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 1.

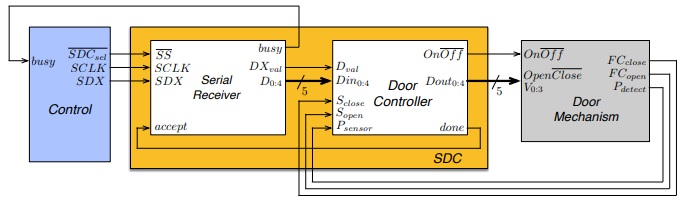
****

Figura 1 – 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 12, tendo como primeiro bit de informação, o bit (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 no nível lógico “0”.

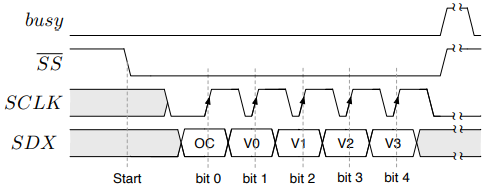
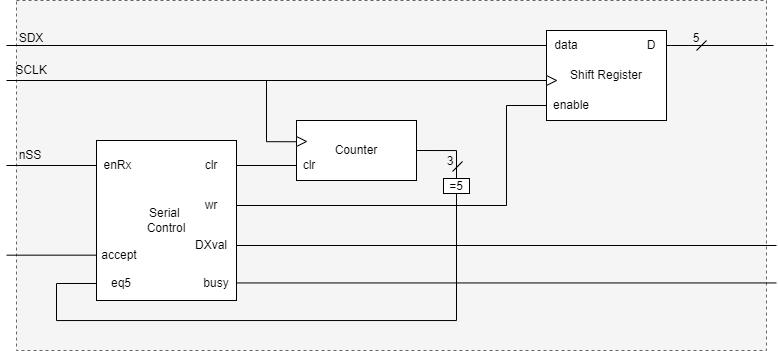


Figura 2 – Protocolo de comunicação do Serial Door Controller

# Serial Receiver

O bloco Serial Receiver do SLCDC é constituído por três blocos principais: i) um bloco de controlo; ii) um contador de bits recebidos; e iii) um bloco conversor série paralelo, designados respetivamente por Serial Control, Counter, e Shift Register. O Serial Receiver deverá ser implementado com base no diagrama de blocos apresentado na Figura 3.

Figura 3 – Diagrama de blocos do Serial Receiver

O bloco *Shift Register* foi implementado de acordo com o diagrama de blocos representado na figura 4.

O bloco *Counter* foi implementado de acordo com o diagrama de blocos representado na figura 5.

O bloco *Serial Control* foi implementado pela máquina de estados representada em *ASM-chart* na figura figura 6.

A máquina de estados do Serial Control é composta por 4 estados: **WAITING**, **RECEIVING**, **END** **e** **WAITING\_ACCEPT**.

No 1º estado (**WAITING**), como o nome indica ficámos à espera que uma trama pudesse começar a ser recebida através da verificação do sinal **EnRx**; caso este tome o valor lógico ‘1’ significa que continuamos no estado **WAITING** e por sua vez o sinal **clr** continua com o valor lógico ‘1’ de modo a que o bloco Counter continue com o valor da contagem com o valor ‘0’ até que uma trama possa começar a ser recebida. Caso contrário, pode-se começar a receber a trama e então prosseguimos para o estado **RECEIVING**. Chegando ao 2º estado (**RECEIVING**), é ativada a saída wr ativando o sinal de **enable** do bloco Shift Register e de seguida espera-se que os 5 bits da trama sejam recebidos através da verificação dos sinais **EnRx** e eq5, respetivamente; caso o sinal **EnRx** tenha o valor lógico ‘0’ significa que ficamos no estado **RECEIVING**, caso contrário iremos fazer a verificação do sinal **eq5**. Por sua vez, se o sinal **eq5** tiver o valor lógico ‘0’ voltaremos para o estado **WAITING**, caso contrário prosseguimos para o estado **END**. Chegando ao 3º estado (**END**), é ativada a saída **DXval** ativando o sinal **Dval** do bloco Dispatcher indicando-lhe que a trama a receber está “pronta” e é ativado também a saída **Busy**, de forma a informar ao Control que ainda estamos a processar a trama , neste caso, a enviá-la. De seguida espera-se que o bloco Dispatcher tenha recebido a trama através da verificação do sinal **Accept**. Caso o valor lógico deste seja ‘0’ mantemo-nos no mesmo estado visto que significa que o bloco Dispatcher ainda não recebeu de facto a trama, caso contrário prosseguimos para o estado **WAITING** **ACCEPT**. Por fim, no estado **WAITING** **ACCEPT** visto que o bloco Serial Receiver e o bloco Dispatcher possuem diferentes clocks e de modo a verificar se o bloco Dispatcher recebeu de facto a trama, é feita novamente uma verificação do sinal **Accept**. Se este tiver o valor lógico ‘1’ significa que pode ainda não ter acabado de receber a trama em questão e por isso permanece no mesmo estado, caso contrário, voltaremos para o estado inicial **WAITING**. Neste estado, mantemos o sinal de **Busy** ativo uma vez que não temos a certeza de o bloco Dispatcher já tratou de facto da trama, como referido anteriormente, daí a necessidade de informar o Control de que ainda não é possível enviar uma nova trama.

A descrição hardware do bloco *Serial Receiver* em VHDL encontra-se no Anexo A.

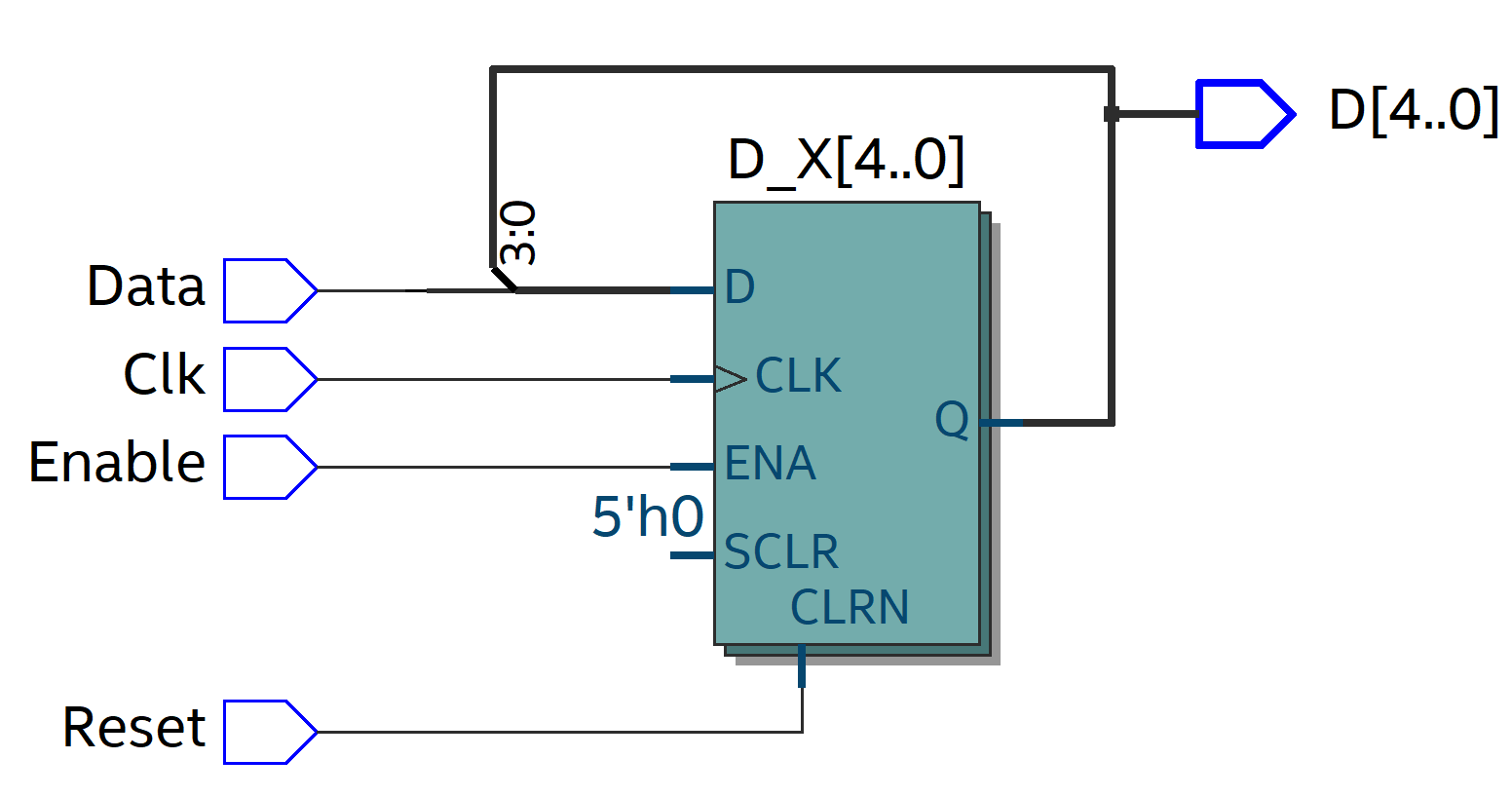


Figura 4 - Diagrama de blocosdo bloco *Shift Register*

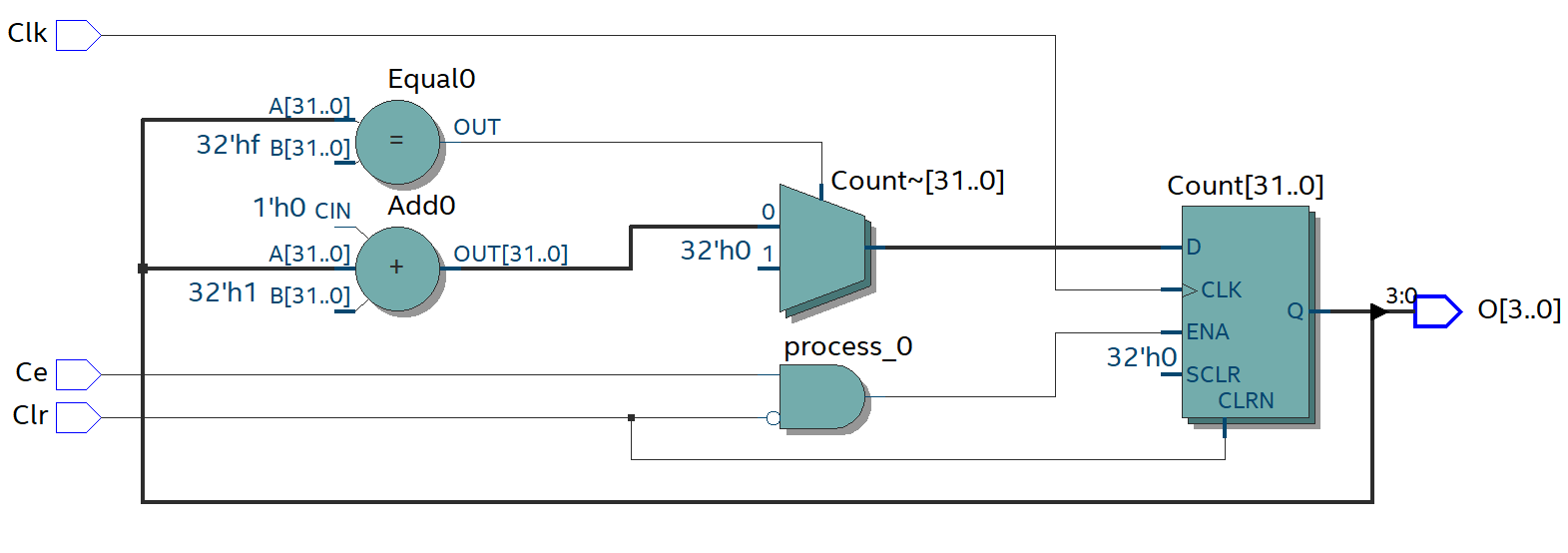


Figura 5 - Diagrama de blocosdo bloco *Counter*

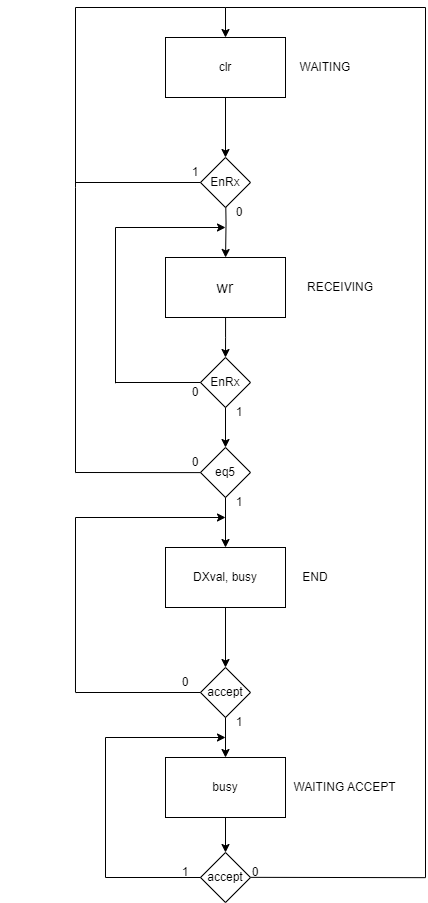


Figura 6 – Máquina de estados do bloco *Serial Control*

1.2 Door Controller

O bloco *Door* *Controller*, 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 e sinal com 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 e colocar o sinal 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*.

O bloco *Door* *Controller* foi implementado pela máquina de estados representada em *ASM-chart* na figura 7.

Assim sendo, este bloco é composto por 4 estados: WAITING, FECHO, ABERTURA e DONE. Inicialmente, no estado WAITING, este bloco aguarda até que seja recebida uma trama válida através da verificação do sinal de entrada DVal. Se tal for o caso, é verificado o bit '0' da trama recebida, que indica se o comando é de abertura ou de fecho. Se o bit for '0', progredi-se para o estado FECHO, caso contrário, avança-se para o estado ABERTURA. No estado FECHO, verifica-se se é detetada uma pessoa na zona da porta através do sensor de presença (sinal Psensor). Caso seja detetada, entramos no estado ABERTURA, interrompendo o fecho da porta e reabrindo-a. Se este não for o caso, aguarda-se até que o sinal Sclose fique ativo (porta fechada), colocando o sinal ONnOFF a nível lógico '1' enquanto a porta se fecha. Posto isto, progredi-se para o estado DONE e o Door Controller sinaliza o Serial Receiver que está pronto para processar uma nova trama através da ativação do sinal done. Para voltar ao estado inicial, verifica-se novamente o sinal Dval de modo a garantir que não houve problemas na transmissão de novas tramas. De volta ao ínicio, se o bit '0' da trama recebida for '1', vamos para o estado ABERTURA. Neste caso, aguarda-se até que a porta abra verificando o sinal Sopen que, enquanto se mantiver a '0', indica que a porta está a abrir a partir do sinal ONnOFF. Após isso, é feita uma nova verificação do bit '0' da trama recebida, visto que, numa situação de fecho onde se detetou a passagem de um indivíduo, deve-se permitir de forma automática o encerramento da porta. Se tal não se verificar, avança-se normalmente para o estado DONE.

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

Com base nas descrições do bloco *Serial Receiver* e *Door Controller* implementou-se o módulo *Serial Door Controller* de acordo com o esquema elétrico representado no Anexo B. Ao contrário do bloco Dispatcher do *Serial LCD Controller,* O bloco *Serial Receiver*e o bloco *Door Controller* não têm uma diferença de velocidade de processamento considerável, visto que o bloco com que interagem (*Door Mechanism*) ao contrário do *LCD*é mais rápido no que toca a tratamento de tramas, fazendo com que o bloco *Door Controller* tenha que ser mais rápido, daí ser excluída a necessidade de se o usar um bloco extra (*ClkDiv*), portanto ambos trabalham com o mesmo sinal de Clk, o da placa DE10-Lite com cerca de 50MHz.

A descrição hardware do bloco *Serial Door Controller* em VHDL encontra-se no Anexo A.

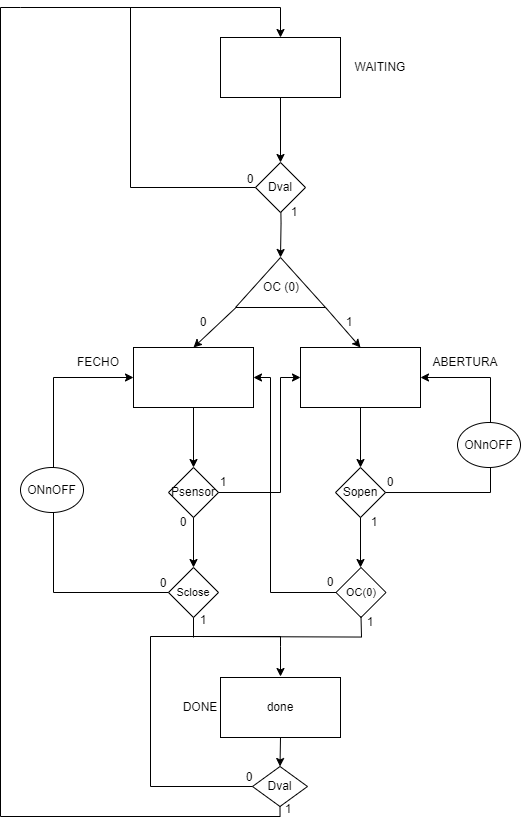


Figura 7 – Máquina de estados do *Door Controller*

# Interface com o *Control*

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem *Kotlin* e seguindo a arquitetura lógica apresentada na Figura 8.

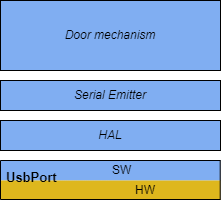


Figura 8 – Diagrama lógico do módulo *Control* de interface com o módulo *Serial Door Controller*

*HAL*, *Serial Emitter e Door Mechanism* desenvolvidos são descritos nas secções 3.1., 3.2 e 3.3, e o código fonte desenvolvido nos Anexos C, D e E, respetivamente.

## *HAL*

O objeto HAL, tem como objetivo servir de ‘ponto de ligação’ entre o UsbPort e o Hardware, ou seja, permitir o funcionamento entre os diversos blocos de código implementados com o Hardware e o software desenvolvido em kotllin. Para isso foram desenvolvidas várias funções como a **isBit**, **readBits**, **writeBits**, **setBits** e **clearBits** que manipulam os bits de entrada e saída do UsbPort. Para se ter noção da última escrita nos bits de saída foi também adicionada uma variável **lastWriting**.

## *Serial Emitter*

O objeto Serial Emmiter tem como função gerar uma trama para o LCD (de acordo com a figura 2). Para tal criou-se 2 funções, **Send** e **isBusy**. Uma vez que tem como objetivo gerar a trama do LCD é necessário interagir com os bits do UsbPort usando então as funções do HAL.

## *Door Mechanism*

O objeto *Door Mechanism* serve simular o mecanismo de abrir e fechar uma porta, através do envio de tramas, usando a função **Send** fornecida pelo *Serial Emitter*. Para isso foram criadas funções, tais como: **open**, **close** e **finished**.

# Conclusões

No módulo Serial SDC Controller foram necessários os módulos Serial Receiver (também presente no Serial LCD Controller) e Door Controller, cada um com os seus submódulos.

De forma a garantir uma correta implementação dos mesmos, foram realizados testes tanto na placa DE10-Lite como no simulador "RTL Simulation" do Quartus onde foram utilizados

os módulos test bench desenvolvidos para cada um dos componentes.

Foram reaproveitados ficheiros já utilizados no módulo Serial LCD Controller como o Serial Receiver, cuja única modificação foi o novo sinal de saída "busy".



A Descrição VHDL do bloco *Serial Door Controller*

library ieee;

use ieee.std\_logic\_1164.all;

entity SerialDoorController is

port

(

-- Input ports

nSDCsel : in std\_logic;

SClk : in std\_logic;

Clk : in std\_logic;

SDX : in std\_logic;

Reset : in std\_logic;

Sclose : in std\_logic;

Sopen : in std\_logic;

Psensor : in std\_logic;

-- Output ports

D : out std\_logic\_vector(4 downto 0);

OnNOff : out std\_logic;

Busy : out std\_logic

);

end SerialDoorController;

architecture structural of SerialDoorController is

component SerialReceiver is

port

(

-- Input ports

Clk : in std\_logic;

SDX : in std\_logic;

SClk : in std\_logic;

nSS : in std\_logic;

Accept : in std\_logic;

Reset : in std\_logic;

-- Output ports

D : out std\_logic\_vector(4 downto 0);

DXval : out std\_logic;

Busy : out std\_logic

);

end component;

component DoorController is

port

(

-- Input ports

Clk : in std\_logic;

Reset : in std\_logic;

Dval : in std\_logic;

OC : in std\_logic\_vector(4 downto 0);

Sclose : in std\_logic;

Sopen : in std\_logic;

Psensor : in std\_logic;

-- Output ports

OnNOff : out std\_logic;

Dout : out std\_logic\_vector(4 downto 0);

Done : out std\_logic

);

end component;

signal Done\_X, Dval\_X, Clk\_X, Busy\_X : std\_logic;

signal OC\_X : std\_logic\_vector(4 downto 0);

begin

U0: SerialReceiver port map (Clk => Clk, SDX => SDX, SClk => SClk, nSS => nSDCsel, Accept => Done\_X, Reset => Reset,

D => OC\_X, DXval => Dval\_X, Busy => Busy);

U1: DoorController port map (Clk => Clk, Reset => Reset, Dval => Dval\_X, OC => OC\_X, Sclose => Sclose, Sopen => Sopen,

Psensor => Psensor,

OnNOff => OnNOff, Dout => D, Done => Done\_X);

end structural;

1. Descrição VHDL do bloco *Serial Receiver*

library ieee;

use ieee.std\_logic\_1164.all;

entity SerialReceiver is

port

(

-- Input ports

Clk : in std\_logic;

SDX : in std\_logic;

SClk : in std\_logic;

nSS : in std\_logic;

Accept : in std\_logic;

Reset : in std\_logic;

-- Output ports

D : out std\_logic\_vector(4 downto 0);

DXval : out std\_logic;

Busy : out std\_logic

);

end SerialReceiver;

architecture structural of SerialReceiver is

component SerialControl is

port

(

-- Input ports

Clk : in std\_logic;

EnRx : in std\_logic;

Accept : in std\_logic;

Eq5 : in std\_logic;

Reset : in std\_logic;

-- Output ports

Clr : out std\_logic;

Wr : out std\_logic;

DXval : out std\_logic;

Busy : out std\_logic

);

end component;

component ShiftRegister is

port

(

-- Input ports

Data : in std\_logic;

Clk : in std\_logic;

Enable : in std\_logic;

Reset : in std\_logic;

-- Output ports

D : out std\_logic\_vector(4 downto 0)

);

end component;

component SerialReceiverCounter IS

port

(

-- Input ports

Clk : in std\_logic;

Ce : in std\_logic;

Clr : in std\_logic;

-- Output ports

O : out std\_logic\_vector(3 downto 0)

);

end component;

signal Clr\_X, Wr\_X, Eq5\_X : std\_logic;

signal O\_X : std\_logic\_vector(3 downto 0);

begin

Eq5\_X <= not O\_X(3) and O\_X(2) and not O\_X(1) and O\_X(0);

U0: SerialControl port map (Clk => Clk, EnRx => nSS, Eq5 => Eq5\_X, Accept => Accept, Reset => Reset,

Wr => Wr\_X, Clr => Clr\_X, DXval => DXval, Busy => Busy);

U1: ShiftRegister port map (Clk => SClk, Reset => Reset, Data => SDX, Enable => Wr\_X,

D => D);

U2: SerialReceiverCounter port map (Clk => SClk , Clr => Clr\_X, Ce => '1',

O => O\_X);

end structural;

A. Descrição VHDL do bloco *Door Controller*

library ieee;

use ieee.std\_logic\_1164.all;

entity DoorController is

port

(

-- Input ports

Clk : in std\_logic;

Reset : in std\_logic;

Dval : in std\_logic;

OC : in std\_logic\_vector(4 downto 0);

Sclose : in std\_logic;

Sopen : in std\_logic;

Psensor : in std\_logic;

-- Output ports

OnNOff : out std\_logic;

Dout : out std\_logic\_vector(4 downto 0);

Done : out std\_logic

);

end DoorController;

architecture behavioral of DoorController is

type STATE\_TYPE is (STATE\_WAITING, STATE\_FECHO, STATE\_ABERTURA, STATE\_DONE);

signal CurrentState, NextState: STATE\_TYPE;

begin

--FLIP-FLOP'S

CurrentState <= STATE\_WAITING when Reset = '1' else NextState when rising\_edge(Clk);

--GENERATE NEXT STATE

GenerateNextState:

process (CurrentState, Dval, OC, Sclose, Sopen, Psensor)

begin

case CurrentState is

when STATE\_WAITING => if (Dval = '1' and OC(0) = '1') then

NextState <= STATE\_ABERTURA;

elsif (Dval = '1' and OC(0) = '0') then

NextState <= STATE\_FECHO;

else

NextState <= STATE\_WAITING;

end if;

when STATE\_FECHO => if (Psensor = '1') then

NextState <= STATE\_ABERTURA;

elsif (Psensor = '0' and Sclose = '0') then

NextState <= STATE\_FECHO;

else

NextState <= STATE\_DONE;

end if;

when STATE\_ABERTURA => if (Sopen = '1' and OC(0) = '1') then

NextState <= STATE\_DONE;

elsif (Sopen = '1' and OC(0) = '0') then

NextState <= STATE\_FECHO;

else

NextState <= STATE\_ABERTURA;

end if;

when STATE\_DONE => if (Dval = '1') then

NextState <= STATE\_WAITING;

else

NextState <= STATE\_DONE;

end if;

end case;

end process;

-- GENERATE OUTPUTS

OnNOff <= '1' when ((CurrentState = STATE\_FECHO and Sclose = '0') or (CurrentState = STATE\_ABERTURA and Sopen = '0')) else '0';

Dout(0) <= '1' when (CurrentState = STATE\_ABERTURA) else OC(0);

Dout(4) <= OC(4);

Dout(3) <= OC(3);

Dout(2) <= OC(2);

Dout(1) <= OC(1);

Done <= '1' when (CurrentState = STATE\_DONE) else '0';

end behavioral;

1. Atribuição de pinos do módulo *Serial Door Controller*

set\_location\_assignment PIN\_N5 -to Clk

set\_location\_assignment PIN\_C10 -to SDX

set\_location\_assignment PIN\_C11 -to SClk

set\_location\_assignment PIN\_D12 -to nSDCsel

set\_location\_assignment PIN\_A12 -to Sclose

set\_location\_assignment PIN\_B12 -to Sopen

set\_location\_assignment PIN\_A13 -to Psensor

set\_location\_assignment PIN\_F15 -to Reset

set\_location\_assignment PIN\_A8 -to D[0]

set\_location\_assignment PIN\_A9 -to D[1]

set\_location\_assignment PIN\_A10 -to D[2]

set\_location\_assignment PIN\_B10 -to D[3]

set\_location\_assignment PIN\_D13 -to D[4]

set\_location\_assignment PIN\_E14 -to OnNOff

set\_location\_assignment PIN\_A11 -to Busy

1. Código *Kotlin* - *HAL*

import isel.leic.UsbPort

// Virtualiza o acesso ao sistema UsbPort

object HAL {

private var lastWriting = 0

// Inicia a classe

fun init() {

UsbPort.write(lastWriting)

}

// Retorna true se o bit tiver o valor lógico ‘1’

fun isBit(mask: Int): Boolean {

val temp = mask and UsbPort.read()

return mask == temp

}

// Retorna os valores dos bits representados por mask presentes no UsbPort

fun readBits(mask: Int): Int = mask and UsbPort.read()

// Escreve nos bits representados por mask o valor de value

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

val a = mask and value

val b = mask.inv() and lastWriting

val c = a or b

UsbPort.write(c)

lastWriting = c

}

// Coloca os bits representados por mask no valor lógico ‘1’

fun setBits(mask: Int) {

writeBits(mask,0xFF)

}

// Coloca os bits representados por mask no valor lógico ‘0’

fun clearBits(mask:Int) {

writeBits(mask,0x00)

}

}

fun main() {

val mask = 0b00001111

HAL.init()

HAL.setBits(mask)

Thread.sleep(2000)

HAL.clearBits(mask)

Thread.sleep(2000)

println(HAL.isBit(mask))

Thread.sleep(2000)

HAL.writeBits(mask, 6)

Thread.sleep(2000)

// Change the value of the input port bits

val currentBits = HAL.readBits(mask)

println(currentBits)

}

1. Código *Kotlin* – *Serial Emitter*

// Envia tramas para os diferentes módulos Serial Receiver.

object SerialEmitter {

private const val clk: Long = 50

enum class Destination {

LCD,

DOOR

}

// Inicia a classe

fun init() {

HAL.init()

HAL.clearBits(nSDCsel\_MASK)

HAL.clearBits(nLCDsel\_MASK)

HAL.clearBits(SCLK\_MASK)

HAL.clearBits(SDX\_MASK)

}

// Envia uma trama para o SerialReceiver identificado o destino em addr e os bits de dados em ‘data’.

fun send(addr: Destination, data: Int) {

Thread.sleep(clk)

var nSSMask = nLCDsel\_MASK

if (addr == Destination.DOOR) {

nSSMask = nSDCsel\_MASK

while (isBusy()) {

Thread.sleep(clk \* 10)

}

}

HAL.clearBits(nSSMask)

for (i in 4 downTo 0) {

HAL.clearBits(SCLK\_MASK)

val sdx = (data shr i) and 1

if (sdx == 1) HAL.setBits(SDX\_MASK) else HAL.clearBits(SDX\_MASK)

HAL.setBits(SCLK\_MASK)

}

HAL.clearBits(SCLK\_MASK)

HAL.setBits(nSSMask)

Thread.sleep(clk)

}

// Retorna true se o canal série estiver ocupado

fun isBusy(): Boolean = HAL.isBit(BUSY\_MASK)

}

fun main() {

SerialEmitter.init()

for (i in 0..31) {

SerialEmitter.send(SerialEmitter.Destination.LCD, i)

Thread.sleep(250)

}

}

1. Código Kotlin – *Door Mechanism*

// Controla o estado do mecanismo de abertura da porta.

object DoorMechanism {

// 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) {

while (!finished()) {

Thread.sleep(1000)

}

SerialEmitter.send(SerialEmitter.Destination.DOOR, velocity shl 1 or 1)

}

// Envia comando para fechar a porta, com o parâmetro de velocidade

fun close(velocity: Int) {

while (!finished()) {

Thread.sleep(1000)

}

SerialEmitter.send(SerialEmitter.Destination.DOOR, velocity shl 1 or 0)

}

// Verifica se o comando anterior está concluído

fun finished() : Boolean = !SerialEmitter.isBusy()

}

fun main() {

DoorMechanism.init()

println("Finished DoorMechanism.init()")

while (true) {

DoorMechanism.open(10)

Thread.sleep(1000)

DoorMechanism.close(5)

}

}