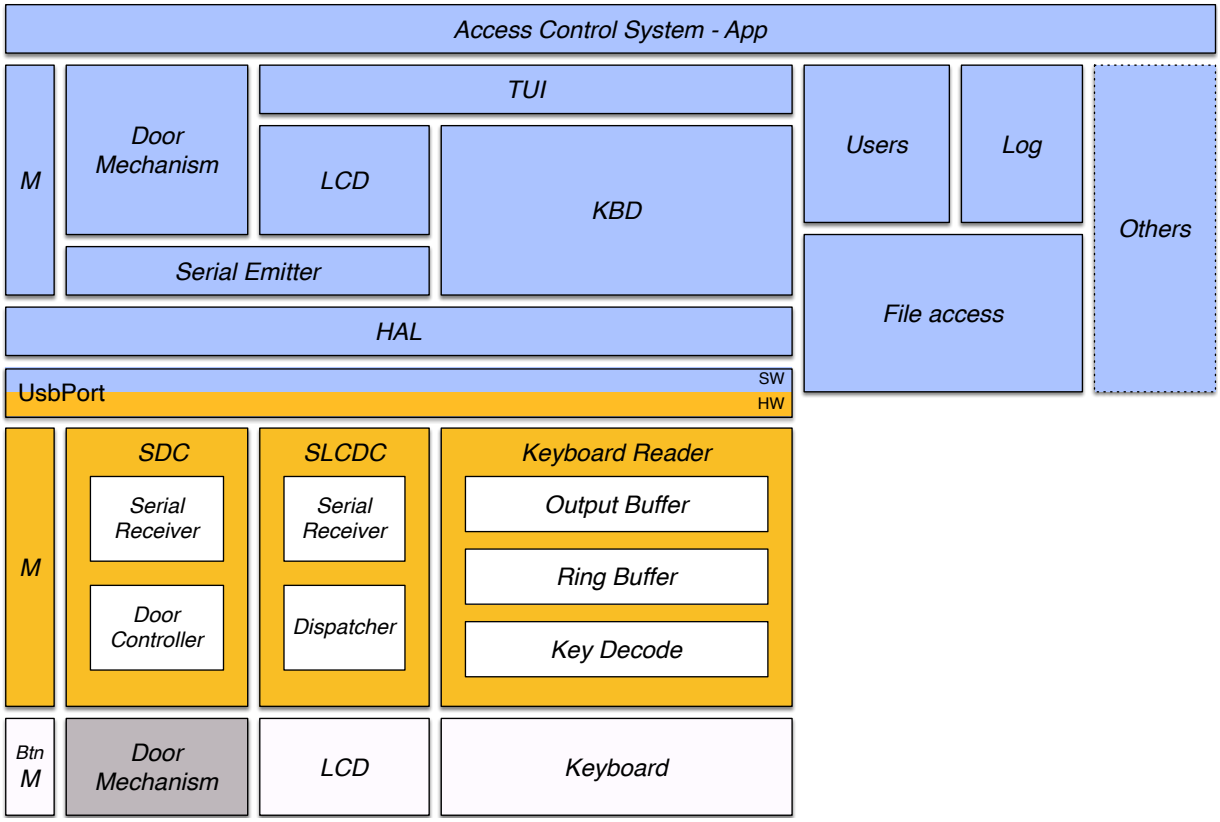


Figura 13 – Diagrama lógico do sistema de controlo de acessos (*Access Control System*)

As assinaturas das principais funções 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() ...
   // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
   fun getKey(): Char ...
   // Retorna a tecla premida, caso ocorra antes do 'timeout' (representado em milissegundos), ou
   NONE caso contrário.
   fun waitKey(timeout: Long): Char ...
}
```

2.4.3 LCD

```
LCD {  // Escreve no LCD usando a interface a 4 bits.
   private const val LINES = 2, COLS = 16; // Dimensão do display.
   // Escreve um nibble de comando/dados no LCD em paralelo
   private fun writeNibbleParallel(rs: Boolean, data: Int) ...
   // Escreve um nibble de comando/dados no LCD em série
   private fun writeNibbleSerial(rs: Boolean, data: Int) ...
   // Escreve um nibble de comando/dados no LCD
   private fun writeNibble(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 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

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

```
DoorMechanism {       // Controla o estado do mecanismo de abertura da porta.
   // Inicia a classe, estabelecendo os valores iniciais.
   fun init() ...
   // Envia comando para abrir a porta, com o parâmetro de velocidade
   fun open(velocity: Int) ...
   // Envia comando para fechar a porta, com o parâmetro de velocidade
   fun close(velocity: Int) ...
   // Verifica se o comando anterior está concluído
   fun finished() : Boolean ...
}
```

3 Calendarização do projeto

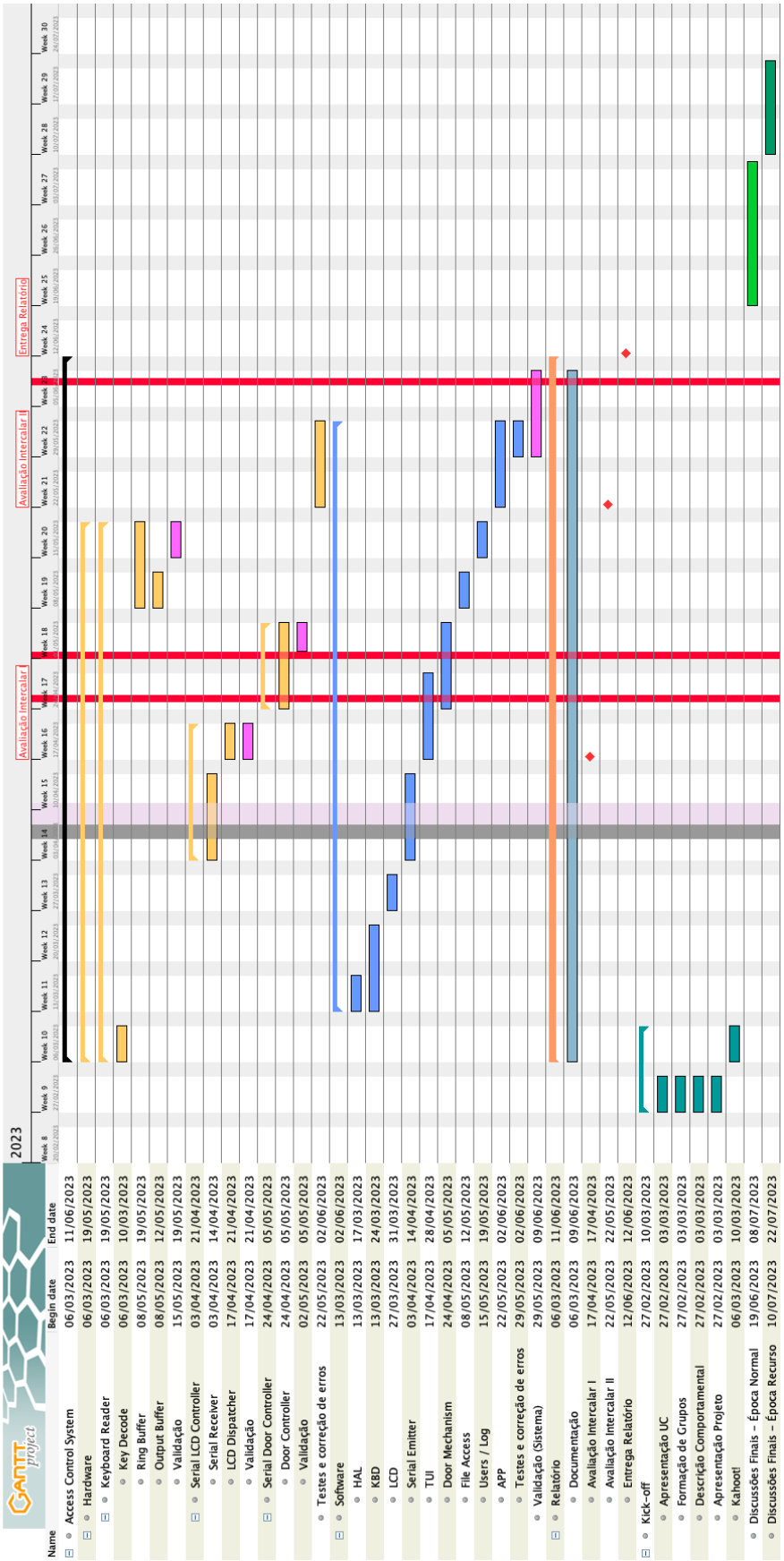


Figura 14 – Diagrama de *Gantt* relativo à calendarização do projeto