

Relatório 2 VHDL - Turma 05
Yan Tavares de Oliveira
202014323



Introdução

Este experimento consiste em duas etapas. Na primeira (questão 1), iremos descrever em VHDL e simular no software ModelSim uma entidade que descreva o comportamento de um multiplexador 8 para 1.

Na segunda etapa (questão 2), iremos descrever em VHDL e simular no software ModelSim uma entidade que descreva o comportamento de um decodificador 4 para 16 bits.

Teoria

Na questão 1, devemos implementar um multiplexador 8 x 1, descrito pela imagem 1. O multiplexador em questão permite selecionar um bit específico de um vetor lógico D (de 8 bits) utilizando seletores de 3 bits.

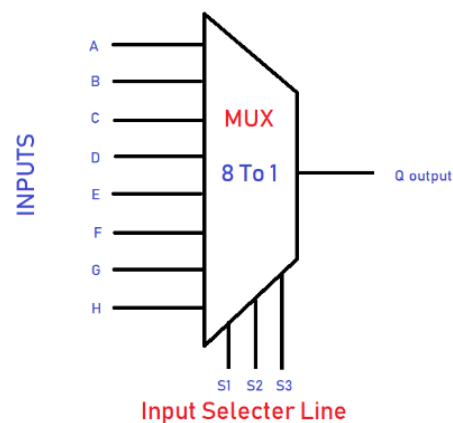


Imagem 1. Representação visual de um multiplexador 8 x 1.

Entradas (S)	Saída (Y)
000	D0
001	D1
010	D2
011	D3
100	D4
101	D5
110	D6
111	D7

Imagem 2. Tabela verdade do multiplexador 8 x 1.

Na questão 2, devemos implementar um decodificador 4 para 16, descrito pela imagem 2. O decodificador em questão produz saídas de 16 bits para as combinações das entradas de 4 bits.

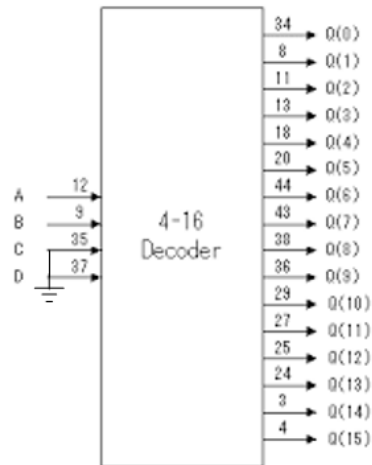


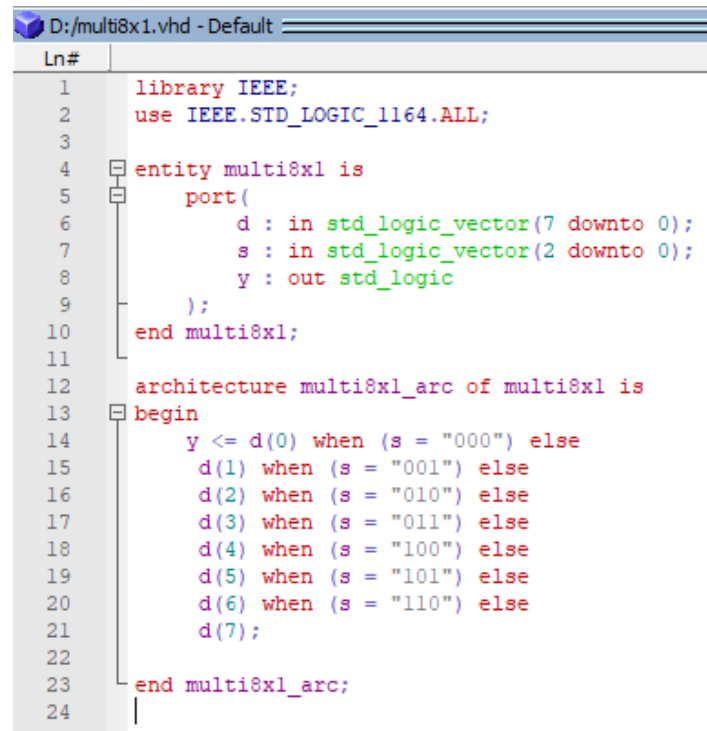
Imagem 3. Representação visual de um decodificador 4 x 16.

Entradas (A)	Saída (Y)
0000	0000 0000 0000 0001
0001	0000 0000 0000 0010
0010	0000 0000 0000 0100
0011	0000 0000 0000 1000
0100	0000 0000 0001 0000
0101	0000 0000 0010 0000
0110	0000 0000 0100 0000
0111	0000 0000 1000 0000
1000	0000 0001 0000 0000
1001	0000 0010 0000 0000
1010	0000 0100 0000 0000
1011	0000 1000 0000 0000
1100	0001 0000 0000 0000
1101	0010 0000 0000 0000
1110	0100 0000 0000 0000
1111	1000 0000 0000 0000

Imagem 4. Tabela verdade do decodificador 4 x 16

Códigos

Na **questão 1**, para implementar o multiplexador descrito na introdução, foi criada uma simulação no software ModelSim com um seletor de 3 bits (S) em forma de vetor lógico, que define qual dos bits presentes no vetor lógico D (8 bits) será apresentado na saída. Para determinar qual bit de D seria representado, foram utilizadas atribuições condicionais when-else.



```
D:/multi8x1.vhd - Default
Ln#
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity multi8x1 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 multi8x1;
11
12 architecture multi8x1_arc of multi8x1 is
13 begin
14     y <= d(0) when (s = "000") else
15         d(1) when (s = "001") else
16         d(2) when (s = "010") else
17         d(3) when (s = "011") else
18         d(4) when (s = "100") else
19         d(5) when (s = "101") else
20         d(6) when (s = "110") else
21         d(7);
22
23 end multi8x1_arc;
24
```

Imagem 5. 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, utilizando clocks para esse fim.

```
D:/multi8x1_tb.vhd - Default *
Ln#
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity multi8x1_tb is end;
5
6  architecture multi8x1_arc of multi8x1_tb is
7
8      component multi8x1
9      port(
10         d : in std_logic_vector(7 downto 0);
11         s : in std_logic_vector(2 downto 0);
12         y : out std_logic
13     );
14     end component;
15
16     signal clkd : std_logic_vector(7 downto 0) := "00000000";
17     signal clks : std_logic_vector(2 downto 0) := "000";
18
19     begin
20         multi8x1_tb : multi8x1 port map (d => clkd, s => clks, y => open);
21         clkd(0) <= not clkd(0) after 1 ns;
22         clkd(1) <= not clkd(1) after 2 ns;
23         clkd(2) <= not clkd(2) after 4 ns;
24         clkd(3) <= not clkd(3) after 8 ns;
25         clkd(4) <= not clkd(4) after 16 ns;
26         clkd(5) <= not clkd(5) after 32 ns;
27         clkd(6) <= not clkd(6) after 64 ns;
28         clkd(7) <= not clkd(7) after 128 ns;
29
30         clks(0) <= not clks(0) after 256 ns;
31         clks(1) <= not clks(1) after 512 ns;
32         clks(2) <= not clks(2) after 1024 ns;
33
34     end multi8x1_arc;
35
36
```

Imagem 6. Código auxiliar da questão 1

Na **questão 2**, para implementar o decodificador descrito na introdução, foi criada uma simulação no software ModelSim com 1 vetor lógico de entrada (A com 4 bits) e um vetor lógico de saída (Y com 16 bits). Para determinar a saída gerada por A,, foram utilizadas atribuições seletivas with-select.

```

D:/decodificador_4X16.vhd - Default *
Ln#
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity decodificador_4X16 is
5
6  Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
7
8        y : out  STD_LOGIC_VECTOR (15 downto 0));
9
10 end decodificador_4X16;
11
12 architecture decodificador_4X16_arc of decodificador_4X16 is begin
13
14     with a select
15
16     y <=  "0000000000000001" when "0000",
17
18           "0000000000000010" when "0001",
19
20           "0000000000000100" when "0010",
21
22           "0000000000001000" when "0011",
23
24           "0000000000010000" when "0100",
25
26           "0000000000100000" when "0101",
27
28           "0000000001000000" when "0110",
29
30           "0000000010000000" when "0111",
31
32           "0000000100000000" when "1000",
33
34           "0000001000000000" when "1001",
35
36           "0000010000000000" when "1010",
37
38           "0000100000000000" when "1011",
39
40           "0001000000000000" when "1100",
41
42           "0010000000000000" when "1101",
43
44           "0100000000000000" when "1110",
45
46           "1000000000000000" when "1111",
47
48           "0000000000000000" when others;
49 end decodificador_4X16_arc;
50

```

Imagem 7. 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:/decodificador_4X16_tb.vhd - Default
Ln#
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity decodificador_4X16_tb is end;
6
7  architecture decodificador_4X16_arc of decodificador_4X16_tb is
8
9      component decodificador_4X16
10     port (
11         a : in std_logic_vector(3 downto 0);
12         y : out std_logic_vector(15 downto 0)
13     );
14 end component;
15
16
17     signal clka : std_logic_vector(3 downto 0) := "0000";
18
19 begin
20     decodificador_4X16_tb : decodificador_4X16 port map (a => clka, y => open);
21     clka(0) <= not clka(0) after 10 ns;
22     clka(1) <= not clka(1) after 20 ns;
23     clka(2) <= not clka(2) after 40 ns;
24     clka(3) <= not clka(3) after 80 ns;
25
26 end decodificador_4X16_arc;
27

```

Imagem 8. Código auxiliar da questão 2

Compilação

Abaixo estão as mensagens de compilação de ambos os projetos, com nenhum erro sendo apresentado em nenhum dos casos

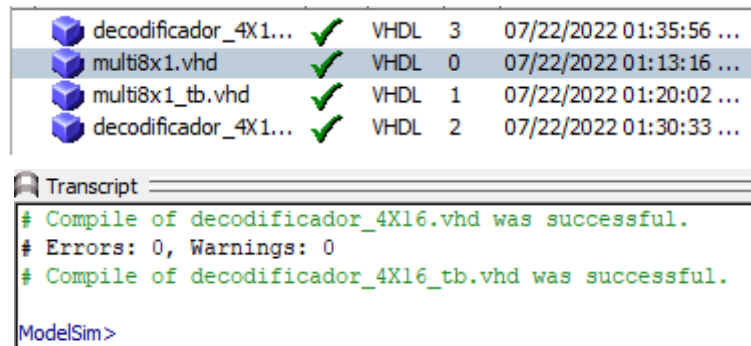


Imagem 9. Mensagem de compilação da questão 1

Simulação

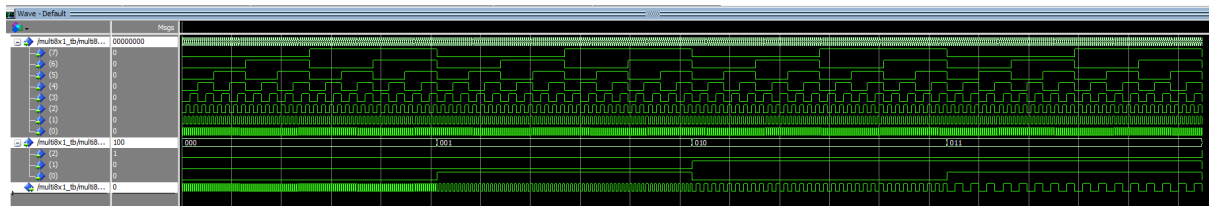


Figura 10. Simulação em forma de onda binária da questão 1.

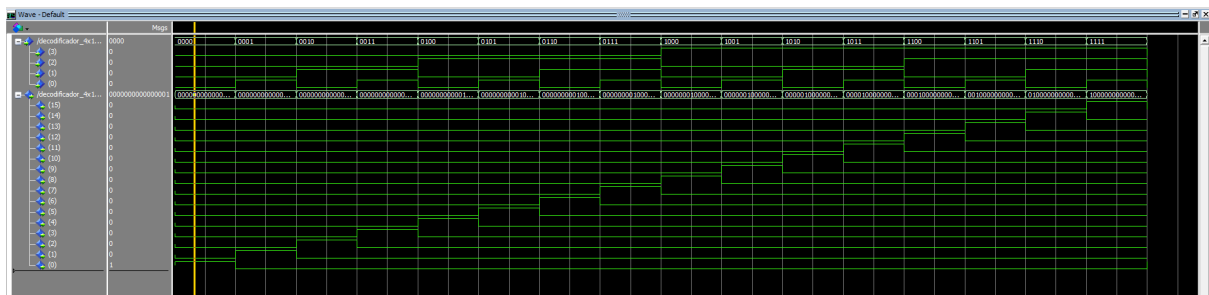


Figura 11. Simulação em forma de onda binária da questão 2.

Análise

Para determinar a ordem dos bits utilizados na simulação e para abranger todas as combinações possíveis, foram utilizados clocks. No projeto 1, foi decidido um tempo de espera de 5 segundos entre cada iteração, enquanto no código 2, foi decidido um tempo de espera de 10 segundos. Pelas simulações geradas, é possível notar que obtemos resultados esperados em ambos os casos, seguindo as tabelas verdades presentes na teoria.

Conclusão

Neste experimento conseguimos descrever com sucesso o comportamento de duas entidades muito utilizadas no mundo da tecnologia - Um multiplexador 8×1 e um decodificador 4×16 . Não foram encontrados erros ou divergências na obtenção dos resultados ou em qualquer etapa dos experimentos, sendo todos os resultados conforme os esperados.