

Experimento 3

João Pedro Fernandes Santos

222025342

TURMA – 09

Introdução

Neste experimento, trabalhamos com a linguagem VHDL, uma ferramenta essencial para a descrição e simulação de circuitos digitais. O foco principal foi entender e aplicar dois tipos importantes de estruturas de atribuição: as condicionais (when-else) e as seletivas (with-select). Essas construções nos permitem expressar decisões dentro de circuitos combinacionais, o que é fundamental no desenvolvimento de módulos digitais mais complexos.

Ao longo da atividade, implementamos dois circuitos clássicos: um multiplexador 8x1, utilizando atribuições condicionais, e um decodificador 4x16, com atribuições seletivas. Com isso, não só colocamos em prática os conceitos teóricos estudados em aula, como também nos familiarizamos com o ambiente de simulação ModelSim, essencial para validar se o comportamento do nosso código está de acordo com a lógica esperada. A realização deste experimento ajudou a reforçar nossa compreensão sobre lógica combinacional, ao mesmo tempo em que introduziu a metodologia de descrição de hardware.

QUESTÃO 1

Teoria

A primeira atividade propôs a implementação de um multiplexador 8x1 utilizando atribuições condicionais na linguagem VHDL. Um multiplexador é um circuito combinacional que seleciona uma, entre várias entradas, para ser encaminhada à saída, com base em sinais de controle. No caso do multiplexador 8x1, temos 8 entradas de dados (D0 a D7), uma saída (Y) e três bits de seleção (S2, S1, S0), que determinam qual das entradas será encaminhada à saída.

No VHDL, utilizamos a estrutura when-else para descrever esse comportamento condicional. Essa forma de atribuição permite expressar de maneira direta qual valor deve ser atribuído à saída com base em diferentes combinações do vetor de seleção. Essa abordagem é ideal para circuitos em que há uma clara correspondência entre combinações de entrada e valores de saída, como é o caso do multiplexador.

<i>S</i>	<i>Y</i>
000	<i>D</i> ₀
001	<i>D</i> ₁
010	<i>D</i> ₂
011	<i>D</i> ₃
100	<i>D</i> ₄
101	<i>D</i> ₅
110	<i>D</i> ₆
111	<i>D</i> ₇

Figura 1. Tabela-verdade do multiplexador 8x1. A entrada *D* é formada pela concatenação dos bits *D*7, *D*6, ..., *D*0, sendo *D*7 o mais significativo e *D*0 o menos significativo.

Código

O código descreve, em VHDL, a implementação do multiplexador 8 x 1 utilizando a estrutura de atribuição condicional (when-else). A entidade MEUCIRCUITO2 possui três portas principais: um vetor de seleção *S*, com 3 bits (de 2 downto 0); um vetor de dados de entrada *D*, com 8 bits (de 7 downto 0); e uma saída *Y* do tipo STD_LOGIC.

No bloco architecture, é feita a associação entre os valores de seleção *S* e a entrada correspondente do vetor *D*. Cada linha da estrutura when-else verifica uma combinação específica dos bits de *S*, indo da menor (000) até a maior (111), e define qual bit de *D* será atribuído à saída *Y*. Por exemplo:

- Quando *S* = "000", a saída *Y* recebe o valor de *D*(0);
- Quando *S* = "001", *Y* recebe *D*(1);
- E assim sucessivamente, até *S* = "111", que direciona *D*(7) para *Y*.

O uso do when-else é adequado nesse caso, pois permite representar o funcionamento do multiplexador de forma sequencial e condicional, seguindo a lógica da tabela-verdade do circuito.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MEUCIRCUITO2 IS
PORT (S: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
D:IN STD_LOGIC_VECTOR(7 DOWNTO 0);
Y: OUT STD_LOGIC);
END MEUCIRCUITO2;

ARCHITECTURE MEUCIRCUITO_ARCH2 OF MEUCIRCUITO2 IS
BEGIN
Y <= D(0) WHEN (S(0) = '0' AND S(1) = '0' AND S(2) = '0') ELSE
D(1) WHEN (S(0) = '1' AND S(1) = '0' AND S(2) = '0') ELSE
D(2) WHEN (S(0) = '0' AND S(1) = '1' AND S(2) = '0') ELSE
D(3) WHEN (S(0) = '1' AND S(1) = '1' AND S(2) = '0') ELSE
D(4) WHEN (S(0) = '0' AND S(1) = '0' AND S(2) = '1') ELSE
D(5) WHEN (S(0) = '1' AND S(1) = '0' AND S(2) = '1') ELSE
D(6) WHEN (S(0) = '0' AND S(1) = '1' AND S(2) = '1') ELSE
D(7) WHEN (S(0) = '1' AND S(1) = '1' AND S(2) = '1');
END MEUCIRCUITO_ARCH2;

```

Figura 2. Código multiplexador 8x1 em VHDL.

Compilação

Após a implementação do código, foi realizada a compilação no ModelSim para verificar a correção do programa. Esse processo é fundamental para garantir que o código está escrito de forma adequada e pode ser simulado corretamente. Como mostrado nas imagens a seguir, nenhum erro de sintaxe foi identificado, indicando que todas as instruções estão de acordo com as regras da linguagem VHDL.

```

# Compile of EXP_3_1.vhd was successful.
# Compile of EXP_3_2.vhd was successful.

```

Figura 4. Compilação mostrando 0 erros, 0 avisos.

Simulação

A figura a seguir apresenta o resultado da simulação do multiplexador 8x1 no ModelSim. Nela, é possível observar o comportamento dos sinais de entrada S (vetor de seleção), D (dados de entrada) e da saída Y.

Durante a simulação, o vetor S varia suas combinações de forma sistemática, percorrendo diferentes valores de 0 a 7 (em binário). A cada mudança em S, a saída Y assume o valor correspondente da entrada D selecionada. Por exemplo, quando S = "101", o multiplexador direciona o valor de D(5) para Y. Como mostrado na simulação, essa relação é respeitada em todos os instantes de tempo analisados.

Além disso, o padrão do vetor D foi definido como uma sequência alternada de 1s e 0s (10101010), facilitando a identificação visual de que a seleção das entradas está sendo feita corretamente. As transições na saída Y coincidem perfeitamente com as mudanças

no vetor de seleção S, comprovando que o circuito implementado atende fielmente à tabela-verdade prevista para o multiplexador 8x1.

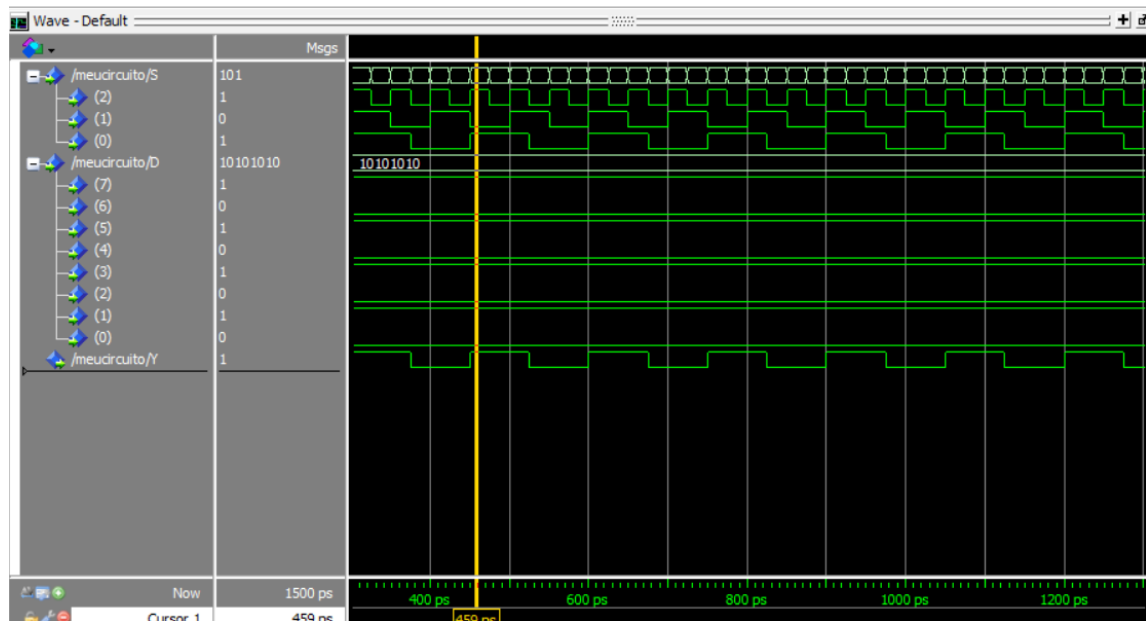


Figura 5. Simulação de variação dos valores de entrada e saída do programa.

Análise

A partir da simulação realizada no ModelSim, foi possível verificar que o multiplexador 8x1 implementado em VHDL com atribuições condicionais está operando corretamente. Cada combinação do vetor de seleção S resultou na saída do valor correspondente no vetor D, conforme definido na tabela-verdade. A variação dos sinais ao longo do tempo mostrou-se coerente, sem atrasos perceptíveis ou comportamentos indesejados. A escolha do padrão alternado nos dados de entrada facilitou a identificação das transições corretas na saída. Isso demonstra que a lógica condicional when-else foi aplicada de forma adequada e que o circuito foi corretamente interpretado pelo simulador.

Conclusão

A implementação do multiplexador 8x1 permitiu aplicar na prática o conceito de atribuições condicionais na linguagem VHDL. O exercício contribuiu significativamente para a compreensão do funcionamento de um dos principais blocos combinacionais da eletrônica digital. Através da simulação, foi possível validar o comportamento esperado do circuito, reforçando a importância da verificação funcional em projetos digitais. A atividade também serviu como base para o uso de estruturas sequenciais na construção de circuitos mais complexos em experiências futuras.

QUESTÃO 2

Teoria

Nesta etapa do experimento, foi proposto o desenvolvimento de um decodificador 4 x 16, utilizando atribuições seletivas (with-select) na linguagem VHDL. Um decodificador é um circuito combinacional que converte um código binário de entrada em um único sinal de saída ativado, de acordo com a combinação recebida. No caso do decodificador 4x16, o circuito possui um vetor de entrada de 4 bits (totalizando 16 combinações possíveis) e um vetor de saída de 16 bits, no qual apenas um dos bits está em nível lógico alto (1) para cada valor de entrada.

A estrutura with-select é especialmente adequada para essa situação, pois permite atribuir um valor específico à saída Y com base no valor da entrada A, de forma mais organizada e legível, com base na tabela-verdade.

<i>A</i>	<i>Y</i>
0000	0000000000000001
0001	0000000000000010
0010	0000000000000100
0011	0000000000001000
0100	0000000000010000
0101	0000000000100000
0110	0000000010000000
0111	0000000010000000
1000	0000000100000000
1001	0000001000000000
1010	0000010000000000
1011	0000100000000000
1100	0001000000000000
1101	0010000000000000
1110	0100000000000000
1111	1000000000000000

Figura 2 - Tabela-verdade do decodificador 4x16

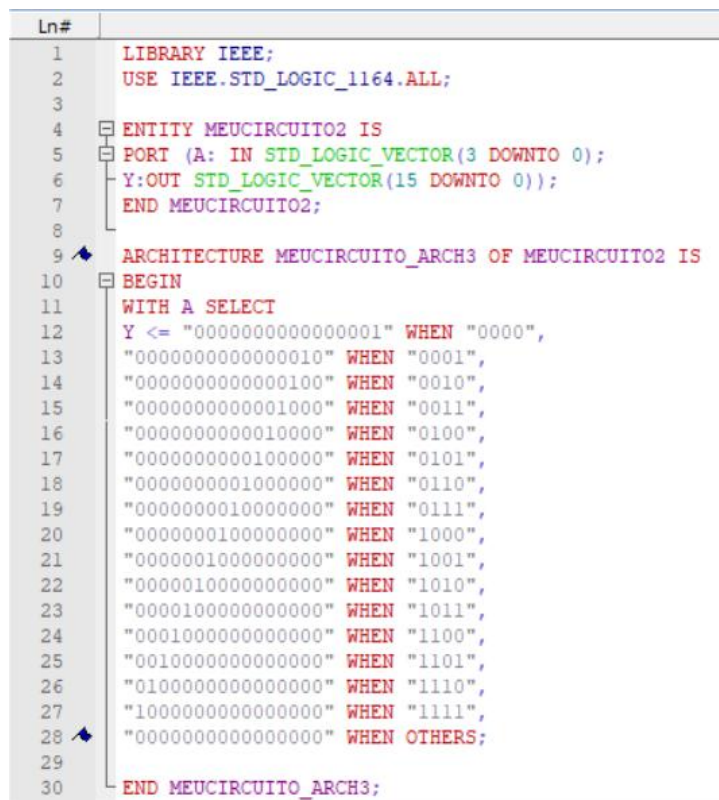
Código

O código desenvolvido define a entidade MEUCIRCUITO2, com um vetor de entrada A(3 downto 0) e um vetor de saída Y(15 downto 0). Dentro da arquitetura MEUCIRCUITO_ARCH3, utilizamos a estrutura with A select para atribuir um valor à saída Y dependendo da combinação binária de A.

Cada linha da instrução with-select associa diretamente uma string binária de 16 bits à variável Y, ativando uma única posição correspondente ao valor binário de entrada. Por exemplo:

- Se A = "0000", então Y = "0000000000000001" (ativa Y(0));
- Se A = "1000", então Y = "0000000100000000" (ativa Y(8));
- E assim por diante, até A = "1111", que ativa o bit mais significativo Y(15).

Um caso adicional when others é incluído para cobrir situações não esperadas (como condições indefinidas), garantindo que a saída seja zerada caso a entrada não esteja claramente definida.



```
Ln# |
1  | LIBRARY IEEE;
2  | USE IEEE.STD_LOGIC_1164.ALL;
3  |
4  | ENTITY MEUCIRCUITO2 IS
5  | PORT (A: IN STD_LOGIC_VECTOR(3 DOWNT0 0);
6  |       Y:OUT STD_LOGIC_VECTOR(15 DOWNT0 0));
7  | END MEUCIRCUITO2;
8  |
9  | ARCHITECTURE MEUCIRCUITO_ARCH3 OF MEUCIRCUITO2 IS
10 | BEGIN
11 |   WITH A SELECT
12 |     Y <= "0000000000000001" WHEN "0000",
13 |         "0000000000000010" WHEN "0001",
14 |         "0000000000000100" WHEN "0010",
15 |         "0000000000001000" WHEN "0011",
16 |         "0000000000010000" WHEN "0100",
17 |         "0000000000010000" WHEN "0101",
18 |         "0000000000100000" WHEN "0110",
19 |         "0000000001000000" WHEN "0111",
20 |         "0000000100000000" WHEN "1000",
21 |         "0000001000000000" WHEN "1001",
22 |         "0000010000000000" WHEN "1010",
23 |         "0000100000000000" WHEN "1011",
24 |         "0001000000000000" WHEN "1100",
25 |         "0010000000000000" WHEN "1101",
26 |         "0100000000000000" WHEN "1110",
27 |         "1000000000000000" WHEN "1111",
28 |         "0000000000000000" WHEN OTHERS;
29 |
30 | END MEUCIRCUITO_ARCH3;
```

Figura 8. Código decodificador 4x16 em VHDL.

Compilação

Após a implementação do código, foi realizada a compilação no ModelSim para verificar a correção do programa. Esse processo é fundamental para garantir que o código está escrito de forma adequada e pode ser simulado corretamente. Como mostrado nas imagens a seguir, nenhum erro de sintaxe foi identificado, indicando que todas as instruções estão de acordo com as regras da linguagem VHDL.

```
# Compile of EXP_3_1.vhd was successful.
# Compile of EXP_3_2.vhd was successful.
```

Figura 4. Compilação mostrando 0 erros, 0 avisos.

Simulação

A simulação apresentada na imagem do ModelSim mostra o comportamento do decodificador ao longo do tempo, à medida que a entrada A assume diferentes valores binários. Como podemos observar, a saída Y reflete corretamente a ativação de apenas um bit para cada combinação de A.

No exemplo destacado pelo cursor , a entrada A = "1000" resulta na ativação de Y(8), como mostra a saída: 00000000100000000, o que está de acordo com a tabela-verdade do decodificador 4x16. As demais posições de Y permanecem em nível lógico baixo (0), como esperado.

Esse comportamento confirma que o circuito está funcionando conforme especificado e que o código VHDL está corretamente implementado.

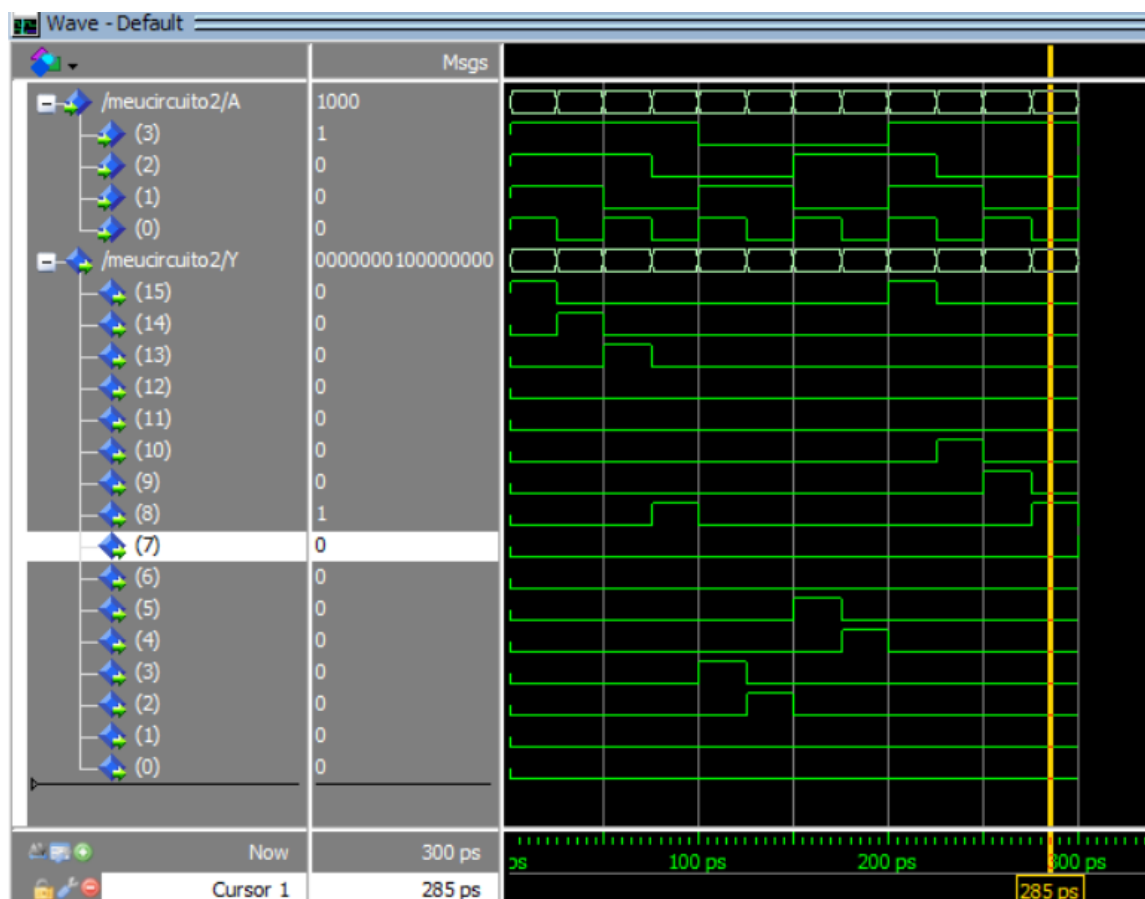


Figura 10. Simulação de variação dos valores de entrada e resposta dos valores de saída.

Análise

A simulação confirma que o decodificador 4x16 está operando corretamente. A estrutura with-select facilitou a implementação e organização do código, tornando as relações entre entrada e saída claras e diretas. A cada variação da entrada A, apenas um único bit da saída Y é ativado, evidenciando o comportamento esperado de um decodificador. A precisão da transição entre os estados e a ausência de conflitos ou múltiplas saídas ativadas indicam que o código foi interpretado corretamente pelo compilador e simulador.

Conclusão

A atividade do decodificador foi essencial para reforçar o uso da estrutura seletiva na linguagem VHDL e para consolidar o entendimento do funcionamento das estruturas “with-select” e “when-else”. A implementação mostrou-se simples, direta e eficiente, demonstrando que a linguagem VHDL permite representar de forma clara o comportamento de circuitos digitais. Além disso, a simulação bem-sucedida confirmou que o código atende integralmente à lógica prevista, o que válida tanto a implementação quanto o aprendizado obtido durante o experimento.