

Universidade de Brasília
Departamento de Engenharia Elétrica

Relatório do Experimento 2

Laboratório de Sistemas Digitais

Autor: Henrique Morcelles Salum
Matrícula: 232003008

Brasília, 21 de outubro de 2024

Sumário

1	Introdução	2
2	Teoria	2
3	Códigos	4
4	Compilação	9
5	Simulação	9
6	Análise	10
7	Conclusão	12

1 Introdução

Este experimento é composto por duas tarefas, em ambas, deve-se implementar funções booleanas com VHDL e simular os circuitos lógicos implementados por meio do software ModelSim, da Intel. As especificidades de cada tarefa serão exploradas a seguir.

Na primeira, implementa-se um somador completo. Um somador completo é um circuito lógico combinacional que realiza a soma de três bits: dois bits de entrada, A e B , e um bit adicional, C (tipicamente chamado de C_{in} , carry-in), que representa o "vai-um" (carry-out) de uma operação anterior. Ele gera duas saídas: S , o resultado da soma dos três bits, e T (tipicamente chamado de C_{out} , carry-out), o "vai-um" que pode ser gerado por essa soma. Para isso, definimos na entidade somadorCompleto os sinais de entrada A , B e C_{in} e os sinais de saída S e C_{out} , que concatenados representam a soma dos três bits de entrada - de acordo com as definições a seguir:

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = A \cdot B + A \cdot C_{in} + B \cdot C_{in}.$$

O valor dos sinais de saída são definidos na "architecture".

Na segunda (Questão 2), deve-se implementar um multiplexador de 4 para 1, utilizando dois vetores de entrada: S , de dois sinais e D , de quatro sinais. A saída é um sinal Y definido como segue:

$$Y = D_3 \cdot \overline{S_1} \cdot \overline{S_0} + D_2 \cdot \overline{S_1} \cdot S_0 + D_1 \cdot S_1 \cdot \overline{S_0} + D_0 \cdot S_1 \cdot S_0$$

2 Teoria

O somador completo é uma função booleana usada para realizar a adição de dois números binários. Essa função tem três entradas binárias: os dois bits de uma posição específica na representação binária dos números somados e um terceiro bit, que representa o carry-in (por isso C_{in}), que pode vir da adição em uma posição de ordem inferior ($1 + 1 = 10$, e "vai-um" para a soma na próxima posição), além de duas saídas: S , que representa o resultado da soma dos três bits de entrada e C_{out} , que representa o carry-out, o bit gerado para a adição na próxima posição do número binário (adição em que será o C_{in}).

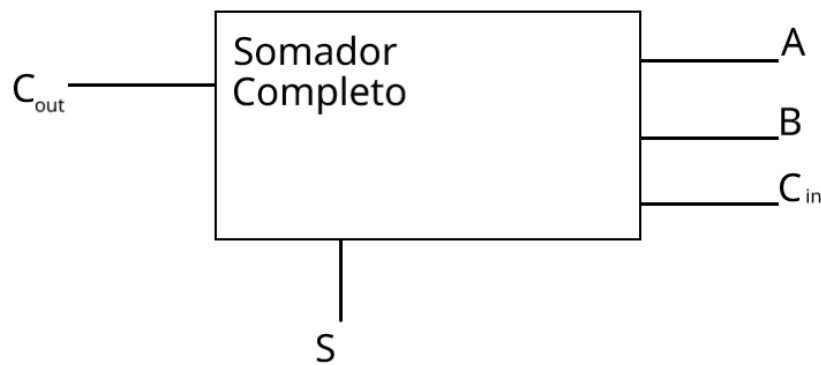


Figura 1: Representação do somador completo

Um multiplexador é um dispositivo digital que seleciona uma entre várias entradas de dados e a encaminha para uma única linha de saída. Essencialmente, o multiplexador atua como um "comutador" controlado por uma entrada de seleção, determinando qual sinal será encaminhado para a saída.

Para entender esse dispositivo, podemos separar as entradas entre entradas de dados, que são as entradas de informações a serem selecionadas, e as entradas de seleção, que são utilizadas para escolher qual das entradas de dados será enviada para a saída. O número de entradas de seleção pode ser calculado em função do número de entradas de dados pela função:

$$f(n) = \lceil \log_2 n \rceil$$

Nesse experimento, implementa-se, especificamente, um multiplexador 4 x 1, ou seja, com quatro entradas de dados e, consequentemente, $\log_2 4 = 2$ entradas de seleção, além de um sinal de saída.

S_1	S_0	Saída
0	0	D_3
0	1	D_2
1	0	D_1
1	1	D_0

Tabela 1: Tabela-verdade multiplexador 4x1

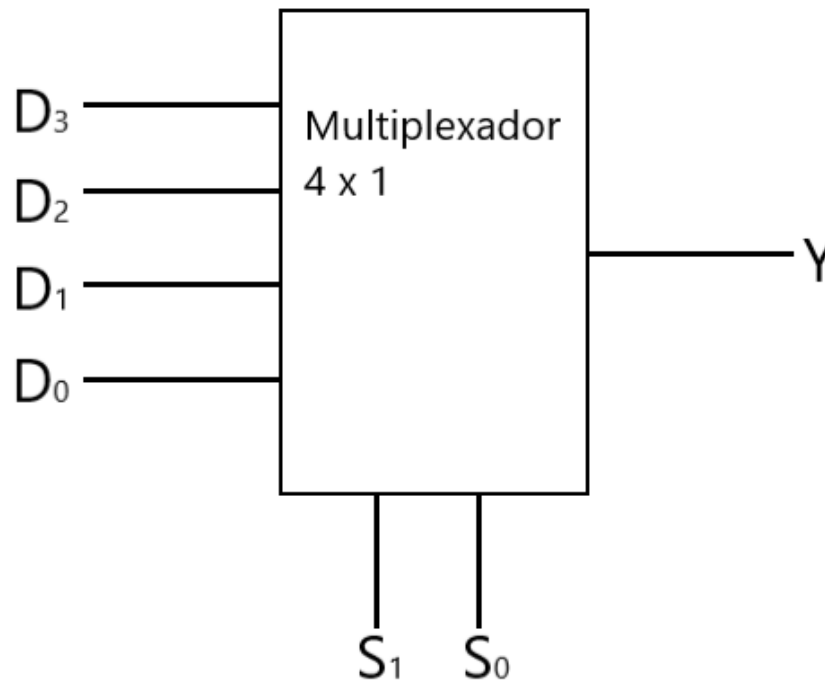


Figura 2: Representação do multiplexador 4 x 1

3 Códigos

Para implementar o somador completo descrito anteriormente, foi feito o código a seguir em VHDL:

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity somadorCompleto is
5     port (
6         A, B, Cin : in std_logic;
7         S, Cout : out std_logic
8     );
9 end somadorCompleto;
10
11 architecture main of somadorCompleto is
12 begin
13
14     S <= A xor B xor Cin;
15     Cout <= (A and B) or (A and Cin) or (B and Cin);
16
```

```
17 end architecture main;
```

Código 1: Código para implementação do somador completo

Além do código acima, foi necessário implementar, também em VHDL, um testbench, para que se pudesse testar o comportamento e a funcionalidade do circuito digital projetado. Com ele, podemos gerar todas as combinações de sinais possíveis e analisar o comportamento do circuito sob cada um desses estímulos. O código para implementar esse testbench é o que segue:

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity tb_somadorCompleto is
5  end tb_somadorCompleto;
6
7  architecture main of tb_somadorCompleto is
8
9      component somadorCompleto is
10         port (
11             A, B, Cin : in std_logic;
12             S, Cout : out std_logic
13         );
14     end component somadorCompleto;
15
16     signal A_tb : std_logic := '0';
17     signal B_tb : std_logic := '0';
18     signal Cin_tb : std_logic := '0';
19     signal S_tb, Cout_tb : std_logic;
20
21 begin
22     instancia_somador : component somadorCompleto
23
24         port map (
25             A => A_tb,
26             B => B_tb,
27             Cin => Cin_tb,
28             S => S_tb,
29             Cout => Cout_tb
30         );
```

```

31
32
33     combinacoes : process
34         variable i : integer := 1;
35     begin
36         if (i mod 2) = 0 then
37             A_tb <= not A_tb;
38         end if;
39         if (i mod 4) = 0 then
40             B_tb <= not B_tb;
41         end if;
42         if (i mod 8) = 0 then
43             Cin_tb <= not Cin_tb;
44         end if;
45
46         i := i + 1;
47         wait for 5 ns;
48     end process combinacoes;
49
50 end architecture main;

```

Código 2: Testbench para o somador completo

Como pode ser visto, nos códigos relativos ao somador completo, a abordagem foi bastante direta: a função foi implementada diretamente pela equação correspondente e o *testbench* gerou todas as oito possíveis combinações das três entradas. A abordagem para o multiplexador foi diferente, como será observado nos códigos subsequentes:

```

1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity Mux4x1 is
5     port (
6         D : in std_logic_vector(0 to 3);
7         S : in std_logic_vector(0 to 1);
8         Y : out std_logic
9     );
10 end entity Mux4x1;
11
12 architecture main of Mux4x1 is

```

```
13
14 begin
15
16     Y <= D(3) when (S = "00") else
17         D(2) when (S = "10") else
18         D(1) when (S = "01") else
19         D(0) when (S = "11") else
20         '-';
21
22 end architecture main;
```

Código 3: Código para implementação do multiplexador 4x1

Aqui, ao invés de implementar a função pela equação correspondente, foi utilizada uma construção de controle de fluxo, em que apenas se definiu o comportamento do circuito para as combinações das entradas de seleção, além de se ter considerado apenas alguns valores (0 e 1) para elas, tratando as outras possibilidades (X, Z, H, ...) como *dont't care* (-). O testbench para esse circuito segue abaixo.

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity tb_Mux4x1 is
5 end entity tb_Mux4x1;
6
7 architecture main of tb_Mux4x1 is
8
9     component Mux4x1 is
10         port (
11             D : in std_logic_vector(0 to 3);
12             S : in std_logic_vector(0 to 1);
13             Y : out std_logic
14         );
15     end component Mux4x1;
16
17     signal D_tb : std_logic_vector(0 to 3) := (others => '1');
18     signal S_tb : std_logic_vector(0 to 1) := (others => '0');
19     signal Y_tb : std_logic;
20
21 begin
```



```
22
23     instancia_Mux4x1 : component Mux4x1
24
25         port map (
26             D => D_tb,
27             S => S_tb,
28             Y => Y_tb
29         );
30
31         variable i : integer := 0;
32     begin
33         if i /= 0 then
34             if (i mod 2) = 0 then
35                 S_tb(0) <= not S_tb(0);
36             end if;
37             if (i mod 4) = 0 then
38                 S_tb(1) <= not S_tb(1);
39             end if;
40         end if;
41
42         i := i + 1;
43         wait for 12.5 ns;
44     end process estimulos;
45
46     begin
47         wait for 12.5 ns;
48         case S_tb is
49             when "00" =>
50                 D_tb(3) <= not D_tb(3);
51             when "10" =>
52                 D_tb(2) <= not D_tb(2);
53             when "01" =>
54                 D_tb(1) <= not D_tb(1);
55             when "11" =>
56                 D_tb(0) <= not D_tb(0);
57             when others =>
58                 D_tb <= D_tb;
59         end case;
60     end process;
```

```

61
62 end architecture main;

```

Código 4: Testbench para o multiplexador 4x1

Ao contrário do *testbench* anterior, aqui não são geradas todas as combinações possíveis dos sinais de entrada. Isso se deve à natureza do multiplexador: nele, é irrelevante a variação dos sinais não selecionados em um momento da simulação. Portanto, basta que se gere as combinações das entradas de seleção e se verifique se o sinal de saída corresponde à entrada de dados que deve ser selecionado por essa combinação. Para que essa simulação fosse mais completa, fez-se com que os sinais de dados selecionados em um momento da simulação assumissem os dois estados possíveis (0 e 1).

4 Compilação

Após escrever os códigos, é necessário compilá-los pelo ModelSim, para que se possa simular os circuitos lógicos. A seguir, estão as mensagens de compilação dos códigos apresentados acima, sem nenhum erro, como pode ser visto no terminal no canto inferior da figura.

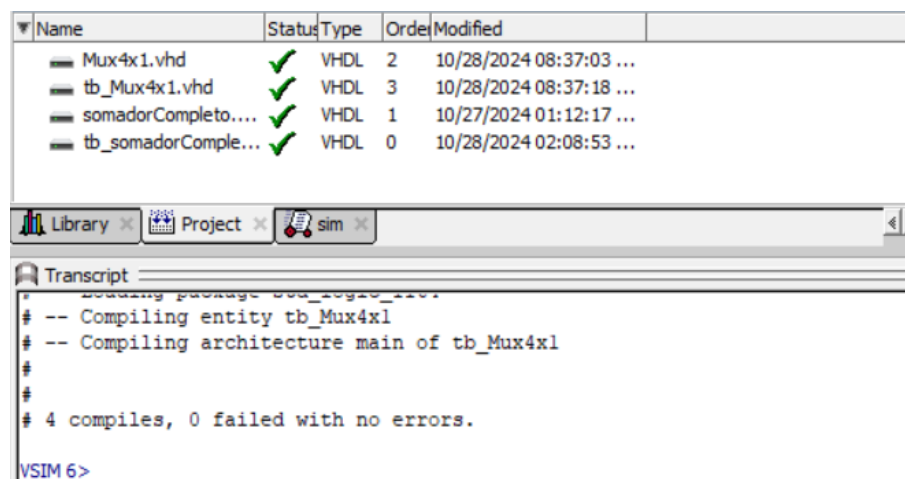


Figura 3: Compilação do somador completo e seu testbench

5 Simulação

Após compilar com sucesso os códigos apresentados, deve-se utilizar o ModelSim para testar o funcionamento dos circuitos lógicos correspondentes a eles. Lá, pode-se utilizar a aba "Waves" para analisar o comportamento dos sinais de saída para cada combinação de valores dos sinais de entrada.

Seguem os prints dos resultados das simulações no ModelSim dos códigos apresentados anteriormente:

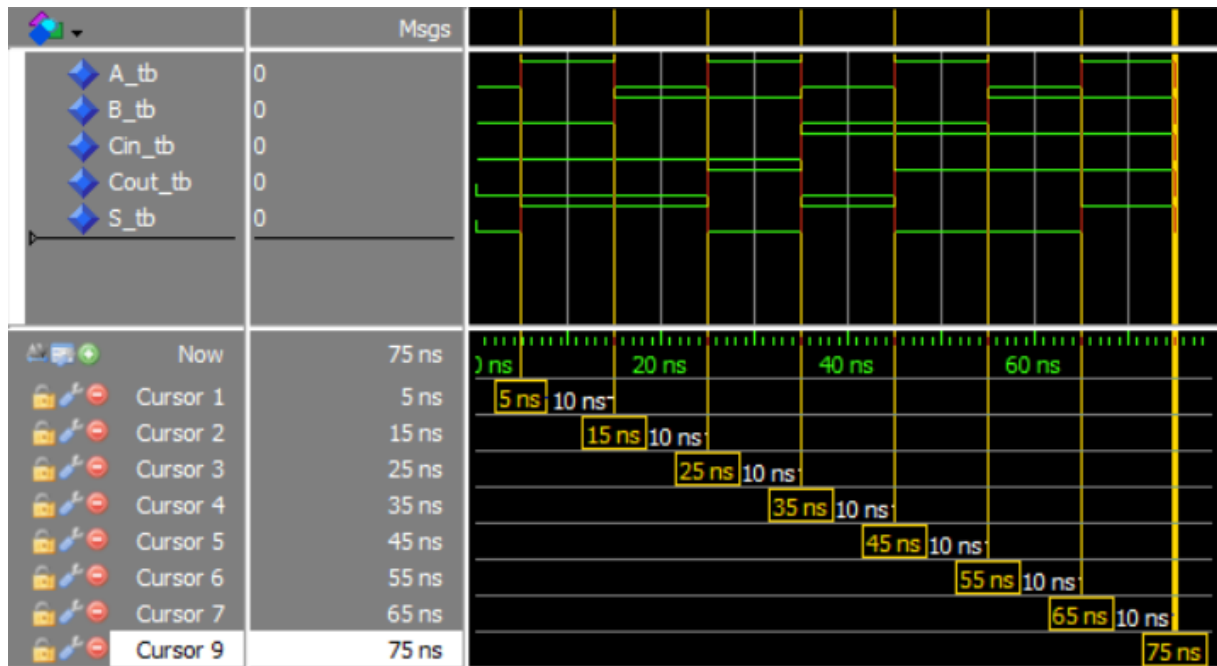


Figura 4: Simulação em forma de onda binária do somador completo

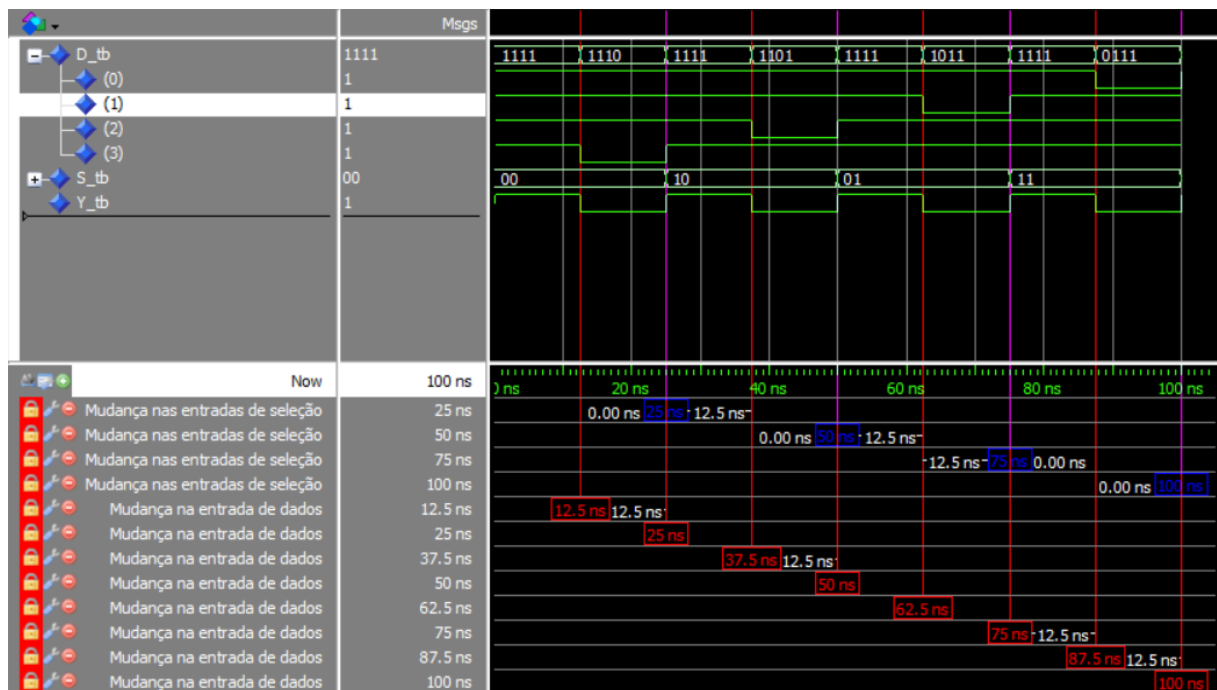


Figura 5: Simulação em forma de onda binária do multiplexador

6 Análise

Para analisar os resultados obtidos, precisamos apenas mover o cursor na aba "Wave" do ModelSim, assim, conseguimos analisar o valor dos sinais de saída para cada combinação de valores dos sinais de entrada e verificar se os resultados obtidos são os esperados.

A fio, estão exibidos os prints que denotam o comportamento dos sinais de saída para cada possível de estímulo de entrada para o circuito do somador completo.



















◆ A_tb	0	◆ A_tb	1
◆ B_tb	0	◆ B_tb	0
◆ Cin_tb	0	◆ Cin_tb	0
◆ Cout_tb	0	◆ Cout_tb	0
◆ S_tb	0	◆ S_tb	1
◆ A_tb	0	◆ A_tb	1
◆ B_tb	1	◆ B_tb	1
◆ Cin_tb	0	◆ Cin_tb	0
◆ Cout_tb	0	◆ Cout_tb	1
◆ S_tb	1	◆ S_tb	0
◆ A_tb	0	◆ A_tb	1
◆ B_tb	0	◆ B_tb	0
◆ Cin_tb	1	◆ Cin_tb	1
◆ Cout_tb	0	◆ Cout_tb	1
◆ S_tb	1	◆ S_tb	0
◆ A_tb	0	◆ A_tb	1
◆ B_tb	1	◆ B_tb	1
◆ Cin_tb	1	◆ Cin_tb	1
◆ Cout_tb	1	◆ Cout_tb	1
◆ S_tb	0	◆ S_tb	1

Figura 6: Todas as combinações possíveis de entradas e as saídas correspondentes

Percebe-se que o comportamento está de acordo com o esperado: a concatenação das saídas C_{out} e S corresponde à soma dos bits de entrada A , B e C_{in} .

Para analisar os resultados para o multiplexador, observou-se o valor da saída Y para cada combinação das entradas de seleção, bem como para cada possível valor da entrada de dados selecionada por cada uma dessas combinações. Abaixo, estão exibidos os prints do comportamento do circuito simulado pelo ModelSim para cada uma dessas combinações. Perceba que cada linha representa uma combinação das entradas de seleção - portanto, uma entrada de dados selecionada - e cada coluna representa um valor para o sinal de entrada de dados selecionado.

◆ D_tb	1111	◆ D_tb	1110
◆ D_tb(0)	1	◆ D_tb(0)	1
◆ D_tb(1)	1	◆ D_tb(1)	1
◆ D_tb(2)	1	◆ D_tb(2)	1
◆ D_tb(3)	1	◆ D_tb(3)	0
◆ S_tb	00	◆ S_tb	00
◆ Y_tb	1	◆ Y_tb	0
◆ D_tb	1111	◆ D_tb	1101
◆ D_tb(0)	1	◆ D_tb(0)	1
◆ D_tb(1)	1	◆ D_tb(1)	1
◆ D_tb(2)	1	◆ D_tb(2)	0
◆ D_tb(3)	1	◆ D_tb(3)	1
◆ S_tb	10	◆ S_tb	10
◆ Y_tb	1	◆ Y_tb	0

  D_tb	1111	  D_tb	1011
 D_tb(0)	1	 D_tb(0)	1
 D_tb(1)	1	 D_tb(1)	0
 D_tb(2)	1	 D_tb(2)	1
 D_tb(3)	1	 D_tb(3)	1
  S_tb	01	  S_tb	01
 Y_tb	1	 Y_tb	0



















  D_tb	1111	  D_tb	0111
 D_tb(0)	1	 D_tb(0)	0
 D_tb(1)	1	 D_tb(1)	1
 D_tb(2)	1	 D_tb(2)	1
 D_tb(3)	1	 D_tb(3)	1
  S_tb	11	  S_tb	11
 Y_tb	1	 Y_tb	0

Tabela 2: Saídas para todas as combinações das entradas de seleção e selecionadas

Nota-se, em especial, os sinais destacados em cada coluna - sempre o sinal de saída Y e o sinal de entrada selecionado D_n . É fácil perceber que aquele é igual a este para qualquer combinação das entradas de seleção e valor do sinal selecionado, ou seja, que a saída é igual ao sinal selecionado, como é, por definição, um multiplexador.

7 Conclusão

Nesse experimento, simulou-se, com sucesso, dois dispositivos digitais de suma importância para a eletrônica digital: o somador completo e o multiplexador 4-para-1. No relatório, percebeu-se o sucesso do experimento e pôde-se analisar devidamente os resultados obtidos pelas simulações, por meio das quais foi possível analisar minuciosamente esses dois circuitos e o seu comportamento para cada estímulo possível.

Concluindo, não foram observadas discrepâncias entre o comportamento esperado e o observado, podendo-se concluir, portanto, o sucesso do experimento.