Universidade de Brasília
Faculdade de Tecnologia
Laboratório de Sistemas Digitais

Relatório 05

Carla de Araujo Clementino Ribeiro Mat:180030736

Professor:
Guilherme de Sousa Torres

1 Objetivos

Implementar um somador de palavras binárias utilizando somadores completos e provar seu funcionamento através de testes.

2 Questões Propostas

1. Escrever em VHDL e simular no ModelSim um somador de palavras de 4 bits, construído utilizando somente somadores completos (entidade desenvolvida no visto 1 do experimento 2, utilizada aqui como "component"). A nova entidade deve ter como entrada dois vetores A e B (com 4 bits cada) e, como saída, um vetor S (com 5 bits).

2. Escrever em VHDL e simular no ModelSim um somador de palavras de 4 bits, construído utilizando o operador '+' do pacote `STD_LOGIC_ARITH`. A nova entidade deve ter como entrada dois vetores A e B (com 4 bits cada) e, como saída, um vetor S (com 5 bits). Dica: declare as entradas A e B como sendo do tipo `STD_LOGIC_VECTOR` (como feito no visto 1) e, na arquitetura, faça a conversão para o tipo `UNSIGNED` usando o comando `unsigned(.)` de modo a poder usar o operador '+'.

3. Escrever em VHDL e simular no ModelSim um um testbench para simular e testar o somador de palavras de 4 bits desenvolvido no visto 1. Esse testbench deve gerar todas as 256 combinações possíveis de valores para A e B, aguardando 500 ns entre combinações e, para cada combinação, comparar a saída do somador do visto 1 (utilizado aqui como device under test ou DUT) com a saída do somador do visto 2 (utilizado aqui como golden model), imprimindo uma mensagem de erro se as saídas não concordarem.

3 Desenvolvimento Teórico

3.1 Somador Parcial

Um somador parcial é um componente básico usado na aritmética binária para adicionar dois bits juntos. Ele é chamado de "parcial"porque não considera o carry (vai um) de soma de bits anteriores. O somador parcial recebe dois bits de entrada A e B e produz dois resultados: a soma (S) e o carry-out (C_{out}).

Seja A e B os bits de entrada, então:

$$\text{Soma } (S): \; S = A \oplus B$$
$$\text{Carry-out } (C_{\text{out}}): \; C_{\text{out}} = A \cdot B$$

3.2 Somador Completo:

Um somador completo é um circuito que leva em conta o carry de soma dos bits anteriores, além de realizar a soma dos bits de entrada. Ele recebe três bits de entrada A, B e C_{in}, onde C_{in} é o carry-in, e produz dois resultados: a soma (S) e o carry-out (C_{out}).

Seja A, B e C_{in} os bits de entrada, então:

$$\text{Soma } (S): \; S = (A \oplus B) \oplus C_{\text{in}} \quad \text{onde } \oplus$$
$$\text{Carry-out } (C_{\text{out}}): \; C_{\text{out}} = (A \cdot B) + (C_{\text{in}} \cdot (A \oplus B))$$

4 Simulação

4.1 Primeira Questão

Considerando a questão proposta, foi utilizado o somador completo desenvolvido no experimento anterior, o qual pode ser visto abaixo:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity somador_completo is
  Port (
      A : in STD_LOGIC;
      B : in STD_LOGIC;
      Cin : in STD_LOGIC;
      S : out STD_LOGIC;
      Cout : out STD_LOGIC);
end somador_completo;

architecture Behavioral of somador_completo is

begin

S <= A xnor B xnor Cin;

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


end Behavioral;

```

Figura 1: Somador Completo. Fonte: Autor.

Em seguida, criamos o top module para que fosse possível realizar simulações do nosso circuito por meio de sinais controlados. Foram utilizados saídas extras para observarmos o comportamento dos carry-outs na simulação:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity top_module 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);
        Cout0: out STD_LOGIC;
        Cout1: out STD_LOGIC;
        Cout2: out STD_LOGIC;
        Cout3: out STD_LOGIC
);
end top_module;

architecture Behavioral of top_module is

Component somador_completo
    Port (
        A : in STD_LOGIC;
        B : in STD_LOGIC;
        Cin : in STD_LOGIC;
        S : out STD_LOGIC;
        Cout : out STD_LOGIC
);
end Component;

signal c0,c1,c2 :STD_LOGIC := '0';
signal U1,U2 :STD_LOGIC_VECTOR(3 downto 0);
signal U3 :STD_LOGIC_VECTOR(4 downto 0);

begin

U1 <= A;
U2 <= B;
sum1: somador_completo port map (A=> U1(0), B=> U2(0), Cin=> '0', S=>U3(0), Cout=>c0);
Cout0 <= c0;
sum2: somador_completo port map (A=> U1(1), B=> U2(1), Cin=> c0, S=>U3(1), Cout=>c1);
Cout1 <= c1;
sum3: somador_completo port map (A=> U1(2), B=> U2(2), Cin=> c1, S=>U3(2), Cout=>c2);
Cout2 <= c2;
sum4: somador_completo port map (A=> U1(3), B=> U2(3), Cin=> c2, S=>U3(3), Cout=>U3(4));

S <= U3;
Cout3 <= U3(4);

end Behavioral;
```

Figura 2: Top Module - Somador Completo. Fonte: Autor.

Analisando o código é possível ver a utilização de 4 somadores para somar 2 palabras de 4 bits, que resulta em uma palavra de 5 bits.

Por fim, foi criado um aquivo de teste para simular todas as possíveis entradas do circuito:

```
Ln#
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3  use IEEE.STD_LOGIC_ARITH.all;
 4  use ieee.std_logic_unsigned.all;
 5
 6  entity test_bench is
 7  end;
 8
 9  architecture behavior of test_bench is
10
11  component top_module is port (
12          A : in STD_LOGIC_VECTOR (3 downto 0);
13          B : in STD_LOGIC_VECTOR (3 downto 0);
14          S : out STD_LOGIC_VECTOR (4 downto 0);
15          Cout0: out STD_LOGIC;
16          Cout1: out STD_LOGIC;
17          Cout2: out STD_LOGIC;
18          Cout3: out STD_LOGIC
19  );
20
21  end component;
22
23  signal Uaux : std_logic_vector (7 downto 0):= (others => '0');
24  signal U1 : std_logic_vector (3 downto 0) := (others => '0');
25  signal U2 : std_logic_vector (3 downto 0):= (others => '0');
26  signal U3 : std_logic_vector (4 downto 0):= (others => '0');
27  signal C0, C1, C2, C3 : std_logic;
28
29  begin
30
31  U1 <= Uaux(3 downto 0);
32  U2 <= Uaux(7 downto 4);
33
34  somador:  top_module port map (
35        A => U1,
36        B => U2, S => U3 ,
37        Cout0 => C0,
38        Cout1 => C1,
39        Cout2 => C2,
40        Cout3 => C3
41  );
42
43  estimulo: process
44
45  begin
46
47  for I in 0 to 255 loop
48        wait for 500 ns; Uaux <= UNSIGNED (Uaux) +1;
49  end loop;
50  wait;
51  end process;
52
53  end behavior;
```

Figura 3: Test Bench - Somador Completo. Fonte: Autor.

Realizando a simulação obtemos os seguintes resultados:

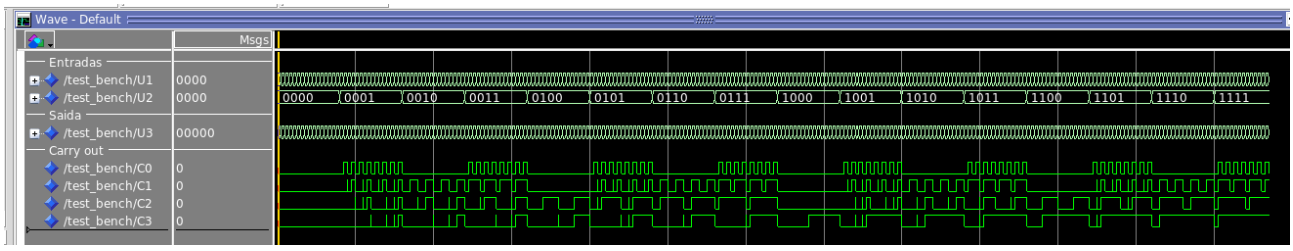

Figura 4: Simulação - Somador CompletoFonte: Autor.

Onde:

- $U1$ e $U2$: entradas de 4 bits;
- $U3$: saída de 5 bits;
- $C0$: carry-out da soma dos bits menos significativos de $U1$ e $U2$;
- $C1$: carry-out da soma do segundo bit menos significativos de $U1$ e $U2$;
- $C2$: carry-out da soma do segundo bit mais significativos de $U1$ e $U2$;
- $C3$: carry-out da soma dos bits mais significativos de $U1$ e $U2$.

4.1.1 Análise da Simulação

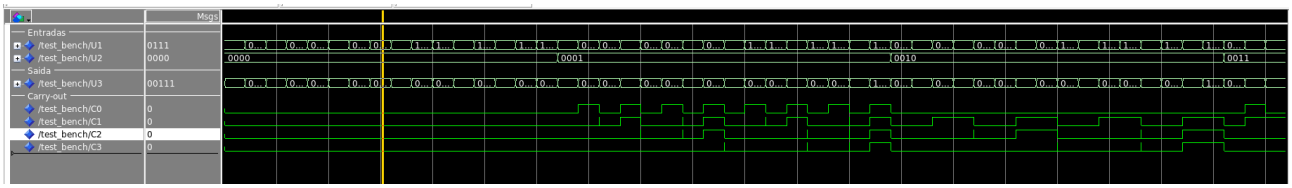

Figura 5: Simulação - Somador Completo. Sem carry-out. Fonte: Autor.

Na imagem acima, podemos analisar um caso em que não ocorre carry-out, pois temos como entradas "0111"(U1) e "0010"(U2), que quando somados resulta em "00111". Confirmamos isso ao analisar os sinais C0, C1, C2 e C3, os quais representam os carry-outs da soma dos bits menos significativos (C0) ao mais significativo (C3).

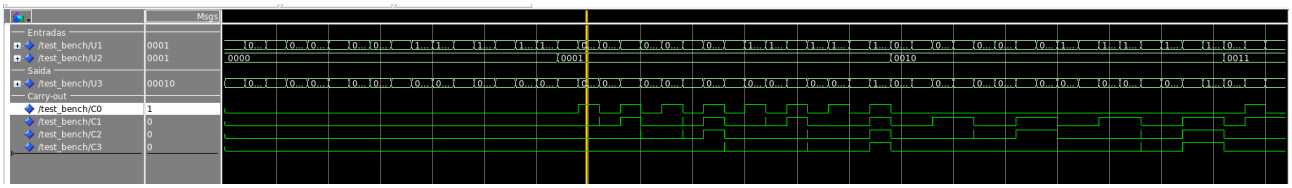

Figura 6: Simulação - Somador Completo. Carry-out no bit menos significativo. Fonte: Autor.

No caso acima, temos a soma de "0001"com "0001", que resultará em um carry-out na soma dos bits menos significativos. Podemos observar isso ao analisar o sinal C0, que indica a ocorrência de carry-out na soma dos bits menos significativos.

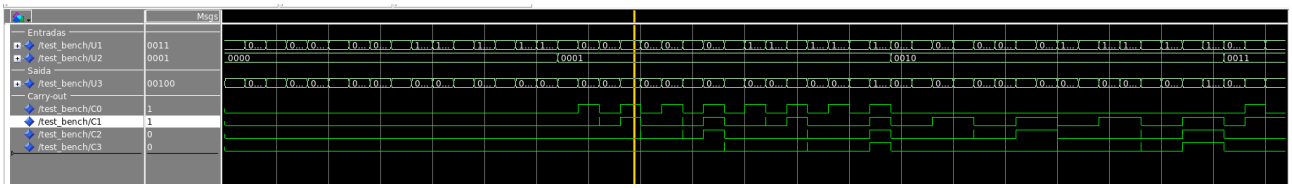

Figura 7: Simulação - Somador Completo. Carry-out nos 2 bits menos significativos. Fonte: Autor.

No exemplo acima, temos a soma de "0011"com "0001", que resultará em dois carry-outs: na soma dos bits menos significativo e na soma do segundo bit menos significativo. Podemos observar isso ao analisar os sinais C0 e C1, que indicam a ocorrência dos carry-outs.

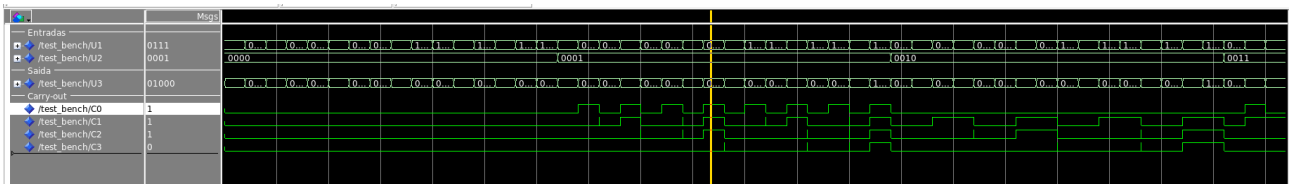

Figura 8: Simulação - Somador Completo. Carry-out nos 3 bits menos significativos. Fonte: Autor.

No figura 8, temos a soma de "0111"com "0001", que resultará em três carry-outs ao somar os 3 bits menos significativos. Podemos observar isso ao analisar os sinal C0, C1 e C2, que indicam a ocorrência dos carry-outs nos 3 bits em questão.

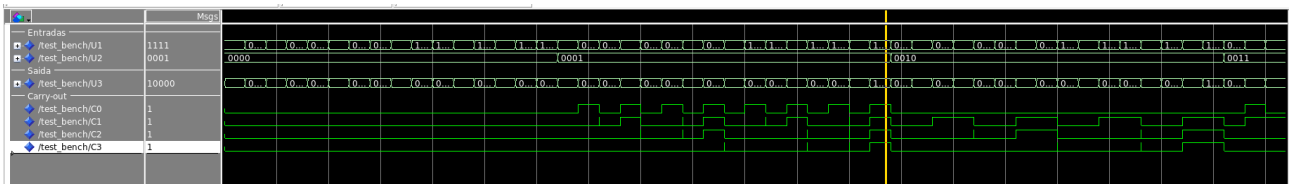

Figura 9: Simulação - Somador Completo. Carry-out em todas as somas. Fonte: Autor.

Por fim, podemos ver um caso que ocorre 4 carry-outs ao somar "1111"com "0001", como é indicado pelos sinais C0 a C3.

Existem outros casos de teste, porém iremos verificá-los por meio do golden model.

4.2 Segunda Questão

Para a segunda questão proposta, foi implementado um somador utilizando o pacote `STD_LOGIC_ARITH`. Essa entidade servirá como modelo de referência ("golden model") para verificar se nosso somador completo está se comportando como o esperado. Tendo isso em mente, primeiramente, implementamos o somador:

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4
5   entity somador_arith is
6     port (
7       A, B: in STD_LOGIC_VECTOR(3 downto 0);
8       S: out STD_LOGIC_VECTOR(4 downto 0)
9     );
10  end somador_arith;
11
12  architecture main of somador_arith is
13  begin
14    S <= UNSIGNED('0' & A) + UNSIGNED('0' & B);
15  end main;
16
```

Figura 10: Somador - Golden Model. Fonte: Autor.

Em seguida, criamos um top module e um arquivo de teste para simular todas as entradas possíveis:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity top_module_arith is
5      Port (
6          A: in STD_LOGIC_VECTOR (3 downto 0);
7          B: in STD_LOGIC_VECTOR (3 downto 0);
8          S: out STD_LOGIC_VECTOR (4 downto 0)
9  );
10 end top_module_arith;
11
12 architecture Behavioral of top_module_arith is
13
14 Component somador_arith
15     Port (
16        A, B: in STD_LOGIC_VECTOR(3 downto 0);
17        S:  out STD_LOGIC_VECTOR(4 downto 0)
18 );
19 end Component;
20
21 signal U1, U2: STD_LOGIC_VECTOR(3 downto 0);
22 signal U3: STD_LOGIC_VECTOR(4 downto 0);
23
24 begin
25
26 U1 <= A;
27 U2 <= B;
28
29 sum: somador_arith port map (A => U1, B => U2, S => U3);
30
31 S <= U3;
32
33 end Behavioral;
```

Figura 11: Top module - Golden Model. Fonte: Autor.

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

entity test_bench_arith is
end;

architecture behavior of test_bench_arith is

component somador_arith 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 Uaux : std_logic_vector (7 downto 0):= (others => '0');
signal U1 : std_logic_vector (3 downto 0) := (others => '0');
signal U2 : std_logic_vector (3 downto 0):= (others => '0');
signal U3 : std_logic_vector (4 downto 0):= (others => '0');

begin

U1 <= Uaux(3 downto 0);
U2 <= Uaux(7 downto 4);

somador:  somador_arith port map (A => U1, B => U2, S => U3);

estimulo: process

begin
```

```vhdl
estimulo: process

begin

for I in 0 to 255 loop
    wait for 20 ns; Uaux <= UNSIGNED (Uaux) +1;
end loop;
wait;
end process;

end behavior;
```

Figura 12: Test bench - Golden Model. Fonte: Autor.

E por fim, foi criado um aquivo de teste para simular todas as possíveis entradas do circuito:

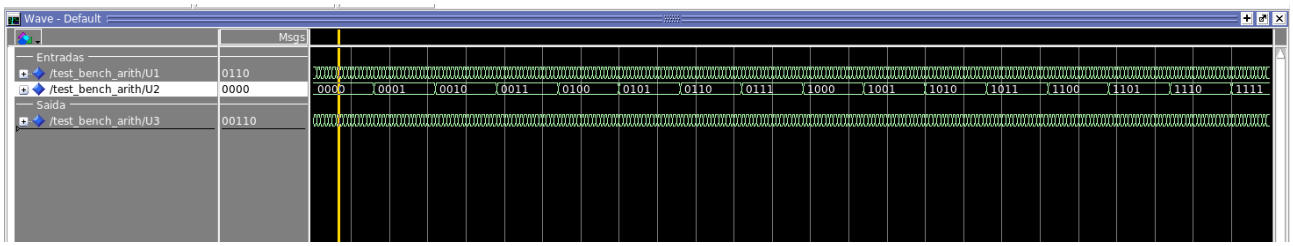

Figura 13: Casos de Teste - Golden Model. Fonte: Autor.

Onde:

- $U1$ e $U2$: entradas de 4 bits;
- $U3$: saída de 5 bits;

4.3 Questão 3

Por últimos, vamos comparar as saídas do somador completo com nosso golden model para verificar se a implementação está certa. Para isso, foi criado um teste bench que irá comparar as duas saídas e nos retornará uma mensagem de sucesso ou erro de acordo.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity test_bench_golden is
end test_bench_golden;

architecture behavior of test_bench_golden is
  component top_module 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);
      Cout0: out STD_LOGIC;
      Cout1: out STD_LOGIC;
      Cout2: out STD_LOGIC;
      Cout3: out STD_LOGIC
    );
  end component;

  component top_module_arith 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 U1: STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
  signal U2: STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
  signal C0, C1, C2, C3: STD_LOGIC;
  signal f_dut, f_gm: STD_LOGIC_VECTOR(4 downto 0);

begin
  somador_test: top_module port map (
    A => U1,
    B => U2,
    S => f_dut,
    Cout0 => C0,
    Cout1 => C1,
    Cout2 => C2,
    Cout3 => C3
  );

  somador_golden: top_module_arith port map (
    A => U1,
    B => U2,
    S => f_gm
  );

  estimulo: process
    variable cont : STD_LOGIC_VECTOR(7 downto 0);
  begin
    cont  := "00000000";
    report "iniciando" severity NOTE;
    for i in 0 to 255 loop
      U1(0) <= cont(0);
      U1(1) <= cont(1);
      U1(2) <= cont(2);
      U1(3) <= cont(3);

      U2(0) <= cont(4);
      U2(1) <= cont(5);
      U2(2) <= cont(6);
      U2(3) <= cont(7);

      wait for 500 ns;

      assert(f_gm = f_dut) report "Falhou: i = " & integer'image(i) severity ERROR;

      cont := cont + 1;

    end loop;

    report "Teste finalizado. Sucesso!" severity NOTE;

    wait;
  end process;
end behavior;
```

Figura 14: Test bench - Golden Model. Fonte: Autor.

Realizando a simulação obtemos:

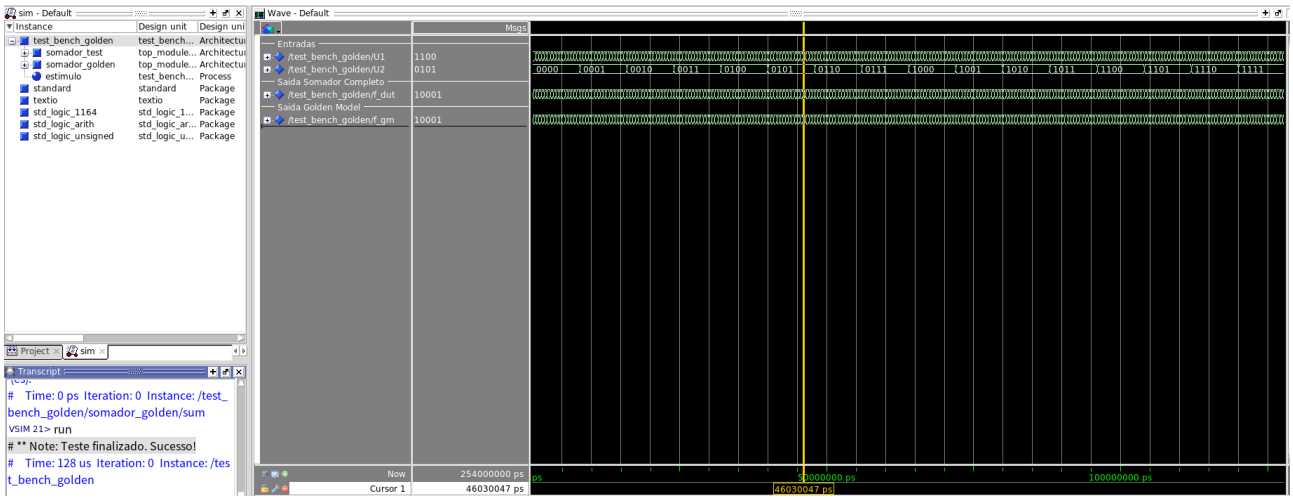

Figura 15: Simulação Final. Fonte: Autor.

Onde:

- $U1$ e $U2$: entradas de 4 bits;
- `f_dut`: saída do somador completo;

- `f_gm`: saída do modelo de referência ("golden model");

Analisando a imagem acima é complicado visualizar se as duas saídas são iguais. Para isso, basta analisar a mensagem no terminal no canto inferior esquerdo, que indica que a simulação ocorreu com sucesso.

Dessa forma, podemos afirmar que a implementação do somador de palavras binárias com 4 somadores completos está funcionando corretamente.