Universidade de Brasília
Departamento da Engenharia Elétrica
Laboratório de Sistemas Digitais

Relatório 02

Introdução à linguagem VHDL

José Antônio Alcântara da Silva de Andrade Mat: 232013031

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

Brasília, DF

2 de novembro de 2024

1 Objetivos

1. Implementar circuitos combinacionais simples utilizando a 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 — os mesmos da sessão anterior — contudo exercendo um foco no uso do software ModelSim para a construção e simulação dos circuitos.

2.1 Exercício 1

O exercício 1 é dividido em três partes, mas todas são relacionadas à construção de um somador completo.

2.1.1 Modelagem do problema

Para a equação lógica 1, requisita-se a construção de um sistema com três entradas (A, B, C_{in}) e uma saída (S), tal que a expressão lógica 1 se cumpra:

$$S = A \oplus B \oplus C_{in} \tag{1}$$

A equação lógica 2 é implementada com as mesmas entradas, mas com uma saída diferente (C_{out}).

$$C_{out} = AB + AC_{in} + BC_{in} \tag{2}$$

Então, a fim de adicionar todas as entradas e saídas necessárias à arquitetura desse circuito, cria-se um novo arquivo `.vhd` (da linguagem VHDL) e adicionou-se as linhas descritas na listagem 1.

Listagem 1: Implementação das entradas e saídas.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity CIRCUIT1 is
5     port (A, B, Cin: in std_logic;
6           S, Cout: out std_logic);
7 end CIRCUIT1;
```

As linhas 1.1 e 1.2 são essenciais à todo código de VHDL, introduzindo as bibliotecas necessárias para a simulação de circuitos lógicos. Em seguida, as linhas 1.4 e 1.7 encapsulam as definições do circuito (aqui definido como) `CIRCUIT1`. Na linha 1.5 há as definições das entradas A, B e Cin, as três definidas como apenas um bit (na biblioteca, definido como `std_logic`), já na linha 1.6 temos as duas saídas S e $Cout$ também definidas como apenas um bit.

2.1.2 Implementação

Agora, para a implementação das operações do circuito, usa-se o código da listagem 2.

Listagem 2: Implementação dos circuitos do somador.

```
1  architecture CIRCUIT1_ARCH of CIRCUIT1 is
2  begin
3      S <= A xor B xor Cin;
4      Cout <= (A and B) or (A and Cin) or (B and Cin);
5  end CIRCUIT1_ARCH;
```

Na linha 2.1 define-se o nome da arquitetura do `CIRCUIT1`, aqui definida como `CIRCUIT1_ARCH`. As linhas 2.2 até 2.5 definem as operações do circuito. Finalmente, implementa-se a equação 1 na linha 2.3 e a equação 2 na linha 2.4. O resultado final será o código na listagem 3.

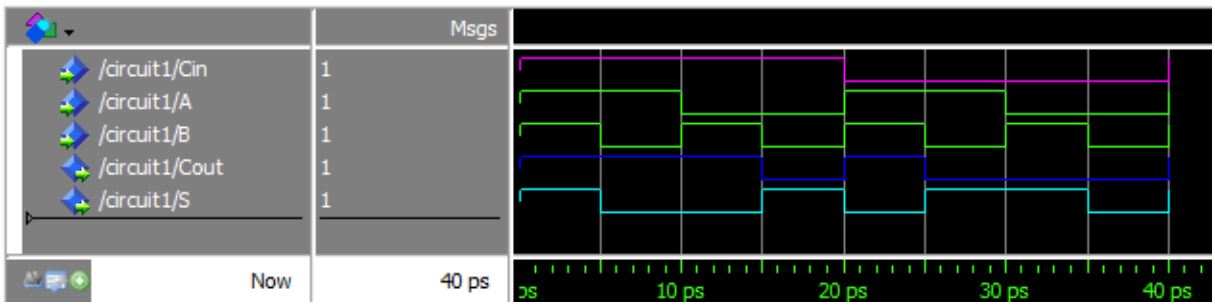
Listagem 3: Código final do somador.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity CIRCUIT1 is
5      port (A, B, Cin: in std_logic;
6            S, Cout: out std_logic);
7  end CIRCUIT1;
8
9  architecture CIRCUIT1_ARCH of CIRCUIT1 is
10 begin
11     S <= A xor B xor Cin;
12     Cout <= (A and B) or (A and Cin) or (B and Cin);
13 end CIRCUIT1_ARCH;
```

2.1.3 Simulação

Finalmente, dirige-se ao ModelSim para realizar a simulação da implementação do sistema. Após a compilação do arquivo `.vhd` no programa (vide figura 3), programa-se as entradas A, B e C_{in} de tal forma que B oscile com um relógio de 10ps, A com 20ps e C_{in} com 40ps. Ao garantir que cada entrada tenha uma variação em potências de 2 (no formato $k \cdot 2^i$ com $k = 10$ e $i \in \mathbb{Z}_-$) garante-se não só que as variações ocorram nas marcações predefinidas do programa, como também permite que todos as combinações possíveis para $C_{in}AB$ sejam atingidas. Realizando a simulação por 40ps, obtém-se o resultado da figura 1.

Figura 1: Simulação do circuito 1.



Na figura 1, as entradas se encontram em verde, o valor *carry in* C_{in} em magenta, a saída S se encontra em ciano e a saída *carry out* C_{out} encontra-se em azul. Nota-se que todos os valores de $C_{in}AB$ foram representados figura 1, ou seja, é possível analisar os valores da onda e assim construir uma tabela verdade (vide tabela 1).

Tabela 1: Tabela verdade do circuito 1.

$C_{in}AB$	$C_{out}S$
111	11
110	10
101	10
100	01
011	10
010	01
001	01
000	00

Mais especificamente, podemos associar o período 0–5ps na figura 1 representa o bit 111 (escrito no formato $C_{in}AB$), com o resultado 1 no *carry out* C_{out} e resultado 1 no somador S, exatamente como o a linha 1 da tabela 1. O período 5–10ps representa, então, o bit 110 com as saídas 1 e 0, como na linha 2 da tabela 1, e assim por diante.

Nota-se que C_{out} e S implementa um somador com *carry in*carry out*. Por exemplo, suponha a entrada 101, presente no intervalo 10–15ps. Temos que a soma $C_{in} + A + B = 1 + 0 + 1 = 10$, ou seja, o somador S deve retornar 0, e o *carry out* C_{out} deve retornar 1. Nota-se esse exato comportamento na linha 3 da tabela 1.

2.2 Exercício 2

O exercício 2 segue a mesma estrutura de solução do exercício 1, apenas agora requisitando a implementação de um multiplexador.

2.2.1 Modelagem do problema

Para a equação lógica 3 precisa-se construir um sistema com seis entradas e uma saída.

$$Y = D_0\overline{S_1}\overline{S_0} + D_1\overline{S_1}S_0 + D_2S_1\overline{S_0} + D_3S_1S_0 \tag{3}$$

Entretanto, tanto por simplicidade como por requisito do roteiro, implementa-se a entrada D como um vetor de dois bits, e a entrada S com um vetor de quatro bits. Finalmente, a saída Y pode ser implementada como um simples bit. Obtém-se, então, o código da listagem 4.

Listagem 4: Implementação das entradas e saída.

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

As linhas 4.1, 4.2, 4.4 e 4.8 seguem a mesma lógica de suas equivalentes na listagem 1 — a única alteração é que, agora, o circuito é chamado de `CIRCUIT2`.

Na linha 4.5, a entrada S é definida como um vetor (`std_logic_vector` na biblioteca) de 2 bits, com índice mais significativo 1, descendo até o índice 0 (`1 downto 0`). Similarmente, na linha 4.6, D é definido como um vetor de 4 bits com índice mais significativo 3 decrescendo até o índice 0. Por último, na linha 4.7, a saída Y é definida.

Aprofundando-se na construção lógica do circuito, pode-se notar que cada termo da equação lógica 3 atua em duas partes, uma relevante ao vetor D e outra ao vetor S. Aqui, o valor resultante dos bits do vetor S age como um filtro para escolher qual valor

do vetor D será transmitido para a saída. O resultado 00 no vetor S transmite o valor de D_0, 01 o valor de D_1, 10 o de D_2 e, por fim, 11 o de D_3. Essa relação não só é explicitamente vista na tabela 2, como também, posteriormente, na simulação (figura 2).

2.2.2 Implementação

Agora, para a implementação da equação lógica 3, escreveu-se o código da listagem 5.

Listagem 5: Desenvolvimento lógico do circuito.

```
1 architecture CIRCUIT2_ARCH of CIRCUIT2 is
2 signal aux: std_logic_vector(3 downto 0);
3 begin
4     aux(0) <= D(0) and not(S(1)) and not(S(0));
5     aux(1) <= D(1) and not(S(1)) and S(0);
6     aux(2) <= D(2) and S(1) and not(S(0));
7     aux(3) <= D(3) and S(1) and S(0);
8     Y <= aux(0) or aux(1) or aux(2) or aux(3);
9 end CIRCUIT2_ARCH;
```

Novamente, as linhas 5.1, 5.3 e 5.9 seguem a mesma lógica de suas equivalentes na listagem 2, a diferença é que, aqui, define-se a arquitetura do circuito como `CIRCUIT2_ARCH`.

Na linha 5.2 define-se um sinal interno `aux`, o qual atuará como um vetor de 4 bits e índice mais significativo 3, descendo até o 0 (`3 downto 0`). Como será visto adiante, este vetor será usado para armazenar o resultado de cada produto da equação lógica 3, para depois realizar sua soma.

Nas linhas 5.4, 5.5, 5.6 e 5.7, associa-se os termos da equação lógica 3 a cada entrada do vetor auxiliar. Por exemplo, na linha 5.5, associa-se ao valor `aux(1)` com o AND da entrada D equivalente, no caso, $D_1\overline{S_1}S_0$.

Para, então, transmitir a saída Y realiza-se a soma lógica (o operador OR) dos valores individuais do vetor auxiliar, como visto na linha 5.8.

O código final está devidamente representado na listagem 6.

Listagem 6: Código final do multiplexador.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity CIRCUIT2 is
5      port (S: in std_logic_vector(1 downto 0);
6            D: in std_logic_vector(3 downto 0);
7            Y: out std_logic);
8  end CIRCUIT2;
9
10 architecture CIRCUIT2_ARCH of CIRCUIT2 is
11 signal aux: std_logic_vector(3 downto 0);
12 begin
13     aux(0) <= D(0) and not(S(1)) and not(S(0));
14     aux(1) <= D(1) and not(S(1)) and S(0);
15     aux(2) <= D(2) and S(1) and not(S(0));
16     aux(3) <= D(3) and S(1) and S(0);
17     Y <= aux(0) or aux(1) or aux(2) or aux(3);
18 end CIRCUIT2_ARCH;
```

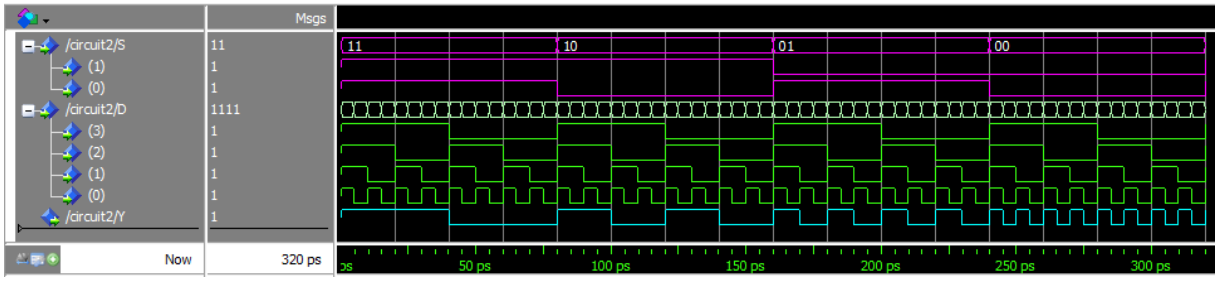
2.2.3 Simulação

Retorna-se, uma vez mais, para o programa ModelSim a fim de realizar uma simulação do sistema. Após compilar o código da listagem 6 (vide figura 3), prepara-se os relógios necessários para obter uma tabela verdade compreensiva do sistema.

Primeiramente, define-se o relógio de D_0 para oscilar em períodos de 10ps. Em seguida, D_1 oscilará em 20ps, D_2 em 40ps e, por fim, D_3 em 80ps. Agora, define-se os valores para o vetor S, com S_0 em 160ps e S_1 em 320ps. Assim, como dito previamente, todos os valores

possíveis serão atingidos. Após realizar a simulação por 320ps, obtém-se o resultado da figura 2.

Figura 2: Simulação do circuito 1.



As entradas S se encontram em magenta, as entradas D em verde e a saída Y em ciano.

Analisando a simulação, nota-se um padrão emergente: a saída Y se assemelha, sempre, à oscilação de alguma das entradas D. Mais especificamente, a semelhança começa com a entrada D_3, quando o valor do vetor S se encontra no 11. Quando ocorre a alteração para o valor 10 do vetor S, a saída Y segue agora a oscilação do termo D_2, e assim adiante. Constrói-se, então, a tabela verdade do sistema, a tabela 2.

Tabela 2: Tabela verdade do circuito 2.

S_1S_0	Y
11	D_3
10	D_2
01	D_1
00	D_0

Relacionando os resultados da tabela 2 com os da figura 2, têm-se que o valor da linha 1 da tabela está presente na região 0–80ps, da linha 2 na região 80–160ps, da linha 3 na região 160–240ps e da linha 4 na região 240–320ps.

2.3 Extra

Durante a compilação dos códigos das listagens 3 e 6, nenhum erro de sintaxe foi detectado, como visto na figura 3.

3 Conclusão

A linguagem VHDL, apesar de demostrar uma dificuldade inicial (visto a necessidade de programas especiais para sua compilação e simulação), apresenta-se como ferramenta extremamente útil para o desenvolvimento de circuitos lógicos e sistemas digitais.

Com isso, a implementação de um somador e de um multiplexador muito auxilia no aprendizado dos pilares da linguagem, o que, por sua vez, garantirá um processo mais fácil durante o desenvolvimento de sistemas mais complexos.

Figura 3: Compilação dos códigos VHDL.