



EESC - USP[®]

Prática N°10 - SEL0606

Alunos (Bancada 5):
Vitor Alexandre Garcia Vaz - 14611432
Gabriel Dezejácomo Maruschi - 14571525

Sumário

1	Introdução	3
1.1	<i>Objetivos</i>	3
1.2	<i>Banco de Registradores</i>	3
2	Materiais e métodos	3
2.1	<i>Interface</i>	4
2.2	<i>Banco de Registradores</i>	6
3	Conclusão	7
3.1	<i>Círculo RTL</i>	7
3.2	<i>Número de células lógicas</i>	8
3.3	<i>Funcionamento do banco de registradores na FPGA</i>	8

Lista de Imagens

1	Esquemático do Banco de Registradores	3
2	Kit DE10-LITE	4
3	Código da interface	5
4	Código do banco de registradores (parte 1)	6
5	Código do banco de registradores (parte 2)	7
6	Visualização do circuito referente à interface	7
7	Visualização do circuito referente ao banco de registradores	8
8	Resumo de funcionamento	8
9	Estágio 1 de funcionamento da FPGA	9
10	Estágio 2 de funcionamento da FPGA	9
11	Estágio 3 de funcionamento da FPGA	10
12	Estágio 4 de funcionamento da FPGA	10
13	Estágio 5 de funcionamento da FPGA	11
14	Estágio 6 de funcionamento da FPGA	11
15	Estágio 7 de funcionamento da FPGA	12
16	Estágio 8 de funcionamento da FPGA	12

1 Introdução

1.1 Objetivos

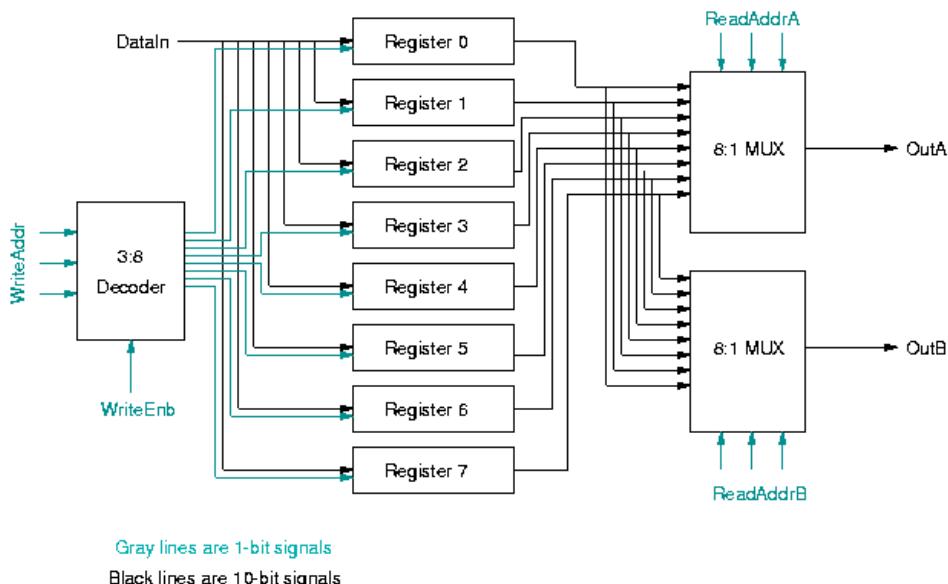
Nesta prática do Laboratório de Sistemas Digitais, implementamos um banco de registradores parametrizável, com clock e enable, utilizando a linguagem VHDL no software Quartus, e testamos no kit DE10-LITE (MAX 10 10M50DAF484C7G) bem como no software ModelSim.

Para isso, utilizamos a associação paralela de registradores conforme a fig.1, em conjunto com uma lógica combinacional para selecionar os registradores de escrita e leitura, utilizando multiplexadores.

1.2 Banco de Registradores

Em geral, bancos de registradores são utilizados dentro de processadores para armazenar dados que estão sendo utilizados no processo de funcionamento da CPU. Possuem capacidade limitada, volatilidade e rápido acesso.

Figure 1: Esquemático do Banco de Registradores



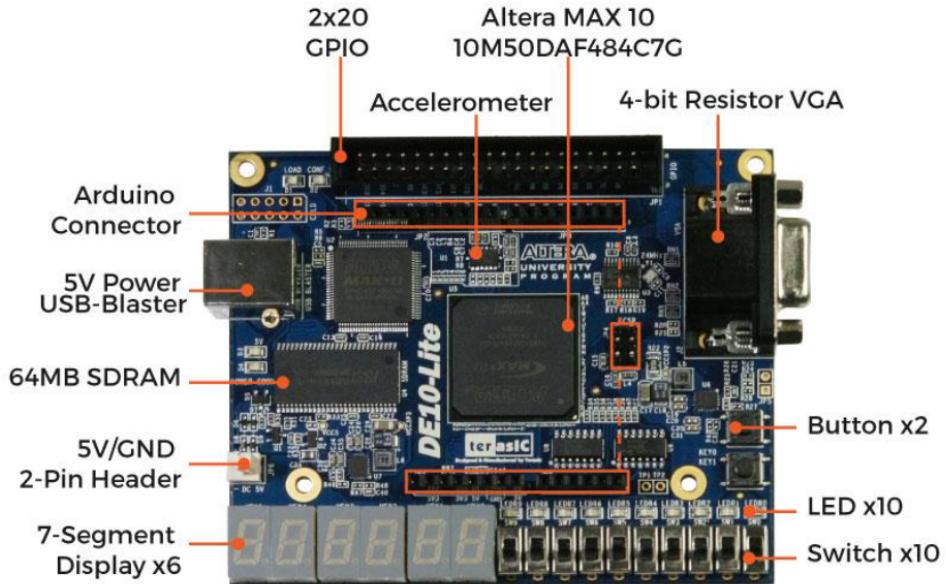
Fonte: [Site da internet](#)

Os sinal *Write Enable* seleciona, por meio do decodificador, os endereços (gerados a partir das entradas *Write Address*) nos quais haverá a escrita do dado *Data In*. Por outro lado, os sinais *Read Address A* e *B* operam em dois multiplexadores a fim de ler os valores de dois registrados simultaneamente.

2 Materiais e métodos

O código utilizado para desenvolver os módulos dos registradores e do Banco de registradores foi escrito em VHDL e compilado no Quartus. A fim de testá-los no DE10-LITE, desenvolvemos também um interface cujo objetivo foi redirecionar os sinais dos pinos para os módulos e vice-versa. Utilizamos também o software ModelSim da Intel para visualizar as ondas de entrada e saída do Contador binário desenvolvido.

Figure 2: Kit DE10-LITE



Fonte: DE10-Lite User Manual

2.1 Interface

A interface, codificada como é mostrado abaixo, direciona as chaves para as entradas do módulo do banco de registradores e as saídas para um sinal auxiliar. Este sinal é exibido nos LEDs e também é redirecionado como entrada do módulo do display de binário para 7 segmentos, um decodificador desenvolvido em práticas passadas.

Figure 3: Código da interface

```
1 -- Version: 1.1
2 -- Date: --/11/2024
3 -- Owners: Gabriel D. Maruschi
4 -- Vitor Alexandre Garcia Vaz
5
6 LIBRARY ieee;
7 USE ieee.std_logic_1164.ALL;
8 USE ieee.std_logic_arith.ALL;
9 USE ieee.std_logic_unsigned.ALL;
10
11 ENTITY DE10_LITE_RegFile IS
12     PORT (
13         SW : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
14         KEY : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
15         LEDR : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
16         HEX0 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
17         HEX1 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
18     );
19 END DE10_LITE_RegFile;
20
21
22 ARCHITECTURE estrutural OF DE10_LITE_RegFile IS
23
24     signal aux1 : STD_LOGIC_VECTOR (3 DOWNTO 0);
25     signal aux2 : STD_LOGIC_VECTOR (3 DOWNTO 0);
26
27 BEGIN
28     RegBank: entity work.regbank
29
30         GENERIC MAP (
31             M => 4,
32             X => 4
33         )
34
35         PORT MAP (
36             CLK => not KEY(0),
37             EN => '1',
38             WE3 => not KEY(1),
39             WD3 => SW(9 DOWNTO 6),
40             A1 => SW(1 DOWNTO 0),
41             A2 => SW(3 DOWNTO 2),
42             A3 => SW(5 DOWNTO 4),
43             RD1 => aux1,
44             RD2 => aux2
45         );
46
47     display1: entity work.hex27seg
48         PORT MAP (
49             hexa => aux1,
50             segments => HEX0
51         );
52
53     display2: entity work.hex27seg
54         PORT MAP (
55             hexa => aux2,
56             segments => HEX1
57         );
58
59     LEDR(3 DOWNTO 0) <= aux1;
60     LEDR(7 DOWNTO 4) <= aux2;
61
62 END estrutural;
```

Fonte: os autores

Para a chamada do módulo do banco, a entrada de Clock, **clk** foi direcionado para o botão 0 A saída pôde ser visualizada nos LEDs do kit e no display HEX0. Vale ressaltar o **GENERIC MAP** especificado na chamada do banco de registradores. Nele, definimos o tamanho e número de registradores ao fornecer o valor de M e X.

2.2 Banco de Registradores

O Banco de Registradores foi desenvolvido em VHDL e utiliza a estrutura de loop *FOR GENERATE* para chamadas múltiplas de registradores. Além disso, há a utilização do decodificador para geração dos endereços a partir dos sinais *Write Address* e de multiplexadores, cuja função já foi explicada.

Figure 4: Código do banco de registradores (parte 1)



```

1 -- Version: 1.1
2 -- Date: --/11/2024
3 -- Owners: Gabriel D. Maruschi
4 --           Vitor Alexandre Garcia Vaz
5
6 LIBRARY ieee;
7 USE ieee.std_logic_1164.ALL; -- tipos de dados lógicos std_logic e std_logic_vector
8 USE ieee.std_logic_arith.ALL; -- operações aritméticas com std_logic_vector
9 USE ieee.std_logic_unsigned.ALL; -- operações aritméticas com std_logic_vector como unsigned
10
11 ENTITY regbank IS
12   GENERIC (
13     M : INTEGER := 4; -- Número de registradores
14     X : INTEGER := 4 -- Largura de cada registrador
15   );
16   PORT (
17     CLK : IN std_logic; -- Clock
18     EN : IN std_logic; -- Habilita o banco de registradores
19     WE3 : IN std_logic; -- Habilita a escrita
20     WD3 : IN std_logic_vector(X-1 DOWNTO 0); -- Dados a serem escritos
21     A1 : IN std_logic_vector(1 DOWNTO 0); -- Seleção do registrador 1 para leitura
22     A2 : IN std_logic_vector(1 DOWNTO 0); -- Seleção do registrador 2 para leitura
23     A3 : IN std_logic_vector(1 DOWNTO 0); -- Seleção do registrador para escrita
24     RD1 : OUT std_logic_vector(X-1 DOWNTO 0); -- Saída do registrador 1
25     RD2 : OUT std_logic_vector(X-1 DOWNTO 0) -- Saída do registrador 2
26   );
27 END regbank;
28
29 ARCHITECTURE estrutural OF regbank IS
30
31   -- Declaração de tipos
32   TYPE reg_array IS ARRAY (0 TO M-1) OF std_logic_vector(X-1 DOWNTO 0);
33
34   -- Declaração de sinais internos
35   signal outDecoder : std_logic_vector(M-1 DOWNTO 0); -- Saída do decodificador
36   signal regEscrita : std_logic_vector(M-1 DOWNTO 0); -- Habilita a escrita em cada registrador
37   signal outRegister : reg_array; -- Saída de cada registrador
38   signal outMux1 : std_logic_vector(X-1 DOWNTO 0); -- Saída do multiplexador 1
39   signal outMux2 : std_logic_vector(X-1 DOWNTO 0); -- Saída do multiplexador 2
40 BEGIN
41
42   -- Decoder 1 : M
43   decoder: entity work.decode24
44     PORT MAP(
45       ENT => A3,
46       OUTPUT0 => outDecoder(0),
47       OUTPUT1 => outDecoder(1),
48       OUTPUT2 => outDecoder(2),
49       OUTPUT3 => outDecoder(3)
50     );

```

Fonte: os autores

Figure 5: Código do banco de registradores (parte 2)

```

52      -- Banco de M registradores
53      registers: FOR i in 0 to M-1 generate
54          regEscrita(i) <= outDecoder(i) AND WE3;
55          reg : entity work.Reg
56              PORT MAP(
57                  clk => CLK AND EN,
58                  en => regEscrita(i),
59                  cr => '0',
60                  D => WD3,
61                  Q => outRegister(i)
62              );
63      END generate registers;
64
65      -- Multiplexadores M : 1
66      mux1: entity work.mux41
67          PORT MAP(
68              INPUT0 => outRegister(0),
69              INPUT1 => outRegister(1),
70              INPUT2 => outRegister(2),
71              INPUT3 => outRegister(3),
72              S => A1,
73              R => outMux1
74          );
75
76      mux2: entity work.mux41
77          PORT MAP(
78              INPUT0 => outRegister(0),
79              INPUT1 => outRegister(1),
80              INPUT2 => outRegister(2),
81              INPUT3 => outRegister(3),
82              S => A2,
83              R => outMux2
84          );
85
86      -- Conectando as saídas dos multiplexadores às saídas do banco de registradores
87      RD1 <= outMux1;
88      RD2 <= outMux2;
89  END estrutural;

```

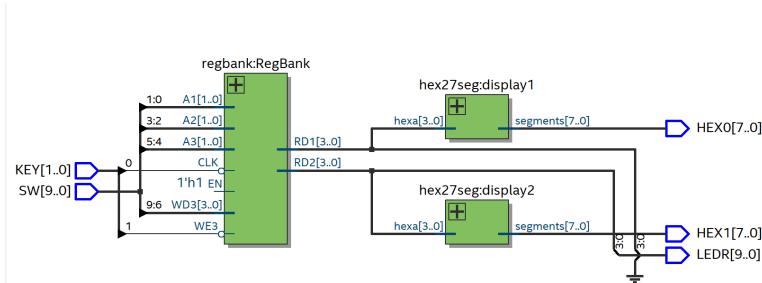
Fonte: os autores

3 Conclusão

3.1 Circuito RTL

A partir do código em VHDL feito em laboratório, e usando a ferramenta de síntese disponibilizada pelo quartus, o seguinte circuito, referente à interface entre o banco de registradores e os pinos da FPGA, foi obtido:

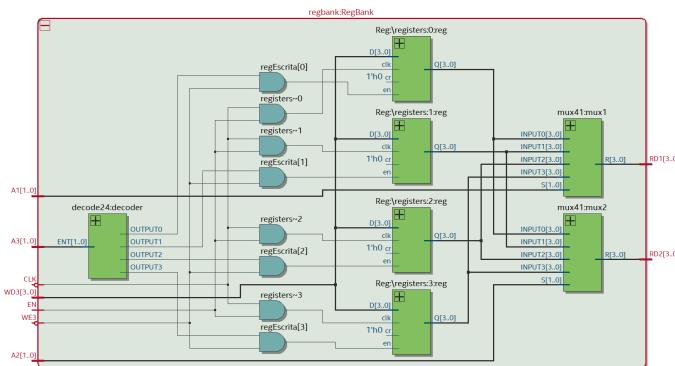
Figure 6: Visualização do circuito referente à interface



Fonte: os autores

E para o banco de registradores em si, obtemos o seguinte circuito:

Figure 7: Visualização do circuito referente ao banco de registradores



Fonte: os autores

3.2 Número de células lógicas

Ademais, por meio do resumo do funcionamento e constituição do circuito descrito em VHDL no software (fig8), chegamos à conclusão de que o circuito sintetizado apresentou um uso de 35 elementos lógicos e 38 pinos do dispositivo reconfigurável (FPGA).

Figure 8: Resumo de funcionamento

Compilation Report - DE10_LITE_RegFile	
Flow Summary	
<<Filter>>	
Flow Status	Successful - Tue Dec 3 15:29:08 2024
Quartus Prime Version	23.1std.0 Build 991 11/28/2023 SC Lite Edition
Revision Name	DE10_LITE_RegFile
Top-level Entity Name	DE10_LITE_RegFile
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	35 / 49,760 (< 1 %)
Total registers	16
Total pins	38 / 360 (11 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

Fonte: os autores

3.3 Funcionamento do banco de registradores na FPGA

O teste do funcionamento do circuito foi feito através da FPGA e, para isso, setamos as chaves SW(3:2) e SW(1:0) em 01₂ e 00₂, respectivamente, para leitura dos registradores 1 e 0, cujos conteúdos serão visualizados tanto através dos segundo e primeiro display quanto nos leds LEDR(7:4) e LEDR(3:0).

TESTE 1

No primeiro teste selecionamos o posicionamento das chaves SW(9:6) (referente ao dado de entrada) em 1111_2 e SW(5:4) (de seleção do registrador de escrita) em 00_2 para escrevermos o dado no registrador 0. A priori, o resultado foi de 0000_2 na saída dos registradores 1 e 0, conforme a fig.9, já que ainda não tínhamos ativado os sinais de clock e enable.

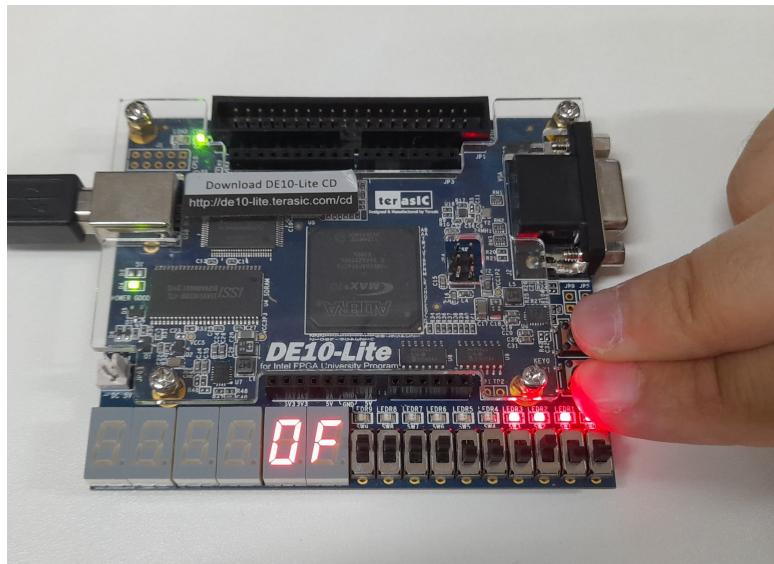
Figure 9: Estágio 1 de funcionamento da FPGA



Fonte: os autores

Porém, após a ativação dos sinais de clock e enable o conteúdo foi escrito no registrador 0, e a saída no primeiro display passou a corresponder com o conteúdo escrito, conforme a fig.10.

Figure 10: Estágio 2 de funcionamento da FPGA



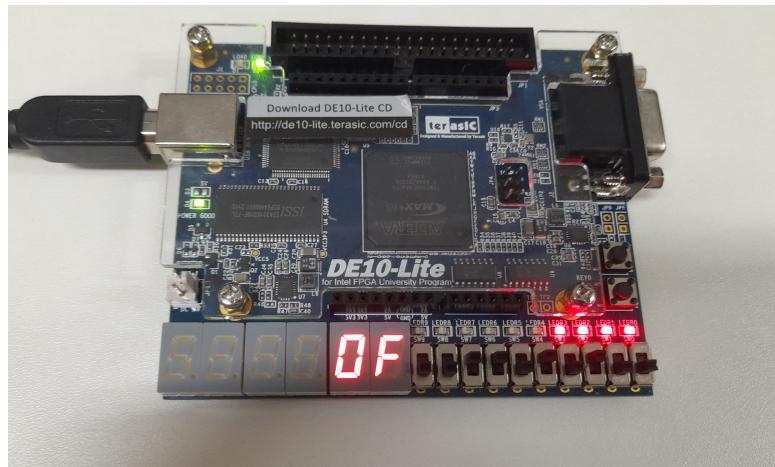
Fonte: os autores

TESTE 2

Já no segundo teste, selecionamos o posicionamento das chaves SW(9:6) (referente ao dado de entrada) em 0011_2 e SW(5:4) (de seleção do registrador de escrita) em 01_2 para escrevermos o dado no registrador 1. A priori, o resultado foi de 0000_2 na saída do registrador 1 conforme a fig.11, já que ainda não tínhamos ativado

os sinais de clock e enable.

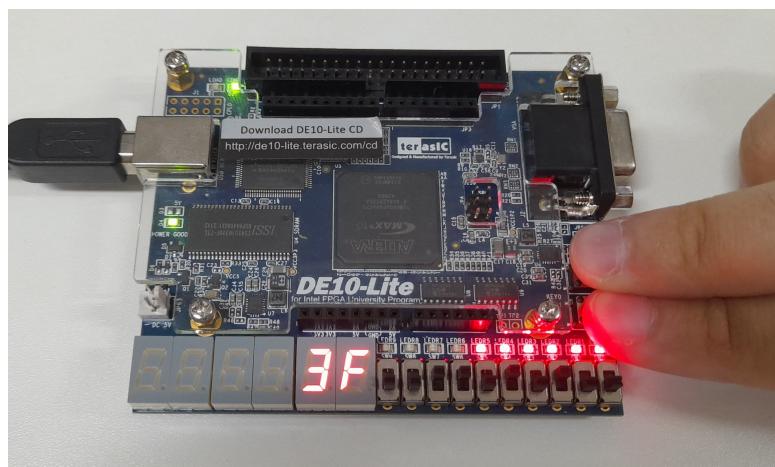
Figure 11: Estágio 3 de funcionamento da FPGA



Fonte: os autores

Porém, após a ativação dos sinais de clock e enable o conteúdo foi escrito no registrador 1, e a saída no segundo display passou a corresponder com o conteúdo escrito, conforme a fig.12.

Figure 12: Estágio 4 de funcionamento da FPGA

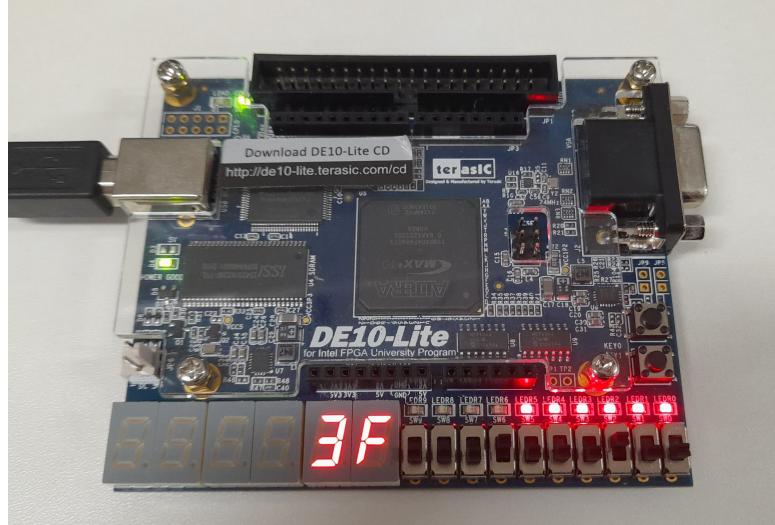


Fonte: os autores

TESTE 3

Para o terceiro teste, selecionamos o posicionamento das chaves SW(9:6) (referente ao dado de entrada) em 0001_2 e SW(5:4) (de seleção do registrador de escrita) em 00_2 para escrevermos o dado no registrador 0. A priori, o resultado foi o mesmo do teste 1, 1111_2 na saída do registrador 0, conforme a fig.13, já que ainda não tínhamos ativado os sinais de clock e enable.

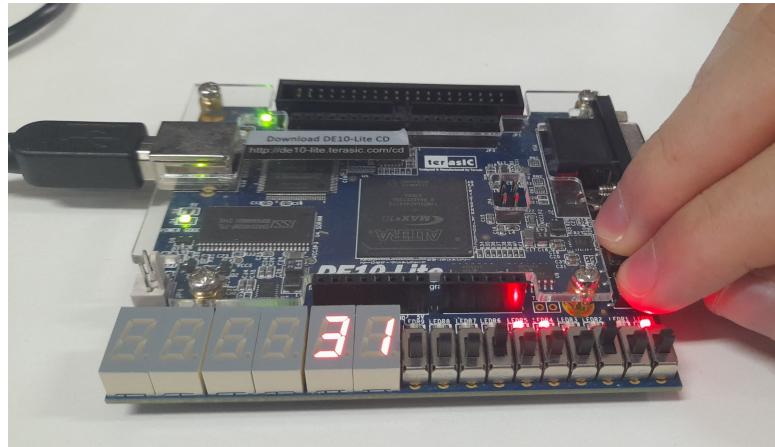
Figure 13: Estágio 5 de funcionamento da FPGA



Fonte: os autores

Porém, após a ativação dos sinais de clock e enable o novo conteúdo foi escrito no registrador 0, e a saída no primeiro display passou a corresponder com o conteúdo escrito, conforme a fig.14.

Figure 14: Estágio 6 de funcionamento da FPGA

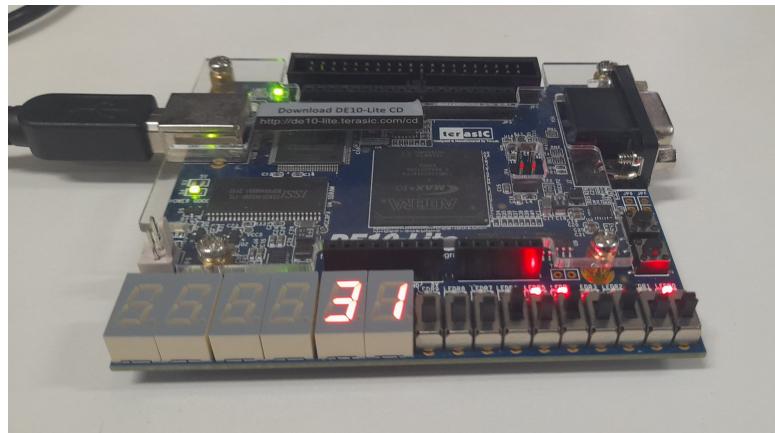


Fonte: os autores

TESTE 4

Para o quarto e último teste, selecionamos o posicionamento das chaves SW(9:6) (referente ao dado de entrada) em 0001_2 e SW(5:4) (de seleção do registrador de escrita) em 0001_2 para escrevermos o dado no registrador 1. A priori, o resultado foi o mesmo do teste 2, 0011_2 na saída do registrador 1, conforme a fig.15, já que ainda não tínhamos ativado os sinais de clock e enable .

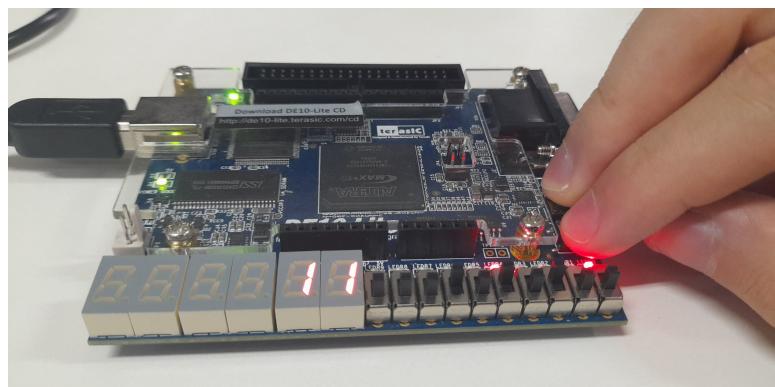
Figure 15: Estágio 7 de funcionamento da FPGA



Fonte: os autores

Porém, após a ativação dos sinais de clock e enable o conteúdo foi escrito no registrador 1, e a saída no primeiro display passou a corresponder com o conteúdo escrito, conforme a fig.16.

Figure 16: Estágio 8 de funcionamento da FPGA



Fonte: os autores