VHDL

Sinais e Tipos de Dados

Rafael Corsi Ferrão - IMT

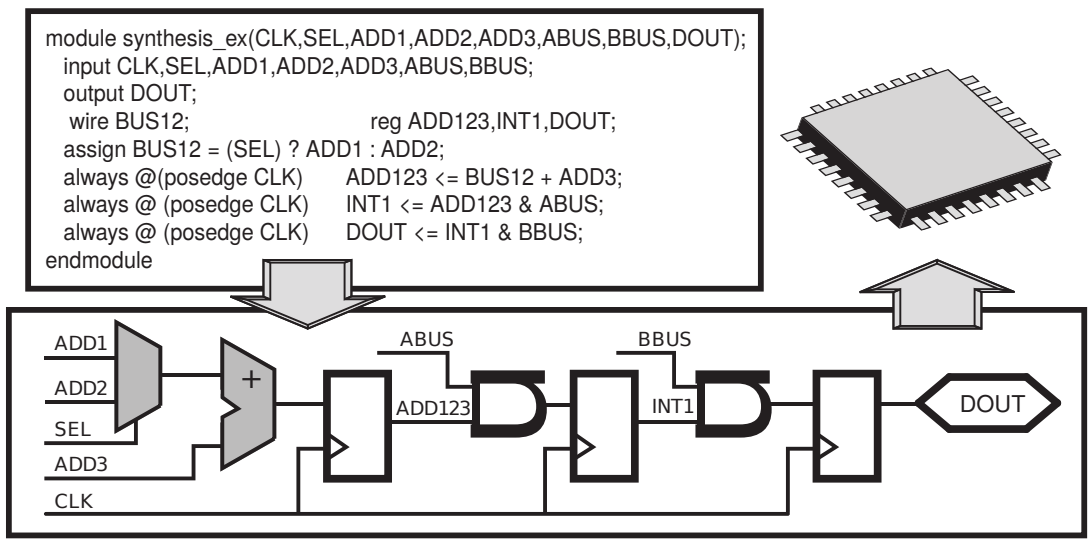
rafael.corsi@maua.br
http://www.maua.br

6 de outubro de 2014

Conteúdo

Conteúdo

- É uma linguagem para descrever HARDWARE
- tinha o objetivo de possibilitar a documentação de hardwares complexos além de possibilitar suas simulações
  - substituindo assim os famosos *schematics*
- atualmente as ferramentas permitem implementar diretamente do HDL no dispositivo
- HDL permite desenvolver senários de debug/teste



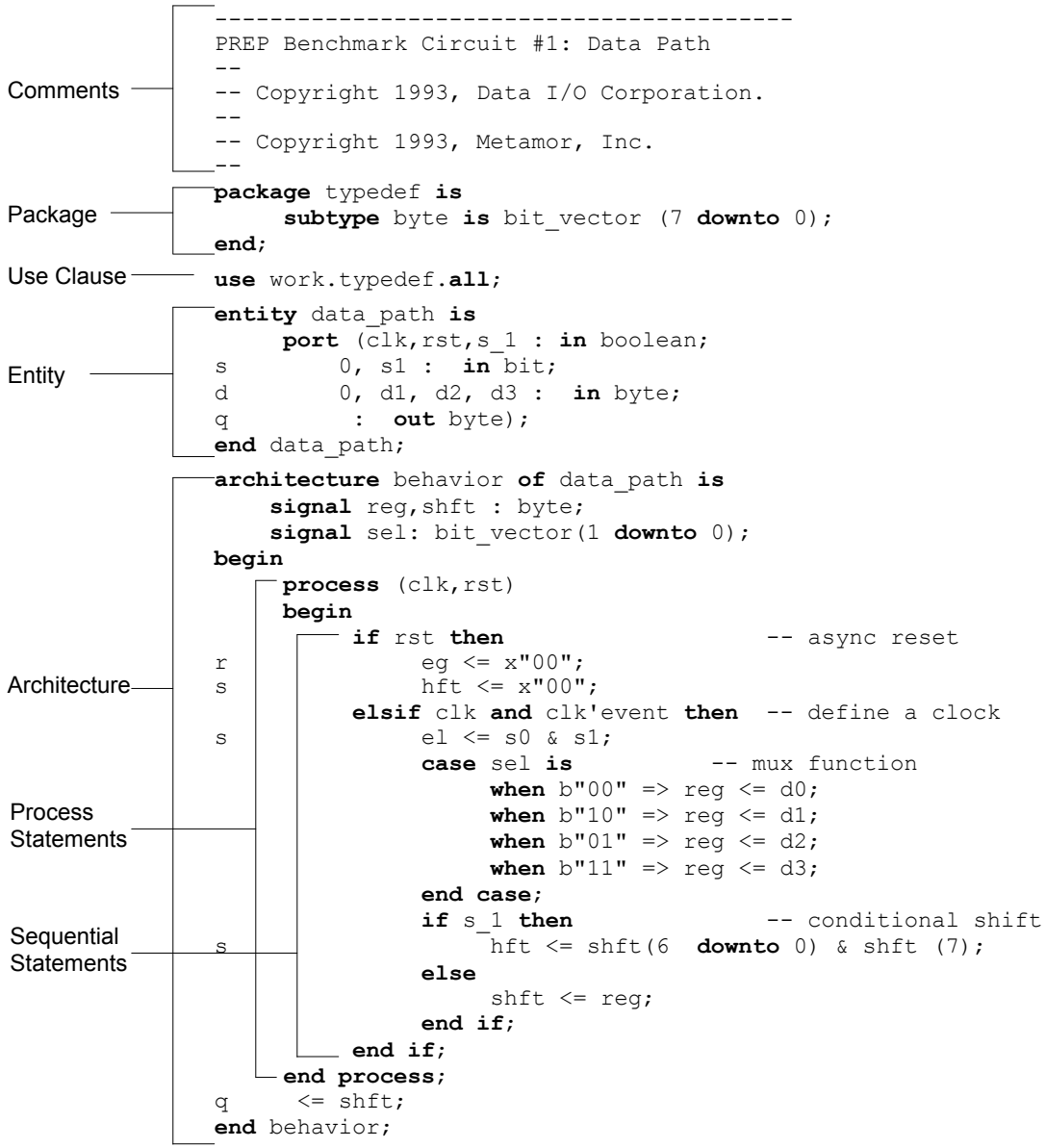
- Verilog
- VHDL

Verilog

- Desenvolvida como proprietária em 1985, aberta como um padrão em 1990
- é a segunda linguagem mais utilizada
  - bastante utilizada nas universidades americanas
- similar com C
  - o que pode confundir já que estamos habituados com programar processador
- fácil erros de implementação

VHSIC (Very High Speed Integrated Circuit) **H**ardware **D**escription **L**anguage

- O VHDL foi adotado como linguagem de HDL do exército americano durante a segunda guerra mundial
- começou a ser exigida pelos militares em novos projetos o que deu mais força a linguagem
- atualmente á a linguagem HDL mais utilizada
- strongly typed language (impões uma série de regras e garantias)




```
-----------------------------------------
PREP Benchmark Circuit #1: Data Path
--
-- Copyright 1993, Data I/O Corporation.
--
-- Copyright 1993, Metamor, Inc.
--
package typedef is
     subtype byte is bit_vector (7 downto 0);
end;
use work.typedef.all;
entity data_path is
     port (clk,rst,s_1 : in boolean;
s         0, s1 :  in bit;
d         0, d1, d2, d3 :  in byte;
q          :  out byte);
end data_path;
architecture behavior of data_path is
    signal reg,shft : byte;
    signal sel: bit_vector(1 downto 0);
begin
    process (clk,rst)
    begin
      if rst then                        -- async reset
r           eg <= x"00";
s           hft <= x"00";
      elsif clk and clk'event then  -- define a clock
s           el <= s0 & s1;
            case sel is              -- mux function
                 when b"00" => reg <= d0;
                 when b"10" => reg <= d1;
                 when b"01" => reg <= d2;
                 when b"11" => reg <= d3;
            end case;
            if s_1 then                    -- conditional shift
s                hft <= shft(6  downto 0) & shft (7);
            else
                 shft <= reg;
            end if;
      end if;
    end process;
q     <= shft;
end behavior;
```

Partes principais de um projeto

Definição

As bibliotecas são pacotes que definem os tipos de dado e suas operações.

Definição

As bibliotecas são pacotes que definem os tipos de dado e suas operações.

- Os principais tipos de dados são definidos em uma norma : **ieee 1164**
- Os dois tipos mais usuais da 1164 são :
  - std_logic
  - std_logic_vector

O tipo std_logic define nove possíveis diferentes valores, sendo eles:

- ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-')

Porém nem todos são sintetizáveis (que podem ser efetivamente implementados em uma FPGA/ASIC), os sintetizáveis são:

- ('0', '1')
- em alguns casos ('Z')

Exemplo de uso:

```
signal a : std_logic;
signal b : std_logic;
signal c : std_logic;

a <= '0';
c <= '1';
b <= c;
```

Um vetor em VHDL (std_logic_vector) é definido por uma coleção de elementos de std_logic.
Vamos considerar um exemplo de um vetor com 8 posições :

```
signal a : std_logic_vector(7 downto 0);
signal b : std_logic_vector(0 to 7);
signal c : std_logic_vector(9 downto 2);
```

Todas as declarações do exemplo possuem 8 valores e são interpretadas pelo sintetizador da mesma maneira salvo o modo de como acessar seus index.

```
a(3 downto 0) <= ('0','0','0','1'); -- Conjunto de std_logic
b(0 to 3)     <= "1000";            -- Binario
c(5 downto 2) <= X"1";              -- Hexadecimal
```

Se por algum motivo quisermos fazer com que todos os valores do vetor sejam 0, temos algumas soluções:

```
-- nada bom
a(7 downto 0) <= ('0','0','0','0','0','0','0','0');
-- melhor
a <= "0000_0000";
-- quase la
a(7 downto 0) <= x"00";
-- agora sim
a(7 downto 0) <= (others => '0');
```

Para carregarmos a biblioteca no projeto basta incluir:

```
library ieee;
use ieee_std_logic_1164.all
```

Com isso podemos usar os tipos std_logic e std_logic_vector
Algumas operações que são possíveis com esses tipos :

- **and**
- **or**
- **xor**

Definição

é o bloco mais simples de um projeto, na entidade defini-se as interfaces do projeto, declarando seus tipos e suas direções.

Definição

é o bloco mais simples de um projeto, na entidade defini-se as interfaces do projeto, declarando seus tipos e suas direções.

Cada entidade é composta por:

1. O nome da entidade;
2. os nomes das portas;
3. as direções (entrada, saída, entrada-saída);
4. e os tipos (vetor, inteiro, natural, . . .);
5. genéricos

As possíveis direções para as portas são:

- **in** : Entrada
- **out** : Saída
- **inout** : Tanto entrada quanto saída, utilizada no normalmente no acesso á memória
- **buffer** : Permite a leitura da saída

A nomenclatura das portas deve respeitar algumas regras tais como :

- não começar com números: 12bis
- deve ser diferente de palavras exclusivas: in, entity, ...

```
ENTITY exemplo1 IS
GENETIC(
    Freq : natural := 100; --MHz
    Teste: std_logic
);
PORT (
    clk   : IN std_logic;
    rdy   : IN std_logic;
    addr  : OUT   std_logic_vector(4 downto 0);
    dado  : INOUT std_logic_vector(3 downto 0)
);
END exemplo1;
```

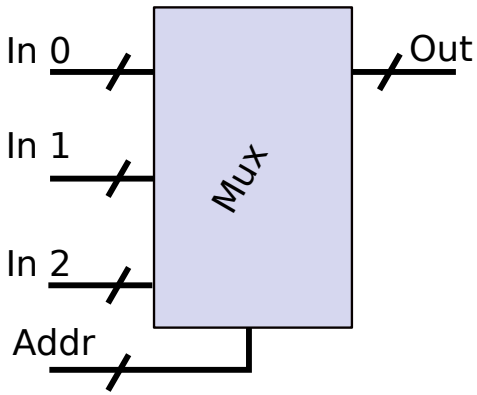
Note que uma porta que é declarada como saída não pode ser acessada como entrada:

```
ENTITY teste IS
PORT (
    in0   : IN  STD_LOGIC;
    dout  : OUT STD_LOGIC
);
END teste;
...
dout <= '1';
in0  <= dout; -- nao e permitido
```

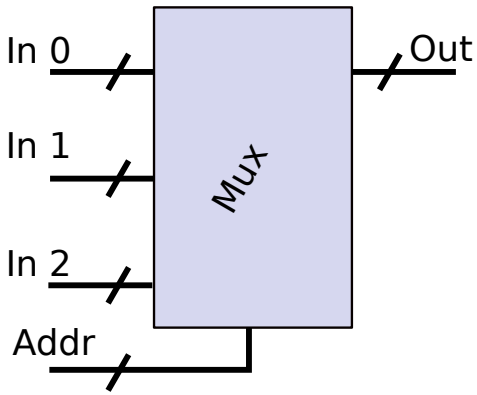
Também não podemos associar diretamente uma entrada a uma saída:

```
dout <= in0
```

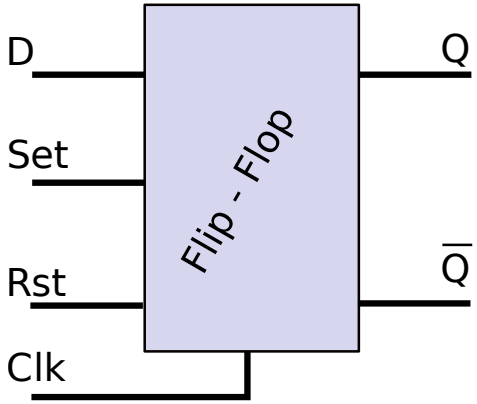
Vamos gerar a entidade de um mux:



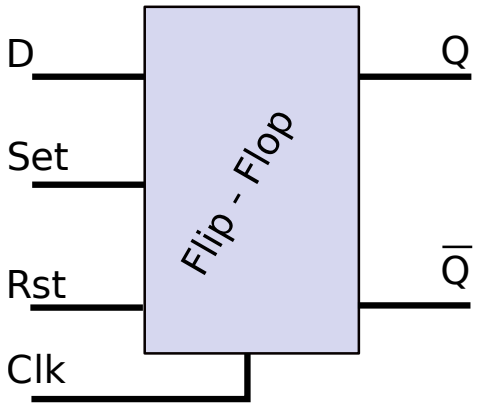
Vamos gerar a entidade de um mux:




```
ENTITY mux IS
PORT (
    in0   : IN  STD_LOGIC;
    in1   : IN  STD_LOGIC;
    in2   : IN  STD_LOGIC;
    addr  : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
    dout  : OUT STD_LOGIC
);
END mux;
```



Vamos gerar a entidade de um flip-flop



Vamos gerar a entidade de um flip-flop

```
ENTITY FlipFlop IS
PORT (
    D    : IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
    Set  : IN  STD_LOGIC;
    Rst  : IN  STD_LOGIC;
    Clk  : IN  STD_LOGIC;
    Q    : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
    Qn   : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
);
END FlipFlop;
```

Além de portas, podemos definir valores genéricos, que são similares aos #defines em c, ou seja, só é utilizado em tempo de compilação.

```
ENTITY FlipFlop IS
GENERIC(
    DataSize = 3
);
PORT (
    D    : IN  STD_LOGIC_VECTOR(DataSize-1 DOWNTO 0);
    Set  : IN  STD_LOGIC;
    Rst  : IN  STD_LOGIC;
    Clk  : IN  STD_LOGIC;
    Q    : OUT STD_LOGIC_VECTOR(DataSize-1 DOWNTO 0);
    Qn   : OUT STD_LOGIC_VECTOR(DataSize-1 DOWNTO 0)
);
END FlipFlop;
```

A utilização de genéricos é importante para a criação de entidades mais flexíveis e de fácil configuração e reutilização.

```
ENTITY BlinkLed IS
GENERIC(
    FrequenciaClk = 10 --MHZ
);
PORT (
    Clk   : IN  std_LOGIC;
    Led   : Out STD_LOGIC
);
END BlinkLed;
```

Detalhes do VHDL:

- não é "case-sensitive"
- ; indicando fim de linha
- a última definição não possui ;

Definição

A descrição da implementação interna de uma entidade é chamada de *architecture*, onde defini-se as relações entre entradas e saídas de uma determinada entidade.

Definição

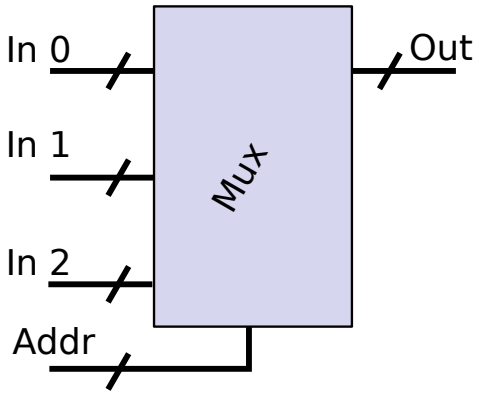
A descrição da implementação interna de uma entidade é chamada de *architecture*, onde defini-se as relações entre entradas e saídas de uma determinada entidade.

Cada arquitetura possui:

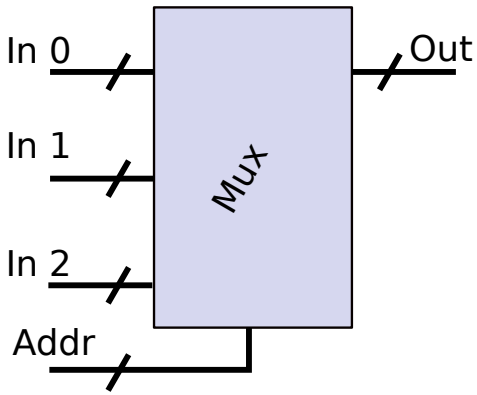
1. O nome da arquitetura;
2. Os registradores internos;
3. As ligações entre as entidades;
4. O comportamento da entidade;

MAUÁ

Vamos gerar agora a arquitetura do mux:



Vamos gerar agora a arquitetura do mux:




```
ARCHITECTURE bhv OF mux IS
begin

 WITH addr SELECT
     dout <= in0 WHEN x"00",
             in1 WHEN x"01",
             in2 WHEN OTHERS;

end bhv;
```

VHDL MUX


```
LIBRARY ieee;
use ieee_std_logic_1164.all;

ENTITY mux IS
PORT (
    in0   : IN  STD_LOGIC;
    in1   : IN  STD_LOGIC;
    in2   : IN  STD_LOGIC;
    addr  : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
    dout  : OUT STD_LOGIC
);
END mux;

ARCHITECTURE bhv OF mux IS
begin

 WITH addr SELECT
     dout  <= in0 WHEN X"00",
              in1 WHEN X"01",
              in2 WHEN OTHERS;

end bhv;
```