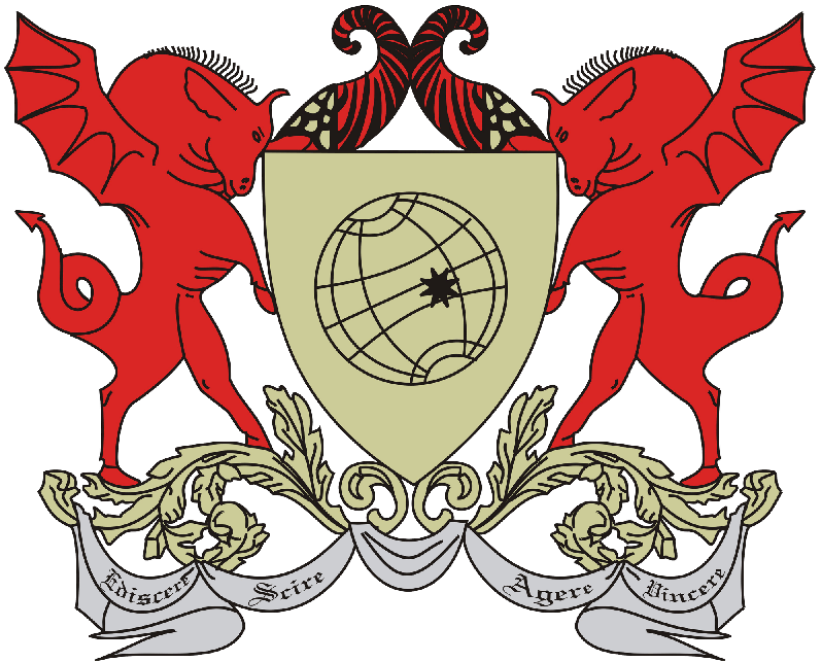
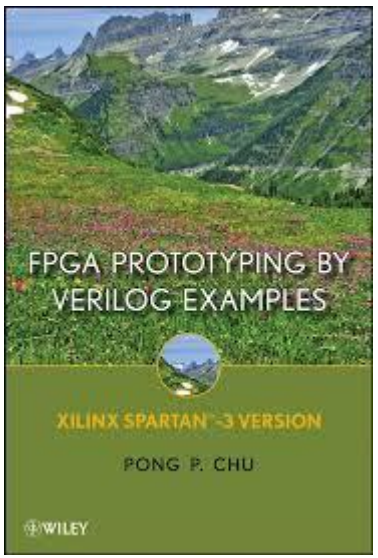


CCF 251 – Introdução aos Sistemas Lógicos

Aula 04 – Verilog (Parte 1)

Prof. José Augusto Nacif – jnacif@ufv.br

Referências Bibliográficas



- CHU, PONG P.. FPGA PROTOTYPING BY VERILOG EXAMPLES. 1.ed. United States of America: JOHN WILEY & SONS, 1959. 521p.

- TALA, Deepak Kumar, ASIC WORLD. Disponível em: <*http://www.asic-world.com/*>. Acesso em: 22 Setembro 2015.

Introdução

- Verilog é uma linguagem de descrição de hardware;
- Desenvolvida em 1980;
- Transferida para IEEE e definida no padrão 1364;
- Ratificada em 1995 e revisada em 2001;
- Se baseia na linguagem C;
- Tem como objetivo projetar modelos de circuito lógico digital;
- É possível implementar componentes com lógica combinacional e sequencial;

Módulo verilog

module [nome módulo] ([Interface - portas de entradas e saídas]);

...

[fios e registradores]

...

[módulos internos]

...

[atribuições e comportamentos]

endmodule

Tipos de dados

- **Sistema com 4 valores**

  Quatro valores básicos são usados na maioria dos tipos de dados.

  - **0:** Para lógica 0, ou condição falsa;
  - **1:** Para lógica 1, ou condição verdadeira;
  - **z:** Para o estado impedância alta;
  - **x:** Para valores não conhecidos;

Tipos de dados

- **Grupo de tipo de dados**

  Verilog tem dois grupos de tipos de dados: fio (wire) e variáveis (reg).

Tipos de dados

- **Grupo de tipo de dados**
  - **Fios (wire)**
    - Representa conexões físicas entre os componentes do hardware;
    - São utilizados nas saídas de atribuições contínuas (assign) e sinais de conexão entre os módulos;
    - Não armazenam valor;

Tipos de dados

- **Grupo de tipo de dados**
  - **Fios (wire)**
    - **Exemplo fios de 1-bit:**

      wire p0, p1;

    - **Exemplo grupo fios (barramento):**

      wire [7:0] data1, data2 ; // 8 bits

      wire [31:0] addr; / / 32 bits

      wire [0:7] revers_data; //index ascendentes devem ser evitados

Tipos de dados

- **Grupo de tipo de dados**
  - **Variáveis (reg)**
    - São usados em saídas de atribuições procedimentais (bloco initial ou always);
    - Armazenam valores;
    - Existem 5 tipos: reg, integer, real, time e realtime;
    - Apenas o tipo reg é sintetizável, os demais são utilizados apenas em simulação;

Tipos de dados

- **Grupo de tipo de dados**
  - **Variáveis (reg)**
    - **Exemplo reg de 1-bit:**

      reg x;

    - **Exemplo grupo reg:**

      reg [7:0] data1, data2 ; // 8 bits

      reg [31:0] cont; // 32 bits

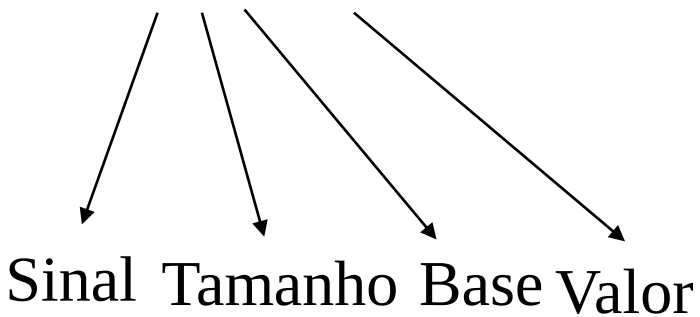
      reg [0:7] revers_data; //index ascendentes devem ser evitados

    - **Exemplo memória:**

      reg [3:0] mem [31:0]; // memória de 32 linhas x 4 bits por linha

Representação numérica

- Representações numéricas em verilog podem ser representadas de várias maneiras.
- Formato geral: [sinal] [tamanho]‘ [base][valor]
  - **Ex: - 5’b00001**



Representação numérica

- [valor]: Termo que especifica o valor do número correspondente a base;
- [base]: Termo que especifica a base do número.
  - b or B: binário
  - o or 0: octal
  - h or H: hexadecimal
  - d or D: decimal
- [tamanho]: Termo que especifica o tamanho do número em bits. Este termo é opcional;
- [sinal]: termo que especifica o sinal do número;

Representação numérica

- **Exemplos**

number	stored value	comment
5'b11010	11010	
5'b11_010	11010	_ ignored
5'o32	11010	
5'h1a	11010	
5'd26	11010	
5'b0	00000	0 extended
5'b1	00001	0 extended
5'bz	zzzzz	z extended
5'bx	xxxxx	x extended
5'bx01	xxx01	x extended
-5'b00001	11111	2's complement of 00001
'b11010	00000000000000000000000000011010	extended to 32 bits
'hee	00000000000000000000000011101110	extended to 32 bits
1	00000000000000000000000000000001	extended to 32 bits
-1	11111111111111111111111111111111	extended to 32 bits

Declaração portas

module [nome modulo]

([modo] [tipo de dado] [nome porta] ,

[modo] [tipo de dado] [nome porta] ,

...

[modo] [tipo de dados] [nome porta]

);

- [modo]: input, output, inout;
- [tipo de dados]: wire ou reg. Caso não seja inserido o tipo de dados da porta, por padrão a mesma será definida como um wire;
- [nome porta]: se refere ao nome da porta;

Declaração portas

```
module eq1
    ( input wire i0,
      input wire i1,
      output wire eq);
    .....
    .....
endmodule
```

ou

```
module eq1
    ( i0,  i1, eq);
   input i0;
   input i1;
   output eq;
   wire i0, i1;
   wire eq;
    .....
    .....
endmodule
```

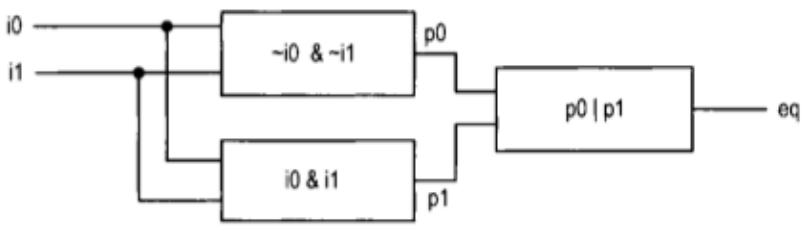
Corpo do programa

- Não é igual a linguagem C, pois as instruções não são executadas sequencialmente. Tal que, o corpo de um programa em verilog opera em paralelo e é executado de maneira concorrente;
- Pode ser escrito de três maneiras:
  - Atribuições contínuas (assign);
  - Instanciação de módulos;
  - Bloco initial e always;

Exemplo comparador 1-bit

input		output
*i*0	*i*1	*eq*
0	0	1
0	1	0
1	0	0
1	1	1

Tabela verdade



Esboço

eq = i0.i1 + i0'.i1'

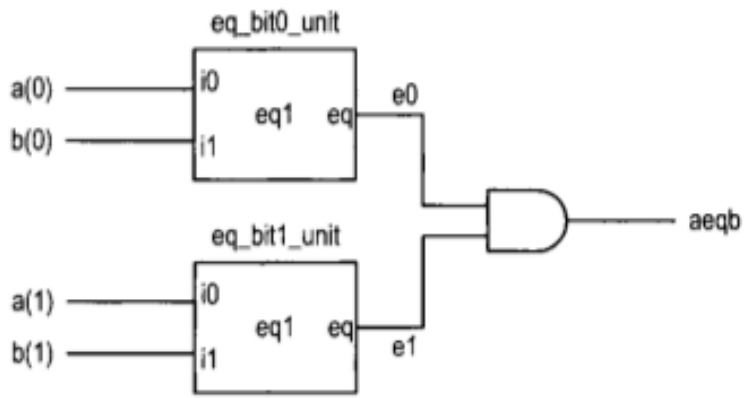
Soma de produtos

Exemplo comparador 1-bit

```
module eq1
(
  input wire i0, i1,
  output wire ep
 );  //Portas de I/O

      //Declaração sinais
      wire p0, p1;

     //Corpo
     //Termos soma de produtos
     assign eq = p0 | p1;
     assign p0 = ~i0 & ~i1;
     assign p1 = i0 & i1;
endmodule
```

Exemplo comparador 2-bit

```
module eq2
(
  input wire [1:0] a, b,
  output wire aeqb
 );  //Portas de I/O


     //Declaração sinais internos
     wire e0, e1;


    //Corpo
    //Instanciando comparador 1 bit duas vezes
    eq1 eq_bit0_unit (.i0(a[0]), .i1(b[0]), .eq(e[0]));
    eq1 eq_bit1_unit (.eq(e[1]), .i0(a[1]), .i1(b[0]));
    assign aeqb = e0 & e1;
endmodule
```



Esboço

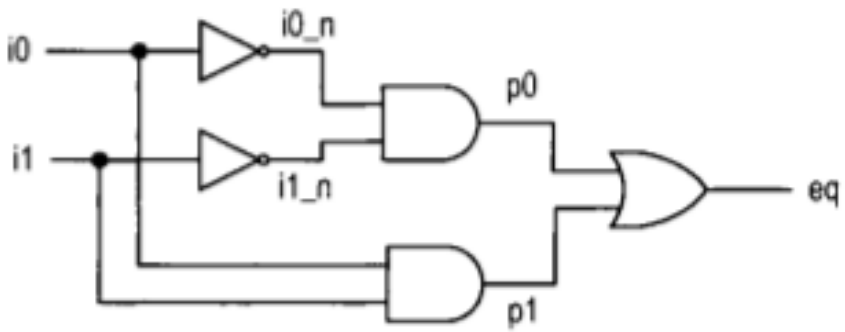
Primitivas

- Verilog inclui um conjunto de primitivas predefinidas que podem ser instanciadas como módulos. Estas primitivas correspondem a blocos de funções simples, tais como: NOT, AND, OR, NAND, NOR, XOR e XNOR.

Primitivas

- Exemplo comparador 1-bit.

```
module eq1_primitive
(
  input wire i0, i1,
  output wire eq
 );  //Portas de I/O


      //Declaração sinais internos
      wire i0_n, i1_n, p0, p1;


     //Instanciação portas primitivas
     not  unit1 (i0_n, i0);
     not  unit2 (i1_n, i1);
     and unit3 (p0, i0_n, i1_n);
     and unit4 (p1, i0, i1);
     or    unit5 (eq, p0, p1);
endmodule
```



Circuito

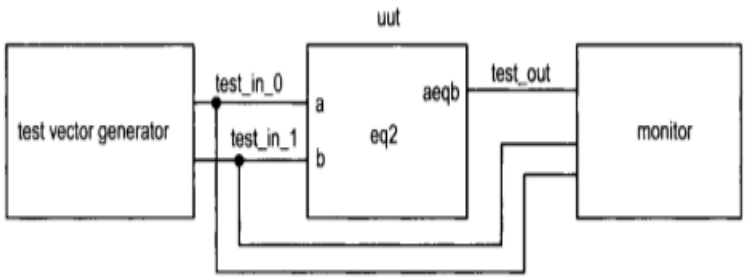
Testbench

- Depois dos módulos construídos o próximo passo é simular no computador via formas de onda e verificar se as entradas e saídas dos módulos estão chegando os valores corretos com base no testbench criado.

Testbench

- Exemplo comparador 2-bit.

```
module eq2_testbench;
        //Declaração sinais internos
        reg  [1:0] test_in0, test_in1;
        wire test_out;

        //Instanciando circuito eq2
        eq2 uut (.a(test_in0), .b(test_in1), aeqb(test_out));

        initial  begin
          #1  test_in0 = 2'b00;
          #0  test_in1 = 2'b00;

          #1  test_in0 = 2'b01;
          #0  test_in1 = 2'b00;

          #1  test_in0 = 2'b01;
          #0  test_in1 = 2'b11;

          #1  test_in0 = 2'b10;
          #0  test_in1 = 2'b11;
          #10 $stop; //10s para a simulação
        end
endmodule
```



Esboço

Lógica combinacional

- A descrição de um projeto em nível combinacional é feita em termos de procedimentos de dois tipos:
  - initial
  - always

Estrutura

initial ou always

Especifica evento sob controle (always)

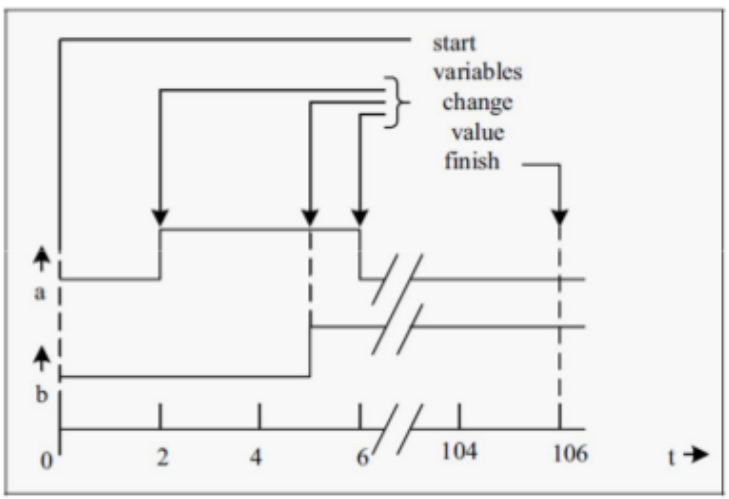
Especifica o evento que ativa a execução do bloco (always)

```
tipo_de_bloco @(lista_de_sensibilidade)
  begin; nome_do_bloco
     declaração de variáveis locais;
     assinalamentos procedimentais;
  end
```

Bloco initial

- São executados apenas uma vez;
- Lado esquerdo deve ser uma variável;
- Exemplo:

```
reg a, b;

Initial  begin
            a = 1'b0;
            b = 1'b0;
        #2  a = 1'b1;
        #3  b = 1'b1;
        #1  a = 1'b0;
        #100 $stop
end
```



Bloco initial

- Instruções de impressão na tela;
  - $display: imprime apenas uma única vez;
  - $monitor: monitora variáveis, ou seja, imprimirá qualquer alteração em uma das variáveis;

```
module initial_begin_end();
    reg clk,reset,enable,data;

    initial begin
        $display("Saídas \n");
        $monitor("%g clk=%b reset=%b enable=%b data=%b",
                        $time, clk, reset, enable, data);
        #1  clk = 0;
        #10 reset = 0;
        #5  enable = 0;
        #3  data = 0;
        #20 $finish;
    end
endmodule
```

```
0 clk=x reset=x enable=x data=x
1 clk=0 reset=x enable=x data=x
11 clk=0 reset=0 enable=x data=x
16 clk=0 reset=0 enable=0 data=x
19 clk=0 reset=0 enable=0 data=0
```

Fonte: ASIC WORLD

Bloco always

- São executados repetidamente e infinitamente;
- É necessário especificar um conjunto de condições que iniciará a execução do bloco;
- Exemplos
- Borda de descida
  - always@(negedge clk)
- Borda de subida
  - always@(posedge clk)
- Tanto na borda de descida quanto na borda de subida
  - always@clk
- Qualquer mudança nas variáveis prt ou clk
  - always@(prt or clk)
- Borda de subida de clk1 ou borda de descida de clk2
  - always@(posedge clk1 or negedge clk2)
- Qualquer alteração nas variáveis dentro do bloco
  - always@*

Operadores

Type of operation	Operator symbol	Description	Number of operands
Arithmetic	+	addition	2
	-	subtraction	2
	*	multiplication	2
	/	division	2
	%	modulus	2
	**	exponentiation	2
Shift	>>	logical right shift	2
	<<	logical left shift	2
	>>>	arithmetic right shift	2
	<<<	logical left shift	2
Relational	>	greater than	2
	<	less than	2
	>=	greater than or equal to	2
	<=	less than or equal to	2
Equality	==	equality	2
	!=	inequality	2
	===	case equality	2
	!==	case inequality	2
Bitwise	~	bitwise negation	1
	&	bitwise and	2
	|	bitwise or	2
	^	bitwise xor	2

Type of operation	Operator symbol	Description	Number of operands
Reduction	&	reduction and	1
	|	reduction or	1
	^	reduction xor	1
Logical	!	logical negation	1
	&&	logical and	2
	||	logical or	2
Concatenation	{ }	concatenation	any
	{ { } }	replication	any
Conditional	? :	conditional	3

Atribuição procedimental

- **Bloqueante:** Atribuições bloqueantes são executados na ordem da codificação das instruções, de maneira sequencial. Símbolo: "=".
  - **Exemplo a = b;**
- **Não bloqueante:** Atribuições não bloqueantes são executados de maneira paralela. Símbolo: "<=".
  - **Exemplo a <= b;**

Atribuição procedural

- **Exemplo**

```
module block_nonblock();
reg a, b, c, d , e, f ;

// Atribuição bloqueante
initial begin
  a = #10 1'b1; // É atribuído valor 1 em a no tempo 10
  b = #20 1'b0; // É atribuído valor 0 em b no tempo 30
  c = #40 1'b1; // É atribuído valor 1 em c no tempo 70
end

// Atribuição não bloqueante
initial begin
  d <= #10 1'b1; //É atribuído o valor 1 no tempo 10
  e <= #20 1'b0; //É atribuído o valor 0 no tempo 20
  f  <= #40 1'b1; //É atribuído o valor 1 no tempo 40
end
endmodule
```

FONTE: ASIC WORLD

Instrução IF

- **Sintaxe**

```
if [expressão boolean]  begin
   [ instrução 1 ];
   [ instrução 2 ];
   ...
end
else begin
   [ instrução 3 ];
   [ instrução 4 ];
   ...
end
```

Instrução IF

- **Exemplo priority encoder**

```
module prio_encoder_i f   ( input wire [4:1] r ,  output reg [2:0] y);


always @*  begin
   if (r[4] = = 1'b1)
      y = 3'b100;
   else if (r[3] = = 1'b1)
      y = 3'b011;
   else if (r[2]==1'b1)
     y = 3'b010;
   else if (r[1]==1'b1)
     y = 3'b001;
    else
     y = 3'b000;
end
endmodule
```

input r	output pcode
1 – – –	100
0 1 – –	011
0 0 1 –	010
0 0 0 1	001
0 0 0 0	000

Esboço

Instrução CASE

- **Sintaxe**

```
case [expressão case]  begin
    [ item ]:
          begin
                [ instrução 1 ];
                [ instrução 2 ];
          end
     [ item ]:
          begin
                [ instrução 3 ];
                [ instrução 4  ];
          end
        .....
       default:
          begin
                [ instrução n ];
                [ instrução n  ];
          end
   endcase
```

Instrução CASE

- **Exemplo priority encoder**

```
module prio_encoder_case  ( input wire [4:1] r ,  output reg [2:0] y);
    always @*  begin
        case(r) begin
                4'b1000, 4'b1001, 4'b1010, 4'b1011,
                4'b1100, 4'b1101, 4'b1110, 4'b1111:
                        y = 3'b100;
                4'b0100, 4'b0101, 4'b0110 , 4'b0111:
                        y = 3'b011;
                4'b0010, 4'b0011:
                        y = 3'b010;
                4'b0001:
                        y = 3'b001;
                4'b0000:
                        y = 3'b000;
            endcase
        end
endmodule
```

input r	output pcode
1 – – –	100
0 1 – –	011
0 0 1 –	010
0 0 0 1	001
0 0 0 0	000

Esboço

Parameter e Constante

- **localparam:** é utilizado para se criar constantes dentro do módulo. Valor não pode ser alterado.

```
module adder_carry_hard_lit (
  input wire [3:0] a, b,
  output wire [3:0] cout //carry-out
 );  //Portas de I/O

      //Declaração constante
      localparam N   = 4,
                        N1 = N-1;

       //Declaração sinais
       wire [N:0] sum_ext;

       //Corpo
       assign sum_ext =  {1'b0, a} + {1'b0, b};
       assign sum = sum_ext[N1:0];
       assign cout = sum_ext[N];
endmodule
```

Parameter e Constante

- **parameter:** Também é utilizado para se criar constantes, mas seu valor pode ser alterado na instanciação do módulo.

```
module adder_carry_insta (
  input wire [3:0] a, b,
  output wire [3:0] cout //carry-out
 );  //Portas de I/O

    //Instaciação  8 - bit adder
    adder-carry-param # ( . N(8))
    unit1   ( .a(a8), .b(b8), .sum(sum8), . cout (c8));

    //Instanciação 4 – bit adder
    adder-carry-param
    unit2    (.a(a4), .b(b4), .sum(sum4), . cout(c4));

endmodule
```

```
module adder_carry_param
#(parameter N=4)
(
  input wire [3:0] a, b,
  output wire [3:0] cout //carry-out
 );  //Portas de I/O

      //Declaração constante
      localparam   N1 = N-1;

       //Declaração sinais
       wire [N:0] sum_ext;

      //Corpo
      assign sum_ext =  {1'b0, a} + {1'b0, b};
      assign sum = sum_ext[N1:0];
      assign cout = sum_ext[N];
endmodule
```

Softwares

- Os softwares necessários para se trabalhar com o desenvolvimento de códigos verilog e simulação em formas de onda são:

  - **Icarus verilog:** Compilador que compila códigos feitos na linguagem verilog;
  - **GtkWave:** Programa que tem como objetivo visualizar as formas de onda apartir de um arquivo gerado pelo compilador Icarus verilog;

Instalação softwares

- Sistema Operacional Windows:
  - **Icarus verilog**

Link: http://bleyer.org/icarus/

  - **GtkWave**

Link: http://www.dspia.com/gtkwave.html

Instalação softwares

- Sistema Operacional Linux:
  - **apt-get install verilog gtkwave**

Compilando com Icarus verilog

- Acessar o prompt de comando (MS-DOS ou Terminal)
- Entrar na pasta do projeto (comando cd)
  - Ex: cd /home/usuario/porta_and
- Executar o comando: **iverilog arquivo.v -o arquivo.vvp –Wall**
  - Ex: iverilog porta_and.v -o porta_and.vvp -Wall
- Atributos do comando acima
  - -o: Gerar o objeto;
  - -Wall: Mostrar avisos;
  - -I: Para compilar módulos verilog separados dentro de uma pasta;

OBS: Se não ocorrer nenhuma mensagem de erro será gerado um arquivo com extensão (.vvp)

Compilando com Icarus verilog

- Com o arquivo.vvp gerado, o próximo passo é criar o arquivo .vcd referente as formas de onda;

- Executar o comando: **vvp arquivo.vvp**
  - **Ex: vvp porta_and.vvp**

OBS: Se não ocorrer nenhuma mensagem de erro será gerado um arquivo com extensão (.vcd)

Visualizando formas de onda GtkWave

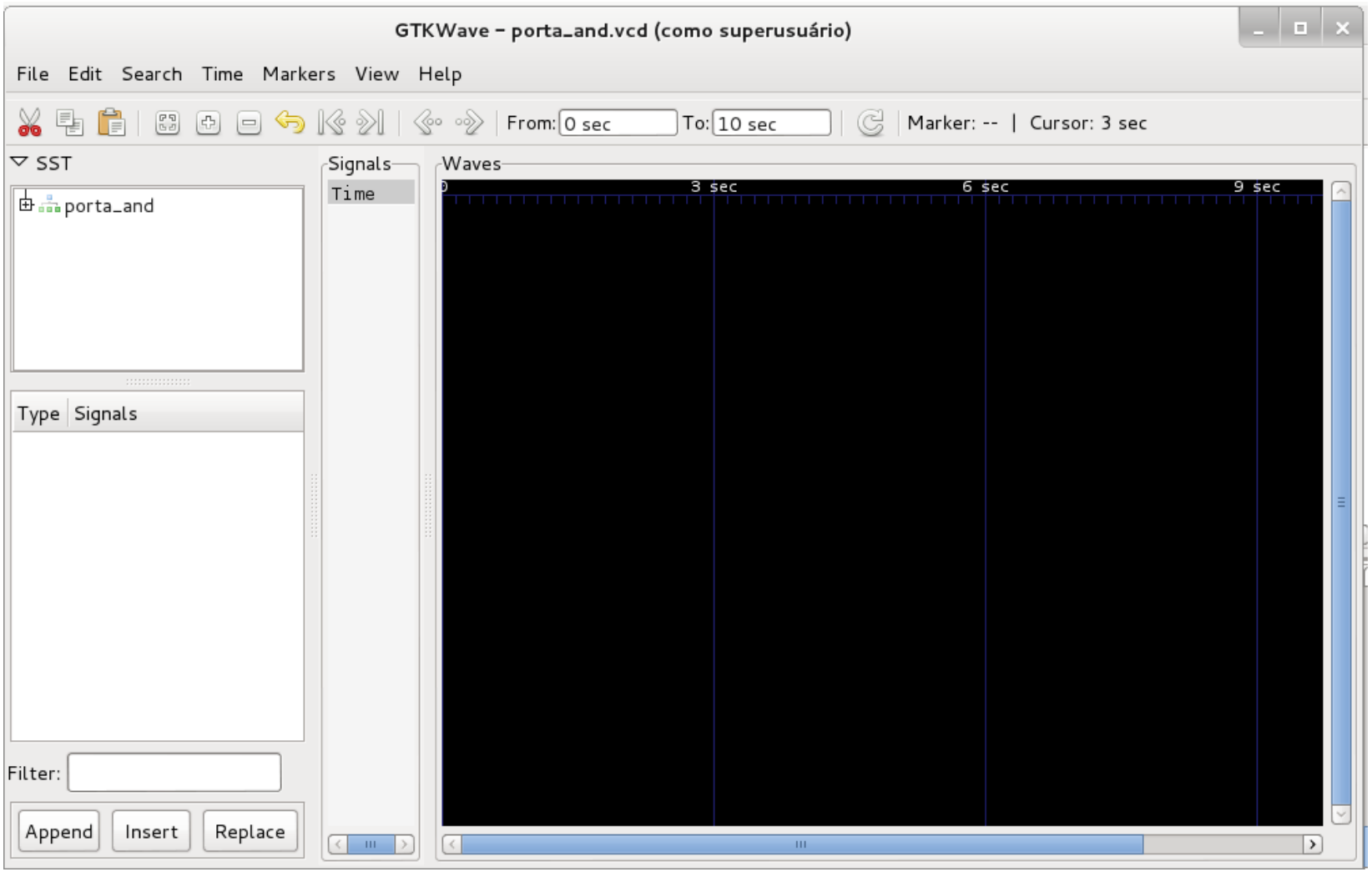
- Com o arquivo.vcd gerado, o próximo passo é abrir este arquivo com o simulador GtkWave;
- Executar o comando: **gtkwave arquivo.vcd**
  - **Ex: gtkwave porta_and.vcd**

Exemplo verilog porta AND

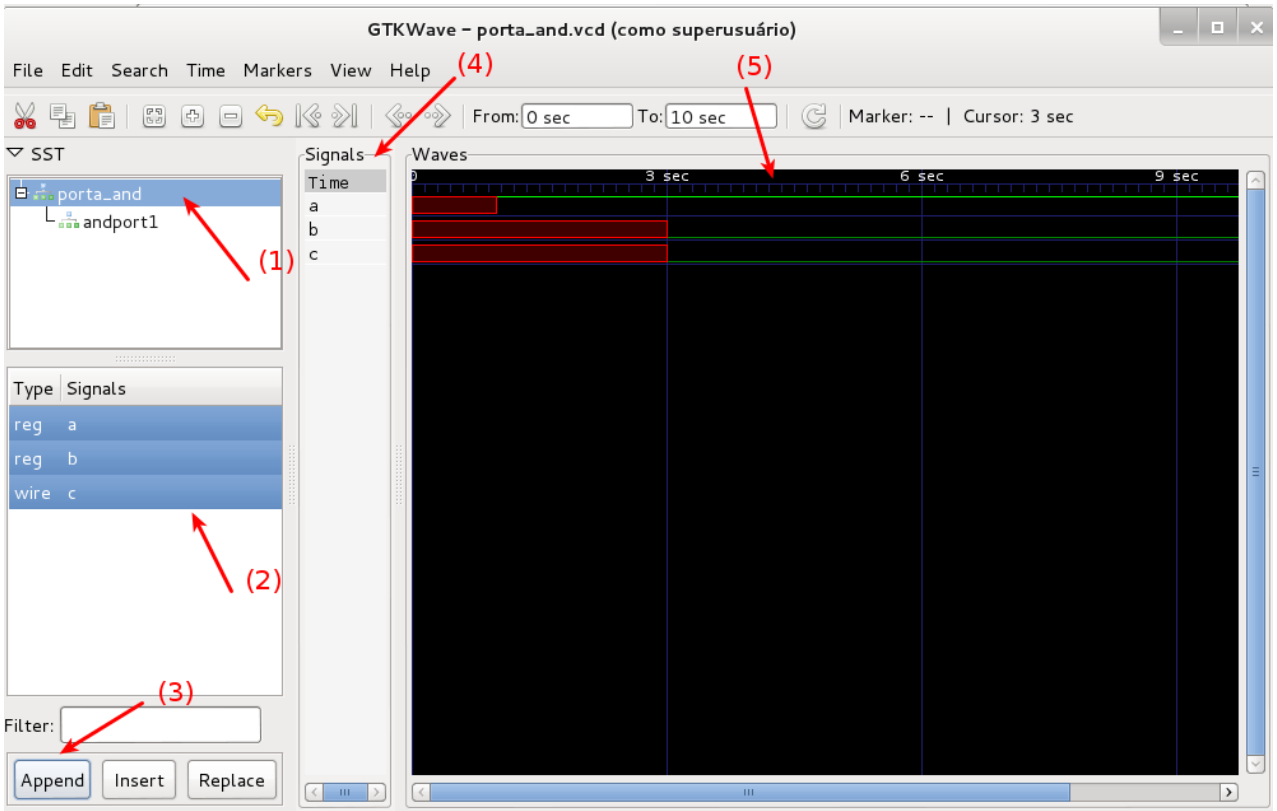
porta_and.v

```
1  // Módulo de simulação (Verilog de simulação)
2  module porta_and();
3    // Registradores para gerar sinais de entrada
4    reg a, b;
5    // Fio conectado ao sinal de saída
6    wire c;
7
8    // Instanciação do módulo a ser simulado
9    and_port andport1 (a,b,c);
10
11   // Início da simulação
12   initial begin
13     // Geração do arquivo com o resultado da simulação
14     $dumpfile("porta_and.vcd");
15     $dumpvars;
16     // Alteração dos sinais de entrada
17     // #1 significa um intervalo de tempo
18     #1 a = 1'b1;
19     #2 b = 1'b0;
20   end
21
22   initial begin
23     // A simulação é finalizada no intervalo de tempo #1000
24     #10 $finish;
25   end
26
27 endmodule
28
29
30 // Módulo a ser simulado (apenas Verilog de síntese)
31 module and_port (a,b,c);
32   input a, b;
33   output c;
34   assign c = a & b;
35 endmodule
```

Exemplo verilog porta AND GtkWave



Exemplo verilog porta AND GtkWave



Para iniciar a simulação é necessário:

(1) Clicar no módulo port_and ;

(2) Selecionar os registradores e a(s) saída(s);

(3) Clicar na opção **Append** para adicionar os registradores e a(s) saída(s) no Signals Time;

(4) Tempo sinais;

(5) Formas de Onda;

Exercício

- **Display de 7 segmentos**