

ENE0040 - Laboratório de Sistemas Digitais - Turma 07

Relatório do Experimento 3

Autor: Henrique Morcelles Salum

Matrícula: 232003008

Universidade de Brasília - UnB

Departamento de Engenharia Elétrica - ENE

Sumário

1	Introdução	2
1.1	Sobre o Experimento	2
1.1.1	Multiplexador 8 para 1	2
1.1.2	Decodificador 4 para 16	2
1.2	Introdução Teórica	3
1.2.1	Multiplexador 8 para 1	3
1.2.2	Decodificador 4 para 16	4
2	Códigos	5
2.1	Multiplexador 8 para 1	5
2.1.1	Arquivo de Design VHDL	5
2.1.2	Testbench	6
2.2	Decodificador 4 para 16	7
2.2.1	Arquivo de Design VHDL	7
2.2.2	Testbench	8
3	Compilação	10
4	Simulação	10
5	Análise	11
5.1	Análise do comportamento do Somador Completo	12
5.2	Análise do comportamento do Multiplexador 4 para 1	12
6	Conclusão	14

1 Introdução

1.1 Sobre o Experimento

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

1.1.1 Multiplexador 8 para 1

Na primeira tarefa, devemos escrever, utilizando atribuições condicionais *when-else*, uma entidade com dois vetores de entrada S , de três bits, e D , de oito bits, e um sinal de saída Y que descreva um Mux 8x1. A tabela-verdade do sistema é a que segue:

S_2	S_1	S_0	Saída
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Tabela 1: Tabela-verdade do multiplexador 8x1

1.1.2 Decodificador 4 para 16

Na segunda tarefa, a entidade que devemos implementar em VHDL, utilizando atribuições seletivas *with-select*, é um decodificador 4 para 16. Ele tem um vetor de entrada A , de quatro bits, e um vetor de saída Y de 16 bits. A tabela-verdade é a que segue:

Entrada				Saída																
A_3	A_2	A_1	A_0	Y_{15}	Y_{14}	Y_{13}	Y_{12}	Y_{11}	Y_{10}	Y_9	Y_8	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	Hex
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001h
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002h
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0004h
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0008h
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0010h
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0020h
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0040h
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0080h
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0100h
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0200h
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0400h
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0800h
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000h
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000h
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000h
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000h

Tabela 2: Tabela-verdade do Decodificador 4 para 16

Note que, na Tabela 2, há uma coluna para a representação em hexadecimal da saída. Quando lidamos com representações binárias muito grandes, é conveniente utilizar essa representação pois, se um número em sua representação binária tem n casas, em sua representação hexadecimal ele terá $\log_2 n$. Além disso, a conversão de binário para hexadecimal é muito fácil: cada algarismo da representação hexadecimal é substituído pela sua representação binária - que tem quatro bits, já que o algarismo hexadecimal de maior valor é o F_{16} , que tem valor $15_{10} = 1111_2$ - e pronto! Representações hexadecimais serão utilizadas nos códigos em VHDL nesse experimento.

1.2 Introdução Teórica

1.2.1 Multiplexador 8 para 1

Um multiplexador, também chamado de Mux, é um dispositivo digital que seleciona uma entre várias entradas de dados como saída. Essencialmente, o multiplexador atua como um “comutador” controlado por entradas de seleção, determinando qual sinal será encaminhado para a saída. Esse dispositivo é extensivamente usado no projeto de máquinas de estados, pois ele é muito conveniente quando é necessário implementar diferentes lógicas a depender do estado atual.

Para entender esse dispositivo, podemos separar as entradas entre entradas de dados, que são 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 n pela função:

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

Nesse experimento, implementamos, especificamente, um multiplexador 8 para 1, ou seja, um multiplexador com oito entradas de dados e, conseqüentemente, $\log_2 8 = 3$ entradas de seleção, além de um sinal de saída.

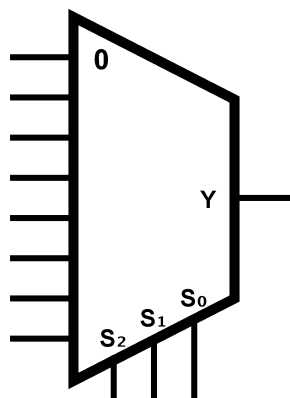


Figura 1: Representação externa do multiplexador 8 x 1

1.2.2 Decodificador 4 para 16

O decodificador, chamado de Dec, é um circuito combinatório com n bits de entrada e 2^n bits de saída que, para cada combinação dos bits de entrada, ativa apenas um bit de saída. Por convenção, o bit ativado costuma estar na posição representada pelos bits de entrada. Nesse experimento, implementamos, especificamente, o Dec 4x16.

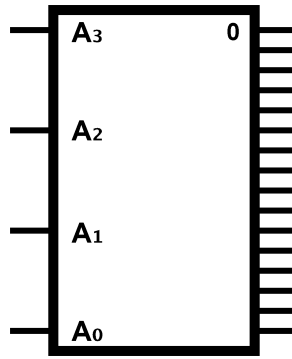
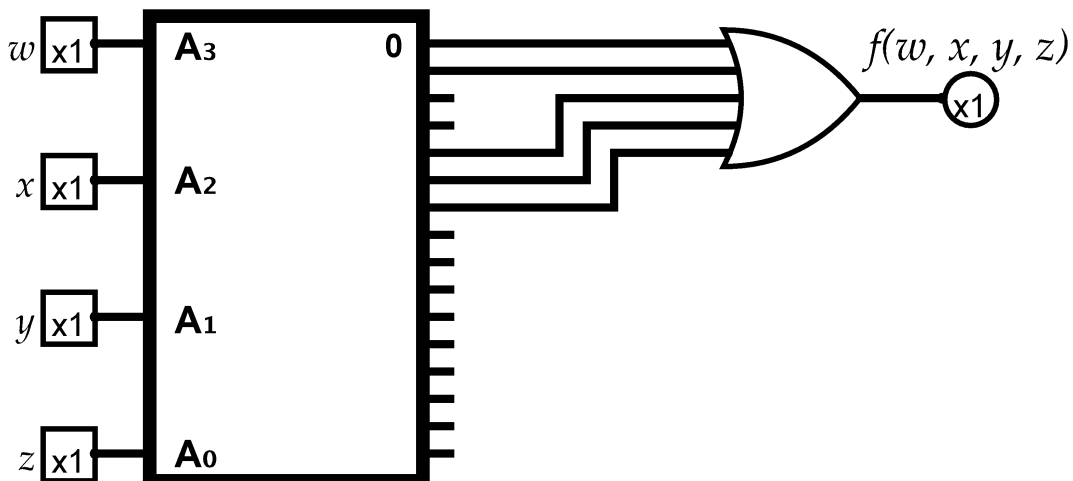


Figura 2: Representação externa do decodificador 4 x 16

Olhando para a Tabela 2, podemos perceber que cada saída do decodificador representa uma combinação dos n bits de entrada, ou seja, um *mintermo* dos bits de entrada. Esse fato justifica o interesse no decodificador: desde que toda função pode ser decomposta na soma dos mintermos que representam as linhas da tabela-verdade que essa função ativa, podemos fazer um *OR* entre saídas específicas do Dec para gerar qualquer função booleana. Por exemplo:

$$\begin{aligned} f(w, x, y, z) &= \bar{w} \bar{x} \bar{y} \bar{z} + \bar{w} \bar{x} \bar{y} z + \bar{w} x \bar{y} \bar{z} + \bar{w} x \bar{y} z + \bar{w} x y \bar{z} \\ &= \sum_{wxyz} m(0, 1, 4, 5, 6) \end{aligned}$$

Podemos implementar $f(w, x, y, z)$ da seguinte forma:



2 Códigos

Nessa seção, serão apresentados os códigos desenvolvidos, em VHDL, para implementar os sistemas digitais apresentados na introdução. Aqui, alguns comentários foram suprimidos para melhor apresentação, mas os códigos enviados junto a este relatório estão todos devidamente comentados.

2.1 Multiplexador 8 para 1

2.1.1 Arquivo de Design VHDL

O código escrito para implementar *design* do Mux 8x1, utilizando atribuições condicionais *when-else* é o que segue:

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity Mux8x1 is
5     port (
6         D: in std_logic_vector(7 downto 0);
7         S: in std_logic_vector(2 downto 0);
8         Y: out std_logic
9     );
10 end entity Mux8x1;
11
12 architecture behavioral of Mux8x1 is
13 begin
14     Y <= D(7) when (S = "111") else
15         D(6) when (S = "110") else
16         D(5) when (S = "101") else
17         D(4) when (S = "100") else
18         D(3) when (S = "011") else
19         D(2) when (S = "010") else
20         D(1) when (S = "001") else
21         D(0) when (S = "000") else
22         '-';
23 end architecture behavioral;
```

Código 1: Código de *design* do multiplexador 8 para 1

2.1.2 Testbench

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 interesse dos sinais de entrada 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  use ieee.numeric_std.all;
4
5  entity tb_Mux8x1 is
6  end entity tb_Mux8x1;
7
8  architecture main of tb_Mux8x1 is
9      component Mux8x1 is
10         port (
11             D: in std_logic_vector(7 downto 0);
12             S: in std_logic_vector(2 downto 0);
13             Y: out std_logic
14         );
15     end component Mux8x1;
16     signal D_tb: std_logic_vector(7 downto 0) := (others => '1');
17     signal S_tb: std_logic_vector(2 downto 0) := (others => '0');
18 begin
19     instancia_Mux8x1: component Mux8x1
20         port map (
21             D => D_tb,
22             S => S_tb,
23             Y => open
24         );
25     -- Processo para variar as entradas de seleção
26     estimulos_seletoras: process
27         variable s_unsigned: unsigned(2 downto 0);
28     begin
29         wait for 12.5 ns;
30         s_unsigned := unsigned(S_tb);
31         s_unsigned := s_unsigned + 1;
32         S_tb <= std_logic_vector(s_unsigned);
```

```
33     end process estimulos_seletoras;
34     -- Processo para variar as entradas selecionadas
35     variacao_dados: process
36         variable i: integer;
37     begin
38         wait for 6.25 ns;
39         i := to_integer(unsigned(S_tb));
40         D_tb(i) <= not D_tb(i);
41         wait on S_tb;
42     end process variacao_dados;
43 end architecture main;
```

Código 2: Testbench do multiplexador 8 para 1

No código acima, o multiplexador construído é instanciado e utilizado como uma “caixa-preta”; associamos às suas entradas sinais do *testbench*, o que é equivalente a conectar cabos nas entradas de um dispositivo digital, e, dentro do *process*, fazemos esses sinais oscilarem. Dessa forma, conseguimos testar o dispositivo criado.

2.2 Decodificador 4 para 16

Como pôde ser visto, nos códigos relativos ao somador completo, a abordagem foi bastante de baixo nível: não utilizamos abstrações fornecidas pela linguagem VHDL, 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.

2.2.1 Arquivo de Design VHDL

O código desenvolvido em VHDL para implementar a “caixa-preta” do multiplexador 4 para 1 é o que segue:

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity Dec4x16 is
5     port (
6         A: in std_logic_vector(3 downto 0);
7         Y: out std_logic_vector(15 downto 0)
8     );
9 end entity Dec4x16;
```



```
10
11 architecture behavioral of Dec4x16 is
12 begin
13     with A select Y <=
14         X"0001" when X"0",
15         X"0002" when X"1",
16         X"0004" when X"2",
17         X"0008" when X"3",
18         X"0010" when X"4",
19         X"0020" when X"5",
20         X"0040" when X"6",
21         X"0080" when X"7",
22         X"0100" when X"8",
23         X"0200" when X"9",
24         X"0400" when X"A",
25         X"0800" when X"B",
26         X"1000" when X"C",
27         X"2000" when X"D",
28         X"4000" when X"E",
29         X"8000" when X"F",
30         "-----" when others;
31 end architecture behavioral;
```

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 ‘when’, 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, fazendo com que as outras possibilidades - X, Z, H, ... - gerassem saída *don't care* (-). O *testbench* do multiplexador é o que segue.

Analogamente ao Somador Completo, no multiplexador 4 para 1, a entidade (*entity*) é equivalente à representação da Figura 1 e a arquitetura (*architecture*), à da ??.

2.2.2 Testbench

O código referente ao *testbench* do multiplexador é o que segue:

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

```
3 use IEEE.numeric_std.all;
4
5 entity tb_Dec4x16 is
6 end entity tb_Dec4x16;
7
8 architecture testbench of tb_Dec4x16 is
9     component Dec4x16 is
10         port (
11             A: in std_logic_vector(3 downto 0);
12             Y: out std_logic_vector(15 downto 0)
13         );
14     end component Dec4x16;
15     signal A_tb: std_logic_vector(3 downto 0) := (others => '0'); --
        ↳ Sinal interno do testbench
16 begin
17     instancia_Dec4x16: component Dec4x16
18         port map (
19             A => A_tb,
20             Y => open
21         );
22
23     estimulos: process
24         variable Uns_A: unsigned(3 downto 0);
25     begin
26         wait for 6.25 ns;
27         Uns_A := unsigned(A_tb);
28         A_tb <= std_logic_vector(Uns_A + 1);
29     end process estimulos;
30 end architecture testbench;
```

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 geremos todas as combinações das entradas de seleção e se verifique se o sinal de saída corresponde à entrada de dados que deve ser selecionada por essa combinação. Para que essa simulação seja mais completa, fez-se com que os sinais de dados selecionados em um momento da simulação oscilassem entre 0 e 1.

3 Compilação

Após escrever os códigos, é necessário compilá-los pelo ModelSim para que se possa simular os sistemas digitais discutidos. Caso a compilação tenha sucesso, sabemos que não houve erros nos códigos apresentados, mas ainda não podemos afirmar que a lógica para implementar os circuitos está correta; isso será analisado nas próximas seções. A seguir, está a mensagem de compilação dos códigos apresentados acima, sem nenhum erro, como pode ser visto no terminal no canto inferior da figura.

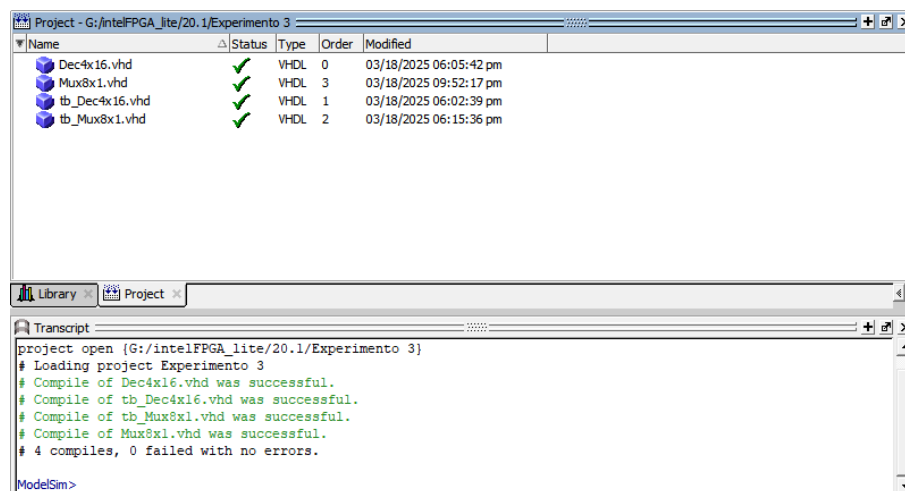


Figura 3: Compilação de todos os códigos apresentados

4 Simulação

Após compilar com sucesso os códigos apresentados, utilizamos o ModelSim para simular o comportamento dos sistemas descritos por eles. Lá, utilizamos a aba “Waves” para analisar o comportamento dos sinais de saída para cada combinação de valores dos sinais de entrada. A forma como as entradas variam segue o que definimos nos *testbenches*, então note que as ondas referentes ao Mux 4x1 não representam todas as combinações das seis entradas. Seguem as imagens das simulações no ModelSim dos sistemas projetados.

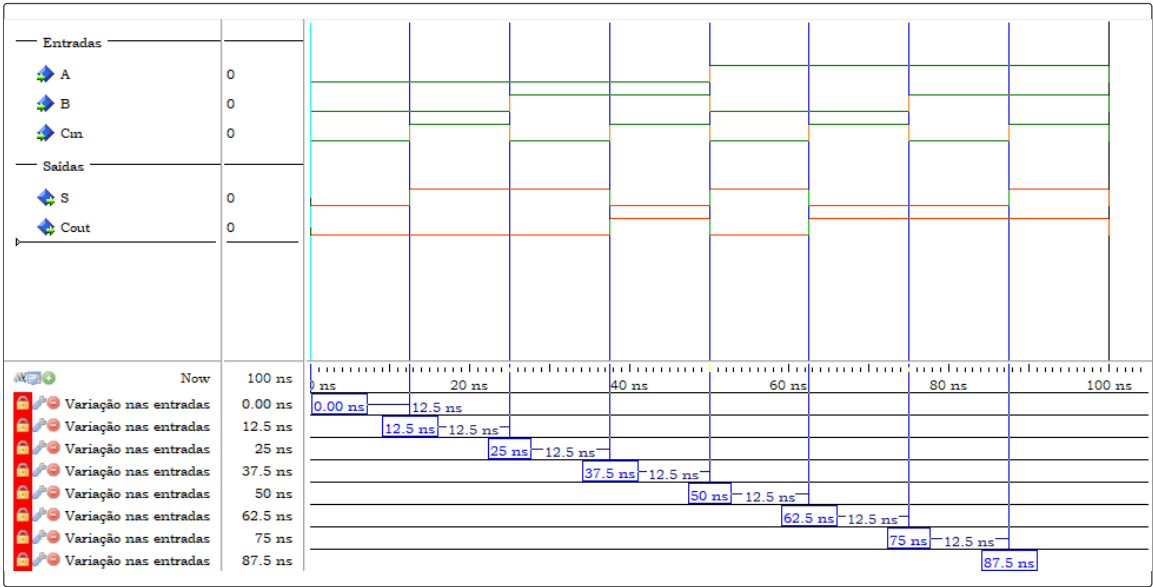


Figura 4: Simulação em forma de onda binária do somador completo

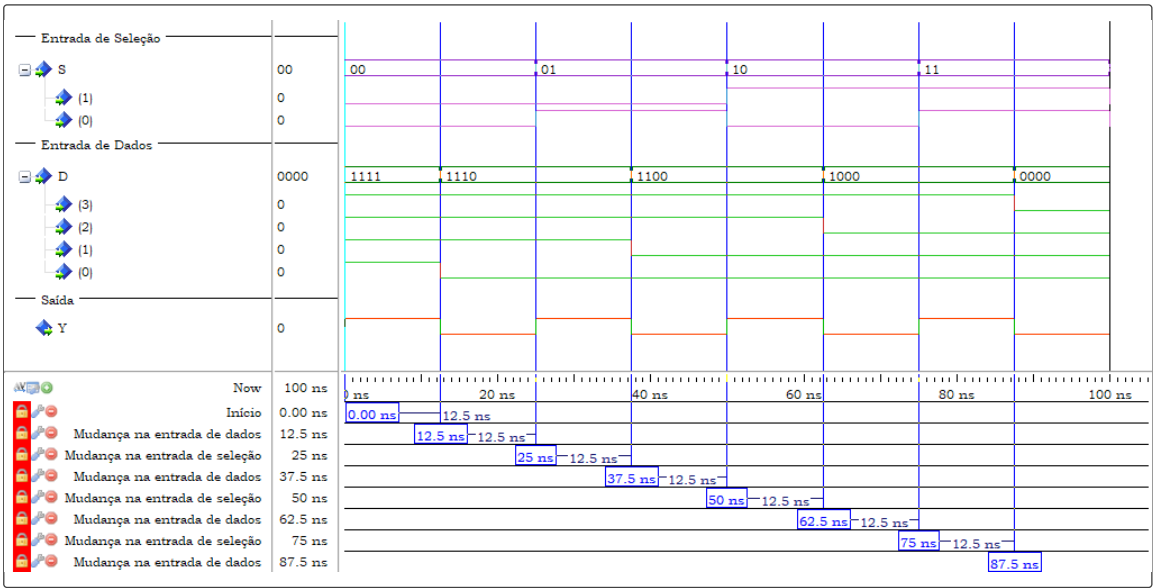


Figura 5: Simulação em forma de onda binária do multiplexador

5 Análise

Para analisar os resultados obtidos, basta olhar para os valores das ondas das entradas e saídas entre cada mudança dos sinais de entrada. Os cursores estão posicionados de forma que, ao selecionar cada um deles, vemos todas as possíveis combinações das entradas e saídas correspondentes na aba à esquerda da tela.

5.1 Análise do comportamento do Somador Completo

Abaixo, exibimos os *prints* que denotam o comportamento dos sinais de saída S e C_{out} para cada combinação dos sinais de entrada A , B e C_{in} .

<div> <div>Entradas</div> <div> <div>A</div> <div>B</div> <div>Cin</div> </div> <div>Saídas</div> <div> <div>S</div> <div>Cout</div> </div> </div>	<div>Msgs</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div>
<div> <div>Entradas</div> <div> <div>A</div> <div>B</div> <div>Cin</div> </div> <div>Saídas</div> <div> <div>S</div> <div>Cout</div> </div> </div>	<div>Msgs</div> <div>0</div> <div>0</div> <div>1</div> <div>1</div> <div>0</div>
<div> <div>Entradas</div> <div> <div>A</div> <div>B</div> <div>Cin</div> </div> <div>Saídas</div> <div> <div>S</div> <div>Cout</div> </div> </div>	<div>Msgs</div> <div>0</div> <div>1</div> <div>0</div> <div>1</div> <div>0</div>
<div> <div>Entradas</div> <div> <div>A</div> <div>B</div> <div>Cin</div> </div> <div>Saídas</div> <div> <div>S</div> <div>Cout</div> </div> </div>	<div>Msgs</div> <div>1</div> <div>0</div> <div>0</div> <div>1</div> <div>1</div>
<div> <div>Entradas</div> <div> <div>A</div> <div>B</div> <div>Cin</div> </div> <div>Saídas</div> <div> <div>S</div> <div>Cout</div> </div> </div>	<div>Msgs</div> <div>1</div> <div>0</div> <div>0</div> <div>1</div> <div>0</div>
<div> <div>Entradas</div> <div> <div>A</div> <div>B</div> <div>Cin</div> </div> <div>Saídas</div> <div> <div>S</div> <div>Cout</div> </div> </div>	<div>Msgs</div> <div>1</div> <div>1</div> <div>1</div> <div>1</div> <div>1</div>

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

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} .

5.2 Análise do comportamento do Multiplexador 4 para 1

Para analisar os resultados da simulação do Mux 4x1, observou-se a saída Y para cada combinação da entrada de seleção. Abaixo, estão exibidos os *prints* que mostram o comportamento do circuito simulado pelo ModelSim para cada uma dessas combinações. Perceba que cada linha representa uma combinação das entradas de seleção e, portanto,

uma entrada de dados selecionada, enquanto cada coluna representa um possível valor do sinal de entrada de dados selecionado.



























	Msgs		Msgs
<p>Entrada de Seleção</p> <p> S</p> <p>(1)</p> <p>(0)</p> <p>Entrada de Dados</p> <p> D</p> <p>(3)</p> <p>(2)</p> <p>(1)</p> <p>(0)</p> <p>Saída</p> <p> Y</p>	<p>00</p> <p>0</p> <p>0</p> <p>1111</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p>	<p>Entrada de Seleção</p> <p> S</p> <p>(1)</p> <p>(0)</p> <p>Entrada de Dados</p> <p> D</p> <p>(3)</p> <p>(2)</p> <p>(1)</p> <p>(0)</p> <p>Saída</p> <p> Y</p>	<p>00</p> <p>0</p> <p>0</p> <p>1110</p> <p>1</p> <p>1</p> <p>1</p> <p>0</p> <p>0</p>
<p>Entrada de Seleção</p> <p> S</p> <p>(1)</p> <p>(0)</p> <p>Entrada de Dados</p> <p> D</p> <p>(3)</p> <p>(2)</p> <p>(1)</p> <p>(0)</p> <p>Saída</p> <p> Y</p>	<p>10</p> <p>1</p> <p>0</p> <p>1100</p> <p>1</p> <p>1</p> <p>0</p> <p>0</p> <p>1</p>	<p>Entrada de Seleção</p> <p> S</p> <p>(1)</p> <p>(0)</p> <p>Entrada de Dados</p> <p> D</p> <p>(3)</p> <p>(2)</p> <p>(1)</p> <p>(0)</p> <p>Saída</p> <p> Y</p>	<p>10</p> <p>1</p> <p>0</p> <p>1000</p> <p>1</p> <p>0</p> <p>0</p> <p>0</p> <p>0</p>
<p>Entrada de Seleção</p> <p> S</p> <p>(1)</p> <p>(0)</p> <p>Entrada de Dados</p> <p> D</p> <p>(3)</p> <p>(2)</p> <p>(1)</p> <p>(0)</p> <p>Saída</p> <p> Y</p>	<p>01</p> <p>0</p> <p>1</p> <p>1110</p> <p>1</p> <p>1</p> <p>1</p> <p>0</p> <p>1</p>	<p>Entrada de Seleção</p> <p> S</p> <p>(1)</p> <p>(0)</p> <p>Entrada de Dados</p> <p> D</p> <p>(3)</p> <p>(2)</p> <p>(1)</p> <p>(0)</p> <p>Saída</p> <p> Y</p>	<p>01</p> <p>0</p> <p>1</p> <p>1100</p> <p>1</p> <p>1</p> <p>0</p> <p>0</p> <p>0</p>
<p>Entrada de Seleção</p> <p> S</p> <p>(1)</p> <p>(0)</p> <p>Entrada de Dados</p> <p> D</p> <p>(3)</p> <p>(2)</p> <p>(1)</p> <p>(0)</p> <p>Saída</p> <p> Y</p>	<p>11</p> <p>1</p> <p>1</p> <p>1000</p> <p>1</p> <p>0</p> <p>0</p> <p>0</p> <p>1</p>	<p>Entrada de Seleção</p> <p> S</p> <p>(1)</p> <p>(0)</p> <p>Entrada de Dados</p> <p> D</p> <p>(3)</p> <p>(2)</p> <p>(1)</p> <p>(0)</p> <p>Saída</p> <p> Y</p>	<p>11</p> <p>1</p> <p>1</p> <p>0000</p> <p>0</p> <p>0</p> <p>0</p> <p>0</p> <p>0</p>

Figura 7: Saídas para todas as combinações das entradas de seleção e de dados 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, ou seja, que a saída é igual ao sinal selecionado, como é, por definição, em um multiplexador.

6 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.