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.

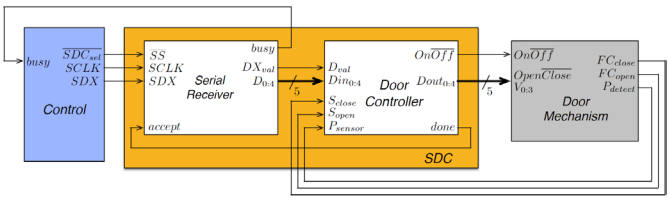

Figura 1 – Diagrama de blocos do módulo *Serial Door Controller*

1 Serial Receiver

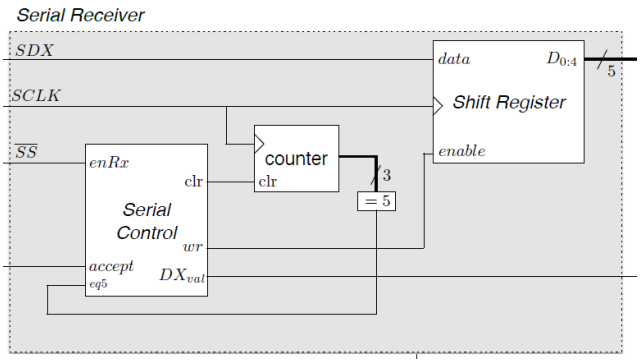

Figura 2 – Diagrama de Blocos do bloco *Serial Receiver*

O bloco Serial Receiver é utilizado tanto no módulo Serial LCD Controller como no módulo Serial Door Controller, desta forma foi possível reutilizar o bloco Serial Receiver nos dois módulos.

O SDCrecebe em série uma mensagem constituída por cinco bits de informação. A comunicação com o SDC realiza-se, tendo como primeiro bit de informação, o bit $OpenClose$ (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".

O bloco *Serial Control* foi implementado pela máquina de estados representada em *ASM-chart* na Figura 3.

Inicialmente o Serial Control encontra-se estado 00 o sinal Clr está ativo de forma que o counter permaneça inativo até que o valor lógico do sinal SS seja 0 indicando que o envio de dados irá começar. Enquanto permanece no estado 01 o sinal WR fica ativo e permanece ativo até que sejam contados 5 bits na transferência de dados. Quando o número de bits recebidos atinge o valor 5 ocorre uma passagem para o estado 10, e permanece neste até que o sinal SS esteja com o valor lógico 1 indicando que o envio de dados terminou. Quando o envio de dados termina, ocorre uma mudança para o estado 11 onde aguarda até que o valor lógico do sinal accept seja 1, indicando a trama foi processada e *accept* seja 1, indicando a trama foi processada, ativando os sinais *DXval* e *busy*.

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

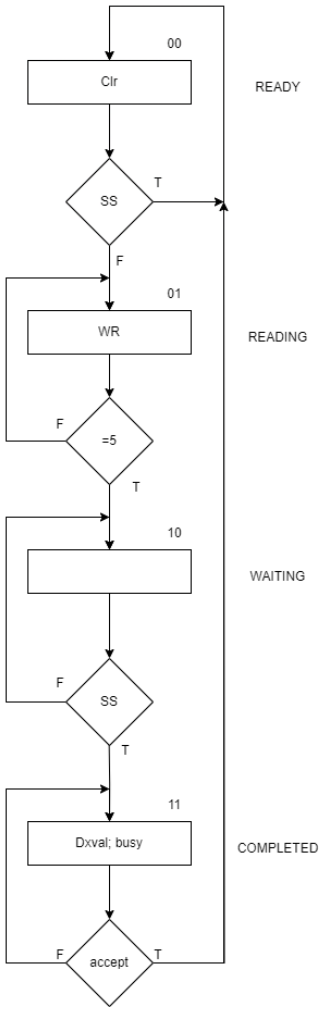

Figura 3 – Máquina de estados do bloco *Serial Control*

3 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 $Onff$ e o sinal $OpenClose$ no 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 $On\overline{ff}$ e colocar o sinal $Open\overline{Close}$ no valor lógico '0', até o sensor de porta fechada (*FCclose*) ficar ativo. Se durante o fecho for detetada uma pessoa na zonada 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 4.

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

Inicialmente o bloco Door Controller encontra-se no estado 000, a aguardar que o nível lógico do sinal Dval seja "1" indicando uma trama válida. Quando recebe a indicação de uma trama válida, verifica o sinal OC de forma a apurar se o comando é para abrir ou fechar a porta. Se o comando é de é para abrir a porta, ocorre uma mudança para o estado 001, onde permanece com os sinais $On\overline{ff}$ (ativando o mecanismo da porta) e $Open\overline{Close}$ ativos até que esta esteja completamente aberta , aguardando que o sinal Sopen tenha valor lógico "1". De seguida verifica se a instrução é de fecho de forma a concluir a mesma, se a instrução é de abrir a porta, ocorre a passagem para o estado 100 em que o sinal done é ativado onde aguarda que o sinal Dval tenha valor lógico "0". Se pelo contrário a instrução é de fechar a porta, a máquina de estados passa do estado 000 para o estado 010 onde verifica através do sinal *Psensor* se foi detetada alguma pessoa na zona da porta. Se não for detetada uma pessoa na zona da porta, passa para o estado 011 onde ativa o sinal $On\overline{ff}$, de forma a ativar o mecanismo da porta e com o sinal $Open\overline{Close}$ com valor lógico a "0" indicando que é para fechar a porta, e permanece neste estado até que a porta esteja fechada, ativando o sinal Scloes. Se for detetada uma pessoa durante o fecho da porta ocorre uma passagem para o estado 001 de forma a abrir a porta. Depois de fechar a porta ocorre a passagem para o estado 100 ativando o sinal done informando que terminou de executar a instrução e aguarda que o sinal Dval tenha valor lógico "0"

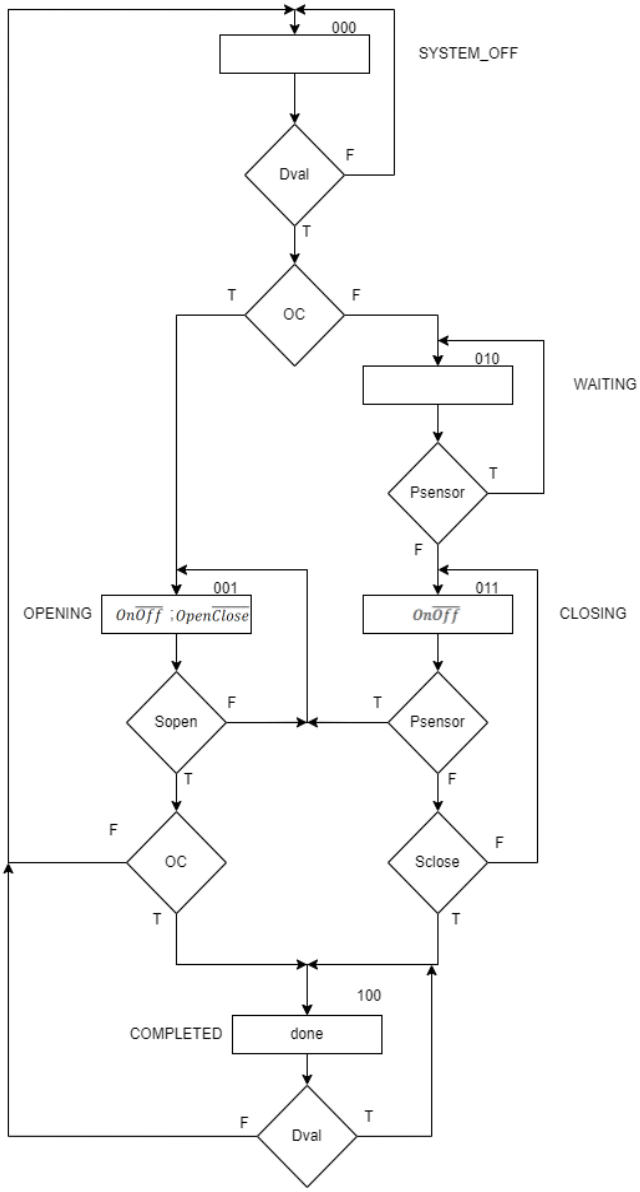

Figura 4 – Máquina de estados do bloco *Door Controller*

Com base nas descrições dos blocos *Serial Receiver* e *Door Controller* implementou-se o módulo *Serial Door Controller* de acordo com o esquema elétrico representado no Anexo C.

4 Interface com o *Control*

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

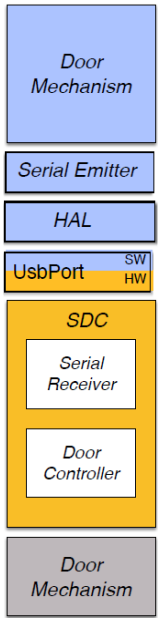

Figura 5 – Diagrama lógico do módulo *Control* de interface com o módulo *Serial Door Controller*

HAL, *Serial Emitter e LCD* desenvolvidos são descritos nas secções 4.1, 4.2 e 4.3, e o código fonte desenvolvido nos Anexos C e D, respetivamente.

4.1 *HAL*

A classe HAL é responsável por comunicar com o bloco UsbPort, fazendo a leitura e a escrita no bloco UsbPort.
De forma a evitar efetuar a leitura do bloco UsbPort repetidamente foi criada uma variável global (written) dentro da classe para guardar o último valor escrito.
Nesta classe temos a função init que inicia a classe, a função isBit() que retorna true se o bit passado na máscara tiver o valor lógico 1. A função readBits() efetua uma leitura tal como a função isBit() mas para um conjunto de bits.
A função writeBits() escreve um valor num conjunto de bits, a função setBits() escreve nos bits da máscara o valor lógico 1. E a função clrBits() coloca o valor lógico 0 no bit indicado na máscara.

4.2 Serial Emitter

A classe Serial Emitter é responsável pelo envio de tramas para os diferentes módulos Serial Receiver.
De forma a distinguir os diferentes sinais necessários para a execução do módulo SLCDC, estes foram mapeados com o objetivo de serem obtidos e identificados no outputPort do módulo UsbPort.
A função send() implementada nesta fase é responsável por criar a situação de envio de uma trama para os módulos SLCDC e SDC.

4.3 Door Mechanism

A classe Door Mechanism é responsável por controlar o estado do mecanismo de fecho e de abertura da porta.
A função open() envia um comando para abrir a porta, com a velocidade passada com parâmetro.
A função close() envia um comando para fechar a porta, com a velocidade passada com parâmetro.
A função finished() verifica se o comando anterior está concluído.

5 Conclusões

O módulo SDC tem como objetivo entregar a informação que lhe chega por parte do Control em forma de uma trama de 5 bits ao Door Mechanism. Para isso, implementámos o Serial Receiver, responsável pela receção dos dados enviados pelo microcontrolador, e o bloco Door Controller que tem a função de entregar os dados recebidos ao Door Mechanism após a trama ser validada. O módulo Serial Receiver foi testado no simulador e posteriormente na placa, apresentando um funcionamento correto. O módulo Door Controller foi testado no simulador. O módulo SDC_USBPORT foi testado na placa onde apresentou um funcionamento correto.

A. Descrição VHDL do bloco *Serial Receiver*

Serial Control

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3
4   entity SerialControl is
5   port( SS,clk,accept,eq5,reset : IN STD_LOGIC;
6         clr, wr, DXval, busy : OUT STD_LOGIC
7   );
8   end SerialControl;
9
10  architecture arq_SerialControl of SerialControl is
11
12  type STATE_TYPE is (READY, READING, WAITING,COMPLETED);
13
14  signal CURRENT_STATE, NEXT_STATE : STATE_TYPE;
15
16  begin
17
18  --    Registo Current State
19
20  CURRENT_STATE <= READY when (reset = '1') else NEXT_STATE when rising_edge(clk);
21
22  --    Máquina de Estados
23
24  GenerateNextState:
25  process (CURRENT_STATE, SS, eq5, accept)
26     begin
27        case CURRENT_STATE is
28           when READY           => if(SS = '0') then
29                                      NEXT_STATE     <= READING;
30                                   else
31                                      NEXT_STATE     <= READY;
32                                   end if;
33
34           when READING         => if(eq5 = '0') then
35                                      NEXT_STATE     <= READING;
36                                   else
37                                      NEXT_STATE     <= WAITING;
38                                   end if;
39           when WAITING         => if(SS = '1') then
40                                      NEXT_STATE     <= COMPLETED;
41                                   else
42                                      NEXT_STATE     <= WAITING;
43                                   end if;
44
45           when COMPLETED       => if(accept <= '0') then
46                                      NEXT_STATE <= COMPLETED;
47                                   else
48                                      NEXT_STATE <= READY;
49                                   end if;
50        end case;
51
52  end process;
53
54  --    Outputs
55  clr   <= '1' when (CURRENT_STATE = READY)     else '0';
56
57  wr    <= '1' when (CURRENT_STATE = READING)   else '0';
58
59  DXval <= '1' when (CURRENT_STATE = COMPLETED) else '0';
60
61  busy <= '1' when (CURRENT_STATE = COMPLETED) else '0';
62
63
64  end arq_SerialControl;
```

Counter

```
1   LIBRARY IEEE;
2   USE IEEE.std_logic_1164.all;
3
4   ENTITY COUNTER_3 IS
5      PORT(
6         CLK : in std_logic;
7         E : in std_logic;
8         CLR: in std_logic;
9         R : out std_logic_vector (2 downto 0) :=(others => '0')
10        );
11  END COUNTER_3;
12
13
14  ARCHITECTURE arq_COUNTER OF COUNTER_3 IS
15
16
17     COMPONENT REGISTOR_RB IS
18     PORT( R : in std_logic_vector(2 downto 0);
19        CLR : in std_logic;
20        CL : in std_logic;
21        E : in std_logic;
22        TC : out std_logic;
23        F : out std_logic_vector (2 downto 0)
24        );
25     END COMPONENT;
26
27
28  COMPONENT SOMADOR3
29     PORT( A : in std_logic_vector(2 downto 0):=(others => '0');
30           B : in std_logic_vector(2 downto 0);
31           CI : in std_logic;
32           R : out std_logic_vector (2 downto 0):=(others => '0')
33           );
34  END COMPONENT;
35
36  SIGNAL  SR: std_logic_vector(2 downto 0):=(others => '0');
37  SIGNAL RS: std_logic_vector(2 downto 0):=(others => '0');
38
39
40
41  BEGIN
42
43
44  USOMADOR : SOMADOR3 port map (
45     A => RS,
46     B(0) => '1',
47     B(1) => '0',
48     B(2) => '0',
49     CI => '0',
50     R => SR
51     );
52
53
54  UREGISTOR : REGISTOR_RB port map (
55     R => SR,
56     CL => CLK,
57     E => E,
58     F => RS,
59     CLR => CLR
60     );
61
62  R <= RS;
63  END arq_COUNTER;
```

Shift Register

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3
4   entity ShiftRegister_SR is
5   port( Sin, CLK, enable, RST: in STD_LOGIC;
6         D : OUT STD_LOGIC_VECTOR(4 downto 0)
7   );
8   end ShiftRegister_SR;
9
10  architecture arq_ShiftRegister_SR of ShiftRegister_SR is
11
12  component ffd
13
14  port(CLK : in std_logic;
15       RESET : in STD_LOGIC;
16       SET : in std_logic;
17       D : IN STD_LOGIC;
18       EN : IN STD_LOGIC;
19       Q : out std_logic
20  );
21  end component;
22
23  signal DSignal: STD_LOGIC_VECTOR(4 downto 0);
24
25  begin
26
27  D(0)<=DSignal(4);
28  D(1)<=DSignal(3);
29  D(2)<=DSignal(2);
30  D(3)<=DSignal(1);
31  D(4)<=DSignal(0);
32
33  Uffd0 : ffd port map (
34            CLK => CLK,
35            RESET => RST,
36            SET => '0',
37            D => Sin,
38            EN => enable,
39            Q => DSignal(0)
40  );
41
42  Uffd1 : ffd port map (
43            CLK => CLK,
44            RESET => RST,
45            SET => '0',
46            D => DSignal(0),
47            EN => enable,
48            Q => DSignal(1)
49  );
50
51  Uffd2 : ffd port map (
52            CLK => CLK,
53            RESET => RST,
54            SET => '0',
55            D => DSignal(1),
56            EN => enable,
57            Q => DSignal(2)
58  );
59
60  Uffd3 : ffd port map (
61            CLK => CLK,
62            RESET => RST,
63            SET => '0',
64            D => DSignal(2),
65            EN => enable,
66            Q => DSignal(3)
67  );
68
69  Uffd4 : ffd port map (
70            CLK => CLK,
71            RESET => RST,
72            SET => '0',
73            D => DSignal(3),
74            EN => enable,
75            Q => DSignal(4)
```

```
76   );
77
78
79   end arq_ShiftRegister_SR;
```

Serial Receiver

```
1    library IEEE;
2    use IEEE.std_logic_1164.all;
3
4    entity SerialReceiver is
5    port(
6       SDX, SCLK, SS, accept, MCLK, reset : in std_logic;
7       DXval, busy : out std_logic;
8       data : out std_logic_vector (4 downto 0)
9    );
10
11   end SerialReceiver;
12
13   architecture arq_SerialReceiver OF SerialReceiver IS
14
15
16   component COUNTER_3
17      PORT(
18         CLK : in std_logic;
19         E: in std_logic;
20         CLR: in std_logic;
21         R : out std_logic_vector (2 downto 0) :=(others => '0')
22         );
23   end component;
24
25
26   component ShiftRegister_SR
27   port( Sin, CLK, enable, RST: in STD_LOGIC;
28         D : OUT STD_LOGIC_VECTOR(4 downto 0)
29   );
30   end component;
31
32
33   component equalTo5
34   port( D : IN STD_LOGIC_VECTOR(2 downto 0);
35         F : OUT STD_LOGIC
36   );
37   end component;
38
39
40   component SerialControl
41   port( SS,clk,accept,eq5,reset : IN STD_LOGIC;
42         clr, wr, DXval, busy : OUT STD_LOGIC
43   );
44   end component;
45
46
47
48   signal wrSignal, clrSignal, eq5Signal, Clk_signal : STD_LOGIC;
49   signal counterSignal : std_logic_vector(2 downto 0);
50
51   begin
52
53
54   uSerialControl : SerialControl port map(
55                     SS => SS,
56                     clk => MCLK,
57                     accept => accept,
58                     wr => wrSignal,
59                     clr => clrSignal,
60                     DXval =>DXval,
61                     reset => reset,
62                     eq5 => eq5Signal,
63                     busy => busy
64   );
65
```

```
66   uShiftRegister : ShiftRegister_SR port map(
67                       Sin => SDX,
68                       CLK => SCLK,
69                       enable => wrSignal,
70                       RST => reset,
71                       D => data
72   );
73
74   uCOUNTER_3 : COUNTER_3 port map(
75                CLK => SCLK,
76                E => wrSignal,
77                CLR => clrSignal,
78                R =>counterSignal
79   );
80
81   uequalTo5 : equalTo5 port map (
82                   D=>counterSignal,
83                   F=>eq5Signal
84   );
85
86   end arq_SerialReceiver;
```

B. Descrição VHDL do bloco *Door Controller*

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3
4   entity DoorController is
5   port( Dval,clk,reset, Sclose, Sopen, Psensor : IN STD_LOGIC;
6         OnOff, done, OpenClose : OUT STD_LOGIC;
7         Din : IN STD_LOGIC_VECTOR(4 downto 0);
8         Dout : OUT STD_LOGIC_VECTOR(4 downto 0)
9   );
10  end DoorController;
11
12  architecture behavioral of DoorController is
13
14
15  type STATE_TYPE is (SYSTEM_OFF,WAITING, OPENING, CLOSING, COMPLETED);
16
17
18  signal CURRENT_STATE, NEXT_STATE : STATE_TYPE;
19
20  begin
21
22  --    Registo Current State
23
24  CURRENT_STATE <= SYSTEM_OFF when (reset = '1') else NEXT_STATE when rising_edge(clk);
25
26  --    Máquina de Estados
27
28  GenerateNextState:
29  process (CURRENT_STATE, Dval, Sclose, Sopen, Psensor, Din)
30     begin
31        case CURRENT_STATE is
32           when SYSTEM_OFF         => if(Dval = '1' and Din(0) = '1') then
33                                         NEXT_STATE     <= OPENING;
34                                      elsif(Dval = '1' and Din(0) = '0') then
35                                            NEXT_STATE     <= WAITING;
36                                      else
37                                         NEXT_STATE     <= SYSTEM_OFF;
38                                      end if;

39           when WAITING            => if(Psensor = '0') then
40                                            NEXT_STATE     <= CLOSING;
41                                      else
42                                            NEXT_STATE     <= WAITING;
43                                      end if;
44           when OPENING            => if(Sopen = '0') then
45                                            NEXT_STATE     <= OPENING;
46                                      elsif(Sopen = '1' and Din(0) = '1') then
47                                         NEXT_STATE     <= COMPLETED;
48                                      elsif(Sopen = '1' and Din(0) = '0') then
49                                            NEXT_STATE     <= SYSTEM_OFF;
50                                      end if;
51
52           when CLOSING            => if(Psensor = '1') then
53                                         NEXT_STATE     <= OPENING;
54                                      elsif(Sclose = '1') then
55                                            NEXT_STATE     <= COMPLETED;
56                                      else
57                                         NEXT_STATE     <= CLOSING;
58                                      end if;
59
60
61           when COMPLETED          => if(Dval = '0') then
62                                         NEXT_STATE     <= SYSTEM_OFF;
63                                      else
64                                         NEXT_STATE     <= COMPLETED;
65                                      end if;
66
67        end case;
68
69  end process;
```

```
70
71      --    Outputs
72      OnOff <= '1' when (CURRENT_STATE = OPENING OR CURRENT_STATE = CLOSING) else '0';
73      OpenClose <= '1' when (CURRENT_STATE = OPENING) else '0';
74      done <= '1' when (CURRENT_STATE = COMPLETED) else '0';
75      Dout <= Din;
76
77      end behavioral;
```

C. Descrição VHDL do bloco *SDC*

```
1   LIBRARY IEEE;
2   USE IEEE.std_logic_1164.all;
3
4   ENTITY SDC IS
5      PORT( MCLK,reset : IN STD_LOGIC;
6            NOT_SS, SCLK, SDX, Sclose, Sopen, Psensor: IN STD_LOGIC; --software
7            OnOff, busy : OUT STD_LOGIC;
8            Dout : OUT STD_LOGIC_VECTOR(4 downto 0)
9      );
10  END SDC;
11
12
13  ARCHITECTURE arq_SDC OF SDC IS
14
15
16  component COUNTER
17     PORT(  CLK : in std_logic;
18        E : in std_logic;
19        clr: in std_logic;
20        R : out std_logic_vector (3 downto 0)
21        );
22  end component;
23
24
25
26
27  COMPONENT SerialReceiver
28     port(
29        SDX, SCLK, SS, accept, MCLK, reset : in std_logic;
30        DXval, busy : out std_logic;
31        data : out std_logic_vector (4 downto 0)
32     );
33  END COMPONENT;
34
35  COMPONENT DoorController
36     port( Dval,clk,reset, Sclose, Sopen, Psensor : IN STD_LOGIC;
37        OnOff, done, OpenClose : OUT STD_LOGIC;
38        Din : IN STD_LOGIC_VECTOR(4 downto 0);
39        Dout : OUT STD_LOGIC_VECTOR(4 downto 0)
40     );
41  END COMPONENT;
42
43  COMPONENT CLKDIV is
44     generic(div: natural := 50000000);
45     port ( clk_in: in std_logic;
46            clk_out: out std_logic);
47  END COMPONENT;
48
49  signal DXvalSignal, SCLKSignal, MCLKDivSignal, acceptDoneSignal: STD_LOGIC;
50  signal DataSignal, DoutSignal : STD_LOGIC_VECTOR (4 downto 0);
51  signal counterSignal : STD_LOGIC_VECTOR (3 downto 0);
52
53  begin
54
55
56  --uCLKDIV : CLKDIV  generic map(2)  port map(
57     --  clk_in => MCLK,
58        -- clk_out => MCLKDivSignal
59  --);
60
```

```
61   uSerialReceiver: SerialReceiver PORT MAP (
62                   SDX    => SDX,
63                   SCLK   => SCLK,
64                   SS     => NOT_SS,
65                   accept => acceptDoneSignal,
66                   MCLK   => MCLK,--MCLKDivSignal,
67                   reset  => reset,
68                   DXval  => DXvalSignal,
69                   data   => DataSignal,
70                   busy   => busy
71                   );
72
73   uDoorController: DoorController PORT MAP (
74                   Dval  => DXvalSignal,
75                   clk   => MCLK,--MCLKDivSignal,
76                   reset => reset,
77                   done  => acceptDoneSignal,
78                   Din   => DataSignal,
79                   Dout  => Dout,
80                   Sclose => Sclose,
81                   Sopen => Sopen,
82                   Psensor => Psensor
83                   );
84
85   end arq_SDC;
86
87
```

D. Descrição VHDL do bloco *SDC_USBPORT*

```
1   LIBRARY IEEE;
2   USE IEEE.std_logic_1164.all;
3
4   ENTITY SDC_USBPORT IS
5      PORT(
6            MCLK,reset, Pswitch, Sopenin, Sclosein : IN STD_LOGIC;
7            Dout : out std_logic_vector(4 downto 0);
8            onOff, OpenClose, Psensor : out std_logic;
9            HEX0        : out std_logic_vector(7 downto 0);
10           HEX1        : out std_logic_vector(7 downto 0);
11           HEX2        : out std_logic_vector(7 downto 0);
12           HEX3        : out std_logic_vector(7 downto 0);
13           HEX4        : out std_logic_vector(7 downto 0);
14           HEX5        : out std_logic_vector(7 downto 0)
15  );
16  END SDC_USBPORT;
17
18
19
20
21  ARCHITECTURE arq_SDC_USBPORT OF SDC_USBPORT IS
22
23  COMPONENT UsbPort
24           PORT
25              (
26                 inputPort:  IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
27                 outputPort :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0)
28              );
29  END COMPONENT;
30
31  COMPONENT SDC
32           PORT(
33              MCLK,reset : IN STD_LOGIC;
34              NOT_SS, SCLK, SDX, Sclose, Sopen, Psensor: IN STD_LOGIC; --software
35              OnOff, busy, OpenClose : OUT STD_LOGIC;
36              Dout : OUT STD_LOGIC_VECTOR(4 downto 0)
37           );
38  END COMPONENT;
39
40  COMPONENT door_mecanism IS
41           PORT( MCLK        : in std_logic;
42                 RST         : in std_logic;
43                 onOff       : in std_logic;
44                 openClose   : in std_logic;
45                 v         : in std_logic_vector(3 downto 0);
46                 Pswitch     : in std_logic;
47                 Sopen       : out std_logic;
48                 Sclose      : out std_logic;
49                 Pdetector   : out std_logic;
50                 HEX0        : out std_logic_vector(7 downto 0);
51                 HEX1        : out std_logic_vector(7 downto 0);
52                 HEX2        : out std_logic_vector(7 downto 0);
53                 HEX3        : out std_logic_vector(7 downto 0);
54                 HEX4        : out std_logic_vector(7 downto 0);
55                 HEX5        : out std_logic_vector(7 downto 0)
56                 );
57  END COMPONENT;
58
59
60
61  signal UsbPortInputSignal, UsbPortOutputSignal : STD_LOGIC_VECTOR (7 downto 0);
62  signal ScloseSignal, SopenSignal, PsensorSignal, OnOffSignal: STD_LOGIC;
63  signal DoutSignal :STD_LOGIC_VECTOR(4 downto 0);
64
65  begin
66
67
68
```

```
69  uUsbPort: UsbPort PORT MAP(
70            inputPort   => UsbPortInputSignal,
71            outputPort  => UsbPortOutputSignal
72  );
73
74
75  uSDC: SDC PORT MAP(
76            MCLK     => MCLK,
77            reset    => reset,
78            NOT_SS   => UsbPortOutputSignal(2), --0x04
79            SCLK     => UsbPortOutputSignal(1), --0x02
80            SDX      => UsbPortOutputSignal(0), --0x01
81            Sclose   => Sclosein,
82            Sopen    => Sopenin,
83            Psensor  => Pswitch,
84            OnOff    => OnOffSignal,
85            OpenClose => OpenClose,
86            busy     => UsbPortInputSignal(7), --0x80
87            Dout     => DoutSignal
88
89  );
90
91
92  --udoor_mecanism: door_mecanism PORT MAP (
93     --     MCLK     => MCLK,
94        -- RST      => reset,
95        --onOff     => OnOffSignal,
96     --openClose => DoutSignal(0), --OC
97     --v          => DoutSignal (4 downto 1),
98     --Pswitch    => Pswitch,
99     --Sopen      => SopenSignal,
100 -- Sclose    => ScloseSignal,
101 -- Pdetector => PsensorSignal,
102    --HEX0       => HEX0,
103    ---HEX1      => HEX1,
104 -- HEX2      => HEX2,
105    --HEX3       => HEX3,
106    --HEX4       => HEX4,
107    --HEX5       => HEX5
108 --);
109
110 Dout <= DoutSignal;
111 onOff <= OnOffSignal;
112 --Sopen <=SopenSignal;
113 --Sclose <= ScloseSignal;
114 Psensor <= Pswitch;
115
116 end arq_SDC_USBPORT;
```

E. Atribuição de pinos do módulo *SDC_USBPORT*

```
#============================================================
# CLOCK
#============================================================
set_location_assignment PIN_P11 -to MCLK

#============================================================
# SW
#============================================================
set_location_assignment PIN_C10 -to reset
set_location_assignment PIN_C11 -to Pswitch
set_location_assignment PIN_D12 -to Sopenin
set_location_assignment PIN_C12 -to Sclosein

#LED

set_location_assignment PIN_A8 -to Dout[0]
set_location_assignment PIN_A9 -to Dout[1]
set_location_assignment PIN_A10 -to Dout[2]
set_location_assignment PIN_B10 -to Dout[3]
set_location_assignment PIN_D13 -to Dout[4]
set_location_assignment PIN_C13 -to onOff
#set_location_assignment PIN_E14 -to Sopen
#set_location_assignment PIN_D14 -to Sclose
set_location_assignment PIN_A11 -to Psensor
set_location_assignment PIN_B11 -to OpenClose



#============================================================
# HEX0
#============================================================
set_location_assignment PIN_C14 -to HEX0[0]
set_location_assignment PIN_E15 -to HEX0[1]
set_location_assignment PIN_C15 -to HEX0[2]
set_location_assignment PIN_C16 -to HEX0[3]
set_location_assignment PIN_E16 -to HEX0[4]
set_location_assignment PIN_D17 -to HEX0[5]
set_location_assignment PIN_C17 -to HEX0[6]
set_location_assignment PIN_D15 -to HEX0[7]

#============================================================
# HEX1
#============================================================
set_location_assignment PIN_C18 -to HEX1[0]
set_location_assignment PIN_D18 -to HEX1[1]
set_location_assignment PIN_E18 -to HEX1[2]
set_location_assignment PIN_B16 -to HEX1[3]
set_location_assignment PIN_A17 -to HEX1[4]
set_location_assignment PIN_A18 -to HEX1[5]
set_location_assignment PIN_B17 -to HEX1[6]
```

```
set_location_assignment PIN_A16 -to HEX1[7]

#============================================================
# HEX2
#============================================================
set_location_assignment PIN_B20 -to HEX2[0]
set_location_assignment PIN_A20 -to HEX2[1]
set_location_assignment PIN_B19 -to HEX2[2]
set_location_assignment PIN_A21 -to HEX2[3]
set_location_assignment PIN_B21 -to HEX2[4]
set_location_assignment PIN_C22 -to HEX2[5]
set_location_assignment PIN_B22 -to HEX2[6]
set_location_assignment PIN_A19 -to HEX2[7]
#============================================================
# HEX3
#============================================================
set_location_assignment PIN_F21 -to HEX3[0]
set_location_assignment PIN_E22 -to HEX3[1]
set_location_assignment PIN_E21 -to HEX3[2]
set_location_assignment PIN_C19 -to HEX3[3]
set_location_assignment PIN_C20 -to HEX3[4]
et_location_assignment PIN_D19 -to HEX3[5]
set_location_assignment PIN_E17 -to HEX3[6]
set_location_assignment PIN_D22 -to HEX3[7]

#============================================================
# HEX4
#============================================================
set_location_assignment PIN_F18 -to HEX4[0]
set_location_assignment PIN_E20 -to HEX4[1]
set_location_assignment PIN_E19 -to HEX4[2]
set_location_assignment PIN_J18 -to HEX4[3]
set_location_assignment PIN_H19 -to HEX4[4]
set_location_assignment PIN_F19 -to HEX4[5]
set_location_assignment PIN_F20 -to HEX4[6]
set_location_assignment PIN_F17 -to HEX4[7]

#============================================================
# HEX5
#============================================================
set_location_assignment PIN_J20 -to HEX5[0]
set_location_assignment PIN_K20 -to HEX5[1]
set_location_assignment PIN_L18 -to HEX5[2]
set_location_assignment PIN_N18 -to HEX5[3]
set_location_assignment PIN_M20 -to HEX5[4]
set_location_assignment PIN_N19 -to HEX5[5]
set_location_assignment PIN_N20 -to HEX5[6]
set_location_assignment PIN_L19 -to HEX5[7]

#============================================================
# End of pin and io_standard assignments
#============================================================
```

F. Código *Kotlin - HAL*

```
object HAL {
           var written = 0b0000_0000
         private var ACTIVE = false
           fun init() { // Inicia a classe
               if(!ACTIVE) {
                   UsbPort.write(written)
                   ACTIVE=true
               }
           }

       // Retorna true se o bit tiver o valor lógico ‘1’
       fun isBit(mask: Int): Boolean = (mask and UsbPort.read()) != 0

       // 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
       /**
       * value -> 0000_1001.
       * mask -> 0000_1111.
       * lastWritten -> 1111_0111.
       * new lastWritten -> 1111_1001.
       * 1º: (value and mask) -> 0000_1001 sets the bits in value to be written to the ones in the
       mask.
       * 2º: (lastWritten and mask.inv()) -> 1111_0000 sets the bits in lastWritten that are not
       in the mask,this operation
       *      sets to 0 all the bits in lastWritten that are not in the mask, preparing it to
       receive the updated value.
       * 3º: (value and mask) or (lastWritten and mask.inv()) -> 0000_1001 or 1111_0000 ->
       1111_1001 sets the bits in
       *      lastWritten that are in the mask to the corresponding bits in value and keeps the
       bits that are not in the mask unchanged.
       */

       fun writeBits(mask: Int, value: Int) {
              written = (value and mask) or (written and mask.inv())
                UsbPort.write(written)
       }

       // Coloca os bits representados por mask no valor lógico ‘1’
       fun setBits(mask: Int) {
           written = (written or mask)
           UsbPort.write(written)
       }

       // Coloca os bits representados por mask no valor lógico ‘0’
       fun clrBits(mask: Int) {
           written = written and mask.inv()
           UsbPort.write(written)
       }

}
```

G. Código *Kotlin – Serial Emitter*

```
/**
 * Mapeamento
 *    n        7   6   5   4   3   2   1   0
 * inputPort :BSY   0   0   0   0   0   0   0      //inputPort(n)
 * outPort   : 0   0   0   0  LCD DOOR SCLK SDX //outputPort(n)
 *                            SS   SS
 * D[0:4] :    -   -   - D(3) D(2) D(1) D(0) RS
 **/

object SerialEmitter {    // Envia tramas para os diferentes módulos Serial Receiver
    enum class Destination {LCD, DOOR}
    private const val MASK_BUSY = 0x80
    private const val MASK_NOT_SS_LCD = 0x08
    private const val MASK_NOT_SS_DOOR = 0x04
    private const val MASK_SCLK = 0x02
    private const val MASK_SDX = 0x01
    private const val DATA_SIZE = 5

    // Inicia a classe
    fun init() {
         HAL.init()
         HAL.setBits(MASK_NOT_SS_LCD)  // SS = 1
         HAL.setBits(MASK_NOT_SS_DOOR)  // SS = 1
         HAL.clrBits(MASK_SCLK)    // SCLK = 0
         HAL.clrBits(MASK_SDX)     // SDX = 0
    }

    // Envia uma trama para o SerialReceiver identificado o destino em addr e os bits de dados
em‘data’.
    fun send(addr: Destination, value: Int) {
        var data = value
        while (isBusy()) { }
        val address = if (addr == Destination.LCD) MASK_NOT_SS_LCD else MASK_NOT_SS_DOOR
        for (i in 0 until DATA_SIZE) {
            HAL.clrBits(MASK_SCLK)                // SCLK = 0
            HAL.clrBits(address)                  // SS = 0
            HAL.writeBits(0x01, 0x01 and  data)   // SDX = data[0]
            data = data shr 1                     // data = data >> 1 to send bit a bit
            HAL.setBits(MASK_SCLK)                // SCLK = 1
        }
        HAL.clrBits(0x01)                        // SDX = 0
        HAL.clrBits(MASK_SCLK)                   // SCLK = 0
        HAL.setBits(address)                     // SS = 1

    }

    // Retorna true se o canal série estiver ocupado
    fun isBusy(): Boolean = HAL.isBit(MASK_BUSY)

}
```

H. Código Kotlin – Door Mechanism

```
/*
* Enviar pelo SerialEmmiter tal como no LCD
* Dados:     D4  D3  D2  D1 D0
*            V3  V2  V1  V0 OC
*
* OC -> 0 Fechar | 1 Abrir
* V3..0 -> Velocidade
* */


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

     // Inicia a classe, estabelecendoos 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 1)
//D4..1 -> Velocidade | D0 -> OC -> 1 Abrir
    }
    // Envia comando para fechar a porta, com o parâmetro de velocidade
    fun close(velocity: Int) {
        SerialEmitter.send(SerialEmitter.Destination.DOOR, (velocity shl 1 )or 0)
//D4..1 -> Velocidade | D0 -> OC -> 0 Fechar
    }
    // Verifica se o comando anterior está concluído
    fun finished(): Boolean = !SerialEmitter.isBusy()
}
```