

REVISÃO SOBRE VERILOG

Prof. Artino Quintino da Silva Filho

Introdução

Verilog
conceitos
básicos

Verilog
Modelagem



Verilog

modelagem

Declaração
concorrente

Descrição
Estrutural

Descrição
procedural

Testbench

Declarações concorrentes

Em qualquer linguagem de descrição de hardware, incluindo Verilog, o conceito de **declaração concorrente** significa que se o código incluir várias dessas declarações, cada uma representará uma parte do circuito.

Usamos a palavra **concorrente** porque *as instruções são consideradas em paralelo e a ordem das instruções no código não importa*. Instanciações de portas lógicas são um tipo de instrução simultânea.

Esta seção apresenta outro tipo de instrução simultânea, chamada de **atribuição contínua**.

Descrição por Atribuições (assign)

Enquanto o uso de portas lógicas permitem a descrição da estrutura de um circuito, as **atribuições contínuas** permitem a descrição da *função* de um circuito.

A palavra assign serve para atribuir um sinal a uma wire, e somente pode ser feita a wires. Com ela, defini-se como os sinais estão interligados em um circuito.

assign



assign – usado para atribuir um valor a uma entrada ou saída.

- **Sintaxe e Semântica:** assign nome_da_entrada/saida = valor;

Exemplo

```
input [3:0] In1;           // Declarando uma entrada, In1, de 4 bits
(...)
assign In1 = 4'b1001;      // In1 recebe a cadeia de 4 bits 1001.
(...)
```

Exemplo

```
input [7:0] In1, In2;      // Declarando duas entradas, In1 e In2, de 8 bits
output Status;            // Declarando uma saída, Status, de 1 bit
(...)
assign Status = (In1 > In2); // * Status recebe 1, se In1 for maior do que In2,
                             // * caso contrário, recebe 0 */
```

Descrição usando parâmetros (parameter)

Permite generalizar o código

parameter



parameter – usado para rotular dados que serão utilizados muitas vezes, contribuindo para a legibilidade do código.

- **Sintaxe e Semântica:**

parameter ROTULO = cadeia de bits;

OBS.: As cadeias 0000, 0001, 0010 foram rotuladas como CLRLD, ADDLD, HOLD, respectivamente. Sempre que for preciso usar essas cadeias no código, não será necessário digitá-las novamente, podendo substituí-las por seus respectivos rótulos.

parameter



Exemplo

```
input [3:0] Instrucao;           // Declarando uma entrada, Instrucao, de 4 bits
(...)
parameter CLR LD = 4b'0000;      // Rotulando a cadeia de 4 bits 0000 como CLR LD
parameter ADD LD = 4b'0001;      // Rotulando a cadeia de 4 bits 0001 como ADD LD
parameter HOLD = 4b'0010;        // Rotulando a cadeia de 4 bits 0010 como HOLD
(...
case (Instrucao)
    CLR LD: begin
        (...)

    end
    ADD LD: begin
        (...)

    end
    HOLD: begin
        (...)

    end
    (...)

endcase
(...)
```

// Declарando uma entrada, Instrucao, de 4 bits

// Rotulando a cadeia de 4 bits 0000 como CLR LD

// Rotulando a cadeia de 4 bits 0001 como ADD LD

// Rotulando a cadeia de 4 bits 0010 como HOLD

// Início de um bloco do tipo 'case'

/* Caso Instrucao seja igual a 0000, execute

* até o próximo 'end' */

/* Caso Instrucao seja igual a 0001, execute

* até o próximo 'end' */

/* Caso Instrucao seja igual a 0010, execute

* até o próximo 'end' */

// Fim do bloco

Verilog

modelagem

Declaração
concorrente

Descrição
Estrutural

Descrição
procedural

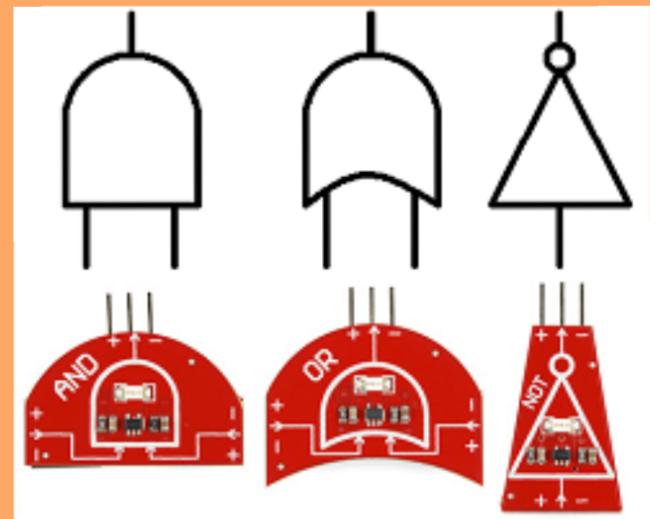
Testbench

Operadores lógicos



- Representa circuitos lógicos usando primitivas da linguagem Verilog
- Verilog comporta um conjunto de instruções primitivas que correspondem às portas lógicas.
- Uma porta lógica é representada indicando o seu nome funcional, a saída e a(s) entrada(s).

nome_funcional [identificador] (saída, entrada)



Operadores lógicos

Nome	Descrição	em Verilog
and	$f = (a \cdot b \cdot \dots)$	and (f, a, b, \dots)
nand	$f = \overline{(a \cdot b \cdot \dots)}$	nand (f, a, b, \dots)
or	$f = (a + b + \dots)$	or (f, a, b, \dots)
nor	$f = \overline{(a + b + \dots)}$	nor (f, a, b, \dots)
xor	$f = (a \oplus b \oplus \dots)$	xor (f, a, b, \dots)
xnor	$f = (a \odot b \odot \dots)$	xnor (f, a, b, \dots)
not	$f = \bar{a}$	not (f, a)
buf	$f = a$	buf (f, a)
notif0	$f = (!e ? \bar{a} : 'bz)$	notif0 (f, a, e)
notif1	$f = (e ? \bar{a} : 'bz)$	notif1 (f, a, e)
bufif0	$f = (!e ? a : 'bz)$	bufif0 (f, a, e)
bufif1	$f = (e ? a : 'bz)$	bufif1 (f, a, e)

Operadores lógicos

Verilog comporta um conjunto de instruções primitivas que correspondem às portas lógicas.

Uma porta lógica é representada indicando o seu nome funcional, saída e entradas.

Exemplo:

1) porta AND, com saída y e duas entradas x1 e x2 pode ser declarada da seguinte forma:

and (y, x1, x2);

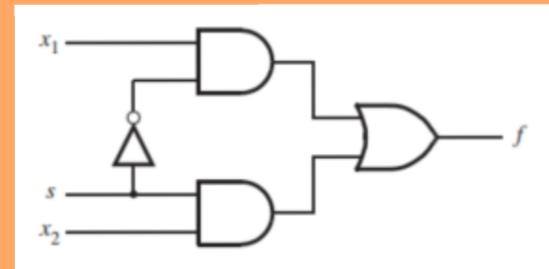
1) porta OR, com quatro entradas pode ser declarada da seguinte forma:

or (y, x1, x2, x3, x4);

Operadores lógicos

Exemplo:

```
module multiplexador (x1,x2,s,f);  
  
input x1, x2, s;  
output f;  
  
not (k, s);  
and (g, k, x1);  
and (h, s, x2);  
or (f, g, h);  
  
endmodule
```



Descrição estrutural



operações lógicas	sintaxe
and	&
or	
xor	^
not	~
nand	~&
nor	~
right shift	>>
left shift	<<
concatenacao	{ }
condicional	?

operações aritméticas	sintaxe
adicao	+
subtracao	-
multiplicacao	*
divisao	/
modulo	%

Concatenação

- Transforma duas variáveis em uma única variável:



```
assign c = {a, b};
```

a	b
c	

```
assign { a, b} = c  
assign {cout, out} = a + b;
```

Descrição estrutural

- Resumir uma quantidade de bits em uma única operação



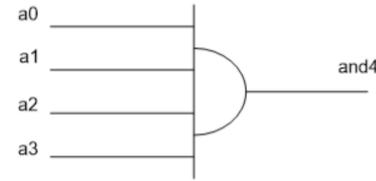
Exemplo:

assign and4 = &a;

Equivale a:

assign and4 = a[3]&a[2]&a[1]&a[0];

Possível também para: & (and), (or), ^ (xor)



Descrição estrutural

- Retornam um valor TRUE (1) ou FALSE (0)



operações de comparação

Igualdade
Desigualdade
Menor que
Menor/igual
Maior que
Maior/igual

sintaxe

==
!=
<
<=
>
>=

Exemplos:

```
reg [3:0] a,b;  
wire comp; // quando a igual a b comp = 1  
assign comp = (a == b);
```

Verilog

modelagem

Declaração
concorrente

Descrição
Estrutural

Descrição
procedural

Testbench

Descrição procedural

- Blocos procedurais
- Descrição Procedural com comandos if-else
- Descrição Procedural com comandos case
- Descrição Procedural com laço
- Tarefas do sistema
- Controle por evento

Descrição procedural

Em adição às declarações concorrentes, Verilog comporta **declarações procedurais - ou declarações sequenciais** -, permitindo descrever um circuito em linguagem de nível mais alto.

Enquanto as declarações concorrentes são executadas em paralelo, as declarações procedurais são executadas na ordem em que aparecem no código.

Um bloco procedural começa com **always @** , seguido de uma lista de sinais que servem para ativar o bloco - chamada de **lista de sensibilidade** -, que afetam diretamente à saída resultante. Ou seja, a lista de sensibilidade diz quando o bloco é executado.

A semântica do bloco **always** diz que se um valor de um sinal em uma lista de sensibilidade muda, então todas as declarações dentro do bloco **always** se alterarão na ordem apresentada.

Descrição procedural

Quando múltiplos blocos de comando são inclusos dentro de um bloco always, o par de comandos **begin-end** é necessário.

Um módulo Verilog pode incluir diversos blocos always, onde cada um representa uma parte do circuito a ser modelado.

Um detalhe importante é que **wires** não podem ser alteradas dentro de blocos procedurais porque representam as vias físicas (fios) que estão permanentemente conectadas às entradas e saídas.

Desta forma, é necessário utilizar **nets** do tipo **regs**, que podem ser alteradas dentro de blocos procedurais.

always



Sintaxe e Semântica

```
always @ (entrada1 or ... or entradaN) begin  
(...)  
código;  
(...)  
end
```

Exemplo

```
input [2:0] ln1, ln2, ln3;           // Declarando três entradas, ln1, ln2 e ln3  
(...)  
always @ (ln1 or ln2 or ln3) begin /* Início do bloco. Sempre que ln1, ln2 ou ln3  
* mudarem de valor, o código dentro do bloco  
* será executado */  
(...)  
código;  
(...)  
end                                // Fim do bloco
```

begin/end



begin/end – utilizado para iniciar uma sequência de comandos. Quando há apenas um comando a ser executado, não é necessária a utilização de begin/end, mas é recomendada, para melhorar a legibilidade do código.

- **Sintaxe e Semântica:**

```
begin
    comando1;
    comando2;
    ...
    comandoN;
end
```

initial



Para o propósito de simulação o Verilog oferece o bloco initial, que funciona igual ao bloco always, mas difere no fato de que as declarações dentro de seu bloco só funcionam no início da simulação.

initial: executa uma unica vez no inicio da simulação
não sintetizável

always: executa repetidamente
sintetizavel

```
...  
initial begin  
    Sum = 0;  
    Carry = 0;  
end  
...
```

```
...  
always @(A or B) begin  
    Sum = A ^ B;  
    Carry = A & B;  
end  
...
```

Descrição procedural

- Blocos procedurais
- Descrição Procedural com comandos if-else
- Descrição Procedural com comandos case
- Descrição Procedural com laço
- Tarefas do sistema
- Controle por evento

Descrição procedural com comandos if-else

A forma geral da declaração **if-else** diz que, caso a **expressão_1** seja verdadeira, então a **declaração_1** é executada.

Else if e **else** são opcionais.

A sintaxe Verilog especifica que quando **else if** ou **else** estão inclusas, eles se relacionarão com um **if** ou **else if** mais recente não finalizado.

Formato:

```
if (condição_1)
    declaração_1
else if (condição_2)
    declaração_2
else
    declaração_3
```

Exemplo:

```
if (reset)
Q = 0;
else
Q = D;
```

Sinais (if/else)



if/else – bloco condicional semelhante ao bloco if/else de Java. Só pode ser utilizado dentro de blocos always.

- **Sintaxe e Semântica:**

```
if (condicao) begin
  ...
  código
  ...
end else begin
  ...
end
```

Sinais (if/else)



Exemplo

```
input [3:0] A, B;  
output [3:0] Out;  
(...)  
if (A > B) begin  
    Out <= A - B;  
end else begin  
    Out <= B - A;  
end  
(...)
```

// Declarando uma entrada, In, de 4 bits
// Declarando uma saída, Out, de 4 bits

/* Início do bloco. Se A for maior do que B:
 * Out recebe o valor da operação A - B,
 * caso contrário:
 * Out recebe o valor da operação B - A */
// Fim do bloco

Descrição procedural

Blocos
procedurais

Descrição
Procedural com
comandos if-else

Descrição
Procedural com
comandos case

Descrição
Procedural com
laço

Tarefas do
sistema

Controle por
evento

Descrição procedural com comandos case

Os bits fornecidos na expressão de controle são verificados para uma correspondência com cada alternativa.

A primeira correspondência bem-sucedida faz com que as instruções associadas sejam executadas.

Cada dígito em cada alternativa é comparado para uma correspondência exata dos quatro valores 0, 1, X ou Z.

Sinais (case/endcase)



case/endcase – bloco condicional, semelhante ao bloco case em Java.

OBS.: A keyword default é usada para descrever uma ação padrão, ou seja, caso a entrada não seja nenhuma das previstas pelo programador, um código “padrão” também definido pelo programador será executado. Seu uso não é obrigatório.

Sinais (case/endcase)



- Sintaxe e Semântica:

```
case (entrada)
possivel_entrada 1: begin
    ...
    código;
end
...
possivel_entrada N: begin
    ...
    código;
end
default: begin
    ...
    código;
end
endcase
```

Sinais (case/endcase)



Exemplo

```
input [1:0] In;          // Declarando uma entrada, In, de 2 bits
output [3:0] Out;        // Declarando uma saída, Out, de 4 bits
(...)
case (In)               // Início do bloco
  2'b00: Out <= 4'b0010; // Caso In seja igual a 00, Out receberá 0010
  2'b01: Out <= 4'b1010; // Caso In seja igual a 01, Out receberá 1010
  2'b10: Out <= 4'b0110; // Caso In seja igual a 10, Out receberá 0110
  2'b11: Out <= 4'b0111; // Caso In seja igual a 11, Out receberá 0111
endcase                  // Fim do bloco
```

Sinais (case/endcase)



Exemplo

```
input [1:0] In;
input Sel;
output [3:0] Out;
(...)

case (In)
  2'b00: Out <= 4'b0010;
  2'b01: Out <= 4'b1010;
  2'b10: begin
    Out <= 4'b0110;
    Sel <= 1;
  end
  default: Out <= 4'b0111;
endcase
```

// Declarando uma entrada, In, de 2 bits
// Declarando uma entrada, Sel, de 1 bit
// Declarando uma saída, Out, de 4 bits

// Início do bloco
// Caso In seja igual a 00, Out receberá 0010
// Caso In seja igual a 01, Out receberá 1010
/* Caso In seja igual a 10:
 * Out receberá 0110 e
 * Sel receberá 1 */

// Caso In seja igual a 11, Out receberá 0111
// Fim do bloco

Descrição procedural

Blocos
procedurais

Descrição
Procedural com
comandos if-else

Descrição
Procedural com
comandos case

Descrição
Procedural com
laço

Tarefas do
sistema

Controle por
evento

Descrição procedural com laço

Verilog inclui quatro tipos de declaração por laço:

- for
- while
- repeat
- forever

for

- A sintaxe **for** é muito semelhante ao loop **for** da linguagem C.
- O índice inicial é avaliado uma vez, antes da primeira iteração do loop, e normalmente executa a inicialização da variável de controle do loop inteiro, como $k = 0$.
- Em cada iteração do loop, o bloqueio de início e fim é executado e, em seguida, a instrução de incremento é avaliada .
- Uma declaração de incremento típica é $k = k + 1$.

Exemplo:

```
for (índice_inicial; índice_final; incremento)
begin
    declaração;
end
```

for

A sintaxe for é semelhante à utilizada na linguagem C.



Exemplo:

```
for (count = 0, count < 10; count = count + 1)
begin
    sum = sum + 5;
end
```

while e repeat

- O loop **while** tem a mesma estrutura da instrução correspondente na linguagem C.
- O loop **repeat** simplesmente especifica um número de vezes para repetir seu bloco begin e end.

while

```
while (condição)
begin
    declaração_1
end
```

repeat

```
repeat (numero_de_vezes)
begin
    declaração_1
end
```

while e repeat



Exemplo:

```
while (contagem < 10)
begin
    soma = soma + 5;
    contagem = contagem + 1
end
```

Exemplo:

```
repeat (contagem)
    soma = soma + 5;
```

Descrição procedural

- Blocos procedurais
- Descrição Procedural com comandos if-else
- Descrição Procedural com comandos case
- Descrição Procedural com laço
- Tarefas do sistema
- Controle por evento

Tarefas do sistema

São utilizadas para inserir funcionalidades no modelo que não serão associadas ao circuito real.

Existem 3 grupos principais de diretivas:

- exibição
- entrada/saída de arquivos
- controle de simulação

Todas as diretivas começam com **\$** e só podem ser utilizadas durante a Simulação.
Elas serão ignoradas na sintetização

Tarefas do sistema

Exibição

\$display: mostra toda a lista no momento em que a declaração é encontrada

\$monitor: sempre que há uma mudança em qualquer argumento, exibe toda a lista no final do passo de tempo

entrada/saída de arquivos

\$fopen: abre um arquivo e retorna o seu valor

\$fclose: fecha o arquivo associado

Controle de simulação

\$finish: faz o simulador sair

\$stop: suspende a simulação

\$time: dá o tempo de simulação

Exemplo:



```
module tb;
initial begin
    reg [7:0] a;
    reg [39:0] str = "Hello";
    time cur_time;
    real float_pt;

    a = 8'hoE;
    float_pt = 3.142;

    $display ("a = %h", a);
    $display ("a = %d", a);
    $display ("a = %b", a);

    $display ("str = %s", str);
#200 cur_time = $time;
    $display ("time = %t", cur_time);
    $display ("float_pt = %f", float_pt);
    $display ("float_pt = %e", float_pt);
end
endmodule
```

Code	Format
%b	Binary values
%o	Octal values
%d	Decimal values
%h	Hexadecimal values
%f	Real values using decimal form
%e	Real values using exponential form
%t	Time values
%s	Character strings
%m	Hierarchical name of scope (no argument required when printing)
%l	Configuration library binding (no argument required when printing)

Exemplo:



```
module tb;
initial begin
    reg [7:0] a;
    reg [39:0] str = "Hello";
    time cur_time;
    real float_pt;

    a = 8'hoE;
    float_pt = 3.142;

    $display ("a = %h", a);
    $display ("a = %d", a);
    $display ("a = %b", a);

    $display ("str = %s", str);
#200 cur_time = $time;
    $display ("time = %t", cur_time);
    $display ("float_pt = %f", float_pt);
    $display ("float_pt = %e", float_pt);
end
endmodule
```

Log após simulação

```
ncsim> run
a = 0e
a = 14
a = 00001110
str = Hello
time = 200
float_pt = 3.142000
float_pt = 3.142000e+00
ncsim: "W,RNQUIE: Simulation is complete."
```

Code	Format
%b	Binary values
%o	Octal values
%d	Decimal values
%h	Hexadecimal values
%f	Real values using decimal form
%e	Real values using exponential form
%t	Time values
%s	Character strings
%m	Hierarchical name of scope (no argument required when printing)
%l	Configuration library binding (no argument required when printing)