Introdução ao VHDL

Ricardo Kerchbaumer

1 O que é VHDL?

A sigla VHDL representa "*VHSIC Hardware Description Language*", que pode ser traduzido como "Linguagem de Descrição de Hardware para Circuitos Integrados de Muito Alta Velocidade". A VHDL é uma linguagem de descrição de hardware que tem suas raízes no desenvolvimento de circuitos integrados, mas também é amplamente usada em dispositivos lógico reconfiguráveis. Vamos explorar os elementos-chave que compõem o significado da VHDL:

- **Linguagem de Descrição de Hardware:** A VHDL é uma linguagem de programação de alto nível usada para descrever o comportamento e a estrutura de sistemas digitais. Ao contrário de linguagens de programação tradicionais, como C++ ou Java, que descrevem a lógica de software, a VHDL é usada para representar sistemas físicos e digitais.

- **Circuitos Integrados de Muito Alta Velocidade (VHSIC):** A VHDL foi originalmente desenvolvida pelo Departamento de Defesa dos Estados Unidos para atender às necessidades de projetar circuitos integrados de alto desempenho, conhecidos como VHSIC. No entanto, sua utilidade se expandiu além da indústria de circuitos integrados, tornando-se uma ferramenta valiosa em diversas áreas.

- **Descrição de Hardware:** A VHDL permite aos engenheiros descrever o comportamento e a estrutura de sistemas digitais, desde portas lógicas simples até sistemas complexos e sistemas embarcados. Isso é feito por meio de uma série de abstrações e construções de alto nível que permitem a **modelagem e simulação** de sistemas digitais de forma eficiente.

2 Ambiente de Desenvolvimento

2.1 Ferramentas de Design VHDL

Ao entrar no mundo da linguagem VHDL, é importante ter conhecimento sobre as ferramentas de design disponíveis para ajudar na criação, simulação, síntese e implementação de sistemas digitais. Neste capítulo, abordaremos as principais ferramentas de design VHDL que são amplamente utilizadas na indústria e no ambiente acadêmico.

2.1.1 Ferramentas de Design VHDL Comuns

Existem várias ferramentas de design VHDL disponíveis, cada uma com suas características e vantagens. Abaixo estão algumas das ferramentas mais comuns:

- **Xilinx Vivado:** O Vivado é uma suíte de ferramentas de design VHDL desenvolvida pela Xilinx. É amplamente utilizado para projetar sistemas digitais em dispositivos da Xilinx, como FPGAs e SoCs (System on Chip). Ele oferece um ambiente completo que inclui síntese, simulação, implementação e depuração.

- **Altera Quartus:** Quartus é a suíte de ferramentas VHDL da Altera (agora parte da Intel). Ele é usado para projetar sistemas digitais em dispositivos FPGA da Altera. A suíte oferece recursos semelhantes ao Vivado, incluindo síntese, simulação e implementação.

- **ModelSim**: ModelSim é uma ferramenta de simulação VHDL amplamente usada para verificar o funcionamento correto de projetos VHDL. Ela oferece uma plataforma de simulação robusta para depuração e teste de código VHDL.

- **Questa**: Questa também é uma ferramenta de simulação semelhante ao ModelSim, desenvolvida pela Siemens.

- **Synopsys Design Compiler:** Design Compiler é uma ferramenta de síntese de alto desempenho usada para traduzir código VHDL em descrições de hardware para implementação em circuitos integrados.

2.1.2 Configurando Seu Ambiente de Desenvolvimento

Antes de começar a escrever e projetar em VHDL, é importante configurar um ambiente de desenvolvimento adequado. Isso inclui a instalação da ferramenta de design VHDL de sua escolha, configurar as bibliotecas e definir opções de compilação. É importante salientar que a ferramenta escolhida deve ser compatível com o hardware utilizado.

3 Estrutura Básica de um Programa VHDL

Um programa VHDL é a base para a descrição de hardware digital. A estrutura básica de um programa VHDL inclui várias partes essenciais, como bibliotecas, entidades e arquiteturas. Vamos analisar cada uma delas em detalhes.

3.1 Bibliotecas

Bibliotecas são coleções de módulos e componentes reutilizáveis que podem ser importados em seu projeto VHDL. As bibliotecas fornecem acesso a uma variedade de funcionalidades prontas, como portas lógicas, registros e contadores. Para usar bibliotecas em seu programa VHDL, você deve declará-las no início do código usando a cláusula “library”. Além disso, você pode usar a cláusula `use` para especificar quais módulos da biblioteca deseja incluir em seu projeto.

Exemplo de declaração de biblioteca:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL; -- Biblioteca para tipos de dados lógicos
use IEEE.STD_LOGIC_ARITH.ALL; -- Biblioteca para operações aritméticas
```

3.2 Entidades

Entidades são descrições de interfaces de componentes que você deseja criar ou usar em seu projeto. Elas definem as portas de entrada e saída, bem como os tipos de dados que serão usados para interagir com o componente. Você pode pensar em entidades como "moldes" para seus componentes, descrevendo como eles se conectam ao resto do sistema.

Exemplo de declaração de entidade:

```
entity MeuComponente is
   Port (
      Entrada1 : in  STD_LOGIC;   -- Porta de entrada
      Saida1   : out STD_LOGIC    -- Porta de saída
   );
end entity MeuComponente;
```

3.3 Arquiteturas

As arquiteturas são implementações específicas de entidades. Cada entidade pode ter várias arquiteturas diferentes que descrevem seu comportamento interno. Uma arquitetura é onde você escreve o código real que define como o componente funciona. Você pode usar uma ou mais arquiteturas para a mesma entidade, dependendo de diferentes implementações.

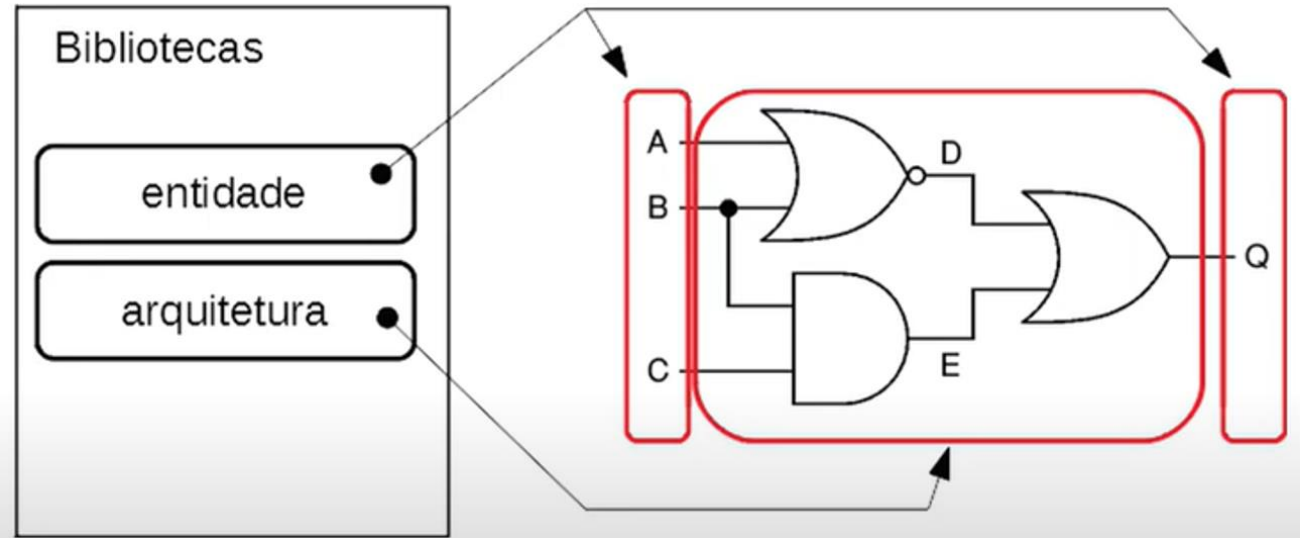
Exemplo de declaração de arquitetura:

```
architecture Comportamento1 of MeuComponente is begin
   -- Implementação do comportamento aqui
end architecture Comportamento1;
```

3.4 Estrutura do Bloco de Código VHDL

A estrutura geral de um bloco de código VHDL segue a seguinte ordem:

- Declaração de bibliotecas (library) e inclusão de módulos (use).
- Declaração da entidade (entity) com portas de entrada e saída.
- Declaração da arquitetura (architecture) para a entidade.
- Implementação do código dentro da arquitetura.
- Fim do código VHDL.



Essa estrutura básica permite que você descreva e implemente efetivamente componentes e sistemas digitais em VHDL. À medida que você avança no aprendizado da linguagem, aprimora suas habilidades de design e começa a criar sistemas mais complexos, essa estrutura básica se mantém, mas você preenche as arquiteturas com lógica específica para atender aos requisitos do seu projeto.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity MeuComponente is
   Port (
      Entrada1 : in  STD_LOGIC;
      Saida1   : out STD_LOGIC
   );
end entity MeuComponente;

architecture Comportamento1 of MeuComponente is
begin
   -- Implementação do comportamento aqui
end architecture Comportamento1;
```

Vamos criar um exemplo simples de um componente que realiza a operação lógica AND em duas entradas. Primeiro, a declaração da biblioteca, seguida pela entidade e, por fim, a arquitetura:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PortaAND2 is
   Port (
      Entrada1 : in  STD_LOGIC;
      Entrada2 : in  STD_LOGIC;
      Saida    : out STD_LOGIC
   );
end entity PortaAND2;

architecture Comportamento of PortaAND2 is
begin
   -- Implementação da operação lógica AND
   Saida <= Entrada1 AND Entrada2;
end architecture Comportamento;
```

Neste exemplo:

- A biblioteca IEEE é declarada e a biblioteca STD_LOGIC_1164 é incluída para permitir o uso de tipos de dados lógicos.
- A entidade “PortaAND2” é definida com duas entradas (Entrada1 e Entrada2) e uma saída (Saida).
- A arquitetura “Comportamento” é declarada, e a implementação da operação lógica AND é fornecida. A saída “Saida” é definida como o resultado da operação lógica AND entre “Entrada1” e “Entrada2”.

Agora, vamos criar um exemplo de um componente que multiplica dois números inteiros sem sinal de 4 bits usando operações aritméticas em VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Multiplicador4Bits is
   Port (
      Entrada1 : in  STD_LOGIC_VECTOR(3 downto 0);
      Entrada2 : in  STD_LOGIC_VECTOR(3 downto 0);
      Saida    : out STD_LOGIC_VECTOR(7 downto 0)
   );
end entity Multiplicador4Bits;

architecture Comportamento of Multiplicador4Bits is
begin
   -- Implementação da multiplicação
   Saida <= std_logic_vector(unsigned(Entrada1) * unsigned(Entrada2));
end architecture Comportamento;
```

Neste exemplo

- Além da biblioteca IEEE e STD_LOGIC_1164, a biblioteca IEEE.NUMERIC_STD é incluída para permitir operações aritméticas.
- A entidade "Multiplicador4Bits" é definida com duas entradas de vetores de 4 bits (Entrada1 e Entrada2) e uma saída de vetor de 8 bits (Saida).
- A arquitetura "Comportamento" é declarada, e a implementação da multiplicação é fornecida usando operações aritméticas. Os valores de entrada são convertidos para números inteiros sem sinal, multiplicados, convertidos novamente para std_logic_vector e atribuídos à saída "Saida".

Esses exemplos demonstram como criar entidades e arquiteturas em VHDL para componentes simples que realizam operações lógicas e aritméticas. A estrutura básica é mantida, mas o comportamento interno varia de acordo com o objetivo do componente.

4 Tipos de Dados em VHDL

Os tipos de dados em VHDL desempenham um papel fundamental na descrição de hardware digital e na manipulação de informações em sistemas digitais. Neste capítulo, exploraremos os tipos de dados mais comuns em VHDL, incluindo tipos escalares, tipos compostos e como criar tipos definidos pelo usuário.

4.1 Tipos de Dados Escalares

Os tipos de dados escalares em VHDL representam informações individuais, como bits ou números inteiros. Aqui estão alguns tipos de dados escalares comuns:

- **BIT**: O tipo BIT representa um valor lógico, sendo 0 para lógica baixa (0) e 1 para lógica alta (1).
- **INTEGER**: O tipo INTEGER é usado para representar números inteiros, geralmente dentro de um intervalo definido.

- **REAL**: O tipo REAL é usado para representar números de ponto flutuante de precisão simples (Normalmente não é utilizado para síntese).

Exemplos:

```
signal sinal_bit : BIT := '1'; -- Inicializando um sinal BIT com valor '1'.
signal contador : INTEGER := 0; -- Inicializando um contador com valor zero.
signal numero_real : REAL := 3.14; -- Inicializando um número real com 3.14.
```

4.2 Tipos de Dados Compostos

Os tipos de dados compostos em VHDL permitem agrupar vários valores escalares em uma única variável. Os tipos de dados compostos mais comuns são:

STD_LOGIC: O tipo STD_LOGIC é usado para representar valores lógicos, como BIT, mas com uma gama mais ampla de estados, como:

- 'U' (indefinido),
- 'X' (desconhecido),
- '0' (lógica baixa),
- '1' (lógica alta),
- 'Z' (alta impedância).

O tipo STD_LOGIC é amplamente utilizado como padrão para a interface de componentes.

STD_LOGIC_VECTOR: O tipo STD_LOGIC_VECTOR é usado para criar vetores de valores lógicos, como por exemplo uma palavra de 8 bits.

INTEGER_VECTOR: O tipo INTEGER_VECTOR é semelhante ao STD_LOGIC_VECTOR, mas para números inteiros.

Exemplos:

```
signal vetor_logico : STD_LOGIC_VECTOR(7 downto 0); -- Vetor de 8 bits.
signal contador : INTEGER_VECTOR(0 to 9); -- Vetor de 10 números inteiros.
```

4.3 Tipos de Dados Definidos pelo Usuário

Você também pode criar tipos de dados definidos pelo usuário em VHDL para representar informações específicas do seu projeto. Isso é feito por meio do uso da palavra-chave `TYPE`. Por exemplo, se você estiver projetando um sistema que lida com cores, pode criar um tipo de dados personalizado para representar cores.

Exemplo:

```
-- Definindo um tipo de dados para representar cores RGB.
TYPE Cor_RGB IS RECORD
  Vermelho  : INTEGER range 0 to 255;
  Verde     : INTEGER range 0 to 255;
  Azul      : INTEGER range 0 to 255;
END RECORD;
```

```
-- Declarando uma variável do tipo Cor_RGB.
signal cor_de_fundo : Cor_RGB;
```

Neste exemplo, criamos um tipo de dados personalizado chamado “Cor_RGB” que contém três campos inteiros para representar as componentes de cor vermelha, verde e azul. Você pode, então, declarar variáveis desse tipo e usá-las em seu código para representar cores.

4.4 Conversões de Tipo

Em VHDL, é possível converter entre diferentes tipos de dados. Por exemplo, você pode converter um valor `INTEGER` em um valor `STD_LOGIC_VECTOR` para representar números em formato binário.

Exemplo:

```
signal valor_inteiro : INTEGER := 7;
signal valor_binario : STD_LOGIC_VECTOR(2 downto 0);

-- Convertendo um valor inteiro em um vetor lógico.
valor_binario <= std_logic_vector(to_unsigned(valor_inteiro, valor_binario'length));
```

Neste exemplo, convertemos o valor inteiro 7 em um vetor lógico de 3 bits.

Entender e usar os tipos de dados corretos é crucial ao programar em VHDL, pois eles determinam como as informações são representadas e manipuladas em sistemas digitais. Isso é essencial para criar descrições de hardware precisas e eficazes.

5 Processos em VHDL

Processos em VHDL desempenham um papel fundamental na definição do comportamento de circuitos digitais. Eles permitem que você descreva como os sinais evoluem ao longo do tempo, criando a lógica e a sequência de operações em seu design.

Neste capítulo, exploraremos em detalhes o conceito de processos em VHDL, como criá-los e exemplos de sua aplicação.

5.1 O Que é um Processo em VHDL?

Em VHDL, um processo é um bloco de código que descreve a evolução de sinais em um circuito digital. Um processo é uma parte essencial de qualquer descrição de hardware em VHDL e é usado para modelar a lógica combinacional e sequencial em sistemas digitais.

5.1.1 Criando um Processo em VHDL

Para criar um processo em VHDL, você usa a palavra-chave “PROCESS” seguida de um conjunto de declarações entre as palavras-chave “BEGIN” e “END PROCESS”.

Um processo pode ser sensível a mudanças em sinais específicos, o que significa que ele será executado quando esses sinais mudarem de valor. Isso é controlado pela lista de sensibilidade.

Exemplo de criação de um processo sensível a mudanças em sinais de entrada:

```
process (EntradaA, EntradaB)
begin
   -- Lógica do processo aqui
   Saida <= EntradaA AND EntradaB;
end process;
```

Neste exemplo, o processo é sensível a mudanças nos sinais EntradaA e EntradaB. Sempre que um desses sinais muda de valor, o processo é executado, atualizando a saída Saida com o resultado da operação AND.

5.1.2 Sensibilidade e Temporização

Em VHDL, a sensibilidade é uma propriedade que determina quando um processo é acionado. A sensibilidade é especificada na cláusula SENSITIVITY LIST ao criar um processo. Além da sensibilidade, os processos também podem ser temporizados usando a cláusula WAIT para controlar atrasos na execução. (NÃO SITETIZAVEL)

Exemplo de processo:

```
process (Clk, Reset)
begin
   if Reset = '1' then
      -- Reinicialização do circuito
      Saida <= '0';
   elsif rising_edge(Clk) then
      -- Lógica baseada na borda de subida do clock
      Saida <= EntradaA AND EntradaB;
   end if;
end process;
```

Neste exemplo, o processo é sensível aos sinais “Clk” e “Reset”. A cláusula “rising_edge(Clk)” garante que o processo só seja executado na borda de subida do sinal de clock, criando uma temporização precisa.

5.1.3 Exemplos de Processos em VHDL

Aqui estão alguns exemplos adicionais de como os processos podem ser usados em VHDL:

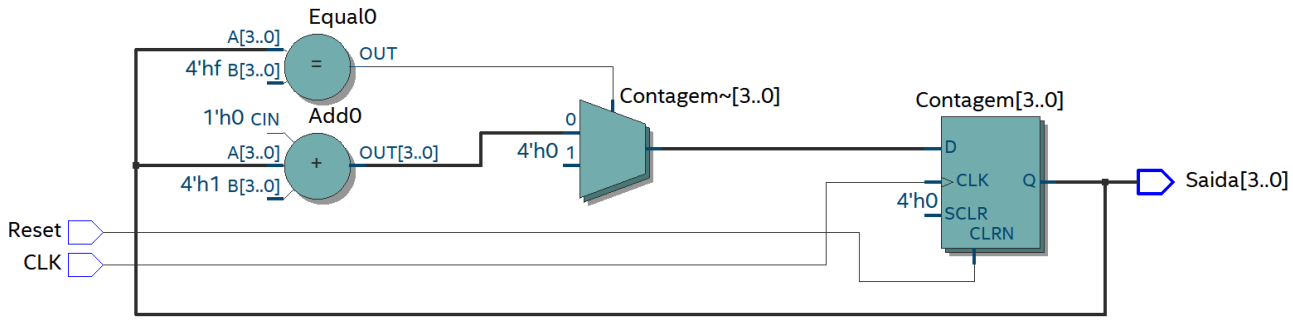
- Multiplexador de 2 para 1:

```
process (Selecao, DadoA, DadoB)
begin
   if Selecao = '0' then
      Saida <= DadoA;
   else
      Saida <= DadoB;
   end if;
end process;
```

- Contador de 4 bits:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;

entity main is
   Port ( Reset : in  STD_LOGIC;
          CLK : in  STD_LOGIC;
          Saida : out STD_LOGIC_VECTOR(3 downto 0)
 );
end main;

architecture Behavioral of main is
begin
  process (Clk, Reset)
   variable Contagem : INTEGER range 0 to 15 := 0;
begin
   if Reset = '1' then
      Contagem := 0;
   elsif rising_edge(Clk) then
      if Contagem = 15 then
         Contagem := 0;
      else
         Contagem := Contagem + 1;
      end if;
   end if;
   Saida <= std_logic_vector(to_unsigned(Contagem, Saida'length));
end process;

end Behavioral;
```

A figura a seguir mostra como o sintetizador interpretou o código VHDL deste exemplo:



Esses exemplos ilustram como os processos são usados para descrever o comportamento de sistemas digitais em VHDL. Eles permitem que você modele a lógica e a sequência de operações de maneira precisa e eficaz.

6 Atribuições em VHDL

A atribuição é uma operação fundamental em VHDL, que permite a você definir o valor de um sinal ou variável. Em VHDL, existem dois tipos principais de atribuição: atribuição de sinal (<=) e

atribuição de variável (:=). Vamos explorar essas atribuições em detalhes e fornecer exemplos de como usá-las.

Porém, antes é necessário explicar o que são sinais e variáveis. Em VHDL, sinais e variáveis são duas formas de armazenar dados, mas possuem diferenças importantes em termos de comportamento e aplicação. Vamos explorar as diferenças entre eles e quando é apropriado usar cada um:

Sinais:

Comportamento síncrono: Sinais em VHDL refletem o valor atual de uma variável somente após a conclusão do ciclo de simulação atual. Isso significa que as alterações em um sinal só se tornam visíveis no próximo ciclo de simulação.

Concorrente: Sinais são usados principalmente em processos concorrentes (como em processos em que a sensibilidade é apenas à mudança de valores de sinais), e eles são mais adequados para modelar a interconexão entre componentes em um design.

Variáveis:

Comportamento assíncrono: Variáveis em VHDL refletem o valor atual imediatamente após a conclusão da atribuição. Isso significa que as alterações em uma variável são imediatamente visíveis na próxima instrução.

Local para processos: Variáveis são mais adequadas para uso local em processos sequenciais, como para cálculos intermediários ou armazenamento temporário de valores. Elas são eficazes para manipulações de dados dentro de um único processo.

Quando usar cada um:

Use sinais quando precisar representar a interconexão entre componentes em um design ou quando precisar refletir o comportamento síncrono de um sistema digital, como em processos concorrentes.

Use variáveis quando precisar de armazenamento temporário de dados dentro de um único processo ou quando precisar refletir o comportamento assíncrono de atribuição de valores em um contexto sequencial.

Exemplos:

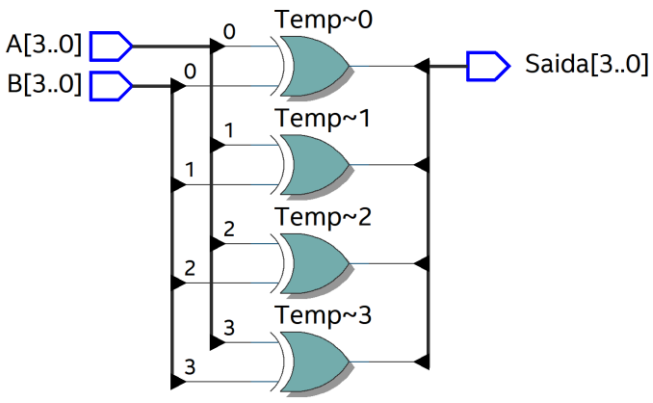
Exemplo de uso de sinal:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
   Port (
      A, B: in STD_LOGIC_VECTOR(3 downto 0);
      Saida: out STD_LOGIC_VECTOR(3 downto 0)
   );
end main;

architecture Behavioral of main is
begin
```

```
    process(A, B)
        variable Temp: std_logic_vector(3 downto 0);
    begin
        Temp := (others => '0'); -- Inicializa Temp com todos os bits em '0'
        for i in 0 to 3 loop
            Temp(i) := A(i) xor B(i); -- Realiza a soma bit a bit usando XOR
        end loop;
        Saida <= Temp; -- Atribui o resultado ao sinal de saída
    end process;
end Behavioral;
```

Neste exemplo, a variável "Temp" é declarada dentro do processo e é usada para armazenar o resultado temporário da operação "XOR" dos bits de A e B. A cada iteração do loop, a variável "Temp" é atualizada de acordo com a operação XOR realizada entre os bits correspondentes de A e B. Finalmente, o resultado é atribuído ao sinal de saída.

O circuito correspondente a este código é:



Exemplo de uso de variável:

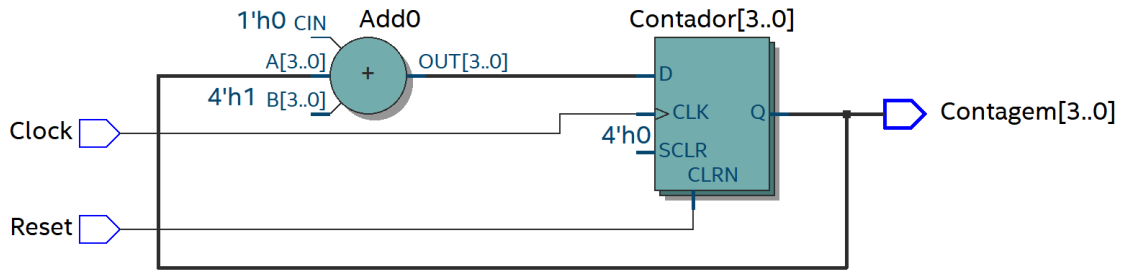
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Contador_Ascendente is
    Port (
        Clock: in STD_LOGIC;
        Reset: in STD_LOGIC;
        Contagem: out STD_LOGIC_VECTOR(3 downto 0)
    );
end Contador_Ascendente;

architecture Behavioral of Contador_Ascendente is
    signal Contador: unsigned(3 downto 0) := (others => '0');
begin
    process(Clock, Reset)
    begin
        if Reset = '1' then
            Contador <= (others => '0');
        elsif rising_edge(Clock) then
```

```
        Contador <= Contador + 1;
    end if;
  end process;

  Contagem <= std_logic_vector(Contador);
end Behavioral;
```

Neste exemplo, o sinal Contador é um sinal interno que armazena o valor do contador. Ele é inicializado com todos os bits em '0'. Durante cada borda de subida do sinal de clock, o valor do contador é incrementado em 1. Se o sinal de reset estiver em '1', o contador é reiniciado para '0000'.

Este exemplo mostra como usar sinais em VHDL para implementar um componente digital simples, como um contador crescente de 4 bits.

O circuito correspondente a este código é:



6.1 Atribuição de Sinal (<=)

A atribuição de sinal (`<=`) é usada para definir o valor de um sinal em VHDL. Ela é amplamente utilizada para atualizar o estado de sinais ou componentes em resposta a eventos ou em um processo. A atribuição de sinal é tipicamente usada dentro de processos e é sensível a mudanças nos sinais.

Exemplo de atribuição de sinal em um processo:

```
process (EntradaA, EntradaB)
begin
  -- Atualização do sinal de saída
  Saida <= EntradaA AND EntradaB;
end process;
```

Neste exemplo, o sinal “Saida” é atualizado com o resultado da operação lógica AND entre “EntradaA” e “EntradaB” sempre que ocorrer uma mudança em um dos sinais de entrada.

6.2 Atribuição de Variável (:=)

A atribuição de variável (:=) é usada para definir o valor de variáveis em VHDL. As variáveis são usadas para realizar cálculos intermediários dentro de processos ou funções. Ao contrário das atribuições de sinal, as atribuições de variável não são sensíveis a mudanças nos sinais.

Exemplo de atribuição de variável em um processo:

```
process (EntradaA, EntradaB)
   variable temp : STD_LOGIC;
begin
   -- Cálculo intermediário usando variável
   temp := EntradaA OR EntradaB;
   Saida <= temp AND NOT EntradaA;
end process;
```

Neste exemplo, a variável “temp” é usada para armazenar temporariamente o resultado da operação lógica OR entre “EntradaA” e “EntradaB”. Em seguida, a variável “temp” é usada em uma segunda operação, e o resultado é atribuído ao sinal “Saida”.

6.3 Resumo sobre Atribuições

- A atribuição de sinal (<=) é sensível a mudanças e geralmente é usada para atualizar o estado de sinais em resposta a eventos em sistemas digitais.
- A atribuição de variável (:=) é usada para cálculos intermediários dentro de processos ou funções. Variáveis são úteis quando você precisa armazenar temporariamente valores e realizar operações complexas.
- É importante observar que as atribuições de variável são restritas a processos ou funções locais, enquanto as atribuições de sinal podem afetar o estado global de sinais em todo o sistema.
- Certifique-se de entender as diferenças entre atribuições de sinal e atribuições de variável para usar a abordagem adequada em seu design em VHDL.

As atribuições são essenciais na descrição de hardware em VHDL, permitindo que você defina como os sinais e variáveis evoluem ao longo do tempo, garantindo que seu design funcione corretamente.

Operadores em VHDL

Os operadores em VHDL desempenham um papel crucial na realização de operações em sinais e variáveis. VHDL oferece uma ampla gama de operadores que podem ser usados para realizar cálculos, comparações e manipulações de dados em sistemas digitais. Neste capítulo, exploraremos vários tipos de operadores em VHDL, com exemplos de como eles são usados.

7 Operadores Aritméticos

Operadores aritméticos em VHDL são usados para realizar cálculos matemáticos em sinais e variáveis. Alguns dos operadores aritméticos comuns incluem:

- + (adição): Realiza a adição de dois valores.
- - (subtração): Realiza a subtração de dois valores.

- (multiplicação): Realiza a multiplicação de dois valores.
- / (divisão): Realiza a divisão de dois valores.
- MOD (módulo): Calcula o restante da divisão entre dois valores.

Exemplo de uso de operadores aritméticos:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity main is
  Port (
    A, B: in STD_LOGIC_VECTOR(7 downto 0);
    Resultado1, Resultado2, Resultado3, Resultado4, Resultado5: out STD_LOGIC_VECTOR(7 downto 0)
  );
end main;

architecture Behavioral of main is
  signal TempA, TempB : INTEGER;
  signal Temp1, Temp2, Temp3, Temp4, Temp5 : INTEGER;
begin
  process (A, B)
  begin
    TempA <= to_integer(unsigned(A));
    TempB <= to_integer(unsigned(B));

    Temp1 <= TempA + TempB;  -- Adição
    Temp2 <= TempA - TempB;  -- Subtração
    Temp3 <= TempA * TempB;  -- Multiplicação

    -- Verifica se o divisor é zero antes de fazer a divisão e o módulo
    if TempB /= 0 then
      Temp4 <= TempA / TempB;  -- Divisão
      Temp5 <= TempA mod TempB;  -- Módulo
    else
      Temp4 <= 0;  -- Define 0 se o divisor for zero
      Temp5 <= 0;  -- Define 0 se o divisor for zero
    end if;

    -- Converte os resultados para std_logic_vector
    Resultado1 <= std_logic_vector(to_unsigned(Temp1, 8));
    Resultado2 <= std_logic_vector(to_unsigned(Temp2, 8));
    Resultado3 <= std_logic_vector(to_unsigned(Temp3, 8));
    Resultado4 <= std_logic_vector(to_unsigned(Temp4, 8));
    Resultado5 <= std_logic_vector(to_unsigned(Temp5, 8));
  end process;
end Behavioral;
```

8 Operadores Lógicos

Operadores lógicos em VHDL são usados para realizar operações de lógica booleana em sinais e variáveis. Alguns dos operadores lógicos comuns incluem:

- AND (E lógico): Realiza a operação lógica "E" entre dois valores.
- OR (OU lógico): Realiza a operação lógica "OU" entre dois valores.
- NOT (NÃO lógico): Inverte o valor de um sinal.
- NAND (NÃO E lógico): Realiza a operação lógica "NÃO E" entre dois valores.
- NOR (NÃO OU lógico): Realiza a operação lógica "NÃO OU" entre dois valores.
- XOR (OU exclusivo lógico): Realiza a operação lógica "OU exclusivo" entre dois valores.

Exemplo de uso de operadores lógicos:

```
signal A, B, Resultado : STD_LOGIC;
Resultado <= A AND B; -- E lógico
Resultado <= A OR B; -- OU lógico
Resultado <= NOT A; -- NÃO lógico
Resultado <= A NAND B; -- NÃO E lógico
Resultado <= A NOR B; -- NÃO OU lógico
Resultado <= A XOR B; -- OU exclusivo lógico
```

9 Operadores de Comparação

Operadores de comparação em VHDL são usados para comparar valores e gerar resultados booleanos. Alguns dos operadores de comparação comuns incluem:

- = (igual a): Verifica se dois valores são iguais.
- /= (diferente de): Verifica se dois valores são diferentes.
- < (menor que): Verifica se um valor é menor que outro.
- `>` (maior que): Verifica se um valor é maior que outro.
- <= (menor ou igual a): Verifica se um valor é menor ou igual a outro.
- `>= (maior ou igual a): Verifica se um valor é maior ou igual a outro.

Exemplo de uso de operadores de comparação:

```
signal A, B : INTEGER;
Resultado1 <= (A = B); -- Verifica se A é igual a B
Resultado2 <= (A < B); -- Verifica se A é menor que B
Resultado3 <= (A > B); -- Verifica se A é maior que B
Resultado4 <= (A /= B); -- Verifica se A é diferente de B
```

10 Outros Operadores

Além dos operadores mencionados, VHDL oferece uma variedade de outros operadores, como:

- Concatenação de vetores: Usado para unir dois vetores em um único vetor.
- Deslocamento e rotação: Usados para deslocar e rotacionar valores em vetores.
- Conversão de tipo: Usado para converter valores entre tipos de dados diferentes.

10.1 Concatenação de Vetores:

A concatenação de vetores é uma operação fundamental em VHDL que permite unir dois vetores em um único vetor. Isso é útil quando você precisa criar um novo vetor combinando vários vetores menores. A sintaxe para concatenação de vetores em VHDL é o operador "&". Por exemplo:

```
vetor_concatenado <= vetor1 & vetor2;
```

Neste exemplo, vetor1 e vetor2 são concatenados em vetor_concatenado, onde vetor1 é colocado primeiro seguido por vetor2.

10.2 Deslocamento e Rotação:

Deslocamento e rotação são operações que permitem mover os valores dentro de um vetor para a esquerda ou para a direita. Em VHDL, essas operações são realizadas usando as funções shift_left, shift_right, rotate_left e rotate_right, fornecidas pela biblioteca NUMERIC_STD. Por exemplo:

```
vetor_deslocado <= shift_left(vetor_original, 2);
vetor_rotacionado <= rotate_right(vetor_original, 1);
```

Nestes exemplos, vetor_deslocado é obtido deslocando os valores de vetor_original duas posições para a esquerda, enquanto vetor_rotacionado é obtido rotacionando os valores de vetor_original uma posição para a direita.

10.3 Conversão de Tipo:

A conversão de tipo em VHDL é usada para converter valores entre diferentes tipos de dados. Isso é útil quando você precisa operar com valores de tipos diferentes ou quando precisa atribuir um tipo de dado a uma variável de outro tipo. Existem várias funções de conversão de tipo em VHDL, como to_integer, to_unsigned, to_signed, std_logic_vector, entre outras. Por exemplo:

```
variavel_integer := to_integer(vetor_std_logic_vector);
vetor_std_logic_vector := std_logic_vector(variavel_integer);
```

Nestes exemplos, to_integer é usado para converter um vetor de std_logic_vector em um valor inteiro, enquanto std_logic_vector é usado para converter um valor inteiro em um vetor de std_logic_vector.

Esses são conceitos fundamentais em VHDL que permitem manipular e operar com dados de maneira eficaz em projetos de hardware digital.

11 Sintaxe da Estrutura de Controle IF/ELSE

A estrutura de controle IF/ELSE em VHDL segue uma sintaxe semelhante a muitas linguagens de programação:

```
IF condição THEN
   -- Bloco de código a ser executado se a condição for verdadeira
ELSIF outra_condição THEN
   -- Bloco de código a ser executado se a primeira condição for falsa e a segunda condição for verdadeira
ELSE
   -- Bloco de código a ser executado se nenhuma das condições anteriores for verdadeira
END IF;
```

- IF: Inicia a estrutura condicional.
- condição: Uma expressão lógica que é avaliada como verdadeira (`TRUE`) ou falsa (`FALSE`).
- THEN: Indica o início do bloco de código a ser executado se a condição for verdadeira.
- ELSIF: Permite testar condições adicionais se a primeira condição for falsa.
- ELSE: Define um bloco de código a ser executado se nenhuma das condições anteriores for verdadeira.
- END IF: Finaliza a estrutura de controle.

11.1 Exemplo de Estrutura de Controle IF/ELSE

Aqui está um exemplo simples que ilustra o uso da estrutura de controle IF/ELSE em VHDL. Suponha que você deseja decidir qual sinal de entrada deve ser selecionado com base em uma variável de controle:

```
signal EntradaA, EntradaB, Saida : STD_LOGIC;
variable Selecao : STD_LOGIC := '1'; -- Variável de controle

-- Estrutura de controle IF/ELSE
IF Selecao = '0' THEN
   Saida <= EntradaA; -- Selecionar EntradaA se Selecao for '0'
ELSE
   Saida <= EntradaB; -- Selecionar EntradaB se Selecao não for '0'
END IF;
```

11.2 Estrutura de Controle IF/ELSE Aninhada

Você pode aninhar estruturas de controle IF/ELSE para lidar com decisões complexas. Por exemplo, considere um caso em que você deseja escolher entre várias opções com base em uma variável de controle:

```
variable Controle : INTEGER := 2; -- Variável de controle

-- Estrutura de controle IF/ELSE aninhada
IF Controle = 1 THEN
   -- Opção 1
   Saida <= "001";
ELSIF Controle = 2 THEN
   -- Opção 2
   Saida <= "010";
ELSIF Controle = 3 THEN
   -- Opção 3
```

```
    Saida <= "100";
ELSE
    -- Opção padrão
    Saida <= "000";
END IF;
```

12 Loops FOR e WHILE

Em VHDL, os loops FOR e WHILE são usados para criar iterações e repetições de código. Eles são cruciais para realizar tarefas repetitivas, como inicializar “arrays”, gerar sequências ou realizar operações iterativas em sistemas digitais. Neste capítulo, exploraremos os loops FOR e WHILE em detalhes, juntamente com exemplos de como usá-los.

12.1 Loop FOR em VHDL

O loop FOR é usado para executar um bloco de código um número fixo de vezes. É especialmente útil quando você precisa repetir uma operação um número conhecido de vezes, como inicializar registros ou vetores. A estrutura básica de um loop FOR em VHDL é a seguinte:

```
for contador in início to fim loop
    -- Bloco de código a ser repetido
end loop;
```

- contador: Uma variável que atua como contador.
- início: O valor inicial do contador.
- fim: O valor final do contador.
- loop: Inicia o bloco de código a ser repetido.
- end loop: Finaliza o loop.

Exemplo de Loop FOR:

Suponha que você deseja inicializar um vetor de 4 bits com zeros usando um loop FOR:

```
signal vetor : STD_LOGIC_VECTOR(3 downto 0);

for i in 0 to 3 loop
    vetor(i) <= '0'; -- Inicializa cada elemento com '0'
end loop;
```

Neste exemplo, o loop FOR executa quatro vezes (de 0 a 3) para inicializar cada elemento do vetor com o valor lógico '0'.