EXPERIMENTO 3. SIMULAÇÃO DE CIRCUITOS DIGITAIS

-Bernardo Rizzone (232013194)

Introdução: Este relatório tem como objetivos a construção de um somador de palavras binarias usando somadores completos em cascata, além da familiarização com o pacote STD_LOGIC_ARITH. Usaremos o ModelSim para a implementação dos códigos e simulação dos mesmos.

1) O objetivo do primeiro exercício e escrever e simular um somador de palavras de 4 bits no ModelSim. O somador e construído utilizando somente somadores completos, que foram desenvolvidos em experimentos anteriores. A nova entidade tem como entrada dois vetores, A e B (com 4 bits cada), e como saída, um vetor S (com 5 bits

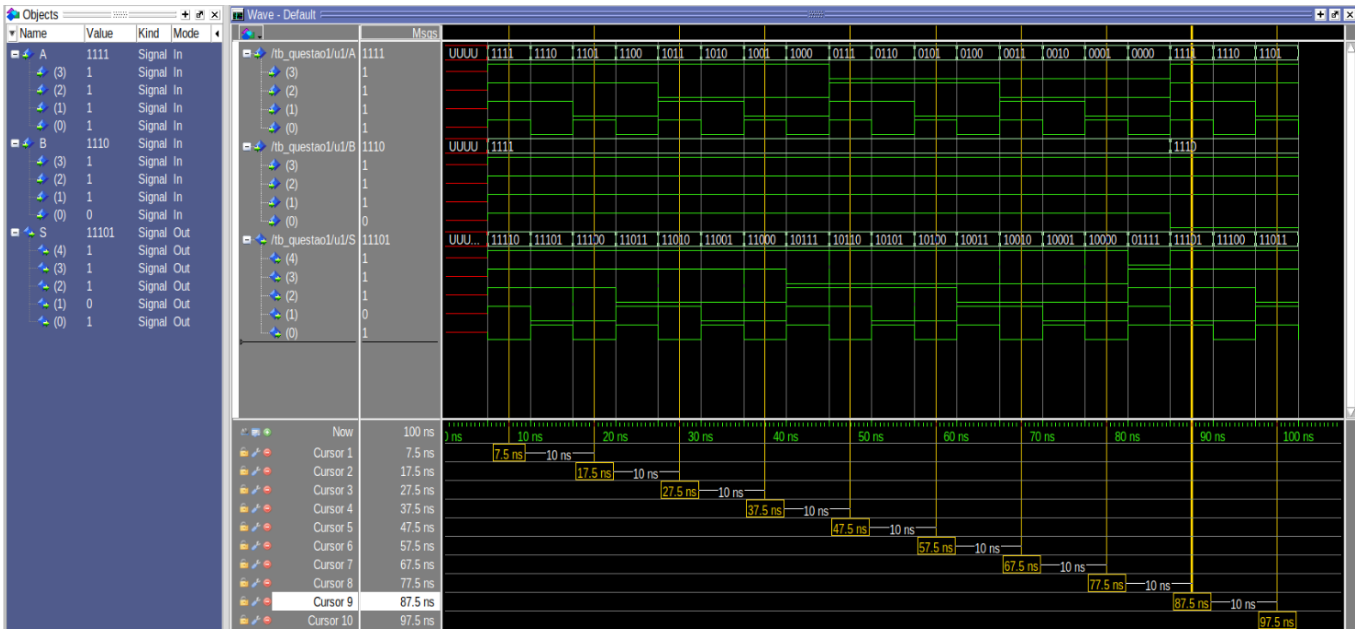
   ***Implementação e simulação**: A implementação do somador de 4 bits e feita utilizando a entidade do somador completo como um componente. Cada bit dos vetores de entrada A e B e alimentado em um somador completo, juntamente com o carry do somador anterior. O resultado de cada somador e então coletado para formar o vetor de saída S.

CÓDGIO:

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5
6
7  entity questaol is
8      port (
9          A   :in std_logic_vector(3 downto 0);
10         B   :in std_logic_vector(3 downto 0);
11         S   :out std_logic_vector(4 downto 0)
12     );
13 end questaol;
14
15
16
17 architecture rtl of questaol is
18
19
20     component SOMADOR is
21         port (
22             A      :in std_logic;
23             B      :in std_logic;
24             Cin    :in std_logic;
25             S      :out std_logic;
26             Cout   :out std_logic
27         );
28     end component;
29
30
31 signal carry_signal :std_logic_vector(4 downto 0);
32
33
34 begin
35     carry_signal(0) <= '0';
36
37
38     sum: for i in 0 to 3 generate
39
40         u0: SOMADOR PORT MAP (A(i), B(i), carry_signal(i), S(i), carry_signal(i+1));
41     end generate;
42
43     S(4) <= carry_signal(4);
44
45 end rtl;
```

TESTBENCH:

```
entity tb_questaol is end;



library ieee;
use ieee.std_logic_1164.ALL;
use std.textio.all;
use ieee.numeric_std.all;



architecture testbench of tb_questaol is



 component questaol is
    port (
        A   :in std_logic_vector(3 downto 0);
        B   :in std_logic_vector(3 downto 0);
        S   :out std_logic_vector(4 downto 0)
    );
 end component;

  signal s  : std_logic_vector(7 downto 0);



begin
    ul: questaol PORT MAP (
        A => s(3 downto 0),
        B => s(7 downto 4),
        S => open
    );



 estimulo: process
    begin
        for i in 255 downto 0 loop
            wait for 5 ns;
            s <= std_logic_vector(to_unsigned(i, 8));
        end loop;
        wait;
```



A SIMULAÇÃO: A simulação do somador de 4 bits e realizada no ModelSim. Os vetores de entrada A e B são variados para verificar a corretude do somador. Os resultados da simulação confirmam que o somador está funcionando corretamente.

2) O objetivo deste exercício e escrever e simular um somador de palavras de 4 bits no modelsim, construído utilizando o operador '+' do pacote STD_LOGIC_ARITH. A nova entidade tem como entrada dois vetores, A e B(com 4 bits cada), e como saída, um vetor S (com 5 bits).

***Implementação e simulação**: A implementação do somador de 4 bits e feita utilizando o operador '+'' do pacote STD_LOGIC_ARITH. As entradas A e B são declaradas como sendo do tipo STD_LOGIC_VECTOR e, na arquitetura, são convertidas para o tipo UNSIGNED usando o comando unsigned(.) de modo a poder usar o operador '+'.

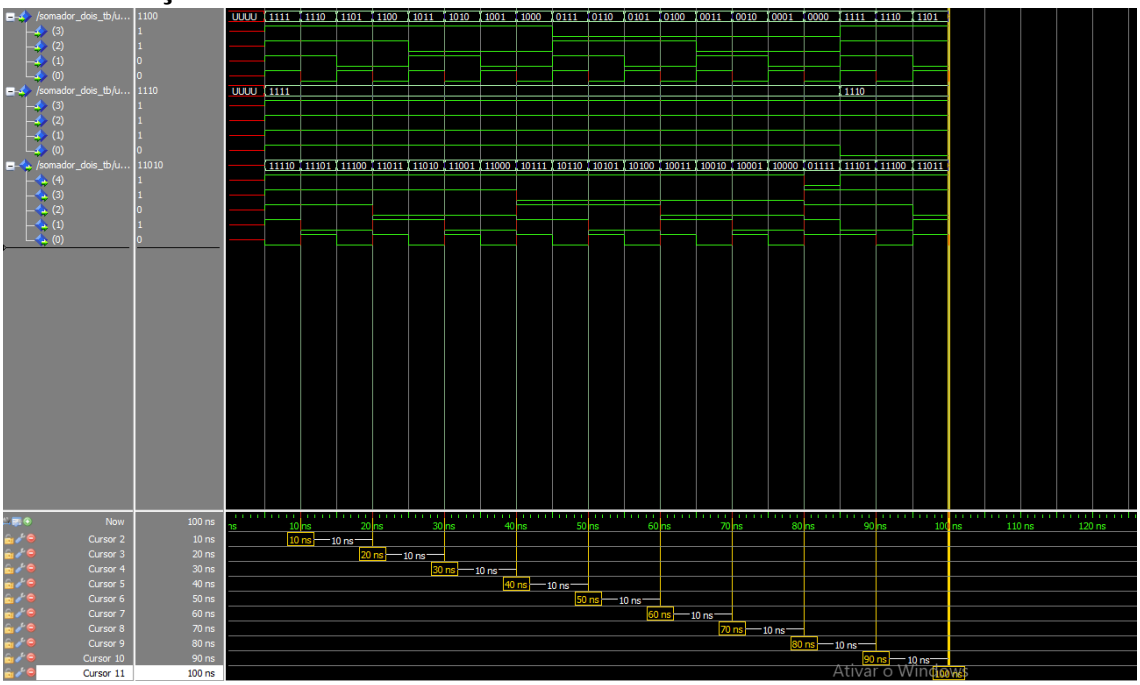
CÓDIGO:

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4
5
6
7  entity somador_dois is
8      port (
9          A   :in std_logic_vector(3 downto 0);
10         B   :in std_logic_vector(3 downto 0);
11         S   :out std_logic_vector(4 downto 0)
12     );
13 end somador_dois;
14
15
16
17 architecture rtl of somador_dois is
18
19 begin
20
21 -- S = f(A, B)
22     S <= unsigned('0' & A) + unsigned('0' & B);
23
24 end rtl;
25
```

TESTBENCH:

```
entity somador_dois_tb is end;


library ieee;
use ieee.std_logic_1164.ALL;
use std.textio.all;
use ieee.numeric_std.all;



architecture testbench of somador_dois_tb is



 component somador_dois is
    port (
        A   :in std_logic_vector(3 downto 0);
        B   :in std_logic_vector(3 downto 0);
        S   :out std_logic_vector(4 downto 0)
    );
 end component;



  signal s  : std_logic_vector(7 downto 0);



begin
    ul: somador_dois PORT MAP (
        A => s(3 downto 0),
        B => s(7 downto 4),
        S => open
    );

 estimulo: process
    begin
        for i in 255 downto 0 loop
            wait for 5 ns;
            s <= std_logic_vector(to_unsigned(i, 8));
        end loop;
        wait;

 end process;
```

SIMULAÇÃO



SIMULAÇÃO: A simulação e iniciada executando o código VHDL no ModelSim. Durante a simulação, os vetores de entrada A e B são variados através de todos os possíveis valores de 4 bits (de “0000” a “1111”). Para cada combinação de A e B, o vetor de saída S e calculado e registrado. Após a conclusão da simulação, os resultados são analisados. Isso envolve a verificação do vetor de saída S para cada combinação de A e B e a comparação dos resultados com as expectativas.

3) O objetivo deste exercício e escrever e simular um testbench no modelsim para simular e testar o somador de 4 palavras desenvolvido no visto 1. Este testbench deve gerar todas as 256 combinações possíveis de valores para A e B, aguardando 500 ns entre combinações. Para cada combinação, o testbench deve comparar a saída do somador do visto 1 com a saída do somador do visto 2, imprimindo uma mensagem de erro caso as saídas não concordarem.

***Implementação e simulação:** A implementação do testbench e realizada no modelsim. Durante a simulação o testbench gera todas as 256 combinações possíveis de valores para A e B, aguardando 500 ns entre as combinações.

CODIGO: TOP_MODULE:

```
entity top_module is end;



library ieee;
use ieee.std_logic_1164.ALL;
use std.textio.all;
use ieee.numeric_std.all;

-- Architecture (Arquitetura)

architecture topmodule_arch of top_module is



 component questaol is
    port (
        A   :in std_logic_vector(3 downto 0);
        B   :in std_logic_vector(3 downto 0);
        S   :out std_logic_vector(4 downto 0)
    );
 end component;



 component somador_dois is
    port (
        A   :in std_logic_vector(3 downto 0);
        B   :in std_logic_vector(3 downto 0);
        S   :out std_logic_vector(4 downto 0)
    );
 end component;



 component tb_somador3 is
    port (
        A   :out std_logic_vector(3 downto 0);
        B   :out std_logic_vector(3 downto 0);
        dut :in std_logic_vector(4 downto 0);
        gm  :in std_logic_vector(4 downto 0)
    );
    end component;
```

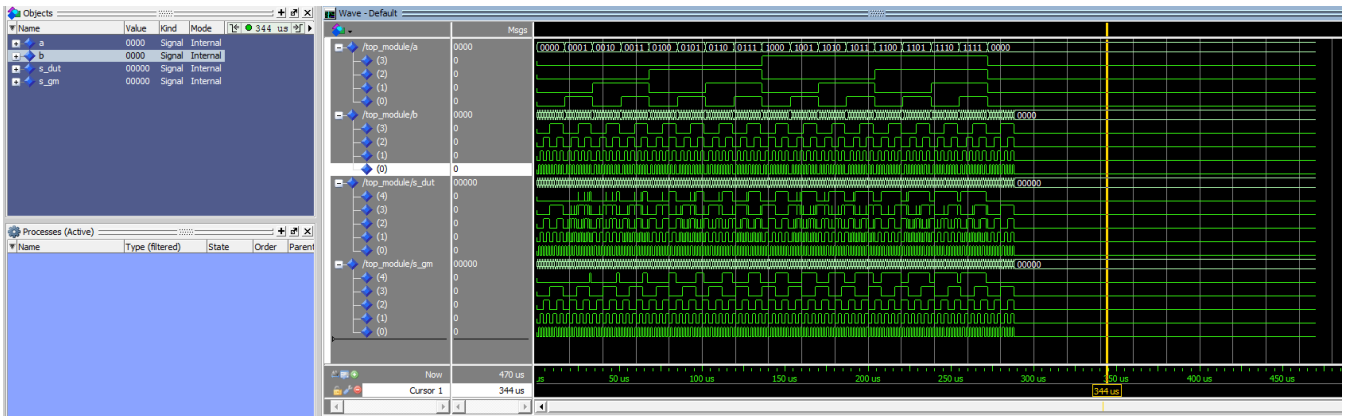
TEST BENCH:

```
library ieee;
use ieee.std_logic_1164.ALL;
use std.textio.all;
use ieee.numeric_std.all;



entity tb_somador3 is
    port (
        A   :out std_logic_vector(3 downto 0);
        B   :out std_logic_vector(3 downto 0);
        dut :in std_logic_vector(4 downto 0);
        gm  :in std_logic_vector(4 downto 0)
    );
end tb_somador3;



architecture testbench_arch of tb_somador3 is



begin



 process
    begin
        report "Iniciando  teste ..."  severity  NOTE;
        for i in 0 to 16 loop
            A <= std_logic_vector(to_unsigned(i, 4));
            for j in 0 to 16 loop
                B <= std_logic_vector(to_unsigned(j, 4));
                wait for 500 ns;

                assert(dut = gm) report "Teste Falhou" severity ERROR;
            end loop;
        end loop;
        report "Teste  finalizado !"  severity  NOTE;
        wait;


 end process;


END
```

CODIGO SOMADOR USADO NOS EXERCICIOS 1 E 3:

```
library IEEE;
use IEEE.std_logic_1164.all;



entity SOMADOR
    is port (
        A       :in std_logic;
        B       :in std_logic;
        Cin     :in std_logic;
        S       :out std_logic;
        Cout    :out std_logic
    );
end SOMADOR;



architecture rtl of SOMADOR is

begin


S <= A xor B xor Cin;


Cout <= (A and B) or (A and Cin) or (B and Cin);


end rtl;
```

SIMULAÇÃO:



SIMULAÇÃO: Após a conclusão da simulação, os resultados são analisados. Isso envolve a verificação das mensagens de erro impressas pelo testbench. Se nenhuma

mensagem de erro for impressa, isso indica que o somador do visto 1(DUT) esta funcionando corretamente, pois suas saídas concordam com as do somador do visto 2(Golden model) para todas as 256 combinações possíveis de valores para A e B.

Conclusão:

Ao logo desses exercíos, exploramos várias facetas do design de hardware, esses exercícios foram uma excelente oportunidade de aprimoramento de nossas habilidades. Conseguimos cumprir todas as metas estabelecidas anteriormente na introdução, demonstramos nosso conhecimento em somadores completos, e também na biblioteca STD_LOGIC_ARITH.