Tutorial
Projeto: ULA de 1 bit

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

30 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.

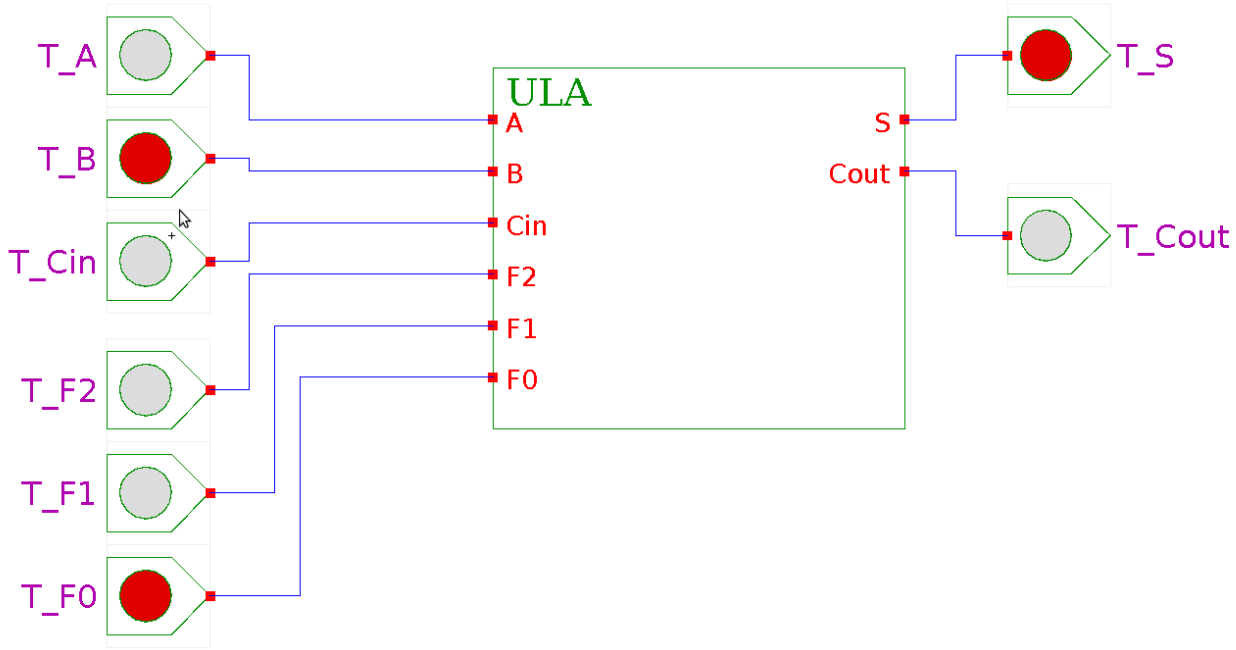

Figura 1.1: Visão da ULA em um nível 0

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

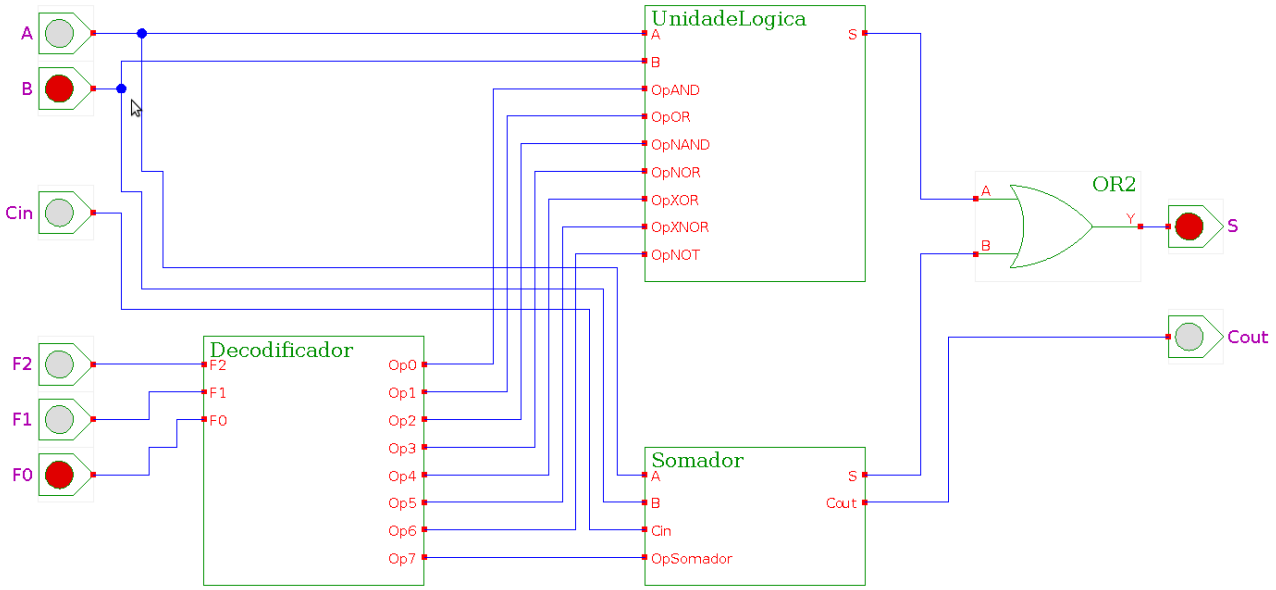

Figura 1.2: Visão da ULA em um nível 1, exposição dos componentes principais.

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.

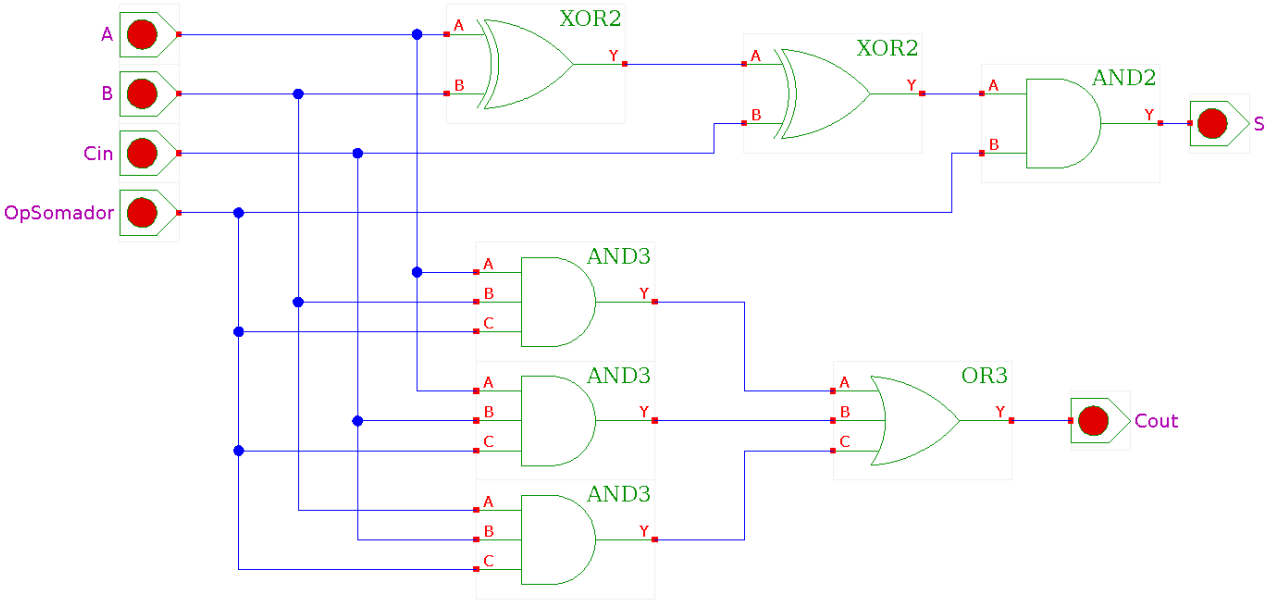

Figura 1.3: Circuito Somador.

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.

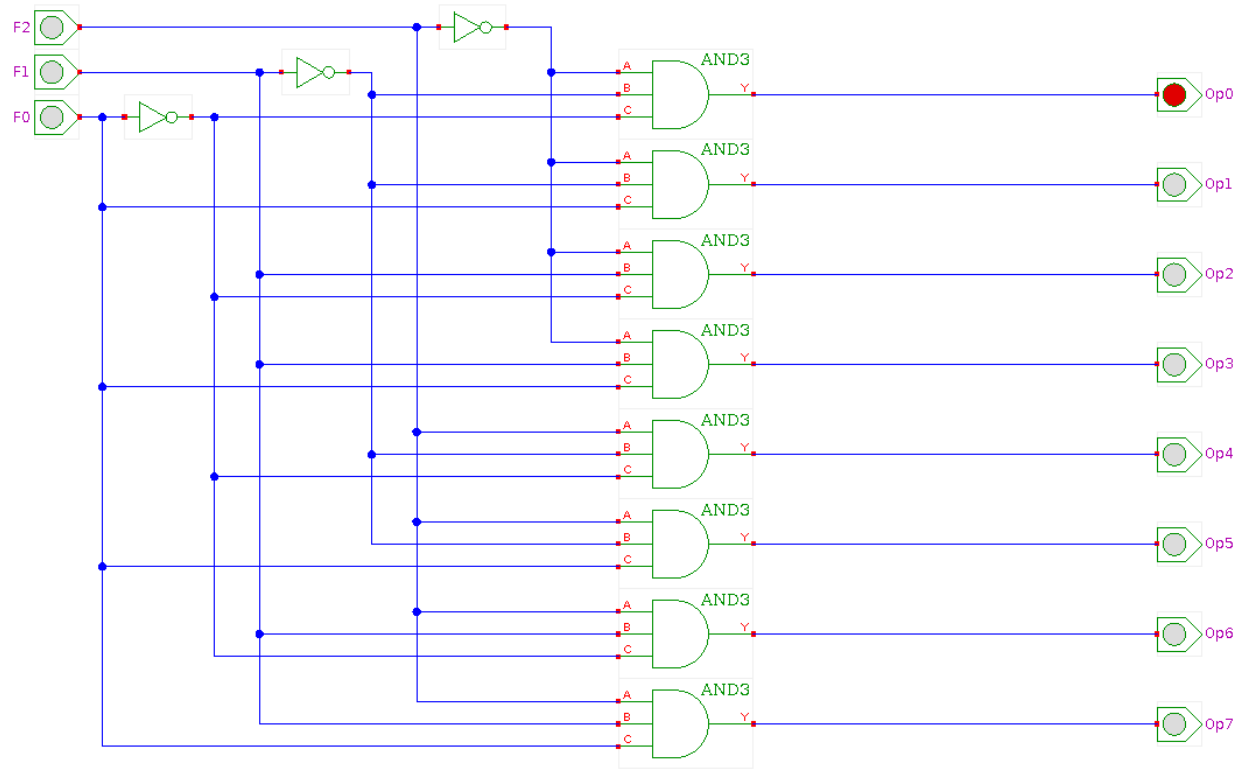

Figura 1.4: Circuito Decodificador.

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.

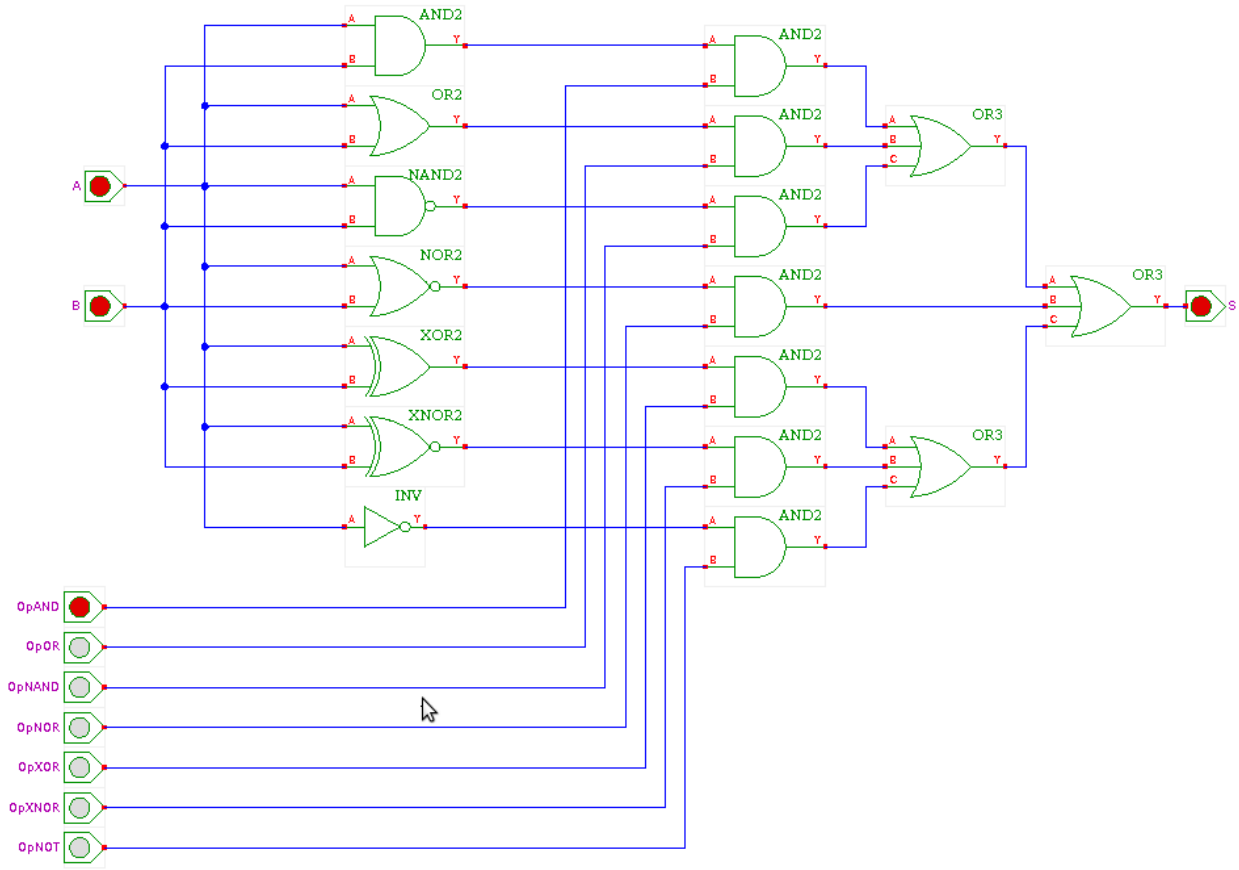

Figura 1.5: Circuito UnidadeLogica.

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,20/07/2011-13:51:31
3  -- Autor: rogerio
4  -- Comentario: Descricao 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 logica of and2 is
15 begin
16   -- Comandos.
17   y <= a and b;
18 end logica;
```

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 library ieee;
7 use ieee.std_logic_1164.all;
8
9 entity and2_tb is
10 end and2_tb;
11
12 architecture logica of and2_tb is
13   --  Declaracao do componente.
14   component and2
15         port (a,b: in std_logic; y: out std_logic);
16   end component;
17   --  Especifica qual a entidade esta vinculada com o componente.
18   for and2_0: and2 use entity work.and2;
19       signal s_t_a, s_t_b, s_t_y: std_logic;
20   begin
21     --  Instanciacao do Componente.
22     and2_0: and2 port map (a=>s_t_a,b=>s_t_b,y=>s_t_y);
23
24     --  Processo que faz o trabalho.
25     process
26     -- Um registro e criado com as entradas e saidas da entidade.
27     -- (<<entrada1>>, <<entradaN>>, <<saida1>>, <<saidaN>>)
28     type pattern_type is record
29     -- entradas.
30         vi_a,vi_b: std_logic;
31     -- saidas.
32         vo_y: std_logic;
33     end record;
34
35     --  Os padroes de entrada sao aplicados (injetados) as entradas.
36     type pattern_array is array (natural range <>) of pattern_type;
37     constant patterns : pattern_array :=
38     (
39         ('0', '0', '0'),
40         ('0', '1', '0'),
41         ('1', '0', '0'),
42         ('1', '1', '1')
43     );
44     begin
45       --  Checagem de padroes.
46       for i in patterns'range loop
47         --  Injeta as entradas.
48         s_t_a <= patterns(i).vi_a;
49         s_t_b <= patterns(i).vi_b;
50
51         --  Aguarda os resultados.
52         wait for 1 ns;
53         --  Checa o resultado com a saida esperada no padrao.
```

```
54          assert s_t_y = patterns(i).vo_y report "Valor de s_t_y nao
               confere com o resultado esperado." severity error;
55
56        end loop;
57        assert false report "Fim do teste." severity note;
58        --  Wait forever; Isto finaliza a simulacao.
59        wait;
60      end process;
61  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.

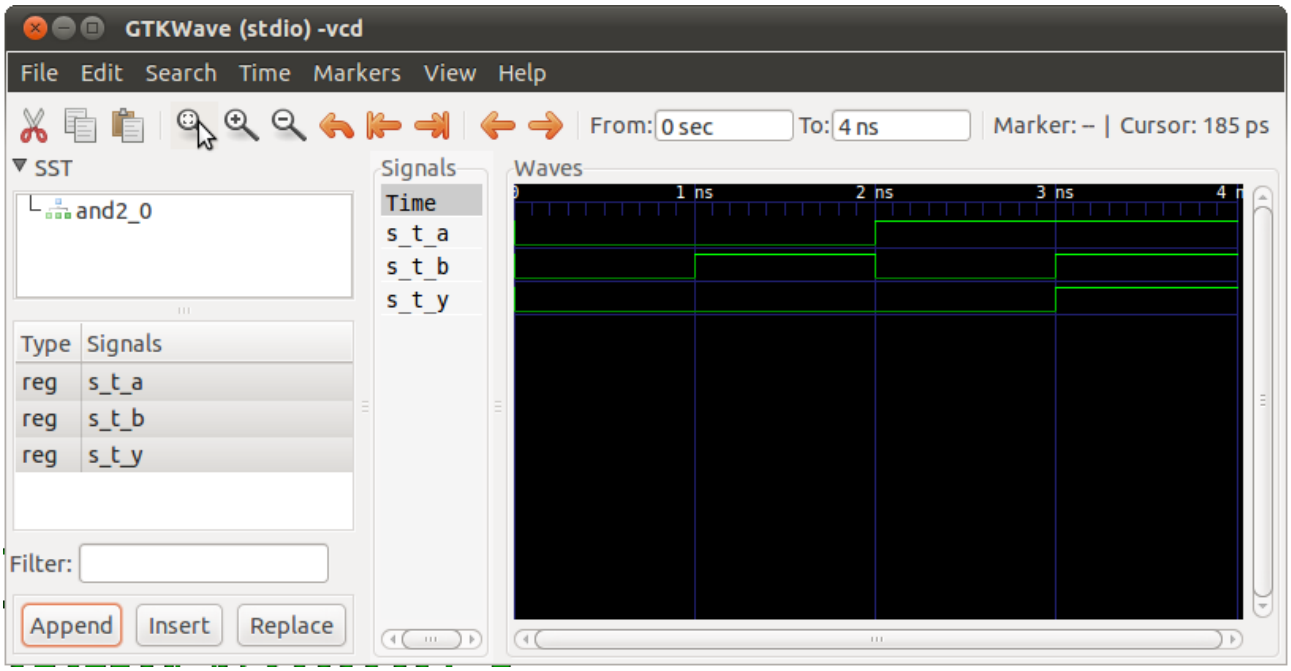

Figura 1.6: Diagrama de Tempo do teste da entidade and2.

Criadas as entidades básicas necessárias, então podemos criar a entidade principal Somador com o comando apresentado no Código 1.5. O Somador foi criado com a arquitetura estrutural com o valor da variável ARCH=estrutural.

Código 1.5: Comando para a criar a entidade Somador.

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

Criada a entidade Somador apenas é necessário terminar a implementação do testbench para esta entidade, que neste caso consiste em criar os casos te teste. 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 1.6.

Código 1.6: Comando para a criar a entidade Inversor.

```
make new PROJECT=inversor ARCH=logica
```

Com todas as dependências do Decodificador satisfeitas, podemos então criá-lo, com o comando apresenta no Código 1.7.

Código 1.7: Comando para a criar a entidade Decodificador.

```
make new PROJECT=decodificador ARCH=estrutural
```

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 1.8.

Código 1.8: 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 1.9.

Código 1.9: 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 1.10.

Código 1.10: 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.