O módulo *Serial Door Controller* é constituído por dois blocos principais: *i*) *Serial Reciever*; *ii*) *Door Controller*  que em conjuntoimplementam a receção em série da informação enviada pelo módulo de controlo e entrega-a posteriormente ao mecanismo da porta.

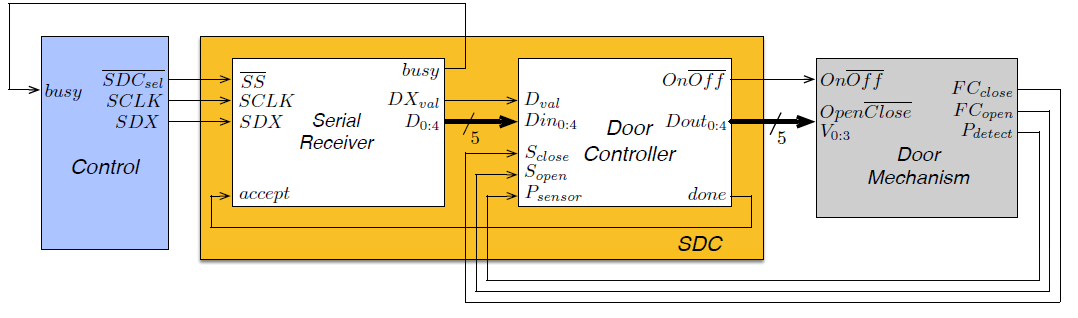


Figura 1 – Diagrama de blocos do *Serial Door Controller*

# Serial Door Controller

O bloco Serial Receiver (descrição no anexo A) é implementado com uma estrutura similar ao bloco *Serial Receiver* do *SLCDC*, sendo dividido em 3 sub-blocos: *i) Counter* (descrição no anexo B) tem como função contar bits recebidos e é contituido por 4 sub-blocos: adder descrição no anexo C) que adiciona o A, B e Ci, sendo este constituído por *full adders*(descrição no anexo D); um MUX2x1(descrição no anexo E), um sub-bloco *Registry* (descrição no anexo F) que tem por função guardar o valor que lhe é passado e é constituído por flipflops (descrição no anexo G) e um *Terminal Count* (descrição no anexo H) que diz se *Counter* chegou ao fim da contagem. *ii)* o bloco *Serial Control* (descrição no anexo I)tem por função controlar e contabilizar o tempo necessário pra processar uma trama.; e *iii)* o bloco *Shift Register* (descrição no anexo J)que é um bloco conversor série paralelo.

O bloco Door Controller (descrição no anexo K) é implementado para que ao receber uma trama válida do *Serial Reciever* proceda à atuação do comando recebido do mecanismo da porta.

Sempre que a porta estiver num processo de fechar, esta só fecha se não houver uma pessoa a passar por ela, abrindo se ainda não tiver fechado completamente.

A partir do momento que a porta fecha completamente, é necessário o utilizador introduzir os respetivos dados de entrada.

Após o utilizador introduzir os seus dados de entrada corretamente, a porta abre completamente e permanece aberta enquanto uma pessoa estiver a passar por ela, iniciando o processo de fecho quando não há ninguém a passar pela porta.

Uma imagem com diagrama, Esquema, Desenho técnico, file

Descrição gerada automaticamente

Figura 1- Diagrama de blocos do Serial Control

Uma imagem com texto, captura de ecrã, Tipo de letra, file

Descrição gerada automaticamente

Figura 2 - Protocolo de comunicação do *Serial Door Controller*

O bloco *Serial Control* foi implementado de acordo com o diagrama de blocos representado na Figura 2. Neste diagrama podemos observar que no seu estado inicial (**NOT\_BUSY**), espera que seja ativado o sinal *enRx* (not SS, proveniente do software). Quando é detetado que este foi ativado, passa para o seu segundo estado (**COUNT**) onde inicia uma contagem a partir do 0. Quando for detetado que *enRx* foi desativado e que a contagem chegou a 5, passa para o terceiro estado (**VALIDATE**), onde indica que a informação recebida pode ser validada, esperando que lhe seja indicado, pelo sinal *accept*, que a trama foi aceite e passando assim para o seu quarto e último estado (**STIL\_BUSY**), onde fica à espera de que o sinal *accept* seja desativado, indicando assim que está pronto a receber uma nova trama.

O bloco *Door Controller* foi implementado pela máquina de estados representada em *ASM-chart* na Figura 3. O diagrama, começa por um estado de espera (**ZEN**), onde aguarda a ativação do sinal *Dval*, que indica que existe uma trama válida. Caso o bit de menor peso desta trama seja 1, a máquina passa para o estado **OPEN\_DOOR**, onde é ativado o sinal *OpenClose* que indica que a porta está a abrir, ativando também o sinal *OnOff*, enquanto a porta não estiver completamente aberta (sinal *Sopen*). Caso o bit de menor peso da trama seja 0, vai para o estado **CLOSE\_DOOR** que não tem o sinal *OpenClose* ativo, que significa que a porta está a fechar, ativando também o sinal *OnOff*. No instante em que é detetada a presença de uma pessoa pela ativação do sinal *Psensor*, a máquina passa para o estado **OPEN\_DOOR**, onde permanece até a porta estar completamente aberta.

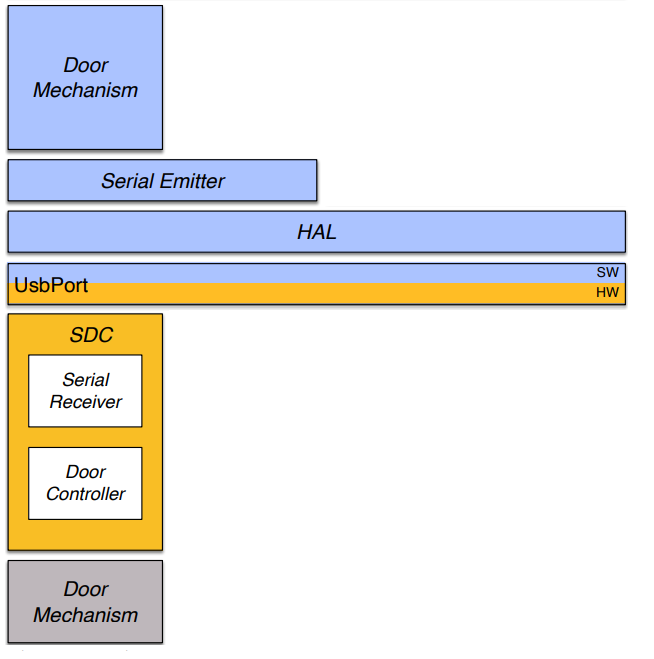
Para chegar ao estado **FINISHED**, ou a porta está aberta e o bit de menor peso da trama é 1, ou a porta está fechada. Neste estado, é ativado o sinal *done* indicando o fim do último comando (bit de menor peso da trama), permanecendo aqui até que o sinal *Dval* seja desativado, onde volta para o seu estado inicial.

Uma imagem com diagrama, Esquema, Desenho técnico, esquemático

Descrição gerada automaticamenteFigura 2 - Diagrama de blocosdo Door Controller

# Interface com o *Door Mechanism*

Implementou-se o módulo *Door Mechanism* em *software*, recorrendo a linguagem *Kotlin* e seguindo a arquitetura lógica apresentada na Figura 4.

Figura 4 – Diagrama lógico do *Seria Door Controller*

Foram utilizadas 4 funções para atingir o objetivo da interface.

## *Descrição das funções utilizadas*

### fun init()

Tem como função estabelecer os valores iniciais para iniciar a classe

### fun open(velocity:Int)

Foi criada para enviar o comando para abrir a porta à velocidade desejada.

### fun close(velocity:Int)

Tem a função oposta da função em 2.1.2., enviando um comando para fechar a porta ao invés de a abrir, à velocidade pretendida.

### fun finished()

Verifica se o comando (open ou close) está concluído.

O código destas funções está explicito no anexo L.

# Conclusões

O maior desafio deste bloco foi o DoorController, pois o SerialReceiver foi reutilizado e a componente software foi constítuida apenas de funções que chamavam outras funções criadas anteriormente.

O ASM é complexo, mas após algum estudo do seu propósito, a implementação fez sentido e após vários testes no simulador e na placa, tudo o que foi implementado funciona como visionado.

2. Descrição VHDL do bloco *Serial Reciever*

library ieee;

use ieee.std\_logic\_1164.all;

entity SDC is

port(

SS, SDX, SCLK, Reset, Sclose, Sopen, Psensor, CLK : in std\_logic;

OnOff, busy : out std\_logic;

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

end SDC;

architecture arc of SDC is

component SerialReceiver

port(

SS, SCLK, CLK, SDX, accept, Reset : in std\_logic;

DXval, busy : out std\_logic;

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

end component;

component DoorController

port(

Dval, Sclose, Sopen, Psensor, Reset, CLK : in std\_logic;

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

OnOff, done : out std\_logic;

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

end component;

signal state, dv : std\_logic;

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

begin

serialR : SerialReceiver port map(

SS => SS,

SCLK => SCLK,

CLK => CLK,

SDX => SDX,

accept => state,

Reset => Reset,

DXval => dv,

busy => busy,

D => data);

doorControl: DoorController port map(

CLK => CLK,

Dval => dv,

Sclose => Sclose,

Sopen => Sopen,

Psensor => Psensor,

Reset => Reset,

Din => data,

OnOff => OnOff,

done => state,

Dout => Dout);

end arc;

1. Descrição VHDL do sub-bloco *Counter*

library ieee;

use ieee.std\_logic\_1164.all;

-- CC -> contador crescente!

entity Counter is

port(

PL, CE, CLK, Reset: in std\_logic;

Data\_in: in std\_logic\_vector(3 downto 0);

TC: out std\_logic;

Q: out std\_logic\_vector(3 downto 0));

end Counter;

architecture arc\_cc of Counter is

component adder

Port(A, B :in std\_logic\_vector(3 downto 0);

Ci: in std\_logic;

Co: out std\_logic;

S: out std\_logic\_vector(3 downto 0));

end component;

component MUX2x1

port(A, B: in std\_logic\_vector(3 downto 0);

S: in std\_logic;

Y: out std\_logic\_vector(3 downto 0));

end component;

component Registry

port(

D:in std\_logic\_vector(3 downto 0);

CLK, E, Reset: in std\_logic;

Q: out std\_logic\_vector(3 downto 0));

end component;

component Terminal\_Count

port(

Q : in std\_logic\_vector(3 downto 0);

TC: out std\_logic);

end component;

signal outadder, outmux, outreg: std\_logic\_vector(3 downto 0);

begin

ad: adder port map(

A => outreg,

B => "0000",

Ci => CE,

S => outadder);

mux: MUX2x1 port map(

A => Data\_in,

B => outadder,

S => PL,

Y => outmux);

reg: Registry port map(

Reset => Reset,

E => '1',

D => outmux,

CLK => CLK,

Q => outreg);

Q <= outreg;

utc: Terminal\_Count port map(

Q => outreg,

TC => TC);

end arc\_cc;

1. Descrição VHDL do sub-bloco *adder*

library ieee;

use ieee.std\_logic\_1164.all;

entity adder is

Port(A, B :in std\_logic\_vector(3 downto 0);

Ci: in std\_logic;

Co: out std\_logic;

S: out std\_logic\_vector(3 downto 0));

end adder;

architecture arc\_adder of adder is

component full\_add

Port(A, B, Cin: in std\_logic;

Cout, S: out std\_logic);

end component;

signal c1, c2, c3: std\_logic;

begin

fa1: full\_add port map(

A => A(0),

B => B(0),

Cin => Ci,

S => S(0),

Cout => c1);

fa2: full\_add port map(

A => A(1),

B => B(1),

Cin => C1,

S => S(1),

Cout => c2);

fa3: full\_add port map(

A => A(2),

B => B(2),

Cin => C2,

S => S(2),

Cout => c3);

fa4: full\_add port map(

A => A(3),

B => B(3),

Cin => C3,

S => S(3),

Cout => Co);

end arc\_adder;

1. Descrição VHDL do sub-bloco full adder

library ieee;

use ieee.std\_logic\_1164.all;

entity full\_add is

Port(A, B, Cin: in std\_logic;

Cout, S: out std\_logic);

end full\_add;

architecture arc\_fa of full\_add is

begin

S <= A xor B xor Cin;

Cout <= (A and B) or (A and Cin) or (B and Cin);

end arc\_fa;

1. Descrição VHDL do sub-bloco MUX2x1

library ieee;

use ieee.std\_logic\_1164.all;

entity MUX2x1 is

port(A, B: in std\_logic\_vector(3 downto 0);

S: in std\_logic;

Y: out std\_logic\_vector(3 downto 0));

end MUX2x1;

architecture arc\_mux of MUX2x1 is

begin

Y(0)<=(S and A(0)) or (not S and B(0));

Y(1)<=(S and A(1)) or (not S and B(1));

Y(2)<=(S and A(2)) or (not S and B(2));

Y(3)<=(S and A(3)) or (not S and B(3));

end arc\_mux;

1. Descrição VHDL do sub-bloco Registry

library ieee;

use ieee.std\_logic\_1164.all;

entity Registry is

port(

D:in std\_logic\_vector(3 downto 0);

CLK, E, Reset: in std\_logic;

Q: out std\_logic\_vector(3 downto 0));

end Registry;

architecture arc\_reg of Registry is

component FFD

port( CLK : in std\_logic;

RESET : in std\_logic;

SET : in std\_logic;

D : in std\_logic;

EN : in std\_logic;

Q : out std\_logic

);

end component;

begin

ff0: FFD port map(

SET => '0',

RESET => Reset,

CLK => CLK,

D => D(0),

EN => E,

Q => Q(0));

ff1: FFD port map(

SET => '0',

RESET => Reset,

CLK => CLK,

D => D(1),

EN => E,

Q => Q(1));

ff2: FFD port map(

SET => '0',

RESET => Reset,

CLK => CLK,

D => D(2),

EN => E,

Q => Q(2));

ff3: FFD port map(

SET => '0',

RESET => Reset,

CLK => CLK,

D => D(3),

EN => E,

Q => Q(3));

end arc\_reg;

1. Descrição VHDL do sub-bloco *FFD*

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

ENTITY FFD IS

PORT( CLK : in std\_logic;

RESET : in STD\_LOGIC;

SET : in std\_logic;

D : IN STD\_LOGIC;

EN : IN STD\_LOGIC;

Q : out std\_logic

);

END FFD;

ARCHITECTURE LogicFunction OF FFD IS

BEGIN

Q <= '0' when RESET = '1' else '1' when SET = '1' else D WHEN rising\_edge(clk) and EN = '1';

END LogicFunction;

1. Descrição VHDL do sub-bloco *Terminal Count*

library ieee;

use ieee.std\_logic\_1164.all;

entity Terminal\_Count is

port(

Q : in std\_logic\_vector(3 downto 0);

TC: out std\_logic);

end Terminal\_Count;

architecture arc\_tc of Terminal\_Count is

begin

TC <= Q(0) and Q(1) and Q(2) and Q(3);

end arc\_tc;

1. Descrição VHDL do sub-bloco *Serial Control*

library ieee;

use ieee.std\_logic\_1164.all;

entity SerialControl is

port(

Reset, enRx, accept, eq5, CLK : in std\_logic;

clr, wr, DXval, cenable, busy : out std\_logic);

end SerialControl;

architecture arc\_sc of SerialControl is

type STATE\_TYPE is (FIRST, SECOND, THIRD, FORTH);

signal CurrentState, NextState: STATE\_TYPE;

begin

CurrentState <= FIRST when Reset = '1' else NextState when rising\_edge(CLK);

GenerateNextState:

process(CurrentState, enRx, eq5, accept)

begin

case CurrentState is

when FIRST =>

if (enRx = '0') then NextState <= SECOND;

else NextState <= FIRST;

end if;

when SECOND =>

if (enRx = '1') then

if(eq5 = '1') then NextState <= THIRD;

else NextState <= FIRST;

end if;

else NextState <= SECOND;

end if;

when THIRD =>

if (accept = '1') then NextState <= FORTH;

else NextState <= THIRD;

end if;

when FORTH =>

if(accept = '0') then NextState <= FIRST;

else NextState <= FORTH;

end if;

end case;

end process;

clr <= '1' when (CurrentState = FIRST) else '0';

cenable <= '1' when (CurrentState = SECOND) else '0';

wr <= '1' when (CurrentState = SECOND) else '0';

DXval <= '1' when (CurrentState = THIRD) else '0';

busy <= '0' when (CurrentState = FIRST) else '1';

end arc\_sc;

1. Descrição VHDL do sub-bloco *Shift Register*

library ieee;

use ieee.std\_logic\_1164.all;

entity ShiftRegister is

port(

data, clk, enable, reset : in std\_logic;

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

end ShiftRegister;

architecture arc\_sr of ShiftRegister is

component FFD

PORT( CLK : in std\_logic;

RESET : in STD\_LOGIC;

SET : in std\_logic;

D : IN STD\_LOGIC;

EN : IN STD\_LOGIC;

Q : out std\_logic

);

end component;

signal f1, f2, f3, f4: std\_logic;

begin

ffd4: FFD port map(

SET => '0',

RESET => reset,

CLK => clk,

D => data,

EN => enable,

Q => f4);

ffd3: FFD port map(

SET => '0',

RESET => reset,

CLK => clk,

D => f4,

EN => enable,

Q => f3);

ffd2: FFD port map(

SET => '0',

RESET => reset,

CLK => clk,

D => f3,

EN => enable,

Q => f2);

ffd1: FFD port map(

SET => '0',

RESET => reset,

CLK => clk,

D => f2,

EN => enable,

Q => f1);

ffd0: FFD port map(

SET => '0',

RESET => reset,

CLK => clk,

D => f1,

EN => enable,

Q => D(0));

D(1) <= f1;

D(2) <= f2;

D(3) <= f3;

D(4) <= f4;

end arc\_sr;

1. Descrição VHDL do bloco *Door Controller*

library ieee;

use ieee.std\_logic\_1164.all;

entity DoorController is

port(

Dval, Sclose, Sopen, Psensor, Reset, CLK : in std\_logic;

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

OnOff, done, OpenClose : out std\_logic;

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

end DoorController;

architecture arc of DoorController is

type STATE\_TYPE is (ZEN, OPEN\_DOOR, CLOSE\_DOOR, FINISHED);

signal CurrentState, NextState: STATE\_TYPE;

begin

CurrentState <= ZEN when Reset = '1' else NextState when rising\_edge(CLK);

GenerateNextState:

process(CurrentState, Din(0), Dval, Sopen, Sclose, Psensor)

begin

case CurrentState is

when ZEN =>

if (Dval = '1' and Din(0) = '1') then NextState <= OPEN\_DOOR;

elsif (Dval = '1' and Din(0) = '0') then NextState <= CLOSE\_DOOR;

else NextState <= ZEN;

end if;

when OPEN\_DOOR =>

if (Sopen = '1' and Din(0) = '0') then NextState <= CLOSE\_DOOR;

elsif (Sopen = '1' and Din(0) = '1') then NextState <= FINISHED;

else NextState <= OPEN\_DOOR;

end if;

when CLOSE\_DOOR =>

if (Psensor = '0' and Sclose = '1') then NextState <= FINISHED;

elsif (Psensor = '1') then NextState <= OPEN\_DOOR;

else NextState <= CLOSE\_DOOR;

end if;

when FINISHED =>

if (Dval = '0') then NextState <= ZEN;

else NextState <= FINISHED;

end if;

end case;

end process;

OnOff <= '1' when ((CurrentState = OPEN\_DOOR and Sopen = '0')

or (CurrentState = CLOSE\_DOOR and Psensor = '0' and Sclose = '0'))

else '0';

OpenClose <= '1'when (CurrentState = OPEN\_DOOR) else '0';

done <= '1' when (CurrentState = FINISHED) else '0';

Dout(4 downto 1) <= Din(4 downto 1);

Dout(0) <= '1' when (CurrentState = OPEN\_DOOR) else '0';

end arc;

1. Código *Kotlin* – *Door Mechanism*

**object DoorMechanism { // Controla o estado do mecanismo de abertura da porta.**

**private const val OPEN = 1**

**// 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()**

**}**