

**Licenciatura em Engenharia Informática e de Computadores**

Relógio de Ponto

(*Working Time Recorder*)

ERRATA

Ricardo Henriques (A48322@alunos.isel.pt)

João Magalhães (A48348@alunos.isel.pt)

Projeto

de

Laboratório de Informática e Computadores

2020 / 2021 verão

1 de julho de 2021

[1 Introdução 2](#_Toc76115536)

[2 Arquitetura do sistema 2](#_Toc76115537)

[A. Descrição CUPL do bloco Door *Controller*  3](#_Toc76115538)

[B. Código *Kotlin* da classe *Door* 4](#_Toc76115539)

[C. Código *Kotlin* da classe *M* 5](#_Toc76115540)

[D. Código *Kotlin* da classe *WorkingTimeRecorder* *- App* 6](#_Toc76115541)

# Introdução

Após envio do relatório final, foi possível verificar algumas incoerências e problemas na implementação do projeto, segue então em formato de errata todas as funções que foram alteradas, sendo estas as que se devem ser consideradas agora como o material usado para a implementação.

Todas as funções estão devidamente identificadas com as suas devidas alocações por classes; todas as classes e funções não envolvidas nesta errada, provem de as mesmas não terem sofrido alterações.

Existe a exceção que no objeto da *Maintence*, a função *restartOperations()*, não se encontra alterada mas sim retirada.

# Arquitetura do sistema

Em relação à arquitetura do sistema em si, foram encontrados dois defeitos nas devidas implementações e por isso alteradas, a primeira alteração encontrada foi em relação à frequência dada aos *clocks* da *DOOR*, os mesmo foram alterados para *100kHZ*, em simulação era possível verificar a aprovação da implementação antes ponderada, mas sem ser em simulação, foi esquecido o fator do *delay*, o que implicava que as instruções da *DOOR*, muitas vezes não eram lidas e consideradas.

A segunda alteração realizada em relação à arquitetura foi a ASM da DOOR, como visível na figura 1., foi adicionada um estado da verificação e aguardo da variável WR, voltar a ‘0’ para garantir a receção e não se ter problemas em fluxo retirando também a indicação de ligar a maquina logo na fase inicial, esperando então só pelas ações da porta quando as mesmas forem chamadas, e a outra alteração foi em relação à implementação do fecho da porta, onde na presença de uma pessoa, dava-se logo a instrução de fecho sem antes ser verificada ainda a permanência de pessoa no local.

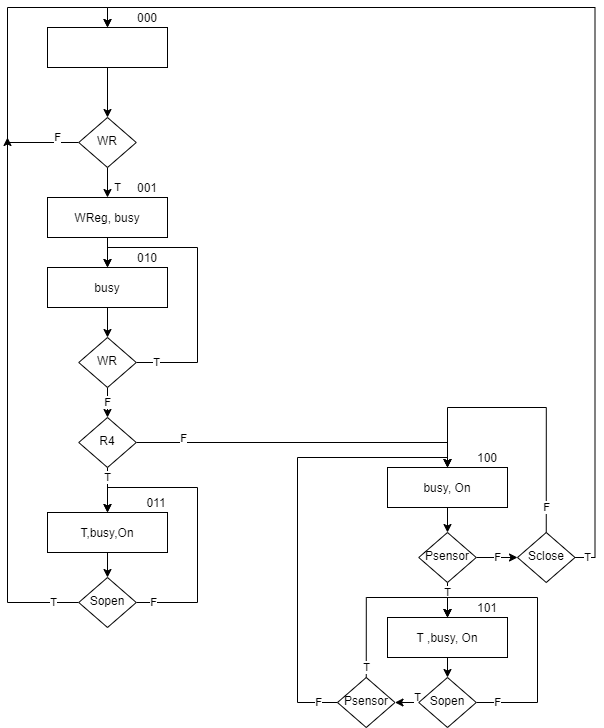
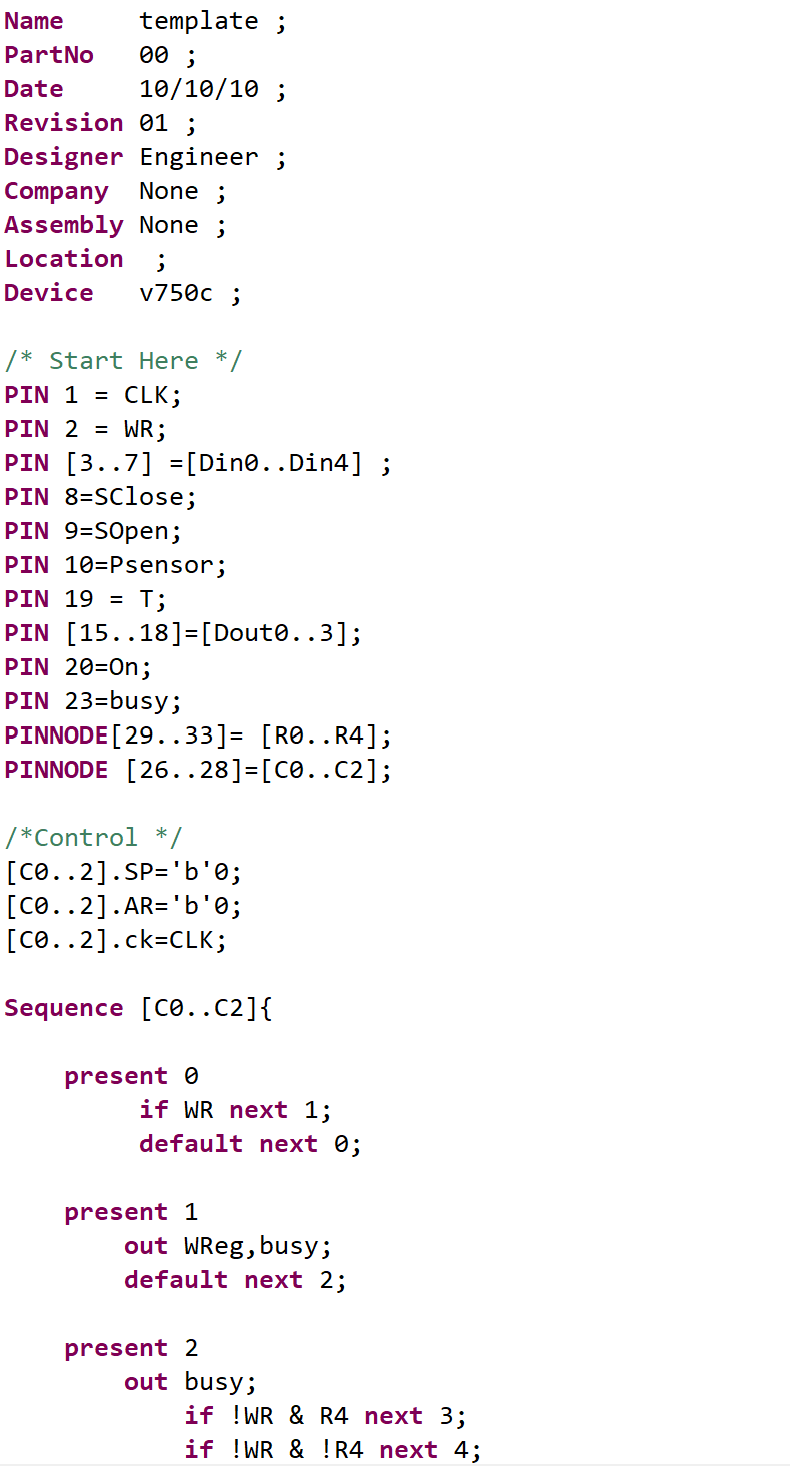
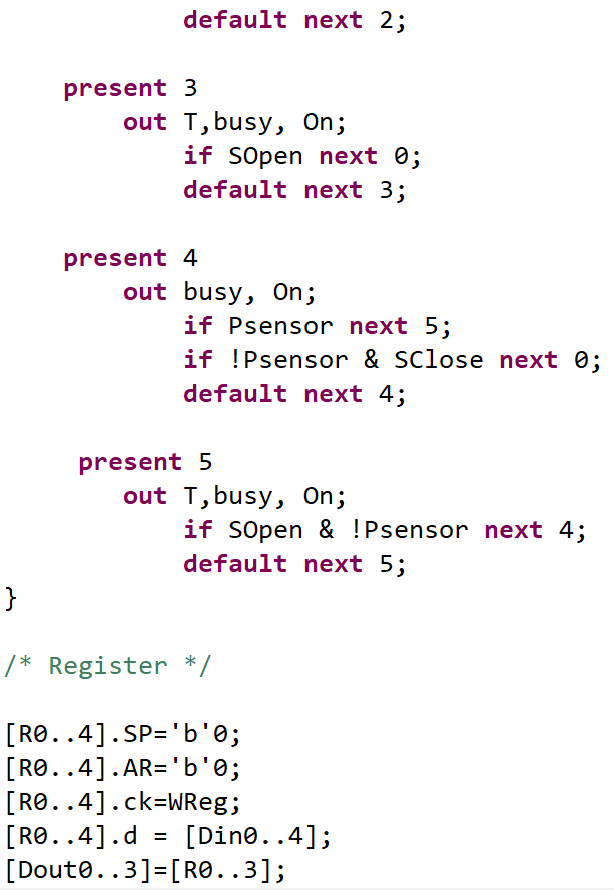


Figura 1. - *ASM DOOR*

1. Descrição CUPL do bloco Door *Controller* 
2. Código *Kotlin* da classe *HAL*

fun readBits(mask: Int):Int {

UsbPort.`in`()

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

return x.and(mask)

}

1. Código *Kotlin* da classe *Door*

fun open(speed:Int){

var spd = speed

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

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

while (!isFinished()){} /\*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 (!isFinished()){} /\*Waiting for the busy signal\*/

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

HAL.setBits(WR\_MASK)

HAL.clrBits(WR\_MASK)

}

1. Código *Kotlin* da classe *M*

private fun systemModem() {

println("What operation you want to proceed? \n 1 - Add User \n 2 - Remove User \n 3 - User List \n 4 - Change Mode \n 5 - ShutDown ")

when (readLine()!!) {

"1" -> addUser()

"2" -> removeUser()

"3" -> userList()

"4" -> APP.mode()

"5" -> shutDown()

else -> operationNotFound()

}

}

private fun operationNotFound() {

println("Operation Not Found \n ")

systemModem()

}

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")

systemModem()

}

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")

}

systemModem()

}

private fun userList() {

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

systemModem()

}

fun shutDown() {

LCD.off()

Users.updateList() }

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

private fun user():Ut {

TUI.writeleft("USER:", WRITEACTIONSLINE)

val userNumb = TUI.key(3, true)

if (userNumb == -1) appPlay()

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

getUser(userNumb)!!

} else {

TUI.lineClear(WRITEACTIONSLINE)

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

Time.sleep(1000)

TUI.lineClear(WRITEACTIONSLINE)

user()

}

}

private fun pass(){

val user = user()

TUI.lineClear(WRITEACTIONSLINE)

for (i in 1..TENTNUMB) {

TUI.writeleft("PASS:", WRITEACTIONSLINE)

val code = TUI.key(4, false)

if (code == -1) appPlay()

if (code == user.pass) {

verifyChangePass(user)

return doorAction(user)

}

TUI.lineClear(WRITEACTIONSLINE)

TUI.writeleft("PASS ERROR", WRITEACTIONSLINE)

Time.sleep(1000)

TUI.lineClear(WRITEACTIONSLINE)

}

return wrongPass()

}

private fun verifyChangePass(user:Ut) {

if (KBD.waitKey(2000) == '#') changePass(user)

}

private fun changePass(user:Ut){

LCD.clear()

TUI.writecenter("NEW PIN",0)

TUI.writeleft("PIN:",1)

val code1 = TUI.key(4,false)

if (code1 == -1) appPlay()

LCD.clear()

TUI.writecenter("CONFIRM PIN",0)

TUI.writeleft("PIN:",1)

val code2 = TUI.key(4,false)

if (code2 == -1) appPlay()

LCD.clear()

if (code1==code2){

user.pass=code1

TUI.writecenter("PIN CONFIRMED",0)

Time.sleep(1500)

return

}

TUI.writecenter("PIN ERROR",0)

changePass(user)

return

}

private fun moveDoor(){

Door.open(DOOR\_OPEN\_VELOCITY)

Time.sleep(3000)

Door.close(DOOR\_CLOSE\_VELOCITY)

while (!Door.isFinished()){}

}