Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Relatório do Experimento 3

Autor: Henrique Morcelles Salum

Matrícula: 232003008

ENE0040 - Laboratório de Sistemas Digitais - Turma 07

Sumário

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 neste 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, consequentemente, $\log_2 8 = 3$ entradas de seleção, além de um sinal de saída.

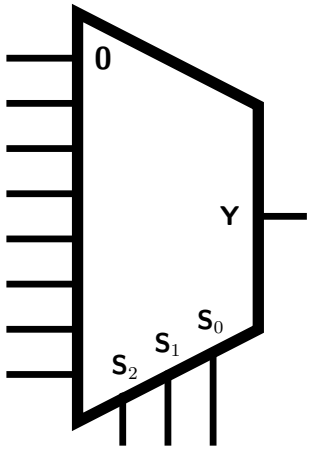


Figura 1: Representação externa do multiplexador 8 x 1

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.

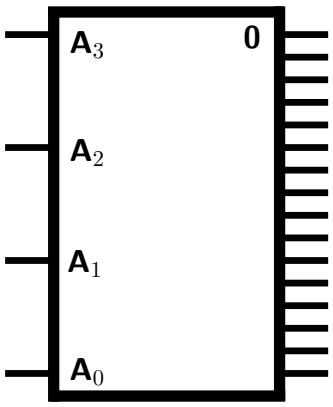


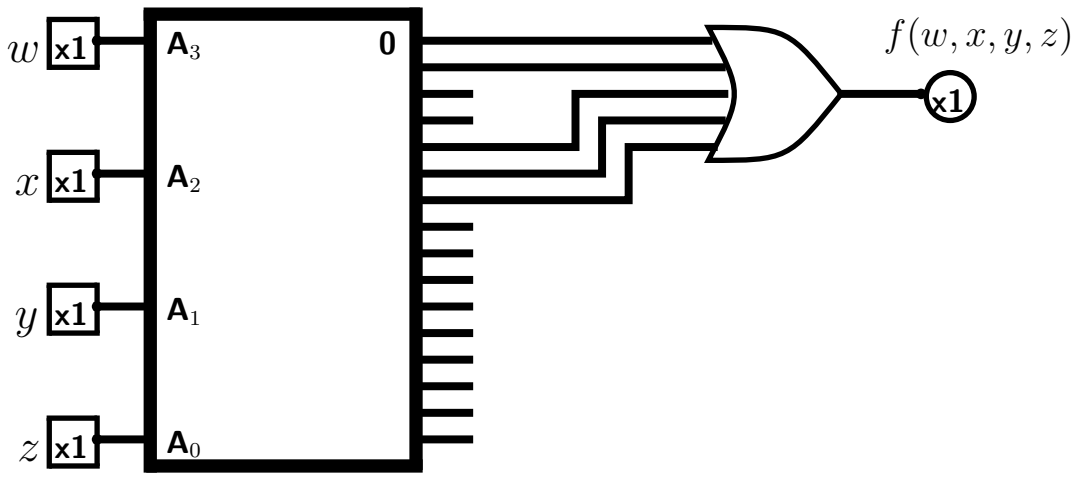
Figura 2: Representação externa do decodificador 4 x 16

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:

$$f(w,x,y,z) = \overline{w}\ \overline{x}\ \overline{y}\ \overline{z} + \overline{w}\ \overline{x}\ \overline{y}z + \overline{w}x\overline{y}\ \overline{z} + \overline{w}x\overline{y}z + \overline{w}xy\overline{z}$$

$$= \sum_{wxyz} m(0,1,4,5,6)$$

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 Descrição de Hardware

O código escrito para implementar o modelo de *hardware* do Mux 8x1, utilizando atribuições condicionais *when-else* está exibido abaixo.

```
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:

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

entity tb_Mux8x1 is
end entity tb_Mux8x1;

architecture main of tb_Mux8x1 is
    component Mux8x1 is
        port (
            D: in std_logic_vector(7 downto 0);
            S: in std_logic_vector(2 downto 0);
            Y: out std_logic
        );
    end component Mux8x1;
    signal D_tb: std_logic_vector(7 downto 0) := (others => '1');
    signal S_tb: std_logic_vector(2 downto 0) := (others => '0');
begin
    instancia_Mux8x1: component Mux8x1
        port map (
            D => D_tb,
            S => S_tb,
            Y => open
        );
    -- Processo para variar as entradas de seleção
    estimulos_seletoras: process
        variable s_unsigned: unsigned(2 downto 0);
    begin
        wait for 12.5 ns;
        s_unsigned := unsigned(S_tb);
        s_unsigned := s_unsigned + 1;
        S_tb <= std_logic_vector(s_unsigned);
    end process estimulos_seletoras;
    -- Processo para variar as entradas selecionadas
    variacao_dados: process
        variable i: integer;
    begin
        wait for 6.25 ns;
        i := to_integer(unsigned(S_tb));
        D_tb(i) <= not D_tb(i);
        wait on S_tb;
    end process variacao_dados;
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

2.2.1 Descrição de Hardware

O código, em VHDL, que descreve o circuito do decodificador 4 para 16, utilizando atribuição seletiva *with-select*, 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 decodificador 4x16

Nesse código, utilizamos o VHDL em alto nível, com a abstração *with-select*. Além disso, representamos os vetores de bits por hexadecimais, precedidos por 'X' no VHDL.

2.2.2 Testbench

Assim como com o multiplexador, precisamos fazer um *testbench* para o decodificador para testarmos o dispositivo implementado. O código desse *testbench* está exibido a seguir.

```
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
10     component Dec4x16 is
11         port (
12             A: in std_logic_vector(3 downto 0);
13             Y: out std_logic_vector(15 downto 0)
14         );
15     end component Dec4x16;
16
17     signal A_tb: std_logic_vector(3 downto 0) := (others => '0');
18 begin
19     instancia_Dec4x16: component Dec4x16
20         port map (
21             A => A_tb,
22             Y => open
23         );
24
25     estimulos: process
26         variable Uns_A: unsigned(3 downto 0);
27     begin
28         wait for 6.25 ns;
29         Uns_A := unsigned(A_tb);
30         A_tb <= std_logic_vector(Uns_A + 1);
31     end process estimulos;
32 end architecture testbench;
```

Código 4: Testbench para o decodificador 4x16

No código acima, diferente do Código 2, em que geramos apenas algumas combinações de interesse, geramos todas as combinações possíveis das entradas. Isso se deve à natureza do decodificador: diferente do multiplexador, todas as combinações possíveis das entradas são de interesse, pois todas interferem na saída a qualquer momento.

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.

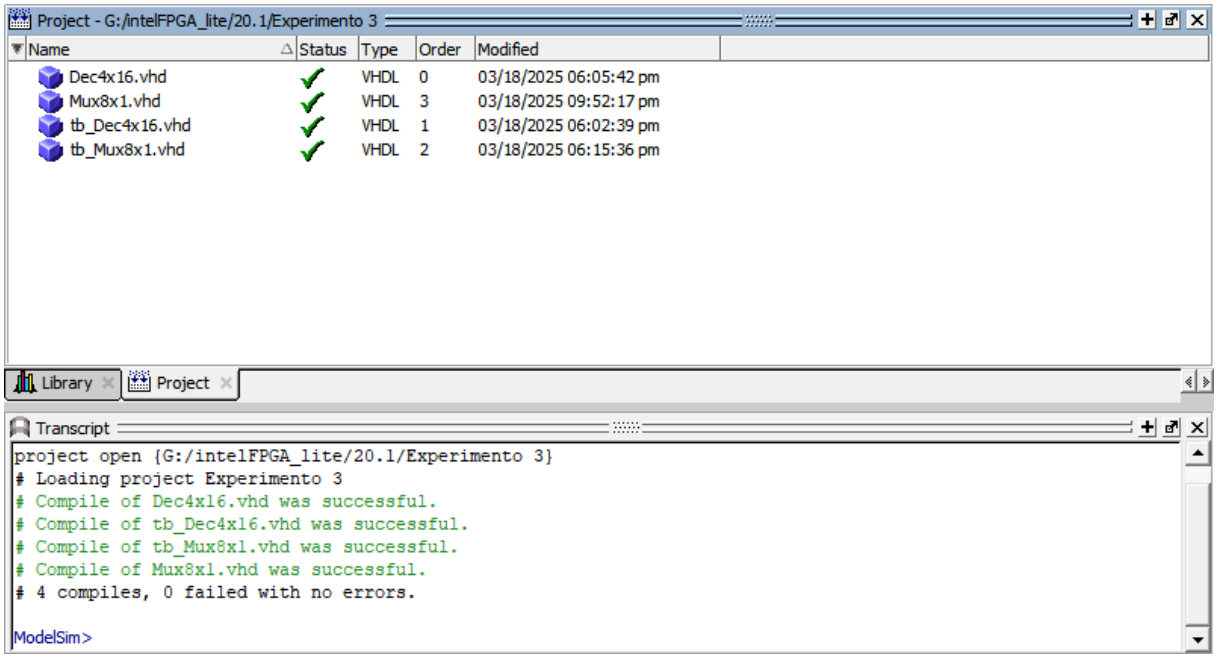


Figura 3: Compilação de todos os códigos apresentados

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 8x1 não representam todas as combinações das seis entradas. Seguem as imagens das simulações no ModelSim dos sistemas projetados.

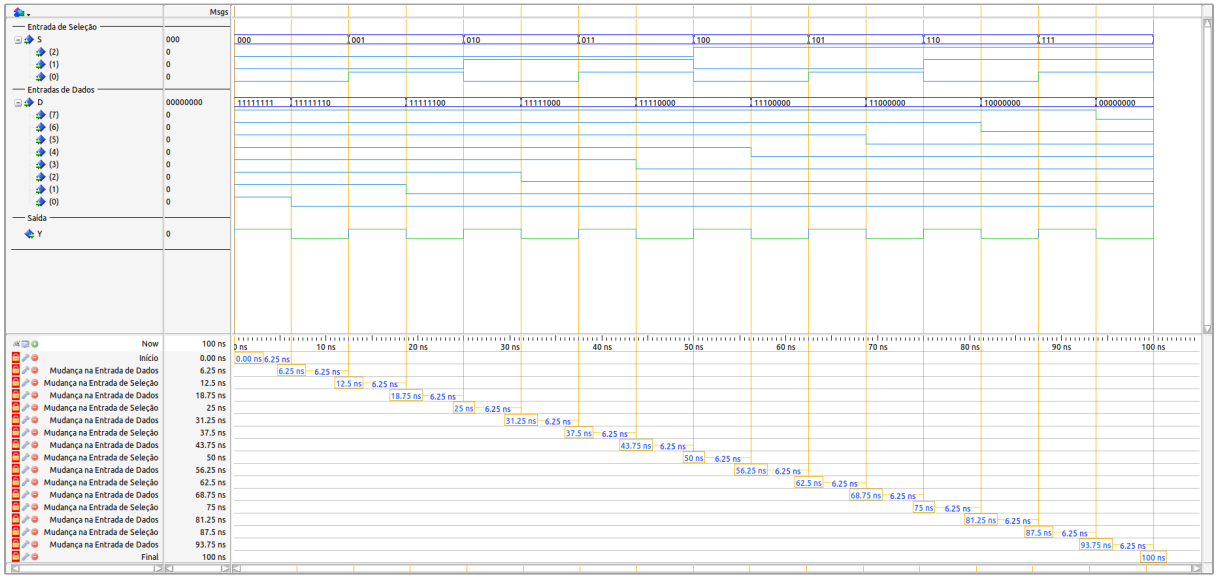


Figura 4: Simulação em forma de onda binária do multiplexador 8 para 1

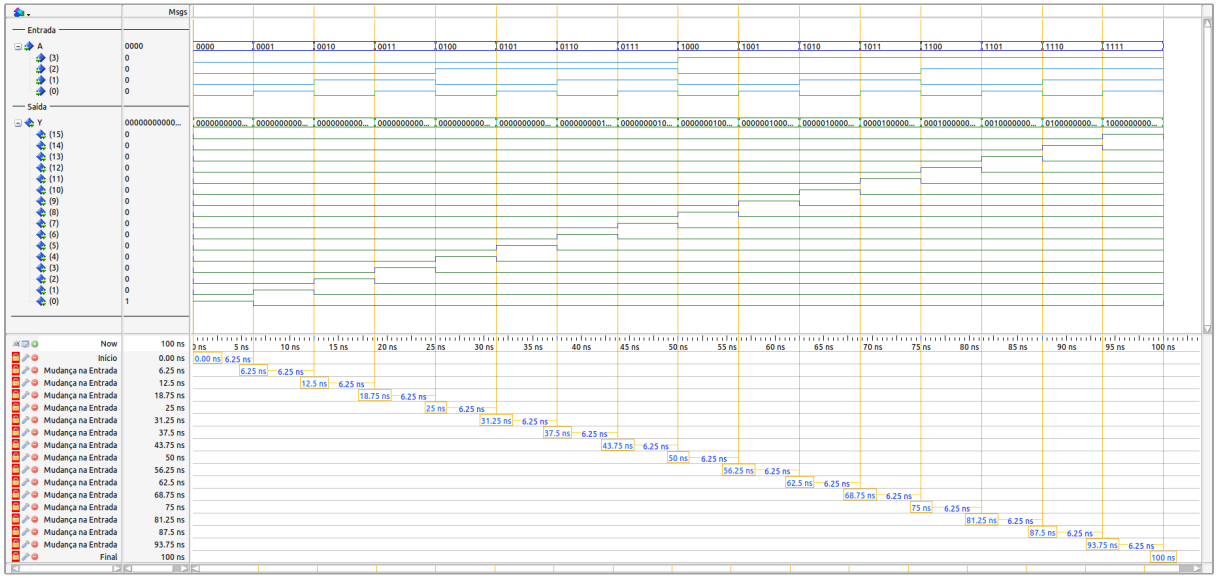


Figura 5: Simulação em forma de onda binária do decodificador 4 para 16

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 Multiplexador 8 para 1

Abaixo, exibimos os *prints* que denotam o comportamento do sinal de saída Y para cada combinação da entrada de seleção S e selecionada D_n.

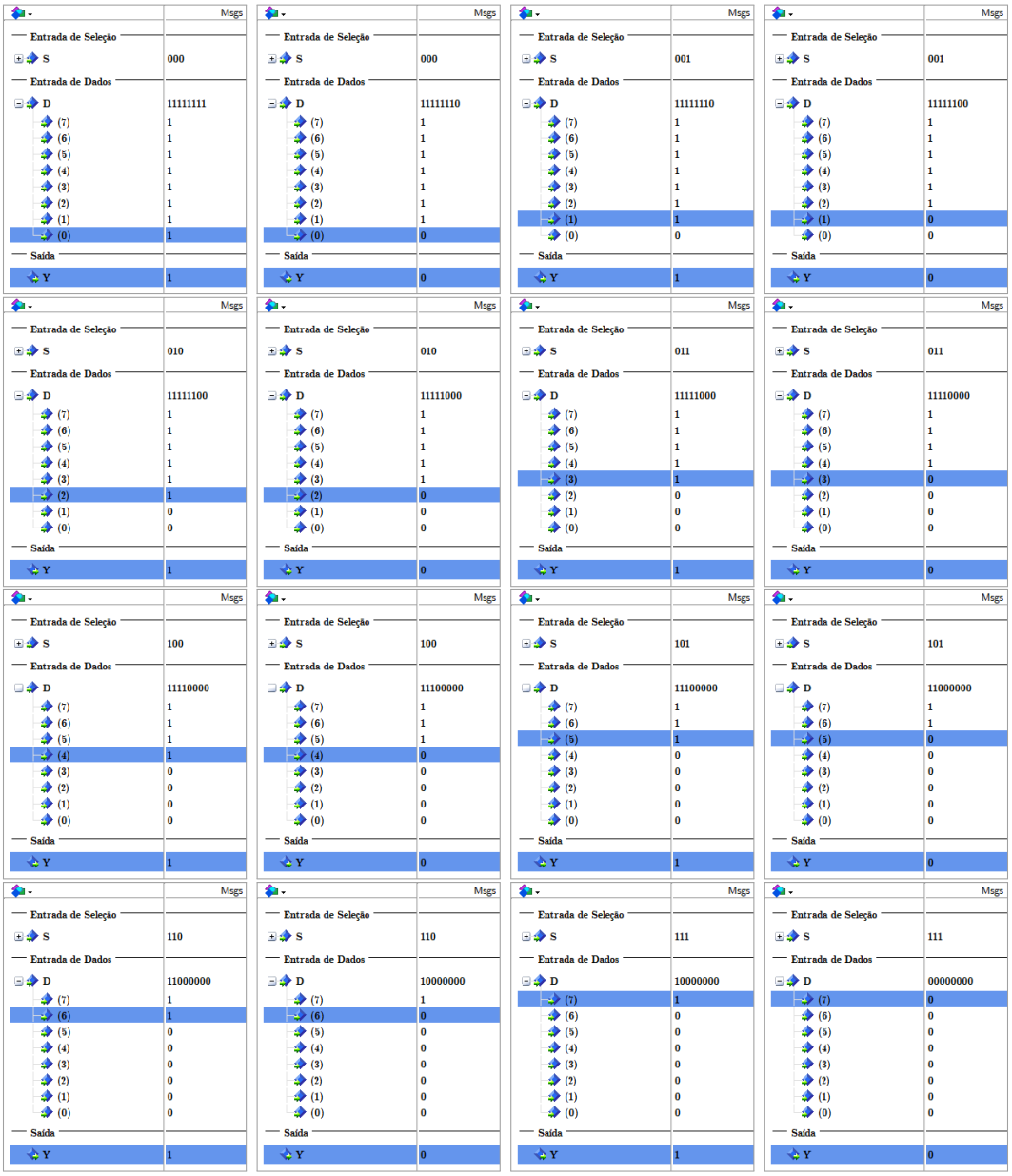


Figura 6: Combinações de entradas de seleção e saídas correspondentes do multiplexador 8 para 1

Percebe-se que o comportamento está de acordo com o esperado: quando a entrada de seleção seleciona a entrada de dados D_n, $Y = D_n$.

5.2 Análise do comportamento do Decodificador 4 para 16

Exibimos abaixo os *prints* da seção à esquerda da aba waves do ModelSim para cada combinação possível da entrada.

Figura 7: Todas as combinações de entradas e as saídas correspondentes do decodificador 4 para 16

Novamente, percebe-se que o comportamento está de acordo com o esperado. Cada um dos *prints* acima representa uma linha da tabela-verdade do decodificador 4 para 16 e, analisando-os, percebemos que a saída está de acordo com o exibido na Tabela 2.

6 Conclusão

Nesse experimento, simulou-se dois dispositivos digitais de suma importância para a eletrônica digital: o Multiplexador 8 para 1 e o decodificador 4 para 16.

Pelo relatório, concluímos o sucesso do experimento ao analisarmos devidamente os resultados obtidos pelas simulações: notamos que o comportamento dos sistemas explorados foi exatamente o esperado para todo possível estímulo, isto é, não foram observadas discrepâncias entre o comportamento esperado e o observado.