Curso de Verilog

CdH

Aula 2

Conteúdo

- Regras para conexão de portas
- Testbench
- Conceito de tempo em verilog
- Análise de Formas de Onda
- Níveis de Abstração: Dataflow

Regras para conexão de portas

- Regras
- Conectando a sinais externos

Regras para conexão de portas

- **Regras**
- Conectando a sinais externos

- Inputs
- Outputs
- Inouts
- Width Matching
- Unconnected ports
- Exemplo

CdH

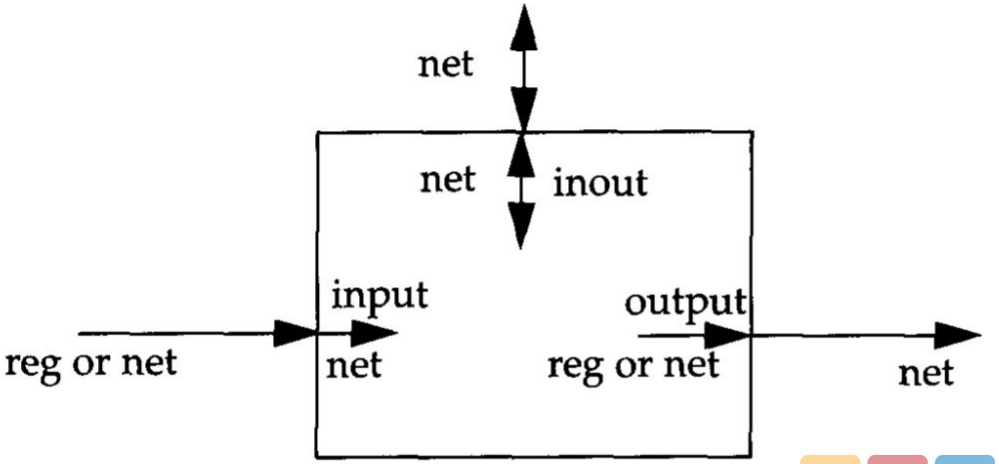
Regras para conexão de portas

- **Regras**
- Conectando a sinais externos

- Inputs
  - Internamente devem ser do tipo wire.
  - Externamente podem estar conectados a um reg ou ser wire.
- Outputs
  - Internamente podem ser do tipo wire ou reg.
  - Externamente devem estar conectados a um wire.
- Inouts
  - Devem ser do tipo wire tanto Interno quanto externamente.

Regras para conexão de portas

- **Regras**
- Conectando a sinais externos



Regras para conexão de portas

- **Regras**
- Conectando a sinais externos

- Width Matching
  - É possível conectar portas com tamanhos diferentes. Porém, a parte não conectada estará flutuando.
- Unconnected ports
  - É permitido deixar portas desconectadas de sinais externos.

Regras para conexão de portas

- **Regras**
- Conectando a sinais externos

- Exemplo:
  - fulladd4(sum,c_out,a,b, c_in)

```
fulladd4 fa0(SUM, , A, B, C_IN);
```

```
module Top;

//Declare connection variables
reg [3:0]A,B;
reg C_IN;
reg [3:0] SUM;
wire C_OUT;

   //Instantiate fulladd4, call it fa0
   fulladd4 fa0(SUM, C_OUT, A, B, C_IN);
```

Regras para conexão de portas

- Regras
- **Conectando a sinais externos**
  - Conectando numa lista ordenada
    - Os sinais devem ser ligados na mesma ordem que a lista de portas do módulo.
  - Conectando por nome
    - Os sinais são conectados pelo seu nome em qualquer ordem.

CdH

Regras para conexão de portas

- Regras
- **Conectando a sinais externos**

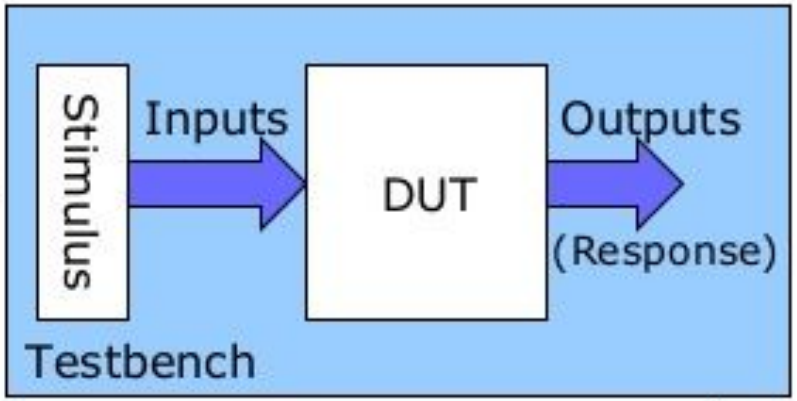
- Exemplo:

```
1  module fulladd4(sum, c_out, a, b, c_in);
2  output [3:0] sum;
3  output c_out
4  input [3:0] a,b;
5  input c_in;
6  ...
7  <module_internals>
8  ...
9  endmodule
```

```
fulladd4 fa_name(.a(x), .b(y), .c_in(carryin), .c_out(carry), .sum(res));
fulladd4 fa_order(res, carry, x, y, carryin );
```

Testbench

- Device Under Test (DUT)- Módulo que será Testado
- Testbench instancia o DUT, aplica estímulos a sua entrada.
- A saída pode ser monitorada usando as System tasks explicadas na aula passada.



FAZENDO UM TESTBENCH

1- Inicialize o módulo sem inputs ou outputs.

```
module seven_tb();
```

FAZENDO UM TESTBENCH

1- Inicialize o módulo sem inputs ou outputs.
2- Declare os inputs do módulo a ser testado como reg e os outputs como wire.

```
module seven_tb();

  reg [2:0] in;
  wire [6:0] out;
```

FAZENDO UM TESTBENCH

1- Inicialize o módulo sem inputs ou outputs.
2- Declare os inputs do módulo a ser testado como reg e os outputs como wire.
3- Instancie o módulo de teste passando como parâmetro os regs e wires criados.

```
module seven_tb();

  reg [2:0] in;
  wire [6:0] out;

 seven display7seg (
    .out(out),
    .IN(in)
 );
```

FAZENDO UM TESTBENCH

1- Inicialize o módulo sem inputs ou outputs.
2- Declare os inputs do módulo a ser testado como reg e os outputs como wire.
3- Instancie o módulo de teste passando como parâmetro os regs e wires criados.
4- Inicie o teste atribuindo valores de entrada
5- Dê um tempo entre cada atribuição usando #valor; para conseguir visualizar as formas de onda.
6- Lembre-se de fechar o módulo.

```
  initial begin
    in = 3'b000;
    #20
    in = 3'b001;
    #20
    in = 3'b010;
    #20
    in = 3'b011;
    #20
    in = 3'b100;
    #20
    in = 3'b101;
    #20
    in = 3'b110;
    #20
    in = 3'b111;
    #20
  end
endmodule
```

CdH

Prática 3 - Testbenches

Níveis de abstração

- Gate Level
- **Dataflow**
- Behavioral Procedural

● O assinalamento de dados para o output é contínuo.

●Usa-se o termo “assign”

Continuous Assignment

- Usado para lógica combinacional
- Executado continuamente pois é sensível a mudanças na expressão
- A variável do lado esquerdo da expressão (o que recebe a atribuição) deve ser do tipo wire.

```
assign a=b+c;
```

Tipo de Operadores em Verilog HDL

- Aritméticos
- Lógicos
- Relação
- Igualdade
- Bitwise
- Redução
- Shift
- Condicional
- Concatenação
- Replicação

Tipo de Operadores em Verilog HDL

- **Aritméticos**
- Lógicos
- Relação
- Igualdade
- Bitwise
- Redução
- Shift
- Condicional
- Concatenação
- Replicação

- Binários
  - *, /, +, -, %
  - Requer 2 operandos.
  - Se algum deles possuir valor x, o resultado da operação será x.
- Unários
  - + e -
  - Utilizados para representar se um operando é positivo ou negativo
  - É aconselhável usar números negativos somente no tipo integer ou real

CdH

Tipo de Operadores em Verilog HDL

- Aritméticos
- **Lógicos**
- Relação
- Igualdade
- Bitwise
- Redução
- Shift
- Condicional
- Concatenação
- Replicação

- &&, ||, !
- Operadores lógicos sempre atribuem como resultado: **0** (falso), **1** (verdadeiro), **x** (desconhecido).
- Se qualquer operando for **x** ou **z,** a operação retorna **x** e geralmente tratado como condição falsa.
- Operadores lógicos aceitam variáveis ou expressões como operandos

CdH

Tipo de Operadores em Verilog HDL

- Aritméticos
- Lógicos
- **Relação**
- Igualdade
- Bitwise
- Redução
- Shift
- Condicional
- Concatenação
- Replicação

- >, < >=, <=
- Se qualquer operando for **x** ou **z**, a operação retorna **x.**

Tipo de Operadores em Verilog HDL

- Aritméticos
- Lógicos
- Relação
- **Igualdade**
- Bitwise
- Redução
- Shift
- Condicional
- Concatenação
- Replicação

- Igualdade Lógica
  - ==, !=
- Caso Lógico
  - ===, !==

Tipo de Operadores em Verilog HDL

- Aritméticos
- Lógicos
- Relação
- Igualdade
- **Bitwise**
- Redução
- Shift
- Condicional
- Concatenação
- Replicação

- ~, &, |, ^, (~^,^~).
- A operação é realizada bit-a-bit em cada um dos operandos.
- Exemplo:

```
// X = 4'b1010, Y = 4'b1101
// Z = 4'b10x1

~X      // Negation. Result is 4'b0101
X & Y   // Bitwise and. Result is 4'b1000
X | Y   // Bitwise or. Result is 4'b1111
X ^ Y   // Bitwise xor. Result is 4'b0111
```

Tipo de Operadores em Verilog HDL

- Aritméticos
- Lógicos
- Relação
- Igualdade
- Bitwise
- **Redução**
- Shift
- Condicional
- Concatenação
- Replicação

- &,~&, |, ~|, ^, (~^,^~).
- A Operação é realizada nos bits do único operando.
- Exemplo:

```
// X = 4'b1010

&X //Equivalent to 1 & 0 & 1 & 0. Results in 1'b0
|X//Equivalent to 1 | 0 | 1 | 0. Results in 1'b1
^X//Equivalent to 1 ^ 0 ^ 1 ^ 0. Results in 1'b0
//A reduction xor or xnor can be used for even or odd parity
//generation of a vector.
```

CdH

Tipo de Operadores em Verilog HDL

- Aritméticos
- Lógicos
- Relação
- Igualdade
- Bitwise
- Redução
- **Shift**
- Condicional
- Concatenação
- Replicação

- >>, <<
- Operandos são um vetor e o número de bits que será shiftado.
- Exemplo:

```
// X = 4'b1100

Y = X >> 1; //Y is 4'b0110. Shift right 1 bit. 0 filled in MSB position
Y = X << 1; //Y is 4'b1000. Shift left 1 bit. 0 filled in LSB position.
Y = X << 2; //Y is 4'b0000. Shift left 2 bits.
```

CdH

Tipo de Operadores em Verilog HDL

- Aritméticos
- Lógicos
- Relação
- Igualdade
- Bitwise
- Redução
- Shift
- **Condicional**
- Concatenação
- Replicação

- ?:
- Possui 3 operandos:
  - <expressão>?<exp_verdadeira>:<exp_falsa>
- Pode ser aninhada.

Tipo de Operadores em Verilog HDL

Operator	Symbols
Unary, Multiply, Divide, Modulus	!, ~, *, /, %
Add, Subtract, Shift	+, - , <<, >>
Relation, Equality	<,>,<=,>=,==,!=,===,!==
Reduction	&, !&,^,^~,|,~|
Logic	&&, ||
Conditional	? :

http://www.asic-world.com/verilog/operators.html

CdH

Concatenação e Replicação

- As concatenações são expressadas usando os caracteres de chave { }, com vírgulas separando as expressões dentro.
- Exemplo:
  - x={a, b [3: 0], c, 4'b1001}
- Números constantes não dimensionados não são permitidos em concatenações.

- As replicações são expressadas da seguinte forma: {n{m}}
- Onde m é replicada n vezes
- Exemplo:
  - reg A;
  - reg [1:0]B;
  - A= 1'b1; B= 2'b00;
  - y= {4{A}}// y=4'b1111
- Replicações e concatenações podem ser aninhadas.
  - Exemplo:
  - y={4{A} , 2{B} }// y= 8'b11110000

Prática 4 - Dataflow