Introdução à VHDL

Responsável: Luiza de Macedo Mourelle

Carga horária: 20 horas

Resumo: Introduzir o conceito de VHDL como linguagem de descrição e modelagem de sistemas digitais.

Conteúdo: VHDL como linguagem de descrição e modelagem de hardware. Conceito de domínio comportamental e estrutural. Sistemas digitais combinacionais em VHDL. Sistemas digitais sequenciais em VHDL. Validação funcional através de simulação.

Bibliografia:

1. The VHDL Cookbook, Peter J. Ashenden;

2. Sistemas Digitais: princípios e aplicações, Ronald J. Tocci, Neal S. Wildmer, Gregory S. Moss;

3. VHDL - Descrição e Síntese de Circuitos Digitais, Roberto d'Amore;

4. Eletrônica Digital Moderna e VHDL, Volnei Pedroni.

Email: ldmm@eng.uerj.br

URL: http://www.eng.uerj.br/~ldmm

Aula 1 − Descrição Comportamental e Estrutural

VHDL significa *Very high speed integrated circuit Hardware Description Language* (Linguagem de Descrição de Hardware para circuito integrado de alta velocidade).

Essa linguagem tornou-se pelo IEEE, em 1987, a linguagem padrão para descrição e modelagem de circuitos digitais.

VHDL permite que um circuito digital seja modelado, validado funcionalmente, através de simulação, e implementado, fisicamente, através de síntese.

A proposta aqui é oferecer uma visão inicial da linguagem, através da modelagem de componentes digitais básicos e sua respectiva simulação.

Vamos trabalhar com níveis de abstração, que nos permitem modelar um componente de acordo com a complexidade que apresenta.

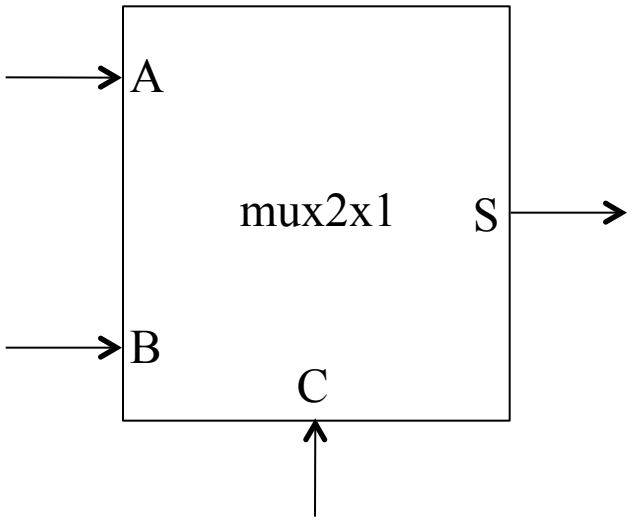
A abstração empregada por projetistas de sistemas digitais pode ser expressa em dois domínios: comportamental e estrutural.

No domínio comportamental, um componente é descrito em termos da relação entre entradas e saídas, abstraindo-se do circuito interno.

Neste domínio, pode-se utilizar uma representação algorítmica, semelhante a uma linguagem de programação, ou fluxo de dados, representando a equação lógica correspondente.

No domínio estrutural, um componente é descrito em termos de sub-componentes internos, interconectados na forma de um circuito.

Exemplo: Considere um multiplexador de 2x1.



Se a entrada C for ‘0’, então a saída S recebe a entrada A, senão a saída S recebe a entrada B.

Vamos, inicialmente, considerar que os sinais A, B e S tem um bit. Obviamente, o sinal C tem um bit também, uma vez que basta um bit para selecionar uma das entradas.

Antes de qualquer iniciativa em descrever o modelo, temos que especificar as portas de entrada e saída, que permitem a interface com o meio externo. Para tal, utilizamos o conceito de entidade (`entity`):

```
entity mux2x1 is
      port (A,B,C: in bit; S: out bit);
end mux2x1;
```

A declaração das portas requer a definição do modo (`in` ou `out`) e do tipo (bit), que define os valores que a porta pode assumir (neste caso, 0 ou 1).

Observe que terminamos um comando com ;.

Outra observação é que VHDL não é sensível a maiúsculas ou minúsculas!

O comportamento do `mux2x1`, na representação algorítmica, é descrito utilizando o conceito de arquitetura (`architecture`):

```
architecture algoritmica of mux2x1 is
begin
      process (A, B, C)
      begin
            if C = '0'
                  then S <= A;
                  else S <= B;
            end if;
      end process;
end algoritmica;
```

Como `C` é um sinal de um bit, o valor binário fica entre apóstrofes.

A atribuição a um sinal é feita através do comando `<=`, como em `S <= A`.

A descrição do componente, em VHDL, é feita dentro do corpo da arquitetura (`architecture`), cujo cabeçalho introduz o identificador desta (`algoritmica`) e a qual componente se refere (`mux2x1`).

Na representação algorítmica, utiliza-se o comando `process`, seguido dos sinais de entrada (lista sensitiva) que estimulam o componente. Observe que `s` é sinal de saída, sendo a resposta do componente, não participando da lista sensitiva.

Veja que nada é dito sobre o circuito interno. Essa abstração, em alguns casos, é interessante, pois permite que se desconsidere a complexidade do componente, que poderia impactar no tempo de modelagem do mesmo.

Na representação em fluxo de dados, precisamos descrever o componente em termos da equação lógica correspondente.

Para isto, utilizamos a tabela verdade abaixo, de onde obtemos a equação lógica baseada na soma de produtos.

A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

$$S = \overline{A}.B.C + A.\overline{B}.\overline{C} + A.B.\overline{C} + A.B.C$$

Esse comportamento, na representação em fluxo de dados, pode ser descrito como:

```
architecture fluxodedados of mux2x1 is

begin
      S <= ((not A)and B and C) or (A and (not B) and
           (not C)) or (A and B and (not C)) or (A and
           B and C);
end fluxo_de_dados;
```

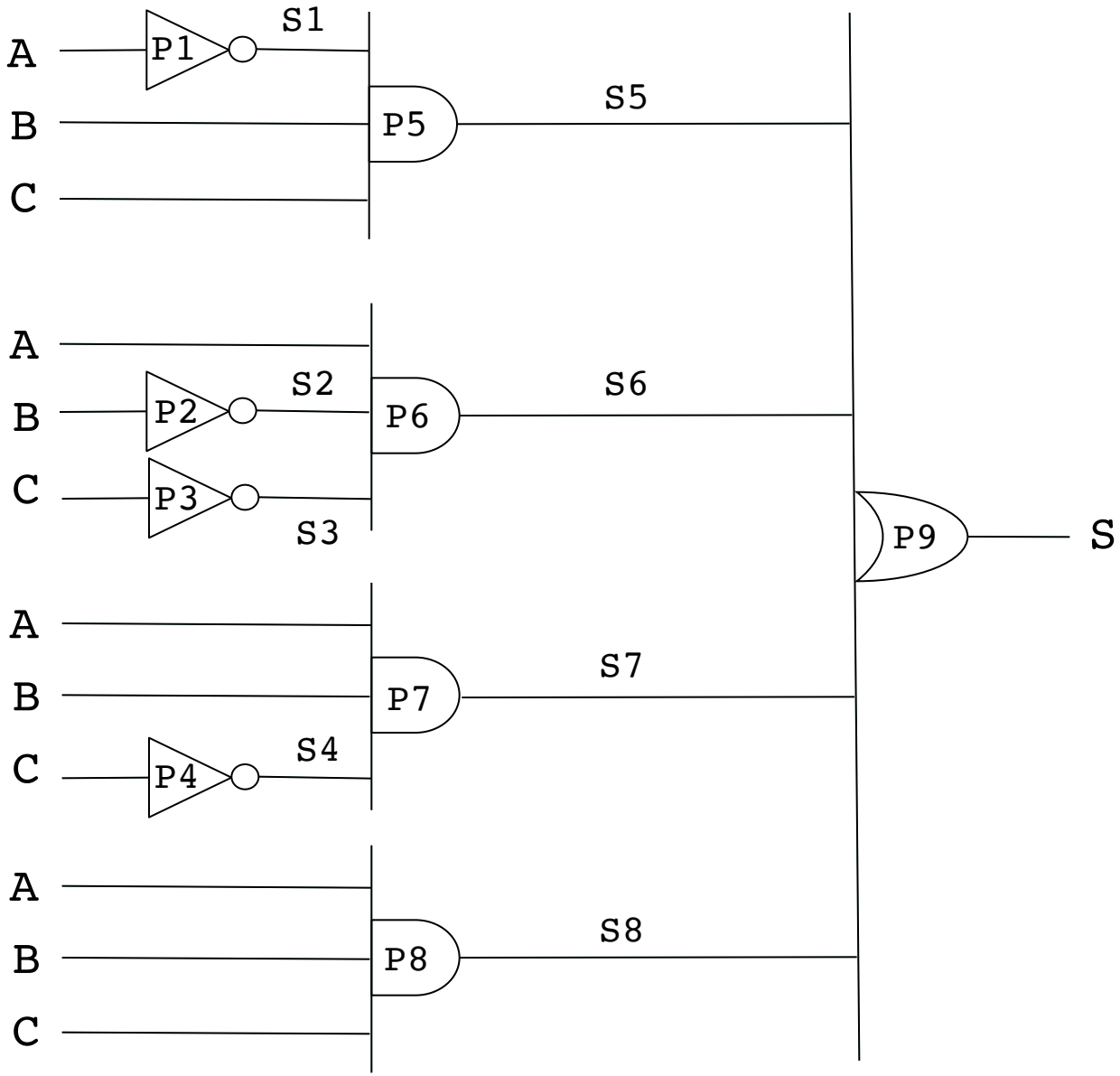
Podemos utilizar sublinha (_) para separar os termos que compõem o identificador da arquitetura, que poderia, assim, ter sido designada `fluxo_de_dados`, uma vez que espaço em branco é delimitador.

Neste caso, estamos utilizando os operadores lógicos `and`, `or` e `not`, que são oferecidos pela linguagem!

Ao comparar as duas representações (algorítmica e fluxo de dados), observamos que, enquanto na primeira não precisamos detalhar a relação entre entradas e saídas, na segunda já temos que deixar mais clara como essa relação se processa. Logo, a complexidade aumenta!

No domínio estrutural, precisamos ter clareza dos componentes internos, que compõem o componente a ser modelado, e suas interconexões.

No caso do `mux2x1`, observamos que a equação lógica traduz esses subcomponentes e suas respectivas interconexões, formando o circuito correspondente.


A
B
C
P1
S1
P5
S5
A
B
C
P2
S2
P6
S6
C
P3
S3
P9
S
A
B
C
P7
S7
C
P4
S4
A
B
C
P8
S8

As portas P1, P2, P3 e P4 implementam a operação lógica `not`.

As portas P5, P6, P7 e P8 implementam a operação lógica `and` com duas entradas.

A porta P9 implementa a operação lógica `or` com quatro entradas.

Os sinais S1, S2, S3, S4, S5, S6, S7 e S8 são sinais internos, diferentemente dos sinais de interface A, B, C e S, já definidos.

Vale observar que, no domínio estrutural, tudo é componente e não mais operador lógico.

Logo, vamos ter que criar os componentes que implementarão as respectivas operações lógicas!

A descrição abaixo corresponde à parte declarativa da arquitetura, que vem entre `architecture` e `begin`, onde declaramos os tipos de componentes (`component`) a serem utilizados e os sinais internos (`signal`), que não tem a definição `in` ou `out` uma vez que correspondem a fios e não portas de entrada e saída:

```
architecture estrutural of mux2x1 is

component inv
      port (E: in bit; S: out bit);
end component;
component and2
      port (E1, E2: in bit; S: out bit);
end component;
component or4
      port (E1, E2, E3, E4: in bit; S: out bit);
end component;
signal S1, S2, S3, S4, S5, S6, S7, S8: bit;
```

A descrição abaixo corresponde ao corpo da arquitetura estrutural, onde instanciamos os componentes e mapeamos as respectivas portas de entrada e saída:

```
begin
      P1: inv port map (A,S1);
      P2: inv port map (B,S2);
      P3: inv port map (C,S3);
      P4: inv port map (C,S4);
      P5: and2 port map (S1,B,C,S5);
      P6: and2 port map (A,S2,S3,S6);
      P7: and2 port map (A,B,S4,S7);
      P8: and2 port map (A,B,C,S8);
      P9: or4 port map (S5,S6,S7,S8,S);
end estrutural;
```

O mapeamento se faz de acordo com a posição do sinal na declaração do componente!

A descrição estrutural introduz novos componentes, que precisam ser definidos para se ter o modelo completo do componente.

Para cada componente, tem-se sempre a opção de descrição no domínio comportamental e estrutural. No entanto, quando se trata de porta lógica, a opção é somente o domínio comportamental, uma vez que os subcomponentes de uma porta lógica são transistores e resistores:

```
entity inv is
      port (E: in bit; S: out bit);
end inv ;

architecture fluxodedados of inv is
begin
      S <= not E;
end fluxodedados;
```

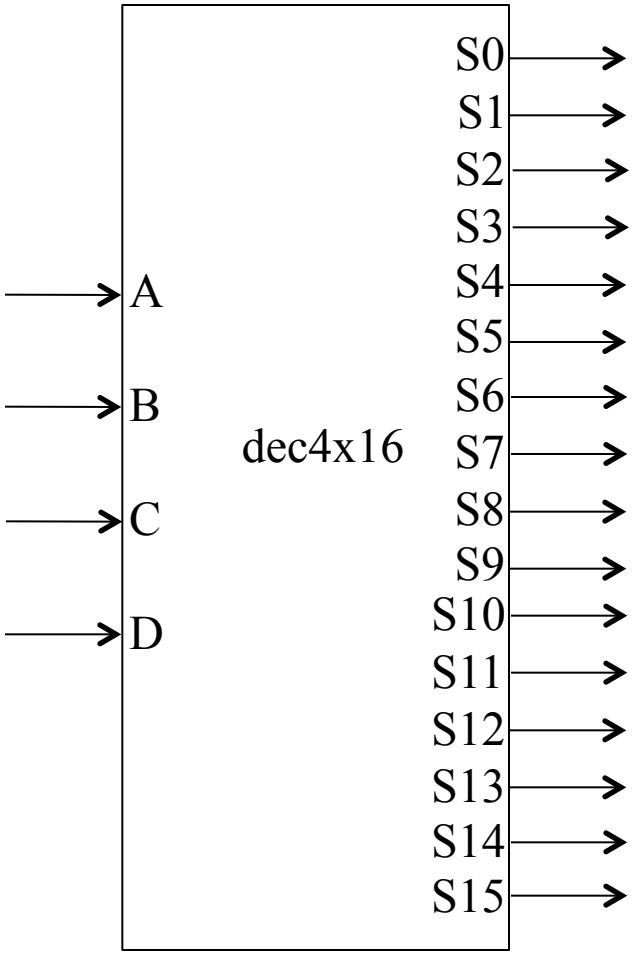
```
entity and2 is
port (E1,E2: in bit; S: out bit);
end and2;

architecture fluxodedados of and2 is
begin
      S <= E1 and E2;
end fluxodedados;

entity or4 is
port (E1,E2,E3,E4: in bit; S: out bit);
end or4;

architecture fluxodedados of or4 is
begin
      S <= E1 or E2 or E3 or E4;
end fluxodedados;
```

Os sinais de interface na entidade (`entity`) do subcomponente (`inv`, `and2`, `or4`) devem ser declarados de acordo com a declaração do subcomponente (`component`) na arquitetura do componente (`mux2x1`) e vice-versa.

Observe que apesar das arquiteturas terem o mesmo nome (`fluxodedados`), elas correspondem a componentes distintos, *i.e.* `inv`, `and2`, `or4`!

Os identificadores dados aos novos componentes não podem ser iguais a operadores lógicos prédefinidos na linguagem, como `not`, `and`, `or`!

Exercício: Apresente o modelo em VHDL do decodificador 4x16, que seleciona uma das saídas de acordo com o código binário presente nas entradas.



se ABCD = "0000", então:

S0 = '1'
S1 = '0'
S2 = '0'
S3 = '0'
S4 = '0'
S5 = '0'
S6 = '0'
S7 = '0'
S8 = '0'
S9 = '0'
S10 = '0'
S11 = '0'
S12 = '0'
S13 = '0'
S14 = '0'
S15 = '0'