O módulo *Serial Door Controller* é constituído por dois blocos principais: *i*) bloco que recebe a informação em série enviada pelo módulo de controlo (Serial Receiver); *ii*) bloco que procede à atuação do comando recebido no mecanismo da porta, (designado por *Door Controller*). Neste caso o módulo *Control*, implementado em *software*, é a entidade consumidora.

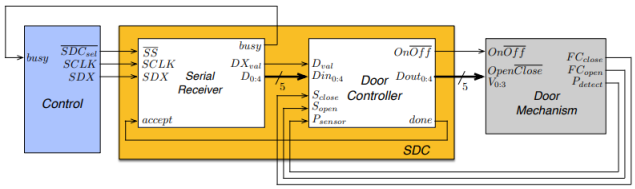
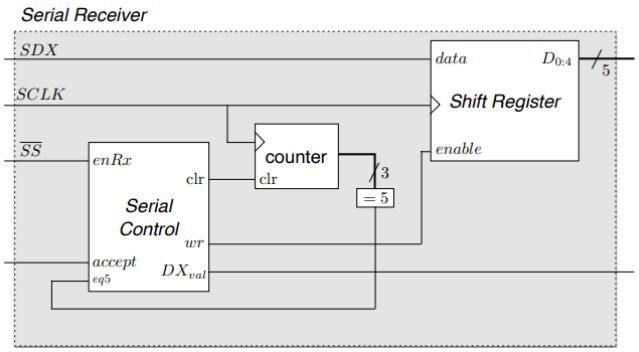


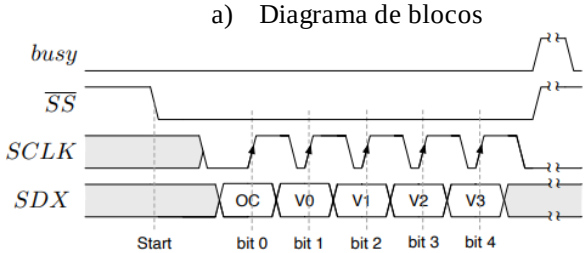
Figura 1 – Diagrama de blocos do módulo *Serial Door Controller*

1 Serial Receiver

O bloco Serial Receiver implementa um descodificador de um teclado matricial 4x3 através de hardware, sendo constituído por três sub-blocos: i) bloco responsável por deslocar um bit para a esquerda (Shift Register); ii) o bloco Counter, responsável por, a cada impulso clock, somar 1 ao valor anteriormente guardado; e iii) o bloco Control, que realiza o controlo do varrimento e o controlo de fluxo, conforme o diagrama de blocos representado na Figura 2A. O controlo de fluxo de saída do bloco Serial Receiver (para o módulo LCD Dispatcher), define que o sinal DXval é ativado quando enRx está ativo e se todos os bits (5 bits) foram recebidos (a partir do eq5 estar com o valor 1), sendo também disponibilizado o código dessa tecla no barramento K 0:4. Apenas é iniciado um novo ciclo de varrimento do teclado quando o sinal accept for ativado e posteriormente desativado. O diagrama temporal do controlo de fluxo está representado na Figura 2b.



a) Diagrama de blocos



b) Diagrama temporal

Figura 2 - Comunicação com Serial Door Controller

2 Serial Control

O bloco Serial Control, tal como o nome indica, controla a leitura da trama recebida pelo Serial Receiver. O bloco aguarda uma transição descendente do sinal not SS que está dependente da comunicação com o Serial LCD Controller e quando essa transição acontece, o bloco Serial Control, através de ligações com os blocos counter e Shift Register, lê bit a bit a trama recebida. No final, caso todos tenham sido lidos com sucesso, o bloco "avisa" o bloco LCD Dispatcher que foi transmitida uma trama válida. Este processo foi explicado e implementado com base na seguinte máquina de estados:

A máquina de estados do bloco Serial Control tem como primeiro estado o estado Clear.

O sinal de saída "clear" está ativado para termos a certeza que o "counter" está a zero.

Neste estado aguarda-se que o sinal enRx passe para o valor lógico "false" para haver uma passagem para o estado seguinte, caso contrário a máquina de estados mantém-se neste estado (após esta transição do sinal enRx para "false" é que o SLCDC armazena os bits da trama nas transições ascendentes do sinal SCLK).

No segundo estado, While, sempre que a confirmação do sinal enRx seja "false" o sinal de saída wr ativa (o que depois, tendo em conta o resto do Serial Receiver, vai "permitir ir trocando de bit que queremos ler") e mantém-se o segundo estado.

Caso o enRx esteja "true", verifica-se o valor lógico do sinal eq5. Tendo o enRx "true" e o eq5 "false" a máquina de estados volta ao primeiro estado pois significa que houve um erro que não permitiu ler os 5 bits e como tal, o processo tem de ser reiniciado.

Se enRx e eq5 estiverem "true", significa que a leitura de todos os bits da trama foi bem sucedida e podemos assim passar ao terceiro estado.

No nosso terceiro estado , Value , a saída DXval encontra-se a "true" pois todos os bits foram lidos com sucesso (sendo assim uma trama válida) e aguarda-se neste estado que o sinal accept esteja a "true" para confirmar que a trama foi processada pelo bloco Dispatcher e para a máquina de estados poder avançar para o quarto e último estado.

No último estado, Wait, a máquina de estados espera que o sinal de entrada encontre-se realmente a "false" e, caso isso aconteça, passa assim para o estado inicial. Optou-se pela criação deste estado para ter completa certeza que a máquina de estados encontrava-se realmente preparada para receber nova trama.

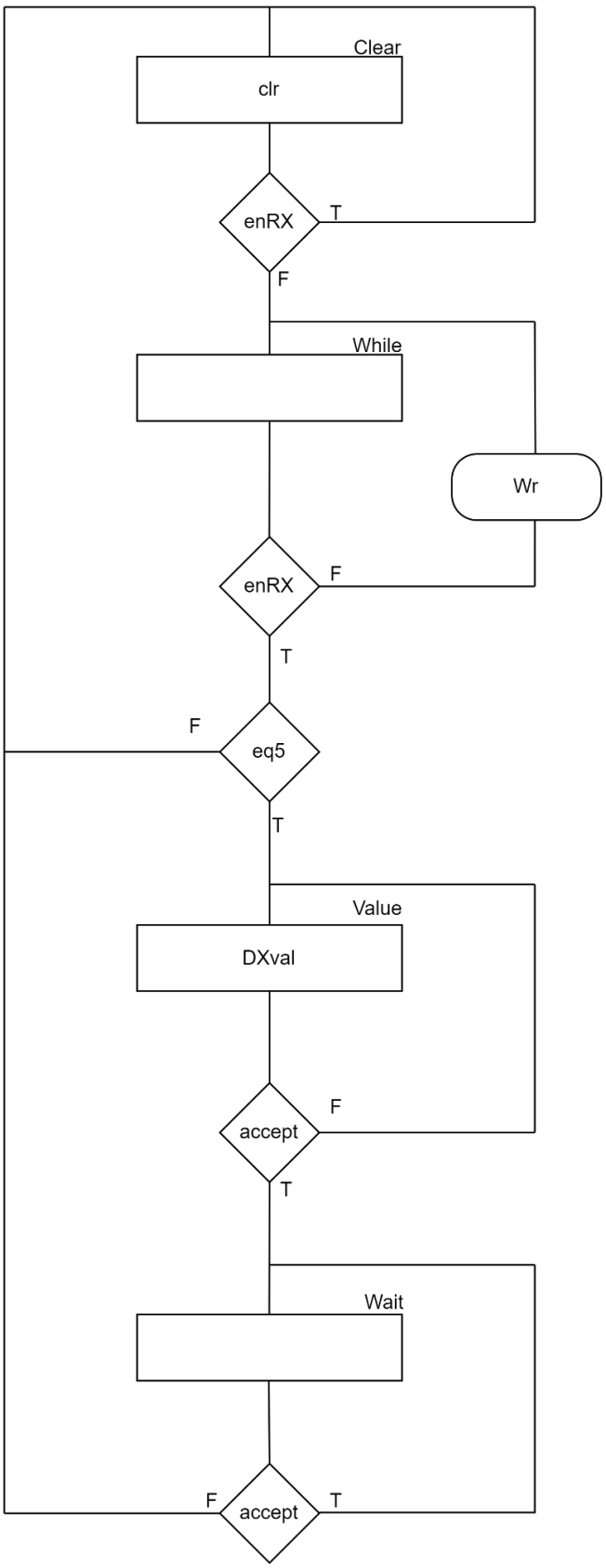


Figura 3 - Máquina de estados Serial Control

3 Door Controller

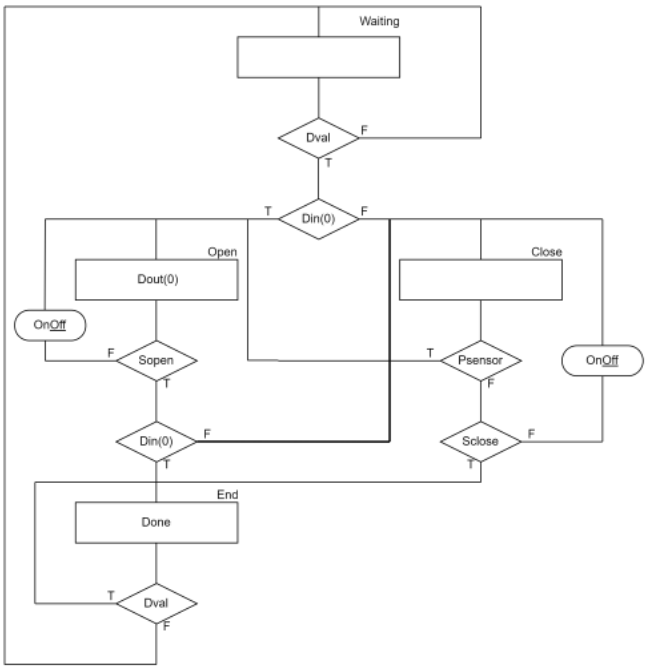


Figura 4 - Máquina de estados do módulo Door Controller

O bloco *Door Controller* quando recebe uma trama válida recebida pelo *Serial Receiver*, procede à atuação do comando recebido no mecanismo da porta.
O processo de como este comando recebido afeta o mecanismo da porta é explicado abaixo.
No primeiro estado, *Waiting*, o *Door Controller* aguarda uma trama válida, quando esta "chega" o sinal *Dval* passa para valor lógico 1 e dependendo do bit de menor peso do *Din*, a máquina de estados vai para o estado *Open* ou estado *Close*. Se o bit de menor peso da trama for 1 a porta passa para o processo de abertura, se for 0 a porta passa para o processo de fecho.
No caso de o bit de menor peso de *Din* for 1, passamos para o estado *Open*. Neste estado, as saídas *OnOff* e o bit de menor peso da saída *Dout* ficam ativas para sinalizar que a *Door* está *On* e que está em processo de abertura respetivamente. Quando a porta estiver totalmente aberta o sinal *Sopen* ativa e passamos assim para o último estado (*End*) onde o sinal de saída *Done* fica ativo para o *Door Controller* sinalizar o *Serial Receiver* que está pronto para processar uma nova trama.
No caso de o bit de menor peso de *Din* ser 0, passamos para o estado *Close*. Neste estado, a saída *OnOff* fica ativa para sinalizar que a *Door* está *On* e o bit de menor peso da saída *Dout* fica a 0 para sinalizar que a *Door* está em processo de fecho. No caso do processo de fecho é necessário que o sinal de entrada *Psensor* esteja inativo para verificar que não há pessoas a passar pela *Door* no processo de fecho da mesma.
Se durante o processo de fecho da *Door* o sinal *Psensor* estiver inativo o processo de fecho da *Door* continua até que o sinal *Sclose* passe para valor lógico 1, o que sinaliza a conclusão do processo de fecho.
Caso apareça uma pessoa na porta durante o processo de fecho, ou seja, se o sinal *Psensor* ativar, o processo de fecho da *Door* tem de ser interrompido e passar para o estado *Open* para a *Door* entrar no processo de abertura. Quando a *Door* estiver

totalmente aberta e caso o sinal *Psensor* já tenha sido inativado (já não há pessoas a passar pela *Door*), a *Door* entra imediatamente no processo de fecho novamente. Quando o processo de fecho terminar, *Sclose* ativa e passamos para o estado *End*, que tem a saída *Done* ativa para o *Door Controller* sinalizar o *Serial Receiver* que está pronto para processar uma nova trama.

4 Conclusão

Concluindo, o módulo Serial Door Controller implementa a receção em série da informação enviada pelo módulo de controlo, entregando-a posteriormente ao mecanismo da porta.

A. Descrição VHDL do bloco *Serial Receiver*

```
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.ALL;

entity serial_receiver is port (
      SDX : in std_logic;
      SCLK : in std_logic;
      MCLK : in std_logic;
      reset : in std_logic;
      not_SS : in std_logic;
      accept : in std_logic;
      D : out std_logic_vector (4 downto 0);
      DXval : out std_logic;
      busy : out std_logic
);

end serial_receiver;

architecture arq of serial_receiver is

component shift_register is port (
      data: in std_logic;
      clk : in std_logic;
      enable : in std_logic;
      D: out std_logic_vector (4 downto 0);
      reset : in std_logic

);
end component;



component counter is port (
      reset : in std_logic;
      ce : in std_logic;
      clk : in std_logic;
      Q : out std_logic_vector (3 downto 0)

);
 end component;


 component serial_control is port (
      enRx : in std_logic;
      accept : in std_logic;
      eq5 : in std_logic;
      clr : out std_logic;
      wr : out std_logic;
      DXval : out std_logic;
      reset : in std_logic;
      clk : in std_logic;
      busy : out std_logic
   );
 end component;

 component 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 component;


 signal s_wr, s_clr, c5_s, enRx, SCLK_s  : std_logic;
 signal d_s: std_logic_vector (3 downto 0);
```

```
begin
u_counter : counter port map (
      clk => SCLK,
      ce => '1',
      reset => s_clr,
      Q(3) => d_s(3),
      Q(2) => d_s(2),
      Q(1) => d_s(1),
      Q(0) => d_s(0)
);

u_shift_register: shift_register port map(
      data => SDX,
      clk => SCLK,
      enable => s_wr,
      D => D,
      reset => reset
);


u_serial_control: serial_control port map (
      enRx => not_SS,
      accept => accept,
      clr => s_clr,
      wr => s_wr,
      eq5 => c5_s,
      DXval => DXval,
      clk => MCLK,
      reset => reset,
      busy => busy
);


c5_s <= (d_s(0) and not d_s(1) and d_s(2));

end arq;
```

B. Descrição VHDL do bloco *SDC*

```
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.ALL;

entity SDC is port (
      SDX : in std_logic;
      SCLK : in std_logic;
      MCLK : in std_logic;
      reset : in std_logic;
      not_SS : in std_logic;
      onoff : out std_logic;
      V : out std_logic_vector (3 downto 0);
      open_close : out std_logic;
      sopen:in std_logic;
      sclose : in std_logic;
      psensor: in std_logic;
      busy: out std_logic
);

end SDC;

architecture arq of SDC is

component clkdiv is generic (div : natural := 25);
   port (
       clk_in : in std_logic;
       clk_out : out std_logic
);
end component;



component serial_receiver is port (
        SDX : in std_logic;
        SCLK : in std_logic;
        MCLK : in std_logic;
        reset : in std_logic;
        not_SS : in std_logic;
        accept : in std_logic;
        D : out std_logic_vector (4 downto 0);
        DXval : out std_logic;
        busy : out std_logic
);
end component;

component door_controller is port (
      dval : in std_logic;
      din: in std_logic_vector (4 downto 0);
      sopen:in std_logic;
      sclose : in std_logic;
      psensor: in std_logic;
      onoff: out std_logic;
      dout : out std_logic_vector (4 downto 0);
      done : out std_logic;
      reset : in std_logic;
      clk : in std_logic
);
end component;



signal done_s, dval_s, on_off_s : std_logic;
signal d_s : std_logic_vector (3 downto 0);
signal din_s : std_logic_vector (4 downto 0);
signal busy_s : std_logic;
signal clk : std_logic;
```

```
begin
u_serial_receiver: serial_receiver port map(
        SDX => SDX,
        SCLK => SCLK,
        MCLK => MCLK,
        reset => reset,
        not_SS => not_SS,
        accept => done_s,
        D => din_s,
        DXval => dval_s,
        busy => busy
);

u_door_controller: door_controller port map (
      dval => dval_s,
      din => din_s,
      dout(0) => open_close,
      dout(4) => V(3),
      dout(3) => V(2),
      dout(2) => V(1),
      dout(1) => V(0),
      done => done_s,
      reset => reset,
      sopen => sopen,
      sclose => sclose,
      psensor =>  psensor,
      clk => clk,
      onoff => onoff
);

u_clkdiv: clkdiv generic map (1000) port map (
   clk_in => MCLK,
   clk_out => clk
);


end arq;
```

C. Atribuição de pinos do SDC

```
set_location_assignment PIN_P11 -to Mclk


set_location_assignment PIN_W5 -to Lines[0]
set_location_assignment PIN_AA14 -to Lines[1]
set_location_assignment PIN_W12 -to Lines[2]
set_location_assignment PIN_AB12 -to Lines[3]
set_location_assignment PIN_AB11 -to Columns[0]
set_location_assignment PIN_AB10 -to Columns[1]
set_location_assignment PIN_AA9 -to Columns[2]


set_location_assignment PIN_A8 -to V[0]
set_location_assignment PIN_A9 -to V[1]
set_location_assignment PIN_A10 -to V[2]
set_location_assignment PIN_B10 -to V[3]
set_location_assignment PIN_D13 -to openclose
set_location_assignment PIN_C13 -to onoff



set_location_assignment PIN_C10 -to reset
set_location_assignment PIN_C11 -to sopen
set_location_assignment PIN_D12 -to sclose
set_location_assignment PIN_C12 -to psensor
```

D. Código *Kotlin - Door Mechanism*

```kotlin
object DoorMechanism { // Controla o estado do mecanismo de abertura da porta.
    // 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){
        val speed = velocity.shl(1) or (1)
        SerialEmitter.send(SerialEmitter.Destination.DOOR,speed)
    }

    // Envia comando para fechar a porta, com o parâmetro de velocidade
    fun close(velocity: Int){
        val speed=velocity.shl(1)
        SerialEmitter.send(SerialEmitter.Destination.DOOR,speed)
    }

    // Verifica se o comando anterior está concluído
    fun finished() : Boolean = !SerialEmitter.isBusy()


}

fun main(){
    DoorMechanism.init()
    DoorMechanism.open(15)
    //DoorMechanism.close(0x02)
    while (!DoorMechanism.finished()){
        //DoorMechanism.open(0x02)
        Thread.sleep(100)
        println("closing")
    }

}
```