O módulo *Door Controller* é constituído por 3 blocos, sendo dois desses hardware e um dos componentes software. A primeira componente baseia-se num controlador em software, a segunda é um controlador em hardware da porta e a última componente é a porta em questão.

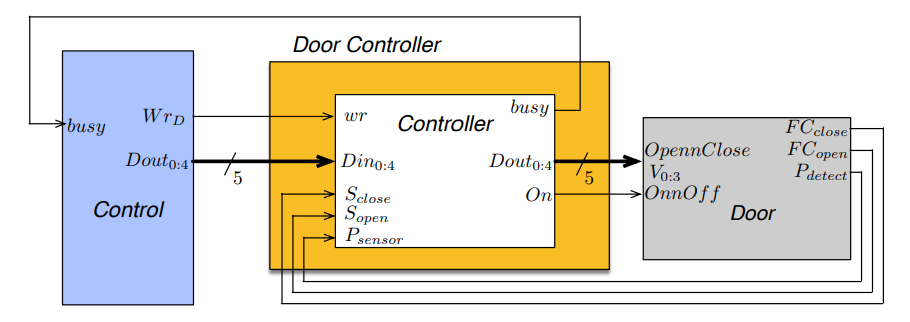
****

Figura 1 – Diagrama de blocos do módulo *Door Controller*

# Controller

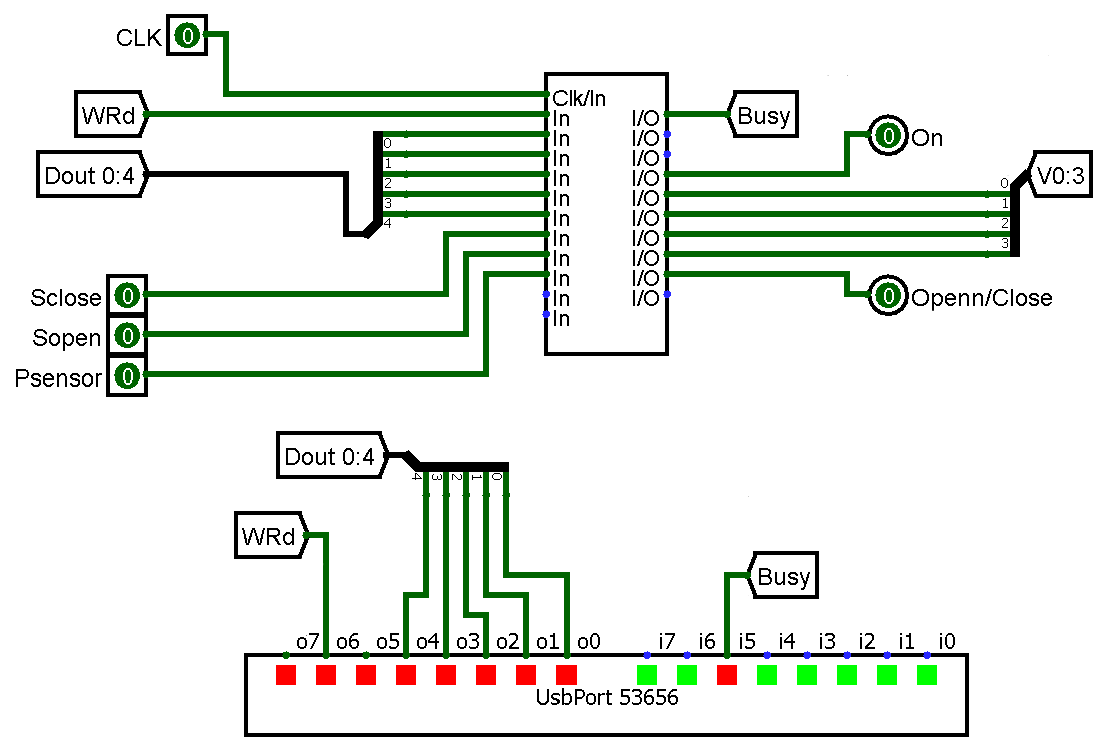
O bloco *Controller* implementa uma máquina de estados que pode ser observada na figura 2 a sua implementação em *logisim* e o seu *ASM-chart* na figura 3. É possível com esta implementação de *hardware* realizar uma ponte do *software* ao dispositivo da porta, tendo assim acesso e controlo no manuseamento da mesma. Sendo possível então enviar um simples comando e obtendo o processamento completo através de uma máquina de estados obter uma porta em funcionamento, com simples funcionamento da parte de hardware só é demonstrada a implementação da *pal* com as conexões de entrada e saídas da mesma na figura seguinte.

Figura 2 – Implementação Logisim do Controller

O bloco *Controller* foi implementado pela máquina de estados representada em *ASM-chart* na 3. Tendo como estado inicial o repouso, é permitida o avanço para estado seguinte conforme a indicação que existe uma leitura de dados a ser feita vinda do *Control* (*WR*), após passagem para o primeiro estado, onde se liga a máquina e a indicação da mesma estar ocupada é retirada desse estado. O estado seguinte difere consoante o bit de maior peso, sendo esse o que indica se a porta irá abrir ou fechar, a variável *T* de saída é a indicação que a operação é de abertura, no caso se for de abertura a porta só tem de se preocupar com tal até ao término da mesma, se for de fecho tem de estar em verificação até à conclusão do fecho se não existe aparência de alguma pessoa, se tal deve de executar o método de abertura e depois pode realizar de novo o método de fecho ( pode ser observado no estado 3 e 4).

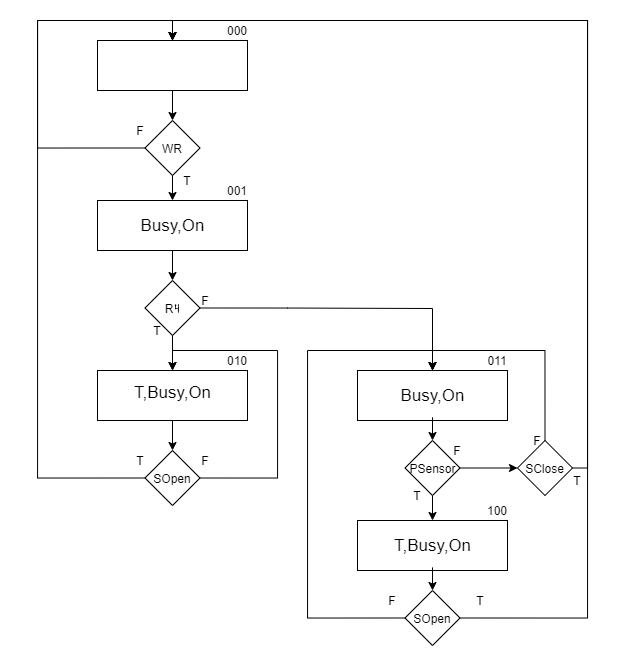
A descrição hardware do bloco *Controller* em CUPL encontra-se no Anexo A.

Figura 3– Máquina de estados do bloco *Controller*

Com base nas descrições do bloco *Controller* implementou-se parcialmente o módulo *Door Controller* de acordo com o esquema elétrico representado no Anexo B. Os *clocks* dados pela máquina são equivalentes ao clocks do modulo do *Key Decode, 1KHz*, permitindo assim que o projeto num todo trabalhe com união e todos poderem usufruir da mesma frequência de trabalho, excetuados os módulos que necessitam de uma menor por observações realizadas nos mesmos.

Em relação à latência de entrada no hardware, é muito baixa devido a ser providenciada por parte do software o que proporciona tempos de resposta em comparação com os que são obtidos através dos mecanismos físicos. Em relação à latência de processamento e de saída da mesma, são valores maior que os obtidos através do software, mas podem ser considerados "desprezáveis”, pois não afetam o funcionamento de um todo.

# Interface com o *Control*

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem *Kotlin* e seguindo a arquitetura lógica apresentada na Figura .





Figura 4 – Diagrama lógico do módulo *Control* de interface com o módulo *Door Controller*

A classe *Door* desenvolvida é descrita na secção 2.1., e o código fonte desenvolvido no Anexo C.

## Classe *Door*

A classe *Door* foi implementada em 3 funções, excluindo a função de inicialização da mesma. A mesma, a função de inicialização baseia-se em colocar o *WR* a *bit* lógico ‘0’ para o sistema saber que se encontra no primeiro estado da máquina e assim só avançar do mesmo quando a chamada de alguma das funções quer de abertura ou de fecho.

Quer a função de abertura e de fecho, possuem um parâmetro que é a velocidade (*speed*), sendo a mesma que vai determinar a velocidade com que a porta executa a sua ação, esta devido a ser representada no máximo com 4 bits, o valor máximo que pode ser alcançado é de 15. O código em *software* não verifica a utilização da existência de pessoas ou se já se encontra aberta ou fechada, o único parâmetro a que tem acesso é o de ocupado (*busy*), o que permite o sistema saber se a ação foi concluída ou não.

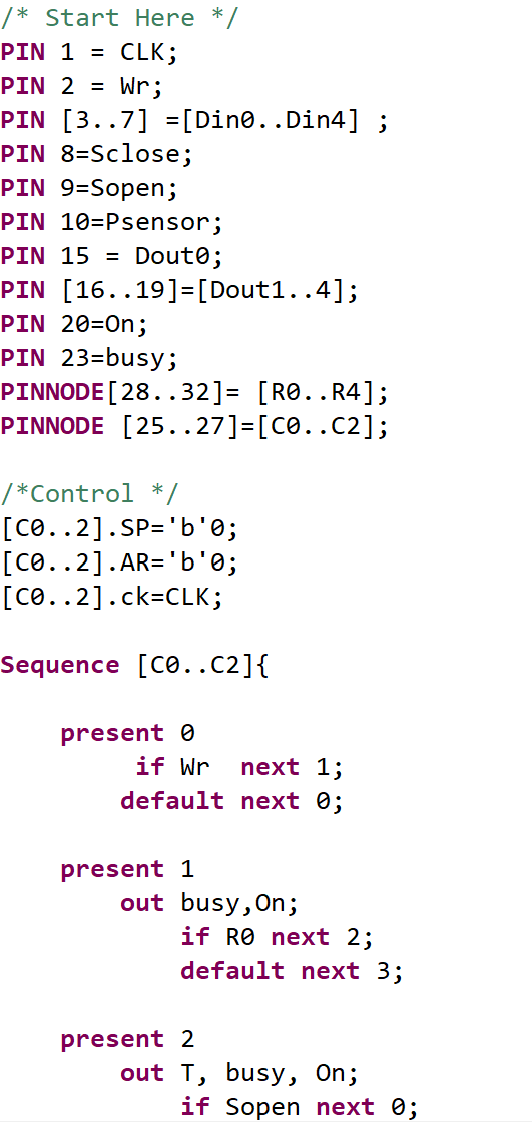
Ambas as funções funcionam então de um método geral, semelhantes, verificando primeiro se a velocidade imposta não ultrapassa o máximo de 15 escrevendo então o valor da máscara de saída baseado na velocidade mais a indicação de se vai ser aberta ou fechada, que provem da adição de *0x10*, pois ao ser usado como o maior bit do inteiro que é aplicado ao *HAL*, permite que o mesmo não altere a velocidade desejada. Ambas as operações são concluídas com a espera da indicação que a porta já não se encontra mais ocupada; esta ação pode ser com a utilização da última função implementada onde a mesma indica se houve término conforme o valor do *bit* lógico de *busy*.

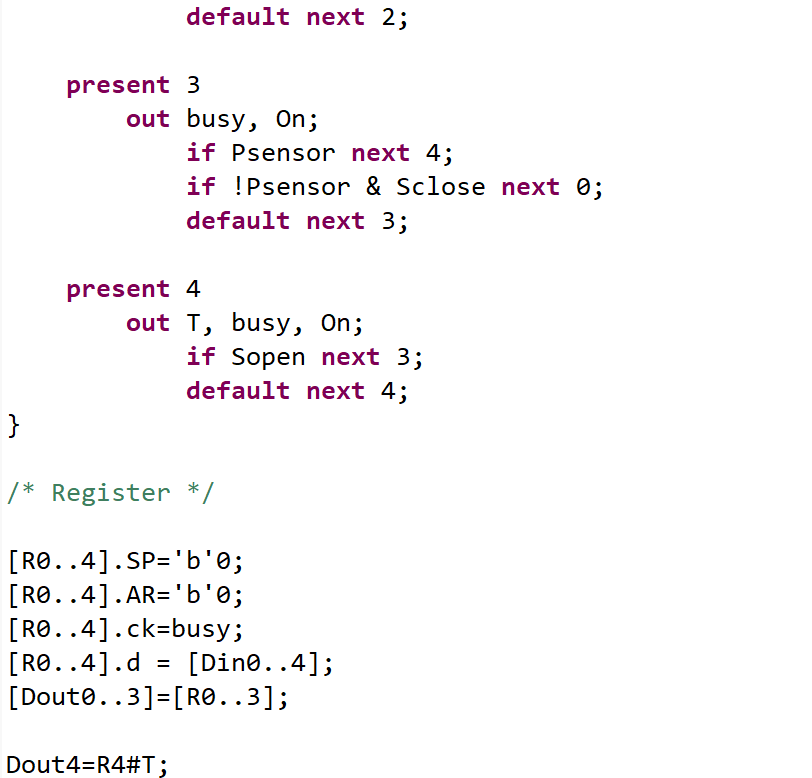
# Conclusões

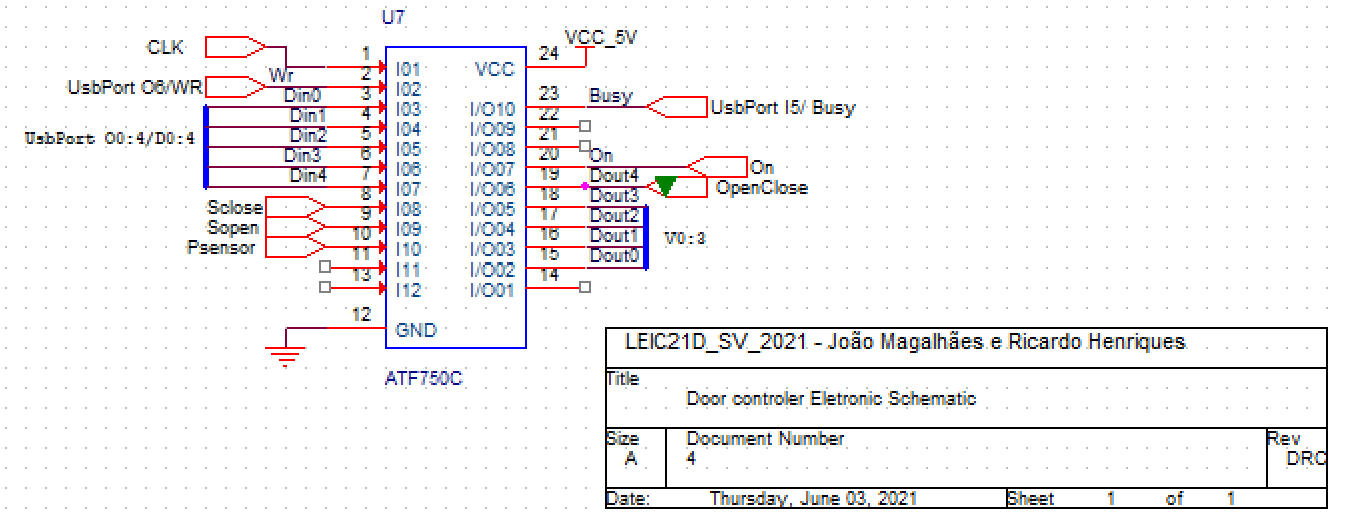
O módulo do *Door Controller,* é implementado com as suas duas vertentes necessárias para tal realização, a sua componente de *software* a de hardware que permitem, em conjunto, que a porta consiga ter e realizar os devidos propósitos. Com a componente *software* é possível ter-se por parte de todo o sistema o conhecimento se a porta se encontra em uso ou não o que acaba por ser prático e vantajoso; *software* também permite mudar a velocidade em questão o que ajuda a mudar conforme as situações que forem impostas e não só neste caso específico.

Tudo isto é possível por intermediário do bloco de *hardware*, que através da sua implementação com a atribuição da máquina de estados permite ao sistema gerir quer as informações transmitidas pelo software quer as informações que vão sendo transmitidas pela porta e realizar o melhor com as mesmas, quer abrir no fecho em caso da presença de pessoa quer nos casos mais simples e normais só de abertura e fecho sem interrupções da mesma.

2. Descrição CUPL do bloco *Controller*





1. Esquema elétrico do módulo *Door Controller*

1. Código *Kotlin* da classe *Door*

object Door{

private const val WrMask = 0x40

private const val BusyMask = 0x20

private const val DOutMask = 0x1F

private const val MaxSpeed = 0x0F

fun init(){

HAL.clrBits(WrMask)

}

/\*\*

\* V0~3 -> 0x0F

\*

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

\*\*/

fun open(speed:Int){

var spd=speed

if (spd > MaxSpeed) spd=MaxSpeed

val x = 0x10 + spd /\*Open action + speed\*/

HAL.writeBits(DOutMask,x)

HAL.setBits(WrMask)

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

HAL.clrBits(WrMask)

}

fun close(speed: Int){

var spd=speed

if (spd > MaxSpeed) spd=MaxSpeed

val x = 0x00 + spd /\*Close action + speed\*/

HAL.writeBits(DOutMask,x)

HAL.setBits(WrMask)

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

HAL.clrBits(WrMask)

}

fun isFinished():Boolean{

while (HAL.isBit(BusyMask)) {

return false

}

return true

}

}

fun main(){

Door.init()

while (true) {

val x = (-100..100).random() /\* TestCode to give random number so door open or close with the value of x\*/

println(x)

if (x <= 0) {

Door.close(12)

} else {

Door.open(12)

}

while (!Door.isFinished()){}

}

}