UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
CONCEPÇÃO ESTRUTURADA DE CIRCUITOS INTEGRADOS

RELATÓRIO I

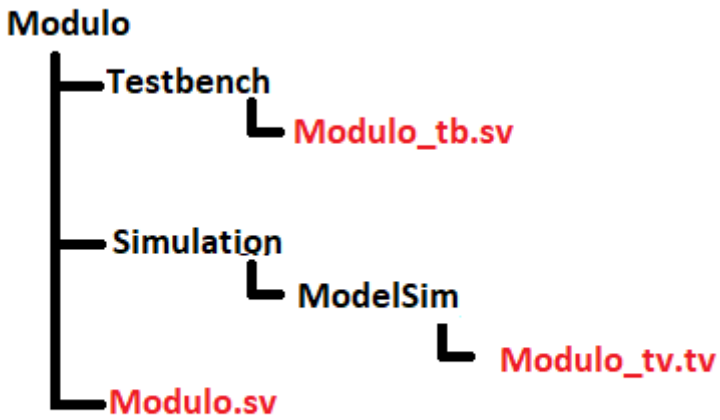
THIAGO ALVES DE ARAUJO

2016019787

1. Organização dos diretórios
2. Configurando o ambiente de projeto
3. Inversor
    3.1. Modelo DUV
    3.2. Goldem model
    3.3. Simulação
4. Tristate
    4.1. Modelo DUV
    4.2. Goldem model
    4.3. Simulação
5. Mux 2:1
    5.1. Modelo DUV
    5.2. Goldem model
    5.3. Simulação

1 – Organização dos diretórios

Antes de iniciarmos a configuração do simulador, devemos criar um diretório para o modulo que irá ser criado. Este diretório irá conter os arquivos de simulação, os *goldem vectors*, os módulos de discrição *duv*, entre outros. A organização das pastas serão dadas da seguinte forma:



Este esquema vai servir para todos os módulos que vão ser criados posteriormente (inversor, tristate, mux etc.).

É importante ressaltar alguns pontos: a extenção ".sv" significa "*sistem verilog*". O arquivo "Modulo.sv" contém uma descrição comportamental em alto nível. Já o arquivo "Modulo_tb.sv" contém uma série de informações que são usadas para simular o comportamento do modulo instanciado. Com ela nós podemos comparar respostas de saída (mediante a entradas aplicadas) com os resultados esperados e verificar a ocorrência de eventuais erros.

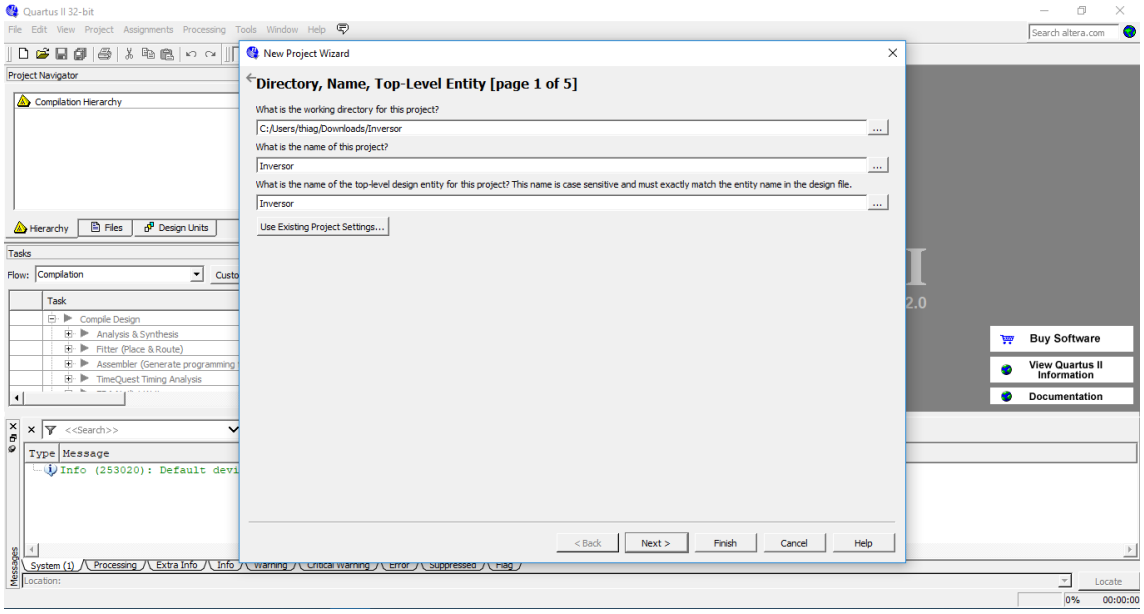
A extensão ".tv" significa "*test vector*". Estes vetores são chamados de "*goldem model*" pois são eles que contem os resultados ideais esperados pelo modulo. Em outras palavras, é nele que está descrita a tabela verdade do modulo (porém ele pode ou não conter todas as possibilidades de

entradas. Para circuitos muito complexos se torna inviável analisar todas as combinações de entradas, então as entradas são pensadas de forma inteligente para analisar por completo o comportamento do modulo utilizando o menor número de entradas possíveis. Estes arquivos podem ser criados em linguagens de alto nível como Python ou C/C++.

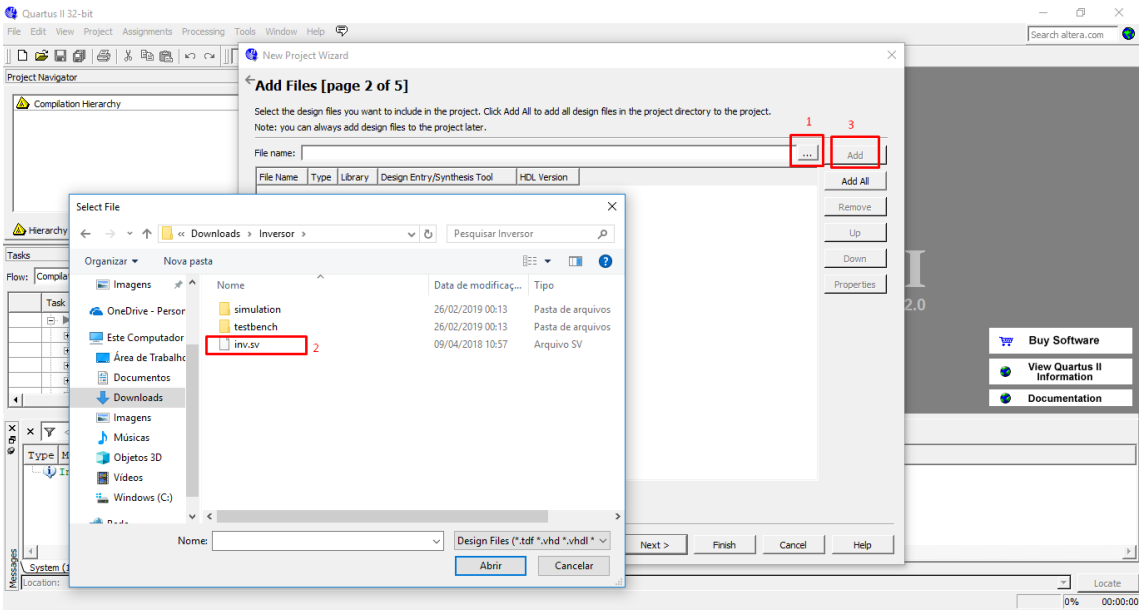
2 – Configuração do ambiente de projeto

Após criados os diretórios com os arquivos podemos iniciar a configurações do nosso ambiente de projeto. O programa utilizado para criar o projeto será o Quartus II 12.0 Web Edition e o ModelSim será utilizado para a simulação RTL.

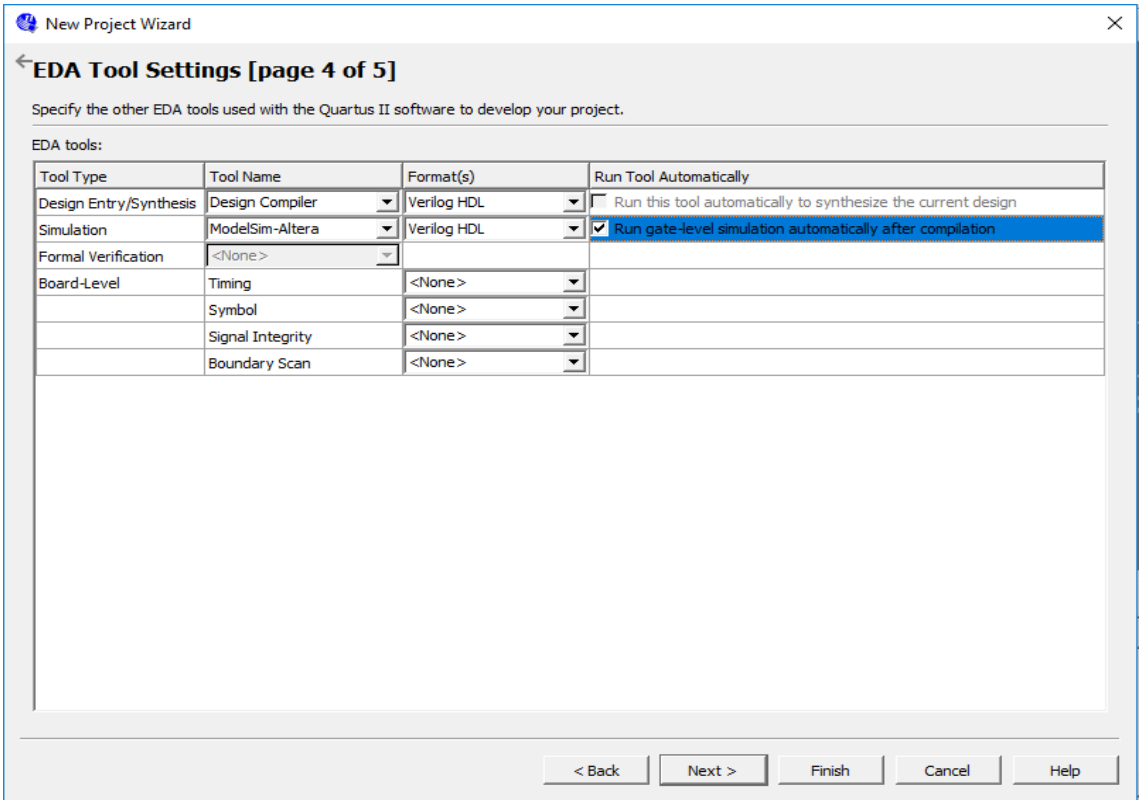
Com o Quartus aberto, vamos em File->New Project Wizard. Selecionamos a pasta que criamos anteriormente e inserimos o nome do projeto.



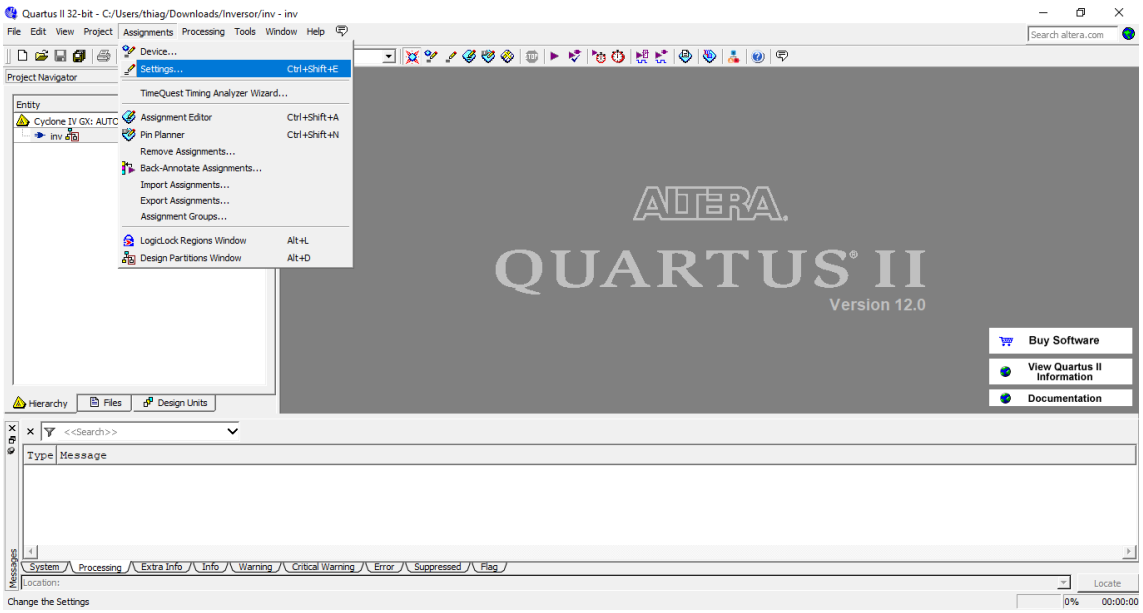
Após isso adicionamos o arquivo “modulo.sv” criado anteriormente



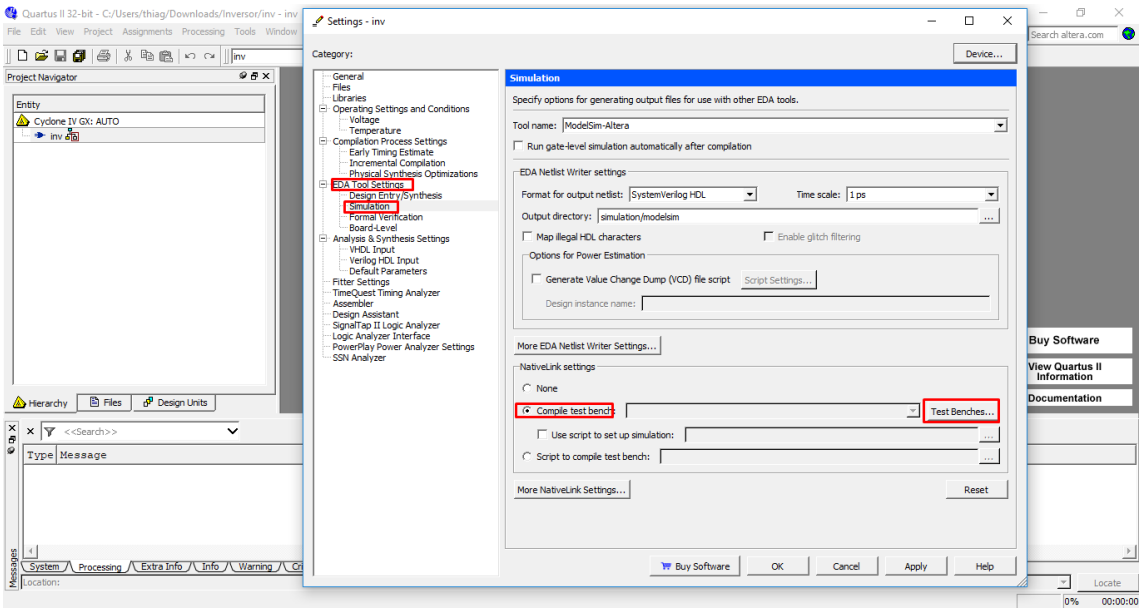
No passo 3 apenas seguimos em frente. Já no passo 4, selecionamos o desing da entrada e a simulação conforme a imagem abaixo:



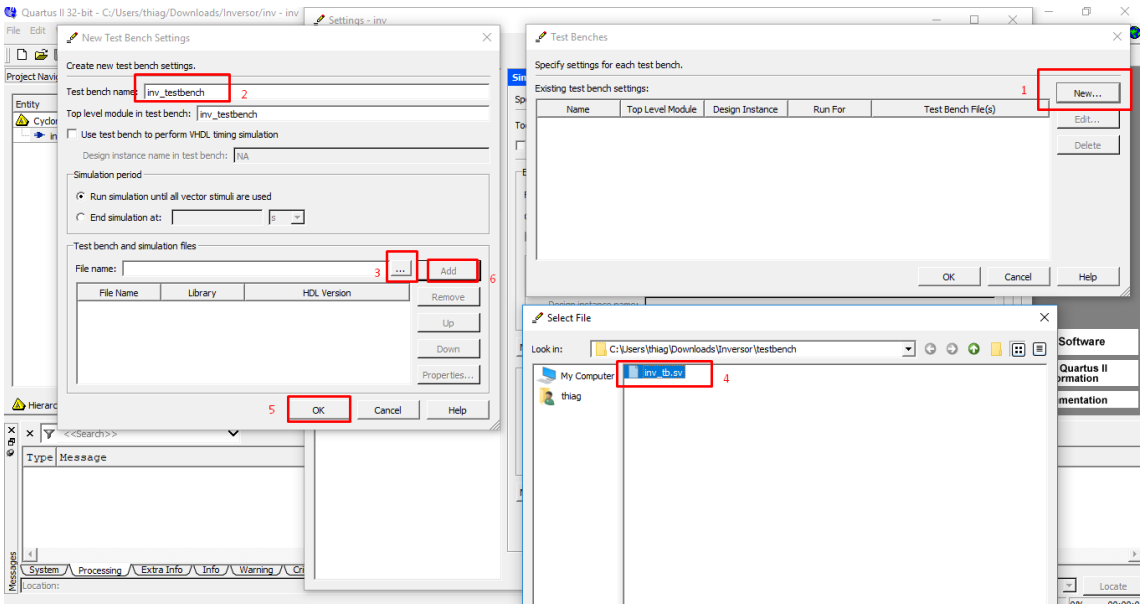
Agora vamos adicionar o nosso testbench. Com o projeto criado, vamos em assignments->settings



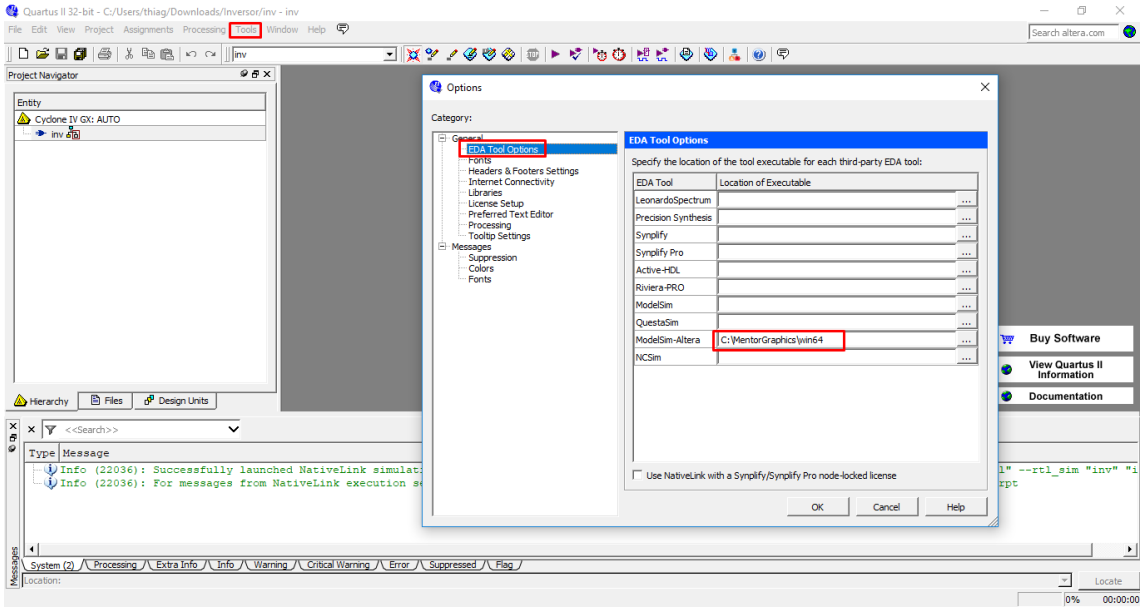
Vamos em simulation->Testbench



Vamos em “new” e adicionamos o nome do arquivo. Depois vamos em “...”, selecionamos o arquivo “inv_tb.sv”, vamos em “add” e “ok”.



Por fim, vamos selecionar o simulador ModelSim. Vamos em tools->options->EDA tool options e adicionamos o caminho da pasta raiz do modelSim



3 – Inversor

Como os diretórios e arquivos do inversor já foram criados/utilizados como exemplo, não precisamos repetir os passos anteriores. Vamos iniciar construindo o modelo duv *(design for verification*).

ENTRADA	SAIDA
A	S
0	1
1	0

3.1 – Modelo DUV

Como mencionado anteriormente, o arquivo “inversor.sv” vai conter uma descrição do modulo que representa o comportamento do circuito. Para o inversor, temos apenas uma entrada (que aqui será descrita como um array de 4bits) e uma saída. A imagem abaixo mostra o comportamento deste modulo descrito em SistemVerylog

```
module inv(a, y);

   input  logic [3:0]a;
   output logic [3:0]y;

   assign y = ~a;

endmodule
```

3.2 – Goldem Model

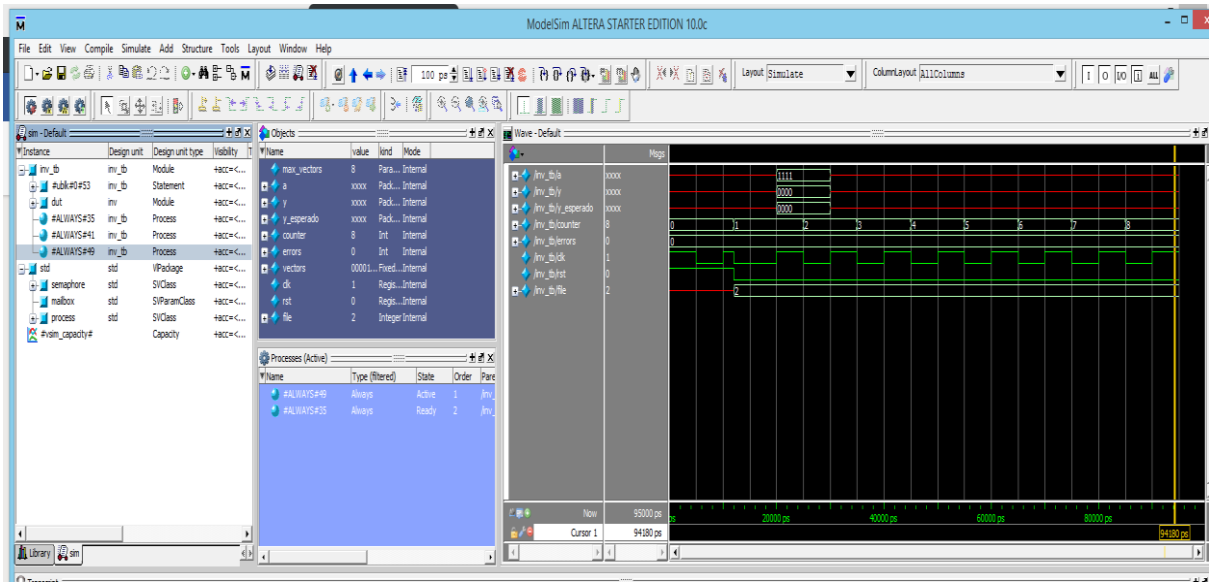
Para gerar o arquivo “inv_tv.tv” foi utilizado um código em C++ que insere em um arquivo partes da tabela verdade o inversor de 4bits. O arquivo está formatado da seguinte forma*: bits de entrada_saida* (ex.: 0000_1111)

Abaixo está o resultado da saída.

```
0000_1111
1111_0000
```

3.3 – Simulação

Para simular, vamos em tools->run EDA simulation tool->run RTL simulation. Aqui podemos ver os pulsos inseridos e as saídas resultantes. Para melhor visualização, basta clicar em “zoom full”.



Na janela “transcript” podemos ver as mensagens de erro encontradas. Como as saídas geradas foram exatamente iguais as saídas ideias, nenhum erro foi encontrado.

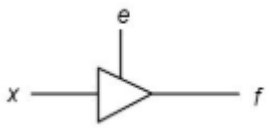
```
# -------------
#
# |linha |   A    |    Y   |
# |  1   |  1111  |  0000  | OK
# Testes Efetuados  = 8
# Erros Encontrados = 0
# Break in Module inv_tb at C:/Users/aluno/Downloads/Inversor/Inv_Aula/testbench/inv_tb.sv line 83
# Simulation Breakpoint: Break in Module inv_tb at C:/Users/aluno/Downloads/Inversor/Inv_Aula/testbench/inv_tb.sv line 83
# MACRO ./inv_run_msim_rtl_verilog.do PAUSED at line 17

VSIM(paused)> {\rtf1\ansi\ansicpg1252\deff0\nouicompat\deflang1046{\fonttbl{\f0\fnil\fcharset0 Calibri;}}
{\*\generator Riched20 10.0.17134}\viewkind4\uc1
```

4 – Tristate

Como o nome sugere, o tristate *(three-state)* pode assumir três valores de saída: 0, 1 ou Z. Caso a entrada En esteja em 0, a saída Y irá assumir valor “Z” (alta impedância). Caso En esteja em 1, o valor da saída é o mesmo da entrada a. As imagens abaixo ilustram esse comportamento.

e	x	f
0	0	Z
0	1	Z
1	0	0
1	1	1



4.1 – Modelo DUV

Analisando o comportamento descrito anteriormente podemos desenvolver uma modelagem para o circuito. Abaixo está o código que representa este modulo.

```
module Tristate(a, en, y);

   input logic [3:0] a;
   input logic en;
   output tri [3:0] y;

   assign y = en ? a : 4'bz;

endmodule
```

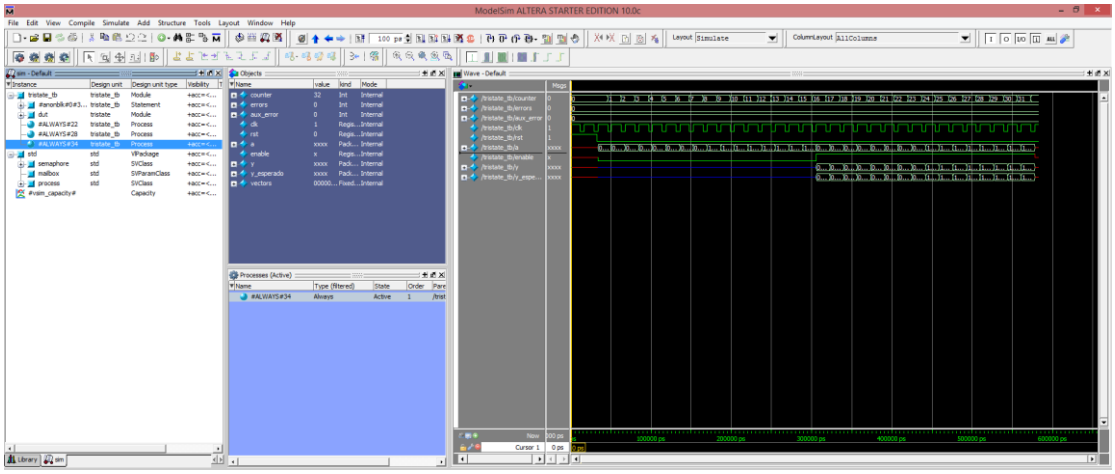
4.2 – Goldem Model

Assim como no modelo gerado anteriormente, podemos representar o comportamento da tabela verdade do tristate utilizando uma linguagem de alto nível de sua preferência. A seguir está o goldem model gerado para o tristate.

```
0_0000_zzzz
0_0001_zzzz
0_0010_zzzz
0_0011_zzzz
0_0100_zzzz
0_0101_zzzz
0_0110_zzzz
0_0111_zzzz
0_1000_zzzz
0_1001_zzzz
0_1010_zzzz
0_1011_zzzz
0_1100_zzzz
0_1101_zzzz
0_1110_zzzz
0_1111_zzzz
1_0000_0000
1_0001_0001
1_0010_0010
1_0011_0011
1_0100_0100
1_0101_0101
1_0110_0110
1_0111_0111
1_1000_1000
1_1001_1001
1_1010_1010
1_1011_1011
1_1100_1100
1_1101_1101
1_1110_1110
1_1111_1111
```

4.3 – Simulação

Seguindo os menos passos anteriores, vamos em run RTL simulation para iniciar a simulação.



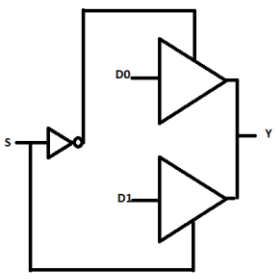
Podemos ver que nenhum erro foi encontrado.

```
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# Iniciando Testbench
#  |  E  |  A  |  Y  |
#  |  0  | 0000 | zzzz | OK
#  |  0  | 0001 | zzzz | OK
#  |  0  | 0010 | zzzz | OK
#  |  0  | 0011 | zzzz | OK
#  |  0  | 0100 | zzzz | OK
#  |  0  | 0101 | zzzz | OK
#  |  0  | 0110 | zzzz | OK
#  |  0  | 0111 | zzzz | OK
#  |  0  | 1000 | zzzz | OK
#  |  0  | 1001 | zzzz | OK
#  |  0  | 1010 | zzzz | OK
#  |  0  | 1011 | zzzz | OK
#  |  0  | 1100 | zzzz | OK
#  |  0  | 1101 | zzzz | OK
#  |  0  | 1110 | zzzz | OK
#  |  0  | 1111 | zzzz | OK
#  |  1  | 0000 | 0000 | OK
#  |  1  | 0001 | 0001 | OK
#  |  1  | 0010 | 0010 | OK
#  |  1  | 0011 | 0011 | OK
#  |  1  | 0100 | 0100 | OK
#  |  1  | 0101 | 0101 | OK
#  |  1  | 0110 | 0110 | OK
#  |  1  | 0111 | 0111 | OK
#  |  1  | 1000 | 1000 | OK
#  |  1  | 1001 | 1001 | OK
#  |  1  | 1010 | 1010 | OK
#  |  1  | 1011 | 1011 | OK
#  |  1  | 1100 | 1100 | OK
#  |  1  | 1101 | 1101 | OK
#  |  1  | 1110 | 1110 | OK
#  |  1  | 1111 | 1111 | OK
# Testes Efetuados  = 32
# Erros Encontrados = 0
# Break in Module tristate_tb at C:/Users/aluno/Downloads/Suanny (2)/Suanny/tristate/testbench/tristate_tb.sv line 57
# Simulation Breakpoint: Break in Module tristate_tb at C:/Users/aluno/Downloads/Suanny (2)/Suanny/tristate/testbench/tristate_tb.sv line 57
# MACRO ./tristate_run_msim_rtl_verilog.do PAUSED at line 17

VSIM(paused)>
```

5 – Mux 2:1

O mux 2:1 pode ser construído utilizando um inversor e dois tistate como mostrado na figura abaixo.



5.1 – Modulo DUV

Abaixo podemos ver a modelagem do mux2:1

```
module mux(d0, d1, s, ns, y);

      input logic [3:0] d0;
      input logic [3:0] d1;
      input logic s;
      output logic ns;
      output tri [3:0] y;

      tristate t0(d0, ns, y);
      tristate t1(d1, s, y);
      inv inversor(s, ns);

endmodule
```

5.2 – Goldem Model

Para o teste, aplicamos duas entradas e variamos o controle s para verificar a saída.

```
0000_1111_0_0000
1111_0000_0_1111
0000_1111_1_1111
1111_0000_1_0000
```

5.3 - Simulação