Licenciatura em Engenharia Informática e Computadores

# Laboratório de Informática e Computadores

Semestre de Verão 2022/2023


## Entrega Final

## Sistema de Controlo de Acessos

Grupo 7:

André Brito - Nº 50487

João Cardoso - Nº 50465

Tiago Gonçalves - Nº 50474

# ÍNDICE

# Alvo de Avaliação

Neste trabalho pretendia-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.

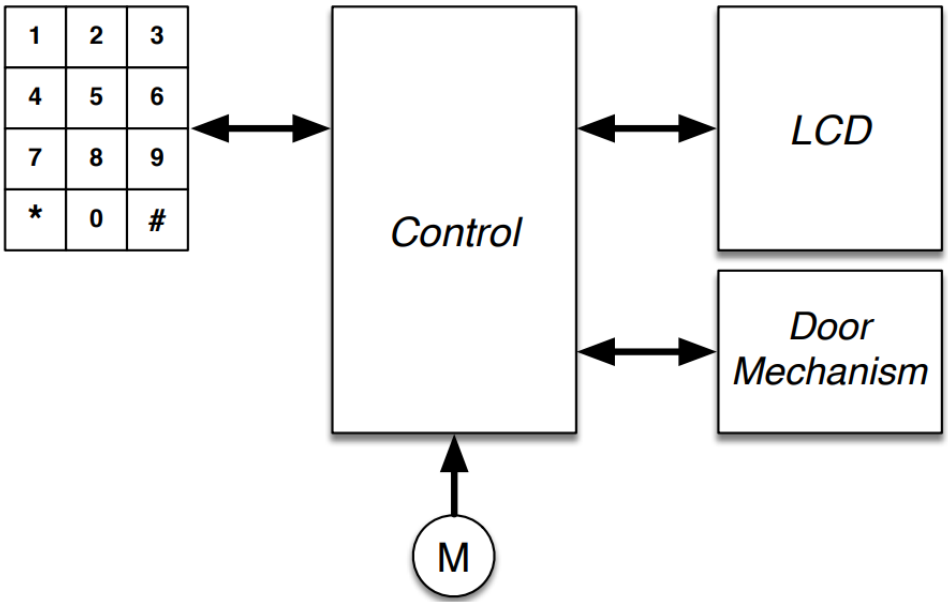


Figura 1 – *Access Control System*

## Modos de Funcionamento

Este sistema tem dois modos de funcionamento, sendo um deles o modo de *Acesso*, havendo a possibilidade de realizar duas ações distintas:

**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.

O segundo modo de funcionamento diz respeito ao modo de *Manutenção* realizado a partir de um PC e que permite quatro ações:

**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.

# Hardware

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.

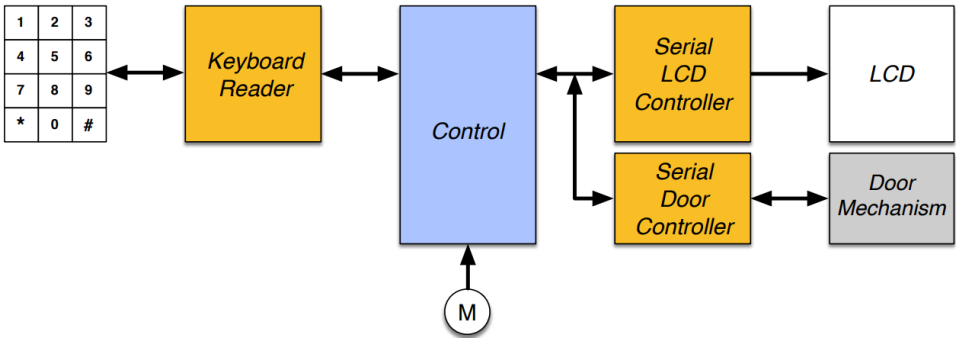


Figura 2 – Arquitetura do *Access Control System*

A arquitetura proposta é constituída por quatro módulos principais:

**Keyboard Reader** – leitor de teclado.

**Serial LCD Controller** - módulo de interface com o LCD.

**Serial Door Controller** - módulo de interface com o mecanismo da porta.

**Control** - módulo de software de controlo.

## Keyboard Reader

O módulo *Keyboard Reader* (Figura 3) 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.

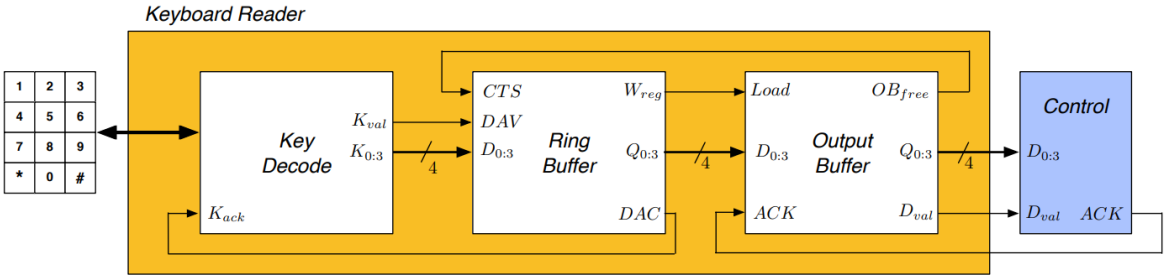


Figura 3 – Diagrama de blocos do *Keyboard Reader*

Este componente é constituído por três blocos principais:

**Key Decode** - descodificador de um teclado matricial 4x3 por hardware.

**Ring Buffer** - estrutura de dados para armazenamento de teclas com disciplina FIFO *(First In First Out*) com capacidade de armazenar até oito palavras de quatro bits.

**Output Buffer** - responsável pela interação com o sistema consumidor, neste caso o módulo *Control*.

## Key Decode

Para descodificar o teclado matricial, o *Key Decode* (Figura 4) é constituído por quatro blocos, sendo um deles o próprio teclado.

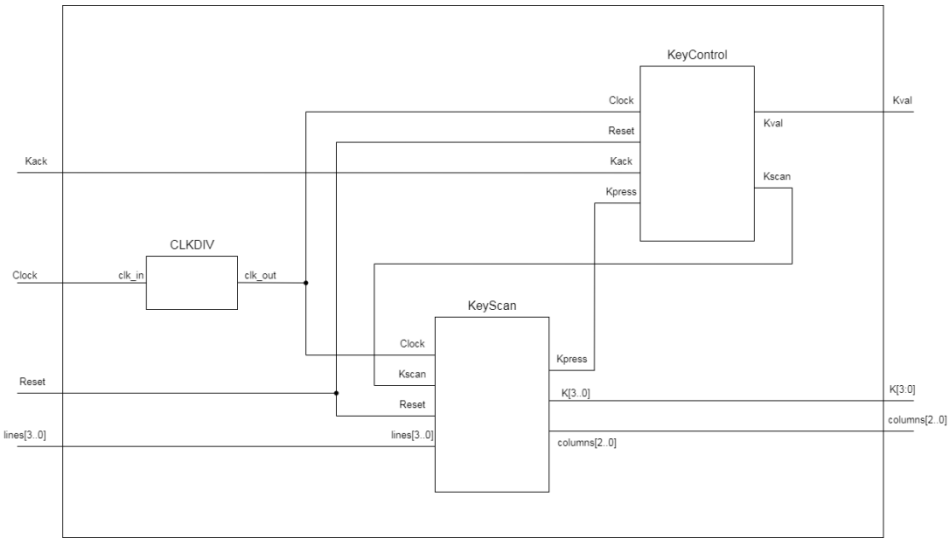


Figura 4 – Diagrama de blocos do *Key Decode*

O bloco *CLK_DIV* foi acrescentado de modo a sincronizar o *Key Decode* com os restantes componentes e foi configurado de maneira a dividir o sinal de *clock* por 1000 vezes.

O controlo de fluxo de saída do bloco *Key Decode* (para o bloco Ring Buffer) define que o sinal *Kval* é ativado quando é detetada a pressão de uma tecla, sendo também disponibilizado o código dessa tecla no barramento *K0:3*. Apenas é iniciado um novo ciclo de varrimento ao teclado quando o sinal Kack for ativado e a tecla premida for libertada. O diagrama temporal do controlo de fluxo está representado na Figura 5.

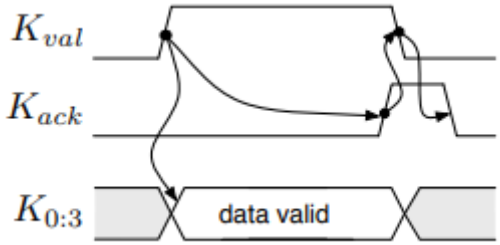


Figura 5 – Diagrama temporal das saídas do bloco *Key Decode*

O bloco Key Scan foi implementado de acordo com o diagrama de blocos representado na Figura 6, e o bloco *Key Control* é uma máquina de estados com o ASM presente na Figura 7.

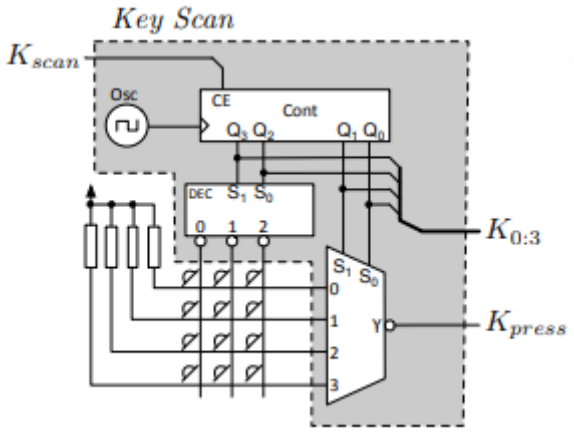


Figura 6 – Diagrama de blocos do *Key Scan*

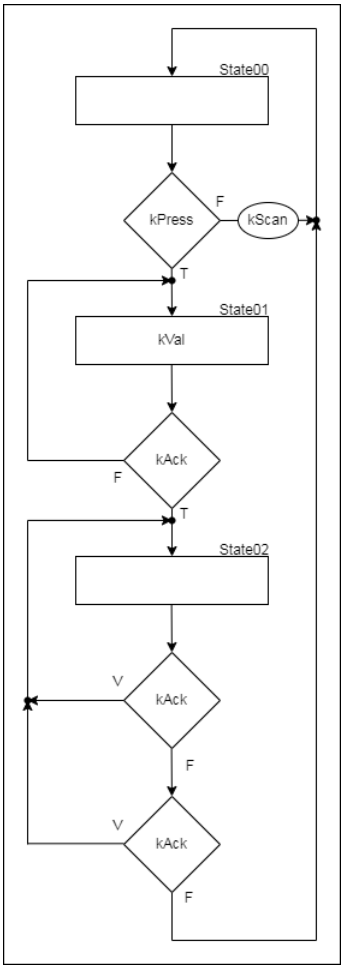


Figura 7 – ASM do *Key Control*

## Ring Buffer

O bloco Ring Buffer (Figura 8) desenvolvido é 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 é baseada numa memória RAM (*Random Access Memory*). O endereço de escrita/leitura, selecionado por putGet, 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). Para obter este efeito, foi implementada uma máquina de estados com o ASM apresentado na Figura 9.

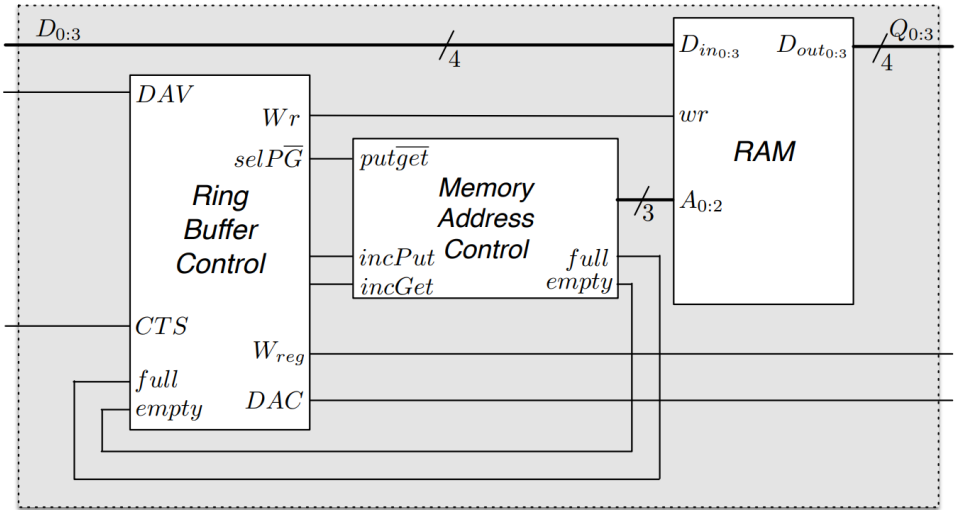


Figura 8 – Diagrama de blocos do *Ring Buffer*

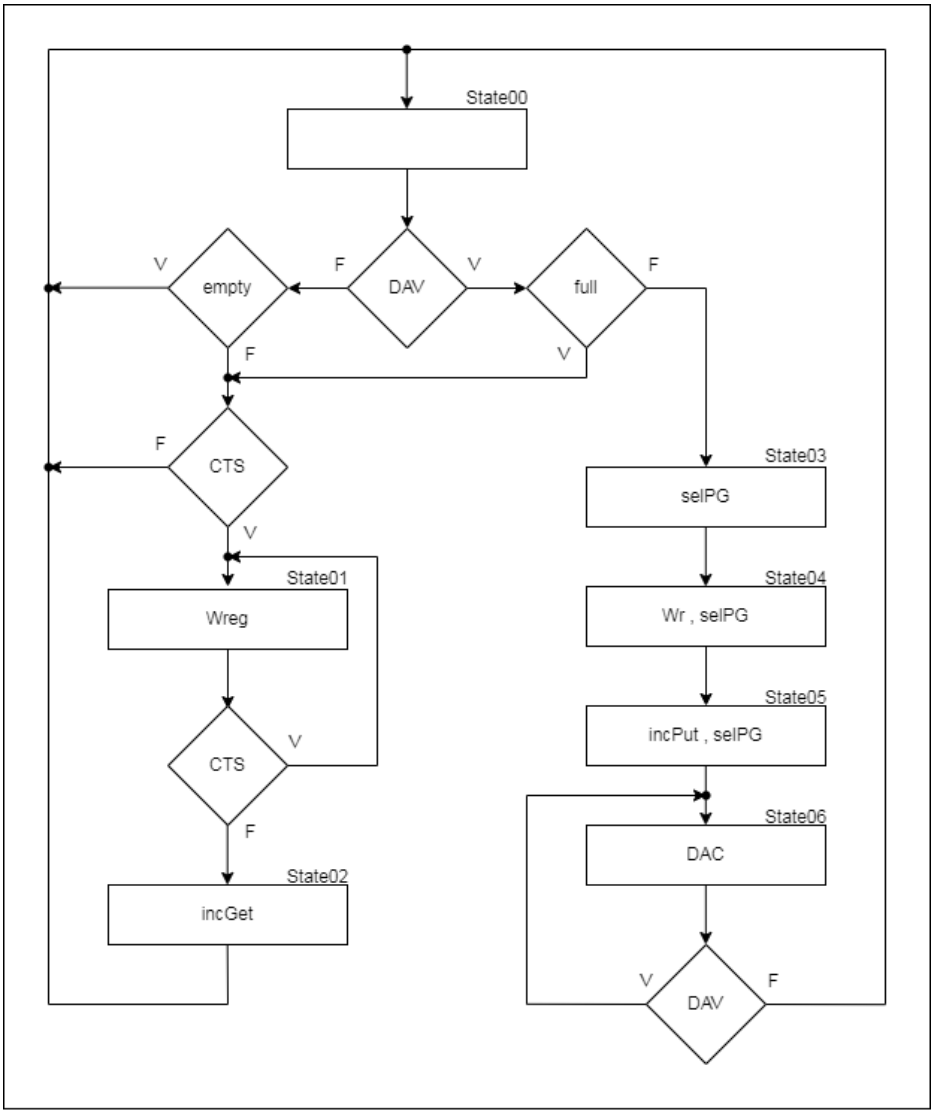


Figura 9 – ASM do *Ring Buffer Control*

Como podemos ver pela Figura 10, o bloco *Memory Address Control* é constituído por 3 contadores e um *Multiplexer* (Mux). O valor de cada um dos dois contadores simples será colocado no Mux cujo selecionador é o sinal putGet, para que à saída deste componente esteja o valor de escrita/leitura. Quanto ao terceiro contador, cuja diferença para os restantes é ter a capacidade de decrementar com a ativação do sinal UD (*UpDown*), serve para verificar quantos valores estão guardados na ROM, para então ativar ou desativar os sinais de *full* ou *empty*.

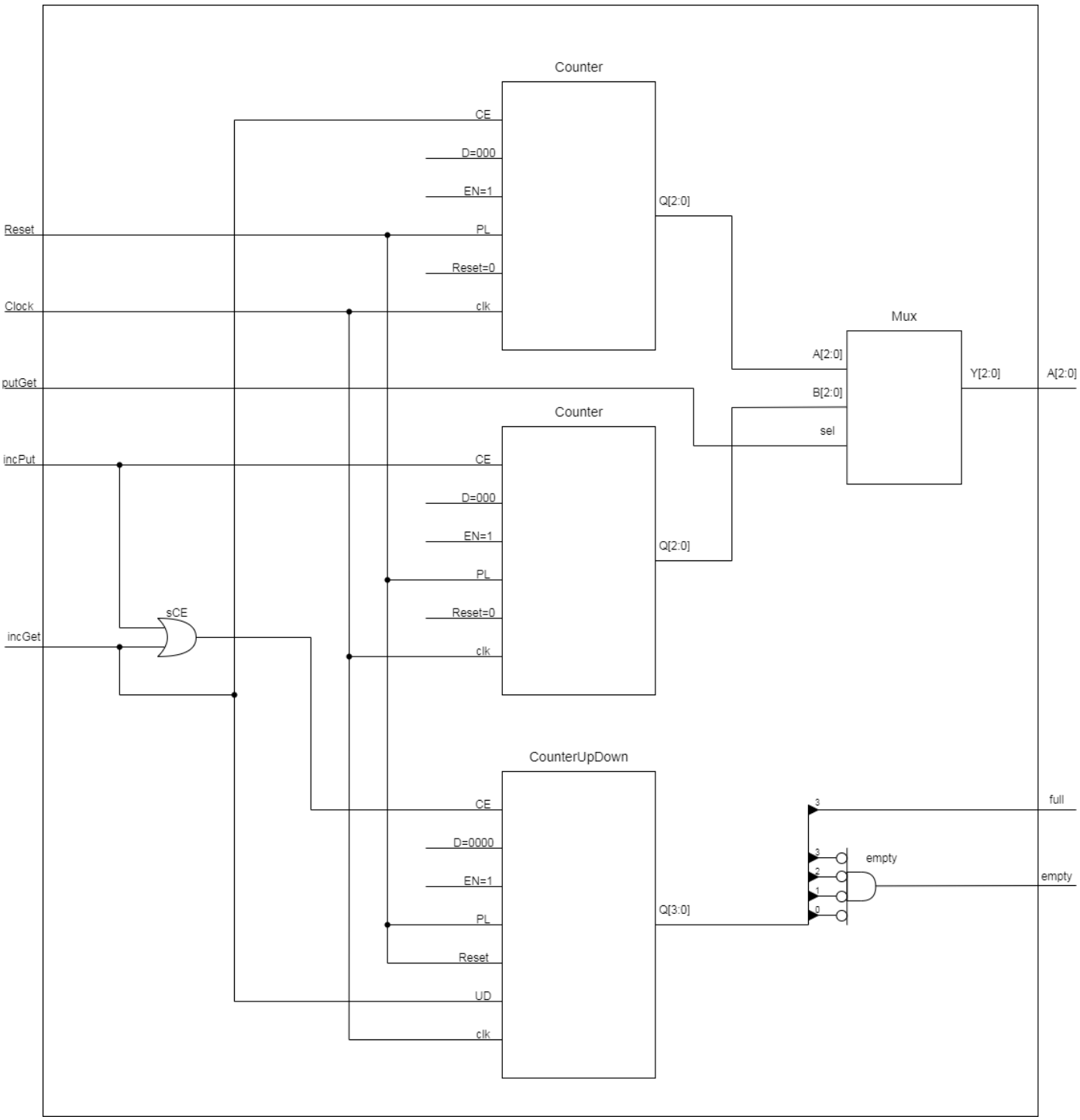


Figura 10 – Diagrama de blocos do *MAC*

## Output Buffer

O Output Buffer (Figura 11) 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 Dval 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 Dval e sinalizar que está novamente disponível para entregar dados ao sistema consumidor, ativando o sinal $OB_{free}$. 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 $OB_{free}$.

Na Figura 12 está o ASM da máquina de estados implementada que trata da ativação e desativação destes sinais.

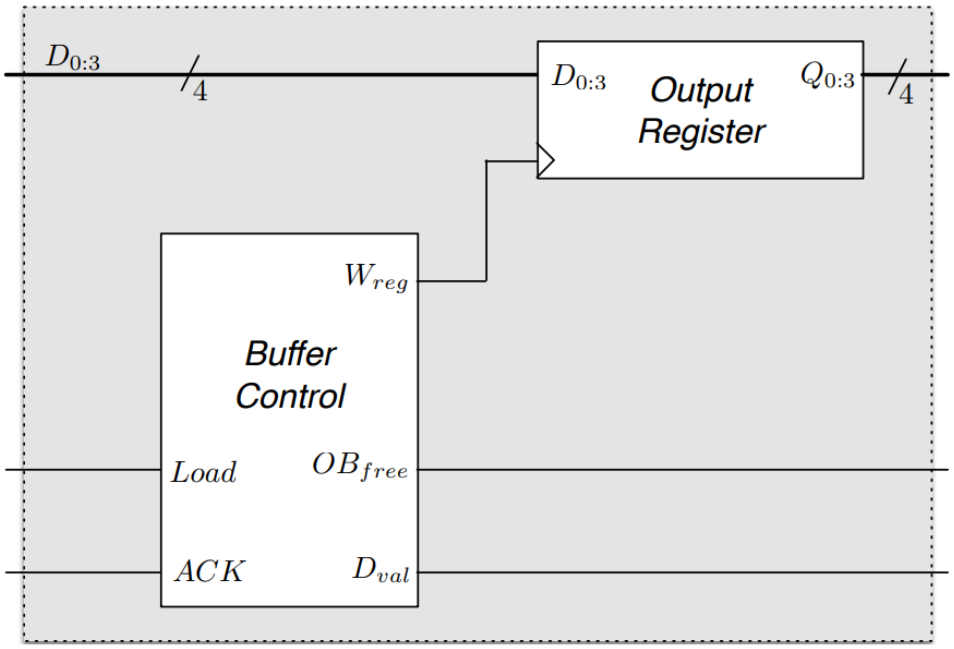


Figura 11 – Diagrama de blocos do *Output Buffer*

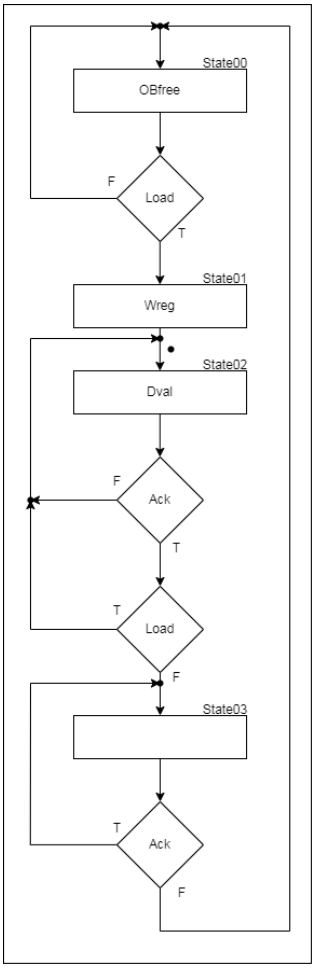


Figura 12 – ASM do *Buffer Control*

## 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 13.

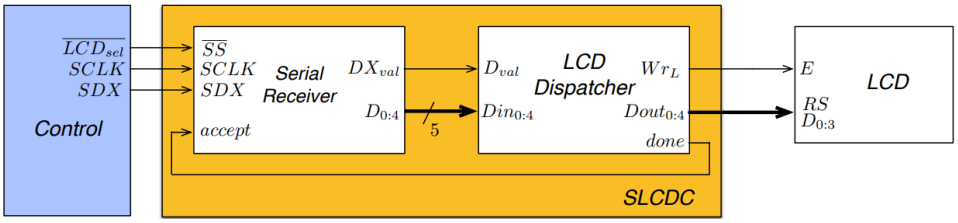


Figura 13 – Diagrama de blocos do *Serial LCD Controller*

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 14, 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.

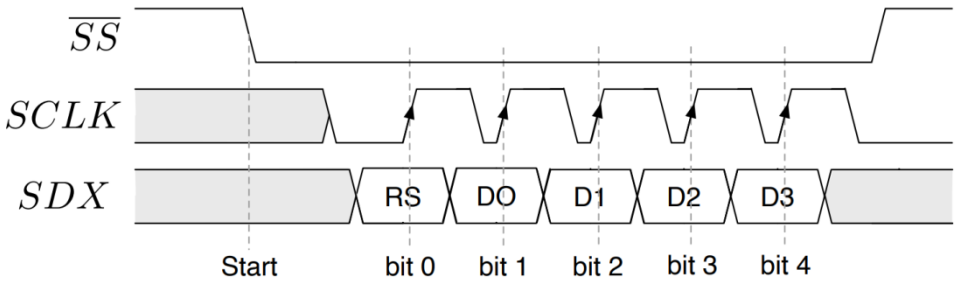


Figura 14 – Protocolo de comunicação com Serial LCD Controller

## Serial Receiver

O bloco Serial Receiver (Figura 15) do SLCDC é constituído por três blocos principais:

**Serial Control** – uma máquina de estados (Figura 16).

**Counter** – um contador de bits recebidos.

**Shift Register** – conversor de tramas em série para paralelo.

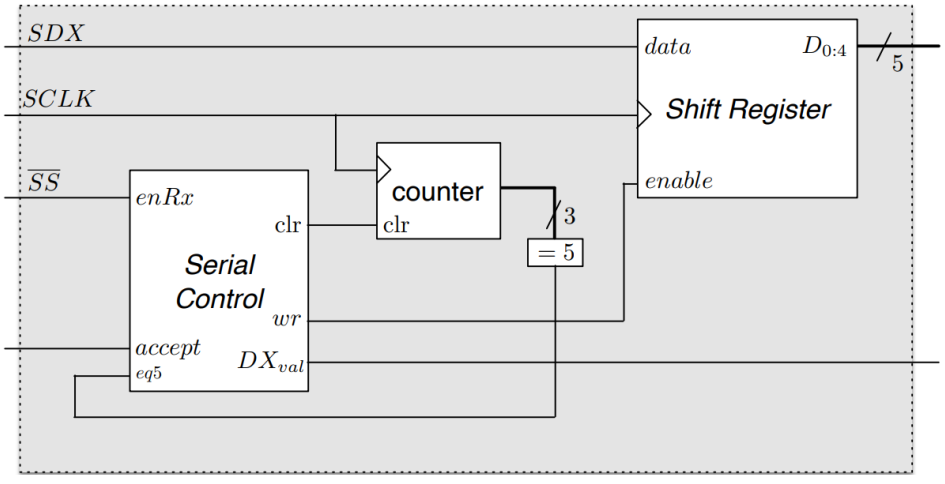


Figura 15 – Diagrama de blocos do *Serial Receiver*

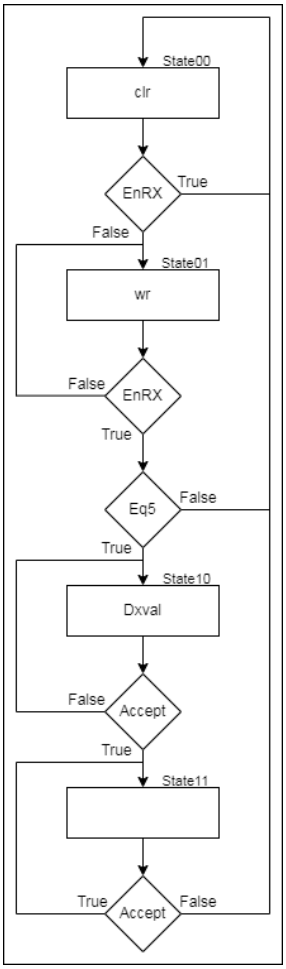


Figura 16 – ASM do *Serial Control*

## Serial 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. A máquina de estados do LCD Dispatcher está representada na Figura 17.

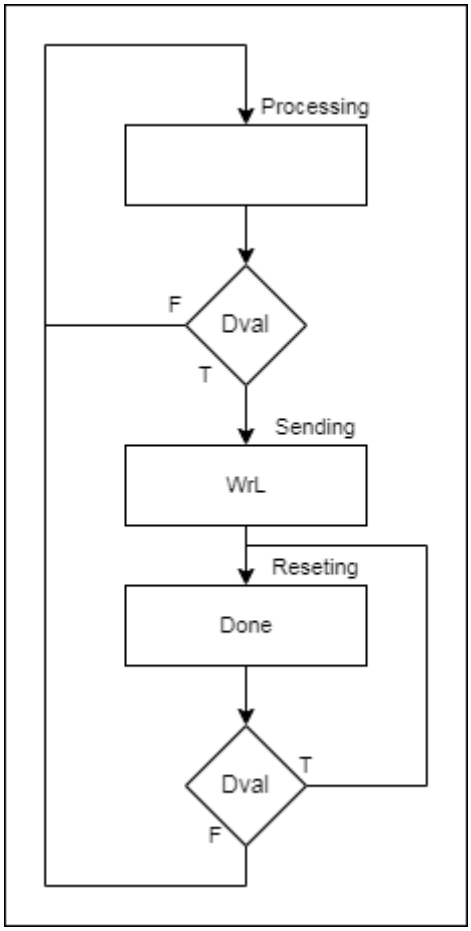


Figura 17 – ASM do *LCD Dispatcher*

## 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 18.

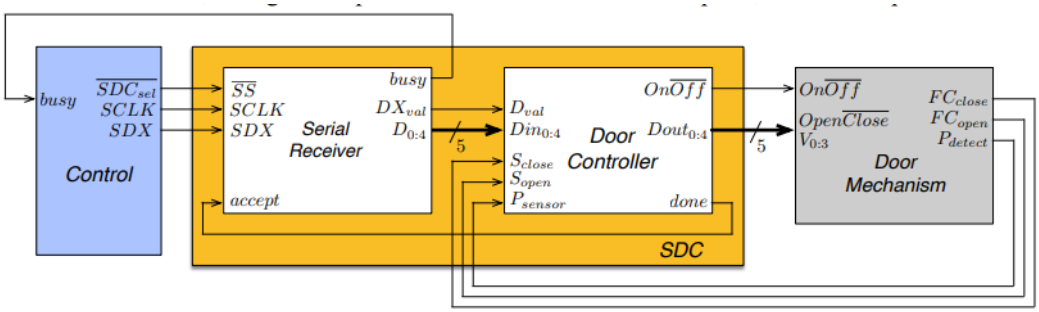


Figura 18 – Diagrama de blocos do *Serial Door Controller*

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 18,

tendo como primeiro bit de informação, o bit openClose (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 com o valor lógico '0'.

O bloco Serial Receiver do *SDC* é exatamente igual ao Serial Receiver do *Serial LCD Controller*.

## Door Controller

O bloco Door Controller (Figura 19), 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 onOff e sinal openClose no valor lógico '1', até o sensor de porta aberta (FCopen) ficar ativo. No entanto, se o comando for de fecho, o Door Controller deverá ativar o sinal onOff e colocar o sinal openClose no valor lógico '0', até o sensor de porta fechada (FCclose) ficar ativo. Se durante o fecho for detetada uma pessoa na zona da porta, através do sensor de presença (Pdetect), 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.

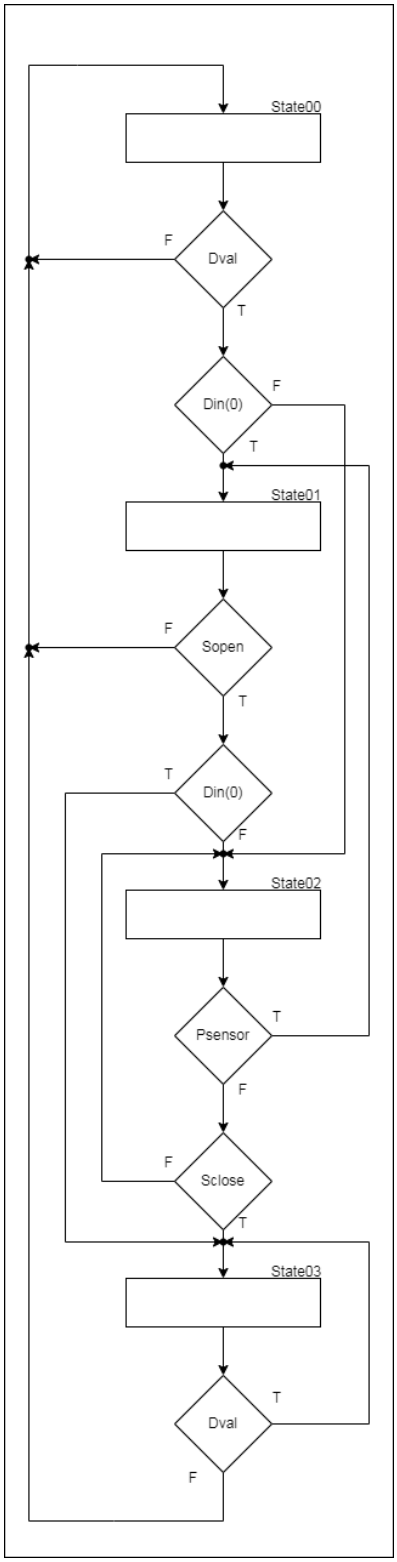


Figura 19 – ASM do *Door Controller*

# Software

## HAL

```
object HAL { // Virtualiza o acesso ao sistema UsbPort
    var x = 0

    // Inicia a classe
    BuildTools
    fun init() {

        x = 0
        UsbPort.write(x)

    }

    // Retorna true se o bit tiver o valor lógico '1'
    BuildTools
    fun isBit(mask: Int): Boolean {
        val res : Int = UsbPort.read() and mask
        return res != 0
    }

    // Retorna os valores dos bits representados por mask presentes no UsbPort
    BuildTools
    fun readBits(mask: Int): Int {
        return UsbPort.read() and mask
    }
```

```
// Escreve nos bits representados por mask o valor de value
BuildTools
fun writeBits(mask: Int, value: Int) {
    x = (value and mask) or (x and mask.inv())
    UsbPort.write(x)
}

// Coloca os bits representados por mask no valor lógico '1'
BuildTools
fun setBits(mask: Int) {
    x = x or mask
    UsbPort.write(x)
}

// Coloca os bits representados por mask no valor lógico '0'
BuildTools
fun clrBits(mask: Int) {
    x = x and mask.inv()
    UsbPort.write(x)
}
```

## KBD

```
object KBD { // Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.
    const val NONE = 0;
    val KACK_MSK = 0x80
    val KVAL_MSK = 0x01
    val KMSK = 0x1E
    val digits = arrayOf('1', '4', '7', '*', '2', '5', '8', '0', '3', '6', '9', '#')

    // Inicia a classe
    BuildTools +1
    fun init() {
        println("Initializing KBD")
    }
// Retorna de imediato a tecla premida ou NONE se não há tecla premida.
BuildTools +1
fun getKey(): Char { // Executa a rotina de ler a key, de acordo com o diagrama temporal
    val key : Int = HAL.readBits(KMSK)

    if (HAL.isBit(KVAL_MSK)) {
        HAL.setBits(KACK_MSK)
        while (HAL.isBit(KVAL_MSK)) {
        }
        HAL.clrBits(KACK_MSK)
        return digits[key.shr( bitCount: 1)]  // Vai buscar o digito em questão ao array de digitos
    }
    return NONE.toChar()
}

// Retorna a tecla premida, caso ocorra antes do 'timeout' (representado em milissegundos)
// ou NONE caso contrário.
BuildTools +1 *
fun waitKey(timeout: Long): Char {
    val t : Long = Time.getTimeInMillis()  // Valor atual de tempo
    // Enquanto o tempo passado for inferior ao timeout, vai se buscar a key.
    while (Time.getTimeInMillis() - t < timeout) {
        val key : Char = getKey()
        if (key != NONE.toChar()) return key
    }
    return NONE.toChar()
}
```

## LCD

```
const val MSK = 0x0F
var isSerial = true

BuildTools +1
object LCD {

    // Escreve um nibble de comando/dados no LCD em paralelo
    BuildTools
    private fun writeNibbleParallel(rs: Boolean, data: Int) {
        if (rs) HAL.setBits(0x10)
        else HAL.clrBits( mask: 0x10)
        HAL.setBits(0x20)
        HAL.writeBits(MSK, data)
        HAL.clrBits( mask: 0x20)
    }
```

```
// Escreve um nibble de comando/dados no LCD em série
BuildTools +1
private fun writeNibbleSerial(rs: Boolean, data: Int) {
    if (rs) {
        val shifted : Int = data.shl( bitCount: 1)
        SerialEmitter.send(SerialEmitter.Destination.LCD, data: shifted or 1)
    } else {
        val shifted : Int = data.shl( bitCount: 1)
        SerialEmitter.send(SerialEmitter.Destination.LCD, shifted)

    }

}


// Escreve um nibble de comando/dados no LCD
BuildTools +1
private fun writeNibble(rs: Boolean, data: Int) {
    if (isSerial) {
        writeNibbleSerial(rs, data)
    } else writeNibbleParallel(rs, data)
}
```

```
// Escreve um byte de comando/dados no LCD
BuildTools
private fun writeByte(rs: Boolean, data: Int) {
    writeNibble(rs, data.shr( bitCount: 4))
    writeNibble(rs, data)
}

BuildTools
private fun writeCMD(data: Int) {
    writeByte( rs: false, data)
}

BuildTools
private fun writeDATA(data: Int) {
    writeByte( rs: true, data)
}
```

```
fun init() {
    Time.sleep( milis: 15)
    writeNibble( rs: false,  data: 0x03)
    Time.sleep( milis: 5)
    writeNibble( rs: false,  data: 0x03)
    writeNibble( rs: false,  data: 0x03)
    writeNibble( rs: false,  data: 0x02)
    writeCMD( data: 0x28)
    writeCMD( data: 0x08)
    writeCMD( data: 0x01)
    writeCMD( data: 0x06)
    writeCMD( data: 0x0f)
}
```

```
// Escreve um caráter na posição corrente.
BuildTools
fun write(c: Char) {
    writeDATA(c.code)
}
// Escreve uma string na posição corrente.
CardosoDev04 +1
fun write(text: String) {
    for (c : Char in text) {
        write(c)
    }
}
// Envia comando para posicionar cursor ('line':0..LINES-1 , 'column':0..COLS-1)
CardosoDev04 +1
fun cursor(line: Int, column: Int) {
    val data : Int = (0x40 * line + 0x80)  + column
    println(data)
    writeByte( rs: false, data)
}
// Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
BuildTools +1
fun clear() {
    writeByte( rs: false,  data: 1)
}
```

## SEM

```
const val SS = 0x20
const val SDX = 0x01
const val SCLK = 0x40
const val sLCD = 0x02
const val sDOOR = 0x04


BuildTools +1
object SerialEmitter { // Envia tramas para os diferentes módulos Serial Receiver.
    BuildTools
    enum class Destination { LCD, DOOR }


    // Inicia a classe
    BuildTools +1
    fun init() {
        HAL.init()
        HAL.setBits(sLCD)
        HAL.setBits(sDOOR)

    }
```

```
// Envia uma trama para o SerialReceiver identificado o destino em addr e os bits de dados em data.
BuildTools +1
fun send(addr: Destination, data: Int) {

    val mask : Int = if (addr == Destination.LCD) sLCD else sDOOR

    HAL.clrBits(mask)
    HAL.clrBits(SCLK)

    for (i : Int in 0 ≤ .. ≤ 4) {
        HAL.clrBits(SCLK)
        HAL.writeBits(SDX, data.shr(i))
        HAL.setBits(SCLK)
        Time.sleep( milis: 1)
    }
    HAL.clrBits(SCLK)
    HAL.setBits(mask)


}

// Retorna true se o canal série estiver ocupado
BuildTools
fun isBusy(): Boolean {
    return HAL.isBit(SS)
}
```

## DMEC

```
object Doormechanism { // Controla o estado do mecanismo de abertura da porta.

    var OC = 0

        // Inicia a classe, estabelecendo os valores iniciais.
        BuildTools
        fun init() {
            OC = 0
        }

    //1 + velocity or 0 + velocity

        // Envia comando para abrir a porta, com o parâmetro de velocidade
        BuildTools +1
        fun open(velocity: Int) {
            OC = 1
            var cmd : Int = velocity.shl( bitCount: 1)
            SerialEmitter.send(SerialEmitter.Destination.DOOR, data: cmd or OC)  // Envia 1velocity

        }
```

```
// Envia comando para fechar a porta, com o parâmetro de velocidade
BuildTools +1
fun close(velocity: Int) {
    var cmd : Int = velocity.shl( bitCount: 1)
    SerialEmitter.send(SerialEmitter.Destination.DOOR, data: cmd or OC) //Envia 0velocity
}

// Verifica se o comando anterior está concluído
BuildTools +1
fun finished() : Boolean {
    return !SerialEmitter.isBusy()  // Se não está ocupado, está terminado.
}
```

## TUI

```
object TUI {
    //Função que lê o input e transforma num inteiro para comparar às credenciais confiadas.
    fun getInt(qty: Int, line: Int, isHidden: Boolean): Int {
        var array : Array<Int> = emptyArray<Int>()
        var clearInput = false //Flag to track whether the input should be cleared

        for (i : Int in 1 ≤ .. ≤ qty) {
            val key : Int = KBD.waitKey( timeout: 5000).code
            if (key != 0.toChar().code && key != '*'.code && key != '#'.code) {
                array += key - 48
                if(!isHidden ) {
                    LCD.write(key.toChar())
                }
                else{
                    LCD.write( text: "*")
                }
            }
            else if (key == '*'.code) {
                clearInput = true // Set the flag to clear the input
                break
            } else {
                loginRoutine()
            }
        }
```

```
        if (clearInput) {
            LCD.cursor(line,  column: 4)
            when (qty) {
                3 -> LCD.write( text: "???")
                4 -> LCD.write( text: "????")
            }
            LCD.cursor(line,  column: 4)
            return getInt(qty, line, isHidden) // Recursive call to get a new input
        }

        return array.joinToString( separator: "").toInt()
    }
```

```
//Função que faz get da Data e Hora e formata-as para o tempo correto.
fun getTimeAndDate(): String{
    while(true){
        val currentTime : LocalTime! = LocalTime.now()
        val formatter : DateTimeFormatter! = DateTimeFormatter.ofPattern( pattern: "HH:mm")
        val formattedTime : String! = currentTime.format(formatter)

        val currentDate : LocalDate! = LocalDate.now()
        val formatter1 : DateTimeFormatter! = DateTimeFormatter.ofPattern( pattern: "dd/MM/yyyy")
        val formattedDate : String! = currentDate.format(formatter1)

        return "$formattedDate $formattedTime"
    }
}
```

```
//Executa a rotina de login.
fun loginRoutine() {
    Doormechanism.close( velocity: 15)
    if(Maintenance.inMaintenance()) {
        Time.sleep( milis: 500)
        LCD.clear()
        LCD.write( text: "Out of service")
        Maintenance.maintenanceRoutine()
        println("Maintenance mode is: " + Maintenance.inMaintenance())
    }
    LCD.clear()
    Time.sleep( milis: 1000)
    LCD.clear()
    LCD.write(getTimeAndDate())
    LCD.cursor( line: 1, column: 0)
    LCD.write( text: "UIN=???")
    LCD.cursor( line: 1, column: 4)
    val id : Int = getInt( qty: 3, line: 1, isHidden: false)
    LCD.clear()
    LCD.write(getTimeAndDate())
    LCD.cursor( line: 1, column: 0)
    LCD.write( text: "PIN=????")
    LCD.cursor( line: 1, column: 4)
    val pin : Int = getInt( qty: 4, line: 1, isHidden: true)
    LCD.clear()
    App.mainLog(id, pin)
}
```

```
fun main() {
    Doormechanism.close( velocity: 15)
    USERS.clearMap()
    USERS.defineMap()
    println(TUI.getTimeAndDate())
    HAL.init()
    SerialEmitter.init()
    LCD.init()
    KBD.init()
    println("Maintenance mode is: " + Maintenance.inMaintenance())
    TUI.loginRoutine()
}
```

## APP

```
var invalidLog = false // Variável que é alterada para true se o login for inválido
const val M_MSK = 0x80
const val NONE = 0;

object App {

    //Executa a main routine de login, com todas as funções para o login.
    fun mainLog(id: Int, pin: Int) {

        val trusted : Boolean = login(id, pin)
        openDoor(trusted,id)
        if (invalidLog) TUI.loginRoutine()
    }

    //Verifica se o ID e PIN inseridos condizem com os do utilizador confiado
    fun login(id: Int, pin: Int): Boolean {
        return  pin == USERS.getUserPin(id)
    }
```

```
//Se o utilizador for o confiado, abre a porta caso contrario fecha e dá Error
fun openDoor(isTrusted: Boolean,id: Int) {
    if (isTrusted) {
        Doormechanism.open( velocity: 15)
        LCD.write( text: "Welcome,")
        LCD.cursor( line: 1, column: 0)
        LCD.write( text: "${USERS.getUsername(id)}")
        Time.sleep( milis: 2000)
        LCD.clear()
        if(USERS.getPhrase(id)?.isNotEmpty() == true && USERS.getPhrase(id) != null && USERS.getPhrase(id) != ""){
            LCD.write( text: "${USERS.getPhrase(id)}")
        }
        if(USERS.getPhrase(id) == "") {
            LCD.clear()
            Doormechanism.close( velocity: 15)
            TUI.loginRoutine()
        }
        var entry : String = createLogEntry(id)
        appendEntry(entry)
```

```
        var key : Char = NONE.toChar()
        val initialTime : Long = Time.getTimeInMillis()
        while(key == NONE.toChar() && Time.getTimeInMillis() - initialTime < 5000) {
            key = KBD.getKey()
        }
        if(key == '#') {
            replacePINRoutine(id)
        }
        else if(key == '*'){
            USERS.removePhrase(id)
        }
        invalidLog = false
        TUI.loginRoutine()
    }

    else {
        Doormechanism.close( velocity: 15)
        LCD.write( text: "Invalid Login")
        invalidLog = true
        Time.sleep( milis: 1500)
        LCD.clear()
    }
```

```
//Cria uma entrada do tipo Log com a data e hora atual
fun createLogEntry(id: Int): String{
    val timeDate : String = TUI.getTimeAndDate()
    return "$timeDate:UIN $id"
}

//Adiciona a entrada ao ficheiros LOGS
fun appendEntry(entry: String){
    val logs = File( pathname: "LOGS.txt")
    FileWriter(logs,  append: true).use { writer -> writer.write( str: "$entry\n")}

}
```

```
//Rotina de alteração do PIN
fun replacePINRoutine(id: Int){
    LCD.clear()
    LCD.write( text: "PIN=????")
    LCD.cursor( line: 0, column: 4)
    val newPIN : Int = TUI.getInt( qty: 4, line: 0,  isHidden: true)
    LCD.clear()
    LCD.write( text: "Confirm your PIN")
    Time.sleep( milis: 3000)
    LCD.clear()
    LCD.write( text: "PIN=????")
    LCD.cursor( line: 0, column: 4)
    val newPINConfirm : Int = TUI.getInt( qty: 4, line: 0,  isHidden: true)

    if(newPIN == newPINConfirm) {
        USERS.replacePIN(id, newPIN)
    }
    else{
        LCD.clear()
        LCD.write( text: "Mismatch")
        Time.sleep( milis: 2000)
        replacePINRoutine(id)
    }
}
```

## Maintenance

```
val maintenanceBox = """
    +------------------+
    | MAINTENANCE MODE |
    +------------------+
""".trimIndent()


object Maintenance {

    fun getBiggestUIN(): Int? {
        var maxUIN: Int? = null

        for (entry : MutableMap.MutableEntry<Int, Triple<Int, String, String>> in USERS.userMap.entries) {
            val uin : Int = entry.key
            if (maxUIN == null || uin > maxUIN) {
                maxUIN = uin
            }
        }

        return maxUIN
    }
```

```
//Entra em modo de manutenção.
fun inMaintenance(): Boolean{
    return when(HAL.isBit(M_MSK)){
        true -> true
        else -> false
    }
}
```

```
//Inicia a rotina de inserção de utilizador
fun insertUser(){

    var UIN: Int? = 0
    var PIN: Int = 0
    var name: String = ""

    println("Please insert the PIN code")
    val tryPIN : Int = readln().toInt()
    if( tryPIN <= 9999){
        PIN = tryPIN
    }
    else{
        println("Please insert a 4 digit number, reloading...")
        Time.sleep( milis: 2000)
        insertUser()
    }
```

```
    println("Please insert the user's name")
    val tryName : String = readln()
    val pattern = Regex( pattern: "^[A-Za-z]+(?: [A-Za-z]+)?\$")
    if(tryName.length <= 16 && tryName.matches(pattern)){
        name = tryName
    }
    else{
        println("Please insert a username with only A-Z And Spaces")
        Time.sleep( milis: 2000)
        insertUser()
    }

    UIN = getBiggestUIN()?.plus( other: 1)
    val entry = "$UIN;$PIN;$name;"
    val file = File( pathname: "USERS.txt")
    FileWriter(file,  append: true).use { writer -> writer.write( str: "$entry\n")}

    println("User with name: $name and UIN $UIN was added")
    maintenanceRoutine()
}
```

```
//Inicia a rotina de remoção de utilizador
fun removeUser(){
    var UIN = 0
    println("Insert the UIN of the User you wish to remove.")
    val tryUIN : Int = readln().toInt()
    if(tryUIN <= 999){
        UIN = tryUIN
    }
    else{
        println("Please insert a 3 digit number, reloading...")
        Time.sleep( milis: 2000)
        removeUser()
    }
```

```
    println("Do you really wish to remove? (Y / N)")
    val userInput : String = readln()
    when(userInput){
        "y" -> {
            USERS.userMap.remove(UIN)
            USERS.defineMap()

            maintenanceRoutine()
        }
        "Y" -> {
            USERS.userMap.remove(UIN)
            USERS.defineMap()

            maintenanceRoutine()
        }
        "n" -> maintenanceRoutine()
        "N" -> maintenanceRoutine()
        else -> {
            println("Invalid Input")
            removeUser()
        }
    }
}
```

```
//Inicia a rotina de inserção de mensagem
fun insertPhrase() {
    var thisUIN = 0
    println("Insert the UIN of the desired user.")
    val tryUIN : Int = readln().toInt()
    if (tryUIN <= 999) {
        thisUIN = tryUIN
    } else {
        println("Please insert a 3-digit number, reloading...")
        Time.sleep( milis: 2000)
        insertPhrase()
    }

    println("Please insert the message you wish to add")
    val phrase : String = readln().toString()
    USERS.defineMap()
    val user : Triple<Int, String, String>? = USERS.userMap[thisUIN]
```

```
    if (user != null) {
        val updatedUser : Triple<Int, String, String> = Triple(user.first, user.second, phrase)
        USERS.userMap[thisUIN] = updatedUser
        USERS.updateDB()
        USERS.defineMap()
        println("Message Added Successfully")
    } else {
        println("User with UIN $thisUIN does not exist.")
    }

    maintenanceRoutine()
}
```

```
fun turnOff(){
    println("Do you confirm you wish to turn off the system? (Y / N)")
    val input : String = readln()

    when(input){
        "y" -> {

            USERS.updateDB()
            USERS.defineMap()
            exitProcess( status: 0)
        }
        "Y" -> {

            USERS.updateDB()
            USERS.defineMap()
            exitProcess( status: 0)
        }
        "n" -> maintenanceRoutine()
        "N" -> maintenanceRoutine()
        else -> {
            println("Invalid input")
            turnOff()
        }
    }
}
```

```
fun exitMaintenance(){
    HAL.init()

    if(!HAL.isBit( mask: 0x80)) {
        TUI.loginRoutine()
    }
    else maintenanceRoutine()
}
```

```
//Executa a rotina principal de manutenção
fun maintenanceRoutine(){
    Doormechanism.close( velocity: 15)
    KBD.isEnabled = false
    println(maintenanceBox)
    println("What would you like to do?")
    println(" 1 - Insert User")
    println(" 2 - Remove User")
    println(" 3 - Add Message to User")
    println(" 4 - Turn the system OFF")


    val optionStr : String = readln()
    if(optionStr != ""){
        when(optionStr.toInt()){
            1 -> insertUser()
            2 -> removeUser()
            3 -> insertPhrase()
            4 -> turnOff()
            else -> maintenanceRoutine()
        }
    }
}
```

## Encriptação/Decriptação

```
object Cryptography {

    fun pinHashing(input: Int?) : Int{
        var output: Int = 0
        val length : Int = input.toString().length
            val firstHalf : String = input.toString().substring(0,length)
            val secondHalf : String = input.toString().substring(length)
            output = (secondHalf + firstHalf).toInt()

        return output
    }

}
```