Universidade de Brasília
Departamento da Engenharia Elétrica
Laboratório de Sistemas Digitais

Relatório 03

Introdução à linguagem VHDL

José Antônio Alcântara da Silva de Andrade Mat: 232013031

Professor:
Eduardo B. R. F. Paiva
Turma 08

Brasília, DF

18 de novembro de 2024

1 Objetivos

1. Implementar circuitos combinacionais simples utilizando atribuições condicionais e seletivas da linguagem VHDL.
2. Desenvolver módulos básicos — somador completo e multiplexador — que podem ser usados futuramente para implementar circuitos mais complexos.
3. Simulação no ModelSim.

2 Atividades

Essa sessão do laboratório tinha como objetivo a resolução de dois exercícios: a modelagem de um multiplexador 8x1 e de um decodificador de 16 bits.

2.1 Exercício 1

O primeiro exercício consiste na modelagem de um multiplexador 8x1.

No exercício, é requisitado a implementação usando-se especificamente expressões "when-else".

2.1.1 Modelagem do Problema

O exercício 1, mais especificamente, demanda a construção de um sistema tal que a tabela verdade siga a especificação da Tabela 1.

Tabela 1: Tabela verdade requisitada pelo exercício 1.

S	Y
000	D_0
001	D_1
010	D_2
011	D_3
100	D_4
101	D_5
110	D_6
111	D_7

Para tal, primeiramente define-se um vetor de entrada D de 8 bits, um vetor de entrada S de 3 bits (o que decide qual entrada D sairá) e uma saída simples Y. Segue-se, na Listagem 1, os pré-requisitos do `CIRCUIT1`, que modelará a Tabela 1.

Listagem 1: Requisitos para funcionalidade do circuito 1.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity CIRCUIT1 is
5     port (S: in std_logic_vector(2 downto 0);
6           D: in std_logic_vector(7 downto 0);
7           Y: out std_logic);
8 end CIRCUIT1;
```

Define-se os requisitos gerais de um programa VHDL nas linhas 1 e 2, seguidos do encapsulamento da definição do `CIRCUIT1` nas linhas 4 e 8; todas as linhas da Listagem 1.

Seguindo a análise da Listagem 1, para as definições de entrada, primeiro cria-se o vetor de 3 bits S na linha 5, e então o vetor de 8 bits D na linha 6. Para saída, define-se um bit simples Y na linha 7.

2.1.2 Implementação

Para a implementação, foi requisitado o uso de expressões "when-else". Tais atuam como um simples "if" de outras linguagens de programação: se uma certa expressão lógica for verdade, então o resultado indicado será transmitido, senão outro resultado será transmitido.

No multiplexador 8x1, usou-se a cadeia de "when-else" na arquitetura `CIRTUIT1_ARCH` representada na Listagem 2.

Listagem 2: Implementação da arquitetura do circuito 1.

```
1  architecture CIRCUIT1_ARCH of CIRCUIT1 is
2  begin
3      Y <= D(0) when S = "000" else
4           D(1) when S = "001" else
5           D(2) when S = "010" else
6           D(3) when S = "011" else
7           D(4) when S = "100" else
8           D(5) when S = "101" else
9           D(6) when S = "110" else
10          D(7) when S = "111";
11 end CIRCUIT1_ARCH;
```

As linhas 1 e 11 da Listagem 2 encapsulam a arquitetura, e a linha 2 indica o início das associações aos bits (de saída).

A estrutura do multiplexador segue, implementado entre as linhas 3 e 10 da Listagem 2. Associa-se cada entrada D com o valor equivalente de S em uma expressão lógica simples. Por exemplo, como visto na Tabela 1 linha 4, temos que o valor 011 do vetor S deve ser associado com D_3 — exatamente o que ocorre na linha 6: se $S =$ "011" então Y deve assumir D_3.

O código está completamente implementado agora, como segue na Listagem 3.

Listagem 3: Código final do circuito 1.

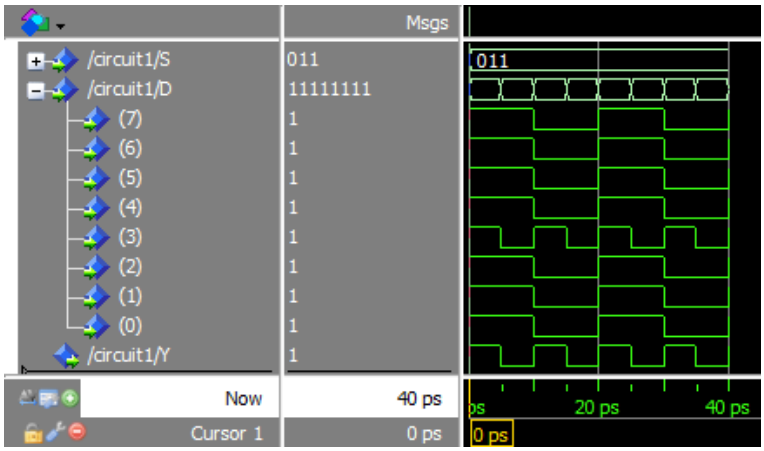
```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity CIRCUIT1 is
5      port (S: in std_logic_vector(2 downto 0);
6            D: in std_logic_vector(7 downto 0);
7            Y: out std_logic);
8  end CIRCUIT1;
9
10 architecture CIRCUIT1_ARCH of CIRCUIT1 is
11 begin
12     Y <= D(0) when S = "000" else
13          D(1) when S = "001" else
14          D(2) when S = "010" else
15          D(3) when S = "011" else
16          D(4) when S = "100" else
17          D(5) when S = "101" else
18          D(6) when S = "110" else
19          D(7) when S = "111";
20 end CIRCUIT1_ARCH;
```

2.1.3 Simulação

Para a simulação do circuito 1, escolhe-se um valor para análise, e força-se tal no vetor de entrada S, e então seleciona-se um período de oscilação específico para cada bit da entrada D, a fim de demonstrar que apenas um tem seu sinal transmitido.

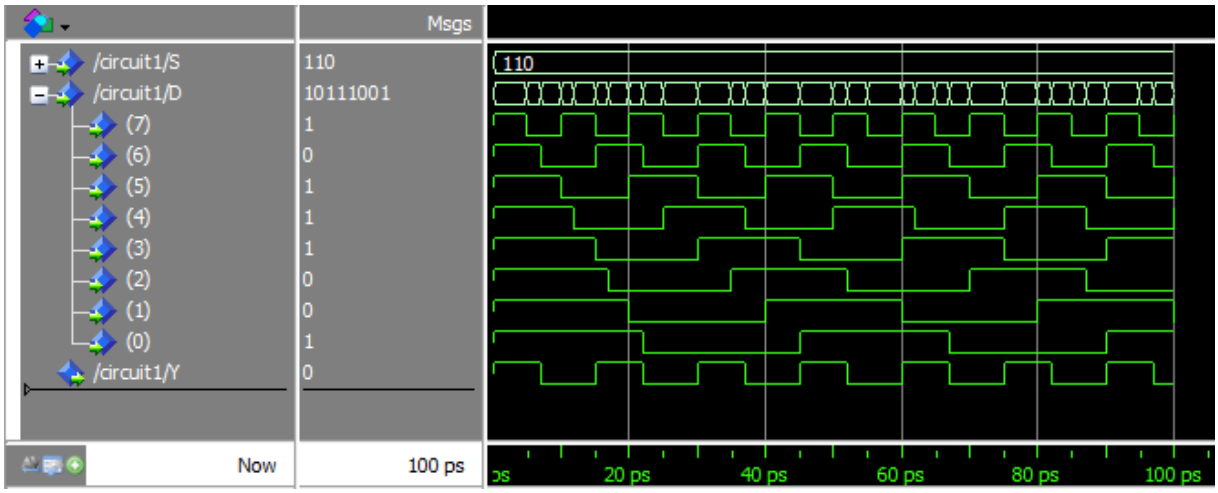
Na primeira simulação, na Figura 1, simulou por 40ps um sistema em que selecionou-se a entrada 011 para o vetor S, uma oscilação de 10 ps para D_3 e 20 ps para todas as outras entradas. Note como a saída Y segue a oscilação de D_3 apenas.

Figura 1: Simulação 1 do circuito 1.



Na segunda simulação, na Figura 2, forçou-se a entrada 110 para o vetor S. Já para o vetor D, fez-se com que o bit de maior significância D_7 oscilasse com um período de 10ps, o próximo, D_6, com 15ps, D_5 com 20ps, e assim por diante. Nota-se, novamente, que o valor obtido na saída Y equivale-se à oscilação do bit do vetor D representado pela entrada S. Nesse caso, $110_2 = 6_{10} \Rightarrow Y = D_6$.

Figura 2: Simulação 2 do circuito 1.



Em ambos os casos, o valor resultante da saída Y é o bit de D correspondente ao valor binário da entrada S. Analogamente, esse resultado é generalizado à toda entrada S.

2.2 Exercício 2

O segundo exercício consiste na modelagem de um decodificador de 4 bits.

No exercício, é requisitado a implementação usando-se especificamente expressões "with-select".

2.2.1 Modelagem do Problema

Novamente, a tabela verdade do exercício é fornecida, e o objetivo resta apenas em sua implementação no sistema. Segue, na Tabela 2, o sistema requerido pelo exercício.

Para a modelagem do circuito do exercício, será necessário um vetor de entrada A de 4 bits e um vetor de saída Y de 16 bits. Realiza-se isso na Listagem 4.

Tabela 2: Tabela verdade requisitada pelo exercício 2.

A	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

Listagem 4: Requisitos para funcionalidade do circuito 2.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity CIRCUIT2 is
5      port (A: in std_logic_vector(3 downto 0);
6            Y: out std_logic_vector(15 downto 0));
7  end CIRCUIT2;
```

A maioria das linhas da Listagem 4 seguem a mesma lógica de suas equivalentes na Listagem 1. As diferenças se encontram na entrada e saída da Listagem 4: esta, na linha 5, agora um vetor de entrada A de 4 bits e aquela, na linha 6, um vetor de saída Y de 16 bits. Com isso, pode-se construir a arquitetura `CIRCUIT2_ARCH` do `CIRCUIT2`.

2.2.2 Implementação

Novamente, fora requisitado um método específico para a solução do sistema. Neste caso, usando expressões "with-select". Seu papel é apenas reduzir a redundância quando escreve-se as expressões "when-else", como será demonstrado na Listagem 5.

Listagem 5: Implementação da arquitetura do circuito 2.

```
1  architecture CIRCUIT2_ARCH of CIRCUIT2 is
2  begin
3      with A select
4      Y <= "0000000000000001" when "0000",
5           "0000000000000010" when "0001",
6           "0000000000000100" when "0010",
7           "0000000000001000" when "0011",
8           "0000000000010000" when "0100",
9           "0000000000100000" when "0101",
10          "0000000001000000" when "0110",
11          "0000000010000000" when "0111",
12          "0000000100000000" when "1000",
13          "0000001000000000" when "1001",
14          "0000010000000000" when "1010",
15          "0000100000000000" when "1011",
16          "0001000000000000" when "1100",
17          "0010000000000000" when "1101",
18          "0100000000000000" when "1110",
19          "1000000000000000" when "1111",
20          "0000000000000000" when others;
21 end CIRCUIT2_ARCH;
```

As linhas 1, 2 e 21 da Listagem 5 servem como definição da arquitetura do `CIRCUIT2`. Em seguida, usa-se a expressão "with-select" como requisitado, na linha 3 da Listagem 5. Seu propósito é definir que as análises condicionais serão realizadas com o vetor A.

Adiciona-se, então, as análises condicionais que implementam a Tabela 2, da linha 4 até a linha 19 da Listagem 5. Por exemplo, a linha 9 da Listagem 5 implementa o caso 5 (linha 6 da Tabela 2): quando o vetor de entrada for 0101 (5 em decimal), o vetor Y deve assumir o valor 0000 0000 0010 0000.

Finalmente, adiciona-se um caso excepcional na linha 20 da Listagem 5 para a funcionalidade do código. Usualmente, deve-se entregar um caso padrão para associar à saída quando nenhuma das entradas for correspondida numa expressão "with-select". Contudo, devido à natureza do decodificador, isso se torna desnecessário, já que todas as entradas possíveis estão associadas. Foi decidido, portanto, representar esse caso excepcional como uma saída completamente nula.

Com isso, a Tabela 2 está completamente implementado. O código final segue na Listagem 6.

Listagem 6: Código final do circuito 2.

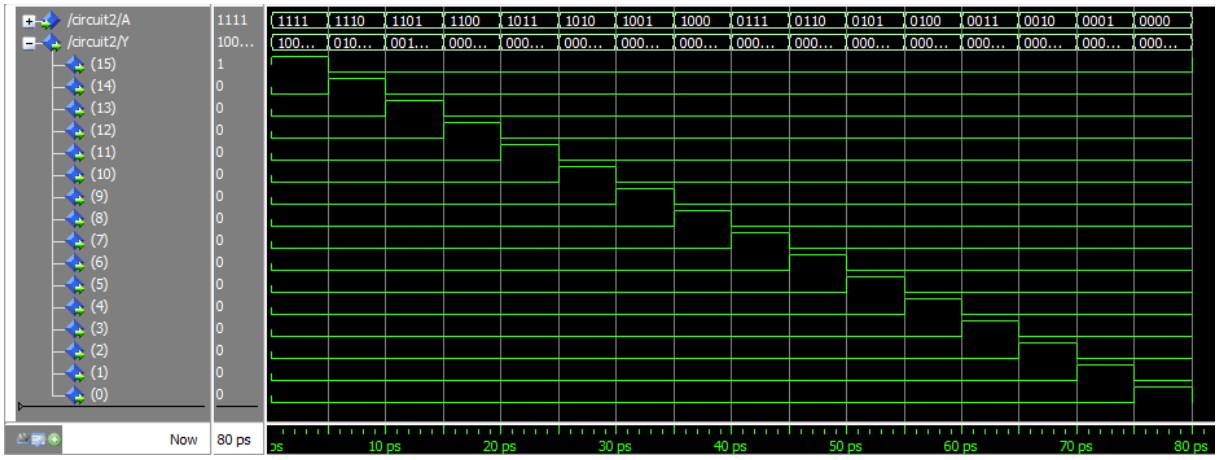
```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity CIRCUIT2 is
5      port (A: in std_logic_vector(3 downto 0);
6              Y: out std_logic_vector(15 downto 0));
7  end CIRCUIT2;
8
9  architecture CIRCUIT2_ARCH of CIRCUIT2 is
10 begin
11     with A select
12     Y <= "0000000000000001" when "0000",
13          "0000000000000010" when "0001",
14          "0000000000000100" when "0010",
15          "0000000000001000" when "0011",
16          "0000000000010000" when "0100",
17          "0000000000100000" when "0101",
18          "0000000001000000" when "0110",
19          "0000000010000000" 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 end CIRCUIT2_ARCH;
```

2.2.3 Simulação

Para a simulação do código (e do decodificador em si) usa-se de um relógio em cada bit do vetor de entrada A. Faz-se com que cada bit oscile com um período duas vezes maior do que o bit de significância menor anterior, a fim de obter todos os casos de valores para o vetor A. Iniciando-se a simulação, obtém-se, então, o resultado da Figura 3.

Nota-se que em cada caso de entrada do vetor A apenas um dos bits do vetor Y é ligado, e sempre aquele o qual, após a transformação para o valor decimal, A representa. Por exemplo, quando a entrada A é 0110 — o período de 45 a 50 ps na Figura 3 — o bit 6 do vetor Y liga ($0110_2 = 6_{10}$).

Figura 3: Simulação do circuito 2.

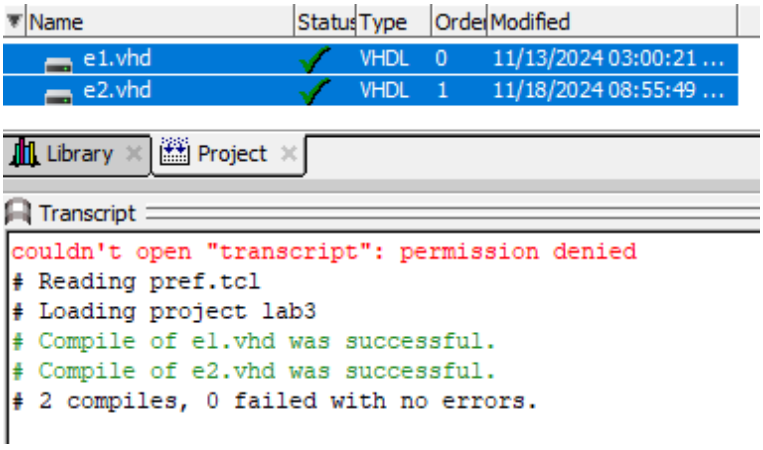


2.3 Extra

2.3.1 Compilação

Não foi encontrado nenhum erro de compilação nas Listagens 3 e 6, como visto na Figura 4.

Figura 4: Compilação dos códigos.



2.3.2 Simplificações

É possível simplificar a leitura da Listagem 5 ao utilizar de números hexadecimais para a saída Y, como demostrado na Listagem 7.

Listagem 7: Código alternativo do circuito 2.

```
architecture CIRCUIT2_ARCH of CIRCUIT2 is
begin
    with A select
    Y <= X"0001" when "0000",
         X"0002" when "0001",
         X"0004" when "0010",
         X"0008" when "0011",
         X"0010" when "0100",
         X"0020" when "0101",
         X"0040" when "0110",
         X"0080" when "0111",
         X"0100" when "1000",
         X"0200" when "1001",
```

```
14          X"0400" when "1010",
15          X"0800" when "1011",
16          X"1000" when "1100",
17          X"2000" when "1101",
18          X"4000" when "1110",
19          X"8000" when "1111",
20          X"0000" when others;
21  end CIRCUIT2_ARCH;
```

3 Conclusão

O uso das expressões “when-else” e “with-select” é essencial para a programação em VHDL, uma vez que seu uso muito facilita na construção de sistemas digitais de grande porte. Além disso, também simplificam a leitura de códigos, reduzindo o processamento mental requerido para a compreensão de uma expressão.