

VHDL

Componentes & IP CORES

4 de outubro de 2015

Rafael Corsi Ferrão - IMT

rafael.corsi@maua.br

<http://www.maua.br>



1. Hierarquia
2. Componente
3. IP CORES

1. Hierarquia

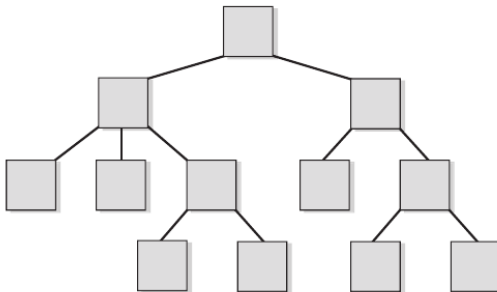
2. Componente

3. IP CORES

ESTILO DE PROJETO: HIERARQUICO VS PLANO



Flat File Relationship



Hierarchical File Relationship

Definição

Uma única entidade define todo o projeto.

- ▶ o projeto consiste num único arquivo
- ▶ difícil manutenção e desenvolvimento
- ▶ a reutilização de códigos fica comprometida
- ▶ a ferramenta de síntese pode ter dificuldade para otimizar
- ▶ grandes projetos vão necessitar maiores tempos de processamento

Definição

A descrição é composta por diversos blocos, sendo um definido como o de maior prioridade.

- ▶ o projeto consiste em diversos arquivos
- ▶ fácil manutenção e desenvolvimento
- ▶ favorece a reutilização de códigos
- ▶ a ferramenta de síntese possui mais facilidade para otimizar pequenos blocos
- ▶ o tempo de processamento é reduzido consideravelmente

1. Hierarquia

2. Componente

3. IP CORES

Definição

Um componente é uma entidade empregada na arquitetura de outra entidade.

- ▶ A utilização de componentes é semelhante a conexão de CIs em um projeto de esquemático
- ▶ um projeto pode ter n componentes
- ▶ um componente pode ser formado por outros componentes

Para a utilização de um componente é preciso primeiramente realizarmos sua declaração, para então mapearmos as portas. A declaração das portas e genéricos do componente é a mesma da entidade.

```
COMPONENT nome_componente
  GENERIC(
    n : tipo := valor
  );
  PORT(
    sinal_a : tipo ;
    sinal_b : tipo
  );
end COMPONENT;
```

A declaração do componente pode ser feita no corpo da arquitetura (antes do begin) ou em pacotes.

Para usarmos um componente, precisamos agora mapear seus generics, entradas e saídas para portas ou sinais da entidade que estamos trabalhando.

Esse mapeamento acontece dentro da região de descrição (após o begin),

```
rotulo : nome_componente
  GENERIC map(
    n => Valor
  );
  PORT MAP(
    sinal_a => a ;
    sinal_b => b
  );
```

Note que usamos o sinal \Rightarrow no mapeamento, isso significa que a porta `sinal_a` é mapeada para o sinal `a`.

Existe outra método de mapeamento que não demonstra explicitamente as portas que estão sendo mapeadas.

```
rotulo : nome_componente  
  GENERIC map(Valor);  
  PORT MAP( a, b);
```

Essa forma aparentemente mais simples possui certas dificuldades no uso:

- ▶ a ordem do mapeamento é importante,
- ▶ não fica claro na hora do mapeamento quais portas estamos mapeando

- ▶ todas as portas de um componente devem ser mapeadas
- ▶ caso uma porta de saída não seja usada, deve-se usar a palavra reservada **OPEN** em seu mapeamento
- ▶ para cada porta de um componente devemos associar um sinal, uma porta de saída ou uma porta de entrada
- ▶ genéricos não precisam necessariamente serem mapeados desde que possuam um valor inicial

- ▶ sabemos que ao lidarmos com botões existe o ressalto (bounce) até a estabilização do nível de sinal.
- ▶ precisamos criar uma entidade capaz de realizar o debounce para evitarmos o erro na leitura desses sinais.
- ▶ o debounce pode aparecer diversas vezes no projeto e também vai ser comum o seu uso em outros projetos. Vamos projetar um debounce que pode ser reutilizado.

Passos :

1. Criar uma entidade mais genérica possível para realizar o debounce
2. usar essa entidade como um componente de outra entidade

```
entity debounce is
generic(
    fclk : natural := 100_000_000; -- frequencia do clk (hz)
    tdb  : natural := 10    -- debounce time (ms)
);
port(
    clk      : in STD_LOGIC; -- clk
    btn_in   : in STD_LOGIC; -- sinal para ser tratado
    btn_out  : out STD_LOGIC -- sinal tratado
);
end debounce;
```

```
ARCHITECTURE rtl OF debounce IS
```

```
BEGIN
```

```
    -- somente para ilustrar,
    -- o sinal aqui nao e tratado
    btn_out <= btn_in;
```

```
END rtl;
```

```
entity top is
generic(
    fclk : natural := 100_000_000 -- frequencia do clk (Mhz)
);
port(
    clk    : in  STD_LOGIC;
    sw     : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
    led    : out STD_LOGIC_VECTOR(15 DOWNTO 0)
);

end top;

ARCHITECTURE rtl OF top IS

    -- Declaramos aqui a utilizacao do componente
    COMPONENT debounce is
        generic(
            fclk : natural := 100_000_000; -- frequencia do clk (Hz)
            tdb  : natural := 10    -- debounce time (ms)
        );
        port(
            clk      : in  STD_LOGIC;
            btn_in   : in  STD_LOGIC;
            btn_out  : out STD_LOGIC
        );
    end COMPONENT;

    -- sinal para mapearmos o sinal tratado
    signal btn : std_logic;
```

```
-- inicio
BEGIN

    -- Aqui mapeamos o componente
    u1 : debounce
        GENERIC MAP(
            fclk => fclk,
            tdb  => 10
        )
        PORT MAP(
            clk      => clk,
            btn_in   => sw(0),
            btn_out  => btn
        );

    led(15 downto 1) <= (others => '0');

    led(0) <= btn;

END rtl;
```


Poderíamos omitir o genérico :

```
u1 : debounce
    PORT MAP(
        clk      => clk,
        btn_in   => sw(0),
        btn_out  => btn
    );
```

Ou modificarmos somente uma das configurações :

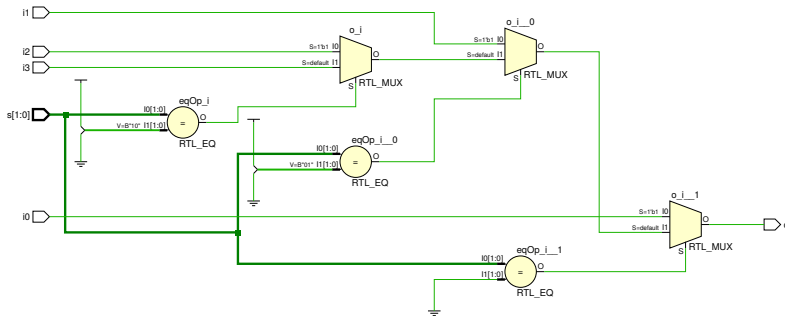
```
u1 : debounce
    GENERIC MAP(
        tdb      => 100
    )
    PORT MAP(
        clk      => clk,
        btn_in   => sw(0),
        btn_out  => btn
    );
```

O mapeamento omitindo os nomes das portas ficaria :

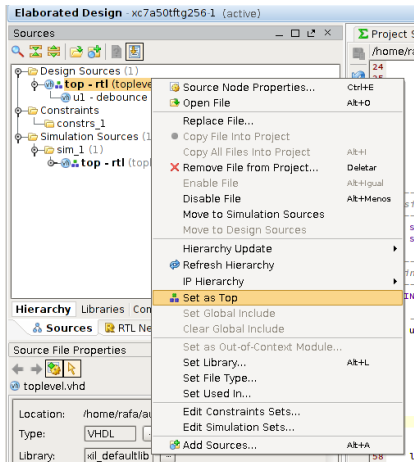
```
u1 : debounce  
    PORT MAP(clk, sw(0), btn);
```

Se não fossemos utilizar a porta de saída, tempo que dizer que ela não está conectada:

```
u1 : debounce  
    PORT MAP(clk, sw(0), OPEN);
```



Precisamos indicar no SW Vivado qual entidade é a de mais alta hierarquia, essa entidade é chamada normalmente de **Top Level**:



1. Hierarquia
2. Componente
3. IP CORES

Definição

IP CORE (Intellectual Property Core) é um termo utilizado para descrever um bloco lógico que pode ser reutilizável em outros projetos.

IP Cores servem para facilitar o desenvolvimento de um projeto, evitando que o desenvolvedor tenha que projetar todas as partes de um projeto. Exemplos de IP CORE:

- ▶ Controlador USB
- ▶ Controlador de memória
- ▶ VGA/HDMI
- ▶ Soft microprocessador
- ▶ ...

Platforms and Solutions

New IP Cores

- UltraScale Integrated 100G Ethernet MAC/PCS
- UltraScale Interlaken
- AXI PCI Express (PCIe) Gen 3
- IBERT for UltraScale GTY Transceivers

Most Requested IP Products

- Memory Interfacing
- Ethernet
- DSP
- Aurora Technology
- Forward Error Correction Solutions
- MicroBlaze™
- EDK Supported IP
- Architecture Wizards
- Video IP
- PCI-Express®
- PCI Express DMA Back-End Core (NWL)

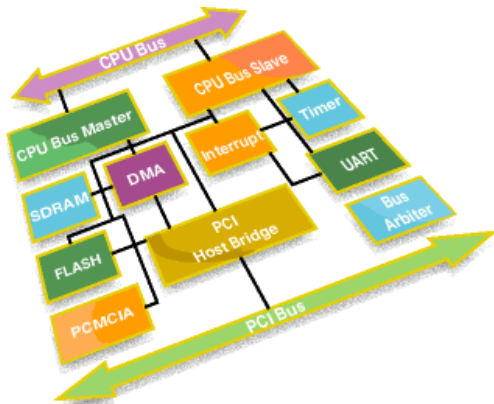
IP podem ser pagos ou gratuitos, em seguida alguns sites que podem ajudar na escolha de um core:

- ▶ <http://www.xilinx.com/products/intellectual-property/> :
IP cores específicos Xilinx
- ▶ <http://opencores.org/> : site que agrega IP cores
opensource

IP podem ser pagos ou gratuitos, em seguida alguns sites que podem ajudar na escolha de um core:

- ▶ <http://www.xilinx.com/products/intellectual-property/> : IP cores específicos Xilinx
- ▶ <http://opencores.org/> : site que agrega IP cores opensource

O IP core é geralmente fornecido com um datasheet (similar a um CI) e o seu uso se dá mapeando-o no projeto como um componente. Um IP CORE pode ser formado por um conjunto de entidades.



Interface gráfica de configuração

