Relatório 5 VHDL - Turma 05

Yan Tavares de Oliveira
202014323

Introdução

Este experimento consiste em 3 etapas. Na primeira (questão 1), iremos descrever em VHDL e simular no software ModelSim uma entidade que descreva o comportamento de um somador de palavras de 4 bits utilizando somadores completos.

Na segunda etapa (questão 2), iremos descrever em VHDL e simular no software ModelSim o mesmo somador de palavras utilizando a biblioteca STD_LOGIC_ARITH.

Finalmente, na terceira etapa (questão 3), iremos descrever em VHDL e simular no software ModelSim um tesbench para comparar ambos os resultados das etapas 1 e 2, e avisar sobre qualquer divergência encontrada.

Teoria

Nas questões 1 e 2, devemos implementar um somador de 4 bits, descrito pela imagem 1, que nos permite realizar a soma de palavras de 4 bits. O primeiro será feito manualmente, enquanto o segundo será obtido por meio da biblioteca STD_LOGIC_ARITH.

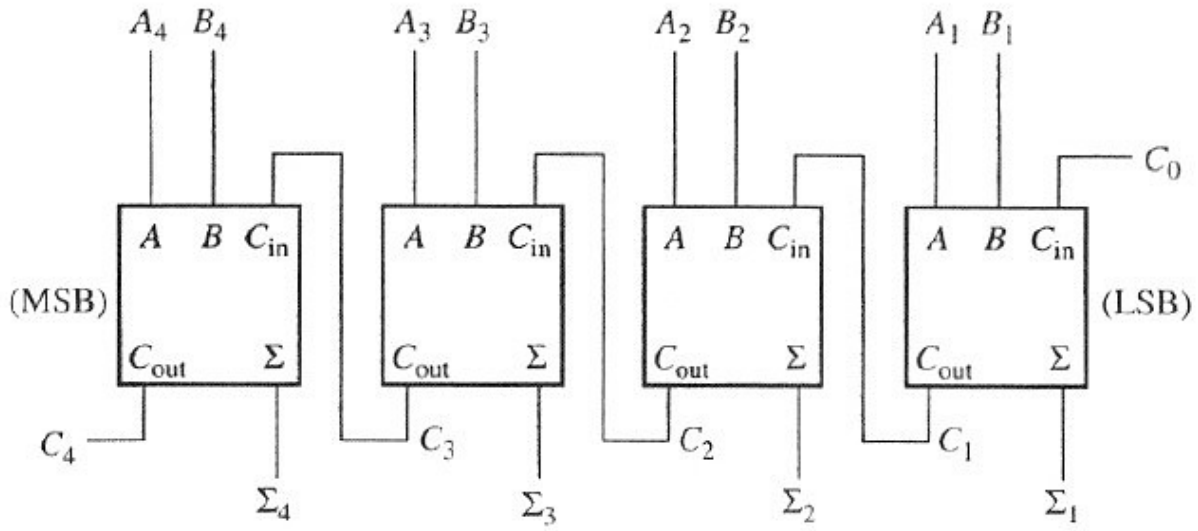


Imagem 1. Representação visual de um somador de 4 bits.

Códigos

Na questão 1, para implementar o somador de palavras, foi utilizado como base um programa que implementa um somador completo. Temos, portanto, os seguintes programas

D:/somador_completo.vhd (/top_module/U0/SC3) - Default

```
Ln#
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3
 4  entity somador_completo is
 5
 6  port (
 7      cin, a, b : IN STD_LOGIC ;
 8      s, cout : OUT STD_LOGIC
 9  );
10
11  end somador_completo;
12
13  architecture main of somador_completo is
14
15  begin
16      s <= a XOR b XOR cin ;
17      cout <= (a AND b) OR (cin AND a) OR (cin AND b) ;
18  end main;
19
```

Imagem 2. Código de somador completo

D:/somador_palavras.vhd (/top_module/U0) - Default

```
Ln#
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3
 4  entity somador_palavras is
 5  port (
 6      a, b     :    in std_logic_vector(3 downto 0);
 7      s    :    out std_logic_vector(4 downto 0)
 8  );
 9  end somador_palavras;
10
11  architecture main of somador_palavras is
12  component somador_completo port (
13      cin, a, b : IN STD_LOGIC ;
14      s, cout : OUT STD_LOGIC
15  );
16  end component;
17
18  signal auxcout    :    std_logic_vector(2 downto 0) := "000";
19
20  begin
21      SC0 : somador_completo port map( a => a(0), b => b(0), cin => '0', cout => auxcout(0), s => s(0));
22      SC1 : somador_completo port map( a => a(1), b => b(1), cin => auxcout(0), cout => auxcout(1), s => s(1));
23      SC2 : somador_completo port map( a => a(2), b => b(2), cin => auxcout(1), cout => auxcout(2), s => s(2));
24      SC3 : somador_completo port map( a => a(3), b => b(3), cin => auxcout(2), cout => s(4), s => s(3));
25  end main;
26
```

Imagem 3. Código principal da questão 1

Foi também criado um código auxiliar de testbench para gerarmos os estímulos necessários para abranger todas as combinações possíveis de entradas.

D:/tb_somador_palavras.vhd - Default *

```
Ln#
 1  library Ieee;
 2  use Ieee.std_logic_1164.all;
 3  use std.textio.all;
 4  use IEEE.STD_LOGIC_ARITH.all;
 5  use Ieee.std_logic_unsigned.all;
 6
 7  entity tb_somador_palavras is
 8  end;
 9
10  architecture behavior of tb_somador_palavras is
11
12  component exp5_1 is port (
13  A: in std_logic_vector (3 downto 0);
14  B: in std_logic_vector (3 downto 0);
15  S: out std_logic_vector (4 downto 0));
16
17  end component;
18
19  signal INPUT_signal : std_logic_vector (7 downto 0):= (others => '0');
20  signal signalAT : std_logic_vector (3 downto 0) := (others => '0');
21  signal signalBT : std_logic_vector (3 downto 0):= (others => '0');
22
23  begin
24
25  signalAT <= INPUT_signal(3) & INPUT_signal (2) & INPUT_signal (1) & INPUT_signal (0);
26
27  signalBT <= INPUT_signal(7) & INPUT_signal (6) & INPUT_signal (5) & INPUT_signal (4);
28
29  dut: exp5_1 port map (A => signalAT, B => signalBT, S => open);
30
31  estimulo: process
32
33  begin
34
35  for I in 0 to 255 loop
36  wait for 500 ns; INPUT_signal <= UNSIGNED (INPUT_signal) +1;
37  end loop;
38  wait;
39  end process;
40
41  end behavior;
```

Imagem 3. Código auxiliar da questão 1

Na questão 2, para implementar o somador de palavras utilizamos a biblioteca STD_LOGIC_ARITH.

D:/somador_arith.vhd (/top_module/U1) - Default

```
Ln#
 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
14  begin
15      s <= unsigned('0' & a) + unsigned('0' & b);
16  end main;
17
```

Imagem 4. Código principal da questão 2

Foi também criado um código auxiliar de testbench para gerarmos os estímulos necessários para abranger todas as combinações possíveis de entradas.

D:/tb_somador_arith.vhd - Default *

```
Ln#
1   library Ieee;
2   use Ieee.std_logic_1164.all;
3   use std.textio.all;
4   use IEEE.STD_LOGIC_ARITH.all;
5   use Ieee.std_logic_unsigned.all;
6
7   entity tb_somador_arith is
8   end;
9
10  architecture behavior of tb_somador_arith is
11
12  component exp5_2 is port (
13  A: in std_logic_vector (3 downto 0);
14  B: in std_logic_vector (3 downto 0);
15  S: out std_logic_vector (4 downto 0));
16
17  end component;
18
19  signal INPUT_signal : std_logic_vector (7 downto 0):= (others => '0');
20  signal signalAT : std_logic_vector (3 downto 0) := (others => '0');
21  signal signalBT : std_logic_vector (3 downto 0):= (others => '0');
22
23  begin
24
25  signalAT <= INPUT_signal(3) & INPUT_signal (2) & INPUT_signal (1) & INPUT_signal (0);
26
27  signalBT <= INPUT_signal(7) & INPUT_signal (6) & INPUT_signal (5) & INPUT_signal (4);
28
29  dut: exp5_2 port map (A => signalAT, B => signalBT, S => open);
30
31  estimulo: process
32
33  begin
34
35  for I in 0 to 255 loop
36  wait for 500 ns; INPUT_signal <= UNSIGNED (INPUT_signal) +1;
37  end loop;
38  wait;
39  end process;
40
41  end behavior;
```

Imagem 5. Código auxiliar da questão 2

Na questão 3, para implementar o testbench que compara o resultado das questões anteriores, foi criado um modelo descrito na figura abaixo

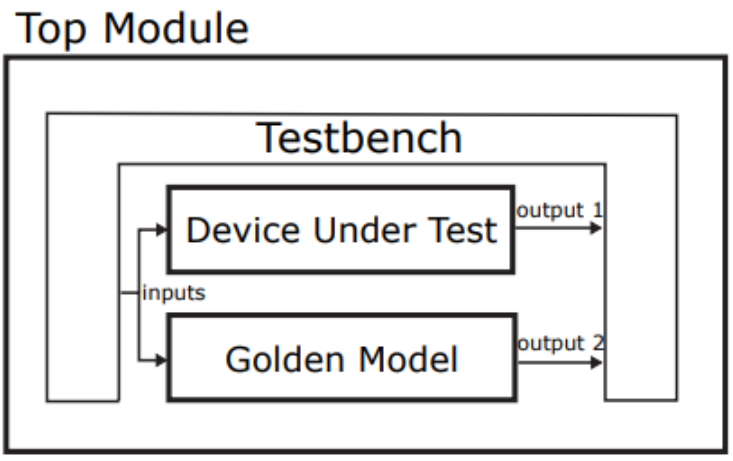


Imagem 6. Modelo de testbench “U” invertido.

Para isso, criamos um top module que unifica um arquivo de testbench, um Device Under Test (DUT) que será o arquivo da questão 1, ou seja, o que queremos testar, e um Golden Model (G_M) que será o arquivo da questão 2, ou seja, o ideal que queremos chegar.

D:/tb.vhd (/top_module/U2) - Default

```
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 tb is port (
7       s_gm, s_dut : in std_logic_vector (4 downto 0);
8       a, b        :       out std_logic_vector(3 downto 0)
9   );
10
11  end tb;
12
13  architecture test of tb is
14  begin
15          process
16                  variable cont : std_logic_vector(7 downto 0);
17          begin
18                  cont  := "00000000";
19                  report "iniciando" severity NOTE;
20                  for i in 0 to 255 loop
21                          a(0) <= cont(0);
22                          a(1) <= cont(1);
23                          a(2) <= cont(2);
24                          a(3) <= cont(3);
25
26                          b(0) <= cont(4);
27                          b(1) <= cont(5);
28                          b(2) <= cont(6);
29                          b(3) <= cont(7);
30                          wait for 500 ns;
31
32                          assert(s_gm = s_dut) report "Falhou: i = " & integer'image(i) severity ERROR;
33
34                          cont := cont + 1;
35                  end loop;
36                  report "Teste finalizado :)" severity NOTE;
37                  wait;
38          end process;
39  end test;
```

Imagem 7. Código de testbench da questão 3

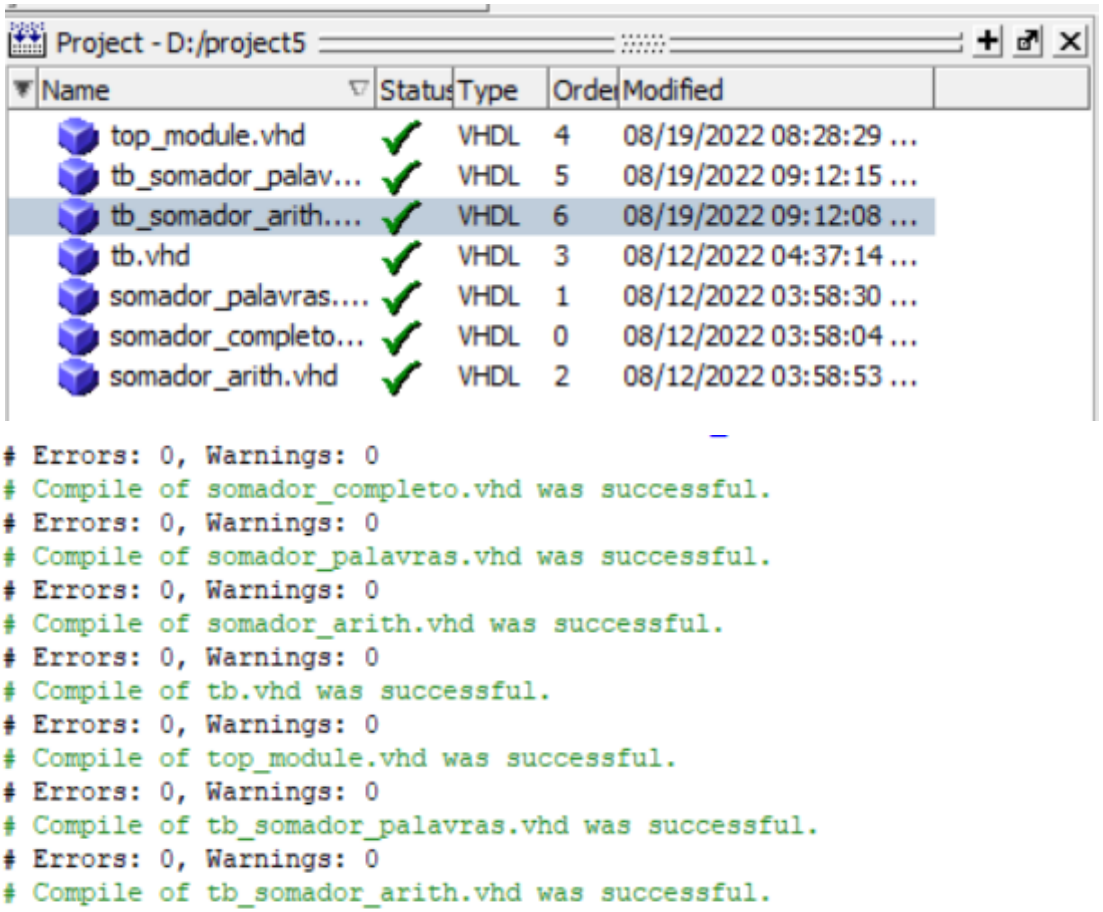
D:/top_module.vhd (/top_module) - Default

```
Ln#
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use IEEE.std_logic_arith.all;
4   entity top_module is end;
5   architecture main of top_module is
6   component somador_palavras
7   port (
8       a, b     :    in std_logic_vector(3 downto 0);
9       s    :    out std_logic_vector(4 downto 0)
10  );
11  end component;
12  component somador_arith
13  port (
14      a, b    :    in std_logic_vector(3 downto 0);
15      s    :    out std_logic_vector(4 downto 0)
16  );
17  end component;
18
19  component tb is
20  port (
21      s_dut, s_gm    :    in std_logic_vector(4 downto 0);
22      a, b        :       out std_logic_vector(3 downto 0)
23
24  );
25  end component;
26
27  signal ax, bx   :     std_logic_vector (3 downto 0);
28  signal s_dutx, s_gmx: std_logic_vector(4 downto 0);
29
30  begin
31      U0: somador_palavras port map(a => ax,b => bx,s => s_dutx);
32      U1: somador_arith port map(a => ax,b => bx,s => s_gmx);
33      U2: tb port map(s_dut => s_dutx, s_gm => s_gmx, a => ax, b => bx);
34  end main;
35
```

Imagem 8. Código de top module da questão 3

Compilação

Abaixo estão as mensagens de compilação do projeto, com nenhum erro sendo apresentado em nenhum dos casos




```
# Errors: 0, Warnings: 0
# Compile of somador_completo.vhd was successful.
# Errors: 0, Warnings: 0
# Compile of somador_palavras.vhd was successful.
# Errors: 0, Warnings: 0
# Compile of somador_arith.vhd was successful.
# Errors: 0, Warnings: 0
# Compile of tb.vhd was successful.
# Errors: 0, Warnings: 0
# Compile of top_module.vhd was successful.
# Errors: 0, Warnings: 0
# Compile of tb_somador_palavras.vhd was successful.
# Errors: 0, Warnings: 0
# Compile of tb_somador_arith.vhd was successful.
```

Imagem 9. Mensagem de compilação

Simulação

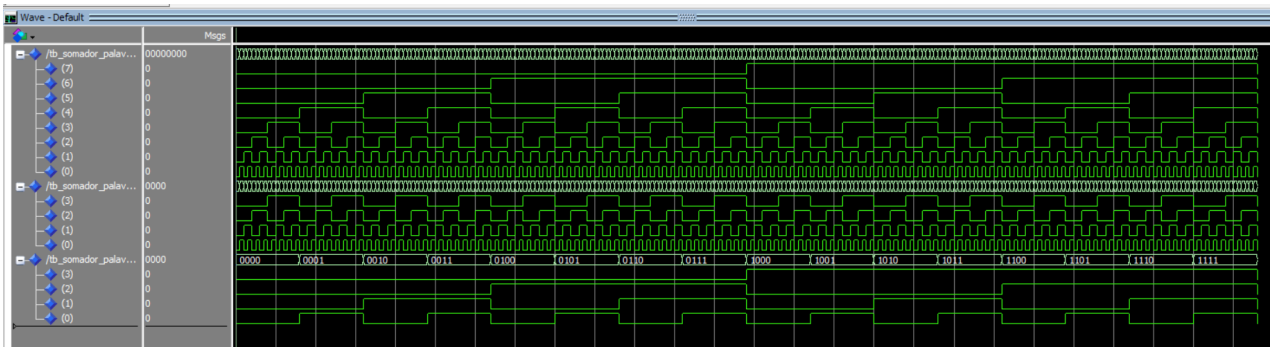


Figura 10. Simulação em forma de onda binária da questão 1.

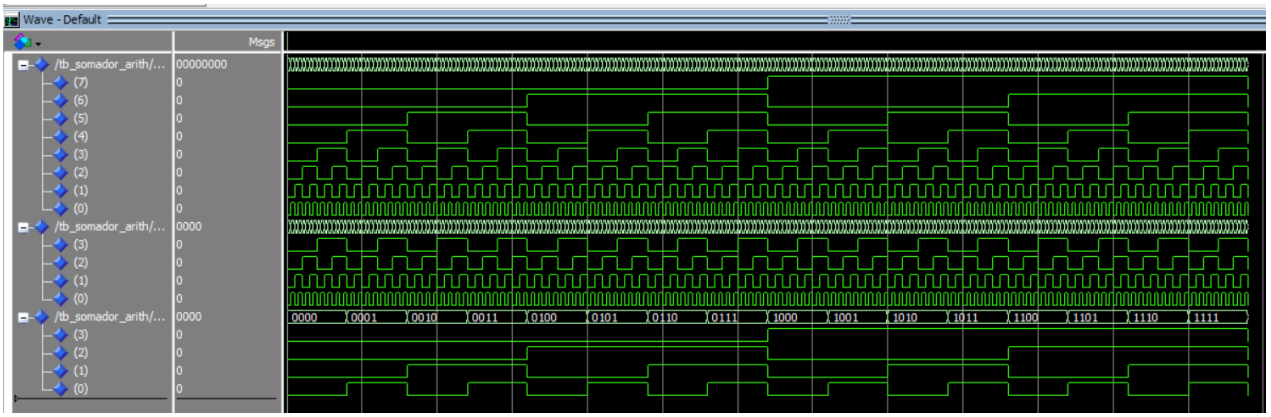
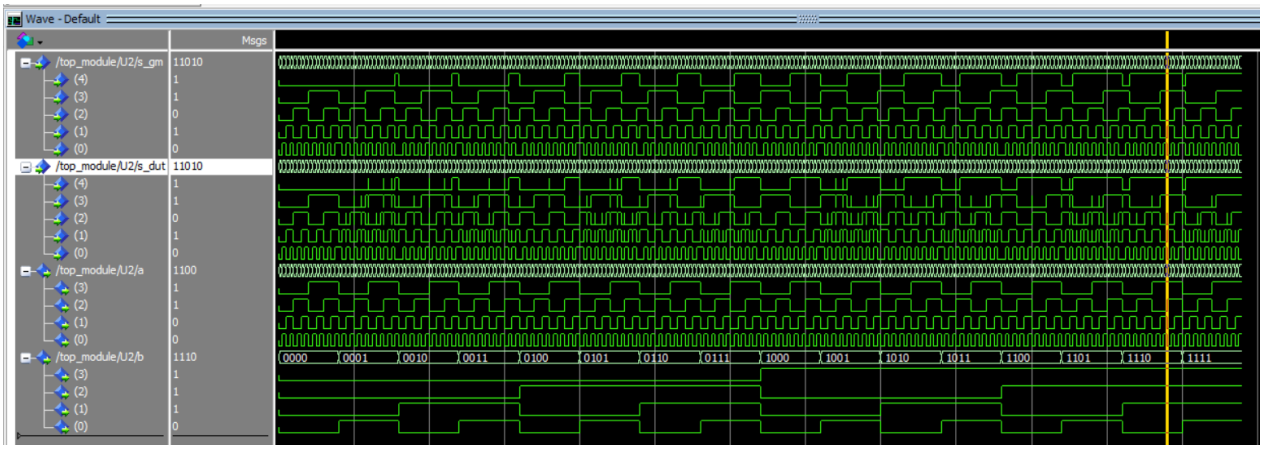


Figura 11. Simulação em forma de onda binária da questão 2.




```
# ** Note: iniciando
#    Time: 0 ps  Iteration: 0  Instance: /top_module/U2
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 0 ps  Iteration: 0  Instance: /top_module/U1
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 0 ps  Iteration: 0  Instance: /top_module/U1
# ** Note: Teste finalizado :)
#    Time: 128 us  Iteration: 0  Instance: /top_module/U2
```

Figura 12. Simulação em forma de onda binária da questão 3.

Análise

Para determinar a ordem dos bits utilizados na simulação e para abranger todas as combinações possíveis, foram utilizados loops com um tempo de espera de 500 nano segundos. Como, ao todo, são 256 combinações possíveis, a simulação durou 128 mil ns. Pelas simulações geradas, é possível notar que não obtivemos divergências em nenhuma combinação possível.

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Conclusão

Como não houve divergência alguma nos dois casos (DUT e G_M), podemos então afirmar que implementamos um somador de palavras de 4 bits funcional por meio de uma simulação no software ModelSim e utilizando como base o código de um somador completo.