Tutorial
Projeto: ULA de 1 bit

Prof. Rogério Aparecido Gonçalves
rogerioag@utfpr.edu.br

31 de dezembro de 2011

1.1 Introdução

1.2 GHDL Code Gen (gcg)

O projeto gcg tem como objetivo a criação de templates ou modelos para facilitar a criação, execução de projetos em VHDL na ferramenta GHDL. O código do gcg está disponível para download em http://code.google.com/p/gcg/. Dentro do diretório do gcg/src tem uma diretório de templates (entidade e testbench), um arquivo README.txt (como os comandos básicos), o Makefile (que faz todo o trabalho). Os comandos de utilização seguem a sintaxe apresentada na Tabela 1.1.

Tabela 1.1: Comandos

Ação	Comando
Criar projeto e arquivos iniciais	make new PROJECT=nomeDoProjeto ARCH=tipoArquitetura IN=porta1,porta2,portaN OUT=porta1,porta2,portaN
Compilar	make compile TESTBENCH=nomeDoProjeto_tb
Executar	make run TESTBENCH=nomeDoProjeto_tb
Visualizar	make view TESTBENCH=nomeDoProjeto_tb
Tudo	make all TESTBENCH=nomeDoProjeto_tb
Apagar diretório de simulação	make clean

Makefile do modelo de projeto foi alterado para permitir a criação dos componentes com o mesmo comando, no mesmo projeto. A próxima seção exemplifica essa ideia de termos componentes formados por subcomponentes, e a forma de criarmos do componente mais simples para o mais complexo.

1.3 Projeto: ULA de 1 bit

Tomemos então como exemplo o nosso projeto de uma ULA (Unidade Lógica e Aritmética) de 1 bit. A Figura 1.1 apresenta a ULA em um nível 0, esta é a visão da entidade de teste, que chamamos de testbench, pois temos o componente ULA e as variáveis/sinais (T_A, T_B, T_Cin, T_F2, T_F1, T_F0, T_S e T_Cout) que pertencerão a essa entidade de teste. Por meio dessas variáveis que os bits dos casos de teste irão ser injetados para que os resultados sejam gerados pela entidade ULA, possibilitando a comparação com o resultado esperado em cada caso de teste.

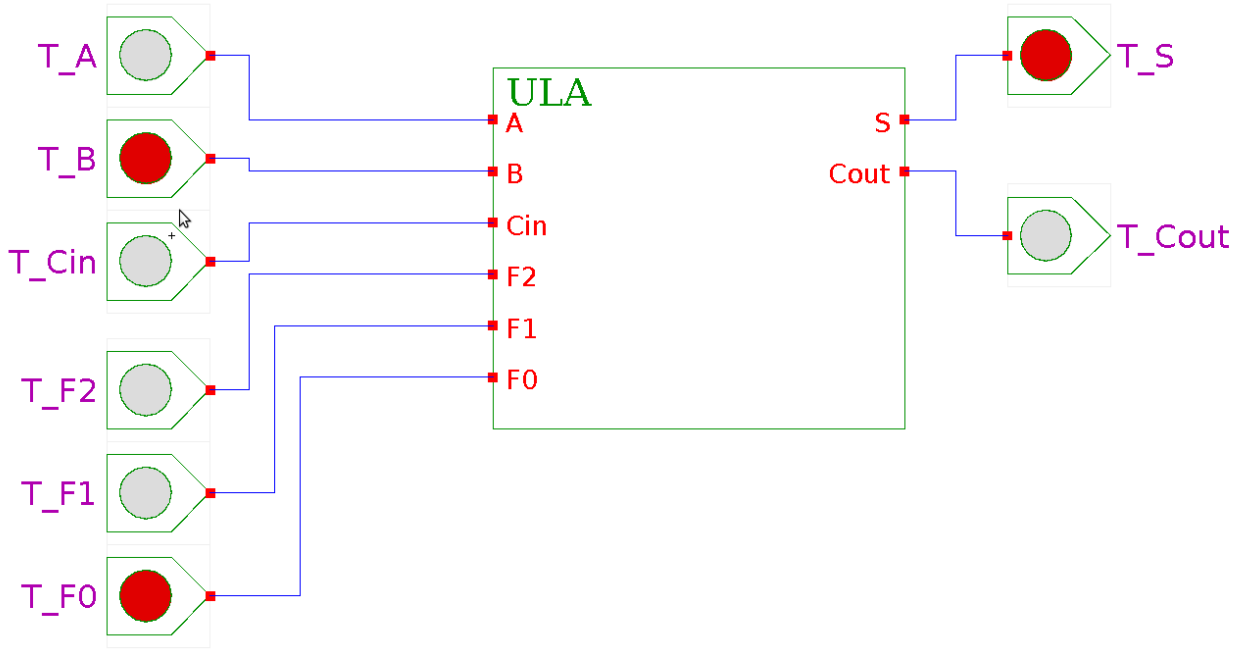

Figura 1.1: Visão da ULA em um nível 0

Em um nível 1, podemos visualizar o detalhamento da ULA, expondo seus componentes, conforme podemos visualizar na Figura 1.2.

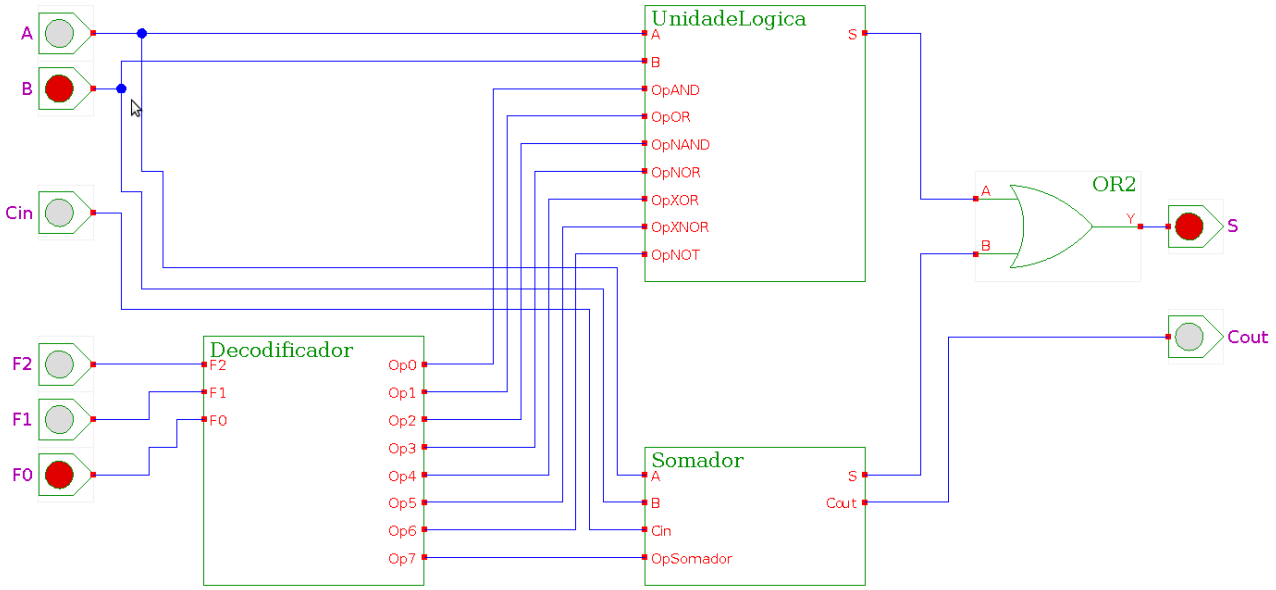

Figura 1.2: Visão da ULA em um nível 1, exposição dos componentes principais.

1.3.1 Análise e Levantamento de componentes

Analisando nossa entidade ULA, conforme Figura 1.2, podemos verificar que a ULA tem um Somador, um Decodificador, uma UnidadeLogica e uma xor2. O circuito da entidade Somador é apresentado na Figura 1.3, podemos ver que realiza a soma das variáveis de entrada A, B e Cin, sendo que Cin é o *Carry* de entrada, o ”vai-um”de uma possível coluna anterior.

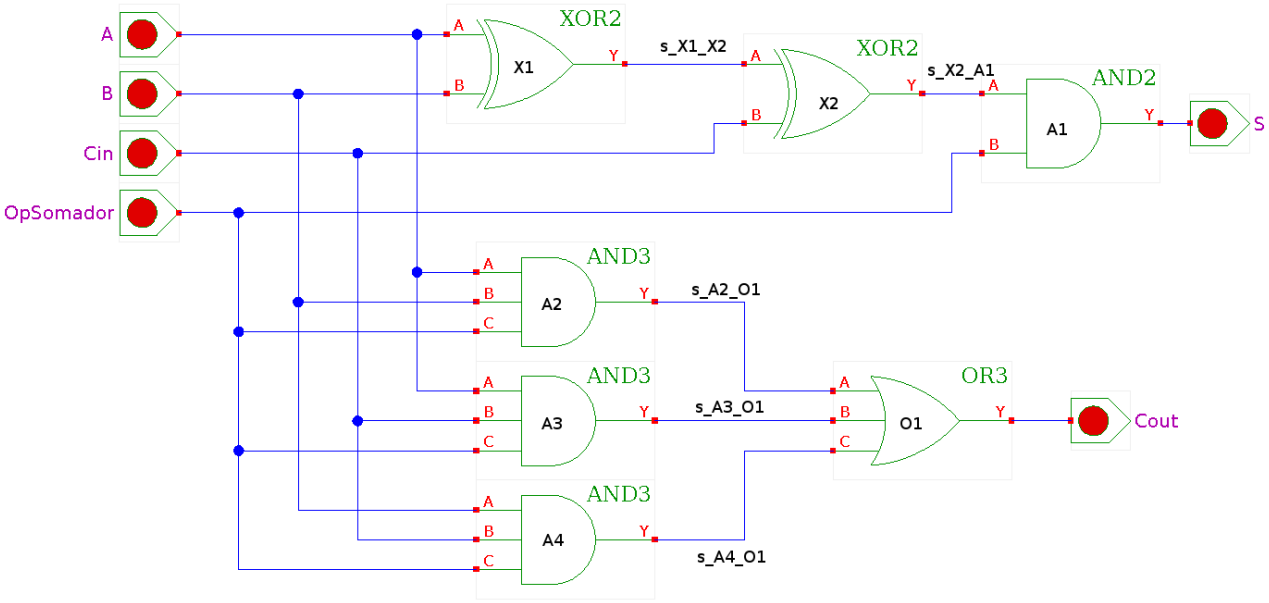

Figura 1.3: Circuito Somador.

O circuito Decodificador é uma circuito para decoficar a operação codificada pela escolha das entradas F_2, F_1 e F_0. O detalhamento da entidade Decodificador é apresentado na Figura 1.4.

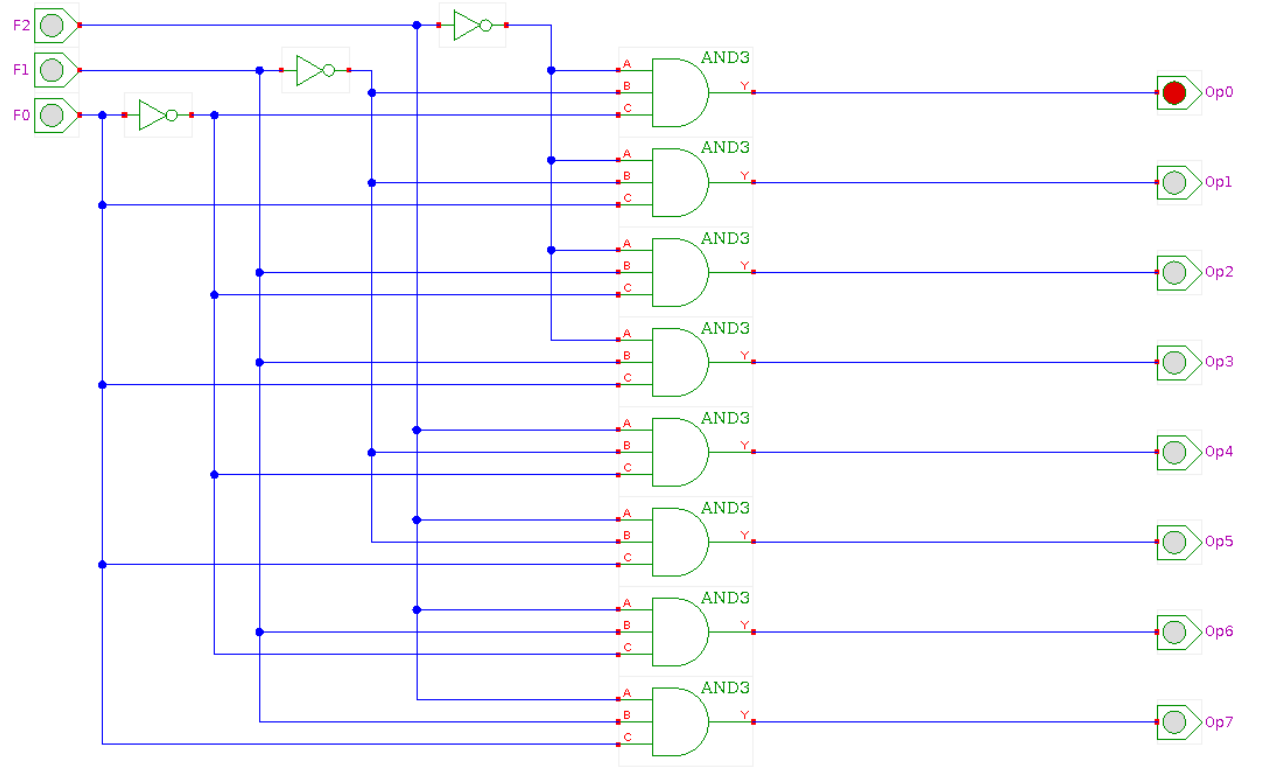

Figura 1.4: Circuito Decodificador.

As operações suportadas pelo decodificador, consequentemente que são realizadas pela ULA, estão listadas na Tabela 1.2.

Tabela 1.2: Operações suportadas

[F_2][F_1][F_0]	Operação	Descrição
000	Op0	Operação AND, realizada pela Unidade Lógica.
001	Op1	Operação OR, realizada pela Unidade Lógica.
010	Op2	Operação NAND, realizada pela Unidade Lógica.
011	Op3	Operação NOR, realizada pela Unidade Lógica.
100	Op4	Operação XOR, realizada pela Unidade Lógica.
101	Op5	Operação XNOR, realizada pela Unidade Lógica.
110	Op6	Operação NOT, realizada pela Unidade Lógica.
111	Op7	Operação SOMA, realizada pela Somador.

As operações suportadas pelo Decodificador de Op0 a Op6 irão selecionar as operações que a Unidade Lógica realiza de OpAND a OpNOT, conforme pode ser visto na Figura 1.5. O sinal Op7 será ligado ao Somador na entrada OpSomador, entrada que habilita a operação do Somador.

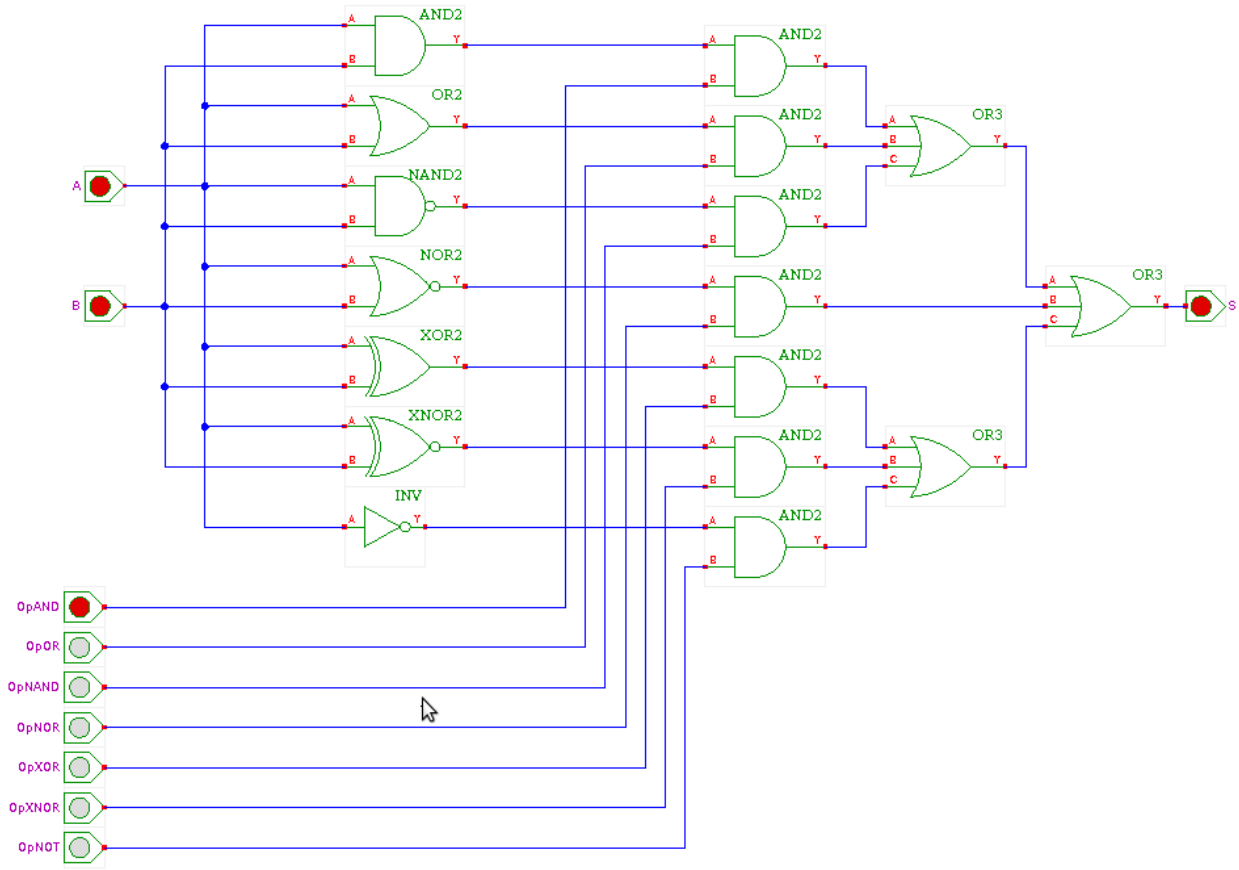

Figura 1.5: Circuito UnidadeLogica.

Feita essa análise e detalhamento de cada componente, verificamos os subcomponentes de cada componente principal da ULA. Assim, podemos resumir a lista de componentes e suas dependências conforme apresentado na Tabela 1.3.

Tabela 1.3: Componentes principais e suas dependências

Componente	Dependências
ULA	Somador, Decodificador, UnidadeLogica e or2
Somador	xor2, and2, and3 e or3
Decodificador	and3 e inversor
UnidadeLogica	or3, and2, or2, nand2, nor2, xor2, xnor2 e inversor

1.4 Criação dos Componentes com o gcg

Para satisfazer as dependências da entidade Somador precisamos então criar os componentes básicos, and2, and3, or3 e xor2. O Código 1.1 apresenta os comandos do Makefile para obter esse resultado. Note que a arquitetura para esses elementos básicos, neste caso portas lógicas, foi definida como lógica, por meio da variável ARCH.

Código 1.1: Comandos para a criação das entidades básicas

```
1  make new PROJECT=and2 ARCH=logica IN=a,b OUT=y
2  make new PROJECT=and3 ARCH=logica IN=a,b,c OUT=y
3  make new PROJECT=or3 ARCH=logica IN=a,b,c OUT=y
4  make new PROJECT=xor2 ARCH=logica IN=a,b OUT=y
```

Se verificarmos na estrutura do projeto, foram criados os arquivos das entidades no diretório src e dos testes no diretório testbench. Esse componentes se quisermos fazer o testbench, tudo bem, mas por serem simples portas não precisaria, logo os arquivos dos testbenchs dessas unidades básicas poderiam ser descartados. O Código 1.2 mostra como o código VHDL para a entidade and2 foi criado pelo make, sendo necessário somente colocarmos o tipo das variáveis e a expressão lógica (y ¡= a and b) que gera o resultado.

Código 1.2: Código VHDL da entidade and2

```
1 -- Projeto gerado via script.
2 -- Data: Qua,20/07/2011-13:51:40
3 -- Autor: rogerio
4 -- Comentario: Descri  o da Entidade: and2.
5
6 library ieee;
7 use ieee.std_logic_1164.all;
8 use ieee.std_logic_unsigned.all;
9
10 entity and2 is
11     port(a,b:in std_logic; y:out std_logic);
12 end and2;
13
14 architecture estrutural of and2 is
15 begin
16     y <= a and b;
17 end estrutural;
```

No diretorio testbench foi criado o arquivo and2_tb.vhd que é o testbench para a entidade and2, conforme podemos ver no Código 1.3, novamente alteramos type para std_logic e criamos os casos de teste.

Código 1.3: Código VHDL do testebench para entidade and2

```
1 -- Testebench gerado via script.
2 -- Data: Qua,20/07/2011-14:18:43
```

```
3 -- Autor: rogerio
4 -- Comentario: Teste da entidade and2.
5
6
7 library ieee;
8 use ieee.std_logic_1164.all;
9
10 entity and2_tb is
11 end and2_tb;
12
13 architecture logica of and2_tb is
14   --  Declara  o do componente.
15   component and2
16         port (a,b: in std_logic; y: out std_logic);
17   end component;
18   --  Especifica qual a entidade est  vinculada com o componente.
19   for and2_0: and2 use entity work.and2;
20       signal s_t_a, s_t_b, s_t_y: std_logic;
21   begin
22     --  Instancia  o do Componente.
23         --  port map (<<p_in_1>> => <<s_t_in_1>>)
24       and2_0: and2 port map (a=>s_t_a,b=>s_t_b,y=>s_t_y);
25
26     --  Processo que faz o trabalho.
27     process
28                 -- Um registro    criado com as entradas e sa das da
                       entidade.
29                 -- (<<entrada1>>, <<entradaN>>, <<saida1>>, <<saidaN
                       >>)
30                 type pattern_type is record
31                     -- entradas.
32                     vi_a,vi_b: std_logic;
33                     -- sa das.
34                     vo_y: std_logic;
35                 end record;
36
37                 --  Os padr es de entrada s o aplicados (injetados)
                       s  entradas.
38                 type pattern_array is array (natural range <>) of
                      pattern_type;
39                 constant patterns : pattern_array :=
40                     (
41                         ('0', '0', '0'),
42                         ('0', '1', '0'),
43                         ('1', '0', '0'),
44                         ('1', '1', '1')
45                     );
46                 begin
47                     --  Checagem de padr es.
48                     for i in patterns'range loop
49                         --  Injeta as entradas.
50                         s_t_a <= patterns(i).vi_a;
51                         s_t_b <= patterns(i).vi_b;
```

```
52
53                          --  Aguarda os resultados.
54                          wait for 1 ns;
55                          --  Checa o resultado com a sa da esperada no
                                padr o.
56                          assert s_t_y = patterns(i).vo_y report "Valor
                               de s_t_y n o confere com o resultado
                               esperado." severity error;
57
58                      end loop;
59                      assert false report "Fim do teste." severity note;
60                      --  Wait forever; Isto finaliza a simula  o.
61                      wait;
62          end process;
63  end logica;
```

Executando o comando make apresentado no Código 1.4, se tudo estiver correto, a entidade and2 será analisada e compilada e o teste será executado e no final será apresentado a tela do gtkwave com o diagrama de tempo (forma de onda).

Código 1.4: Comando para executar o testbench da entidade and2.

```
1  make all TESTBENCH=and2_tb
```

O diagrama do resultado pode ser visualizado na Figura 1.6.

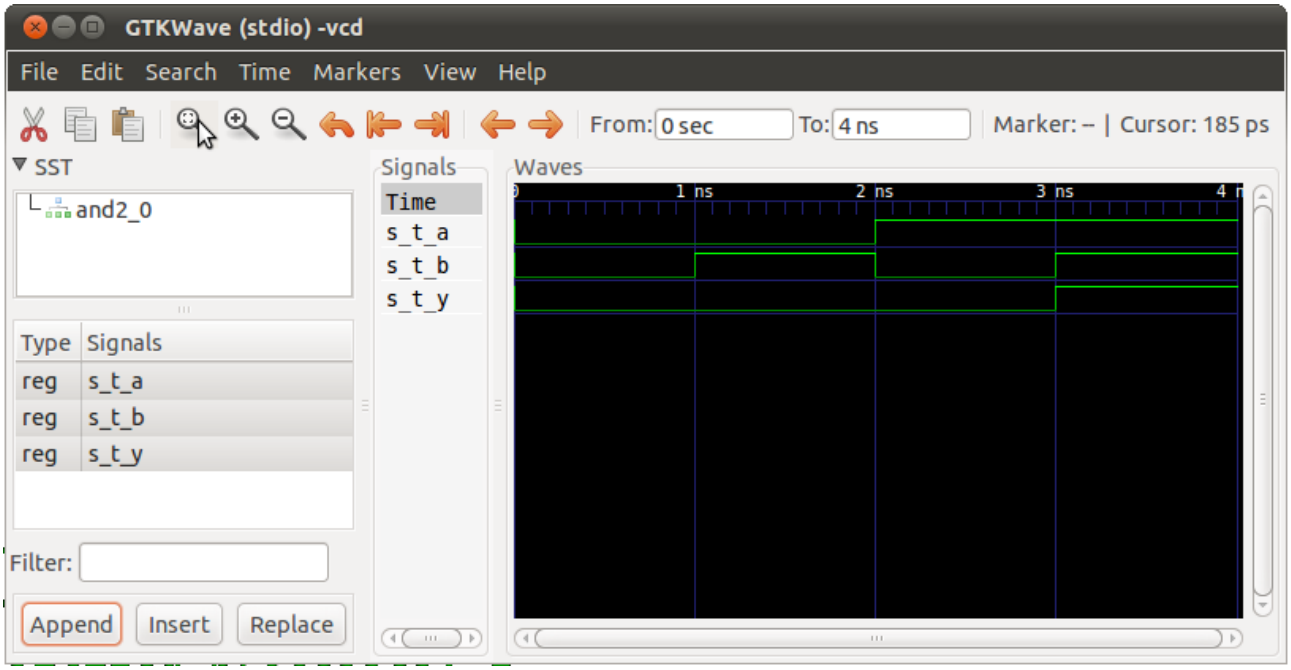

Figura 1.6: Diagrama de Tempo do teste da entidade and2.

O Código 1.5 mostra o código VHDL para a entidade and3 que foi criado pelo comando make, da mesma forma sendo necessário somente colocarmos o tipo das variáveis e a expressão lógica (y ¡= a and b and c) para que o resultado seja gerado corretamente.

Código 1.5: Código VHDL da entidade and3

```
1 -- Projeto gerado via script.
2 -- Data: Qua,20/07/2011-13:51:40
```

```
3 -- Autor: rogerio
4 -- Comentario: Descri  o da Entidade: and3.
5
6 library ieee;
7 use ieee.std_logic_1164.all;
8 use ieee.std_logic_unsigned.all;
9
10 entity and3 is
11     port (a,b,c: in std_logic; y: out std_logic);
12 end and3;
13
14 architecture logica of and3 is
15 begin
16 -- Comandos.
17     y <= a and b and c;
18 end logica;
```

No diretorio testbench também foi criado o arquivo and3_tb.vhd que é o testbench para a entidade and3, conforme podemos ver no Código 1.6, novamente alteramos ¡¡type¿¿ para std_logic e criamos os casos de teste.

Código 1.6: Código VHDL do testebench para entidade and3

```
1 -- Testebench gerado via script.
2 -- Data: Qua,20/07/2011-13:51:40
3 -- Autor: rogerio
4 -- Comentario: Teste da entidade and3.
5
6 library ieee;
7 use ieee.std_logic_1164.all;
8 use ieee.std_logic_unsigned.all;
9
10 entity and3_tb is
11 end and3_tb;
12
13 architecture logica of and3_tb is
14   --  Declara  o do componente.
15   component and3
16         port (a,b,c: in std_logic; y: out std_logic);
17   end component;
18   --  Especifica qual a entidade est  vinculada com o componente.
19   for and3_0: and3 use entity work.and3;
20       signal s_t_a, s_t_b, s_t_c, s_t_y: std_logic;
21   begin
22     --  Instancia  o do Componente.
23         --  port map (<<p_in_1>> => <<s_t_in_1>>)
24       and3_0: and3 port map (a=>s_t_a,b=>s_t_b,c=>s_t_c,y=>s_t_y);
25
26     --  Processo que faz o trabalho.
27     process
28                 -- Um registro    criado com as entradas e sa das da
                       entidade.
29                 -- (<<entrada1>>, <<entradaN>>, <<saida1>>, <<saidaN
                       >>)
30                 type pattern_type is record
31                     -- entradas.
```

```
32                      vi_a, vi_b, vi_c: std_logic;
33                      -- sa das.
34                      vo_y: std_logic;
35                  end record;
36
37                  --  Os padr es de entrada s o aplicados (injetados)
                        s  entradas.
38                  type pattern_array is array (natural range <>) of
                       pattern_type;
39                  constant patterns : pattern_array :=
40                      (
41                          ('0', '0', '0', '0'),
42                          ('0', '0', '1', '0'),
43                          ('0', '1', '0', '0'),
44                          ('0', '1', '1', '0'),
45                          ('1', '0', '0', '0'),
46                          ('1', '0', '1', '0'),
47                          ('1', '1', '0', '0'),
48                          ('1', '1', '1', '1')
49                      );
50                  begin
51                      --  Checagem de padr es.
52                      for i in patterns'range loop
53                          --  Injeta as entradas.
54                          s_t_a <= patterns(i).vi_a;
55                          s_t_b <= patterns(i).vi_b;
56                          s_t_c <= patterns(i).vi_c;
57
58                          --  Aguarda os resultados.
59                          wait for 1 ns;
60                          --  Checa o resultado com a sa da esperada no
                                padr o.
61                          assert s_t_y = patterns(i).vo_y report "Valor
                               de s_t_y n  o  confere com o resultado
                               esperado." severity error;
62
63                      end loop;
64                      assert false report "Fim do teste." severity note;
65                      --  Wait forever; Isto finaliza a simula   o.
66                      wait;
67          end process;
68  end logica;
```

Executando o comando make apresentado no Código 1.7, se tudo estiver correto, a entidade and3 será analisada e compilada e o teste será executado e no final será apresentado a tela do gtkwave com o diagrama de tempo (forma de onda).

Código 1.7: Comando para executar o testbench da entidade and3.

```
1  make all TESTBENCH=and3_tb
```

O diagrama do resultado pode ser visualizado na Figura 1.7.

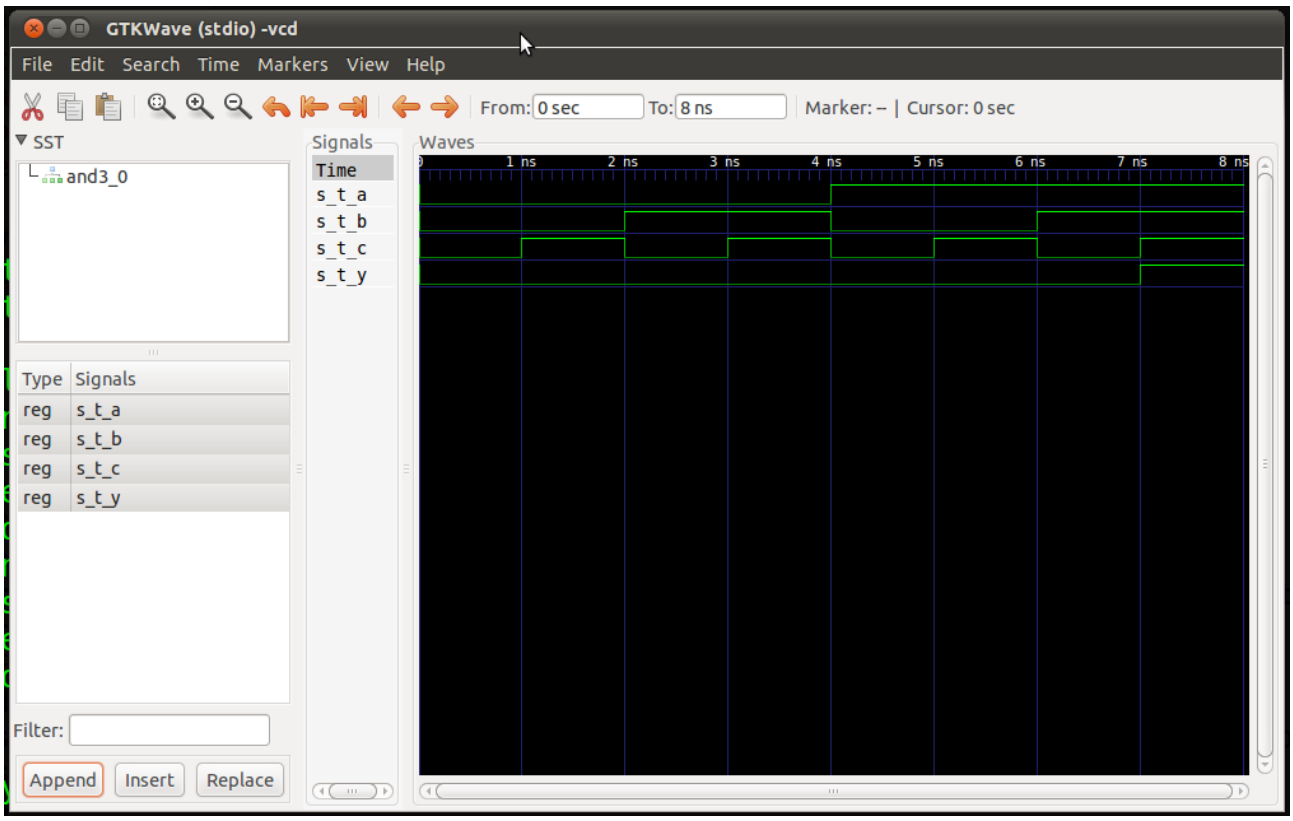

Figura 1.7: Diagrama de Tempo do teste da entidade and3.

O Código 1.8 mostra o código VHDL para a entidade xor2 que foi criado pelo comando make, da mesma forma sendo necessário somente colocarmos o tipo das variáveis e a expressão lógica (y ¡= a xor b) para que o resultado seja gerado corretamente.

Código 1.8: Código VHDL da entidade xor2

```
1 -- Projeto gerado via script.
2 -- Data: Qua,20/07/2011-13:51:40
3 -- Autor: rogerio
4 -- Comentario: Descri  o da Entidade: xor2.
5
6 library ieee;
7 use ieee.std_logic_1164.all;
8 use ieee.std_logic_unsigned.all;
9
10 entity xor2 is
11     port(a,b:in std_logic; y:out std_logic);
12 end xor2;
13
14 architecture logica of xor2 is
15 begin
16     y <= a xor b;
17 end logica;
```

No diretorio testbench também foi criado o arquivo xor2_tb.vhd que é o testbench para a entidade xor2, conforme podemos ver no Código 1.9, novamente alteramos ¡¡type¿¿ para std_logic e criamos os casos de teste.

Código 1.9: Código VHDL do testebench para entidade xor2

```
1 -- Testebench gerado via script.
```

```
2 -- Data: Qua,20/07/2011-13:51:42
3 -- Autor: rogerio
4 -- Comentario: Teste da entidade xor2.
5
6 library ieee;
7 use ieee.std_logic_1164.all;
8 use ieee.std_logic_unsigned.all;
9
10 entity xor2_tb is
11 end xor2_tb;
12
13 architecture logica of xor2_tb is
14   --  Declara  o do componente.
15   component xor2
16         port (a,b: in std_logic; y: out std_logic);
17   end component;
18   --  Especifica qual a entidade est  vinculada com o componente.
19   for xor2_0: xor2 use entity work.xor2;
20       signal s_t_a, s_t_b, s_t_y: std_logic;
21   begin
22     --  Instancia  o do Componente.
23         --  port map (<<p_in_1>> => <<s_t_in_1>>)
24       xor2_0: xor2 port map (a=>s_t_a,b=>s_t_b,y=>s_t_y);
25
26     --  Processo que faz o trabalho.
27     process
28                 -- Um registro    criado com as entradas e sa das da
                       entidade.
29                 -- (<<entrada1>>, <<entradaN>>, <<saida1>>, <<saidaN
                       >>)
30                 type pattern_type is record
31                     -- entradas.
32                     vi_a, vi_b: std_logic;
33                     -- sa das.
34                     vo_y: std_logic;
35                 end record;
36
37                 --  Os padr es de entrada s o aplicados (injetados)
                       s  entradas.
38                 type pattern_array is array (natural range <>) of
                       pattern_type;
39                 constant patterns : pattern_array :=
40                     (
41                         ('0', '0', '0'),
42                         ('0', '1', '1'),
43                         ('1', '0', '1'),
44                         ('1', '1', '0')
45                     );
46                 begin
47                     --  Checagem de padr es.
48                     for i in patterns'range loop
49                         --  Injeta as entradas.
50                         s_t_a <= patterns(i).vi_a;
```

```
51                          s_t_b <= patterns(i).vi_b;
52
53                          --  Aguarda os resultados.
54                          wait for 1 ns;
55                          --  Checa o resultado com a sa da esperada no
                                padr o.
56                          assert s_t_y = patterns(i).vo_y report "Valor
                               de s_t_y n o confere com o resultado
                               esperado." severity error;
57
58                      end loop;
59                      assert false report "Fim do teste." severity note;
60                      --  Wait forever; Isto finaliza a simula  o.
61                      wait;
62          end process;
63  end logica;
```

Executando o comando make apresentado no Código 1.10, se tudo estiver correto, a entidade xor2 será analisada e compilada e o teste será executado e no final será apresentado a tela do gtkwave com o diagrama de tempo (forma de onda).

Código 1.10: Comando para executar o testbench da entidade xor2.

```
1  make all TESTBENCH=xor2_tb
```

O diagrama do resultado pode ser visualizado na Figura 1.8.

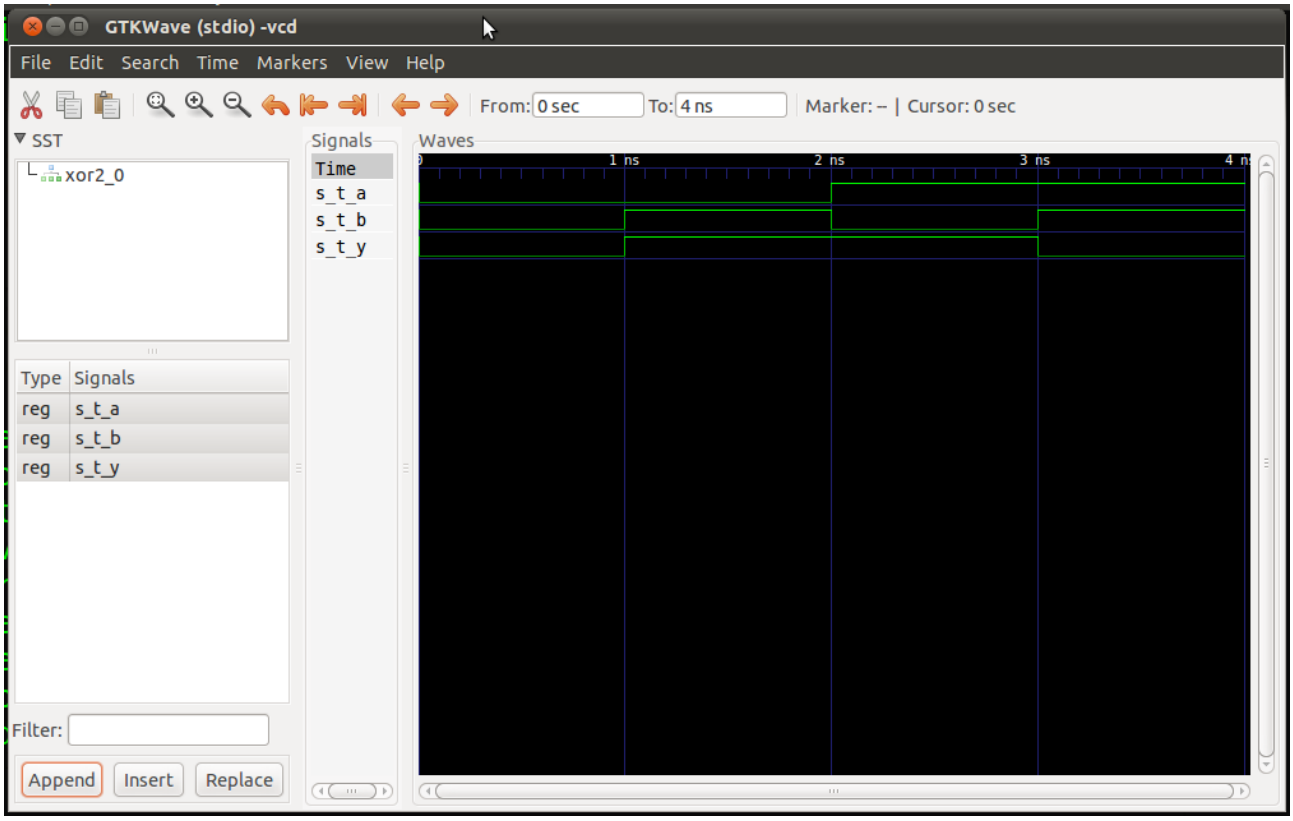

Figura 1.8: Diagrama de Tempo do teste da entidade xor2.

O Código 1.11 mostra o código VHDL para a entidade or3 que foi criado pelo comando make, da mesma forma sendo necessário somente colocarmos o tipo das variáveis e a expressão lógica (y ¡= a xor b) para que o

resultado seja gerado corretamente.

Código 1.11: Código VHDL da entidade or3

```
1 -- Projeto gerado via script.
2 -- Data: Qua,20/07/2011-13:51:40
3 -- Autor: rogerio
4 -- Comentario: Descri  o da Entidade: or3.
5
6 library ieee;
7 use ieee.std_logic_1164.all;
8 use ieee.std_logic_unsigned.all;
9
10 entity or3 is
11     port(a,b,c:in std_logic; y:out std_logic);
12 end or3;
13
14 architecture estrutural of or3 is
15 begin
16     y <= a or b or c;
17 end estrutural;
```

No diretorio testbench também foi criado o arquivo or3_tb.vhd que é o testbench para a entidade or3, conforme podemos ver no Código 1.12, novamente alteramos ¡¡type¿¿ para std_logic e criamos os casos de teste.

Código 1.12: Código VHDL do testebench para entidade or3

```
1 -- Testebench gerado via script.
2 -- Data: Qua,20/07/2011-13:51:40
3 -- Autor: rogerio
4 -- Comentario: Teste da entidade or3.
5
6 library ieee;
7 use ieee.std_logic_1164.all;
8
9 entity or3_tb is
10 end or3_tb;
11
12 architecture logica of or3_tb is
13   --  Declara  o do componente.
14   component or3
15         port (a,b,c: in std_logic; y: out std_logic);
16   end component;
17   --  Especifica qual a entidade est  vinculada com o componente.
18   for or3_0: or3 use entity work.or3;
19       signal s_t_a, s_t_b, s_t_c, s_t_y: std_logic;
20   begin
21     --  Instancia  o do Componente.
22         --  port map (<<p_in_1>> => <<s_t_in_1>>)
23       or3_0: or3 port map (a=>s_t_a,b=>s_t_b,c=>s_t_c,y=>s_t_y);
24
25     --  Processo que faz o trabalho.
26     process
27                 -- Um registro    criado com as entradas e sa das da
                       entidade.
```

```
28                  -- (<<entrada1>>, <<entradaN>>, <<saida1>>, <<saidaN
                       >>)
29                  type pattern_type is record
30                      -- entradas.
31                      vi_a,vi_b,vi_c: std_logic;
32                      -- sa das.
33                      vo_y: std_logic;
34                  end record;
35
36                  --  Os padr es de entrada s o aplicados (injetados)
                        s  entradas.
37                  type pattern_array is array (natural range <>) of
                       pattern_type;
38                  constant patterns : pattern_array :=
39                      (
40                          ('0', '0', '0', '0'),
41                          ('0', '0', '1', '1'),
42                          ('0', '1', '0', '1'),
43                          ('0', '1', '1', '1'),
44                          ('1', '0', '0', '1'),
45                          ('1', '0', '1', '1'),
46                          ('1', '1', '0', '1'),
47                          ('1', '1', '1', '1')
48                      );
49                  begin
50                      --  Checagem de padr es.
51                      for i in patterns'range loop
52                          --  Injeta as entradas.
53                          s_t_a <= patterns(i).vi_a;
54                          s_t_b <= patterns(i).vi_b;
55                          s_t_c <= patterns(i).vi_c;
56
57                          --  Aguarda os resultados.
58                          wait for 1 ns;
59                          --  Checa o resultado com a sa da esperada no
                                padr o.
60                          assert s_t_y = patterns(i).vo_y report "Valor
                               de s_t_y n  o confere com o resultado
                               esperado." severity error;
61
62                      end loop;
63                      assert false report "Fim do teste." severity note;
64                      --  Wait forever; Isto finaliza a simula  o.
65                      wait;
66          end process;
67  end logica;
```

Executando o comando make apresentado no Código 1.13, se tudo estiver correto, a entidade or3 será analisada e compilada e o teste será executado e no final será apresentado a tela do gtkwave com o diagrama de tempo (forma de onda).

Código 1.13: Comando para executar o testbench da entidade or3.

```
1  make all TESTBENCH=or3_tb
```

O diagrama do resultado pode ser visualizado na Figura 1.9.

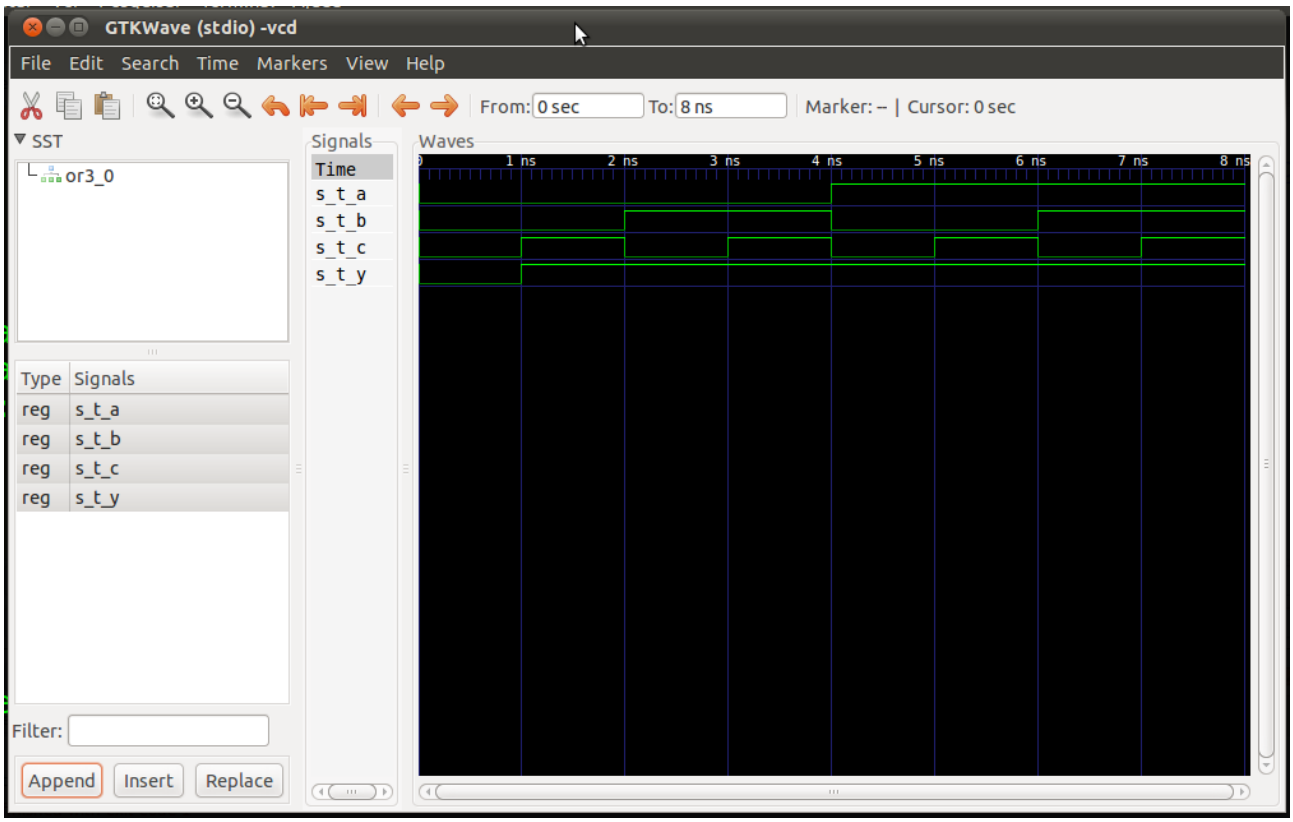

Figura 1.9: Diagrama de Tempo do teste da entidade or3.

Criadas as entidades básicas necessárias, então podemos criar a entidade principal Somador com o comando apresentado no Código 1.14. O Somador foi criado com a arquitetura estrutural com o valor da variável ARCH=estrutural.

Código 1.14: Comando para a criar a entidade Somador.

```
make new PROJECT=somador ARCH=estrutural IN=A,B,Cin,OpSomador OUT=S,
    Cout
```

Criada a entidade Somador apenas é necessário terminar a implementação da arquitetura e do testbench para a entidade.

O Código 1.15 mostra o código VHDL para a entidade somador que foi criado pelo comando make, sendo necessário implementarmos sua função lógica pelo mapeamento das estruturas (port map), já que definimos sua arquitetura como estrutural.

Código 1.15: Código VHDL da entidade somador

```
-- Projeto gerado via script.
-- Data: Qua,20/07/2011-13:51:40
-- Autor: rogerio
-- Comentario: Descri  o da Entidade: somador.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity somador is
    port(a, b, cin, opSomador:in std_logic; s, cout: out std_logic);
```

```
12 end somador;
13
14 architecture estrutural of somador is
15     component and2
16         port(a, b: in std_logic; y: out std_logic);
17     end component;
18
19     component xor2
20         port(a, b: in std_logic; y: out std_logic);
21     end component;
22
23     component and3
24         port(a, b, c: in std_logic; y: out std_logic);
25     end component;
26
27     component or3
28          port(a, b, c: in std_logic; y: out std_logic);
29     end component;
30
31     signal s_X1_X2, s_X2_A1, s_A1_O1, s_A2_O1, s_A3_O1, s_A4_O1:
          std_logic;
32 begin
33     X1: xor2 port map(a=>a, b=>b, y=>s_X1_X2);
34     X2: xor2 port map(a=>s_X1_X2, b=>cin, y=>s_X2_A1);
35     A1: and2 port map(a=>s_X2_A1, b=>opSomador, y=>s);
36     A2: and3 port map(a=>a, b=>b, c=>opSomador, y=>s_A2_O1);
37     A3: and3 port map(a=>a, b=>cin, c=>opSomador, y=>s_A3_O1);
38     A4: and3 port map(a=>b, b=>cin, c=>opSomador, y=>s_A4_O1);
39     O1: or3 port map(a=>s_A2_O1, b=>s_A3_O1, c=>s_A4_O1, y=>cout);
40 end estrutural;
```

No diretorio testbench também foi criado o arquivo somador_tb.vhd que é o testbench para a entidade somador, conforme podemos ver no Código 1.16, novamente alteramos ¡¡type¿¿ para std_logic e criamos os casos de teste.

Código 1.16: Código VHDL do testebench para entidade somador

```
1 -- Testebench gerado via script.
2 -- Data: Qua,20/07/2011-13:51:40
3 -- Autor: rogerio
4 -- Comentario: Descri  o da Entidade: somador.
5
6 library ieee;
7 use ieee.std_logic_1164.all;
8 use ieee.std_logic_unsigned.all;
9
10 entity somador_tb is
11 end somador_tb;
12
13 architecture estrutural of somador_tb is
14   --  Declara  o do componente.
15   component somador
16     port(a, b, cin, opSomador:in std_logic; s, cout: out std_logic);
17   end component;
18   --  Especifica qual a entidade est  vinculada com o componente.
```

```
19  for somador_0: somador use entity work.somador;
20      signal t_a, t_b, t_cin, t_opSomador, t_s, t_cout: std_logic;
21  begin
22    --  Instancia  o do Componente.
23    somador_0: somador port map (a=>t_a, b=>t_b, cin=>t_cin, opSomador
          =>t_opSomador, s=>t_s, cout=>t_cout);
24
25    --  Processo que faz o trabalho.
26    process
27                -- Um registro    criado com as entradas e sa das da
                      entidade.
28                type pattern_type is record
29                    -- entradas.
30                    vi_a, vi_b, vi_cin, vi_opSomador: std_logic;
31                    --  sa das.
32                    vo_s, vo_cout : std_logic;
33                end record;
34
35                --  Os padr es de entrada s o aplicados (injetados)
                      s  entradas.
36                type pattern_array is array (natural range <>) of
                      pattern_type;
37                constant patterns : pattern_array :=
38                    (('0', '0', '0', '1', '0', '0'),
39                     ('0', '0', '1', '1', '1', '0'),
40                     ('0', '1', '0', '1', '1', '0'),
41                     ('0', '1', '1', '1', '0', '1'),
42                     ('1', '0', '0', '1', '1', '0'),
43                     ('1', '0', '1', '1', '0', '1'),
44                     ('1', '1', '0', '1', '0', '1'),
45                     ('1', '1', '1', '1', '1', '1'),
46                     ('1', '1', '1', '0', '0', '0')
47                    );
48                    begin
49                        --  Checagem de padr es.
50                        for i in patterns'range loop
51                            --  Injeta as entradas.
52                            t_a <= patterns(i).vi_a;
53                            t_b <= patterns(i).vi_b;
54                            t_cin <= patterns(i).vi_cin;
55                            t_opSomador <= patterns(i).vi_opSomador;
56                            --  Aguarda os resultados.
57                            wait for 1 ns;
58                            --  Checa o resultado com a sa da
                                  esperada no padr o.
59                            assert t_s = patterns(i).vo_s
60                                report "Valor de t_s  n  o  confere com
                                      o resultado esperado." severity
                                      error;
61                            assert t_cout = patterns(i).vo_cout
62                                report "Valor de t_cout  n  o  confere
                                      com o resultado esperado."severity
                                      error;
```

```
63                          end loop;
64                          assert false report "Fim do teste." severity
                               note;
65                          --  Wait forever; Isto finaliza a simula  o.
66                          wait;
67          end process;
68  end estrutural;
```

Executando o comando make apresentado no Código 1.17, se tudo estiver correto, a entidade somador será analisada e compilada e o teste será executado e no final será apresentado a tela do gtkwave com o diagrama de tempo (forma de onda).

Código 1.17: Comando para executar o testbench da entidade somador.

```
1  make all TESTBENCH=somador_tb
```

O diagrama do resultado pode ser visualizado na Figura 1.10.

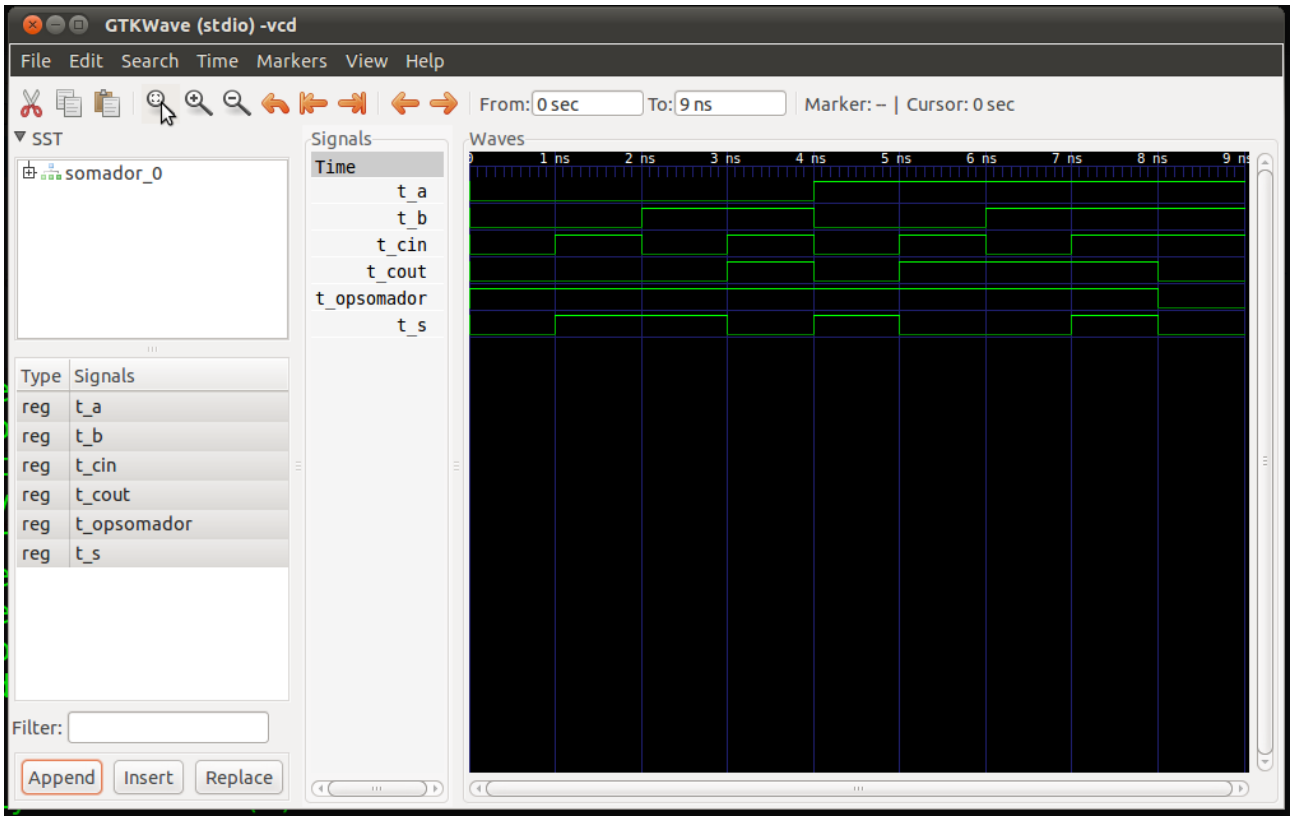

Figura 1.10: Diagrama de Tempo do teste da entidade somador.

Com o Somador funcionando, podemos ir para a próxima entidade, o Decodificador. O Decodificador depende de algumas entidades básicas, a maioria já foi criada para satisfazer as dependências da entidade Somador, restando apenas a criação de um inversor, conforme o Código 1.18.

Código 1.18: Comando para a criar a entidade Inversor.

```
1  make new PROJECT=inversor ARCH=logica IN=a OUT=y
```

O Código 1.19 mostra o código VHDL para a entidade inversor que foi criado pelo comando make, da mesma forma sendo necessário somente colocarmos o tipo das variáveis e a expressão lógica (y ¡= not a) para que o resultado seja gerado corretamente.

Código 1.19: Código VHDL da entidade inversor

```
1 -- Projeto gerado via script.
2 -- Data: Sex,30/12/2011-23:36:18
3 -- Autor: rogerio
4 -- Comentario: Descri  o da Entidade: inversor.
5
6 library ieee;
7 use ieee.std_logic_1164.all;
8 use ieee.std_logic_unsigned.all;
9
10 entity inversor is
11     port (a: in std_logic; y: out std_logic);
12 end inversor;
13
14 architecture logica of inversor is
15
16
17 begin
18   -- Comandos.
19   y <= not a;
20 end logica;
```

No diretorio testbench também foi criado o arquivo inversor_tb.vhd que é o testbench para a entidade inversor, conforme podemos ver no Código 1.20, novamente alteramos ¡¡type¿¿ para std_logic e criamos os casos de teste.

Código 1.20: Código VHDL do testebench para entidade inversor

```
1 -- Testebench gerado via script.
2 -- Data: Sex,30/12/2011-23:36:18
3 -- Autor: rogerio
4 -- Comentario: Teste da entidade inversor.
5
6 library ieee;
7 use ieee.std_logic_1164.all;
8 use ieee.std_logic_unsigned.all;
9
10 entity inversor_tb is
11 end inversor_tb;
12
13 architecture logica of inversor_tb is
14   --  Declara  o do componente.
15   component inversor
16     port (a: in std_logic; y: out std_logic);
17   end component;
18   --  Especifica qual a entidade est  vinculada com o componente.
19   for inversor_0: inversor use entity work.inversor;
20       signal s_t_a, s_t_y: std_logic;
21   begin
22     --  Instancia  o do Componente.
23     --  port map (<<p_in_1>> => <<s_t_in_1>>)
24     inversor_0: inversor port map ( a=>s_t_a, y=>s_t_y);
25
26     --  Processo que faz o trabalho.
```

```
27      process
28          -- Um registro    criado com as entradas e sa das da entidade
               .
29          -- (<<entrada1>>, <<entradaN>>, <<saida1>>, <<saidaN>>)
30          type pattern_type is record
31              -- entradas.
32               vi_a: std_logic;
33              -- sa das.
34               vo_y: std_logic;
35          end record;
36
37          --  Os padr es de entrada que s o aplicados (injetados)  s
               entradas.
38          type pattern_array is array (natural range <>) of pattern_type
               ;
39          -- Casos de teste.
40          constant patterns : pattern_array :=
41          (
42              ('0', '1'),
43              ('1', '0'),
44              ('0', '1'),
45              ('0', '1'),
46              ('1', '0'),
47              ('1', '0')
48          );
49          begin
50          --  Checagem de padr es.
51          for i in patterns'range loop
52              --  Injeta as entradas.
53              s_t_a <= patterns(i).vi_a;
54
55              --  Aguarda os resultados.
56              wait for 1 ns;
57              --  Checa o resultado com a sa da esperada no padr o.
58              assert s_t_y = patterns(i).vo_y report "Valor de s_t_y
                   n  o  confere  com  o  resultado  esperado." severity error;
59
60          end loop;
61          assert false report "Fim do teste." severity note;
62          --  Wait forever; Isto finaliza a simula  o.
63          wait;
64      end process;
65  end logica;
```

Executando o comando make apresentado no Código 1.21, se tudo estiver correto, a entidade inversor será analisada e compilada e o teste será executado e no final será apresentado a tela do gtkwave com o diagrama de tempo (forma de onda).

Código 1.21: Comando para executar o testbench da entidade inversor.

```
1  make all TESTBENCH=inversor_tb
```

O diagrama do resultado pode ser visualizado na Figura 1.11.

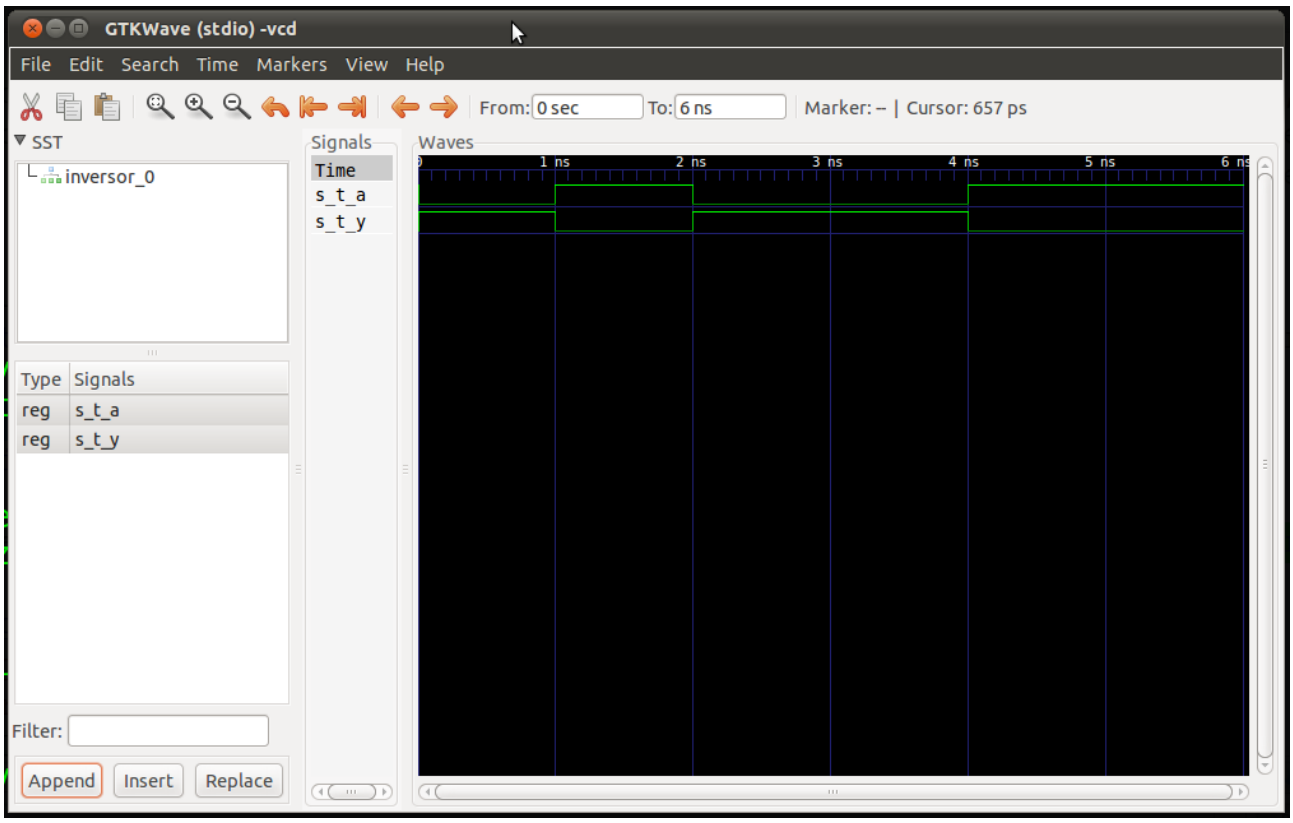

Figura 1.11: Diagrama de Tempo do teste da entidade inversor.

Com todas as dependências do Decodificador satisfeitas, podemos então criá-lo, com o comando apresenta no Código 1.22.

Código 1.22: Comando para a criar a entidade Decodificador.

```
make new PROJECT=decodificador ARCH=estrutural IN=F0,F1,F2 OUT=Op0,
    Op1,Op2,Op3,Op4,Op5,Op6,Op7
```

Implementa-se a entidade e testbench do decodificador, deixando-o funcionando.

A próxima entidade a ser criada então é a Unidade Lógica. Para esta entidade precisamos criar os componentes básicos que ainda não temos no projeto e que precisaremos para satisfazer a entidade unidadeLogica, sendo eles or2, nand2, nor2, xnor2. O comando para criá-los é apresentado no Código 1.23.

Código 1.23: Comando para a criar entidades básicas para a Unidade Lógica.

```
make new PROJECT=or2 ARCH=logica
make new PROJECT=nand2 ARCH=logica
make new PROJECT=nor2 ARCH=logica
make new PROJECT=xnor2 ARCH=logica
```

Tendo todas as dependências satisfeitas, podemos então criar a entidade unidadeLogica, conforme o Código 1.24.

Código 1.24: Comando para a criar a entidade unidadeLogica.

```
make new PROJECT=unidadeLogica ARCH=estrutural
```

Implementa-se a entidade e testbench da unidadeLogica, deixando-a funcionando.

Até aqui, implementamos todos os componentes principais necessários para a implementação da ULA. Então podemos criar a entidade ULA, com o comando apresentado no Código 1.25.

Código 1.25: Comando para a criar a entidade ULA.

```
make new PROJECT=ula ARCH=estrutural
```

Verifica-se a implementação da entidade ULA e em seu testbench é necessário apenas a criação dos casos de teste. Testamos e a deixamos funcionando. A técnica do menor para o maior, por composição permite ir testando e vendo os resultados dos componentes até atingir o resultado maior que é o projeto como um todo.

Se iniciássemos pela entidade ULA (do maior para o menor) iríamos poder testá-la somente no final, quando todos os componentes estivessem terminados, até lá, muitos erros iriam acontecer.