Autores: Diego Passos / Manuel Carvalho / Nuno Sebastião / Pedro Miguens Matutino
LEIC 24D, G5 – André Monteiro nº43842, Rúben Said nº47526, Umera Aktar nº50562

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

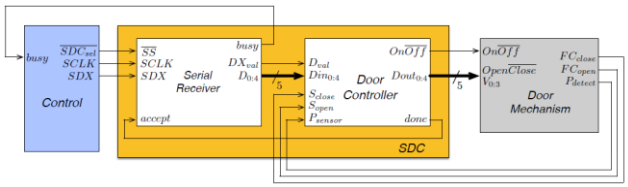

Figura 1 – Diagrama de blocos do *Serial Door Controller*

1 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.

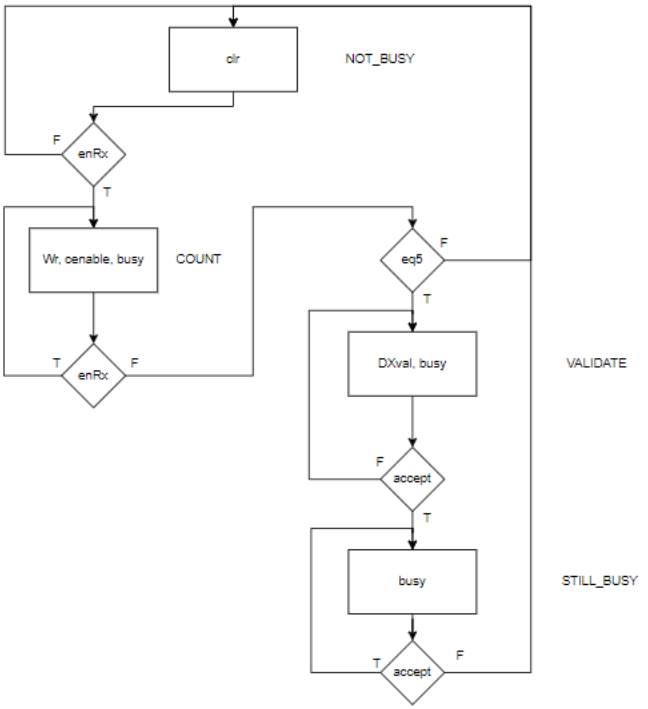

Figura 2- Diagrama de blocos do Serial Control

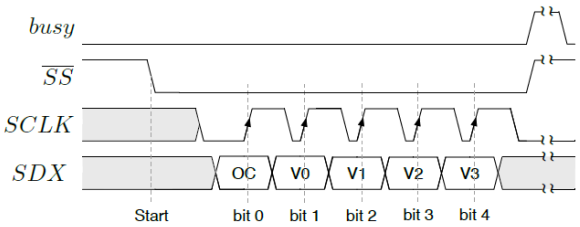

Figura 3 - Protocolo de comunicação do *Serial Door Controller*

O bloco *Serial Control* foi implementado de acordo com o diagrama de blocos representado na Figura . 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 4. 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.

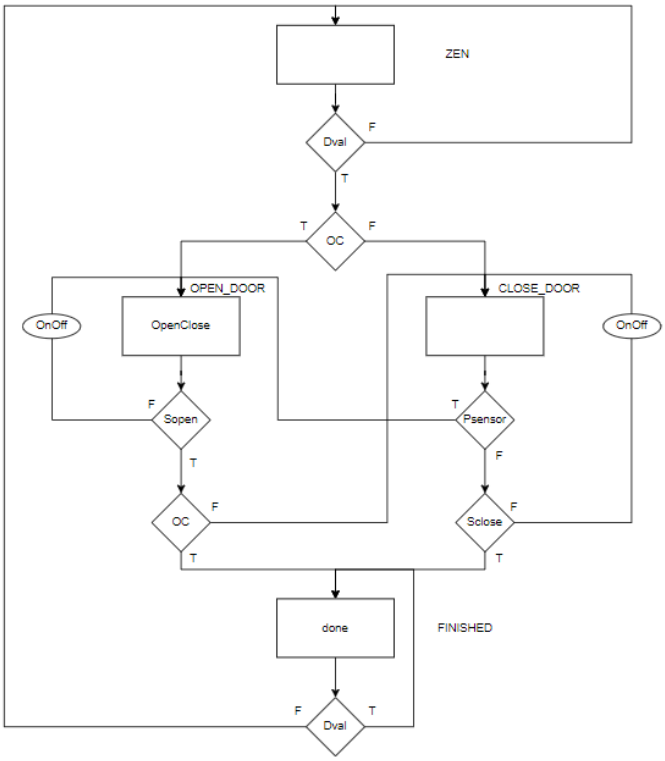

Figura 4 - Diagrama de blocos do Door Controller

2 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 5.

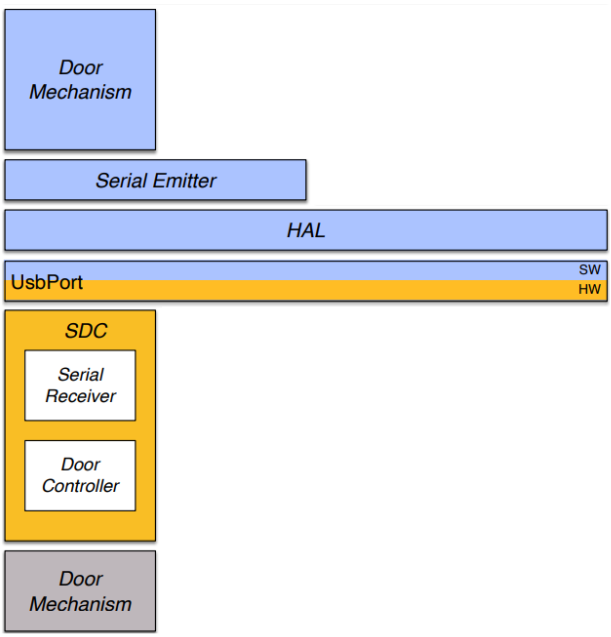

Figura 5 – Diagrama lógico do *Serial Door Controller*

Foram utilizadas 4 funções para atingir o objetivo da interface.

2.1 *Descrição das funções utilizadas*

2.1.1 fun init()

Tem como função estabelecer os valores iniciais para iniciar a classe

2.1.2 fun open(velocity:Int)

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

2.1.3 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.

2.1.4 fun finished()

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

O código destas funções está explicito no anexo L.

3 Conclusões

O maior desafio deste bloco foi o DoorController, pois o SerialReceiver foi reutilizado e a componente software foi constituida 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.

A. 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;
```

B. 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;
```

C. 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;
```

D. 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;
```

E. 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;
```

F. 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;
```

G. 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;
```

H. 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;
```

I. 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;
```

J. 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;
```

K. 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;
```

L. Código *Kotlin – Door Mechanism*

```
@file:Suppress("ControlFlowWithEmptyBody")

object DoorMechanism { // Controla o estado do mecanismo de abertura da porta.
    private const val OPEN = 0x01

    // 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()
}
```