# Projecto de uma placa com DIP-Switches e LCD com ligação à placa DETIUA

Liliana Rocha Nicolau Lopes da Costa 27611

**Ano Lectivo de 2006/2007**
**Universidade de Aveiro**

## 1. Objectivos

→ Construir uma placa que pode ser ligada com a placa DETIUA através do barramento de extensão

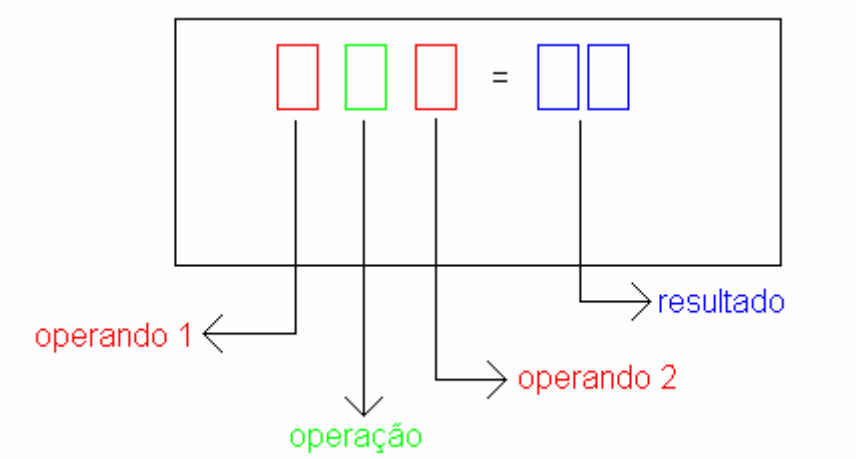
→ Desenvolver um projecto que permiter ler os DIP-switches que representam dois operandos e um operador e visualizar o resultado da operação no LCD

## 2. Projecto da Placa

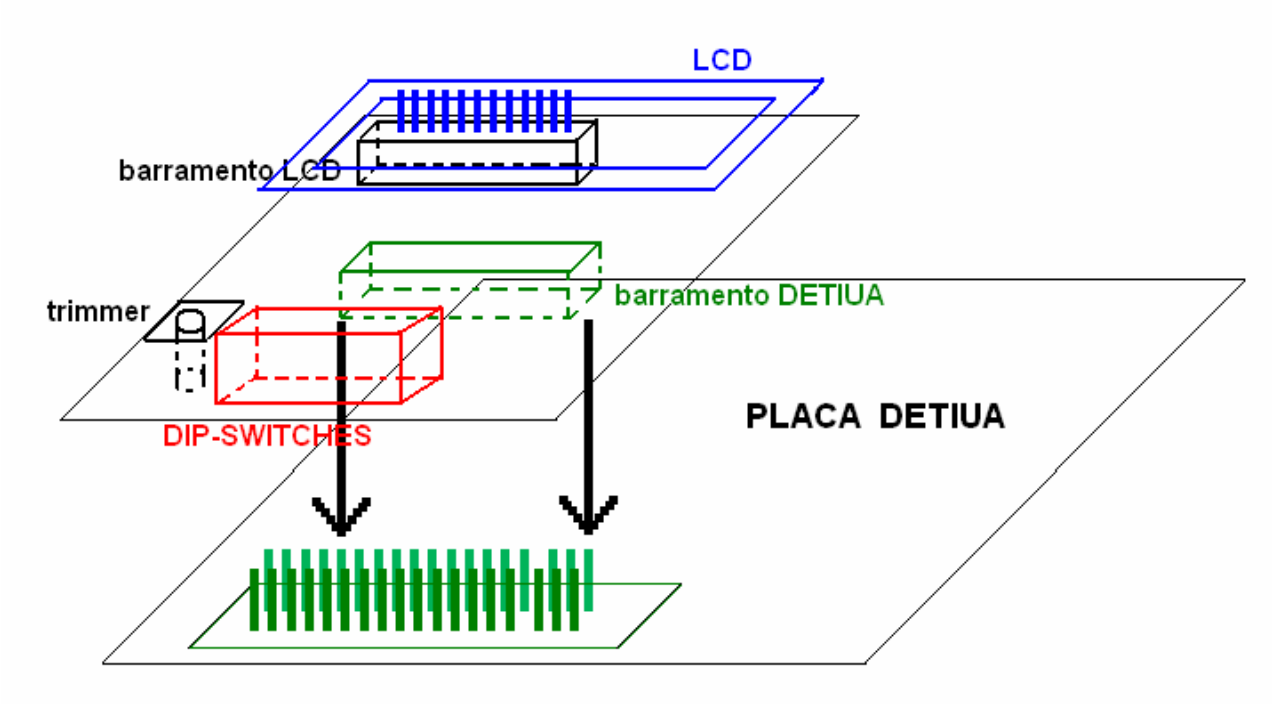
Foi escolhido um LCD LM052L, cujo controlador é um HD44780, e que possui duas linhas de 16 caracteres.A placa possui um trimmer que permite controlar a intensidade luminosa do LCD.

Existem 10 DIP-switches que permitem seleccionar os valores pretendidos para os operandos, cujo valor máximo decimal é 9, assim como o tipo de operação a realizar, que pode ser adição, subtracção, multiplicação ou divisão, da seguinte forma:


No LCD deverá aparecer a seguinte informação:



A placa é ligada à placa DETIUA da seguinte forma:

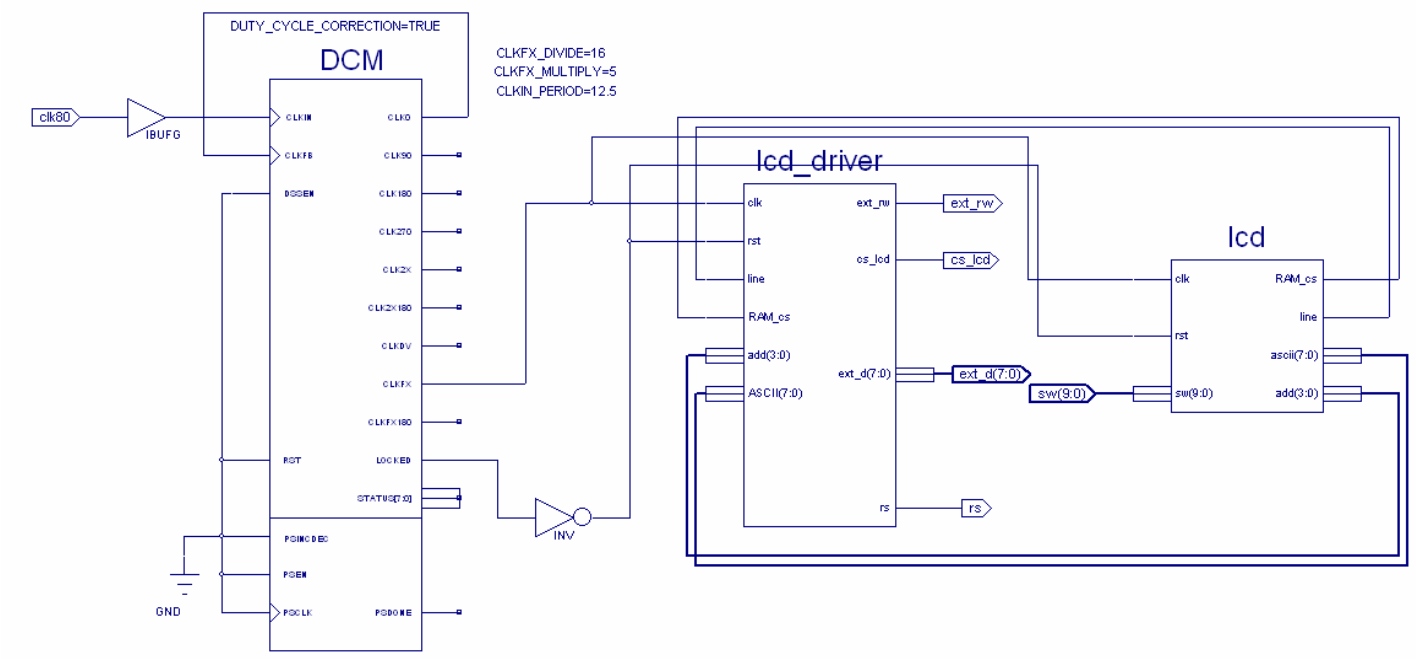



Em anexo, estão os esquemas referentes à placa projectada.

## 3. Projecto da tarefa de Demonstração de Funcionamento da Placa

Este projecto é baseado nos projectos efectuados nas aulas práticas e nos tutorials referentes à utilização e interface com LCD's e DIP-switches.

O esquema usado é o seguinte:




A unidade DCM é usada para converter o sinal de relógio de 80 MHz num sinal de relógio de 25 MHz, multiplicando o sinal de 80 MHZ por 5 e dividindo-o por 16.

A unidade LCD_driver é a unidade de controlo que faz o interface entre os barramentos de dados/controlo com as ligações físicas com a placa projectada que contém o LCD e os DIP-switches. Esta unidade possui as seguintes funcionalidades:
- possui uma memória RAM, de tamanho 16x2, que armazena os 32 caracteres a serem visualizados ou escritos no LCD (o seu interface com o módulo LCD é feito através dos barramentos ASCII e add e dos sinais RAM_cs e line);
- faz o chip select do LCD (através do sinal lcd_cs);
- activa a escrita de dados no LCD (nunca se utiliza a leitura de dados para esta tarefa) (através do sinal ext_rw);
- envia as instrucções que permitem visualizar os dados no LCD, assim como os dados (através do barramento ext_d[7:0] e do sinal rs);
- inicializa o LCD por instrução, quando necessário.

A unidade LCD é a que faz o interface com os DIP-switches, lendo o valor que neles é colocado pelo utilizador (barramento sw[9:0]) e actualiza os caracteres a visualizar no LCD. Para isso:
- faz o chip-select da RAM do módulo LCD_driver (através do sinal RAM_cs);
- seleciona a linha do LCD onde pretende escrever (através do sinal line);
- seleciona o número do caracter dentro da linha que pretende alterar, sendo que cada linha possui 16 caracteres (através do barramento add[3:0]);
- envia o código ascii do caracter a alterar (através do sinal ascii).

O sinal de relógio de ambos os módulos é de 25 MHZ, que vem da saída CLKFX do bloco DCM. O sinal de reset é o proveniente da saída LOCKED do DCM, que é posteriormente invertido para que seja um sinal de reset activo ao valor lógico '0'.

## → *Descrição do módulo LCD*

```
entity LCD is
    Port( clk: in std_logic;
          rst: in std_logic;
          sw: in std_logic_vector(9 downto 0);
          ascii: out std_logic_vector(7 downto 0);
          RAM_cs: out std_logic;
          line: out std_logic;
          add: out std_logic_vector(3 downto 0)
          );
end LCD;
```

**A funcionalidade destes sinais já foi descrita anteriormente.**

```
architecture Behavioral of LCD is

signal state: natural range 0 to 31; -- sinal auxiliar que define os 32 carateres do LCD
signal linha: std_logic; -- sinal que selecciona uma das linhas do LCD
signal address: std_logic_vector(3 downto 0); -- sinal que dá o nº caracter dentro da linha
signal asc: std_logic_vector(7 downto 0); -- sinal do código ascii do caracter a imprimir no LCD
signal idx: integer range 0 to 3; -- fases do relógio
signal op1, op2: std_logic_vector(3 downto 0); -- operandos
signal result_bcd: std_logic_vector(7 downto 0); -- resultado em BCD
signal result: std_logic_vector(7 downto 0); -- resultado em binário
signal div_res: std_logic_vector(3 downto 0); -- resultado da divisão
signal sinal_op, sinal_res: character; -- sinais da operação e do resultado
signal dividendo, divisor: std_logic_vector(3 downto 0);

-- os operandos possuem apenas um digito de 0 a 9 em BCD


type ROM is array (0 to 81) of std_logic_vector(7 downto 0);
constant conv_table: ROM := (
      0=>x"00", 1=>x"01", 2=>x"02", 3=>x"03", 4=>x"04", 5=>x"05", 6=>x"06", 7=>x"07", 8=>x"08", 9=>x"09", 10=>x"10",
      11=>x"11", 12=>x"12", 13=>x"13", 14=>x"14", 15=>x"15", 16=>x"16", 17=>x"17", 18=>x"18", 19=>x"19", 20=>x"20",
      21=>x"21", 22=>x"22", 23=>x"23", 24=>x"24", 25=>x"25", 26=>x"26", 27=>x"27", 28=>x"28", 29=>x"29", 30=>x"30",
      31=>x"31", 32=>x"32", 33=>x"33", 34=>x"34", 35=>x"35", 36=>x"36", 37=>x"37", 38=>x"38", 39=>x"39", 40=>x"40",
      41=>x"41", 42=>x"42", 43=>x"43", 44=>x"44", 45=>x"45", 46=>x"46", 47=>x"47", 48=>x"48", 49=>x"49", 50=>x"50",
      51=>x"51", 52=>x"52", 53=>x"53", 54=>x"54", 55=>x"55", 56=>x"56", 57=>x"57", 58=>x"58", 59=>x"59", 60=>x"60",
      61=>x"61", 62=>x"62", 63=>x"63", 64=>x"64", 65=>x"65", 66=>x"66", 67=>x"67", 68=>x"68", 69=>x"69", 70=>x"70",
      71=>x"71", 72=>x"72", 73=>x"73", 74=>x"74", 75=>x"75", 76=>x"76", 77=>x"77", 78=>x"78", 79=>x"79", 80=>x"80",
      81=>x"81");
```

**Este módulo altera o valor dos caracteres a visualizar no LCD, consoante os valores inseridos pelo utilizador nos DIP-switches, através da sua escritana memória RAM da unidade de controlo LCD_driver. Para isso, procedem-se às seguintes operações:**

**1)** os valores dos operandos são armazenados nos sinais internos op1 e op2 em binário, sendo o primeiro operando op1 correspondente aos quatro primeiros DIP-switches (a contar da esquerda para a direita) e o segundo operando op2 correspondente ao segundo grupo de quatro DIP-switches.

```
-- geração dos operandos em binário
op1 <= sw(9 downto 6);
op2 <= sw(5 downto 2);
```

**2)** A operação a realizar sobre os operandos é determinada a cada transição positiva do sinal de relógio e é dada pelo valor dos dois últimos DIP-switches. Consoante o valor binário destes é atribuída uma das quatro operações possíveis.

```
-- geração do sinal de operação
process(clk, rst)
begin
if rst = '0' then
   null;
elsif rising_edge(clk) then
case sw(1 downto 0) is
      when "00" => sinal_op <= '+';

      when "01" => sinal_op <= '-';

      when "10" => sinal_op <= '*';

      when "11" => sinal_op <= '/';

      when others => null;
end case;
end if;
end process;
```

**3)** O resultado, ainda em valor binário, é calculado assim como o sinal resultante da operação:
- para adição, multiplicação e divisão, assume-se que os valores são sempre positivos e por isso o resultado é sempre positivo, sendo o caracter correspondente ao sinal do resultado *sinal_res* um espaço;
- para subtracção, se o operando 1 for superior ao operando 2, então o resultado é positivo; na situação contrária, o resultado é negativo e, por isso, o caracter associado ao sinal do resultado *sinal_res* é '-';

```
-- geração do resultado em binário
process(clk, rst)
begin
if rst = '0' then
   null;
elsif rising_edge(clk) then
case sw(1 downto 0) is
      when "00" => result <= ("0000" & op1) + ("0000" & op2);
                   sinal_res <= ' ';

      when "01" => if op1 >= op2 then
                        result <= ("0000" & op1) - ("0000" & op2);
                        sinal_res <= ' ';
                   else
                        result <= ("0000" & op2) - ("0000" & op1);
                        sinal_res <= '-';
                   end if;

      when "10" => result <= op1*op2;
                   sinal_res <= ' ';

      when "11" => result <= "0000" & div_res;
                   sinal_res <= ' ';

      when others => result <= (others => '0');
                   sinal_res <= ' ';
end case;
end if;
end process;
```

- para a divisão é criado outro processo que impede a divisão por zero e que, a cada ciclo de relógio verifica se o dividendo é superior ao divisor e faz o seguinte:

**a) 1º ciclo de relógio**
- verifica que a operação escolhida é a divisão, coloca os valores dos operandos nos sinais internos *dividendo* e *divisor* e inicializa o resultado da divisão *div_res* a zero; por exemplo, dividendo = 5 e divisor = 2;

**b) 2º ciclo de relógio**
- verifica que (*dividendo* >= *divisor*) = TRUE;
- *dividendo* = 2;
- *divisor* = 2;
- *div_res* = 1;

**c) 3º ciclo de relógio**
- verifica que (*dividendo* >= *divisor*) = TRUE;
- *dividendo* = 0;
- *divisor* = 2;
- *div_res* = 2;

**d) 4º ciclo de relógio**
- verifica que (*dividendo* >= *divisor*) = FALSE;
- *dividendo* = 0;
- *divisor* = 2;
- *div_res* = 2;

```
-- processo de divisão
Division: process(clk, rst)
begin
   if rst = '0' then div_res  <= (others => '0');
   elsif rising_edge(clk) then
      if sw(1 downto 0) = "11" then
         divisor <= op2;
         dividendo <= op1;
         div_res <= (others => '0');
      elsif (dividendo >= divisor) and divisor /= "0000" then
         dividendo <= dividendo - divisor;
         div_res <= div_res + 1;
      end if;
   end if;
end process Division;
```

**4)** É necessário converter o resultado final em binário para o seu valor em BCD (que é o seu valor decimal), para depois determinar o código ascii correspondente e, para isso, usa-se uma memória ROM cujo índice possui uma correspondência directa com o valor em BCD (armazenado em hexadecimal) do resultado final da operação realizada.

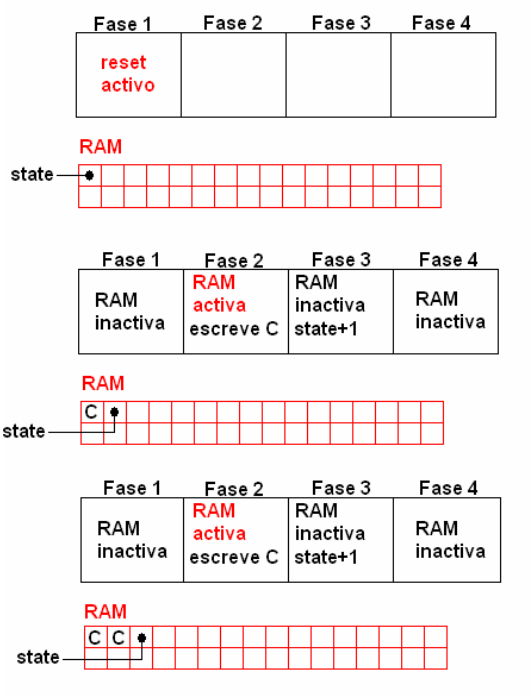
```
-- conversão do resultado em binário para BCD
result_bcd <= conv_table(conv_integer(result));
```

**5)** É necessário definir quatro fases de relógio que vão possibilitar a correcta escrita de valores no LCD, respeitando as restrições temporais. O valor do sinal interno idx será ao longo do tempo igual a 0,1,2,3,0,1,2,3,0,1,2,3,…

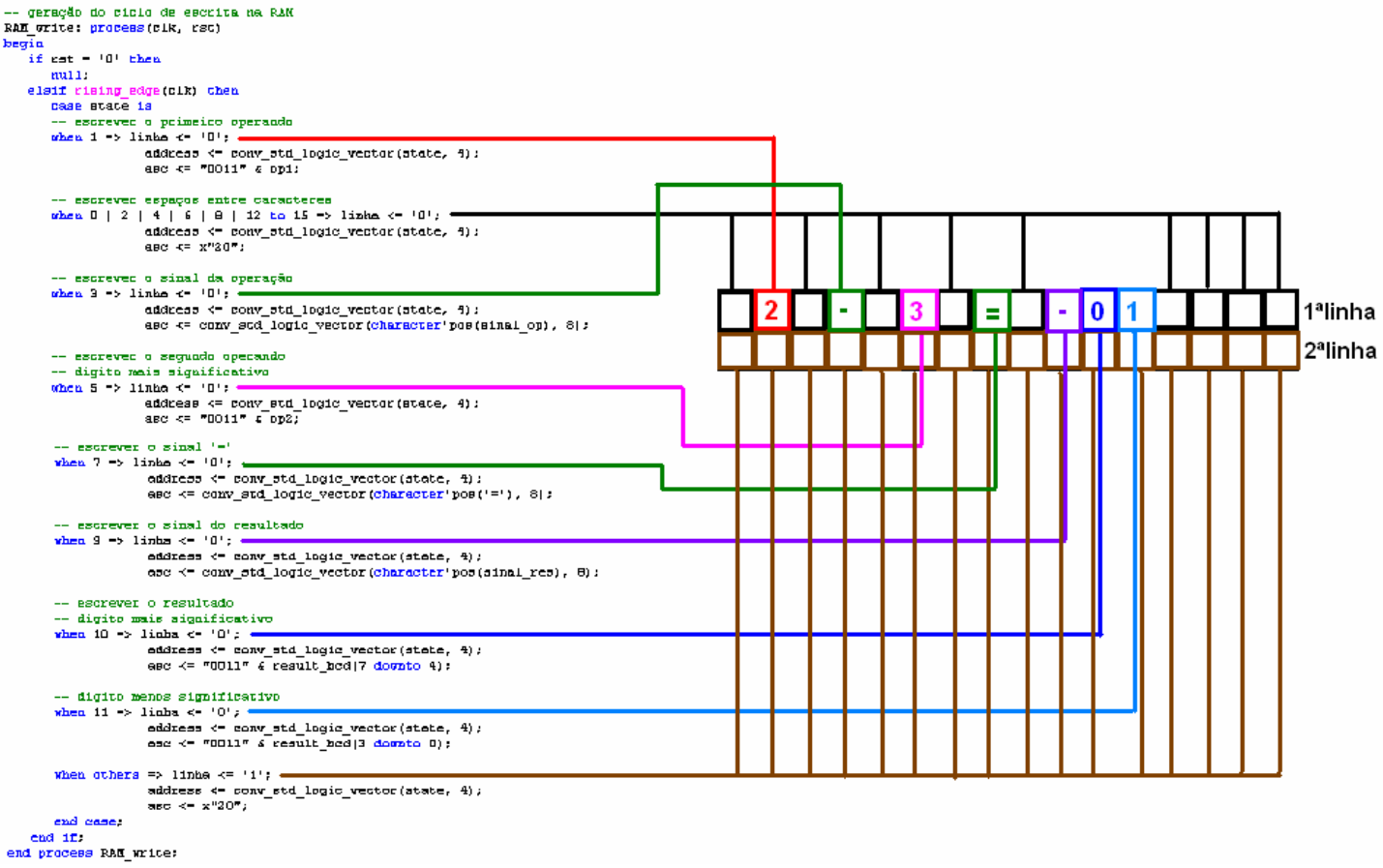
```
-- geração das fases de relógio
index_ger: process(clk, rst)
begin
   if rst = '0' then
      idx <= 0;
   elsif rising_edge(clk) then
      if idx = 3 then idx <= 0;
      else idx <= idx + 1;
      end if;
   end if;
end process index_ger;
```

**6)** O seguinte processo permite definir quando é que a escrita na memória RAM de todos caracteres a visualizar no LCD é efectuada da seguinte forma:

```
-- geração dos estados do ciclo de escrita na RAM
RAM_write_cycle: process(clk, rst)
begin
    if rst = '0' then
        RAM_cs <= '1';
        state <= 0;
    elsif rising_edge(clk) then
        case idx is
            when 0 => RAM_cs <='1';
            when 1 => RAM_cs <='0'; --activo
            when 2 => RAM_cs <='1';
                      state <= state + 1;
            when 3 => null;
        end case;
    end if;
end process RAM_write_cycle;
```




**7)** O seguinte esquema explica como é que se procede à escrita dos caracteres pretendidos nas correspondentes posições da memória RAM (notar que as posições da memória RAM têm uma correspondência com as posições dos caracteres no ecrã do LCD).



**8)** Por fim, os valores dos sinais internos são atribuídos aos correspondentes sinais externos do módulo LCD que fazem ligação com o módulo LCD_driver.

```
    ascii <= asc;
    add <= address;
    line <= linha;

end Behavioral;
```

## → *Descrição do módulo LCD_driver*

**Este módulo possui uma memória RAM que armazena os valores a escrever no LCD, como já foi dito, e a sua necessidade justifica-se pelo facto de os caracteres a visualizar irem mudando ao longo do tempo através da acção do utilizador sobre os DIP-switches, havendo uma necessidade de escrever constantemente para o LCD valores que mudam aleatoriamente e, por isso, tem de existir uma forma de armazenamento destes, para que a visualização de dados no LCD seja transparente ao utilizador face às alterações que este induz.**

**Este módulo possui também um sinal de relógio interno *clk_lcd*, para que sejam respeitadas as restrições temporais necessárias para a correcta envio de dados e instrucções para o LCD.**

```
entity LCD_driver is
    Port( clk: in std_logic;   -- relogio de 25 MHz
          rst: in std_logic;  -- reset (activo a '0')
          rs: out std_logic; -- sinal que seleciona dados/instrução
          ext_d: out std_logic_vector(7 downto 0); -- codigo ASCII do caracter a visualizar no LCD
          ext_rw: out std_logic; -- LCD read/write
          cs_lcd: out std_logic; -- ship select do LCD (activo a '1')
          line: in std_logic; -- linha do display LCD ('0' primeira linha e '1' segunda linha)
          add: in std_logic_vector(3 downto 0); -- nº caracter dentro de uma linha (de 0 a 15)
          ASCII: in std_logic_vector(7 downto 0); -- codigo ascii do caracter a guardar no endereço add na memoria do LCD
          RAM_cs: in std_logic -- chip select da memória RAM do LCD(activo a '0')
      );
end LCD_driver;

architecture Behavioral of LCD_driver is

type RAM is array (0 to 31) of std_logic_vector(7 downto 0);
signal LCD_RAM: RAM;

signal clk_lcd : std_logic;
signal div: unsigned(14 downto 0); -- sinal para divisao do relogio de 25 MHz
signal reset: std_logic;
signal cmd: std_logic_vector(7 downto 0);
signal s_rs: std_logic;
signal idx: integer range 0 to 3;
signal cs: std_logic;
signal cnt: natural range 0 to 64;
```

**1)** Este processo faz simplesmente com que o código ascii, proveniente do sinal externo *ASCII,* do caracter a alterar na memória RAM seja escrito na respectiva posição de memória. O cálculo da posição é feito convertendo o valor em binário da concatenação entre o bit do sinal *line* (que indica em qual das duas linhas está a posição pretendida) e o valor do vector *add* (que indica qual o caracter dentro da linha, de 1 a 16) num valor inteiro, que corresponde ao índice de memória pretendido. Isto ocorre para a transição negativa do sinal enable do chip select do LCD, porque é neste momento que ocorre o *lacth* dos dados.

```
RAM_write: process(rst, RAM_cs)
begin
    if rst = '0' then
        null;
    elsif falling_edge(RAM_cs) then
        LCD_RAM(conv_integer(line & add)) <= ASCII;
    end if;
end process RAM_write;
```

| Control SIgnal | Function |
|---|---|
| E | Causes data/control state to be latched<br>Rising Edge = Latches control state (RS and R_W)<br>Falling Edge = Latches data |

**2)** Este processo gera o relógio local referido anteriormente, sendo o seu valor aproximadamente igual a 25 MHz / $2^{14}$ = 1.5 KHz, o que corresponde a um período de cerca de 667 us.

```
local_clock: process(clk, rst)
begin
   if rst = '0' then
      div <= (others => '0');
   elsif rising_edge(clk) then
      div <= div + 1;
   end if;
end process local_clock;

   clk_lcd <= div(div'left); -- clk_lcd é o relógio local
```

**3)** Este processo cria quatro fases para o relógio local, de forma semelhante à que foi implemetada no outro módulo.

```
index_ger: process(clk_lcd, rst)
begin
   if rst = '0' then
      idx <= 0;
   elsif rising_edge(clk_lcd) then
      if idx = 3 then
         idx <= 0;
      else
         idx <= idx + 1;
      end if;
   end if;
end process index_ger;
```

**4)** Este processo descreve o ciclo de escrita no LCD:
- com o sinal de reset global activo, o LCD é desactivado, o sinal *rs* é colocado a zero (para indicar que vai ser enviada uma instrução) e no barramento de dados *ext_d* é enviada a instrução, que nada faz.
- para cada ciclo de relógio são definidas as seguintes operações por fase de relógio:

**a)** 1ª fase: definir se a informação que é enviada pelo barramento *ext_d* é uma instrução ou se são dados (*rs* = '1' corresponde a dados e *rs* = '0' corresponde a instrução ou comando);

**b)** 2ªfase: activar o LCD, colocando o seu chip select *lcd_cs* a '1' e enviar o comando/dados (que estão armazenados no sinal interno *cmd*) pelo barramento de 8 bits ext_d;

**c)** 3ª fase: desactivar o LCD;

**d)** 4ª fase: não se faz nada (notar que na quarta fase não se procede a nenhuma operação útil, ela só é definida porque dividir um sinal de relógio por 3 fases com precisão não é possível).

```
LCD_write_cycle: process(clk_lcd, rst)
begin
   if rst = '0' then
      cs <= '0'; --desactivar o chip select
      rs <= '0';
      ext_d <= (others => '0');
   elsif rising_edge(clk_lcd) then
      case idx is
         when 0 => rs <= s_rs;
         -- escrever dados, activar o chip select
         when 1 => cs <= '1';
                   ext_d <= cmd;
         -- desactivar o chip select
         when 2 => cs <= '0';
         when 3 => null;
      end case;
   end if;
end process LCD_write_cycle;

   ext_rw <= '0';
   cs_lcd <= cs;
```

| | |
|---|---|
| RS | Register Select Control<br>1 = LCD in data mode<br>0 = LCD in command mode |

| | |
|---|---|
| R_W | Read / Write control<br>1 = LCD to write data<br>0 = LCD to read data |

- Por fim, é definido que se vai escrever sempre no LCD, colocando o valor '0' no sinal externo *ext_rw* e o valor do sinal interno do chip-select *cs* é copiado para o sinal externo correspondente *cs_lcd*.

**5)** Este é o processo responsável pela leitura dos valores dos caracteres a imprimir no LCD, que estão armazanedos na memória RAM, e respectivo envio para o LCD. Também se procede à inicialização do LCD por instrução quando o sinal de reset interno está activo.

**a)** Se o sinal de reset global *rst* for activado, então:
- o sinal de reset local *reset*, responsável pelo processo de inicialização do LCD, é também activado;
- coloca-se o LCD em modo de dados (sinal *rs* = '1');
- a instrução enviada, que é colocada no sinal interno *cmd,* não faz nada;
- o contador de comandos *cnt* é inicializado.

```
-- ler os dados da RAM e escrever no LCD
process(rst, clk_lcd) is
begin
   if rst = '0' then
        cmd <= x"00"; -- não há informação
        s_rs <= '1';
        reset <= '1'; -- inicializar o LCD
        cnt <= 0; -- inicializar o contador de comandos
```

**b)** Agora é necessário enviar os caracteres que estão armazenados na memória RAM para o LCD através do barramento de dados. Para cada ciclo de relógio local, efectuam-se as seguintes operações:
- na segunda fase de relógio, o contador de comandos é inicializado; este contador tem 34 valores, que correspondem aos 32 caracteres do display (16 caracteres por 2 linhas), mais duas instruções necessárias para mover o cursor para o início de cada linha.
- se o sinal *reset* estiver inactivo, procede-se ao processo de vizualização de dados no LCD:

**a)** *cnt* = 0: primeiro coloca-se o cursor na posição correspondente ao caracter mais à esquerda da primeira linha, através do envio da instrução x"80", que corresponde ao endereço DDRAM (Display Data RAM) da primeira posição da primeira linha. As instruções do tipo:

| RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | ADD | ADD | ADD | ADD | ADD | ADD | ADD |

estabelecem qual o endereço da posição da memória do display onde se pretende escrever, sendo os endereços os seguintes (informação proveniente do datasheet do controlador HD44780):

| Display position | 1 | 2 | 3 | 4 | 5 | ..... | 39 | 40 |
|---|---|---|---|---|---|---|---|---|
| DDRAM address (hexadecimal) | 00 | 01 | 02 | 03 | 04 | ..... | 26 | 27 |
| | 40 | 41 | 42 | 43 | 44 | ..... | 66 | 67 |

**b)** *cnt* = 1 até *cnt* = 16: os dados referentes às posições de memória 0 ("*add* = 00000") a 15 (*add* = "01111") da RAM (os 16 caracteres da primeira linha) são enviados pelo barramento de dados para o LCD, para serem visualizados; pela forma como o LCD é configurado na sequência inicial, após cada envio de dados, o cursor é "shiftado" automaticamente e, por isso, não há perigo de escrever dois caracteres na mesma posição.

**c)** *cnt* = 17: o cursor é mudado para o início da segunda linha, através do envio da instrução x"C0". Tendo em conta novamente que:

| RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | ADD | ADD | ADD | ADD | ADD | ADD | ADD |

De facto obtém-se [DB6:DB0] = "1 0 0 0 0 0 0" = x"40", que é o endereço da DDRAM para a primeira posição da segunda linha do display do LCD.

**d)** *cnt* = 18 até *cnt* = 33: os dados referentes à posições de memória 16 (*add* = "1 0000") a 31 (add = "1 1111") da RAM (os 16 caracteres da segunda linha) são enviados pelo barramento de dados para o LCD, para serem visualizados.

```
  elsif rising_edge(clk_lcd) then
     if idx = 2 then
        cnt <= cnt + 1; -- incrementa o contador de comandos
     end if;
     if (reset = '0') then
        case cnt is
           -- visualização de dados (16 caracteres) na 1ª linha do LCD
           -- inicializa o cursor
           when 0 => cmd <= x"80";
                     s_rs <= '0';
           -- dados
           when 1 to 16 => cmd <= LCD_RAM(conv_integer(cnt - 1));
                     s_rs <= '1';
           -- visualização de dados (16 caracteres) na 2ª linha do LCD
           -- instrução
           when 17 => cmd <= x"C0";
                     s_rs <= '0';
           -- dados
           when 18 to 33 => cmd <= LCD_RAM(conv_integer(cnt - 2));
                     s_rs <= '1';
          -- instrução
           when others => cmd <= x"80";
                     s_rs <= '0';
        end case;
```

**6)** Se o sinal *reset* estiver activo, então procede-se à inicialização do LCD:
- **Function Set**: instrução x"38", que coloca a interface como sendo dual-line (de 8 bits), seleciona o número de linhas como 2 linhas e escolhe o formato dos caracteres como sendo de 5x8 dots;
- depois é necessário esperar 37 us;
- **Display On/Off**: instrucção x"0C", que coloca o display a on, coloca o cursor a off e desactiva o blinking do cursor (ou seja, o cursor não irá piscar);
- esperar 37 us;
- **Entry Mode Set**: instrução x"06", que configura que após cada conjunto de dados recebidos o cursor sofre um shift para a direita (incremento do shift do cursor), mas que especifica que o shift do display é desactivado;
- esperar 37 us;
- **Clear Display**: instrução x"01", que limpa o display inteiro e coloca o endereço da DDRAM a 0 (primeira posição da primeira linha).

```
        else
           -- se o sinal de reset do LCD estiver activo
           case cnt is
              -- Function set (8-bit, dual-line)
              when 0 => cmd <= x"38";
                        s_rs <= '0';
              -- Display On/Off (display on, cursor oo, cursor blink off)
              when 48 => cmd <= x"0C";
                        s_rs <= '0';
              -- EntryMode set (increment on, display shift off)
              when 52 => cmd <= x"06";
                        s_rs <= '0';
              -- Limpar o display
              when 56 => cmd <= x"01";
                        s_rs <= '0';
              -- Inicializar cursor
              when 60 => cmd <= x"02";
                        s_rs <= '0';

              -- reset
              when 63 => reset <= '0';
                         cnt <= 0;
              when others => null;
           end case;
        end if;
     end if;

end process;

end Behavioral;
```