Circuitos Lógicos

Profa. Grace S. Deaecto

Faculdade de Engenharia Mecânica / UNICAMP
13083-860, Campinas, SP, Brasil.
grace@fem.unicamp.br

Segundo Semestre de 2014

NOTA AO LEITOR

Estas notas de aula foram inteiramente baseadas nas seguintes referências :

- T. Floyd, "*Digital Fundamentals*", 10th Edition, Prentice Hall, 2009.
- R. J. Tocci, N. S. Widmer, G. L. Moss, "*Sistemas Digitais : Princípios e Aplicações*", Prentice-Hall, 2007.
- S. Brown, Z. Vranesic, "Fundamentals of Digital Logic with Verilog Design", McGrawHill, 2003.
- I. V. Iodeta, F. G. Capuano, "*Elementos de Eletrônica Digital*", Editora Érica, 2006.
- V. A. Pedroni, "*Circuit Design and Simulation with VHDL*", 2^{nd} Edition, MIT, 2010.

Introdução ao VHDL

- VHDL é uma linguagem de descrição de hardware que permite que um circuito eletrônico complexo seja descrito com sentenças, como uma linguagem de programação, possibilitando que ele seja simulado e sintetizado.
- Apesar de se assemelharem com linguagens de programação tradicionais, representam a descrição de sistemas em que cada parte opera de forma concorrente, ou seja, simultânea.
- A operação na forma sequencial deve ser explicitada no programa.
- VHDL é a abreviação de VHSIC - Very High Speed Integrated Circuit HDL - Hardware Description Language.

Introdução ao VHDL

- Esta descrição é importante, pois permite que o mesmo código seja usado com diversas tecnologias assegurando portabilidade e longevidade para o projeto.
- Esta linguagem foi adotada pelo IEEE (Institute of Electrical and Eletronics Engineers) sendo designada como padrão 1076-1993 do IEEE.
- Existem três maneiras diferentes de descrever um circuito digital usando VHDL : estrutural, fluxo de dados e comportamental.

Introdução ao VHDL

- **and**, **or**, **not**, **nand**, **nor**, **xor**, **xnor** são exemplos de palavras reservadas em vhdl.
- Os dois elementos essenciais para qualquer programa são **entity** e **architecture**.
- Não há diferença de interpretação entre letras maiúsculas ou minúsculas.
- A biblioteca padrão do IEEE deve ser declarada no início do programa :

```
library ieee;
use ieee.std_logic_1164.all;
library work;
use work.all;
```

- A biblioteca `work` corresponde ao conjunto de arquivos salvos pelo projeto ou programa em questão.

Introdução ao VHDL - Fluxo de dados

A **entity** descreve uma função lógica através do seu nome, do modo (in, out, inout, buffer) e tipo de dados (`bit`, `std_logic`, etc) . Por exemplo, para uma porta AND chamada "AND2" com duas entradas A, B e uma saída X, temos :

```
entity AND2 is
port: (A,B : in bit; X : out bit);
end entity AND2
```

A **architecture** descreve a operação interna da função lógica. A sua descrição em fluxo de dados é a seguinte :

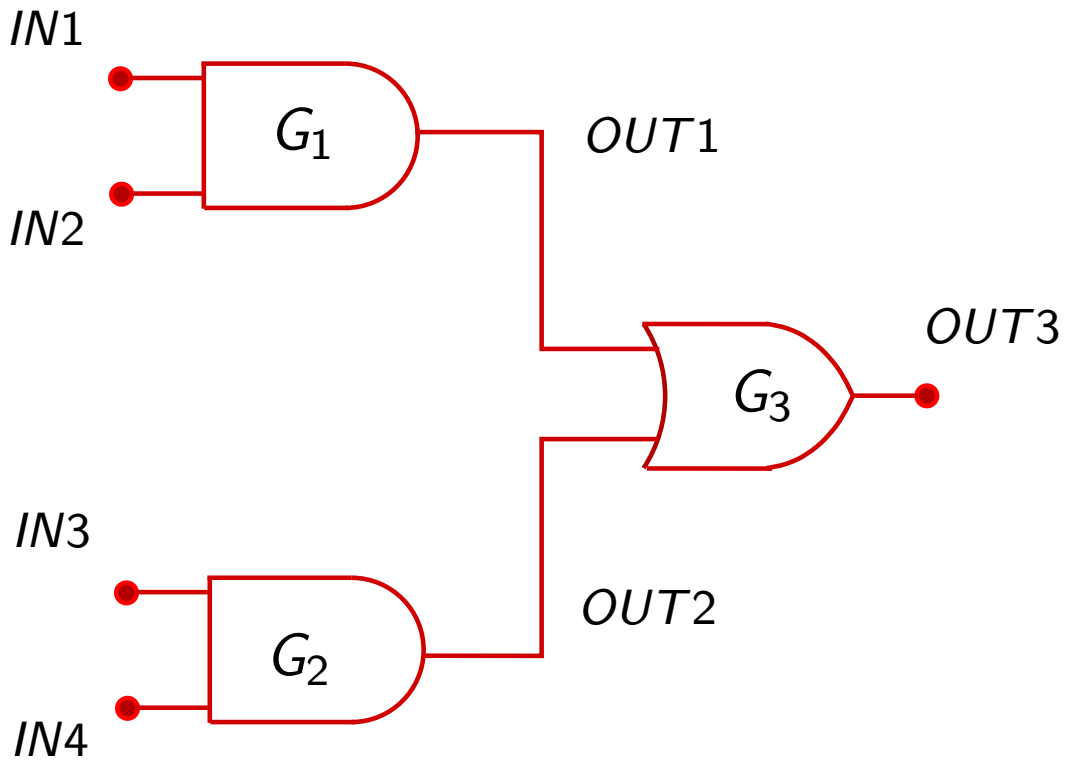
```
architecture comportamento of AND2 is
begin
X <= A and B;
end architecture comportamento
```

Introdução ao VHDL - Estrutural

- A abordagem estrutural pode ser pensada como se instalássemos um circuito em uma placa e conectássemos cada componente com fios.
- Nesta abordagem, descrevemos as funções lógicas e especificamos como elas são interconectadas.
- O **component** é uma forma de predefinir uma função lógica simples ou complexa para ser usada repetidamente em um programa ou em outros programas. Como se tivéssemos uma caixa de CIs disponível para construir um circuito.
- O **signal** pode ser entendido como uma forma de especificar uma conexão de fio entre componentes.

Introdução ao VHDL - Estrutural

- Para explicar a descrição na forma estrutural vamos considerar o seguinte circuito digital na qual denominamos AND2OR

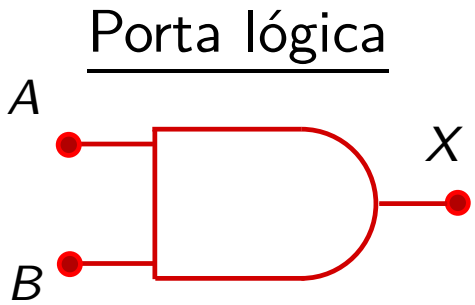


Introdução ao VHDL - Estrutural

- Por simplicidade, consideramos que existem as seguintes descrições de fluxo de dados já predefinidas.

```
entity AND2 is
port: (A,B : in bit; X : out bit);
end entity AND2

architecture funcAND2 of AND2 is
begin
X <= A and B;
end architecture funcAND2
```



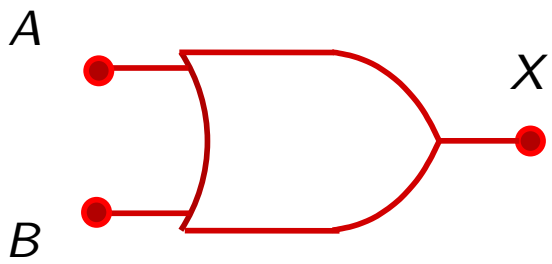
Introdução ao VHDL - Estrutural

- Para a porta OR com duas entradas, temos :

```
entity OR2 is
port: (A,B : in bit; X : out bit);
end entity OR2

architecture funcOR2 of OR2 is
begin
X <= A or B;
end architecture funcOR2
```

Porta lógica



Introdução ao VHDL - Estrutural

- Ao invés de repetir inúmeras vezes cada programa, podemos usar uma declaração de componente para especificar cada uma das portas lógicas. Esta declaração deve seguir a seguinte estrutura

```
component nome_do_componente is
port: (definicao_das_portas);
end component nome_do_componente
```

- Para usá-lo, devemos escrever uma declaração inicial para cada porta lógica.

- Em vhdl, **signal** são fios internos que interconectam componentes. No circuito AND2OR temos que OUT1 e OUT2 são declarados como signal.

Introdução ao VHDL - Estrutural

- O programa na aborgadem estrutural está apresentado a seguir. O símbolo (–) significa comentário.

```
-- Programa para o circuito AND2OR
entity AND2OR is
port: (IN1, IN2, IN3, IN4: in bit;
OUT3: out bit );
end entity AND2OR;

archictecture funcAND2OR of AND2OR is
component AND2 is
port: (A, B: in bit; X out bit);
end component AND2;

component OR2 is
port: (A, B: in bit; X out bit);
end component OR2;
```

Introdução ao VHDL - Estrutural

```
-- Continuacao do programa AND2OR
-- Declaracao de sinal

signal OUT1, OUT2: bit;
begin

-- Componentes instanciais
G1: AND2 port map(A=>IN1, B=>IN2, X=>OUT1);
G2: AND2 port map(A=>IN3, B=>IN4, X=>OUT2);
G3: OR2 port map(A=>OUT1, B=>OUT2, X=>OUT3);

end archictecture funcAND2OR;
```

Note que a simples leitura do programa nos permite montar o circuito físico, considerando todas as ligações entre as portas lógicas.

Introdução ao VHDL - Fluxo de Dados

- Apenas para fins de comparação, o mesmo circuito descrito com a abordagem de fluxo de dados é mais simples sendo dado por :

```
-- Programa para o circuito AND2OR
entity AND2OR is
port: (IN1, IN2, IN3, IN4: in bit;
OUT3: out bit );
end entity AND2OR;

archictecture funcAND2OR of AND2OR is
OUT3 <= (IN1 and IN2) or (IN3 and IN4)
end archictecture funcAND2OR;
```

Introdução ao VHDL - Comportamental

- A outra abordagem é a comportamental.
- Uma descrição puramente comportamental, não nos fornece nenhuma dica de como o circuito é construído.
- Como o próprio nome diz, esta abordagem é baseada no comportamento da resposta e não na estrutura do circuito.

```
archictecture funcAND2OR of AND2OR is
process (IN1, IN2, IN3, IN4)
begin
if (IN1 and IN2 )='1' then
OUT3<='1';
elseif (IN3 and IN4 )='1' then
OUT3<='1';
end if;
end process
end archictecture funcAND2OR;
```

Exemplos

- Vale lembrar que os programas escritos em vhdl possuem seu fluxo de execução nas formas concorrentes.
- A execução na forma sequencial deve ser imposta utilizando comandos como o **process**, **function** ou **procedure.**
- A seguir vamos apresentar alguns exemplos de programas descritos em vhdl.

Somador de 1 bit

```
library ieee;
use ieee.std_logic_1164.all;

entity somador1b is
port (x, y, cin : in std_logic;
s, cout : out std_logic);
end entity somador1b;

archictecture funcsomador1b of somador1b is
begin
s <= (x xor y) xor (cin);
cout <= ((x xor y) and cin) OR (x and y);
end architecture funcsomador1b;
```

Somador de 4 bits

```
library ieee;
use ieee.std_logic_1164.all;

entity somador4b is
port (x, y : in std_logic_vector(3 downto 0);
sm : out std_logic_vector(4 downto 0);
end entity somador4b;

archictecture funcsomador4b of somador4b is
component somador1b is
port (a, b, cin : in std_logic;
s, cout : out std_logic;
end component;
signal cin : std_logic_vector(3 downto 0);
```

Somador de 4 bits

```
begin
s0 : somador1b port map (x(0), y(0), cin(0), sm(0), cin(1));
s1 : somador1b port map (x(1), y(1), cin(1), sm(1), cin(2));
s2 : somador1b port map (x(2), y(2), cin(2), sm(2), cin(3));
s3 : somador1b port map (x(3), y(3), cin(3), sm(3), sm(4));
cout<=sm(4);
end architecture funcsomador4b;
```

Multiplexador 8 para 1

```
library ieee;
use ieee.std_logic_1164.all;
entity mux8p1 is
port (a : in std_logic_vector(7 downto 0);
sel : in std_logic_vector(2 downto 0); s : out std_logic);
end entity mux8p1;
archictecture funcmux8p1 of mux8p1 is
begin

s <= a(0) when sel = "000" else
     a(1) when sel = "001" else
     a(2) when sel = "010" else
     a(3) when sel = "011" else
     a(4) when sel = "100" else
     a(5) when sel = "101" else
     a(6) when sel = "110" else  a(7);

end architecture funcmux8p1;
```

Registrador de 4 bits

```
library ieee;
use ieee.std_logic_1164.all;

entity reg4bits is
port ( d : in std_logic_vector(3 downto 0); clk : in
std_logic;
q : out std_logic_vector(3 downto 0));
end entity reg4bits

archictecture funcreg4bits of reg4bits is
begin process(clk)
if  (clk'event AND clk = '1') then  -- ou rising_edge(clk)
q<=d
end if
end process
end architecture funcreg4bits;
```

Flip-flop JK

```
library ieee;
use ieee.std_logic_1164.all;
entity ffjk is
port ( j, k, clk, rst : in std_logic; q : inout std_logic;
end ffjk
archictecture funcffjk of ffjk is
begin process(clk, rst)
if  (rst = '1') then  q<='0';
elseif  (rising_edge(clk)) then
   if (j='1') and (k='1') then q<= not q;
   elseif (j='1') and (k='0') then q<= '1';
   elseif (j='0') and (k='1') then q<= '0';
end if;
end if;
end funcffjk;
```