Utilização acadêmica da linguagem Verilog para construção de Sistemas Digitais

Eduardo Stefanello
UFFS

Resumo: O tema foi escolhido visando a necessidade em dominar a linguagem no segundo semestre de Ciência da Computação da UFFS e levando em conta a dificuldade que eu, assim como outros alunos do curso estão tendo em desenvolver seus trabalhos em Verilog, este artigo buscou exemplificar o funcionamento básico da linguagem.

Palavras Chave: Sistemas Digitais, Verilog.

1 - Introdução:

Na escala acadêmica, bem como na escala empresarial, a construção de sistemas digitais para automação de processos, sejam comerciais, industriais ou residenciais, tem exigido dos profissionais/estudantes da área, recursos cada vez mais avançados para confecção dos projetos.

Nesse artigo, será trazido à discussão o uso da linguagem Verilog para criação desses sistemas. Segundo Luiz Machado Chiesse, Verilog é uma linguagem de descrição de hardware (HDL), usado para especificar o padrão de comportamento de um sistema, antes deste ser implementado em hardware. Verilog é usado tanto para simulação quanto para síntese de sistemas.

Há disponível um material muito amplo sobre o assunto, principalmente em sites de universidades. Aqui utilizaremos artigos da universidades brasileiras UTFPR e UFPE e da universidade britânica de Cambridge buscando exemplificar o funcionamento básico e as principais funções e formas de sintaxe da linguagem, além de regras de implementação que devem ser seguidas.

2 - Visão Geral da Linguagem:

De acordo com Gordon e Chiesse, Verilog é uma linguagem HDL *(hardware description language – linguagem de descrição de hardware)* Padrão IEEE, usada em

projetos de sistemas digitais, da simulação do projeto até a síntese do mesmo através de um FPGA (*Field Programmable Gate Array - Matriz de Portas Programável em Área)* em diferentes níveis de abstração (portas, estruturas comportamentais, estruturas lógicas e laços) (The Semantic Challenge of Verilog HDL, Mike Gordon, Cambridge).

A Implementação em Verilog consiste em um ou mais módulos, em um sistema fechado que contém os modelos de hardware que serão executados por simuladores.

Cada módulo é formado essencialente de entradas e saídas, sendo introduzidos registradores (que armazenarão os valores) e fios (que ligarão os componentes uns nos outros, além de dispositivos lógicos comparadores *(if, else e case)* e laços *(always),* entre outros*.*

Segunda Marcelo Chiesse, A síntese se dá pela tradução do código *HDL* para o dispositivo que se está trabalhando *(FPGA, ASIC...)* em que as bibliotecas dos mesmos são as mesmas usadas na tradução e na operação de síntese.

3 - Estrutura de Linguagem

Para se trabalhar em verilog, assim como qualquer linguagem de programação, as regras de sintaxe da linguagem devem ser respeitadas para que o código possa ser compilado, simulado e eventualmente sintetizado. A seguir veremos os comandos e formatos referentes à programação em Verilog.

3.1 - Módulos

Em Verilog, o conceito fundamental para a implementação de um sistema/circuito é o módulo. Tudo que será executado pelo sistema deve estar dentro de um módulo. Podendo ser utilizados mais de um módulo em um sistema (Camadas de Abstração) (Chiesse).

Qualquer módulo Verilog está estruturado entre as palavras chave “module” e “endmodule”. Dentro do módulo são declaradas as portas (entradas, saídas, sinais bidirecionais e vetores), declaração dos tipos de dados (fios ou barramentos e registradores), declaração do fluxo de dados *(assign)* e descrição de comportamentos, ou seja, tudo o que ocorrerá na execução do bloco do módulo.

Exemplo:

```
module nome_do_modulo ( /*declaração de entradas e saídas*/);
      /*desenvolvimento do módulo*/
endmodule
```

Cada módulo de um sistema deve ter um nome diferente, o nome aparece após a palavra *module* e antes das declarações de entrada e saída.

Como Verilog é uma linguagem *case sentivive* os nomes *nome_do_modulo* e *Nome_Do_Modulo* são nomes diferentes.

3.2 - Portas

Para que a execução de um módulo funcione de maneira correta, ocorrem três passos básicos, na seguinte ordem: entrada, processamento e saída. Conforme Chiesse, esses são os meios primarios de comunicação com um módulo.

Na declaração de um módulo em Verilog são definidas a(s) entrada(s) e a(s) saída(s) no início. Essas entradas e saídas podem ser do tipo *input* ou *output,* simples ou em forma de vetor, ou *inout* que são sinais bidirecionais (funcionam tanto como entrada quanto saída).

Durante a execução do módulo podem ser atribuídos os valores das entradas e das saídas (através de atribuição direta ou de cálculos).

Exemplo:

```
module nome_do_modulo ( input nome_entrada,          //entrada simples
                        output [3:0] nome_saida,     //saída array de 4 bits
                        inout a, b);                 //dois sinais bidirecionais
      /*desenvolvimento do módulo*/
endmodule
```

Observe que cada tipo declarado no módulo é separado por vírgula e os dados não necessariamente precisam estar em linhas diferentes, essa forma é usada para dar mais legibilidade ao código.

3.3 - Tipos de Dados

Existem basicamente dois tipos de dados em Verilog: *wire* (fio ou barramento) e *reg* (registrador) que podem variar em com ou sem bit de sinal *(wire, reg, wire singed, reg signed)* e tipos simples ou *arrays (*vetores);

a) vetores/*arrays*

Vetores são tipos de dados que armazenam mais do que um único sinal (valor de

bit). Vetores podem assumir diversos tamanhos, que são declarados entre o tipo de dado ou porta e o nome do componente, por exemplo: *wire [x:0] nome, reg [x:y] nome)* em que *x* e *y* são o intervalo de bits que o tipo declarado irá armazenar/conduzir.

Exemplo:

```
module nome_do_modulo ( /*declaração de entradas e saídas*/);
      wire [7:0] barramento_A;      //fio de 8 bits
      reg registro1;                //registrador de 1 bit
      reg signed [3:0] registro2;   //registrador de 4 bits com sinal
      /*restante do desenvolvimento do módulo*/
endmodule
```

3.4 - Comando Assign

Este comando é utilizado para modelar circuitos combinacionais, de forma a combinar um uma entrada com uma saída, um fio com um registrador... a fim de que a saída mude sempre que a entrada for alterada, um registrador numa constante, numa operação aritmética... O comando assign cria uma ligação física entre um componente e outro no sistema.

Exemplo:

```
module nome_do_modulo (/*declaração de entradas e saídas*/);
      wire meu_fio;
      reg meu_reg;
      assign meu_reg = meu_fio;
      /*restante do desenvolvimento do módulo*/
endmodule
```

Neste exemplo, sempre que o valor de *meu_fio* for alterado, a mesma alteração ocorrerá em *meu_reg*.

3.5 - Operadores

De acordo com Chiesse, Verilog utiliza operadores matemáticos e lógicos muito semelhantes aos utilizados por linguagens de programação convencionais (C, Java, Phyton...), são eles:

- Operadores Aritméticos
  - (+): Adição;
  - (-): Subtração;
  - (*): Multiplicação;
  - (/): Divisão;

    - (%): Módulo ou resto da divisão.
- Operadores Lógicos:
    - (!) ou (~): Negação
    - (&&): And/e;
    - (||): Or/ou.
        - Podem ainda serem usadas as formas hibridas dos operadores lógicos acima: (!&): Nand e (!|): Nor
- Comparadores:
    - (<): Menor que;
    - (>): Maior que;
    - (<=): Menor ou igual que;
    - (>=): Maior ou igual que;
    - (==): Igual à:
    - (!=): Diferente de.
- *Shift:*
    - (<<): Rotaciona os bits à esquerda;
    - (>>): Rotaciona os bits à direita.
- Concatenação:
    - ({...}): Concatena as variáveis que estiverem dentro das chaves em uma única variável, na ordem que forem colocadas. Deve ser respeitada a soma da quantidade de bits de cada variável concatenada a fim de o valor “caber” na nova variável.

3.6 - Bloco *always*:

O Bloco always é uma parte do código que será executada sempre que houver mudança em determinada condição durante a execução do módulo, como por exemplo, na utilização de um circuito sequencial baseado em *clock*, em que determinada ação ocorrerá na borda de subida ou de descida do sinal/pulso de *clock*.

Exemplo:

```
always @(posedge clk) begin
      a <= ~a;    //atribuição de valor a a
end
```

No caso acima, devido ao uso da palavra *posedge* a atribuição acontecerá em cada descida da borda de clock, ou seja, sempre que o clock passar de nível lógico alto para baixo. Ao invés de *posedge* poderia ter sido utilizado a palavra *negedge* para a atribuição ocorrer na subida da borda de clock ou ainda nao ser utilizada nenhuma das duas palavras, para a atribuição ocorrer sempre que houver mudança no valor de clock (Felipe de A. Souza).

Na condição do bloco *always* podem ser usados comparadores e operadores lógicos.

Na declaração do bloco, devem ser usadas as palavras begin/eng para delimitar a área de execução do bloco de comandos. Essas mesmas palavras também devem ser usadas nos próximos itens para o mesmo objetivo.

3.7 - Blocos Condicionais:

Na definição de Souza, Todas as condicionais em Verilog só podem ser executadas dentro de blocos *always.* Condicionais são utilizadas para executar determinados comando se a condição for verdadeira no momento de verificação da condição.

Diferente do bloco *always*, as condicionais *if/else* e *case* não serão executadas sempre que a condição for verdeira, mas apenas se for verdadeira no momento em que o bloco *always* ao qual pertence for executado.

a) *if/else*:

Exemplo:

```
always @(/*condição do bloco always*/) begin
      if (sinal == 1) begin
            saida <= 5;
      end
      else begin
            saida <= 3;
      end
end
```

No caso acima, se a variável *sinal* tiver valor igual à 1, então será atribuído o valor 5 à variável *saida*, caso contrário será atribuído o valor 3 à *saida*. O comando *if/else* é mais utilizado quando há poucas possibilidades para determinada condição.

b) *case*:

Exemplo:

```
always @(/*condição do bloco always*/) begin
      case (sinal)
            0: saida <= 3;
            1: saida <= 5;
            default: saida <= 0;
      endcase
end
```

No caso acima, se a variável *sinal* tiver valor igual à 0, entao será atribuído o valor 3 à variável *saida, caso for 1,* o valor a ser atribuído será 5, caso nenhum dos dois casos forem atendidos será atribuído o valor 0 à *saida*.

Para executar mais de uma linha em cada um dos case, devem ser usadas as palavras *begin/end* para delimitar o bloco de cada case. O comando case é mais utilizado quando houverem vários casos possíveis, cada um com uma resposta diferente.

3.8 - Laços de Repetição:

De forma semelhante aos laços de repetição em outras linguagens de programação, em Verilog há três tipos de laços de repetição: *while, for, repeat.* Todo laço deve ser executado dentro de um bloco *always*.

A execução do bloco de comandos de um laço se repete até que sua condição de existência se torne falsa.

a) *while*:

Exemplo:

```
always @(/*condição do bloco always*/) begin
      while (counter < 50) begin
            counter = counter + 1; //incremento de 1 a cada interação
      end
end
```

No exemplo acima, cada vez que a condição do bloco *always* se tornar verdadeira, se a condição *counter é menor que cinquenta* for verdadeira a linha dentro do bloco *while* será executada quantas vezes necessário até que a condição se torne falsa.

b) *for*:

Exemplo:

```
always @(/*condição do bloco always*/) begin
      for (counter = 1; counter < 50, counter = counter + 1) begin
            clk = !clk;
      end
end
```

Neste exemplo, o que ocorre é o mesmo que no exemplo anterior, com a diferença que o valor inicial da variável *counter* é atribuído na chamada do comando, assim como o incremento de 1 ocorre sempre no final da execução do bloco *for.* Nesse caso a cada interação, o sinal de clock será invertido.

c) *repeat*:

Exemplo:

```
always @(/*condição do bloco always*/) begin
      repeat (15) begin
            clk = !clk;
      end
end
```

No comando *repeat*, haverá a repetição do bloco de comandos de acordo com o parâmetro informado na chamada de função, neste caso, quinze vezes.

Segundo Reza, este comando não pode ser usado na síntese dos circuitos.

3.9 - *initial:*

O comando *initial* é executado uma única vez, no início da execução do código, ele geralmente é utilizado para atribuir valores às variáveis que serão utilizadas pelos blocos always, laços ou condicionais dentro deles ou por chamadas de outros módulos.

Exemplo:

```
module nome_do_modulo ( input [3:0] a, outbut [3:0] b);
      assign b = a;
      initial begin
            a = 10;
      end
endmodule
```

Neste exemplo, o valor de *a* foi atribuído apenas uma vez, consequentemente o valor da saída será o mesmo até que durante a execução o valor se altere.

3.10 - Comando *parameter:*

Este comando é utilizado para rotular dados que serão utilizados muitas vezes no código.

Exemplo:

```
module nome_do_modulo (outbut [3:0] b);
      parameter nome_prmtr = 10;
      assign b = nome_prmtr;
endmodule
```

Na implementação acima, a variável *b* irá receber o valor 10.

3.11 - Notas:

De acordo com a página Asic-World, para efetuar testes em um código escrito em Verilog, temos a opção de utilizar um simulador como o aplicativo de análise GTKWave (entre outros) ou imprimir os valores de saída na tela do terminal, com o comando *$display(“%d”, nome_da_varivel);* (parâmetros idênticos à função *printf* em C).

Para testes no aplicativo de análise GTKWave, dese ser utilizada a função *$dumpvars(x, nome_da_variavel);* para cada variável, para que seus resultados sejam expressos na tela do aplicativo.

Para atribuir valores, usamos por padrão, valores decimais. Porém podemos usar, por exemplo a notação 9'b000011110, em que inserimos o conjunto de 9 bits que representa o valor 30 em decimal. Também podem ser inseridos valores em hexadecimal, por exemplo: 4'hA, em que inserimos o conjunto de 4 bits que representam o valor 10 em decimal.

4 - Referências:

1. SOUZA, Felipe de Assis. Tutorial Verilog, Universidade Federal de Pernambuco, <http://www.cin.ufpe.br/~eaa3/Arquivos/Verilog/Tutorial%20Verilog.pdf> Acesso em 03 abr. 2016
2. GORDON, Mike. The Semantic Challenge of Verilog HDL – Mike Gordon – University of Cambridge UK - <http://www.pld.guru/_hdl/4/-cl.cam.ac.uk/~mjcg/verilog/V.pdf> Acesso em 02 abr. 2016
3. REZA, Yuri. Hardware Description Language (HDL) Para que precisamos de uma Linguagem de Descrição de Hardware? Modelar, Representar e Simular Hardware Digital Concorrete - <http://slideplayer.com.br/slide/367940/> Acesso em 03 abr. 2016
4. SILVA, Luiz Marcelo Chiesse da. Curso Básico de Verilog. Universidade

Tecnológica Federal do Paraná, Campus de Cornélio Procópio. Disponível em: <http://paginapessoal.utfpr.edu.br/chiesse/disciplinas/logica-reconfiguravel/verilog/Curso%20Basico%20de%20Verilog.pdf>. Acesso em: 04 abr. 2016

5. Portal Asic World, Diretório sobre Verilog <http://www.asic-world.com/verilog/index.html> Acesso em 16 abr, 2016