UNIDADE LÓGICA ARITMÉTICA EM VHDL

Mathaus C. Huber, Eduardo Marques

[1]Universidade Federal de Pelotas (UFPEL) – Discentes do Curso Superior de Ciência da Computação
R. Gomes Carneiro, 1 - Centro – 96075-630 – Pelotas – RS – Brazil

mchuber@inf.ufpel.edu.br, edsmarques@inf.ufpel.edu.br

Abstract. *This article describes the first work of the digital techniques discipline, given to the second semester of the Computer Science course at the Federal University of Pelotas, which refers to the creation of an architectural project, vhdl description, and prototyping of a logical unit. arithmetic, having as its final objective the implementation of some functions, they are: addition, subtraction, multiplication, division, or, and, not a, not b.*

Resumo. *Este artigo descreve o primeiro trabalho da disciplina de técnicas digitais, conferida ao segundo semestre do curso de Ciência da Computação, da Universidade Federal de Pelotas, no qual se refere a criação de um projeto arquitetural, descrição em vhdl, e prototipação de uma unidade lógica aritmética, tendo como objetivo final a implementação de algumas funções, são elas: adição, subtração, multiplicação, divisão, or, and, not a, not b.*

1. Informação Geral

Neste trabalho foi utilizado uma FPGA (que consiste em um arranjo de células lógicas ou blocos lógicos configuráveis contidos em um único circuito integrado), a linguagem de descrição de hardware (VHDL) para implementar tais funções, além do software Quartus II da Altera.

1.1. Funcionamento da ULA

Muitas das ações dos computadores são executadas pela ULA. Esta recebe dados dos registradores, que são processados e os resultados da operação são armazenados nos registradores de saída. Outros mecanismos movem os dados entre esses registradores e a memória. Uma unidade de controle controla a ULA, através de circuitos que dizem que operações a ULA deve realizar.

Em muitos projetos a ULA também leva ou gera as entradas ou saídas de um conjunto de códigos de condições ou de um registrador de estado. Esses códigos são usados para indicar casos como vai-um (empresta-um), excesso (overflow), divisão-por-zero etc.

2. Implentação

Para a realização da unidade lógica aritmética, foi necessária a separação de cada função em pequenos arquivos vhdl, de modo a facilitar a visualização do código e também tornar mais simples o momento de encontrar erros ou falhas de projeto.

3. Somador

O somador de 4 bits é praticamente baseado em um projeto de uma das nossas aulas práticas de técnicas digitais, onde foram implementados três arquivos vhdl, um meio somador, outro somador completo, e por fim o somador4bits utilizado como ”top level”que ligava todos os outros a partir de ”port maps”, assim como pode se observar na figura 1.

```
1   LIBRARY  IEEE;
2   USE IEEE.std_logic_1164.all;
3
4   ENTITY somador IS
5     PORT(
6     a, b: in std_logic_vector(3 downto 0);
7     s: out std_logic_vector(3 downto 0);
8     c4: out std_logic;
9     Flag_Zero : out STD_LOGIC;
10    Flag_Sinal : out STD_LOGIC;
11    Flag_Overflow : out STD_LOGIC
12    );
13  end somador;
14
15  architecture hardware of somador is
16
17  SIGNAL c: std_logic_vector(2 downto 0);
18
19  COMPONENT meio_somador
20    PORT (
21    a, b: in std_logic;
22    s, c_out: out std_logic
23    );
24  END COMPONENT;
25
26  COMPONENT somador_completo
27    PORT (-
28    a, b, c_in: in std_logic;
29    s, c_out: out std_logic
30    );
31  END COMPONENT;
32
33  signal prop: std_logic_vector(2 downto 0);
34  signal valor : std_logic_vector(3 downto 0);
35  signal c3 : std_logic;
36  signal c2 : std_logic;
37
38  begin
39    soma1: meio_somador
40      port map(a => a(0), b => b(0), s => s(0), c_out => c(0));
41
42    soma2: somador_completo
43      port map(a => a(1), b => b(1), c_in => c(0), s => s(1), c_out => c(1));
44
45    soma3: somador_completo
46      port map(a => a(2), b => b(2), c_in => c(1), s => s(2), c_out => c(2));
47
48    soma4: somador_completo
49      port map(a => a(3), b => b(3), c_in => c(2), s => s(3), c_out => c4);
50
51      Flag_Zero <= not(valor(0) or valor(1) or valor(2) or valor(3));
52      Flag_Sinal <= valor(3);
53      Flag_Overflow <= c3 xor c2;
54
55  end hardware;
```

Figura 1. somador vhdl

4. Subtrator

O projeto consiste em desenvolver o módulo subtrator, que dado dois vetores de 4 bits x1 e y1 (respectivamente), retorna x1 - y1, e a suas respectivas flags, do tipo sinal, zero e overflow. O arquivo vhdl principal, ainda é composto por mais um arquivo vhdl, através de um component, esse sendo uma parte de seu subtrator, denominado ”FullSub”, assim como pode se observar na figura 2.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity subtrator is
port
(
 x1: in std_logic_vector(3 downto 0);
 y1: in std_logic_vector(3 downto 0);
 S: out std_logic_vector(3 downto 0);
 cout: out std_logic;
 Flag_Zero : out STD_LOGIC;
 Flag_Overflow : out STD_LOGIC;
 Flag_Sinal : out STD_LOGIC
);
end subtrator;

architecture subtrator_completo of subtrator is
component FullSub is
 Port( x: in std_logic;
       y: in std_logic;
       bin: in std_logic;
       bout: out std_logic;
       dif: out std_logic
      );
end component;
Signal C: std_logic_vector (3 downto 1);
Signal cin: std_logic;
SIGNAL ynvertido: STD_LOGIC_VECTOR (3 downto 0);
SIGNAL resultado: std_logic_vector (3 downto 0);
SIGNAL Flag_Zero_somador: std_logic;
SIGNAL Flag_Sinal_somador: std_logic;
SIGNAL Flag_Overflow_somador: std_logic;
signal armazena_borrow : std_logic;
signal borrow : std_logic;

begin

FS0: FullSub port map (x => x1(0), y => y1(0), Bin => cin, Bout => C(1), Dif => S(0));
FS1: FullSub port map (x => x1(1), y => y1(1), Bin => C(1), Bout => C(2), Dif => S(1));
FS2: FullSub port map (x => x1(2), y => y1(2), Bin => C(2), Bout => C(3), Dif => S(2));
FS3: FullSub port map (x => x1(3), y => y1(3), Bin => C(3), Bout => cout, Dif => S(3));

   Flag_Zero <= not(resultado(0) or resultado(1) or resultado(2) or resultado(3));
   Flag_Overflow <= Flag_Overflow_somador;
   armazena_borrow <= not borrow;
   Flag_Sinal <= resultado(3);
end subtrator_completo;
```

Figura 2. subtrator vhdl

5. And

O projeto consiste em desenvolver o módulo and, que dado dois vetores de 4 bits A e B, retorne A and B, e a suas respectivas flags, do tipo sinal, zero. Assim como pode se observar na figura 3.

6. Or

O projeto consiste em desenvolver o módulo or, que dado dois vetores de 4 bits A e b, retorne A or B, e a suas respectivas flags, do tipo sinal, zero. Assim como pode se observar na figura 4.

7. Inversor A

O projeto consiste em desenvolver o módulo inversor A, que dado um vetor de 4 bits A, retorne not(A), e a suas respectivas flags, do tipo sinal, zero. Assim como pode se observar na figura 5.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity a_and_b is
    Port ( x : in  STD_LOGIC_VECTOR (3 downto 0);
           y : in  STD_LOGIC_VECTOR (3 downto 0);
           saida : out  STD_LOGIC_VECTOR (3 downto 0);
           Flag_Zero : out STD_LOGIC;
           Flag_Sinal : out STD_LOGIC);
end a_and_b;

architecture Behavioral of a_and_b is
signal valor : std_logic_vector (3 downto 0);
begin
   Gen_1: For I IN 3 downto 0 generate
          saida(I) <= x(I) and y(I);
   end generate;
   Flag_Zero <= not(valor(0) or valor(1) or valor(2) or valor(3));
   Flag_Sinal <= valor(3);
end Behavioral;
```

Figura 3. aandb vhdl

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity a_or_b is
    Port ( x : in  STD_LOGIC_VECTOR (3 downto 0);
           y : in  STD_LOGIC_VECTOR (3 downto 0);
           saida : out  STD_LOGIC_VECTOR (3 downto 0);
           Flag_Zero : out STD_LOGIC;
           Flag_Sinal : out STD_LOGIC);
end a_or_b;

architecture Behavioral of a_or_b is

signal valor : std_logic_vector(3 downto 0);
begin
   Gen_1: For I IN 3 downto 0 generate
          saida(I) <= x(I) or y(I);
   end generate;

   Flag_Zero <= not(valor(0) or valor(1) or valor(2) or valor(3));
   Flag_Sinal <= valor(3);
end Behavioral;
```

Figura 4. aorb vhdl

8. Inversor B

O projeto consiste em desenvolver o módulo inversor A, que dado um vetor de 4 bits B, retorne not(B), e a suas respectivas flags, do tipo sinal, zero. Assim como pode se observar na figura 6.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity a_not is
    Port ( x : in  STD_LOGIC_VECTOR (3 downto 0);
           saida : out  STD_LOGIC_VECTOR (3 downto 0);
           Flag_Zero : out STD_LOGIC;
           Flag_Sinal : out STD_LOGIC);
end a_not;

architecture Behavioral of a_not is
SIGNAL valor : STD_LOGIC_VECTOR(3 downto 0);
begin
   Gen_1: For I IN 3 downto 0 generate
          saida(I) <= not x(I);
   end generate;
   Flag_Zero <= not (valor(0) or valor(1) or valor(2) or valor(3));
   Flag_Sinal <= valor(3);

end Behavioral;
```

Figura 5. nota vhdl

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity b_not is
    Port ( x : in  STD_LOGIC_VECTOR (3 downto 0);
           saida : out  STD_LOGIC_VECTOR (3 downto 0);
           Flag_Zero : out STD_LOGIC;
           Flag_Sinal : out STD_LOGIC);
end b_not;

architecture Behavioral of b_not is
SIGNAL valor : STD_LOGIC_VECTOR(3 downto 0);
begin
   Gen_1: For I IN 3 downto 0 generate
          saida(I) <= not x(I);
   end generate;
   Flag_Zero <= not (valor(0) or valor(1) or valor(2) or valor(3));
   Flag_Sinal <= valor(3);

end Behavioral;
```

Figura 6. notb vhdl

9. Multiplicador

O projeto consiste em desenvolver o módulo multiplicador, que dado um vetor de 4 bits A, retorne a multiplicação por 2 e por 4, e a suas respectivas flags, do tipo sinal, zero e overflow. Assim como pode se observar na figura 7.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity multiplicador is
    Port ( a: in  STD_LOGIC_VECTOR (3 downto 0);
           Operacao: in  BIT;
           s : out  STD_LOGIC_VECTOR (3 downto 0);
           Flag_Zero : out STD_LOGIC;
           Flag_Sinal : out STD_LOGIC;
           Flag_Overflow : out STD_LOGIC
           );
end multiplicador;

architecture Behavioral of multiplicador is
COMPONENT multi_por_2 is
    Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
           desloca : in BIT;
           s : out  STD_LOGIC_VECTOR (3 downto 0);
           Flag_Zero : out STD_LOGIC;
           Flag_Overflow: out STD_LOGIC;
           Flag_Sinal : out STD_LOGIC
           );
end COMPONENT multi_por_2;

COMPONENT multi_por_4 is
    Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
           desloca : in BIT;
           s : out  STD_LOGIC_VECTOR (3 downto 0);
           Flag_Zero : out STD_LOGIC;
           Flag_Overflow: out STD_LOGIC;
           Flag_Sinal : out STD_LOGIC
           );
end COMPONENT multi_por_4;
signal dsl: BIT;
signal flag_over_mult, flag_sin_mult, flag_zero_mult: std_LOGIC;
signal multiplica_two, multiplica_four: std_LOGIC_VECTOR(3 downto 0);
begin
T0: multi_por_2 port map(a, dsl, multiplica_two, flag_over_mult, flag_sin_mult, flag_zero_mult);
T1: multi_por_4 port map(a, dsl, multiplica_four,flag_over_mult, flag_sin_mult, flag_zero_mult);
process(Operacao, multiplica_two, multiplica_four)
begin
   case Operacao is
   when '0' =>
   s <= multiplica_two;
   when '1' =>
   s <= multiplica_four;
   end case;
   end process;
end Behavioral;
```

Figura 7. multiplicador vhdl

10. Divisor

O projeto consiste em desenvolver o módulo divisor, que dado um vetor de 4 bits A, retorne a divisão por 2 e por 4, e a suas respectivas flags, do tipo sinal, zero e overflow. Assim como pode se observar na figura 8:

11. Ula

O arquivo alu.vhd, abrange a Unidade Lógica Aritmética, onde a mesma não é o arquivo principal, não sendo setado como top level, porém é sem dúvida a parte mais importante do projeto, nele optamos por não usar um arquivo mux8x1 e optamos por um case principal, que utiliza with select, fazendo a chamada das operações. Foi utilizado alguns sinais, para ligarem os port maps nas saídas de cada arquivo.vhd respectivo, a operação de seleção funciona da forma como a figura 9:

11.1. Circuito Projeto

O circuito é composto por um arquivo top level, nomeado Seg7Ula, onde conecta as saidas dos dois displays de sete segmentos, onde as entradas A e B e as operações entram na ula, saem com os resultados e as respectivas flags, passam pelo arquivo Seg7Ula que posteriormente os envia para suas saídas. Assim como pode se observar na figura 10.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity divisor is
    Port ( a: in  STD_LOGIC_VECTOR (3 downto 0);
           Operacao: in  BIT;
           s : out  STD_LOGIC_VECTOR (3 downto 0);
           Flag_Zero : out STD_LOGIC;
           Flag_Sinal : out STD_LOGIC;
           Flag_Overflow : out STD_LOGIC
           );
end divisor;

architecture Behavioral of divisor is
COMPONENT divi_por_2 is
    Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
           desloca : in BIT;
           s : out  STD_LOGIC_VECTOR (3 downto 0);
           Flag_Zero : out STD_LOGIC;
           Flag_Overflow: out STD_LOGIC;
           Flag_Sinal : out STD_LOGIC
           );
end COMPONENT divi_por_2;

COMPONENT divi_por_4 is
    Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
           desloca : in BIT;
           s : out  STD_LOGIC_VECTOR (3 downto 0);
           Flag_Zero : out STD_LOGIC;
           Flag_Overflow: out STD_LOGIC;
           Flag_Sinal : out STD_LOGIC
           );
end COMPONENT divi_por_4;
signal dsl: BIT;
signal divide_four, divide_two: std_LOGIC_VECTOR(3 downto 0);
signal flag_over_divi, flag_sin_divi, flag_zero_divi: std_LOGIC;
begin
T0: divi_por_2 port map(a, dsl, divide_two, flag_over_divi, flag_sin_divi, flag_zero_divi);
T1: divi_por_4 port map(a, dsl, divide_four,flag_over_divi, flag_sin_divi, flag_zero_divi);
process(Operacao, divide_two, divide_four)
begin
   case Operacao is
   when '0' =>
   s <= divide_two;
   when '1' =>
   s <= divide_four;
   end case;
   end process;
end Behavioral;
```

Figura 8. divisor vhdl

11.2. Circuito Ula

O circuito da ula é composto por oito operações, que são feitas após a entrada A e B de 4 bits serem feitas através dos switches da placa, onde os switches de A vão de SW[0] até SW[3], e os de vão de SW[14] até SW[17], da forma que é explicada no manual da ula, criado para auxiliar no manejo das operações e devidas funções da placa, onde as saídas são ligas por alguns sinais que passam para os dois displays de sete segmentos via port maps, e retorna os sinais dos leds R[6] ao R[8]. Assim como pode se observar na figura 11.

```
108     process (Operacao, Not_A, Not_b, A_mais_B, A_veses, A_menos_B, AandB, A_divi, AorB)
109     begin
110        case Operacao is
111           when "000" =>
112              Z <= A_mais_B;
113              Flag_Zero <= Flag_Zero_somador;
114              Flag_Sinal <= Flag_Sinal_somador;
115              Flag_Overflow <= Flag_Overflow_somador;
116           when "001" =>
117              Z <= A_menos_B;
118              Flag_Zero <= Flag_Zero_subtrator;
119              Flag_Sinal <= Flag_Sinal_subtrator;
120              Flag_Overflow <= Flag_Borrow_subtrator;
121           when "010" =>
122              Z <= Not_A;
123              Flag_Zero <= Flag_Zero_inversora;
124              Flag_Sinal <= Flag_Sinal_inversora;
125              Flag_Overflow <= '0';
126           when "011" =>
127              Z <= Not_B;
128              Flag_Zero <= Flag_Zero_inversorb;
129              Flag_Sinal <= Flag_Sinal_inversorb;
130              Flag_Overflow <= '0';
131           when "100" =>
132              Z <= AorB;
133              Flag_Zero <= Flag_Zero_or;
134              Flag_Sinal <= Flag_Sinal_or;
135              Flag_Overflow <= '0';
136           when "101" =>
137              Z <= AandB;
138              Flag_Zero <= Flag_Zero_and;
139              Flag_Sinal <= Flag_Sinal_and;
140              Flag_Overflow <= '0';
141           when "110" =>
142              Z <= A_divi;
143              Flag_Zero <= Flag_Zero_divisor;
144              Flag_Sinal <= Flag_Sinal_divisor;
145              Flag_Overflow <= Flag_Overflow_divisor;
146           when "111" =>
147              Z <= A_veses;
148              Flag_Zero <= Flag_Zero_somador;
149              Flag_Sinal <= Flag_Sinal_multiplicador;
150              Flag_Overflow <= Flag_Overflow_multiplicador;
151        end case;
152     end process;
153
154  end Behavioral;
```

Figura 9. Operações

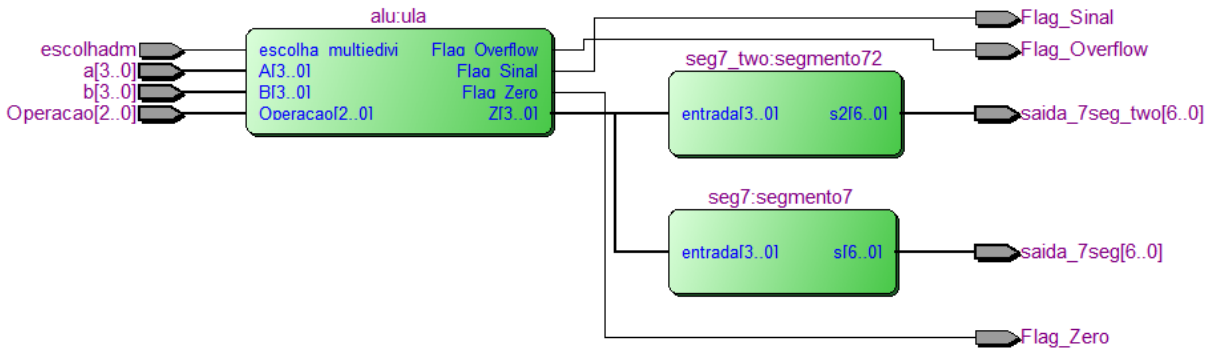

Figura 10. Circuito

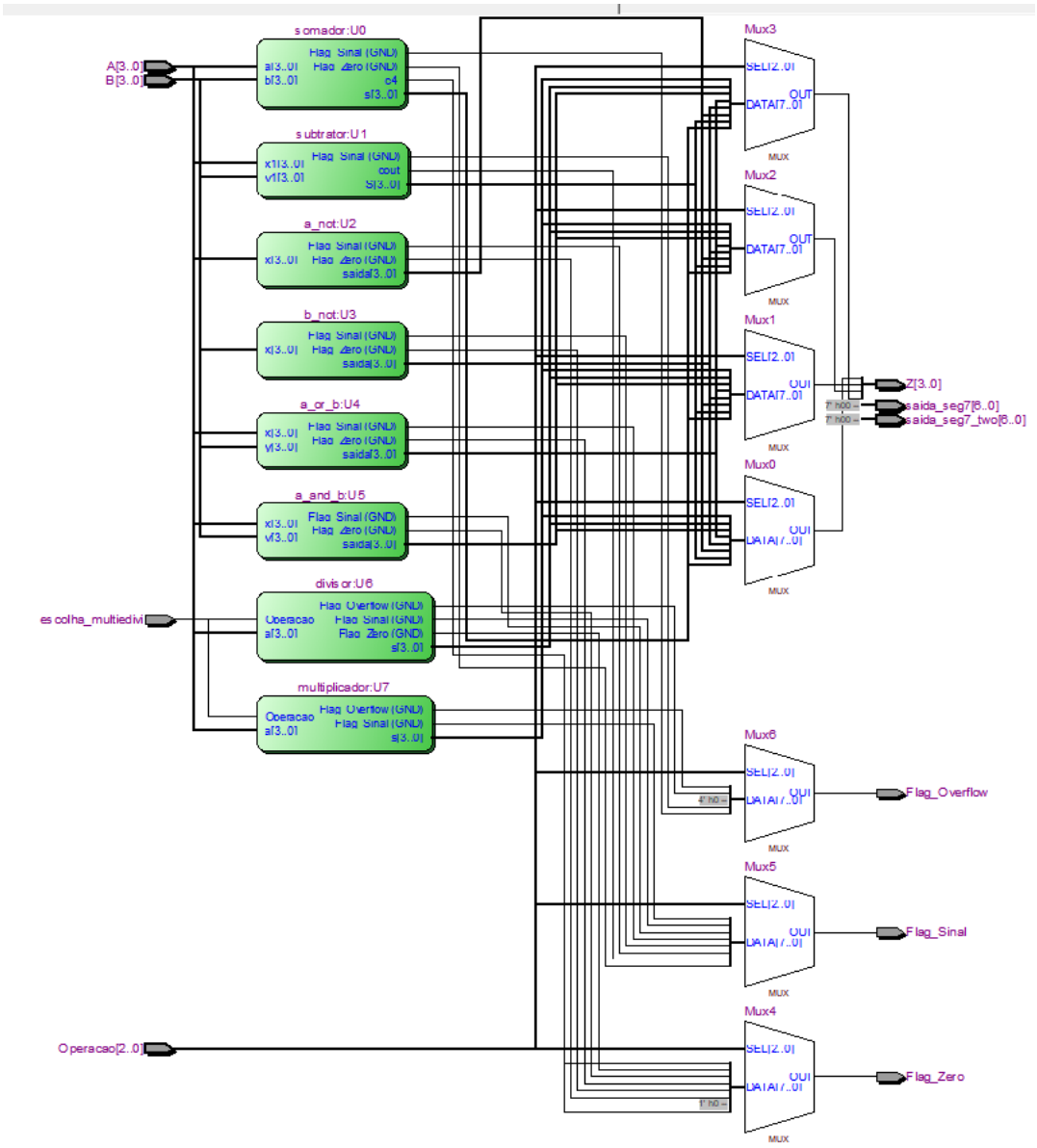

Figura 11. Circuito Ula