Grupo 5 - Turma A

Sexto laboratório de Circuitos Digitais

MC 613 - Primeiro Semestre de 2010

Professor: Guido Araújo

Henrique Serapião Gonzales	RA: 083636
Marcelo Galvão Póvoa	RA: 082115
Tiago Chedraoui Silva	RA: 082941

8 de junho de 2010

1 Questão 1

Um detector de sequência de *n-bits* é um circuito bastante comum e de utilidades diversas. A implementação teve entradas unárias assíncronas nos *push buttons*, portanto foi necessária a sincronização das mesmas (Algoritmo 4). Ambas simulações possuem as mesmas entradas e mostram a detecção. Geralmente, pode ser implementado de duas formas diferentes:

1.1 Registrador de Deslocamento

Esse método é o mais simples, uma vez que usa um registrador de deslocamento de *n-bits* com saída paralela. Se todas as posições da saída forem iguais ao padrão, a sequência foi detectada.

No entanto, há um detalhe importante a se considerar: a saída só pode ser $'1'$ se todos os n bits foram entrados (e corretamente). Logo, o registrador deve ser capaz de diferenciar nos bits de saída quais foram entrados pelo usuário e quais são de inicialização. Esse problema foi contornado usando um contador de bits entrados que não precisa contar além de n.

Essa implementação é totalmente genérica em relação ao padrão desejado.

1.2 Máquina de Estados

Usando uma máquina de estados, tem-se um comportamento mais analítico da detecção do padrão. Há *n+1* estados, sendo um para cada próximo bit esperado do padrão e o último relativo também ao primeiro bit mas indicando que a sequência acabou de ser encontrada.

A dificuldade desse método está em determinar, para cada estado, qual é próximo estado em caso de falha no bit lido (função conhecida como *failure function*), o que é feito manualmente. Em caso de acerto no bit lido, o próximo estado é $estado + 1$. Uma vantagem em relação ao registrador é que não é preciso contar quantos bits entraram.

Essa implementação não consegue ser totalmente genérica devido à função de falha, porém basta construí-la para o padrão desejado.

Listing 1: Componente do detector de padrão 1.1 em VHDL

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY pat_shift_det IS
5    GENERIC (
6      PAT_LEN: INTEGER := 8
7    );
8    PORT (clk, rstn, en: IN STD_LOGIC;
9          patt: IN STD_LOGIC_VECTOR(PAT_LEN-1 downto 0);
10         inp: IN STD_LOGIC;
11         shreg: OUT STD_LOGIC_VECTOR(PAT_LEN-1 downto 0);
12         ok: OUT STD_LOGIC);
13 END pat_shift_det;
14
15 ARCHITECTURE behav OF pat_shift_det IS
16   SIGNAL qb: INTEGER range 0 to PAT_LEN := 0;
17   SIGNAL x: STD_LOGIC_VECTOR(0 to PAT_LEN-1);
18   --dados lidos são deslocados em x
19 BEGIN
20   PROCESS (clk, rstn)
21   BEGIN
```

```
22      IF (rstn = '0') THEN
23        qb <= 0;
24        x  <= (others => '0');
25      ELSIF (rising_edge(clk)) THEN
26        --desloca os bits caso uma entrada esteja ativa
27        IF (en = '1') THEN
28          x(0 to PAT_LEN-2) <= x(1 to PAT_LEN-1);
29          x(PAT_LEN-1) <= inp;
30
31          IF (qb < PAT_LEN) THEN
32            qb <= qb + 1; --incr número de bits lidos
33          END IF;
34        END IF;
35      END IF;
36    END PROCESS;
37
38    shreg <= x;
39
40    ok <= '1' WHEN (x = PATT and qb = PAT_LEN)
41    ELSE '0';
42  END behav;
```

Listing 2: Top-level do detector 1.1 em VHDL

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY pat_shift IS
5    GENERIC (
6      PAT_LEN: INTEGER := 8
7    );
8    PORT (clk, rstn: IN STD_LOGIC;
9          pb0, pb1: IN STD_LOGIC;
10         shreg: OUT STD_LOGIC_VECTOR(PAT_LEN-1 downto 0);
11         ok: OUT STD_LOGIC);
12 END pat_shift;
13
14 ARCHITECTURE behav OF pat_shift IS
15   CONSTANT PATT:
16     STD_LOGIC_VECTOR(0 to PAT_LEN-1):="10110001";
17   SIGNAL zero, um: STD_LOGIC; --sinais síncronos
18
19   SIGNAL det_in, det_en: STD_LOGIC;
20 BEGIN
21   bf0: ENTITY WORK.buff
22     PORT MAP (clk, not pb0, zero);
23   bf1: ENTITY WORK.buff
24     PORT MAP (clk, not pb1, um);
25
26   --Converte sinais para entrada binária
27   det_in <= '1' WHEN um = '1'
28   ELSE '0';
29
30   det_en <= '1' WHEN (zero = '1' or um = '1')
31   ELSE '0';
32
33   det0: ENTITY WORK.pat_shift_det
```

```
34      PORT MAP (clk, rstn, det_en, PATT,
35                det_in, shreg, ok);
36  END behav;
```

Listing 3: Top-level do detector 1.2 em VHDL

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.numeric_std.all;
4
5   ENTITY pat_state IS
6     PORT (clk, rstn: IN STD_LOGIC;
7           pb0, pb1: IN STD_LOGIC;
8           cur_st: OUT STD_LOGIC_VECTOR(2 downto 0);
9           ok: OUT STD_LOGIC);
10  END pat_state;
11
12  ARCHITECTURE behav OF pat_state IS
13    CONSTANT PAT_LEN: INTEGER := 8;
14    SUBTYPE index is INTEGER range 0 to PAT_LEN-1;
15    TYPE int_array is ARRAY(0 to PAT_LEN-1) OF index;
16    CONSTANT PATT: --padrão desejado de busca
17       STD_LOGIC_VECTOR(0 to PAT_LEN-1) := "10110001";
18    CONSTANT FAIL: --próximo estado se a busca falhar em i
19                   --deve ser coerente com o padrão PATT
20       int_array := (0, 1, 0, 2, 1, 3, 1, 0);
21
22    SIGNAL zero, um: STD_LOGIC; --sinais síncronos
23    SIGNAL st_no, nxt_st: INTEGER range 0 to PAT_LEN:=0;
24    SIGNAL look_pos: index;
25    --estados 0 e PAT_LEN correspondem ao primeiro bit,
26    --mas no último a sequência acabou de ser encontrada
27    --look_pos é o índice real do próximo bit esperado
28
29    COMPONENT buff IS
30      PORT (clk, d: IN STD_LOGIC;
31            q: OUT STD_LOGIC);
32    END COMPONENT buff;
33  BEGIN
34    bf0: COMPONENT buff
35      PORT MAP (clk, not pb0, zero);
36    bf1: COMPONENT buff
37      PORT MAP (clk, not pb1, um);
38
39    PROCESS (clk, rstn, st_no, zero, um)
40    BEGIN
41      IF (rstn = '0') THEN
42        st_no <= 0;
43      ELSIF (rising_edge(clk)) THEN
44        --Transição da Máquina de Estados
45        IF (zero = '1' and PATT(look_pos) = '0') THEN
46          st_no <= nxt_st;
47        ELSIF (um = '1' and PATT(look_pos) = '1') THEN
48          st_no <= nxt_st;
49        ELSIF (zero = '1' or um = '1') THEN
50          st_no <= FAIL(look_pos);
51        END IF;
```

```
52      END IF;
53    END PROCESS;
54
55    --Saídas e sinais de controle da Máquina de Estados
56    --Calcula próximo estado se a entrada vier correta
57    PROCESS (st_no)
58    BEGIN
59      CASE st_no IS
60        WHEN 0 to PAT_LEN-1 =>
61          ok <= '0';
62          look_pos <= st_no;
63          nxt_st <= st_no + 1;
64        WHEN PAT_LEN =>
65          ok <= '1';
66          look_pos <= 0;
67          nxt_st <= 1;
68      END CASE;
69    END PROCESS;
70
71    cur_st <= std_logic_vector(to_unsigned(st_no, 3));
72  END behav;
```

Listing 4: Componente de amostragem de push button (usado em vários projetos) em VHDL

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY buff IS
5    PORT (clk, d: IN STD_LOGIC;
6          q: OUT STD_LOGIC);
7  END buff;
8
9  ARCHITECTURE rtl OF buff IS
10   TYPE state_type IS (st_wait, st_high);
11   SIGNAL state: state_type;
12 BEGIN
13   PROCESS (clk) BEGIN
14     IF (clk'event and clk = '1') THEN
15       CASE state IS
16         WHEN st_wait =>
17           IF (d = '1') THEN
18             state <= st_high;
19           ELSE
20             state <= st_wait;
21           END IF;
22         WHEN st_high =>
23           IF (d = '0') THEN
24             state <= st_wait;
25           ELSE
26             state <= st_high;
27           END IF;
28       END CASE;
29     END IF;
30   END PROCESS;
31
32   PROCESS (state, d) BEGIN
33     CASE state IS
```

```
34        WHEN st_wait =>
35          IF (d = '1') THEN
36            q <= '1';
37          ELSE
38            q <= '0';
39          END IF;
40        WHEN st_high =>
41          q <= '0';
42      END CASE;
43    END PROCESS;
44 END rtl;
```

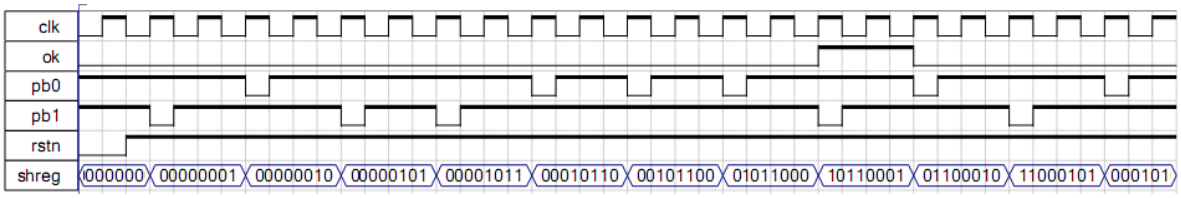

Figura 1.1: Simulação do detector de padrão 1.1

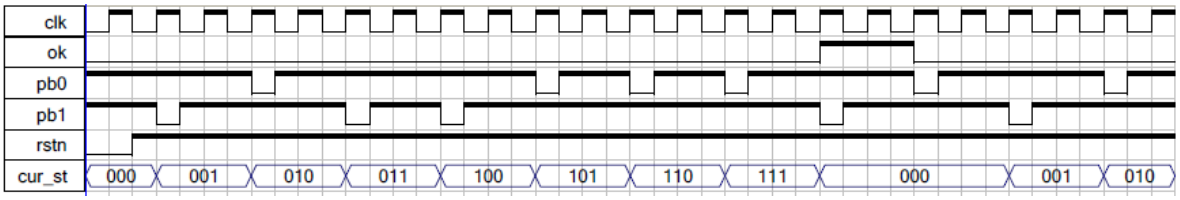

Figura 1.2: Simulação do detector de padrão 1.2

2 Questão 2

Um detector de palavras em hexadecimal (4 bits) pode ser facilmente construído a partir de quatro componentes detectores com registrador de deslocamento da Questão 1 (Alg. 1), cada um recebendo paralelamente um fluxo de bits para detectar. Como esses detectores são genéricos, basta extrair os padrões da palavra C0FFEE e enviar na entrada dos componentes.

A entrada do circuito é mais simples que a anterior: quatro bits são entrados por vez amostrados por um *push button*. As saídas correspondem a última letra lida no display de 7 segmentos e o led que indica detecção (AND lógico de todos os componentes). A simulação mostra sucintamente a ocorrência de detecção.

Listing 5: Top-level do detector de palavras em VHDL

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.numeric_std.all;
4
```

```
5  ENTITY word_detect IS
6    PORT (clk, rstn: IN STD_LOGIC;
7          x: IN STD_LOGIC_VECTOR(3 downto 0);
8          nxt: IN STD_LOGIC;
9          seg0: OUT STD_LOGIC_VECTOR(6 downto 0);
10         ok: OUT STD_LOGIC);
11 END word_detect;
12
13 ARCHITECTURE behav OF word_detect IS
14   CONSTANT W_LEN: NATURAL := 6;
15
16   SUBTYPE hexa IS STD_LOGIC_VECTOR(3 downto 0);
17   SUBTYPE stream IS
18     STD_LOGIC_VECTOR(W_LEN-1 downto 0);
19   TYPE hword IS ARRAY(3 downto 0) OF stream;
20                              --COFFEE--
21   CONSTANT WORDS: hword := ("101111",
22                             "101111",
23                             "001111",
24                             "001100");
25
26   SIGNAL sn_nxt: STD_LOGIC; --sinal síncrono
27   SIGNAL last_x: hexa := x"0"; --último número entrado
28
29   SIGNAL det_oks: STD_LOGIC_VECTOR(3 downto 0);
30
31   COMPONENT conv_7seg IS
32   PORT (x: IN STD_LOGIC_VECTOR(3 downto 0);
33         y: OUT STD_LOGIC_VECTOR(6 downto 0));
34     END COMPONENT conv_7seg;
35 BEGIN
36   bf0: ENTITY WORK.buff
37     PORT MAP (clk, not nxt, sn_nxt);
38
39   pdet0: ENTITY WORK.pat_shift_det GENERIC MAP (6)
40     PORT MAP (clk, rstn, sn_nxt, WORDS(0),
41                 x(0), ok => det_oks(0));
42   pdet1: ENTITY WORK.pat_shift_det GENERIC MAP (6)
43     PORT MAP (clk, rstn, sn_nxt, WORDS(1),
44                 x(1), ok => det_oks(1));
45   pdet2: ENTITY WORK.pat_shift_det GENERIC MAP (6)
46     PORT MAP (clk, rstn, sn_nxt, WORDS(2),
47                 x(2), ok => det_oks(2));
48   pdet3: ENTITY WORK.pat_shift_det GENERIC MAP (6)
49     PORT MAP (clk, rstn, sn_nxt, WORDS(3),
50                 x(3), ok => det_oks(3));
51
52   PROCESS (clk, rstn)
53   BEGIN
54     IF (rstn = '0') THEN
55       last_x <= x"0";
56     ELSIF (rising_edge(clk)) THEN
57             IF (sn_nxt = '1') THEN
58         last_x <= x;
59             END IF;
60     END IF;
61   END PROCESS;
62
```

```
63    s0: COMPONENT conv_7seg
64        PORT MAP (last_x, seg0);
65
66    ok <= '1' WHEN (det_oks = "1111")
67    ELSE '0';
68 END behav;
```

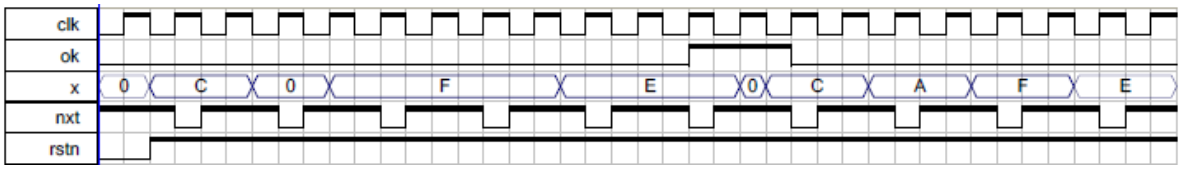

Figura 2.1: Simulação do detector de palavras

3 Questão 3

Utilizando as mega funções disponíveis para a FPGA, projetou-se, inicialmente, uma memória RAM de 64B com operações de escrita e leitura.RAM possui as seguintes pinagens:

- As linhas de endereço (6 bits para acesso a todas as posições dos bytes)
- Pinos de entradas e saídas de dados.
- E = Chip enable, possibilita utilização da RAM se valor é igual 1.
- W = write enable, possibilita escrita na ram se valor igual a 1.
- G = output enable, possibilita acessar dados lidos da RAM.

Vale a pena observar que se W= 1 e G = 1 representaria escrita e leitura simultâneas no mesmo endereço, logo, devido a inconsistência dessa ação, não é realizado nem escrita nem leitura.

Listing 6: Memória RAM de 64B em VHDL

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  ENTITY ram_64B IS
6    PORT
7    (
8      clk       :IN STD_LOGIC ;
9      input     : IN STD_LOGIC_VECTOR (7 DOWNTO 0);  -- dados entrada
10     output    : OUT STD_LOGIC_VECTOR (7 DOWNTO 0); -- saida
11     address   : IN STD_LOGIC_VECTOR (5 DOWNTO 0);  -- endereco
12     wren      : IN STD_LOGIC;                      -- W write-enable
13     chipen    : IN STD_LOGIC;                      -- E chip-enable
14     rden      : IN STD_LOGIC                       -- G read-enable
15   );
16 END ram_64B;
17
```

```
18 ARCHITECTURE behav OF ram_64B IS
19   SIGNAL address_sig : STD_LOGIC_VECTOR (5 DOWNTO 0);
20   SIGNAL clock_sig : STD_LOGIC ;
21   SIGNAL clken_sig : STD_LOGIC ;
22   SIGNAL data_sig: STD_LOGIC_VECTOR (7 DOWNTO 0);
23   SIGNAL wren_sig,rden_sig: STD_LOGIC;
24   SIGNAL q_sig : STD_LOGIC_VECTOR (7 DOWNTO 0);
25
26 COMPONENT ram1p IS
27   PORT
28   (
29     address   : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
30     clock   : IN STD_LOGIC  := '1';
31     data    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
32     wren    : IN STD_LOGIC ;
33     q   : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
34   );
35 END COMPONENT ram1p;
36
37 BEGIN
38   PROCESS(clk,wren,rden,chipen)
39    BEGIN
40       IF(chipen = '1') THEN
41         --write
42         IF(wren='1' and rden ='0') THEN
43           wren_sig <= '1';
44           rden_sig <= '0';
45         --read
46         ELSIF(wren='0' and rden ='1') THEN
47           wren_sig <= '0';
48           rden_sig <= '1';
49         --write and read -> nao
50         ELSE
51           wren_sig <= '0';
52           rden_sig <= '0';
53         END IF;
54       ELSE
55         wren_sig <= '0';
56         rden_sig <= '0';
57       END IF;
58   END PROCESS;
59
60   PROCESS(q_sig, chipen, rden_sig) BEGIN
61     IF(chipen = '1' and rden_sig = '1') THEN
62       output<=q_sig;
63     ELSE
64       output<= (others=>'0');
65     END IF;
66   END PROCESS;
67
68   ramp : ram1p PORT MAP (
69       address,clk,input,wren_sig,q_sig
70     );
71 END behav;
```

Posteriormente, é possível projetar um sistema de memória RAM com 256B através do uso de quatro módulos do componente anterior.

Utilizando as 10 toggle switches, carrega-se os 10 bits de endereço no primeiro ciclo (Led verde aceso), e no segundo ciclo(Led verde apagado) carrega-se os 8 bits de dados mais os bits de enable das operações de leitura e escrita.

Para a escolha da ram, utilizamos um multiplexador que avaliam os 4 bits mais significativos do endereço. Se a ram é escolhida, seu chip enable passa a valer 1, caso contrário,valerá 0.

Além disso, os displays de 7 segmentos foram usados para apresentarem o resultado da operação da leitura da ram. E adicionamos um push button para a mudança de estados e a confirmação de execução dos dados.

Listing 7: Memória RAM de 256B em VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.numeric_std.all;

ENTITY ram4x64B IS
  PORT
  (
    clk       : IN STD_LOGIC;
    input     : IN STD_LOGIC_VECTOR (7 DOWNTO 0);     -- dados entrada
    output    : BUFFER STD_LOGIC_VECTOR (7 DOWNTO 0); -- saida
    wren      : IN STD_LOGIC;                         -- W write-enable
    rden      : IN STD_LOGIC;
    chmod     : IN STD_LOGIC;
    modled    : OUT STD_LOGIC;
        seg1, seg0: OUT STD_LOGIC_VECTOR(6 downto 0)     --sete segmentos
  );
END ram4x64B;

ARCHITECTURE behav OF ram4x64B IS
  SIGNAL sel: STD_LOGIC_VECTOR (3 DOWNTO 0);
  SIGNAL  q_sig,q_sig1,q_sig2,q_sig3,q_sig4: STD_LOGIC_VECTOR (7 DOWNTO 0); -- saida

  COMPONENT conv_7seg IS
  PORT (x: IN STD_LOGIC_VECTOR(3 downto 0);
        y: OUT STD_LOGIC_VECTOR(6 downto 0));
  END COMPONENT conv_7seg;

  COMPONENT ram_64B IS
  PORT
  (
    clk       : IN STD_LOGIC ;
    input     : IN STD_LOGIC_VECTOR (7 DOWNTO 0);  -- dados entrada
    output    : OUT STD_LOGIC_VECTOR (7 DOWNTO 0); -- saida
    address   : IN STD_LOGIC_VECTOR (5 DOWNTO 0);  -- endereco
    wren      : IN STD_LOGIC;                      -- W write-enable
    chipen    : IN STD_LOGIC;                      -- E chip-enable
    rden      : IN STD_LOGIC                       -- G read-enable
  );
  END COMPONENT ram_64B;

  TYPE state_type IS ( wraddress,wdata);
  SIGNAL state : state_type := wraddress;
  SIGNAL data_sig,address_sig: STD_LOGIC_VECTOR (7 DOWNTO 0):="00000000";
  SIGNAL chmod_sig,wren_sig,rden_sig: STD_LOGIC :='0'; --modo de  amostrado

  COMPONENT buff IS
```

```
47   PORT (clk, d: IN STD_LOGIC;
48           q: OUT STD_LOGIC);
49   END COMPONENT buff;
50
51  BEGIN
52   bf: COMPONENT buff
53     PORT MAP (clk, chmod,chmod_sig);
54
55   -- Transicoes da Maquina de Estados
56   PROCESS (state, clk)
57   BEGIN
58     IF (rising_edge(clk)) THEN
59       CASE state IS
60         WHEN  wraddress => --leitura de enderço
61           address_sig <= input;
62           modled<='1';
63           IF(chmod_sig='1') THEN
64             state <=wdata;
65           END IF;
66         WHEN wdata =>
67           data_sig<=input;
68           modled<='0';
69           IF(chmod_sig='1') THEN
70             state <= wraddress;
71           END IF;
72       END CASE;
73     END IF;
74   END PROCESS;
75
76   PROCESS(address_sig)
77     variable ram_sel:STD_LOGIC_VECTOR (1 downto 0);
78   BEGIN
79     ram_sel:=address_sig(7 downto 6);
80
81     case ram_sel is
82       when "00" =>
83         sel<="0001";
84       when "01" =>
85         sel<="0010";
86       when "10" =>
87         sel<="0100";
88     when others =>
89       sel<="1000";
90     end case;
91   END PROCESS;
92
93   ram1: ram_64B PORT MAP (
94     clk,data_sig,q_sig1,address_sig(5 downto 0),wren,sel(0),not wren);
95
96   ram2: ram_64B PORT MAP (
97     clk,data_sig,q_sig2,address_sig(5 downto 0),wren,sel(1),not wren);
98
99   ram3: ram_64B PORT MAP (
100    clk,data_sig,q_sig3,address_sig(5 downto 0),wren,sel(2),not wren);
101
102  ram4: ram_64B PORT MAP (
103    clk,data_sig,q_sig4,address_sig(5 downto 0),wren,sel(3),not wren);
104
```

```
105   PROCESS(clk)
106   BEGIN
107     IF (clk'event and clk = '1') THEN
108       IF(rden ='1') THEN
109       case sel is
110         when "0001" =>
111           output<=q_sig1;
112         when "0010"=>
113           output<=q_sig2;
114         when "0100" =>
115           output<=q_sig3;
116         when others =>
117           output<=q_sig4;
118       end case;
119       END IF;
120     END IF;
121   END PROCESS;
122
123   --leitura da memoria RAM
124   cseg0: COMPONENT conv_7seg
125     PORT MAP (output(3 downto 0), seg0);
126   cseg1: COMPONENT conv_7seg
127     PORT MAP (output(7 downto 4), seg1);
128 END behav;
```

4 Questão 4

A via de dados especificada é bastante simples, mas representa a decodificação das instruções e uso de barramento de uma CPU. Basicamente, há uma série de multiplexadores envolvidos: *(i)* escolha da operação, *(ii)* escolha do registrador de origem/destino, *(iii)* uso do barramento, etc.

Com uma implementação comportamental em VHDL é fácil fazer essa lógica usando índices de um vetor de registradores (incluindo um para a saída). Note que a saída é a *amostra* do valor do registrador quando ocorre a operação *10*.

A demonstração é feita com *clock* manual, entrada paralela (*4-bits*) e saída em 7-segmentos. A simulação mostra alguns exemplos do funcionamento de cada *OpCode*.

Listing 8: Top-level da via de dados em VHDL

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  ENTITY via_dados IS
6    PORT (clk, rstn: IN STD_LOGIC;
7          opcode: IN STD_LOGIC_VECTOR(5 downto 0);
8          inp: IN STD_LOGIC_VECTOR(3 downto 0);
9          outp: OUT STD_LOGIC_VECTOR(3 downto 0);
10         seg0: OUT STD_LOGIC_VECTOR(6 downto 0));
11 END via_dados;
12
13 ARCHITECTURE behav OF via_dados IS
14   SUBTYPE reg IS STD_LOGIC_VECTOR(3 downto 0);
15   TYPE vreg IS ARRAY(0 to 3) OF reg;
```

```
16   SUBTYPE reg_index IS NATURAL range 0 to 3;
17
18   SIGNAL regs: vreg;
19   SIGNAL opr: STD_LOGIC_VECTOR(1 downto 0);
20   SIGNAL rd, rs: reg_index;
21   SIGNAL reg_disp: reg;
22
23   COMPONENT conv_7seg IS
24   PORT (x: IN STD_LOGIC_VECTOR(3 downto 0);
25         y: OUT STD_LOGIC_VECTOR(6 downto 0));
26   END COMPONENT conv_7seg;
27 BEGIN
28   --decodifica parâmetros de controle
29   opr <= opcode(5 downto 4);
30   rd <= to_integer(unsigned( opcode(3 downto 2) ));
31   rs <= to_integer(unsigned( opcode(1 downto 0) ));
32
33   PROCESS (clk, rstn)
34   BEGIN
35     IF (rstn = '0') THEN
36       regs <= (others => x"0");
37       reg_disp <= x"0";
38     ELSIF (rising_edge(clk)) THEN
39       CASE opr IS
40         WHEN "00" =>
41           regs(rd) <= regs(rs);
42         WHEN "01" =>
43           regs(rd) <= inp;
44         WHEN "10" =>
45           --memoriza o conteudo atual do registrador
46           --para mostrar no display
47           reg_disp <= regs(rs);
48         WHEN others =>
49
50       END CASE;
51     END IF;
52   END PROCESS;
53
54   s0: COMPONENT conv_7seg
55   PORT MAP (reg_disp, seg0);
56   outp <= reg_disp;
57 END behav;
```

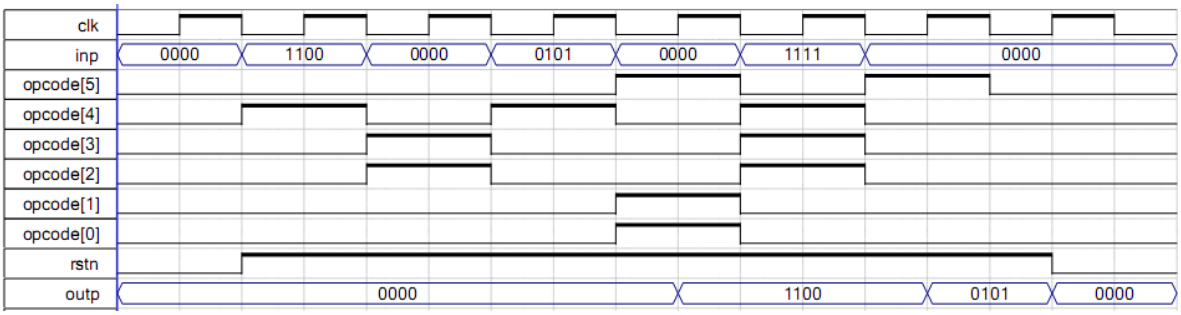

Figura 4.1: Simulação da via de dados

5 Questão 5

Utilizando o módulo vgacon e o arquivo de exemplo que contém uma demostração de uma bola percorrendo o espaço visível e quicando nas laterais do monitor , realizamos alterações no código de modo que a bola deixasse um rastro pelo caminho percorrido. Para isso, existiu a criação de uma variável que não permitiria a escrita em posições anteriores percorridas pela bola.

Outros processos foram desenvolvidos , dentre eles:

- Atualiza_cor: altera a cor da bola sempre que ela mudasse de direção, ou seja, sempre que ela atingisse os cantos da tela. Vale observar, que ao ocorrem mudanças simultâneas na direção vertical e horizontal, a cor deveria mudar somente uma vez, para isso adicionamos as condições nas linhas 35 e 40 do código abaixo.

- Reset : criou-se um novo estado (apaga_quadro - ver linha 83 do código abaixo), que retorna a bola ao pixel na posição (0,0), ponto superior esquerdo da tela, e apaga os rastros na tela.

Além disso, o arquivo fornecido no formato MIF - usado para especificar conteúdo inicial da RAM - foi alterado para a obtenção de uma tela preta, assim, todos os campos do arquivo passaram a ter um valor 0.

Listing 9: Arquivo responsável pelo comportamento da bola

```
1   --codigo omitido
2   SIGNAL cor : UNSIGNED(2 downto 0); --cor atual da bola
3   SIGNAL custom_we: STD_LOGIC; --não escrita em células antigas permite
4                                --exibir o rastro da bola
5   ------------------------------------------------------------------------
6   -- Brilho do pixel
7   ------------------------------------------------------------------------
8   -- O brilho do pixel é branco quando os contadores de linha e coluna, que
9   -- indicam o endereço do pixel sendo escrito para o quadro atual, casam com a
10  -- posição da bola (sinais pos_x e pos_y). Caso contrário,
11  -- o pixel é preto.
12
13  atualiza_cor: PROCESS (clk27M, rstn, cor)
14    type direcao_t_x is (direita, esquerda);
15    type direcao_t_y is (desce,cima);
16    variable direcao_x : direcao_t_x := direita;
```

```
17      variable direcao_y : direcao_t_y := desce;
18
19    VARIABLE nxt_cor: UNSIGNED(2 downto 0);
20    BEGIN
21    IF (cor = "111") THEN
22      nxt_cor := "001";
23    ELSE
24        nxt_cor := cor + "001";
25    END IF;
26
27      IF (rstn = '0') THEN
28         cor <= "001";
29         direcao_y := desce;
30         direcao_x := direita;
31      ELSIF (clk27M'event and clk27M = '1') THEN
32      IF (atualiza_pos_y = '1' ) THEN
33        IF (pos_y = 0 and direcao_y = cima) THEN
34          direcao_y := desce;
35          IF(pos_x /= 127 and pos_x /= 0)THEN
36            cor <= nxt_cor;
37          END IF;
38        ELSIF (pos_y = 95 and direcao_y = desce) THEN
39          direcao_y := cima;
40          IF(pos_x /= 127 and pos_x /= 0)THEN
41            cor <= nxt_cor;
42          END IF;
43        END IF;
44      END IF;
45
46      IF (atualiza_pos_x = '1') THEN
47        IF (pos_x = 0 and direcao_x = esquerda ) THEN
48          cor <= nxt_cor;
49          direcao_x:=direita;
50        ELSIF (pos_x = 127 and direcao_x = direita) THEN
51          cor <= nxt_cor;
52          direcao_x := esquerda;
53        END IF;
54      END IF;
55    END IF;
56    END PROCESS;
57
58    pixel <= "000" WHEN (estado = apaga_quadro)
59    ELSE std_logic_vector(cor) WHEN (col = pos_x) and (line = pos_y)
60    ELSE "000";
61
62    custom_we <= '1' WHEN (col = pos_x) and (line = pos_y)
63             else '0';
64
65    -- O endereço de memória pode ser construído com essa fórmula simples,
66    -- a partir da linha e coluna atual
67    addr  <= col + (128 * line);
68
69    ---------------------------------------------------------------------------
70    -- Processos que definem a FSM (finite state machine), nossa máquina
71    -- de estados de controle.
72    ---------------------------------------------------------------------------
73
74    -- purpose: Esta é a lógica combinacional que calcula sinais de saída a partir
```

```
75  --          do estado atual e alguns sinais de entrada (Máquina de Mealy).
76  -- type   : combinational
77  -- inputs : estado, fim_escrita, timer
78  -- outputs: proximo_estado, atualiza_pos_x, atualiza_pos_y, line_rstn,
79  --          line_enable, col_rstn, col_enable, we, timer_enable, timer_rstn
80  logica_mealy: process (estado, fim_escrita, timer, custom_we)
81  begin  -- process logica_mealy
82    case estado is
83      when apaga_quadro => if fim_escrita = '1' then
84                             proximo_estado <= inicio;
85                           else
86                             proximo_estado <= apaga_quadro;
87                           end if;
88                           atualiza_pos_x <= '0';
89                           atualiza_pos_y <= '0';
90                           line_rstn      <= '1';
91                           line_enable    <= '1';
92                           col_rstn       <= '1';
93                           col_enable     <= '1';
94                           we             <= '1';
95                           timer_rstn     <= '0';
96                           timer_enable   <= '0';
97
98
99  --codigo omitido
```

Conclusão