

**Licenciatura em Engenharia Informática e de Computadores**

Relógio de Ponto

(*Working Time Recorder*)

Ricardo Henriques (A48322@alunos.isel.pt)

João Magalhães (A48348@alunos.isel.pt)

Projeto

de

Laboratório de Informática e Computadores

2020 / 2021 verão

24 de junho de 2021

[1 Introdução 2](#_Toc75691455)

[2 Arquitetura do sistema 3](#_Toc75691456)

[A. Interligações entre o HW e SW 6](#_Toc75691457)

[B. Esquemas Elétricos do Sistema 7](#_Toc75691458)

[C. Descrição CUPL do bloco *Key Decode* 8](#_Toc75691459)

[D. Descrição CUPL do bloco *Key Buffer* 9](#_Toc75691460)

[E. Descrição CUPL do bloco *Key Transmitter* 10](#_Toc75691461)

[F. Descrição CUPL do bloco *Controller* 11](#_Toc75691462)

[G. Código *Kotlin* da classe *HAL* 12](#_Toc75691463)

[H. Código *Kotlin* da classe *KBD* 13](#_Toc75691464)

[I. Código *Kotlin* da classe *SerialReceiver* 14](#_Toc75691465)

[J. Código *Kotlin* da classe *LCD* 15](#_Toc75691466)

[K. Código *Kotlin* da classe *TUI* 17](#_Toc75691467)

[L. Código *Kotlin* da classe *FileAccess* 18](#_Toc75691468)

[M. Código *Kotlin* da classe *Users* 19](#_Toc75691469)

[N. Código *Kotlin* da classe *LOG* 20](#_Toc75691470)

[O. Código *Kotlin* da classe *Door* 22](#_Toc75691471)

[P. Código *Kotlin* da classe *M* 23](#_Toc75691472)

[Q. Código *Kotlin* da classe *WorkingTimeRecorder* *- App* 25](#_Toc75691473)

1. Introdução

Neste projeto implementa-se um sistema de Relógio de Ponto (*Working* *Time Recorder*) que permite registar a entrada/saída de um colaborador através de um número de identificação de utilizador (*User Identification Number – UIN*) e um código de acesso (*Personal Identification Number - PIN*). O sistema regista a entrada/saída após a inserção correta de um par *UIN* e *PIN*, para além do registo o sistema permite o acesso através de uma porta de acesso.

O sistema de relógio de ponto 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*) e uma chave de manutenção (designada por *M*) que define se o sistema de relógio de ponto está em modo de Manutenção. O diagrama de blocos do sistema de controlo de acessos é apresentado na Figura 1.

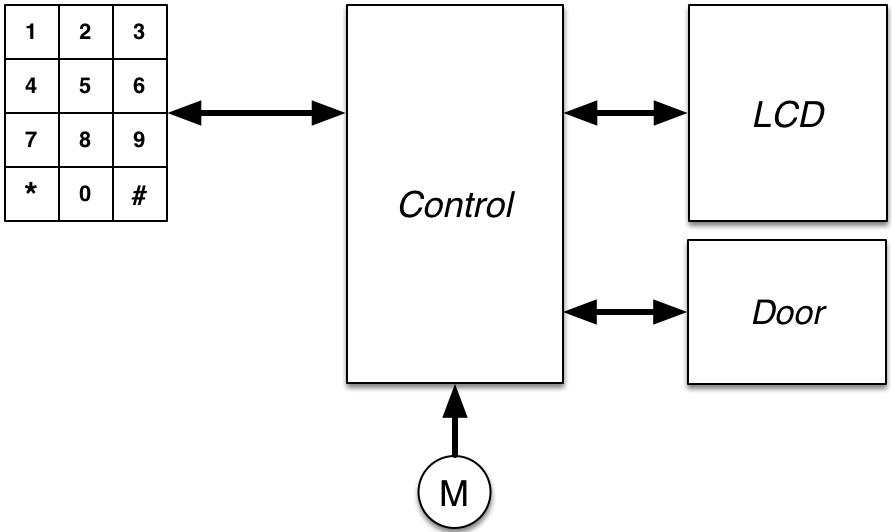


Figura 1 – Sistema de relógio de ponto (*Working Time Recorder*)

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

* **Registo -** Para registar a entrada/saída das instalações, o colaborador 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 a hora de entrada e saída, apresentando também o tempo de trabalho acumulado, acionando a abertura da porta. Todas as entradas/saídas deverão ser registadas com a informação de data/hora e *UIN* num ficheiro de registos, designado por *Log File, com* um registo de entrada/saída por linha, com o seguinte formato *“REGIST\_DATE;[>](entrada) ou [<](saída);UIN;NAME”*.
* **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 limpa todos os dígitos, se não contiver dígitos aborta o comando em curso; quando o dado a introduzir for um campo numérico e for premida uma tecla não numérica, esta deve ser ignorada.

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

* **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, letras maiúsculas e/ou minúsculas e/ou um espaço entre nomes.
* **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.
* **Listar os utilizadores –** Permite visualizar no ecrã do PC a lista dos utilizadores presentes nas instalações, apresentando o *UIN,* o nome e a hora de entrada.
* **Desligar –** Permite desligar o sistema de controlo de acessos, este termina após a confirmação do utilizador e a reescrita do ficheiro com a informação dos utilizadores. Esta informação deverá ser armazenada num ficheiro de texto (com um utilizador por linha, com o formato “*UIN;PIN;NAME;ENTRY\_DATE*”) 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, realizadas através do teclado do PC, não podem ser realizadas ações no teclado do utilizador e no LCD deve constar a mensagem “*Out of Service*”.

# Arquitetura do sistema

O sistema é implementado através de uma 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*; *iii*) um módulo de interface e controlo do mecanismo da porta, designado por *Door Controller*; e *iv*) um módulo de controlo, designado por *Control*. Os módulos *i*), *ii*) e *iii*) deverão ser implementados em *hardware*, enquanto o módulo de controlo deverá ser implementado em *software* a executar num PC usando linguagem *Kotlin*.

****

Figura 2. – Arquitetura do sistema que implementa o relógio de ponto (*Working Time Recorder*)

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

A implementação do módulo *Control* foi realizada em *software*, usando a linguagem Java e seguindo a arquitetura lógica apresentada na Figura 3.­­­­­­­



Figura 3. – Diagrama lógico do Relógio de Ponto (*Working Time Recorder*)

O *Keyborad* *Reader* é um módulo que é constituído por três sub-módulos de hardware e uma entidade consumidora de software, atra´ves deste módulo é possível ler o valor das teclas capturadas no *keyboard*, processá-las e envia-las a entidade consumidora de forma *bit* a *bit*; algo vantajoso nesta implementação é que é permitido ter-se até três valores de *key* em memória. Sendo este um dos módulos que usufrui de contacto direto com um dos objetos, sendo este o *keyboard*, uma senão a peça de hardware mais importante do sistema devido às suas funcionalidades, que devido à descodificação por parte deste módulo é depois possível serem processadas pela entidade consumidora em software.

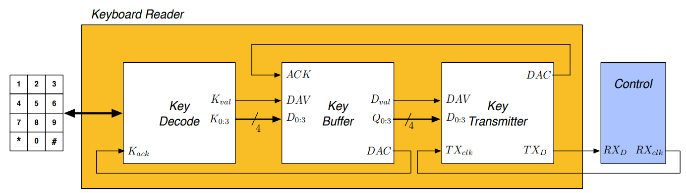


Figura 4. – Diagrama lógico do do módulo *Keyboard Reader*

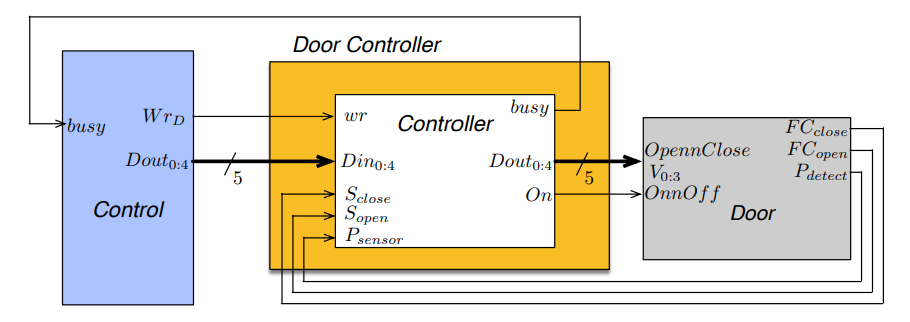
****O módulo *Door Controller* é constituído por 3 blocos, sendo dois desses hardware e um dos componentes software, como possível ser observado na figura 5. A primeira componente baseia-se num controlador em software, a segunda num controlador em hardware da porta e a última componente a porta em questão. Com este bloco e interligação entre hardware e software é permitido, em conjunto, reconhecendo as instruções obtidas e indicadas por parte do seu software para depois tratamento dos dados, permite gerir o funcionamento de abertura e fecho da porta e o seu gerenciamento da velocidade, transmitindo para o bloco de hardware e usando então uma máquina de estados para o processamento das instruções.

Figura 5. – Diagrama lógico do módulo *Door Controller*

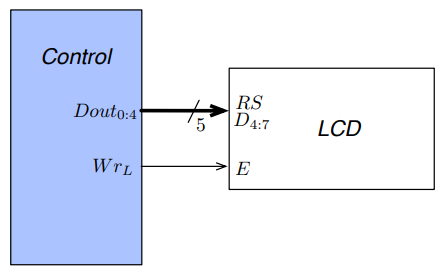
****Sendo o *LCD* em si, uma entidade que processa a data recebida, o seu módulo alem do *LCD* necessita de um bloco de software, sendo este um *Control* ,como possível ser visualizado na figura 6, é o bloco de *software* que trabalha e processa o código conforme as funções a serem executadas e fornece ao *LCD* os dados necessários para a execução das mesmas, tendo de ser inicializado com diversos códigos inicialmente para se depois poder executar os troços de código necessário; sendo este módulo auxiliado com funções de código da classe *TUI*, permite que em implementações futuras se possa adaptar e ir de encontro às necessidades do utilizador.

Figura 6. – Diagrama lógico do *LCD*

Sobre a figura 3, em relação ao diagrama lógico do Relógio de Ponto, é possível observar três módulos adjuntos, sendo estes os *Users*, *Log e FileAccess*. Devido à construção hierárquica é possível ser observado que o *File* *Access* fornece as instruções quer aos *Users* quer ao *Log*, isto devido a terem ficheiros de textos diversos e a sua acessão deve ser realizada da mesma maneira, sendo então essa leitura e escrita de ficheiro feita pelo módulo da base *File* *Access*, em relação aos dois módulos superiores, o do *Users* permite a atualização da lista dos mesmos, realizando alterações desde a adição e remoção à lista fornecida pelo ficheiro de texto, até só à sua simples visualização enquanto o módulo do *Log* se baseia em escrever as ações dos utilizadores em função de entrada ou saída. De referenciar, que o *FileAccess* foi feito da maneira mais generalizada, possível, para que o seu funcionamento, não seja único e exclusivamente, para este trabalho. A parte do tratamento de dados, e a forma como o expor, para as futuras iterações, são trabalhadas pelas camadas acima, como já referenciado.

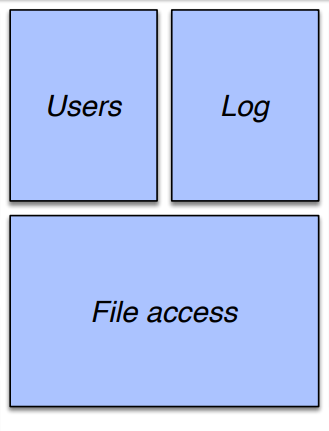


Figura 7. -Diagrama lógico dos *Users*, *Log e FileAccess*

Na figura 8, pode ser visualizado o diagrama do módulo de Manutenção, é com este módulo que dá possibilidade de haver um maior gerenciamento de quem pode trabalhar com a máquina, neste caso os *users*; este módulo é independente de outro devido à sua característica e importância. Só estabelece a conexão com hardware para a verificação e indicação se por parte do utilizador pretende a máquina no mesmo modo ou não, com este módulo é então possível remover e adicionar *users*; podendo também voltar a trabalhar num funcionamento normal ou até mesmo desligar a máquina. Este módulo não é suposto estar exposto para os clientes que usarão o relógio, mas sim para alguém que tenha o poder de gerenciamento para a máquina. Para se poder usar este botão, basta coloca-lo com o valor logico ‘1’ e completar um processo de colocar um user ou uma palavra passe, para entrar neste modo.

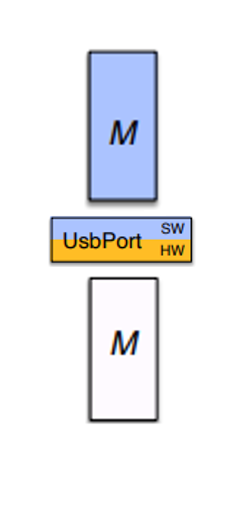


Figura 7. -Diagrama lógico da Manutenção

O módulo da *APP* é o topo da relação hierárquica do sistema, é o modulo que junta a conexão entre todos e assim permite o funcionamento como um todo, é este o módulo que através da sua componente total de software gere as ações principais para a realização do sistema, como as ações sequenciais quer do pedido do user, ao processamento do seu PIN e abertura e fecho da porta sem esquecendo dos devidos registos feitos nos ficheiros de texto, como é possível ter a outra vertente como visualizada no modulo anterior onde faz a verificação e chamada à função de manutenção possibilitando assim gerir o sistema. É este o módulo que interliga todos os módulos criados e dando assim um funcionamento a todos os módulos em conjuntos.. Este módulo pode ser alterando, realizando chamadas sequenciais de maneira diferente possibilitando ter a aplicação com as funcionalidades desejadas.

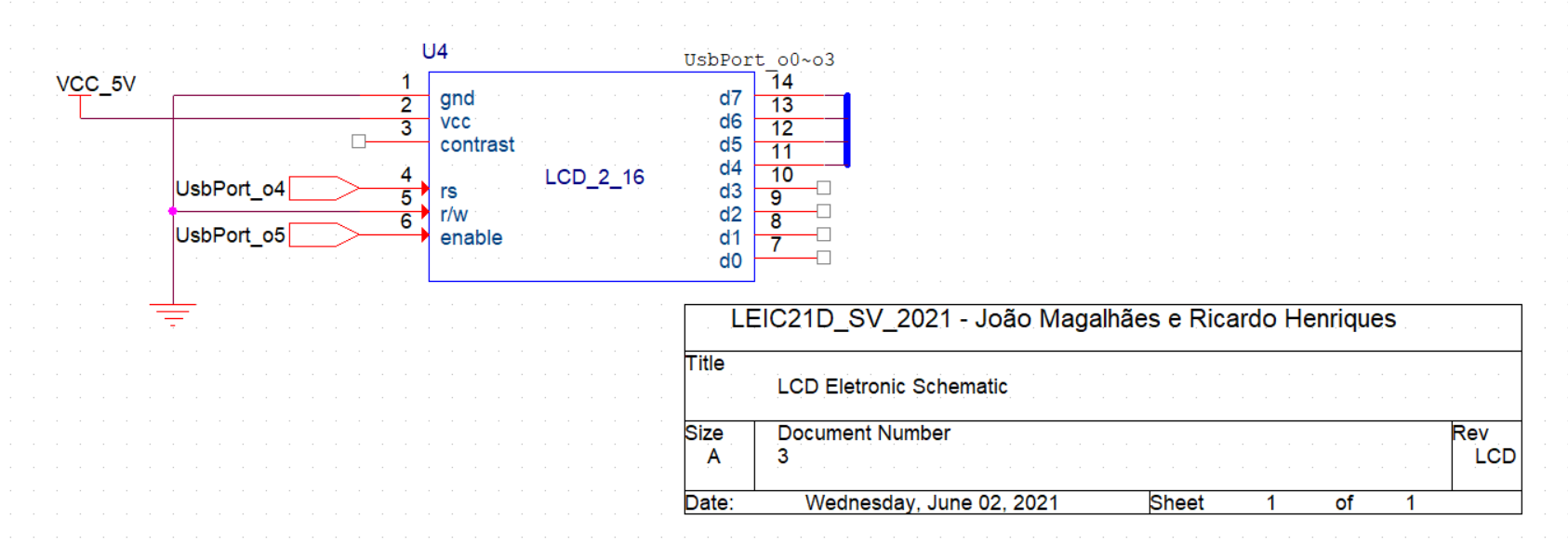
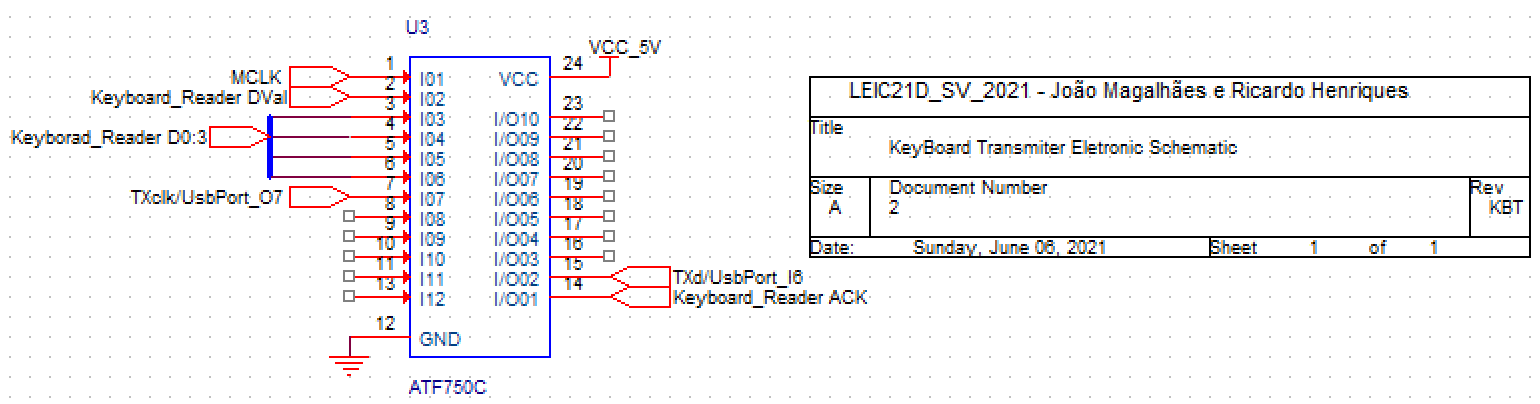
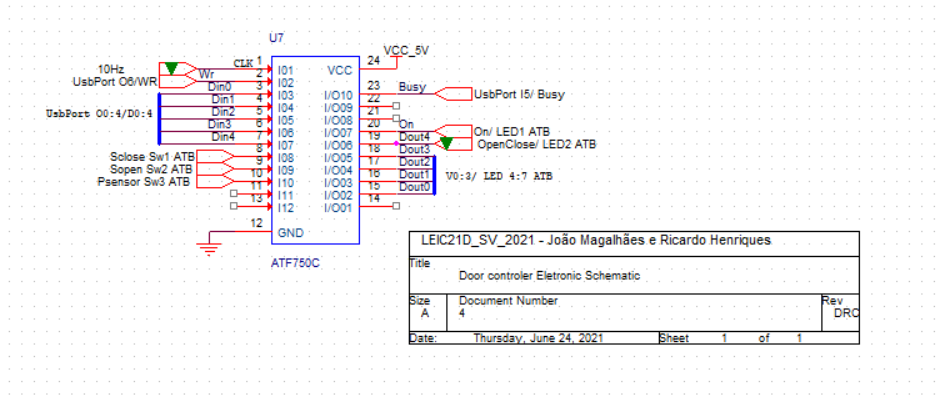
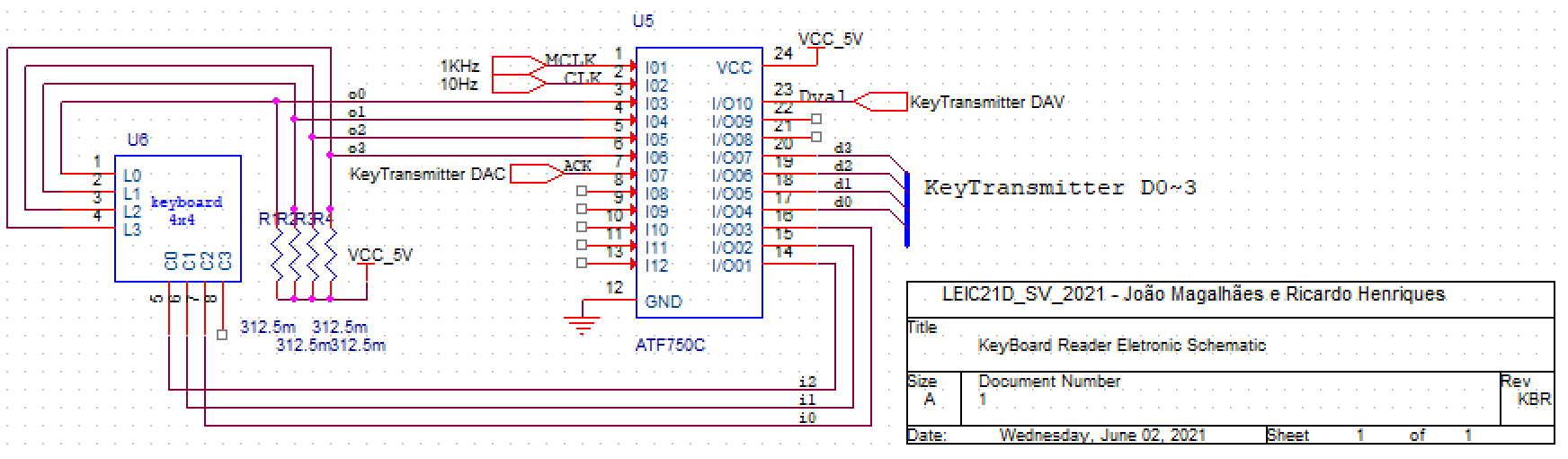
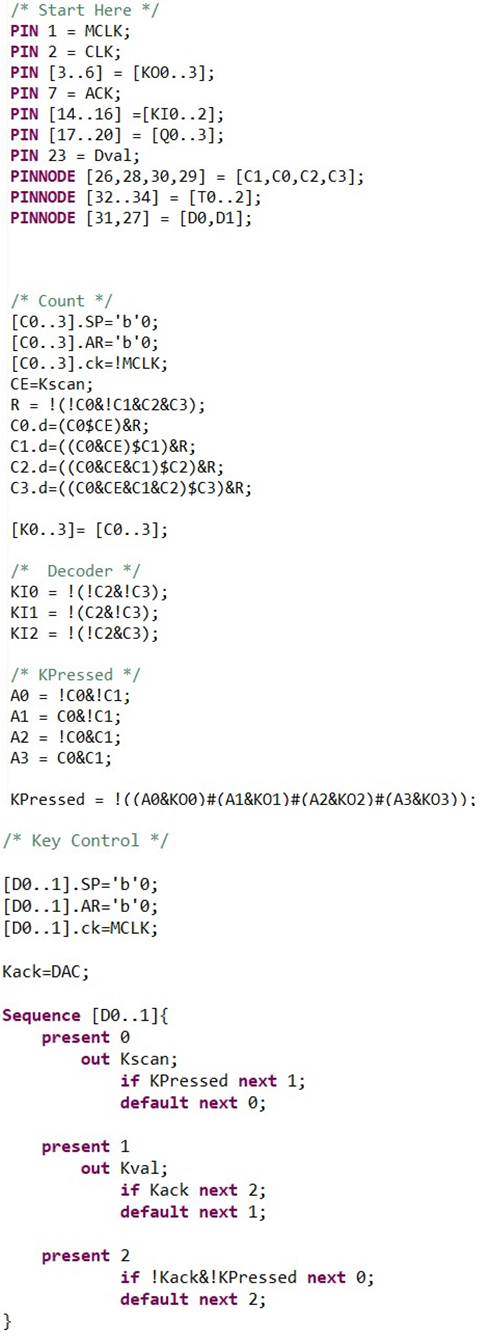


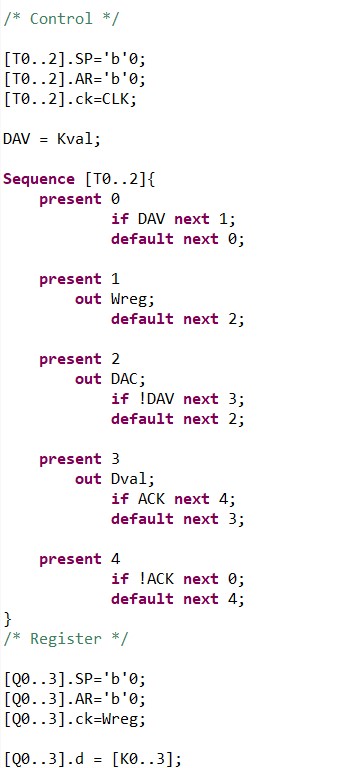
Figura 9. - Diagrama lógico da *APP*

1. Interligações entre o HW e SW

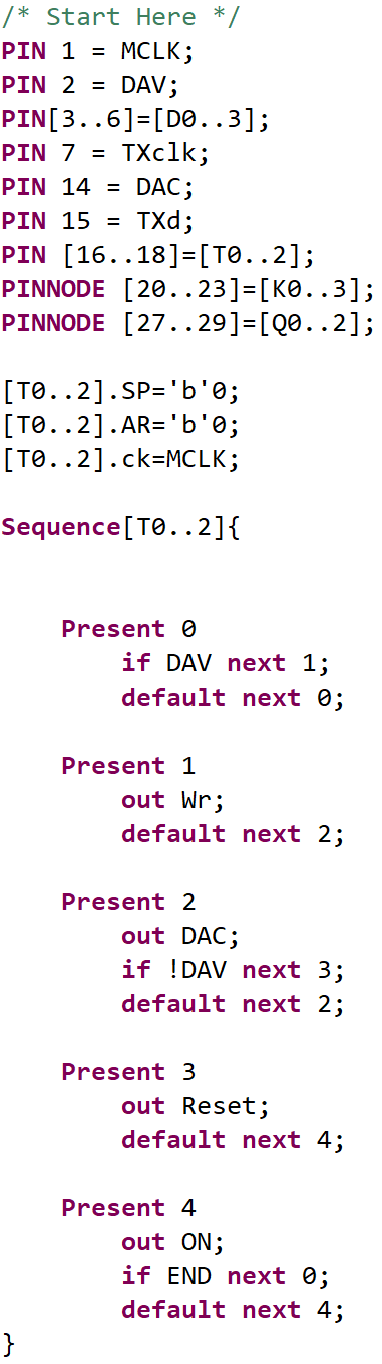
Pare ser possível haver uma ligação entre as entidades de *Hardware* com as entidades de *Software*, foi utilizado um *USBPORT*, um interface que permite essa mesma passagem necessária, através de alguns *drivers* que se coloca no *laptop* e ficheiros de configuração no projeto de código; é com esse aparelho que é permitida essa transição, é possível verificar os bits usados no *USBPORT* na figura 10.

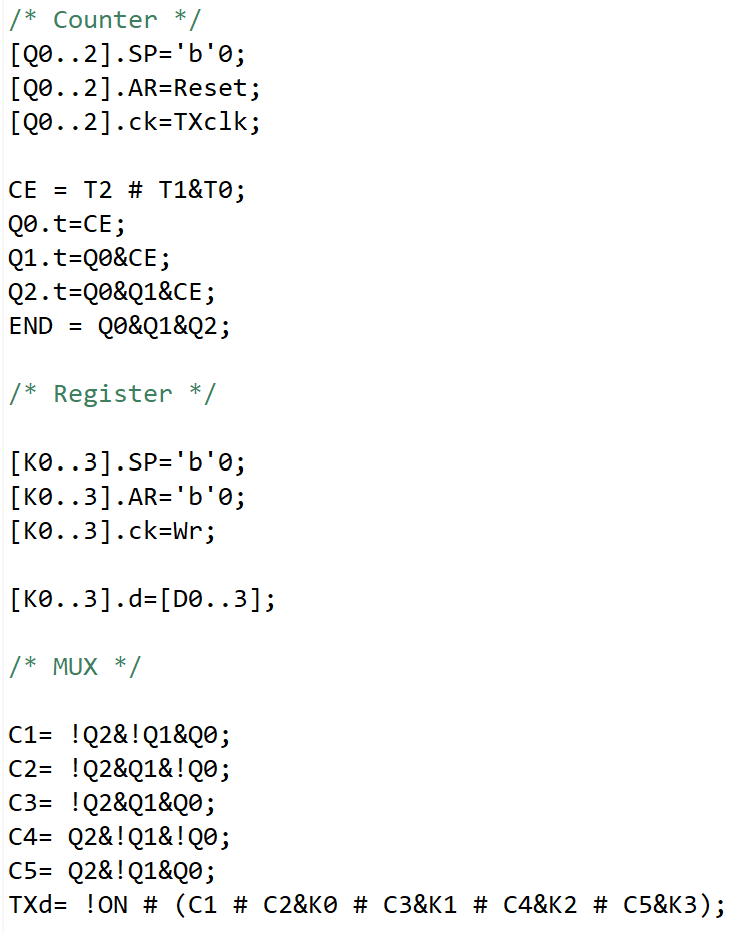
Figura 10. -Interligações entre o HW e SW

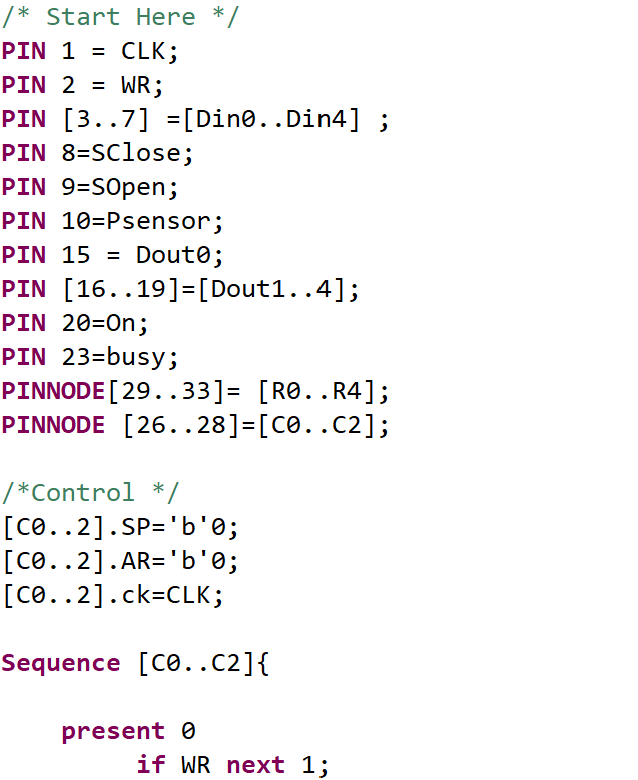
1. Esquemas Elétricos do Sistema
2.  Descrição CUPL do bloco *Key Decode*
3. Descrição CUPL do bloco *Key Buffer*

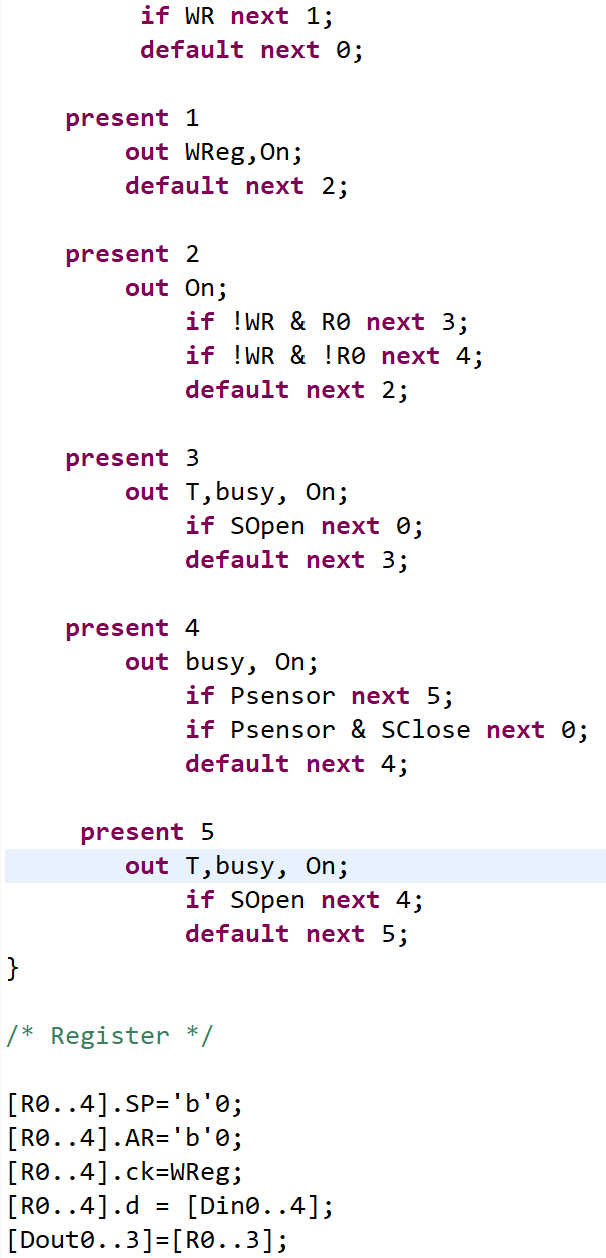


1. Descrição CUPL do bloco *Key Transmitter*





1.  Descrição CUPL do bloco *Controller*



1. Código *Kotlin* da classe *HAL*

object HAL {

var out = 0

fun init() {

UsbPort.out(out.inv())

}

fun isBit(mask: Int): Boolean {

return readBits(mask) > 0

}

fun readBits(mask: Int):Int {

val x = UsbPort.`in`().inv()

return x.and(mask)

}

fun writeBits(mask: Int, value: Int) {

out = mask.inv().and(out)

out = value.and(mask).or(out)

UsbPort.out(out.inv())

}

fun setBits(mask: Int){

out = mask.or(out)

UsbPort.out(out.inv())

}

fun clrBits(mask:Int){

out = mask.inv().and(out)

UsbPort.out(out.inv())

}

}

1. Código *Kotlin* da classe *KBD*

object KBD {

const val NONE = 0

private const val ACK\_MASK = 0x80

private const val DVAL\_MASK = 0x10 //0x80 -> 0x10 is for simulation purposes

private const val KEY\_VALUE = 0x0F

private const val SERIAL\_INTERFACE = true

private val KEYBOARD= charArrayOf('1', '4', '7','\*','2','5','8','0','3','6','9','#')

fun init() {

HAL.clrBits(ACK\_MASK)

}

fun getKey():Char{

if(SERIAL\_INTERFACE) return getKeySerial()

else return getKeyParallel()

}

private fun getKeySerial():Char{

val x = KeyReceiver.rcv()

if (x == -1) return NONE.toChar()

return KEYBOARD[x]

}

private fun getKeyParallel():Char {

var x:Char = NONE.toChar()

if (HAL.isBit(DVAL\_MASK)) {

x=KEYBOARD[HAL.readBits(KEY\_VALUE)]

HAL.setBits(ACK\_MASK)

while (HAL.isBit(DVAL\_MASK)){} /\*Waiting for Dval to be 0\*/

HAL.clrBits(ACK\_MASK)

return x

}

return x

}

fun waitKey(timeout: Long): Char {

val temp = Time.getTimeInMillis() + timeout

do {

val x = getKey()

if (x != NONE.toChar())

return x

} while (Time.getTimeInMillis() <= temp)

return NONE.toChar()

}

}

1. Código *Kotlin* da classe *SerialReceiver*

object KeyReceiver {

private const val TX\_CLK = 0x80

const val TXD = 0x40

private const val NUMB\_ITERATION = 6

private val KEY\_ITERATION = (1..4)

fun init(){

HAL.clrBits(TX\_CLK)

}

fun rcv():Int {

var count = 0

var s = -1.0

if(!HAL.isBit(TXD)) {

s=0.0

while (count <= NUMB\_ITERATION) {

HAL.setBits(TX\_CLK)

Time.sleep(5)

HAL.clrBits(TX\_CLK)

Time.sleep(5)

val x = HAL.readBits(TXD)

if (count in KEY\_ITERATION) {

if (x > 0) s += ((2.0).pow(count -1)) /\* Recreation on just one number of the key value \*/

}

count++}}

return s.toInt() /\* If s = -1 the higher code will understand like incorrect value \*/

}

}

1. Código *Kotlin* da classe *LCD*

object LCD {

private const val LINES = 2

private const val COLS = 16 // Display dimension

private const val ENABLE = 0x20

private const val RS = 0x10

private const val LCD\_DATA = 0x0F

private const val LCD\_LINE = 0x40 //If wanted to write at the second line just need to add 0x40

private const val DISPLAY\_CLEAR = 0x01

private const val CURSOR\_CMD = 0x80

private const val DISPLAY\_OFF = 0x08 //D -> 0 C&B -> Dont Care

private fun writeNibble(rs: Boolean, data: Int) {

// RS -> UsbPort.i4

if (rs){

HAL.setBits(RS)

}else{

HAL.clrBits(RS)

}

// EnableOn -> i5

HAL.setBits(ENABLE)

//Data

HAL.writeBits(LCD\_DATA,data)

// EnableOff -> i5

HAL.clrBits(ENABLE)

Time.sleep(2)

}

private fun writeByte(rs: Boolean, data: Int) {

writeNibble(rs,data.shr(4)) // /16 == ShiftRight 4 times

Time.sleep(2)

writeNibble(rs,data)

}

private fun writeCMD(data: Int) {

writeByte(false,data)

}

private fun writeDATA(data: Int) {

writeByte(true,data)

}

fun init() {

/\*\*

\* All the "fly" variables, like 5 or 0x08.. It's for the LCD configuration

\* They're times and commands got in the manual

\*/

Time.sleep(80)

writeNibble(false,0x03)

Time.sleep(5)

writeNibble(false,0x03)

Time.sleep(1)

writeNibble(false,0x03)

writeNibble(false,0x02)

//N foi 1 porque quer-se escrever em duas linhas, o de F = 0 para se obter o tamanho de 5 por 8 em cada caracter

writeCMD(0x28)

writeCMD(0x08)

writeCMD(0x01)

//de I/D =1 porque quer-se que o cursor incremente ao escrever e o de S=0 para não haver um Shift no display logo na sua inicialização

writeCMD(0x06)

writeCMD(0x0F)

}

// Char write at the position.

fun write(c: Char) {

writeDATA(c.toInt())

}

// String write at the position.

fun write(text: String) {

text.forEach { write(it) }

}

fun cursor(line: Int, column: Int) {

val x = column + (line\* LCD\_LINE)

writeCMD(x+ CURSOR\_CMD)

}

fun clear() {

writeCMD(DISPLAY\_CLEAR)

}

fun off(){

writeCMD(DISPLAY\_OFF)

}

}

1. Código *Kotlin* da classe *TUI*

object TUI {

private const val LCD\_COLUMNS = 16

fun key(l:Int, vis :Boolean):Int{

var s = -1.0

var i =0

do {

val x = KBD.waitKey(5000)

if ((s == -1.0 && x == '\*') || x == KBD.NONE.toChar()) return -1

if (x == '\*') {

lineClear(1)

if (l==4) writeleft("PIN:",1) /\* If l=4 -> Pin, if l=3 -> User \*/

else writeleft("USER:",1)

s=-1.0

i=0

} else {

if (i ==0)s+=1.0

if(x in ('0'..'9')) {

s += (x - '0') \* ((10.0).pow(l - i - 1)) //

i++

if (!vis) {

LCD.write('\*')

} else {

LCD.write(x) }}}

}while (i<l)

return s.toInt()

}

fun writeleft(s:String,line:Int){

LCD.cursor(line,0)

LCD.write(s)

}

fun writecenter(s:String,line:Int){

val size = (LCD\_COLUMNS-s.length)/2.toInt()

LCD.cursor(line,size)

LCD.write(s)

}

fun writeright(s:String,line:Int){

val size = LCD\_COLUMNS-s.length

LCD.cursor(line,size)

LCD.write(s)

}

fun time():String{

val time = LocalDateTime.now()

val format = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm")

return time.format(format)

}

fun lineClear(line:Int){ /\* Function to prevent clear all the screen but just one line\*/

writeleft(" ",line) }}

1. Código *Kotlin* da classe *FileAccess*

object FileAcess {

fun read(file:String): Array<String?> {

val list = File(file).inputStream()

val lineList = arrayOfNulls<String>(1000)

list.bufferedReader().useLines { lines -> lines.forEach { lineList[it.split(';')[0].toInt()] = it } }

return lineList

}

fun write(file:String, list: List<String>,logNUsers:Boolean) {

val text = FileWriter(file, logNUsers)

for (i in list.indices) {

if (logNUsers) text.append("\n${list[i]}")

else text.append(list[i])

}

text.close()

}

}

1. Código *Kotlin* da classe *Users*

object Users {

private var userlist= arrayOfNulls<Ut>(1000)

fun init(){

listUser()

}

private fun toUser(userId:Int, it: List<String>):Ut{

return Ut(userId,it[1].toInt(),it[2],it[3].toLong(),it[4].toLong())

}

fun listUser(): Array<Ut?> {

val newList = userlist

var i = 0

FileAcess.read("USERS.txt").forEach { line ->

if (line != null){

newList[i] = toUser(i, line.split(';'))

i++}}

return newList}

fun removeUser(userId:Int){

val userList = listUser()

userList[userId] = null

}

fun addUser(userPass:Int, userName:String):Ut?{

for(i in 0..999){

if (userlist[i] == null) {

userlist[i] = Ut(i,userPass,userName,0,0L)

return userlist[i]!! }}

return null}

fun getUser(indiceUser: Int): Ut? {

val list = userlist

if (indiceUser == -1) return null

return list[indiceUser]

}

fun updateList(){

val lista = arrayOfNulls<String>(1000)

val lastList = userlist

for (i in lastList.indices) {

val user = lastList[i]

if (user != null) {

lista[i]= "${i};${user.pass};${user.name};${user.acumulateTime};${user.entryTime}}}

FileAcess.write("USERS.txt",lista.filterNotNull().toList(),false) }

fun updateUser(userId: Int,userAcumulateTime:Long,userEntryTime:Long){

val list = userlist

list[userId]!!.acumulateTime=userAcumulateTime

list[userId]!!.entryTime=userEntryTime

updateList()

}

}

1. Código *Kotlin* da classe *LOG*

object LogFile {

private var listLog= mutableListOf<String>()

fun entryUser(worker:Ut,time:Long){

val calendar = calendarLog(time)

val user = "-> ${worker.user}:${worker.name}"

val entry = calendar + user

listLog += entry

logUpdate(listLog.toList())

}

private fun logUpdate(list: List<String>){

FileAcess.write("LOG.txt",list,true)

}

fun awayUser(worker:Ut,time:Long){

val calendar = calendarLog(time)

val user = "<- ${worker.user}:${worker.name}"

val away = calendar + user

listLog += away

logUpdate(listLog.toList())

}

fun calendarLog(time:Long):String{

val calendar=Calendar.getInstance()

calendar.setTimeInMillis(time)

val ampm = calendar.get(Calendar.AM\_PM)

val hour = timeAmPm(ampm,calendar.get(Calendar.HOUR))

val minute= calendar.get(Calendar.MINUTE)

val day = calendar.get(Calendar.DATE)

val month = calendar.get(Calendar.MONTH) + 1 /\*\* Because indeces \*\*/

val year = calendar.get(Calendar.YEAR)

return "$day/$month/$year $hour:$minute "

}

fun calendarAway(time: Long):String{

val calendar=Calendar.getInstance()

calendar.setTimeInMillis(time)

val dayWeek = intToDay(calendar.get(Calendar.DAY\_OF\_WEEK))

val ampm = calendar.get(Calendar.AM\_PM)

val hour = timeAmPm(ampm,calendar.get(Calendar.HOUR))

val minute= calendar.get(Calendar.MINUTE)

return "$dayWeek.$hour:$minute "

}

private fun timeAmPm(ampm:Int, time: Int):Int{

return if (ampm == 1 ) time + 12

else time

}

private fun intToDay(day:Int):String{

return when(day){

1 -> "Sun"

2 -> "Mon"

3 -> "Tue"

4 -> "Wed"

5 -> "Thu"

6 -> "Fri"

else -> "Sat"

}

}

}

1. Código *Kotlin* da classe *Door*

object Door{

private const val WR\_MASK = 0x40

private const val BUSY\_MASK = 0x20

private const val D\_OUT\_MASK = 0x1F

private const val MAX\_SPEED = 0x0F

private const val OPEN\_CMD = 0x10

private const val CLOSE\_CMD = 0x00

fun init(){

HAL.clrBits(WR\_MASK)

}

/\*\*

\* V0~3 -> 0x0F

\*

\* OpenClose -> OPEN = 0x10 & CLOSE = 0x00

\*/

fun open(speed:Int){

var spd = speed

if (spd > MAX\_SPEED) spd = MAX\_SPEED

val x = OPEN\_CMD + spd /\*Open action + speed\*/

while (HAL.isBit(BUSY\_MASK)){} /\*Waiting for the busy signal\*/

HAL.writeBits(D\_OUT\_MASK, x)

HAL.setBits(WR\_MASK)

HAL.clrBits(WR\_MASK)

}

fun close(speed: Int){

var spd=speed

if (spd > MAX\_SPEED) spd=MAX\_SPEED

val x = CLOSE\_CMD + spd /\*Close action + speed\*/

while (HAL.isBit(BUSY\_MASK)){} /\*Waiting for the busy signal\*/

HAL.writeBits(D\_OUT\_MASK,x)

HAL.setBits(WR\_MASK)

HAL.clrBits(WR\_MASK)

}

fun isFinished():Boolean{

return !HAL.isBit(BUSY\_MASK)

}

}

1. Código *Kotlin* da classe *M*

object Maintenance {

fun init(){

LCD.clear()

TUI.writecenter("Out Of Service",0)

TUI.writecenter("\*\_\*",1)

systemModem()

}

private fun systemModem(){

println("What operation you want to proceed? \n 1 - Add User \n 2 - Remove User \n 3 - User List \n 4 - ShutDown ")

when(readLine()!!){

"1" -> addUser()

"2" -> removeUser()

"3" -> userList()

"4" -> shutDown()

else -> operationNotFound()

}

}

private fun operationNotFound(){

println("Operation Not Found \n ")

restartOperations()

}

private fun addUser(){

val name = name()

val pass = pass()

val add= Users.addUser(pass,name)

if (add != null) println("You user is: $add")

else println("Sorry, userList is full xOxO")

restartOperations()

}

private fun name():String{

print("UserName(Máx 16 Chars): ")

val name = readLine()!!

if (name.length>16){

println("Please choose one UserName with maxium 16 chars")

name()

}

return name

}

private fun pass():Int{

print("UserPass(4 Chars): ")

val pass = readLine()!!

if (pass.length !=4){

println("Please choose one UserPass with 4 chars")

pass()

}

return pass.toInt()

}

fun removeUser(){

println("Insert UserID")

val userId = readLine()!!.toInt()

val user = Users.getUser(userId)

if (user != null){

println(user.name + ",Is this the User to remove? Yes or No")

val confirmation = readLine()!!

if ("Y" in confirmation || "y" in confirmation) {

Users.removeUser(userId)

}

}else{

println("User Not Found \n")

}

restartOperations()

}

private fun userList(){

Users.listUser().filterNotNull().forEach{println(it)}

restartOperations()

}

fun shutDown(){

LCD.off()

Users.updateList()

}

private fun restartOperations(){

APP.mode()

}

}

1. Código *Kotlin* da classe *WorkingTimeRecorder* *- App*

object APP {

private const val DOOR\_OPEN\_VELOCITY= 11

private const val DOOR\_CLOSE\_VELOCITY= 11

private const val MMASK = 0x80

private const val TENTNUMB = 3

private const val WRITEACTIONSLINE = 1

private fun user():Ut {

TUI.writeleft("USER:", WRITEACTIONSLINE)

while (HAL.isBit(KeyReceiver.TXD)){

if (HAL.readBits(MMASK) != 0) Maintenance.init()

}

val userNumb= TUI.key(3, true)

return if (getUser(userNumb)!= null) {

getUser(userNumb)!!

} else {

lineClear(WRITEACTIONSLINE)

TUI.writeleft("USER NOT FOUND", WRITEACTIONSLINE)

Time.sleep(1000)

lineClear(WRITEACTIONSLINE)

user()

}

}

private fun pass(){

val user = user()

lineClear(WRITEACTIONSLINE)

for (i in 1..TENTNUMB) {

TUI.writeleft("PASS:", WRITEACTIONSLINE)

val code = TUI.key(4, false)

if (code == user.pass) {

return doorAction(user)

}

lineClear(WRITEACTIONSLINE)

TUI.writeleft("PASS ERROR", WRITEACTIONSLINE)

Time.sleep(1000)

lineClear(WRITEACTIONSLINE)

}

return wrongPass()

}

private fun wrongPass(){

return pass()

}

private fun doorAction(worker:Ut){

LCD.clear()

if(worker.entryTime > 0L) awayDoor(worker)

else entryDoor(worker)

}

private fun moveDoor(){

Door.open(DOOR\_OPEN\_VELOCITY)

Time.sleep(3000)

Door.close(DOOR\_CLOSE\_VELOCITY)

}

private fun entryDoor(worker:Ut){

val entryTime = Time.getTimeInMillis()

TUI.writecenter("Welcome", 0)

TUI.writecenter(worker.name, 1)

moveDoor()

LogFile.entryUser(worker,entryTime)

Users.updateUser(worker.user,worker.acumulateTime,entryTime)

}

private fun awayDoor(worker: Ut){

val awayTime = Time.getTimeInMillis()

val acumulateTime = awayTime - worker.entryTime

TUI.writeleft(LogFile.calendarAway(worker.entryTime),0)

TUI.writeleft(LogFile.calendarAway(awayTime),1)

TUI.writeright(millisToHours(acumulateTime),1)

Time.sleep(3000)

LCD.clear()

TUI.writecenter("Good-Bye",0)

TUI.writecenter(worker.name,1)

moveDoor()

LogFile.awayUser(worker,awayTime)

Users.updateUser(worker.user,acumulateTime,0L)

}

private fun millisToHours(millis:Long):String{

var time = millis

val hour = (time/(60\*60\*1000))

time -= hour \* (60 \* 60 \* 1000)

val minutes = (time/(60\*1000))

time -= minutes\*(60\*1000)

return ("${hour}:${minutes}")

}

private fun lineClear(line:Int){ /\* Function to prevent clear all the screen but just one line\*/

TUI.writeleft(" ",line)

}

private fun restart(){

Time.sleep(1500)

LCD.clear()

mode()

}

private fun appPlay (){

LCD.clear()

TUI.writeright(TUI.time(),0)

pass()

restart()

}

fun mode(){

if (HAL.readBits(MMASK) != 0) Maintenance.init()

else appPlay()

}

}