O bloco *Serial Receiver* do S*LCDC* é constituído por três blocos principais: *i*) um bloco de controlo; *ii*) um contador de bit srecebidos; e *iii*) um bloco conversor série paralelo, designados respetivamente por *Serial Control, Counter*, e *Shift Register*

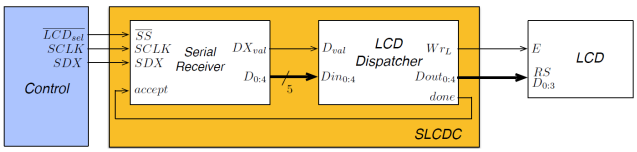

Figura 1 – Diagrama de blocos do módulo *Serial LCD Controller*

1 Serial Receiver

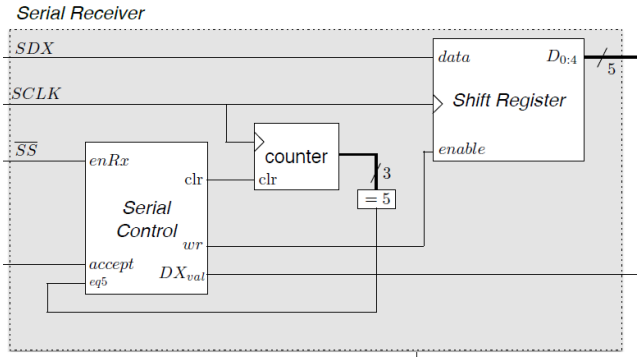

Figura 2 – Diagrama de Blocos do bloco *Serial Receiver*

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 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.

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

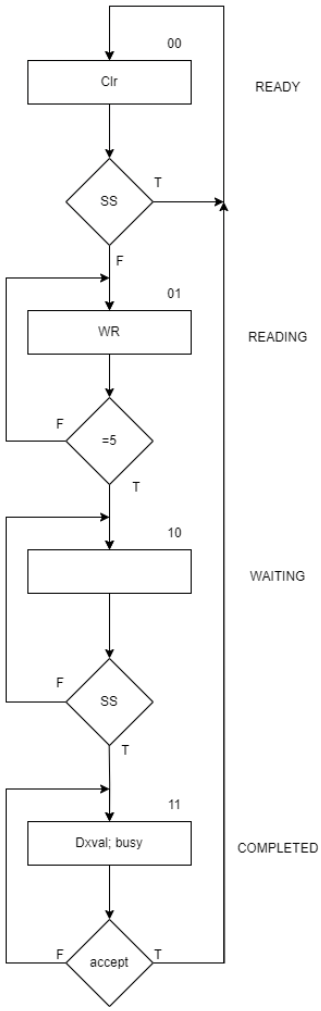

Figura 3 – Máquina de estados do bloco *Serial Control*

3 Dispatcher

O bloco *Dispatcher* entrega a trama recebida pelo *Serial Receiver* ao *LCD* através da ativação do sinal *WRL*, após este ter recebido uma trama válida, indicado pela ativação do sinal *DXval*.
O *LCD* processa as tramas recebidas de acordo com os comandos definidos pelo fabricante, não sendo necessário esperar pela sua execução para libertar o canal de receção série. Assim, o *Dispatcher* pode sinalizar ao *Serial Receiver* que a trama foi processada, ativando o sinal *done*.

O bloco *Dispatcher* foi implementado pela máquina de estados representada em *ASM-chart* na Figura 4.

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

Inicialmente o bloco Dispatcher encontra-se no estado 00, a aguardar que o valor lógico do sinal Dval seja 1 indicando uma trama válida. Quando recebe a indicação de uma trama válida, ocorre uma mudança para o estado 01, onde permanece com o sinal WRL ativo (ativando o Enable do LCD) até que sejam atingidos 12 ciclos de relógio de forma a escrever no LCD (mínimo 230 ns).
Ao atingir os 12 ciclos de relógio ocorre a passagem para o estado 10 onde o sinal done é ativado e mantem-se no estado 10 até que valor lógico do sinal Dval esteja desativado de modo a evitar múltiplas escritas, quando o sinal Dval estiver desativado regressa ao estado 00.

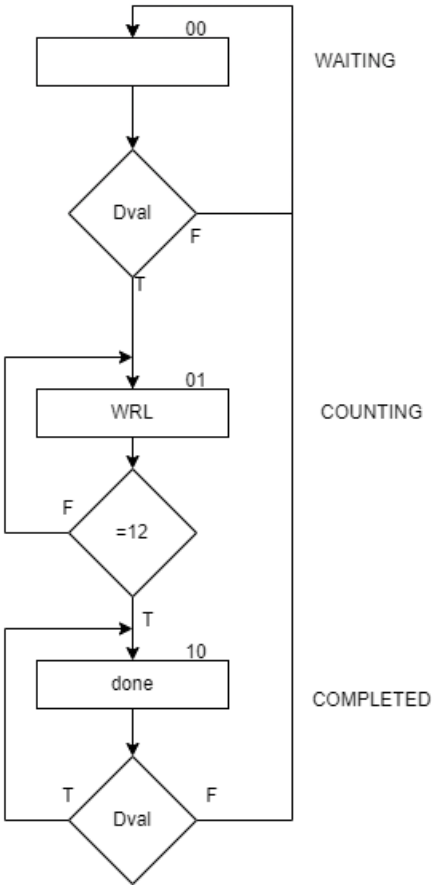

Figura 4 – Máquina de estados do bloco *Dispatcher*

Com base nas descrições dos blocos *Serial Receiver* e *Dispatcher* implementou-se o módulo *Serial LCD 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.

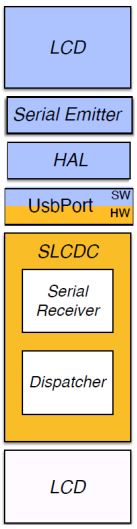

Figura 5 – Diagrama lógico do módulo *Control* de interface com o módulo *Serial LCD Controller*

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 LCD

A classe LCD é responsável por escrever no LCD usando a interface a 4 bits.
A função writeNibbleParalel() escreve um nibble de comando ou dados no LCD em paralelo.
Afunção writeNibbleSerial() recorre à função send() do SerialEmitter para escrever um nibble de comando ou dados no LCD em série.
A função writeNibble() escreve um nibble de comandos ou dados no LCD.
A função writeByte() escreve um byte de comandos ou dados no LCD.
A função writeCMD() escreve um comando no LCD.
A função writeDATA() escreve dados no LCD.
A função write(c:Char) escreve um caracter na posição corrente
A função write(c:String) escreve uma string na posição corrente.
A finção cursor() posiciona o cursor na posição desejada.
A função clear() limpa o ecrã e coloca o cursor em (0,0).

5 Conclusões

O módulo SLCDC tem como objetivo entregar a informação que lhe chega por parte do Control em forma de uma trama de 5 bits ao LCD. Para isso, implementámos o módulo Serial Receiver, responsável pela receção dos dados enviados pelo microcontrolador, e o bloco Dispatcher que tem a função de entregar os dados recebidos ao LCD após a trama ser validada. O módulo Serial Receiver foi testado no simulador e posteriormente na placa, apresentando um funcionamento correto. O módulo Dispatcher foi testado no simulador e mais tarde foi testado na placa, no módulo SLCDC_USBPORT, juntamente com o software, onde também observámos o correto funcionamento deste módulo.

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 *Dispatcher*

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3
4   entity Dispatcher is
5   port( Dval,clk,reset, Eq12 : IN STD_LOGIC;
6         WrL, done, enable, clr: OUT STD_LOGIC;
7         Din : IN STD_LOGIC_VECTOR(4 downto 0);
8         Dout : OUT STD_LOGIC_VECTOR(4 downto 0)
9   );
10  end Dispatcher;
11
12  architecture behavioral of Dispatcher is
13
14
15
16
17  type STATE_TYPE is (WAITING, COUNTING,COMPLETED);
18
19
20  signal CURRENT_STATE, NEXT_STATE : STATE_TYPE;
21
22  begin
23
24
25
26
27  --    Registo Current State
28
29  CURRENT_STATE <= WAITING when (reset = '1') else NEXT_STATE when rising_edge(clk);
30
31  --    Máquina de Estados
32
33  GenerateNextState:
34  process (CURRENT_STATE, Dval, Eq12)
35     begin
36        case CURRENT_STATE is
37           when WAITING         => if(Dval = '1') then
38                                      NEXT_STATE     <= COUNTING;
39                                   else
40                                      NEXT_STATE     <= WAITING;
41                                   end if;
42
43           when COUNTING        => if(Eq12 = '1') then
44                                      NEXT_STATE     <= COMPLETED;
45                                   else
46                                      NEXT_STATE     <= COUNTING;
47                                   end if;

48
49           when COMPLETED       => if(Dval = '0') then
50                                      NEXT_STATE     <= WAITING;
51                                   else
52                                      NEXT_STATE     <= COMPLETED;
53                                   end if;
54
55        end case;
56
57  end process;
58
59  --    Outputs
60  WrL <= '1' when (CURRENT_STATE = COUNTING) else '0';
61  done <= '1' when (CURRENT_STATE = COMPLETED) else '0';
62  clr <= '1' when (CURRENT_STATE = WAITING) else '0';
63
64  Dout <= Din when (CURRENT_STATE = COUNTING);
65
66  end behavioral;
```

C. Descrição VHDL do bloco *SLCDC*

```
1   LIBRARY IEEE;
2   USE IEEE.std_logic_1164.all;
3
4   ENTITY SLCDC IS
5      PORT( MCLK,reset : IN STD_LOGIC;
6            NOT_SS, SCLK, SDX: IN STD_LOGIC; --software
7            WrL : OUT STD_LOGIC;
8            Dout : OUT STD_LOGIC_VECTOR(4 downto 0)
9      );
10  END SLCDC;
11
12
13  ARCHITECTURE arq_SLCDC OF SLCDC IS
14
15  component COUNTER
16     PORT(  CLK : in std_logic;
17        E : in std_logic;
18        clr: in std_logic;
19        R : out std_logic_vector (3 downto 0)
20        );
21  end component;
22
23
24
25  component equalTo12Dispatcher
26  port( D : IN STD_LOGIC_VECTOR(3 downto 0);
27        F : OUT STD_LOGIC
28  );
29  end component;
30
31
32  COMPONENT SerialReceiver
33     port(
34        SDX, SCLK, SS, accept, MCLK, reset : in std_logic;
35        DXval, busy : out std_logic;
36        data : out std_logic_vector (4 downto 0)
37     );
38  END COMPONENT;
39
40  COMPONENT Dispatcher
41     port( Dval,clk,reset, Eq12 : IN STD_LOGIC;
42           WrL, done, clr : OUT STD_LOGIC;
43           Din : IN STD_LOGIC_VECTOR(4 downto 0);
44           Dout : OUT STD_LOGIC_VECTOR(4 downto 0)
45     );
46  END COMPONENT;
47
48  COMPONENT CLKDIV is
49     generic(div: natural := 50000000);
50     port ( clk_in: in std_logic;
51            clk_out: out std_logic);
52  END COMPONENT;
53
54  signal DXvalSignal, SCLKSignal, MCLKDivSignal, acceptDoneSignal, eq12Signal, WrLSignal,
    enableSignal,clrSignal : STD_LOGIC;
55  signal DataSignal : STD_LOGIC_VECTOR (4 downto 0);
56  signal counterSignal : STD_LOGIC_VECTOR (3 downto 0);
57
58  begin
59
60  WrL <= WrLSignal;
61
62  --uCLKDIV : CLKDIV generic map(2)  port map(
63     --  clk_in => MCLK,
64        -- clk_out => MCLKDivSignal
65  --);
66
67
68  UCOUNTER: COUNTER port map(
69     Clk => MCLK,--MCLKDivSignal,
70     E => WrLSignal,
71     clr => clrSignal,
72     R => CounterSignal
73  );
74
```

```
75  uequalTo12Dispatcher : equalTo12Dispatcher port map (
76                  D=>counterSignal,
77                  F=>eq12Signal
78  );
79
80  uSerialReceiver: SerialReceiver PORT MAP (
81                  SDX    => SDX,
82                  SCLK   => SCLK,
83                  SS     => NOT_SS,
84                  accept => acceptDoneSignal,
85                  MCLK   => MCLK,--MCLKDivSignal,
86                  reset  => reset,
87                  DXval  => DXvalSignal,
88                  data   => DataSignal
89                  );
90
91  uDispat: Dispatcher PORT MAP (
92                  Dval  => DXvalSignal,
93                  clk   => MCLK,--MCLKDivSignal,
94                  reset => reset,
95                  WrL   => WrLSignal,
96                  done  => acceptDoneSignal,
97                  Din   => DataSignal,
98                  Dout  => Dout,
99                  Eq12 => eq12Signal,
100                 clr => clrSignal
101
102                 );
103
104
105
106
107 end arq_SLCDC;
108
109
```

D. Descrição VHDL do bloco *SLCDC_USBPORT*

```
1   LIBRARY IEEE;
2   USE IEEE.std_logic_1164.all;
3
4   ENTITY SLCDC_USBPORT IS
5      PORT(
6
7            MCLK,reset : IN STD_LOGIC;
8            WrL , WrLED, SDX,SCLK : OUT STD_LOGIC;
9            Dout : OUT STD_LOGIC_VECTOR(4 downto 0);
10           LDout : OUT STD_LOGIC_VECTOR(4 downto 0)
11
12
13     );
14  END SLCDC_USBPORT;
15
16
17
18
19  ARCHITECTURE arq_SLCDC_USBPORT OF SLCDC_USBPORT IS
20
21  COMPONENT UsbPort
22           PORT
23              (
24                 inputPort:  IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
25                 outputPort :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0)
26              );
27  END COMPONENT;
28
29  COMPONENT SLCDC
30           PORT( MCLK,reset : IN STD_LOGIC;
31                 NOT_SS, SCLK, SDX: IN STD_LOGIC; --software
32                 WrL  : OUT STD_LOGIC;
33                 Dout : OUT STD_LOGIC_VECTOR(4 downto 0)
34           );
35  END COMPONENT;
36
37
38  COMPONENT CLKDIV is
39     generic(div: natural := 50000000);
40     port ( clk_in: in std_logic;
41             clk_out: out std_logic);
42  END COMPONENT;
43
44
45
46  signal UsbPortInputSignal, UsbPortOutputSignal : STD_LOGIC_VECTOR (7 downto 0);
47  signal MCLKDivSignal, WrLSignal: STD_LOGIC;
48  signal DoutSignal : STD_LOGIC_VECTOR (4 downto 0);
49
50  begin
51
52  --uCLKDIV : CLKDIV   port map(
53     --  clk_in => MCLK,
54        -- clk_out => MCLKDivSignal
55  --);
56
57
58  uUsbPort: UsbPort PORT MAP(
59           inputPort   => UsbPortInputSignal,
60           outputPort  => UsbPortOutputSignal
```

```
61  );
62
63
64  uSLCDC: SLCDC PORT MAP(
65            MCLK     => MCLK,
66            reset    => reset,
67            NOT_SS   => UsbPortOutputSignal(3), --0x08
68            SCLK     => UsbPortOutputSignal(1), --0x02
69            SDX      => UsbPortOutputSignal(0), --0x01
70            WrL      => WrLSignal,
71            Dout     => DoutSignal
72
73  );
74
75  SDX <= UsbPortOutputSignal(0); --0x01;
76  SCLK <= UsbPortOutputSignal(1); --0x02;
77  LDout <= DoutSignal;
78  Dout <= DoutSignal;
79  WrL <= WrLSignal;
80  WrLED <= WrLSignal;
81
82  end arq_SLCDC_USBPORT;
```

E. Atribuição de pinos do módulo *SLDC_USBPORT*

```
#============================================================
# CLOCK
#============================================================

set_location_assignment PIN_P11 -to MCLK

#============================================================
# SW
#============================================================
set_location_assignment PIN_C10 -to reset



#============================================================
#LCD
#============================================================
set_location_assignment PIN_W8     -to Dout[0]
set_location_assignment PIN_V5     -to WrL


set_location_assignment PIN_W11    -to Dout[1]
set_location_assignment PIN_AA10   -to Dout[2]
set_location_assignment PIN_Y8     -to Dout[3]
set_location_assignment PIN_Y7     -to Dout[4]


#============================================================
# LED
#============================================================
set_location_assignment PIN_A8 -to LDout[0]
set_location_assignment PIN_A9 -to WrLED
set_location_assignment PIN_A10 -to LDout[1]
set_location_assignment PIN_B10 -to LDout[2]
set_location_assignment PIN_D13 -to LDout[3]
set_location_assignment PIN_C13 -to LDout[4]
#set_location_assignment PIN_E14 -to LEDR[6]
#set_location_assignment PIN_D14 -to LEDR[7]
set_location_assignment PIN_A11 -to SDX
set_location_assignment PIN_B11 -to SCLK



#============================================================
# 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 – LCD

```
/**
 * LCD 16*2
 * Display positions
 *
 * 1º line: 0x00 to 0x0F
 * 2º line: 0x40 to 0x4F
 *
 * DDRAM : Display data RAM
 * CGRAM : Character generator RAM
 * ACG   : CGRAM address
 * ADD   : DDRAM address (cursor address)
 * AC    : address counter used for DD and CGRAM addresses
 * DDRAM 0 0 1 ADD ADD ADD ADD ADD ADD
 *
 */
object LCD {
    // Escreve no LCD usando a interface a 4bits
    private const val LINES = 2
    const val COLS = 16  // Dimensão do display.
    private const val DISPLAY_ON = 0xF //Mascara para ligar o display,
    private const val DISPLAY_OFF = 0x8 //Mascara para desligar o display
    private const val DISPLAY_SET = 0x3 //Function set
    private const val MASK_ENTRYMODE = 0x6 //Mascara para entry mode
    private const val DISPLAY_SET_NIBBLE = 0x2 //set 4 bits
    private const val MASK_PARALLEL_RS = 0x10 //Mascara para o bit de RS
    private const val MASK_LOW_DATA = 0x0F //Mascara para os 4 bits menos significativos
    private const val MASK_HIGH_DATA = 0xF0 //Mascara para os 4 bits mais significativos
    private const val DISPLAY_CLEAR : Int = 0x1 //Mascara para instrução de limpar o display
    private const val DISPLAY_CONFIG :Int=0x28//Mascara para configurar as linhas e a fonte do LCD


    // Escreve um nibble de comando/dados no LCD em paralelo
    //data -> d3..d0, recebe os quatro bits de menor peso
    private fun writeNibbleParallel(rs: Boolean, data: Int){
        if (rs) {
            HAL.setBits(MASK_PARALLEL_RS) //RS = 1
        } else {
            HAL.clrBits(MASK_PARALLEL_RS)//RS = 0
        }
        HAL.writeBits(MASK_LOW_DATA, data) //d3..d0
    }


    // Escreve um nibble de comando/dados no LCD em série
    private fun writeNibbleSerial(rs: Boolean, data: Int) {
        var d = data
        if (rs) d = (data shl 1) or 0x01 else d = (d shl 1) or 0x00
        SerialEmitter.send(SerialEmitter.Destination.LCD, d)
    }


    // Escreve um nibble de comando/dados no LCD
    private fun writeNibble(rs: Boolean, data: Int) {
        writeNibbleSerial(rs,data)
```

```
    }

    // Escreve um byte de comando/dados no LCD
    private fun writeByte(rs: Boolean, data: Int) {
        writeNibble(rs, (MASK_HIGH_DATA and data) shr 4)
        writeNibble(rs, MASK_LOW_DATA and data)
    }

    // Escreve um comando no LCD
    private fun writeCMD(data: Int) {
        writeByte(false, data)

    }
    // Escreve um dado no LCD
    private fun writeDATA(data: Int) {
        writeByte(true, data)

    }

    // Envia a sequência de iniciação para comunicação a 4 bits.
    fun init() {
        SerialEmitter.init()
        Time.sleep(15)
        writeNibble(false, DISPLAY_SET)
        Time.sleep(5)
        writeNibble(false, DISPLAY_SET)
        Time.sleep(1)
        writeNibble(false, DISPLAY_SET)
        writeNibble(false, DISPLAY_SET_NIBBLE)
        // Function Set, interface a 4 bits
        writeCMD(DISPLAY_CONFIG) // define N:1, F:0
        writeCMD(DISPLAY_OFF)  // display off
        writeCMD(DISPLAY_CLEAR)    // clear
        writeCMD(MASK_ENTRYMODE)  // define I/D:1, S:0
        writeCMD(DISPLAY_ON) // display on
    }

    // Escreve um caracter na posição corrente.
    fun write(c: Char) =
        writeDATA(c.code)

    // Escreve uma string na posição corrente.
    fun write(text: String)  {
        for (c in text) {
            write(c)
        }

    }

    // Envia comando para posicionar cursor (‘line’:0..LINES-1 , ‘column’:0..COLS-1) fun
cursor(line: Int, column: Int) ...
    fun cursor(line: Int, column: Int): Unit {
         if (line >= LINES || line < 0 || column >= COLS || column < 0) return
        writeCMD((line * 0x40 + column) or 0x80)
    }
```

```
    // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
    fun clear() {
        writeCMD(DISPLAY_CLEAR)
        cursor(0,0)
    }

}
```