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

Licenciatura em Engenharia Informática e de Computadores

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

Pedro Miguens Matutino (pedro.miguens@isel.pt)
Diego Passos (diego.passos@isel.pt)
Manuel Carvalho (manuel.carvalho@isel.pt)
Nuno Sebastião (nuno.sebastiao@isel.pt)

Projeto
de
Laboratório de Informática e Computadores
2022 / 2023 verão

04 de abril de 2023

1 Introdução

Neste projeto implementa-se um sistema de controlo de acessos (*Access Control System*), que permite 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.

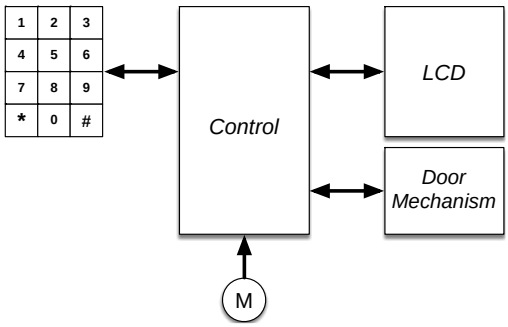

Figura 1 – Sistema de controlo de acessos (*Access Control System*)

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.

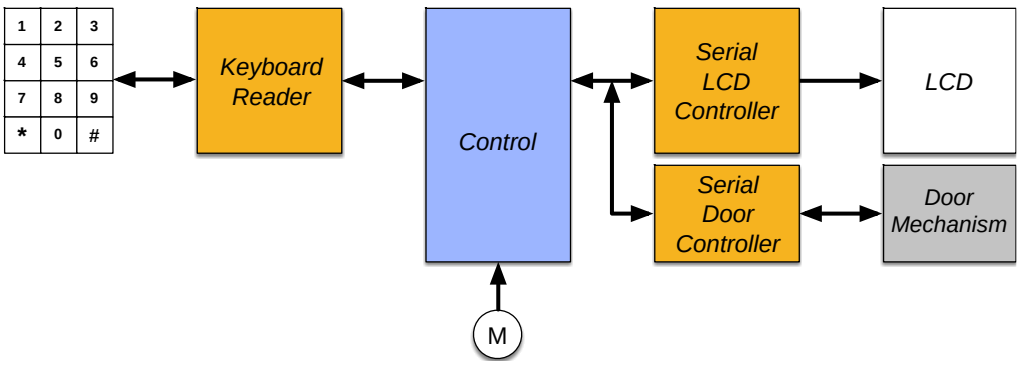

Figura 2 – Arquitetura do sistema que implementa o Sistema de Controlo de Acessos (*Access Control System*)

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.

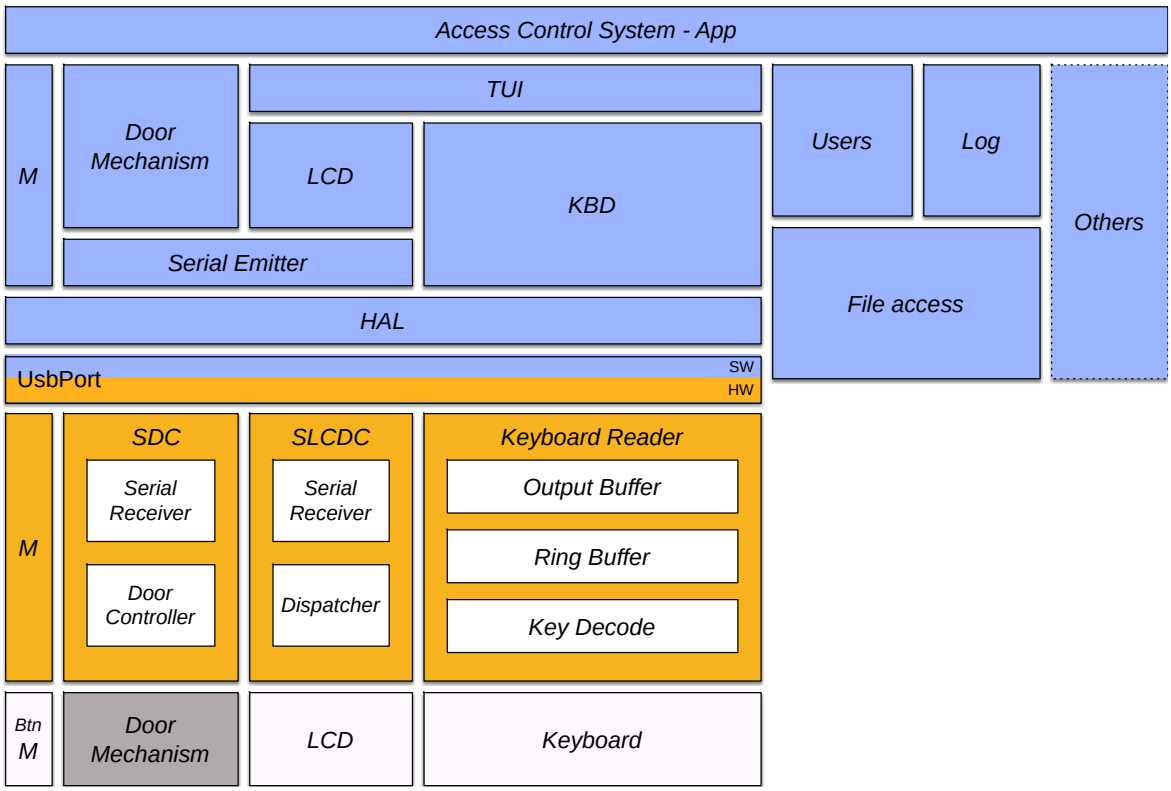

Figura 3 – Diagrama lógico do Sistema de Controlo de Acessos (*Access Control System*)

A. Interligações entre o HW e SW

A interligação entre o Hardware e o Software é feita pelo UsbPort que consiste em ler e escrever dados no Hardware (ler os estados da placa e escrever nos outputs da placa) a partir de funções implementadas em Software.

O objeto HAL é essencial para fazer a interligação entre o HW e o SW pois as suas funções são as mais básicas de manipulação de bits na placa e necessárias para operações mais complexas.

Para além do objeto HAL, esta interligação é feita a partir do mesmo, nos objetos KBD, Serial Emitter e M.

No objeto KBD, as funções do HAL são chamadas para:
- verificar se o Kval está a 0 ou 1, com a função isBit();
- ler os estados da placa com K como máscara, com a função readBits();
- por o Kack a 0, com a função clrBits().

No objeto Serial Emitter, as funções do HAL são chamadas para:
- colocar os bits correspondentes a SSLCD e em seguida os correspondentes a SSDOOR, a 1, com a função setBits();
- dependendo do address da função send, é colocado nos bits correspondentes a SSDOOR ou a SSLCD a 0, com a função clrBits();
- pôr os bits do SCLOCK a 0, com a função clrBits();
- pôr os bits do SDX a 0, com a função clrBits(), ou a 1, com a função setBits(), dependendo se a data estiver a 0 ou não;
- colocar os bits de SCLOCK a 1 (setBits()) e em seguida a 0 (clrBits());
- dependendo do address da função send, é colocado nos bits correspondentes a SSDOOR ou a SSLCD a 1, com a função setBits();
- verificar se busy está a 1 ou a 0, com a função isBit().

No objeto M, as funções do HAL são chamadas para:
- verificar se o bit m está a 1 ou a 0 (se a chave de manutenção está ligada ou não), com a função isBit().

B. Código *Kotlin - HAL*

```
object HAL {
    private const val OFF = 0x00
    private const val INIT_STATE = 0x00
    private var lastState = INIT_STATE
    private var initialize = false

    // inicia o usbport a 0 e atualiza o lastState para 0
    fun init() {
        if (!initialize) {
            UsbPort.write(INIT_STATE)
            lastState = INIT_STATE
            initialize = true
        }
    }

    // le o(s) bit(s) indicado(s) na máscara
    fun readBits(mask: Int) = UsbPort.read().and(mask)

    // verifica se o bit indicado na máscara está on ou off
    fun isBit(mask: Int) = UsbPort.read().and(mask) != OFF

    // coloca o(s) bit(s) indicado(s9 na máscara a on
    fun setBits(mask: Int) {
        lastState = lastState.or(mask)
        UsbPort.write(lastState)
    }

    // coloca o(s) bit(s) indicado(s9 na máscara a off
    fun clrBits(mask: Int) {
        lastState = lastState.and(mask.inv())
        UsbPort.write(lastState)
    }

    // coloca o valor indicado em value no(s) bit(s) indicado(s) na máscara
    fun writeBits(mask: Int, value: Int) {
        lastState = value.or(lastState.and(mask.inv()))
        UsbPort.write(lastState)
    }
}
```

C. Código *Kotlin - KBD*

```
object KBD {
    private const val Kval = 0x01
    private const val Kack = 0x01
    private const val K = 0x1E
    const val waitTime = 2000L
    private const val sleepTime = 10L
    private const val CODE1 = 0x00
    private const val CODE2 = 0x04
    private const val CODE3 = 0x08
    private const val CODE4 = 0x01
    private const val CODE5 = 0x05
    private const val CODE6 = 0x09
    private const val CODE7 = 0x02
    private const val CODE8 = 0x06
    private const val CODE9 = 0x0A
    private const val CODE0 = 0x07
    private const val CODEEXT = 0x03
    private const val CODEHASH = 0x0B

    // inicia o HAL e coloca o bit correspondente a K Ack a 0
    fun init() {
        HAL.init()
        HAL.clrBits(Kack)
    }

    const val NONE = 0.toChar()

    // le o que é escrito no teclado e devolve o que leu, caso não consiga ler nada devolve NONE
    private fun getKey(): Int {
        if (HAL.isBit(Kval)) {
            val c = when (HAL.readBits(K).shr(1)) {
                CODE1 -> '1'.code
                CODE2 -> '2'.code
                CODE3 -> '3'.code
                CODE4 -> '4'.code
                CODE5 -> '5'.code
                CODE6 -> '6'.code
                CODE7 -> '7'.code
                CODE8 -> '8'.code
                CODE9 -> '9'.code
                CODEEXT -> '*'.code
                CODE0 -> '0'.code
                CODEHASH -> '#'.code
                else -> NONE.code
            }
            HAL.setBits(Kack)
            while (HAL.isBit(Kval)) {
                Thread.sleep(sleepTime)
            }
```

```
            HAL.clrBits(Kack)
            return c
        }
        return NONE.code
    }

    // chama o getKey() até passar o tempo de timeout ou até ler uma tecla
    fun waitKey(timeout: Long): Int {
        val timeInit = System.currentTimeMillis()
        while (true) {
            val time = System.currentTimeMillis()
            val c = getKey()
            if (c != NONE.code) return c
            if (time - timeInit >= timeout) return NONE.code
        }
    }
}
```

D. Código *Kotlin – SerialEmitter*

```
object SerialEmitter {

    enum class Destination { LCD, DOOR }
    private const val SLEEPTIME = 1L
    private const val SSLCD = 0x02
    private const val SDX = 0x04
    private const val SCLOCK = 0x08
    private const val SSDOOR = 0x10
    private const val busy = 0x20
    private const val MAX_LENGTH = 5
    private const val OFF = 0x00

    // inicia o HAL e coloca os bits relativos ao SS tanto do lcd como da door a 1 (pois são active low)
    fun init() {
        HAL.init()
        HAL.setBits(SSLCD)
        HAL.setBits(SSDOOR)
    }

    // escreve 5 bits de informação, 1 a 1, no usbport
    fun send(addr: Destination, data: Int) {
        val mask = if (addr == Destination.DOOR) SSDOOR else SSLCD
        HAL.clrBits(mask)
        for (i in 0 until MAX_LENGTH) {
            HAL.clrBits(SCLOCK)
            val b = data.and(1.shl(i))
            if (b == OFF) {
                HAL.clrBits(SDX)
            } else {
                HAL.setBits(SDX)
            }
            HAL.setBits(SCLOCK)
        }
        HAL.clrBits(SCLOCK)
        HAL.setBits(mask)
        Thread.sleep(SLEEPTIME)
    }

    // verifica a condição do sinal busy
    fun isBusy() = HAL.isBit(busy)
}
```

E. Código *Kotlin - LCD*

```
object LCD {
    private const val RS = 0x04
    private const val Enable = 0x02
    private const val DATA = 0x78
    private const val LOW = 0x0F
    private const val SLEEPTIME = 1L
    private const val SLEEPTIME1 = 40L
    private const val SLEEPTIME2 = 5L
    private const val SLEEPWRITE = 40
    private const val SLEEPSET = 230
    private const val SLEEPCLR = 270
    private const val ON = 0x01
    private const val OFF = 0x00
    private const val SET8BITS = 0x03
    private const val SET4BITS = 0x02
    private const val NUM_LINES_CHAR = 0x28
    private const val DISPLAY_OFF = 0x08
    private const val DISPLAY_CLEAR = 0x01
    private const val ENTRY_MODE = 0x06
    private const val DISPLAY_ON_CONTROL = 0x0E
    private const val SERIAL = true

    // Escreve um nibble de comando/dados no LCD em paralelo
    private fun writeNibbleParallel(rs: Boolean, data: Int) {
        HAL.writeBits(RS, (if (rs) ON else OFF).shl(2))
        HAL.writeBits(DATA, data.shl(3))
        Thread.sleep(0, SLEEPWRITE)
        HAL.setBits(Enable)
        Thread.sleep(0, SLEEPSET)
        HAL.clrBits(Enable)
        Thread.sleep(0, SLEEPCLR)
    }

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

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

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

```
}

// Escreve um comando no LCD
private fun writeCMD(data: Int) {
   writeByte(false, data)
}

// Escreve um dado no LCD
private fun writeDATA(data: Int) {
   writeByte(true, data)
}

// Envia a sequência de iniciação para comunicação a 4 bits.
fun init() {
   SerialEmitter.init()
   Thread.sleep(SLEEPTIME1)
   writeNibble(false, SET8BITS)
   Thread.sleep(SLEEPTIME2)
   writeNibble(false, SET8BITS)
   Thread.sleep(SLEEPTIME)
   writeNibble(false, SET8BITS)
   Thread.sleep(SLEEPTIME)
   writeNibble(false, SET4BITS)
   Thread.sleep(SLEEPTIME)
   writeCMD(NUM_LINES_CHAR)
   Thread.sleep(SLEEPTIME)
   writeCMD(DISPLAY_OFF)
   Thread.sleep(SLEEPTIME)
   writeCMD(DISPLAY_CLEAR)
   Thread.sleep(SLEEPTIME)
   writeCMD(ENTRY_MODE)
   Thread.sleep(SLEEPTIME)
   writeCMD(DISPLAY_ON_CONTROL)
}

// Escreve um caráter na posição corrente.
fun write(c: Char) {
   writeDATA(c.code)
}

// Escreve uma “string” na posição corrente.
fun write(text: String) {
   text.forEach { write(it) }
}

// Envia comando para posicionar cursor (‘line’:0..LINES-1 , ‘column’:0..COLS-1)
fun cursor(line: Int, column: Int) {
   writeByte(false, (line * 4 + 8) * 16 + column)
}

// Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
```

```
    fun clear() {
        writeCMD(DISPLAY_CLEAR)
    }
}
```

F. Código *Kotlin* - *DoorMechanism*

```
@file:Suppress("ControlFlowWithEmptyBody")

object DoorMechanism { // Controla o estado do mecanismo de abertura da porta.
    private const val OPEN = 0x01

    // Inicia a classe, estabelecendo os valores iniciais.
    fun init() {
        SerialEmitter.init()
    }

    // Envia comando para abrir a porta, com o parâmetro de velocidade
    fun open(velocity: Int) {
        SerialEmitter.send(SerialEmitter.Destination.DOOR, velocity.shl(1) or OPEN)
    }

    // Envia comando para fechar a porta, com o parâmetro de velocidade
    fun close(velocity: Int) {
        SerialEmitter.send(SerialEmitter.Destination.DOOR, velocity.shl(1))
    }

    // Verifica se o comando anterior está concluído
    fun finished() = !SerialEmitter.isBusy()
}
```

G. Código *Kotlin - TUI*

```
import kotlin.math.pow

object TUI {
    private const val MAX_TEXT_LENGTH = 16
    private const val MAX_UIN_LENGTH = 2 //including 0
    private const val MAX_PIN_LENGTH = 3 //including 0
    private const val TIMEOUT = 5000L
    private const val SECOND_LINE = 1
    private const val FIRST_COL = 0
    private const val EMPTY = 0
    private const val ASTERISK = '*'.code
    private const val BASE = 10
    private const val UIN = "UIN:"
    private const val PIN = "PIN:"
    private const val CLEAR_LINE = "                "

    // inicia o lcd e o kbd
    fun init() {
        LCD.init()
        KBD.init()
    }

    // escreve um caracter no lcd
    private fun writeChar(char: Char) {
        LCD.write(char)
    }

    // escreve um inteiro no lcd
    private fun writeInt(i: Int) {
        LCD.write(i.toChar())
    }

    // escreve uma frase no lcd
    fun writeString(text: String) {
        if (text.length >= MAX_TEXT_LENGTH) {
            for (i in EMPTY until MAX_TEXT_LENGTH) {
                writeChar(text[i])
            }
            nextLine()
            for (i in MAX_TEXT_LENGTH until text.length) {
                writeChar(text[i])
            }
        } else
            LCD.write(text)
    }

    // le uma tecla do kbd
    fun readKey(): Int {
```

```
        return KBD.waitKey(TIMEOUT)
    }

    // limpa o display do lcd
    fun clear() {
        LCD.clear()
    }

    // poe o cursor no inicio da segunda linha
    fun nextLine() {
        LCD.cursor(SECOND_LINE, FIRST_COL)
    }

    //pede ao utilizador para introduzir o seu UIN
    fun writeUIN() {
        writeString(UIN)
    }

    // pede ao utilizador para introduzir o seu PIN
    fun writePIN() {
        writeString(PIN)
    }

    // Pede o novo pin, e depois pede uma confirmaçao. Caso sejam iguais retorna esse valor, caso contrario
retorna null
    fun newPIN(): Int? {
        writePIN()
        val pin = readPIN()
        nextLine()
        writePIN()
        if (readPIN() == pin)
            return pin
        return null
    }

    //le o UIN intrduzido pelo utilizador
    fun readUIN(): Int? {
        return readInput(MAX_UIN_LENGTH, UIN)
    }

    //le o PIN intrduzido pelo utilizador
    fun readPIN() = readInput(MAX_PIN_LENGTH, PIN)

    //le o 'input' intrduzido pelo utilizador
    //caso seja lido um asterisco durante este processo, ele limpa o que ja foi lido e começa a ler do início, caso
ainda estivesse vazio, para de ler
    // se houver um timeout, para de ler
    private fun readInput(size: Int, type: String): Int? {
        var r = EMPTY
        var unit = BASE.toDouble().pow(size.toDouble()).toInt()
        var empty = true
```

```
        for (i in EMPTY..size) {
            val c = readKey()
            if (c == KBD.NONE.code) {
                clear()
                return null
            } else if (empty && c == ASTERISK) {
                clear()
                if (type == PIN) writePIN() else writeUIN()
            } else if (c == ASTERISK) {
                return null
            } else {
                empty = false
                r += (c - '0'.code) * unit
                unit /= BASE
                val print = if (type == PIN) ASTERISK else c
                writeInt(print)
            }
        }
        return r
    }

    // limpa a segunda linha, sem alterar o conteudo da primeira, e prepara-se para escrever no inico da segunda
linha
    fun clearSecondLine() {
        nextLine()
        writeString(CLEAR_LINE)
        nextLine()
    }
}
```

H. Código *Kotlin - FileAccess*

```
import java.io.BufferedReader
import java.io.FileReader
import java.io.PrintWriter

object FileAccess {

    // le o conteúdo de um ficheiro txt
    fun readFile(name: String): HashSet<String> {
        val reader = BufferedReader(FileReader(name))
        val list = HashSet<String>()
        var line: String?
        line = reader.readLine()
        while (line != null) {
            list.add(line)
            line = reader.readLine()
        }
        return list
    }

    // escreve o conteúdo do 'output' num ficheiro txt
    fun writeFile(name: String, output: ArrayList<String>) {
        val writer = PrintWriter(name)
        for (i in output) {
            writer.println(i)
        }
        writer.close()
    }
}
```

I. Código *Kotlin - Users*

```
object Users {
    private const val FILENAME = "USERS.txt"
    private const val UIN_MASK = 0x00
    private const val PIN_MASK = 0x01
    private const val NAME_MASK = 0x02
    private const val MSG_MASK = 0x03
    private const val SPLITTER = ';'
    private const val EMPTY_MSG = ""

    // le o ficheiro txt e guarda os users registados
    fun getUsers(): HashMap<Int, User> {
        val map = HashMap<Int, User>()
        val list = FileAccess.readFile(FILENAME)
        for (line in list) {
            val a = line.split(SPLITTER)
            val user = User(a[UIN_MASK].toInt(), a[PIN_MASK].toInt(), a[NAME_MASK], a[MSG_MASK])
            map[user.uin] = user
        }
        return map
    }

    // escreve cada user em userList no ficheiro USERS.txt
    fun writeUsers(userList: HashMap<Int, User>) {
        val list = ArrayList<String>()
        for (user in userList.values) {
            list.add("${user.uin}$SPLITTER${user.pin}$SPLITTER${user.name}$SPLITTER${user.msg}$SPLITTER")
        }
        FileAccess.writeFile(FILENAME, list)
    }

    class User(id: Int, pw: Int, n: String, message: String = EMPTY_MSG) {
        val uin: Int = id
        var pin: Int = pw
        val name: String = n
        var msg: String = message

    }

    // le o conteudo do ficheiro USERS
    fun init(): java.util.HashMap<Int, User> {
        return getUsers()
    }
}
```

J. Código *Kotlin - Log*

```
object Log {
    private const val FILENAME = "Log File.txt"
    fun init(): HashSet<String> {
        return getLog()
    }

    // le e guarda num ArrayList o conteúdo do ficheiro Log File
    private fun getLog(): HashSet<String> {
        val log = HashSet<String>()
        val list = FileAccess.readFile(FILENAME)
        for (l in list) {
            log.add(l)
        }
        return log
    }

    // escreve o conteúdo do log no ficheiro Log File
    fun writeLog(log: HashSet<String>) {
        val writer = PrintWriter(FILENAME)
        for (l in log) {
            writer.println(l)
        }
        writer.close()
    }
}
```

L. Código *Kotlin* da classe *M*

```
object M {
   private const val m = 0x40
   fun init() {
      HAL.init()
   }

   // verifica se a chave de manutenção está ligada
   fun isM() = HAL.isBit(m)
}
```

M. Código *Kotlin – Access Control System - App*

```
import kotlin.math.abs
import kotlin.system.exitProcess

object App {
    private var log = HashSet<String>() // lista de logs
    private var users = HashMap<Int, Users.User>() // lista de users registados
    private const val DATE_FORMAT = "yyyy-MM-dd HH:mm"
    private const val PIN_CHANGED = "Pin changed successfully."
    private const val MSG_ADDED = "Message added successfully."
    private const val MAX_NAME_LENGHT = 16 // tamanho maximo do nome do user
    private const val DEFAULT_SPEED = 10
    private const val DISPLAY_TIME = 1000L
    private const val MAX_USERS = 1000
    private const val CLEAR_MSG = '*'.code
    private const val CHANGE_PIN = '#'.code
    private const val ALGORITHM_KEY = "PBKDF2WithHmacSHA512"
    private const val ITERATIONS = 50
    private const val KEY_LENGTH = 128
    private const val SECRET_KEY = "RandomSecret"
    private const val NO_MSG = ""
    private const val CONFIRM = 'y'
    private const val FULL_DATABASE = "Database is full."
    private const val INSERT_NEW_USER = "Insert user's name (Max $MAX_NAME_LENGHT characters)"
    private const val INSERT_UIN = "Insert UIN"
    private const val INSERT_MSG = "Insert message"
    private const val INSERT_PIN = "Insert PIN"
    private const val USER_REMOVAL = "UIN to remove?"
    private const val LOGIN_FAIL = "Login Failed."
    private const val WRONG_COMMAND = "No such command."
    private const val OUT_OF_SERVICE = "Out of service"
    private const val COMMANDS = "Commands: New | Del | AddMsg | Exit"

    //função de assistencia à encirptaçao do pin
    private fun ByteArray.toHexString(): String = HexFormat.of().formatHex(this)

    //função de assistencia à encirptaçao do pin
    private fun Int.toCharArray(): CharArray = toString().toCharArray()

    // Gera código de encriptação de 4 digitos representativos do pin do user
    private fun generateHash(password: Int): Int {
        val combinedSalt = "$password$SECRET_KEY"
        val factory: SecretKeyFactory = SecretKeyFactory.getInstance(ALGORITHM_KEY)
        val spec: KeySpec = PBEKeySpec(password.toCharArray(), combinedSalt.toByteArray(), ITERATIONS,
KEY_LENGTH)
        val key: SecretKey = factory.generateSecret(spec)
        val hash: ByteArray = key.encoded
        return abs(hash.toHexString().hashCode() % 10000)
    }
```

```
//inicia as classes subsequentes e fecha a porta
fun init() {
    M.init()
    DoorMechanism.init()
    TUI.init()
    users = Users.init()
    log = Log.init()
    DoorMechanism.close(DEFAULT_SPEED)
}

// regista um user novo se houver espaço
private fun insertUser() {
    if (users.size == MAX_USERS) {
        TUI.writeString(FULL_DATABASE)
    } else {
        var id = 0
        while (id < users.size) {
            if (users[id] == null) break
            else id++
        }
        var name: String
        do {
            println(INSERT_NEW_USER)
            name = readln()
        } while (name.length > MAX_NAME_LENGHT)
        println(INSERT_PIN)
        val pin = readln().toInt()
        val newUser = Users.User(id, generateHash(pin), name)
        users[id] = newUser
    }
    usersList()
}

//remove um user
private fun removeUser() {
    println(USER_REMOVAL)
    val uin = readln().toInt()
    println("Confirm removal of user $uin? (Y|N)")
    val confirm = readln()
    if (confirm.lowercase().first() == CONFIRM) {
        users.remove(uin)
    }
    usersList()
}

// coloca uma mensagem num user
private fun insertMessage() {
    println(INSERT_UIN)
    val uin = readln().toInt()
    println(INSERT_MSG)
```

```
        val message = readln()
        users[uin]?.msg = message
        TUI.writeString(MSG_ADDED)
        usersList()
    }

    // regista a lista de users atualizada e os logs, depois desligando o sistema
    private fun turnOff() {
        TUI.clear()
        DoorMechanism.close(DEFAULT_SPEED)
        Users.writeUsers(users)
        Log.writeLog(log)
        exitProcess(0)
    }

    // imprime a lista de users registados
    private fun usersList() {
        for (user in users.values) {
            println("${user.uin}, ${user.pin}, ${user.name}, ${user.msg}")
        }
    }

    // dá 'login' caso o uin e pin existam na base de dados, abre e fecha a porta, e ainda pode remover a
mensasagem ou
    // alterar o pin currente
    fun logIn(uin: Int, pin: Int): Users.User? {
        val user = users[uin]
        val pw = generateHash(pin)
        if (user != null && user.pin == pw) {
            TUI.clear()
            TUI.writeString(user.name)
            Thread.sleep(DISPLAY_TIME)
            openDoor()
            val l = getTime() + " ${user.uin}"
            log.add(l)
            if (user.msg != NO_MSG) {
                msg(user)
            }
            if (TUI.readKey() == CHANGE_PIN) {
                changePIN(user)
            }
            closeDoor()
        } else {
            TUI.clear()
            TUI.writeString(LOGIN_FAIL)
            Thread.sleep(DISPLAY_TIME)
        }
        return user
    }

    //escreve a mensagem de um user e, caso seja premida a tecla *, remove-a
```

```
private fun msg(user: Users.User) {
    TUI.clear()
    TUI.writeString(user.msg)
    if (TUI.readKey() == CLEAR_MSG) {
        user.msg = NO_MSG
        TUI.clear()
    }
}

//altera o pin atribuido a um user
private fun changePIN(user: Users.User) {
    TUI.clear()
    val newPin = TUI.newPIN()
    if (newPin != null) {
        user.pin = generateHash(newPin)
        TUI.clear()
        TUI.writeString(PIN_CHANGED)
    }
}

//fecha a porta
private fun closeDoor() {
    DoorMechanism.close(DEFAULT_SPEED)
    while (!DoorMechanism.finished());
}

//abre a porta
private fun openDoor() {
    DoorMechanism.open(DEFAULT_SPEED)
    while (!DoorMechanism.finished());
}

// vai buscar a data e hora
private fun getTime(): String {
    val formatter = DateTimeFormatter.ofPattern(DATE_FORMAT)
    return LocalDateTime.now().format(formatter)
}

// escreve a data e hora no lcd
fun printTime() {
    TUI.clear()
    TUI.writeString(getTime())
    TUI.nextLine()
}

// le o uin introduzido
fun getUIN(): Int? {
    TUI.writeUIN()
    val user = TUI.readUIN()
    TUI.clearSecondLine()
    return user
```

```
    }

    // le o pin introduzido
    fun getPIN(): Int? {
        TUI.writePIN()
        return TUI.readPIN()
    }

    //manutenção
    fun mKey() {
        TUI.clear()
        TUI.writeString(OUT_OF_SERVICE)
        println(COMMANDS)
        val command = readln()
        when (command.lowercase()) {
            "new" -> insertUser()
            "del" -> removeUser()
            "addmsg" -> insertMessage()
            "exit" -> turnOff()
            else -> println(WRONG_COMMAND)
        }
    }
}

fun main() {
    App.init()
    while (true) {
        if (!M.isM()) {
            App.printTime()
            val user = App.getUIN()
            if (user != null) {
                val pin = App.getPIN()
                if (pin != null)
                    App.logIn(user, pin)
            }
        } else {
            while (M.isM()) {
                App.mKey()
            }
        }
    }
}
```

N. Atribuição de pinos do módulo *SCA*

```
set_global_assignment -name TOP_LEVEL_ENTITY "DE10_Lite"
set_global_assignment -name DEVICE_FILTER_PACKAGE FBGA
set_global_assignment -name SDC_FILE DE10_Lite.sdc
set_global_assignment -name INTERNAL_FLASH_UPDATE_MODE "SINGLE IMAGE WITH ERAM"

# clock
set_location_assignment PIN_P11 -to CLK

# inputs
set_location_assignment PIN_C10 -to M
set_location_assignment PIN_C11 -to Pswitch
set_location_assignment PIN_F15 -to Reset

#Leds
set_location_assignment PIN_A8 -to Door_data[0]
set_location_assignment PIN_A9 -to Door_data[1]
set_location_assignment PIN_A10 -to Door_data[2]
set_location_assignment PIN_B10 -to Door_data[3]
set_location_assignment PIN_D13 -to OC
set_location_assignment PIN_C13 -to OO
set_location_assignment PIN_E14 -to Sopen
set_location_assignment PIN_D14 -to Sclose
set_location_assignment PIN_A11 -to Psensor

#Keypad
set_location_assignment PIN_W5     -to  I[0]
set_location_assignment PIN_AA14   -to      I[1]
set_location_assignment PIN_W12    -to  I[2]
set_location_assignment PIN_AB12   -to      I[3]
set_location_assignment PIN_AB11   -to      O[0]
set_location_assignment PIN_AB10   -to      O[1]
set_location_assignment PIN_AA9    -to      O[2]

#LCD
set_location_assignment PIN_W8     -to LCD_RS
set_location_assignment PIN_V5     -to LCD_EN
set_location_assignment PIN_W11    -to LCD_DATA[0]
set_location_assignment PIN_AA10   -to LCD_DATA[1]
set_location_assignment PIN_Y8     -to LCD_DATA[2]
set_location_assignment PIN_Y7     -to LCD_DATA[3]

#HEX0
set_location_assignment PIN_C14 -to HEX0[0]
set_location_assignment PIN_E15 -to HEX0[1]
set_location_assignment PIN_C15 -to HEX0[2]
set_location_assignment PIN_C16 -to HEX0[3]
set_location_assignment PIN_E16 -to HEX0[4]
set_location_assignment PIN_D17 -to HEX0[5]
set_location_assignment PIN_C17 -to HEX0[6]
```

```
set_location_assignment PIN_D15 -to HEX0[7]

#HEX1
set_location_assignment PIN_C18 -to HEX1[0]
set_location_assignment PIN_D18 -to HEX1[1]
set_location_assignment PIN_E18 -to HEX1[2]
set_location_assignment PIN_B16 -to HEX1[3]
set_location_assignment PIN_A17 -to HEX1[4]
set_location_assignment PIN_A18 -to HEX1[5]
set_location_assignment PIN_B17 -to HEX1[6]
set_location_assignment PIN_A16 -to HEX1[7]

#HEX2
set_location_assignment PIN_B20 -to HEX2[0]
set_location_assignment PIN_A20 -to HEX2[1]
set_location_assignment PIN_B19 -to HEX2[2]
set_location_assignment PIN_A21 -to HEX2[3]
set_location_assignment PIN_B21 -to HEX2[4]
set_location_assignment PIN_C22 -to HEX2[5]
set_location_assignment PIN_B22 -to HEX2[6]
set_location_assignment PIN_A19 -to HEX2[7]

#HEX3
set_location_assignment PIN_F21 -to HEX3[0]
set_location_assignment PIN_E22 -to HEX3[1]
set_location_assignment PIN_E21 -to HEX3[2]
set_location_assignment PIN_C19 -to HEX3[3]
set_location_assignment PIN_C20 -to HEX3[4]
set_location_assignment PIN_D19 -to HEX3[5]
set_location_assignment PIN_E17 -to HEX3[6]
set_location_assignment PIN_D22 -to HEX3[7]

#HEX4
set_location_assignment PIN_F18 -to HEX4[0]
set_location_assignment PIN_E20 -to HEX4[1]
set_location_assignment PIN_E19 -to HEX4[2]
set_location_assignment PIN_J18 -to HEX4[3]
set_location_assignment PIN_H19 -to HEX4[4]
set_location_assignment PIN_F19 -to HEX4[5]
set_location_assignment PIN_F20 -to HEX4[6]
set_location_assignment PIN_F17 -to HEX4[7]

#HEX5
set_location_assignment PIN_J20 -to HEX5[0]
set_location_assignment PIN_K20 -to HEX5[1]
set_location_assignment PIN_L18 -to HEX5[2]
set_location_assignment PIN_N18 -to HEX5[3]
set_location_assignment PIN_M20 -to HEX5[4]
set_location_assignment PIN_N19 -to HEX5[5]
set_location_assignment PIN_N20 -to HEX5[6]
set_location_assignment PIN_L19 -to HEX5[7]
```