Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Relatório do Experimento 4

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 funções lógicas com VHDL e simulá-las por meio do software ModelSim, da Intel. É importante notar que utilizamos o *top module* nesta implementação, isto é, além dos circuitos referentes às questões e dos *testbenches*, criamos mais um arquivo em que instanciamos e conectamos os estímulos do *testbench* e o sistema da questão. As especificidades de cada tarefa serão exploradas a seguir.

1.1.1 Primeira Tarefa

Na primeira tarefa, devemos implementar um dispositivo com três bits de entrada, A, B e C e dois bits de saída X e Y. A lógica que relaciona as entradas e saídas desse dispositivo é explicitada pelas funções booleanas a seguir.

$$X = \overline{A}BC + A\overline{B}\,\overline{C} + AB$$
$$Y = \overline{A}\,\overline{B} + \overline{A}B\overline{C} + ABC$$

A tabela-verdade desse dispositivo está exibida a seguir.

Entradas			**Saídas**	
A	B	C	X	Y
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

Tabela 1: Tabela-verdade do dispositivo da questão 1

Para implementá-las, podemos utilizar apenas dois multiplexadores 4 para 1 (dispositivo desenvolvido no experimento 2) e uma porta inversora.

1.1.2 Segunda Tarefa

Na segunda tarefa, o dispositivo que devemos implementar em VHDL e simular no ModelSim tem sete bits de entrada, A, B, C, D, E, F e G, e um bit de saída S. A lógica entrada-saída desse dispositivo é estabelecida de acordo com a equação a seguir.

$$\begin{aligned} S =& FG + ABCD\overline{E}\,\overline{F}G + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E}\,\overline{F}G + A\overline{B}CDEF\overline{G} \\ &+ \overline{A}BCD\overline{E}F\overline{G} + ABCDE\overline{F}\,\overline{G} + A\overline{B}\,\overline{C}DE\overline{F}\,\overline{G} \end{aligned}$$

Para melhor apresentação desse documento, não exibiremos a tabela-verdade desse dispositivo, que teria 128 linhas. Ao invés disso, representaremos a função como o somatório dos mintermos que a formam.

$$\sum_{A,C,D,E,F,G} \begin{matrix} m(1,3,7,11,15,19,23,27,31,35,39,43,47,51,55,58,59,63,67,71, \\ 75,76,79,83,87,91,95,99,103,107,111,115,119,121,123,124,127) \end{matrix} \quad (1)$$

Para implementar esse dispositivo, nos é permitida a utilização de um decodificador 4 para 16, um multiplexador 8 para 1 (sistemas desenvolvidos no experimento 3) e três portas 'OU'.

1.2 Introdução Teórica

Ambos os dispositivos utilizados nesse experimento - multiplexador e decodificador - já foram extensivamente explicados nos relatórios anteriores. Aqui, portanto, a explicação será breve: só nos interessa lembrar que, no multiplexador, as entradas de dados podem servir para implementar funções das entradas de seleção e que cada saída do decodificador é um dos 2^n mintermos das suas n entradas. Como essas propriedades serão utilizadas na realização de cada tarefa aqui discutida, está explicado nas próximas seções.

1.2.1 Primeira Tarefa

Nessa tarefa, só nos é permitida a utilização de dois multiplexadores 4 para 1 - dotados de duas entradas de seleção cada - implementados em VHDL pelo código a seguir.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity Mux4x1 is
    port (
        D: in std_logic_vector(3 downto 0);
        S: in std_logic_vector(1 downto 0);
        Y: out std_logic -- Saída
    );
end entity Mux4x1;

architecture behavioral of Mux4x1 is
begin
    Y <= D(3) when (S = "11") else
         D(2) when (S = "10") else
         D(1) when (S = "01") else
         D(0) when (S = "00") else
         '-';
end architecture behavioral;
```

Código 1: Código para implementação do multiplexador 4 para 1

Note, porém, que as funções X e Y que precisamos implementar nesse dispositivo dependem de três variáveis. Nesse caso, podemos introduzir uma das entradas nas saídas, como mostra a tabela-verdade a seguir.

Entradas		**Saídas**	
A	B	$X(C)$	$Y(C)$
0	0	0	1
0	1	C	$\overline{C}$
1	0	$\overline{C}$	0
1	1	1	C

Tabela 2: Tabela-verdade do dispositivo da questão 1

No circuito, isso significa inserir a entrada C nas entradas de dados do multiplexador, como mostra a figura a seguir.

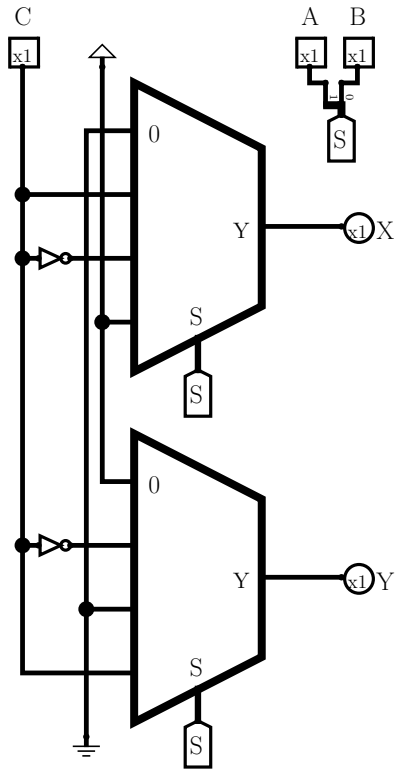

Figura 1: Implementação do dispositivo da questão 1 com a variável C inserida

Sem nenhuma razão além de facilidade de compreensão, escolhemos introduzir o C, mas poderíamos, sem nenhum prejuízo, introduzir o A ou o B. Nesses casos, seria conveniente reorientar a tabela-verdade original para facilitar a inserção da variável escolhida na saída, mas é claro que isso não geraria nenhum prejuízo ao sistema.

1.2.2 Segunda Tarefa

Nessa questão, somos restritos a utilizar um multiplexador 8 para 1 e um decodificador 4 para 16, implementados como segue.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity Mux8x1 is
    port (
        D: in std_logic_vector(7 downto
        ↪ 0);
        S: in std_logic_vector(2 downto
        ↪ 0);
        Y: out std_logic
    );
end entity Mux8x1;

architecture behavioral of Mux8x1 is
begin
    Y <= D(7) when (S = "111") else
         D(6) when (S = "110") else
         D(5) when (S = "101") else
         D(4) when (S = "100") else
         D(3) when (S = "011") else
         D(2) when (S = "010") else
         D(1) when (S = "001") else
         D(0) when (S = "000") else
         '-';
end architecture behavioral;
```

Código 2: Mux 8x1

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity Dec4x16 is
    port (
        A: in std_logic_vector(3 downto 0);
        Y: out std_logic_vector(15 downto 0)
    );
end entity Dec4x16;

architecture behavioral of Dec4x16 is
begin
    with A select Y <=
        X"0001" when X"0",
        X"0002" when X"1",
        X"0004" when X"2",
        X"0008" when X"3",
        X"0010" when X"4",
        X"0020" when X"5",
        X"0040" when X"6",
        X"0080" when X"7",
        X"0100" when X"8",
        X"0200" when X"9",
        X"0400" when X"A",
        X"0800" when X"B",
        X"1000" when X"C",
        X"2000" when X"D",
        X"4000" when X"E",
        X"8000" when X"F",
        "----------------" when others;
end architecture behavioral;
```

Código 3: Dec 4x16

O problema é o parecido com o da questão anterior: um decodificador 4 para 16 gera todos os mintermos de **quatro** variáveis, mas temos sete! Aqui podemos utilizar o multiplexador para resolver esse problema: fazemos das outras três variáveis as entradas de seleção do multiplexador 8 para 1 e conseguimos implementar a função.

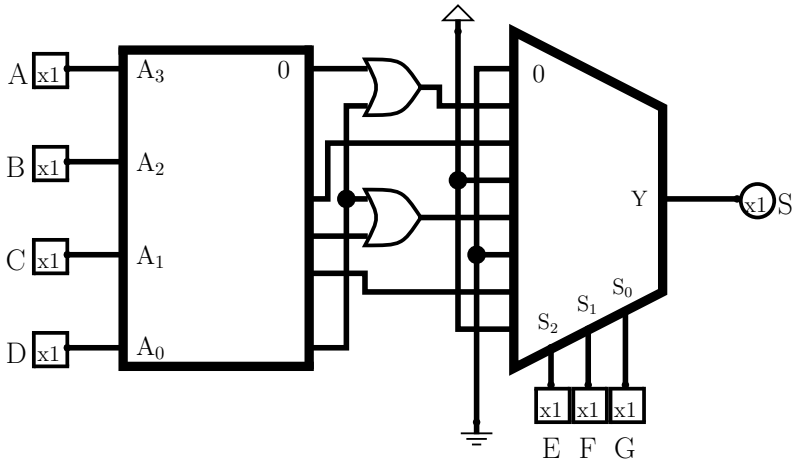

Figura 2: Implementação do dispositivo da questão 1 com a variável C inserida

Note que o "truque" com o Mux é o mesmo de antes, mas, ao invés de inserir apenas uma variável nas entradas de dados, inserimos mintermos. Além disso, novamente a escolha de A, B, C e D como entradas do decodificador e de E, F e G como entradas do multiplexador é apenas por conveniência; poderíamos fazer de outra forma e, mesmo que de maneira mais confusa, teríamos o exato mesmo resultado.

2 Códigos

Nessa seção, apresentamos os códigos utilizados para implementar os dispositivos apresentados na seção anterior. Os códigos que implementam os multiplexadores e o decodificador já foram apresentados na introdução, pois não são objetos deste experimento.

2.1 Questão 1

A seguir estão exibidos os códigos para a implementação e teste do dispositivo da questão 1.

2.1.1 Descrição de Hardware

Nessa implementação, foi necessário utilizar uma arquitetura estrutural (*structural*), isto é, não descrevemos apenas o comportamento abstrato do dispositivo; precisamos instanciar outros sistemas e conectá-los por meio de sinais internos.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity questao1 is
5      port (
6          A, B, C: in std_logic;
7          X, Y: out std_logic
8      );
9  end entity questao1;
10
11 architecture structural of questao1 is
12     component Mux4x1 is
13         port (
14             D: in std_logic_vector(3 downto 0);
15             S: in std_logic_vector(1 downto 0);
16             Y: out std_logic
17         );
18     end component Mux4x1;
19     signal D_tb: std_logic_vector(3 downto 0);
20     signal S_tb: std_logic_vector(1 downto 0);
21     signal D2_tb: std_logic_vector(3 downto 0);
22     signal S2_tb: std_logic_vector(1 downto 0);
```

```
23 begin
24     Mux_1 : component Mux4x1
25         port map (
26             D => D_tb,
27             S => S_tb,
28             Y => X
29         );
30 
31     Mux_2 : component Mux4x1
32         port map (
33             D => D2_tb,
34             S => S2_tb,
35             Y => Y
36         );
37 
38     S_tb <= A & B;
39     D_tb <= '0' & C & (not C) & '1';
40 
41     S2_tb <= A & B;
42     D2_tb <= '1' & (not C) & '0' & C;
43 end architecture structural;
```

Código 4: Sistema da questão 1

2.1.2 Testbench

Perceba que o *testbench* aqui é diferente dos desenvolvidos nos experimentos anteriores: a *entity* não está vazia, há sinais de saída e apenas geramos estímulos nesses sinais, não conectamos os "cabos" (sinais internos) às entradas do sistema que testamos. Isso ocorre porque instanciaremos esse *textbench* no *top module* e só lá conectaremos os "cabos" dele com os sinais de entrada do sistema.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4 
5  entity tb_questao1 is
6      port (
7          A, B, C: out std_logic
8      );
9  end tb_questao1;
10 
11 architecture testbench of tb_questao1 is
12     signal X: std_logic_vector(2 downto 0) := (others => '0');
13 begin
14     estimulos: process
```

```
    begin
        wait for 7 ns;
        X <= std_logic_vector(unsigned(X) + 1);
    end process estimulos;
    A <= X(2);
    B <= X(1);
    C <= X(0);
end architecture testbench;
```

Código 5: Testbench do sistema da questão 1

2.1.3 Top Module

Aqui, no *top module*, instanciamos e conectamos tudo. É como se fosse onde ligamos o sistema ao resto do circuito.

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

entity topModule is
end entity topModule;

architecture structural of topModule is
    component questao1 is
        port (
            A, B, C: in std_logic;
            X, Y: out std_logic
        );
    end component questao1;

    component tb_questao1 is
        port (
            A, B, C: out std_logic
        );
    end component tb_questao1;

    signal A_tb: std_logic;
    signal B_tb: std_logic;
    signal C_tb: std_logic;
begin
    instancia_testbench: component tb_questao1
        port map (
            A => A_tb,
            B => B_tb,
            C => C_tb
        );

```

```
    instancia_questao1: component questao1
        port map (
            A => A_Tb,
            B => B_tb,
            C => C_tb,
            X => open,
            Y => open
        );
end architecture structural;
```

Código 6: Top module da questão 1

2.2 Questão 2

Os códigos referentes à implementação do sistema da questão 2 estão apresentados abaixo. Neles, segue-se o padrão exposto na Subseção 2.1: a arquitetura do sistema é estrutural, não comportamental e o *testbench* não faz as conexões com o dispositivo; utilizamos o *top module* para ligar os estímulos ao circuito.

2.2.1 Descrição de Hardware

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

entity questao2 is
    port (
        A, B, C, D, E, F, G: in std_logic;
        S: out std_logic
    );
end entity questao2;

architecture structural of questao2 is
    component Dec4x16 is
        port (
            A: in std_logic_vector(3 downto 0);
            Y: out std_logic_vector(15 downto 0)
        );
    end component Dec4x16;

    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;
```

```
26
27      signal A_tb: std_logic_vector(3 downto 0);
28      signal Y_tb: std_logic_vector(15 downto 0);
29      signal D_tb: std_logic_vector(7 downto 0);
30      signal S_tb: std_logic_vector(2 downto 0);
31      signal Y2_tb: std_logic;
32  begin
33      instancia_Dec4x16: component Dec4x16
34          port map (
35              A => A_tb,
36              Y => Y_tb
37          );
38      instancia_Mux8x1: component Mux8x1
39          port map (
40              D => D_tb,
41              S => S_tb,
42              Y => Y2_tb
43          );
44      A_Tb <= A & B & C & D;
45      D_tb <= '1' & (Y_tb(10) or Y_tb(11)) & '0' & (Y_tb(9) or Y_tb(15)) & '1' &
        ↪  Y_tb(7) & (Y_tb(0) or Y_tb(15)) & '0';
46      S_tb <= E & F & G;
47      S <= Y2_tb;
48  end architecture structural;
```

Código 7: Sistema da questão 2

2.2.2 Testbench

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.numeric_std.all;
4
5   entity tb_questao2 is
6       port (
7           A, B, C, D, E, F, G: out std_logic
8       );
9   end tb_questao2;
10
11  architecture testbench of tb_questao2 is
12      signal X: std_logic_vector(6 downto 0) := (others => '0');
13  begin
14      estimulos: process
15      begin
16          wait for 1 ns;
17          X <= std_logic_vector(unsigned(X) + 1);
```

```
    end process estimulos;
    A <= X(6);
    B <= X(5);
    C <= X(4);
    D <= X(3);
    E <= X(2);
    F <= X(1);
    G <= X(0);
end architecture testbench;
```

Código 8: Testbench da questão 2

2.2.3 Top Module

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

entity q2_topModule is
end entity q2_topModule;

architecture structural of q2_topModule is
    component questao2 is
        port (
            A, B, C, D, E, F, G: in std_logic;
            S: out std_logic
        );
    end component questao2;

    component GoldenModel is
        port (
            A, B, C, D, E, F, G: in std_logic;
            S: out std_logic
        );
    end component GoldenModel;

    component tb_questao2 is
        port (
            A, B, C, D, E, F, G: out std_logic
        );
    end component tb_questao2;

    signal X: std_logic_vector(6 downto 0);
    signal q2_output: std_logic;
    signal goldenModel_output: std_logic;
begin
    instancia_tb_questao2 : component tb_questao2
```

```
33          port map (
34              A => X(6),
35              B => X(5),
36              C => X(4),
37              D => X(3),
38              E => X(2),
39              F => X(1),
40              G => X(0)
41          );
42
43      instancia_questao2 : component questao2
44          port map (
45              A => X(6),
46              B => X(5),
47              C => X(4),
48              D => X(3),
49              E => X(2),
50              F => X(1),
51              G => X(0),
52              S => q2_output
53          );
54
55      instancia_goldenModel : component questao2
56          port map (
57              A => X(6),
58              B => X(5),
59              C => X(4),
60              D => X(3),
61              E => X(2),
62              F => X(1),
63              G => X(0),
64              S => goldenModel_output
65          );
66
67      comparacao: process(q2_output, goldenModel_output)
68      begin
69          if q2_output /= goldenModel_output then
70              report "Erro!!";
71          end if;
72      end process comparacao;
73
74
75  end architecture structural;
```

Código 9: Top module da questão 2

2.2.4 Golden Model

Perceba que, no código acima, outro componente, além do dispositivo da questão 2, é instanciado: o *Golden Model*. Como existem $2^7 = 128$ possíveis combinações de entradas para o dispositivo dessa questão, foi necessário criar um modelo ideal de como o dispositivo deve funcionar para que pudéssemos testá-lo apropriadamente.

Nesse modelo, como pode ser visto no código a seguir, apenas definimos se a saída é '1' ou '0' em cada caso, de acordo com o exposto em (1), não é necessária nenhuma lógica ou conhecimento sobre circuitos lógicos.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity GoldenModel is
5      port (
6          A, B, C, D, E, F, G: in std_logic;
7          S: out std_logic
8      );
9  end entity GoldenModel;
10
11 architecture behavioral of GoldenModel is
12 begin
13     process(A, B, C, D, E, F, G)
14         variable I: std_logic_vector(6 downto 0);
15     begin
16         I := std_logic_vector'(A & B & C & D & E & F & G);
17         case I is
18             when "0000001" | "0000011" | "0000111" | "0001011" |
19             "0001111" | "0010011" | "0010111" | "0011011" |
20             "0011111" | "0100011" | "0100111" | "0101011" |
21             "0101111" | "0110011" | "0110111" | "0111010" |
22             "0111011" | "0111111" | "1000011" | "1000111" |
23             "1001011" | "1001100" | "1001111" | "1010011" |
24             "1010111" | "1011011" | "1011111" | "1100011" |
25             "1100111" | "1101011" | "1101111" | "1110011" |
26             "1110111" | "1111001" | "1111011" | "1111100" |
27             "1111111" =>
28                 S <= '1';
29             when others =>
30                 S <= '0';
31         end case;
32     end process;
33 end architecture behavioral;
```

Código 10: *Golden Model* da questão 2

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.

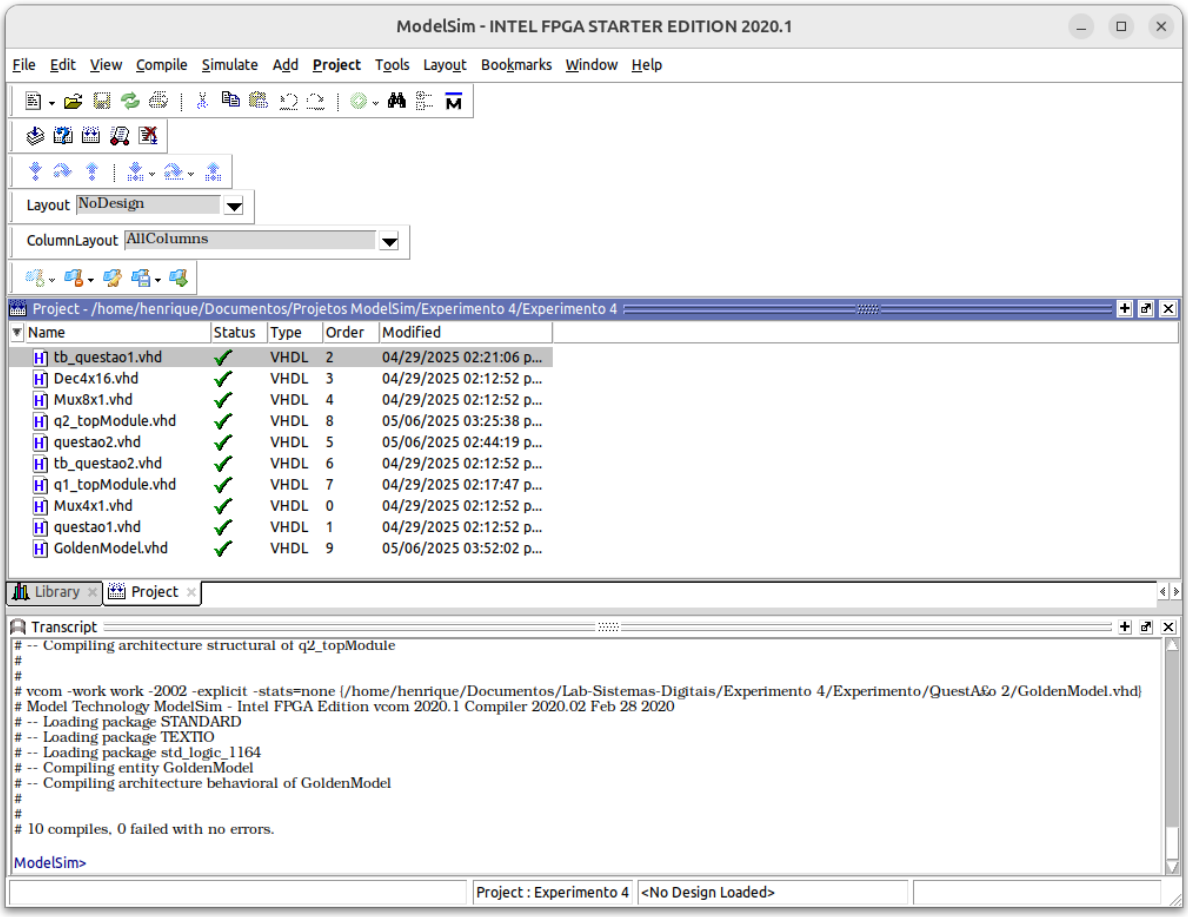

Figura 3: Compilação de todos os códigos apresentados

4 Simulação

Os gráficos de onda das entradas e saídas dos dispositivos das questões 1 e 2 estão exibidos a seguir. Note que no primeiro, como definido no *testbench*, os sinais de entrada variam a cada 12,5 nanosegundos (ns) no decorrer de 100 ns de simulação, resultando em 8 variações - todas as combinações possíveis dos sinais de entrada. No segundo, as variações ocorrem a cada 1 ns e a simulação dura 128 ns.

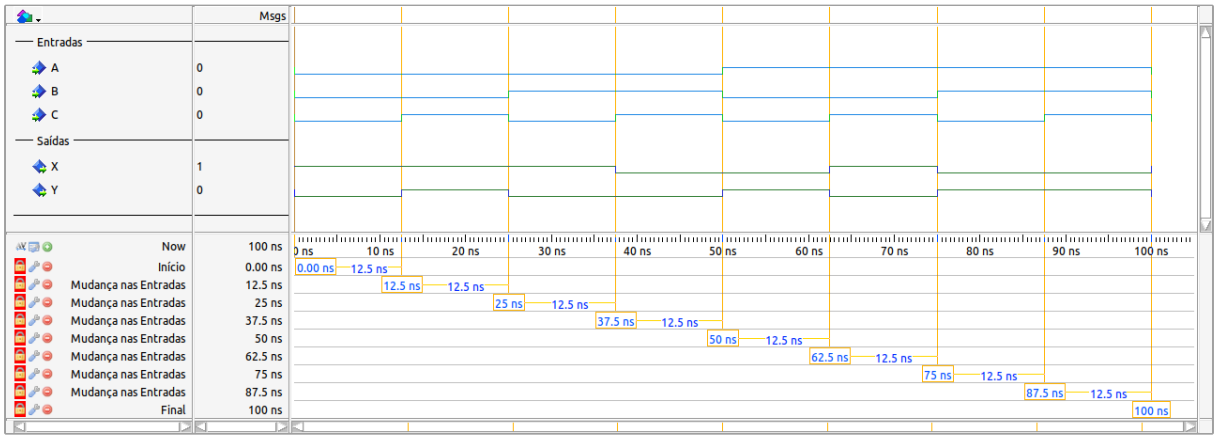

Figura 4: Simulação em forma de onda binária do dispositivo da questão 1

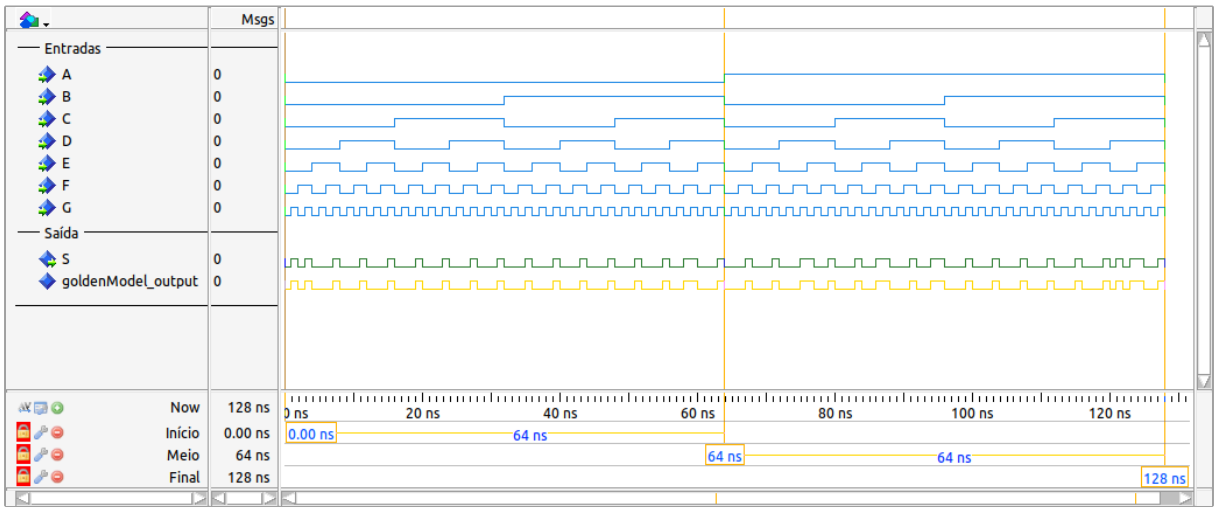

Figura 5: Simulação em forma de onda binária do dispositivo da questão 2

Perceba que, na segunda imagem, há duas ondas de saída. A segunda, em amarelo, representa o *Golden Model*. Note também que, na segunda simulação, não foram colocados cursores a cada variação das entradas. Isso deixaria a imagem muito poluída, então optamos por posicionar cursores apenas no início, meio e fim da simulação.

5 Análise

5.1 Questão 1

O dispositivo da questão 1 tem três entradas e, portanto, oito saídas. Nesse caso, seguiremos o procedimento padrão: exibiremos *prints* das entradas e saídas para cada possível combinação das entradas.

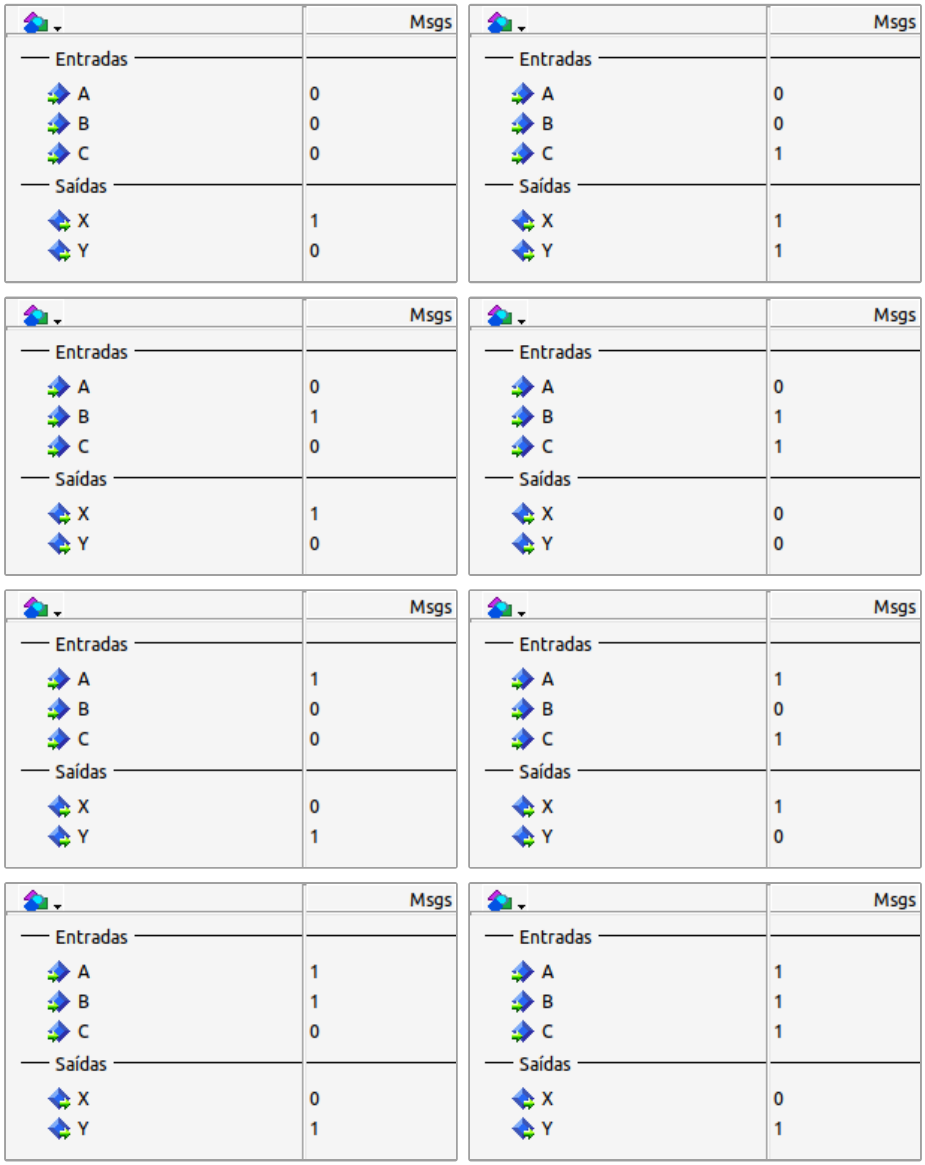

Figura 6: Todas as combinações de entradas do dispositivo da questão 1

Cada *print* na figura acima representa uma linha da tabela-verdade. Comparando o resultado obtido com a Tabela 1, é evidente que a implementação da questão 1 apresentada está correta. Claro que só consideramos os casos em que as entradas variam entre '0' e '1', os outros valores fornecidos pelo tipo *std_logic* são ignorados. Pela implementação do multiplexador, porém, sabemos que a saída seria *don't care* (−) em qualquer outro caso.

5.2 Questão 2

Na análise da segunda questão, não é necessário (e seria impraticável) exibir os *prints* de todas as possíveis combinações das entradas. Ao invés disso, utilizaremos o *Golden Model*. Imediatamente, é fácil perceber que os sinais de saída S e $goldenModel\_output$ são idênticos ao olhar Figura 5, mas, além disso, podemos notar que a mensagem "Erro!!" não foi exibida no *Transcript* em nenhum momento da execução, como mostrado no canto inferior da figura a seguir. Dessa forma, fica claro que o dispositivo da questão 2 foi implementado corretamente.

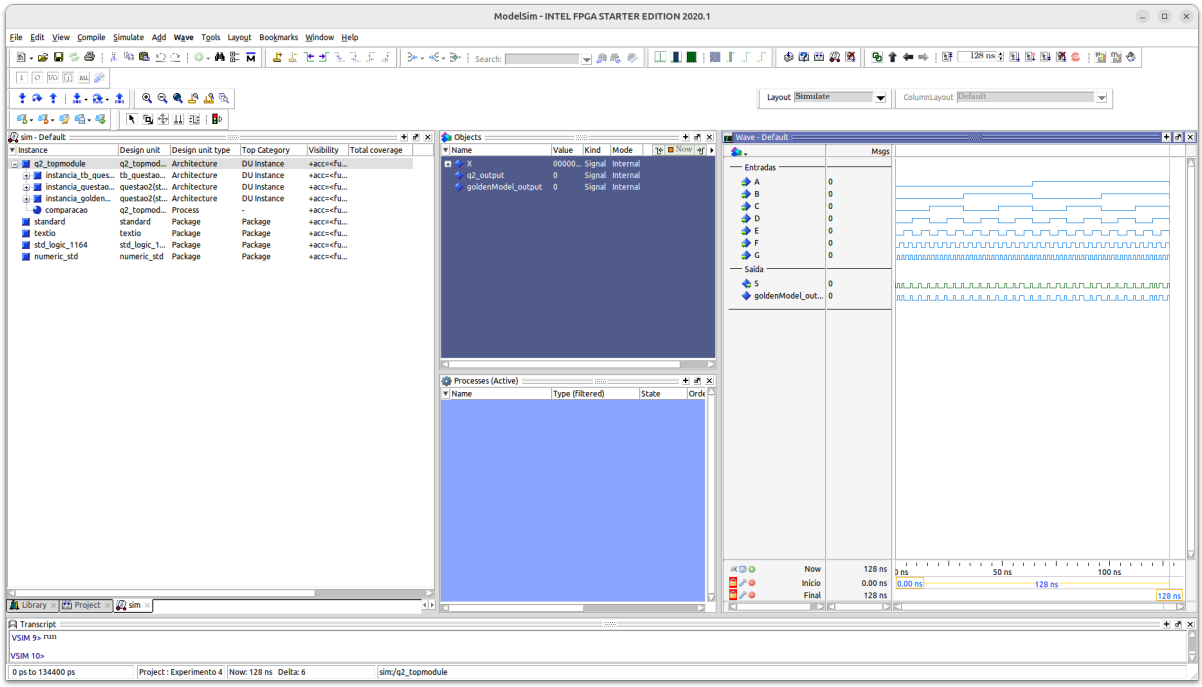

Figura 7: Tela do ModelSim após a simulação

6 Conclusão

Nesse experimento, colocamos em prática os conhecimentos sobre multiplexadores e decodificadores explorados nos relatórios anteriores, implementando funções booleanas para descrever a lógica entrada-saída de dois sistemas digitais. Além disso, introduzimos o conceito de *Golden Model* e utilizamos isso para verificar a funcionalidade do sistema da questão 2.

Finalmente, com as práticas introduzidas nesse relatório, conseguimos verificar a correção dos códigos da questão 1 e 2, isto é, verificamos que os sistemas foram devidamente implementados pelos arquivos de descrição de *hardware* apresentados na Seção 2.