

# Aprendendo a utilizar a Ferramenta Modelsim



Professor: Lucas Cambuim (lfsc)  
Adaptado por : Lucas Amorim,  
Matheus Costa e Anderson  
Henrique



# Visão da Ferramenta ModelSim

- É um simulador computacional para análise de sistemas digitais

# Visão da Ferramenta ModelSim

- Possui alta fidelidade de resultados

# Visão da Ferramenta ModelSim

- Possui alta fidelidade de resultados
  - os resultados obtidos na simulação refletem fielmente os resultados reais do circuito rodando na FPGA.

# Visão da Ferramenta ModelSim

- Atualmente é o simulador de sistemas digitais **mais aceito** tanto pelo mundo acadêmico como pela indústria.

# Facilidades oferecidas pela ferramenta

- Mais rápido para simular e corrigir o seu código
  - Tempo de compilação é muito menor do que o tempo de síntese em plataforma.

# Facilidades oferecidas pela ferramenta

- Suporta a linguagem SystemVerilog entre outras linguagens

# Facilidades oferecidas pela ferramenta

- Oferece diversas maneiras de encontrar erros de código.

# Facilidades oferecidas pela ferramenta

- Permite inserir características físicas reais do circuito digital no código.
  - Inserção de tempo de propagação de sinal

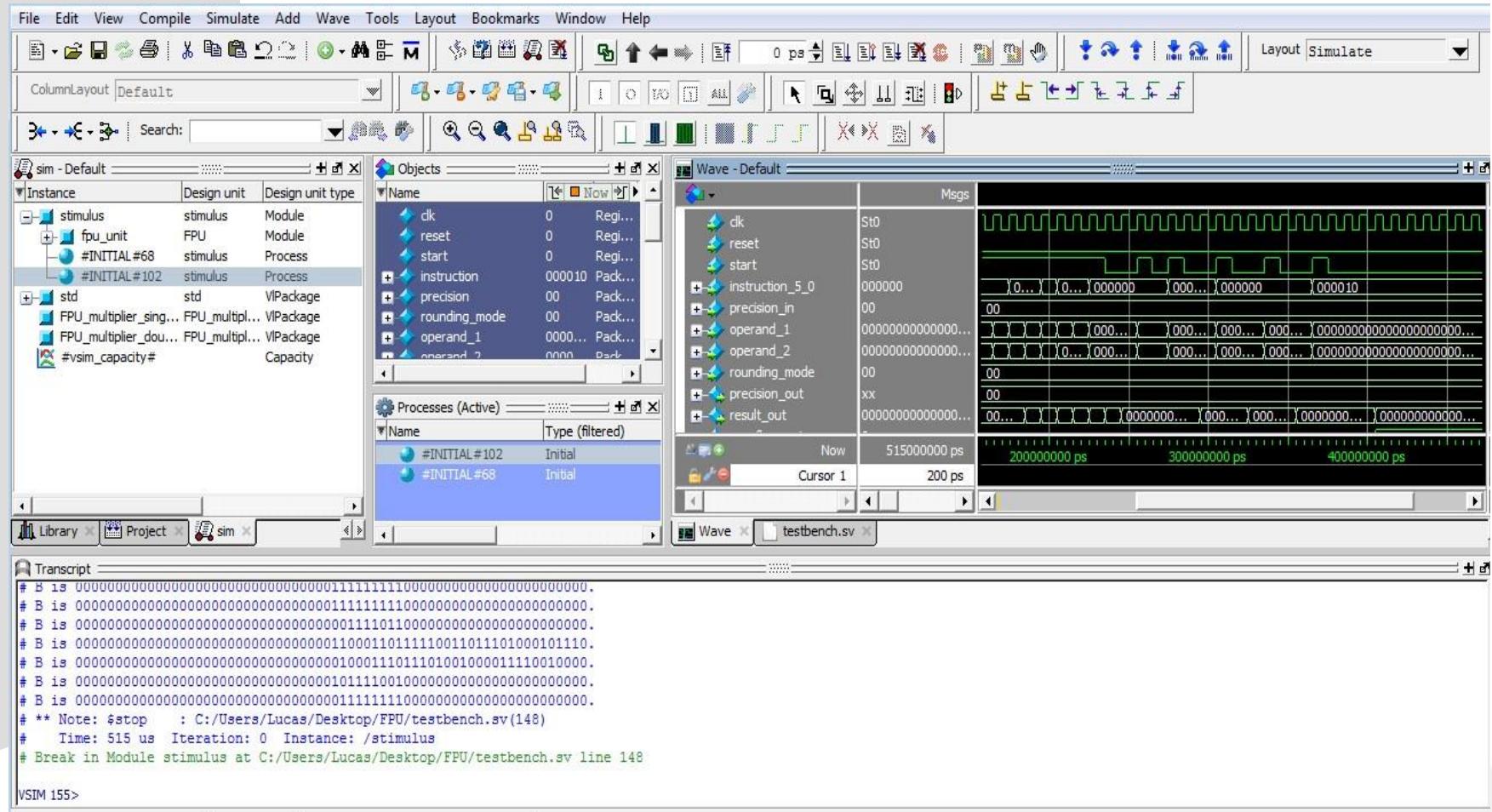
# Facilidades oferecidas pela ferramenta

- Manuseio por IDE gráfica ou scripts.

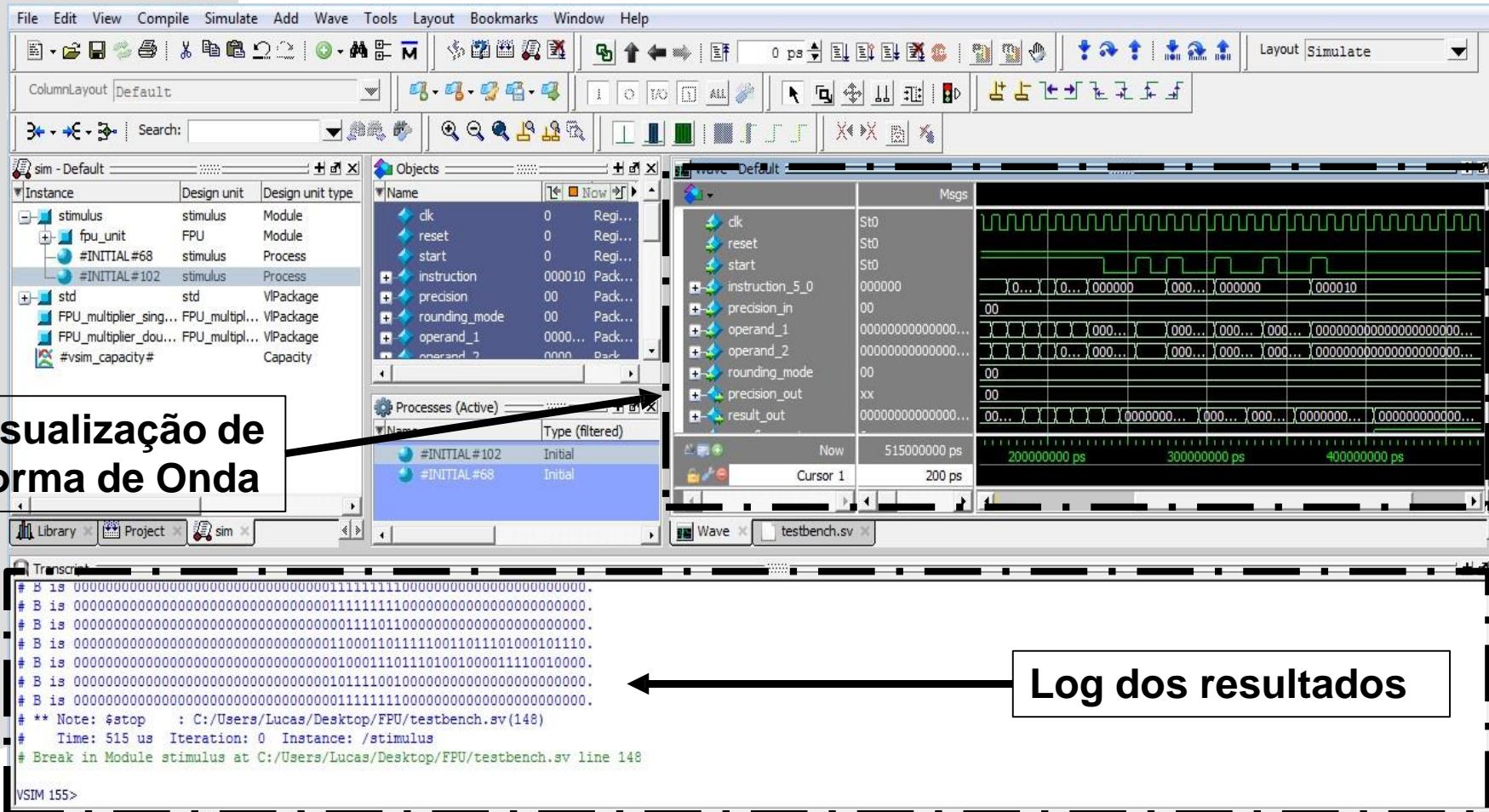
# Obtendo a ferramenta

- **Site da altera:**
  - <https://www.intel.com/content/www/us/en/programmable/downloads/download-center.html>
- **Nos computadores do CIn:**
  - Pesquisar por:  ModelSim - INTEL FPGA STARTER EDITION 10.6d

# Visão da Ferramenta

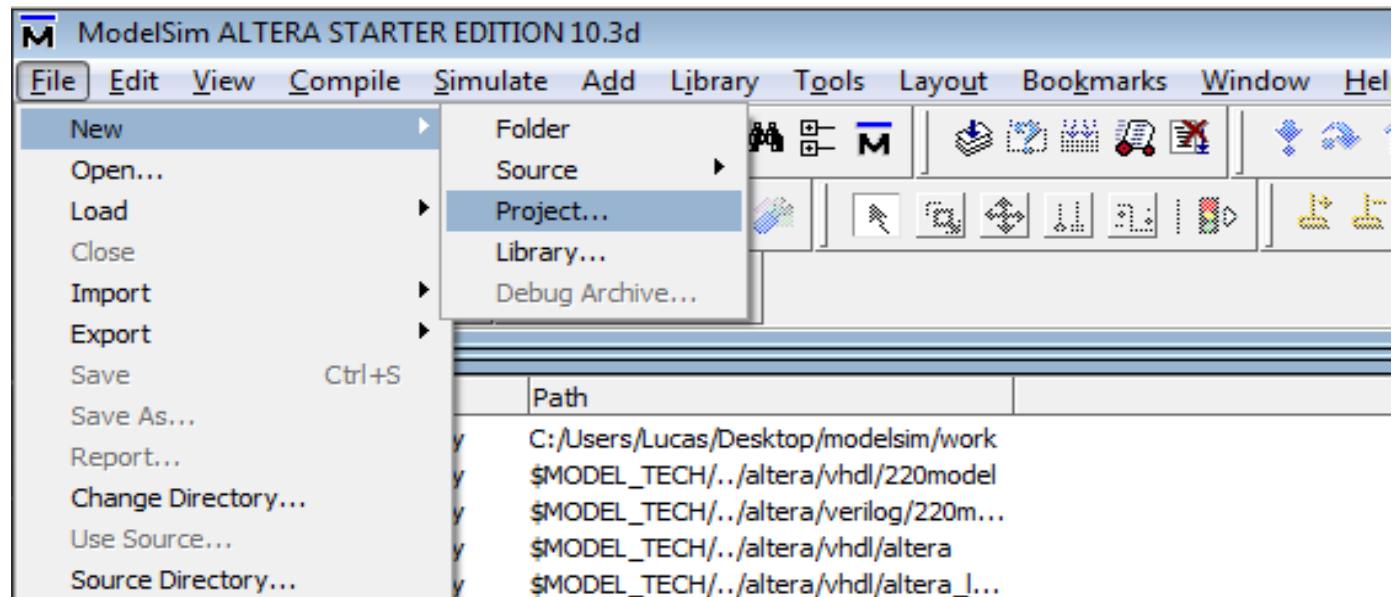


# Recursos importantes da Ferramenta



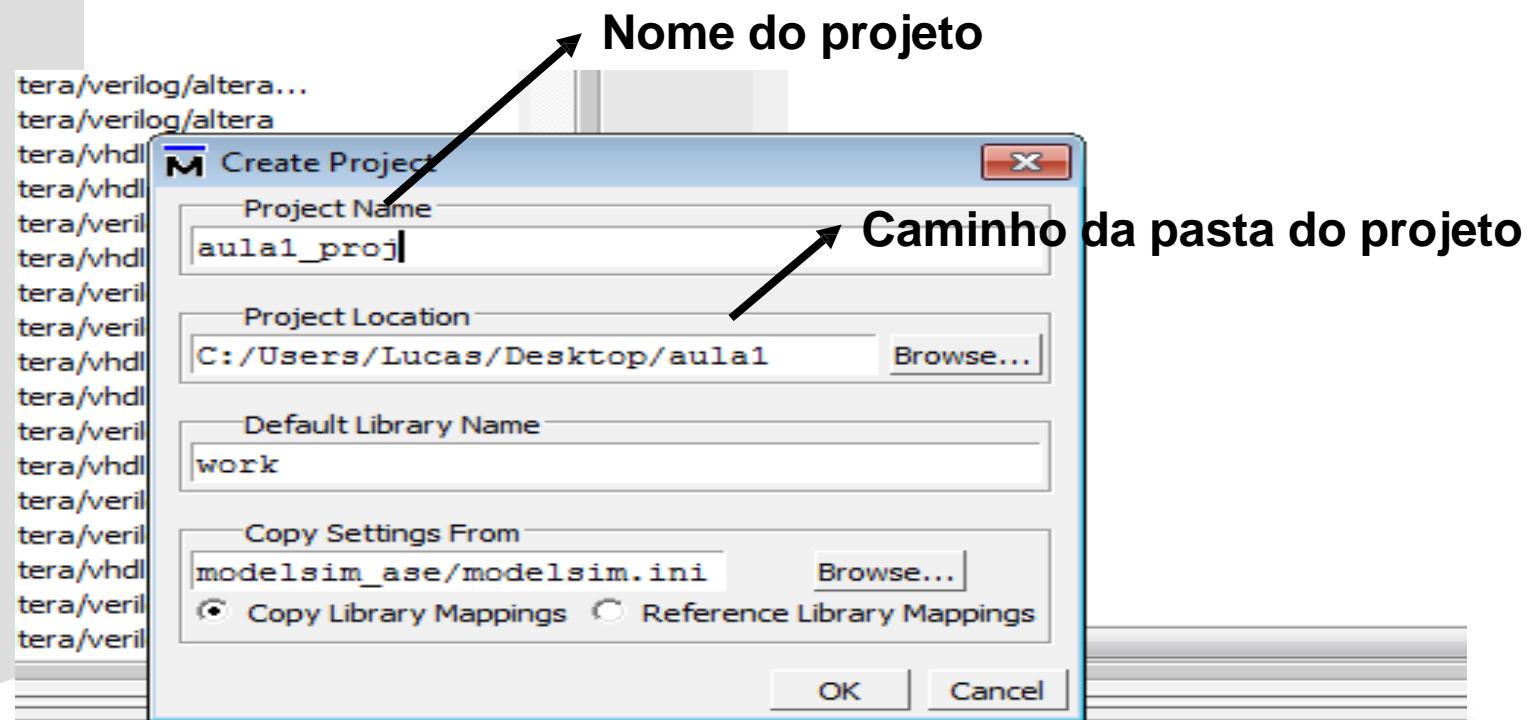
# Aprendendo a usar a ferramenta em etapas

## 1) Criando um projeto (vá em File > new > Project)



# Aprendendo a usar a ferramenta em etapas

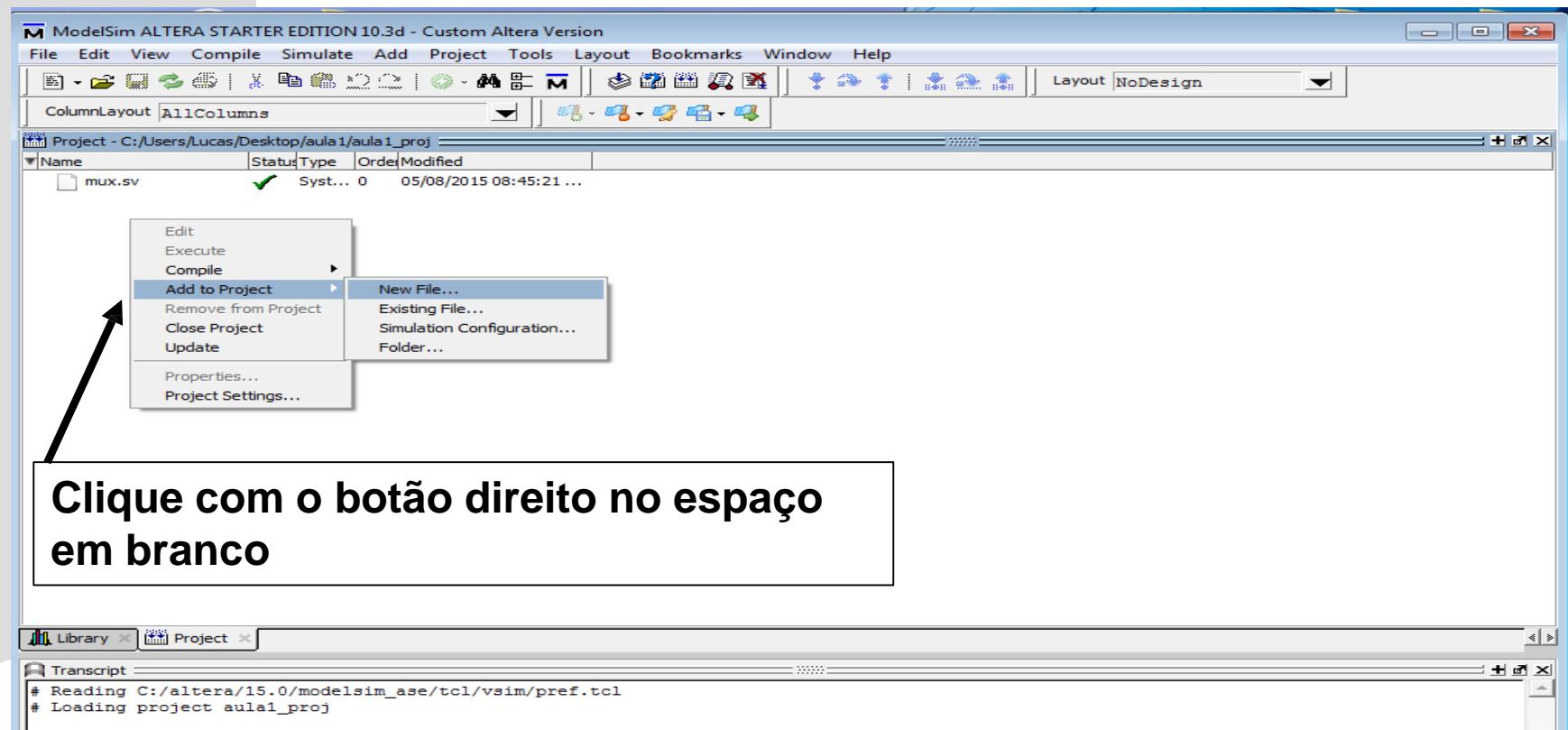
## 1) Criando um projeto



**Obs: Crie uma pasta antes**

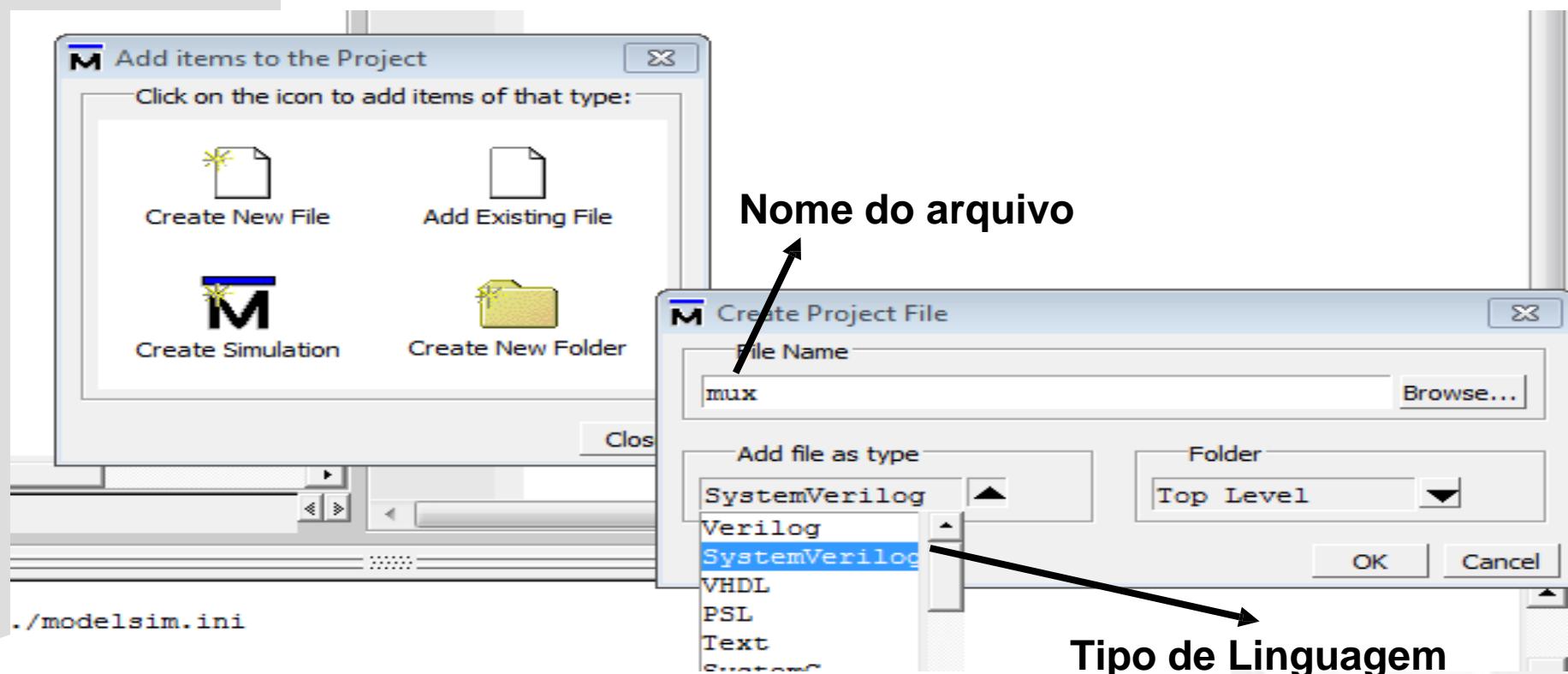
# Aprendendo a usar a ferramenta em etapas

## 2) Criando uma implementação em System Verilog



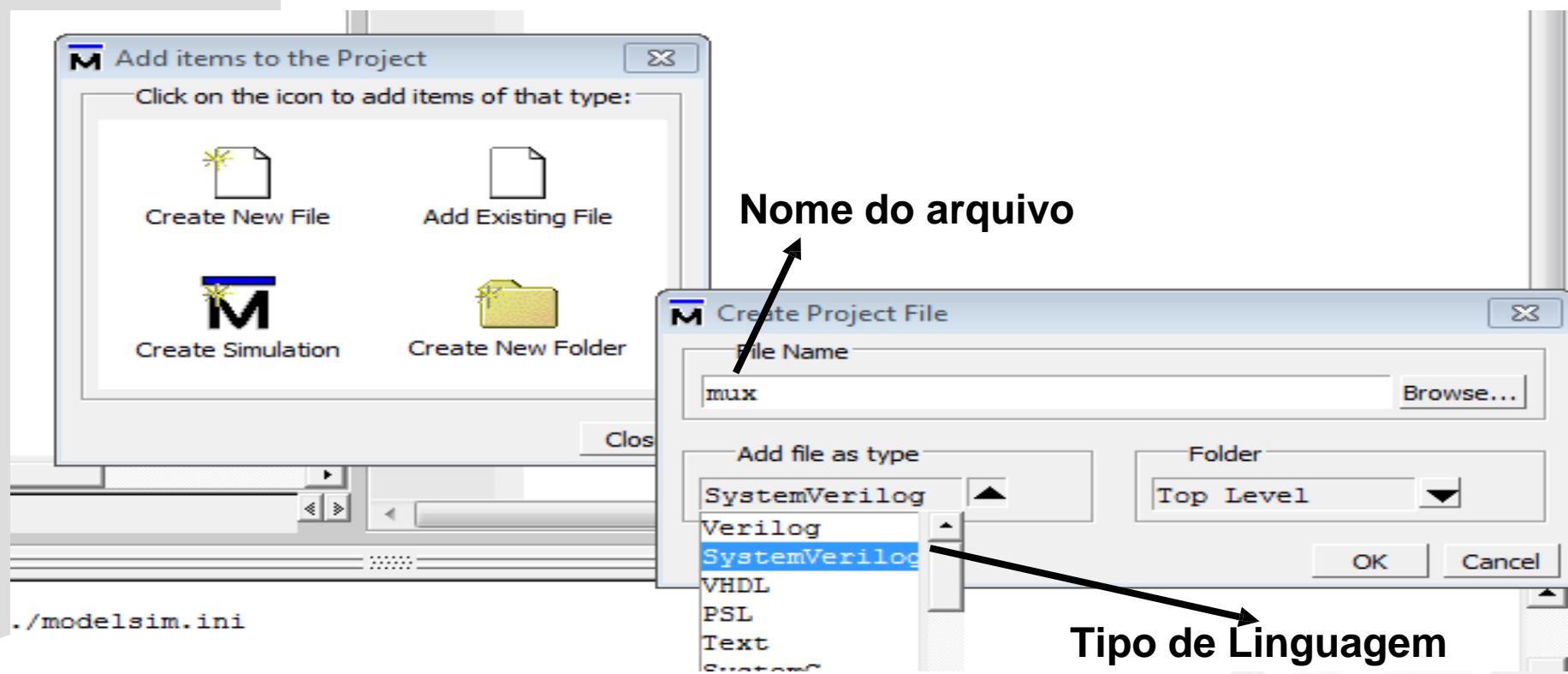
# Aprendendo a usar a ferramenta em etapas

## 2) Criando uma implementação em System Verilog



# Aprendendo a usar a ferramenta em etapas

## 2) Criando uma implementação em System Verilog



**Obs: O nome do arquivo é o mesmo nome do modulo que será implementado nesse arquivo**

# Aprendendo a usar a ferramenta em etapas

## 2) Criando uma implementação em System Verilog

The screenshot shows the ModelSim ALTERA STARTER EDITION 10.3d interface. The main window displays a System Verilog module named 'mux' with the following code:

```
module mux
    (output logic f,
     input logic a,b,sel);
    and #2 g1(f1,a,n_sel),
         g2(f2,b,sel);
    or #2 g3(f,f1,f2);
    not g4(n_sel,sel);
endmodule
```

The code editor has tabs for 'Project' and 'mux.sv'. The transcript window at the bottom shows the loading of project files:

```
# Reading C:/altera/15.0/modelsim_ase/tcl/vsim/pref.tcl
# Loading project teste
# reading C:/altera/15.0/modelsim_ase/win32aloem/..../modelsim.ini
# Loading project aula1
# reading C:/altera/15.0/modelsim_ase/win32aloem/..../modelsim.ini
```

# Aprendendo a usar a ferramenta em etapas

## 2) Criando uma implementação em System Verilog

ModelSim ALTERA STARTER EDITION 10.3d - Custom Altera Version

File Edit View Compile Simulate Add Source Tools Layout Bookmarks Window Help

ColumnLayout A11Columns

Project - C:/Users/Lucas/Desktop/aula 1/aula1\_proj

Name Status Type Order Modified

mux.sv Syst... 0 05/08/2015 08:18:16 ...

C:/Users/Lucas/Desktop/aula 1/mux.sv - Default

```
Ln#  
1 module mux  
2   (output logic f,  
3    input logic a,b,sel);  
4  
5   and #2 g1(f1,a,n_sel),  
6     g2(f2,b,sel);  
7  
8   or #2 g3(f,f1,f2);  
9   not g4(n_sel,sel);  
10  
11 endmodule  
12
```

Library Project

Transcript

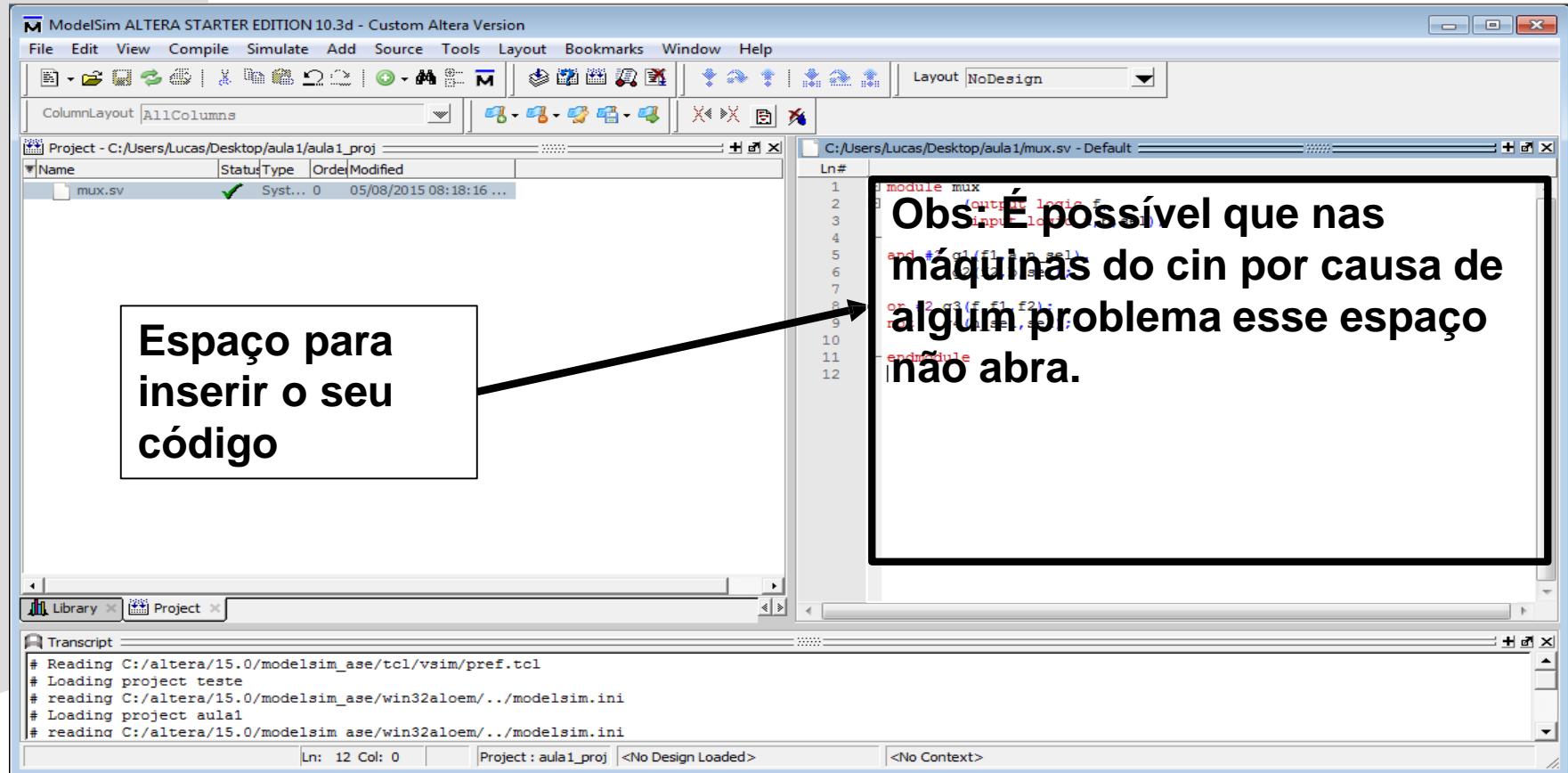
```
# Reading C:/altera/15.0/modelsim_ase/tcl/vsim/pref.tcl  
# Loading project teste  
# reading C:/altera/15.0/modelsim_ase/win32aloem/.../modelsim.ini  
# Loading project aula1  
# reading C:/altera/15.0/modelsim_ase/win32aloem/.../modelsim.ini
```

Ln: 12 Col: 0 Project: aula1\_proj <No Design Loaded> <No Context>

Espaço para inserir o seu código

# Aprendendo a usar a ferramenta em etapas

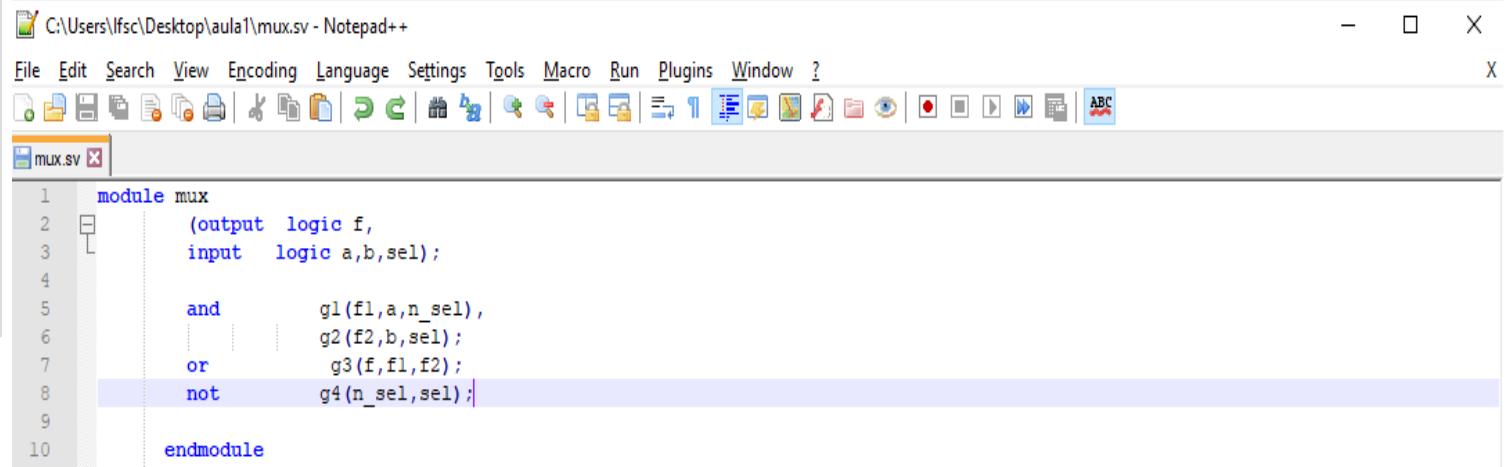
## 2) Criando uma implementação em System Verilog



# Aprendendo a usar a ferramenta em etapas

## 2) Criando uma implementação em System Verilog

Mas, não tem problema. Você pode usar o notepad++ ou qualquer ferramenta de edição de código em SystemVerilog.



The screenshot shows a Notepad++ window titled "mux.sv". The code is a SystemVerilog module definition for a multiplexer (MUX). The module has one output logic variable "f" and three input logic variables "a", "b", and "n\_sel". The implementation uses four gates: AND, OR, NOT, and an unlabeled gate (likely an XOR or XNOR). The code is as follows:

```
1 module mux
2   (output logic f,
3   input  logic a,b,n_sel);
4
5   and      g1(f1,a,n_sel),
6           g2(f2,b,n_sel);
7   or       g3(f,f1,f2);
8   not      g4(n_sel,sel);
9
10  endmodule
```

# Aprendendo a usar a ferramenta em etapas

## 2) Criando uma implementação em System Verilog

```
module mux  
  (output logic f,  
   input logic a,b,sel);  
  
  and      g1(f1,a,n_sel),  
          g2(f2,b,sel);  
  or       g3(f,f1,f2);  
  not      g4(n_sel,sel);  
  
endmodule: mux
```

Copie este  
código  
exemplo e cole  
no arquivo  
mux.sv

# Aprendendo a usar a ferramenta em etapas

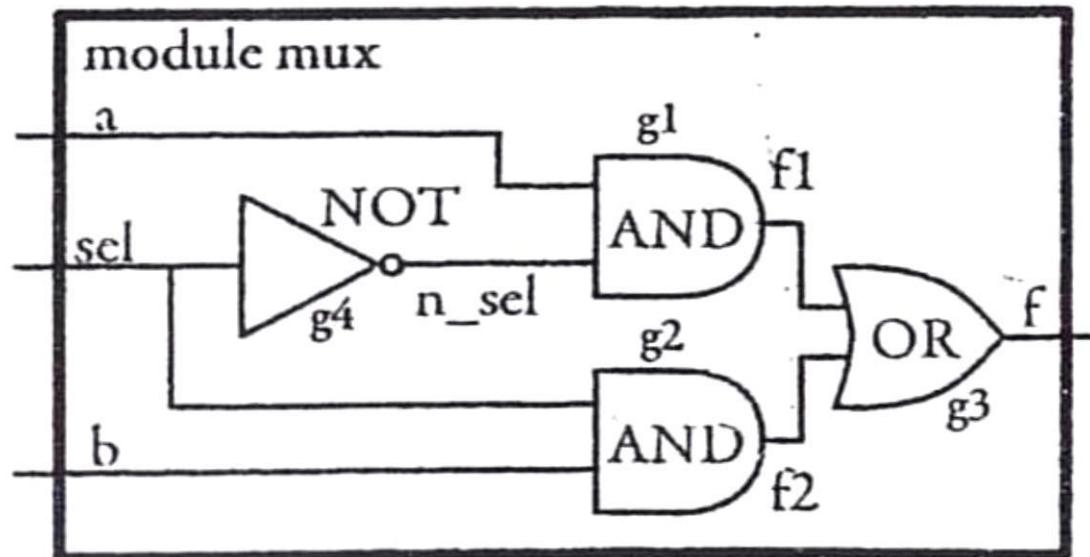
## 2) Criando uma implementação System Verilog

```
Nome do módulo  
Portas do módulo  
module mux  
(output logic f, input logic a,b,sel);  
  
and g1(f1,a,n_sel),  
or g2(f2,b,sel);  
not g3(f,f1,f2);  
not g4(n_sel,sel);  
  
Portas (gates) primitivas  
são instanciadas e  
nomeadas  
endmodule
```

Conexões elétricas são nomeadas

# Aprendendo a usar a ferramenta em etapas

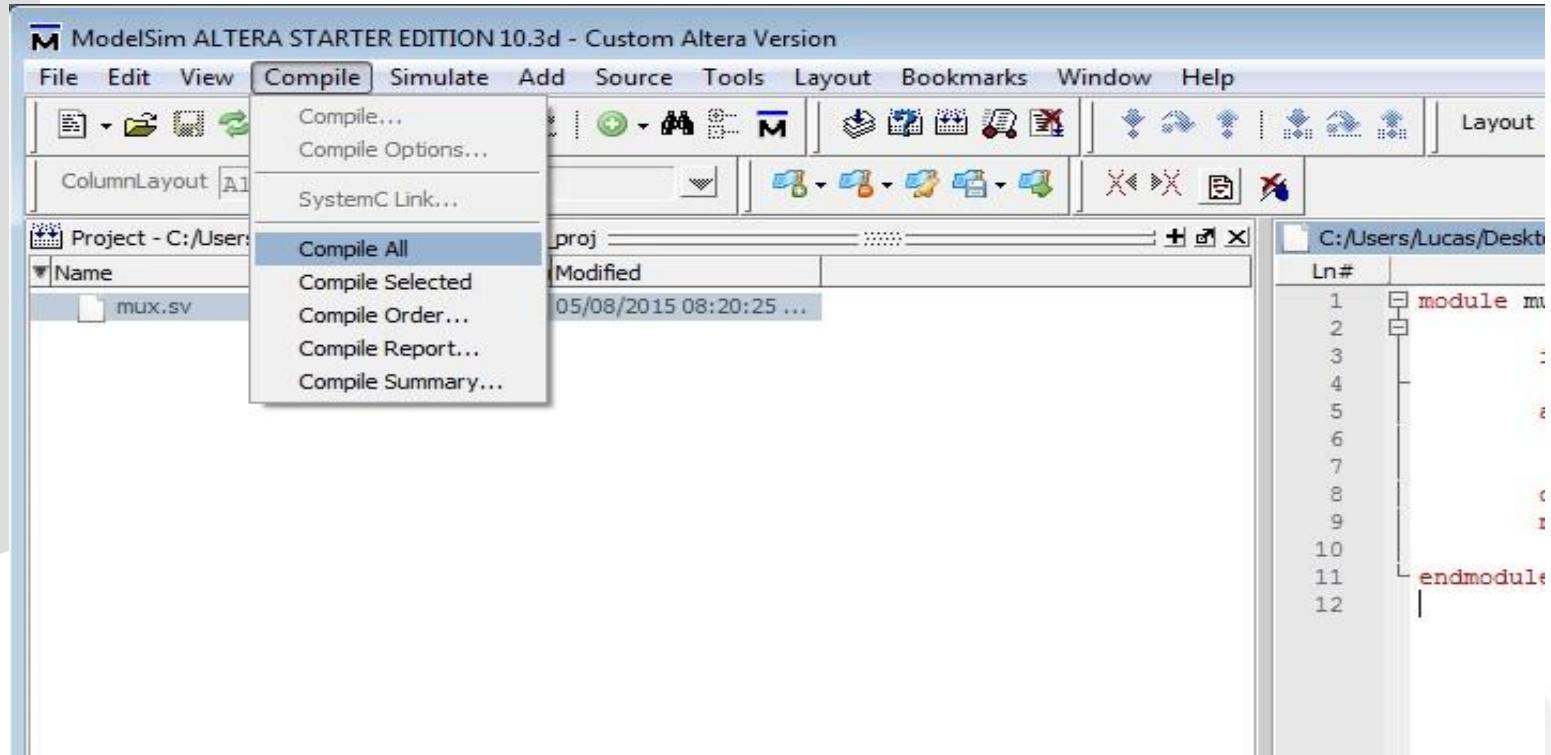
Visão em nível de portas lógicas desta implementação



$$f = a \cdot \text{sel}' + b \cdot \text{sel}$$

# Aprendendo a usar a ferramenta em etapas

## 3) Compilando todos os códigos do projeto (vá em Compile > compile all)



# Aprendendo a usar a ferramenta em etapas

O que falta para podermos simular o código

?

# Aprendendo a usar a ferramenta em etapas

O que falta para podermos simular o código

?

**Gerar algum dado de entrada**

# Aprendendo a usar a ferramenta em etapas

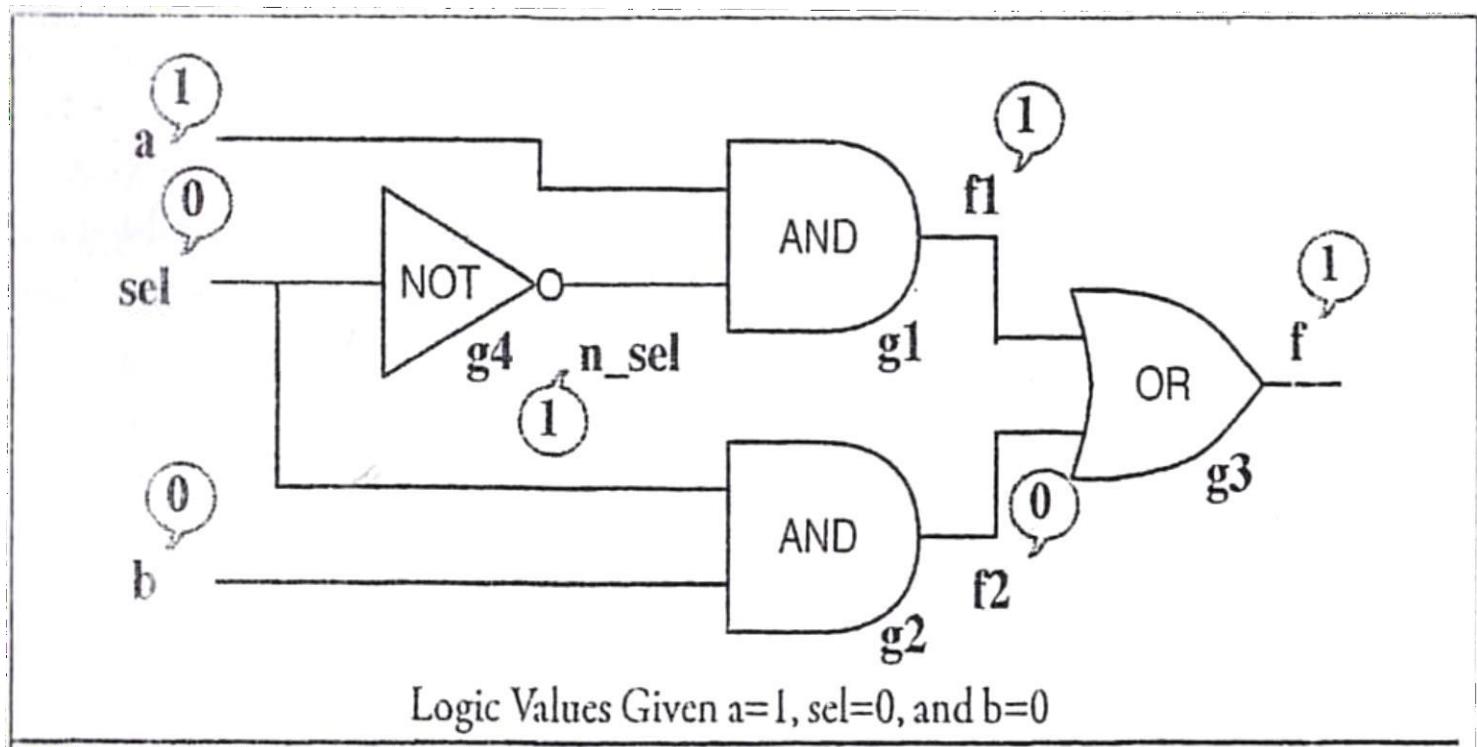
O que falta para podermos simular o código

?

**...e em seguida verificar a resposta do módulo**

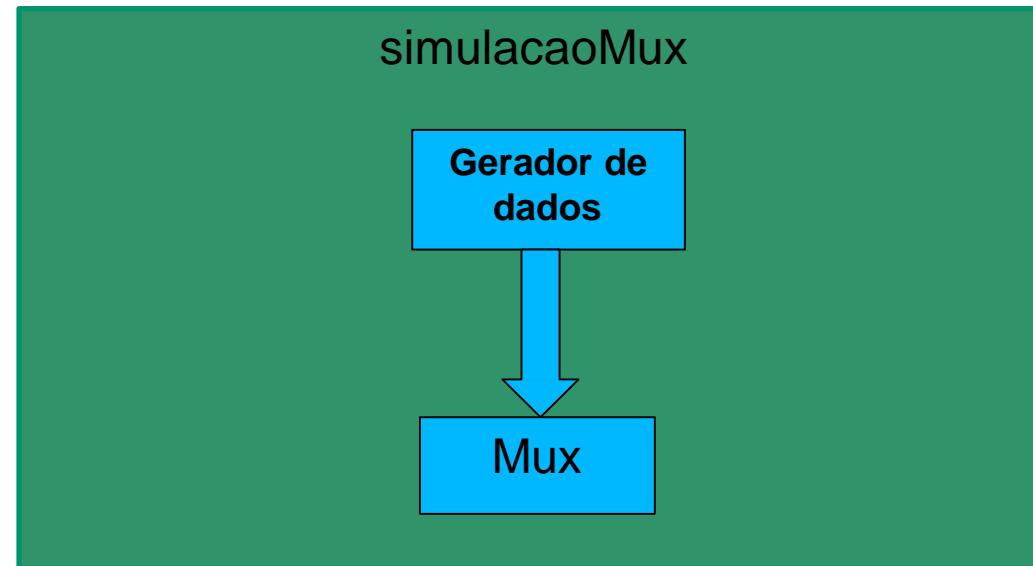
# Aprendendo a usar a ferramenta em etapas

Exemplo do comportamento do modulo para um dado de entrada



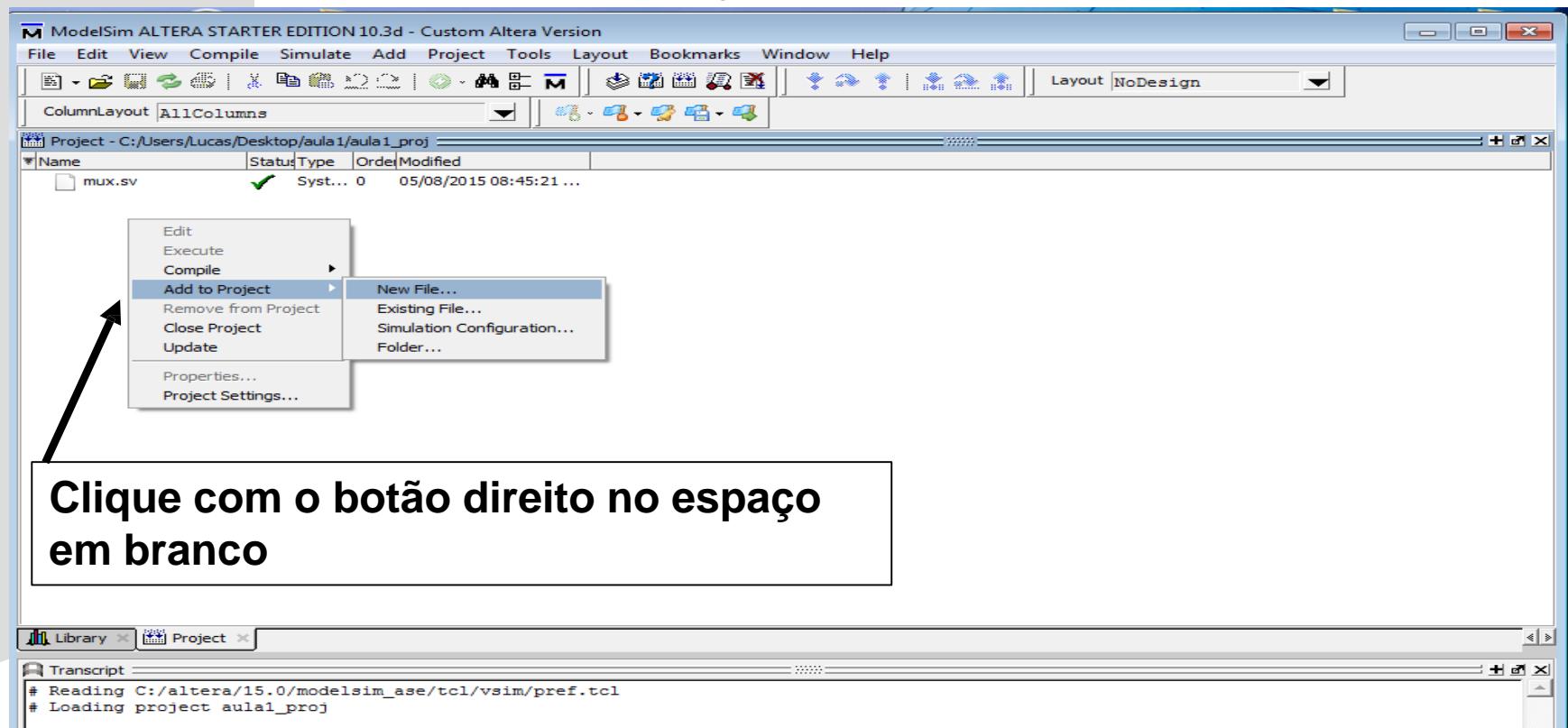
# Aprendendo a usar a ferramenta em etapas

Ilustração do modulo gerador de sinal com o módulo que será testado



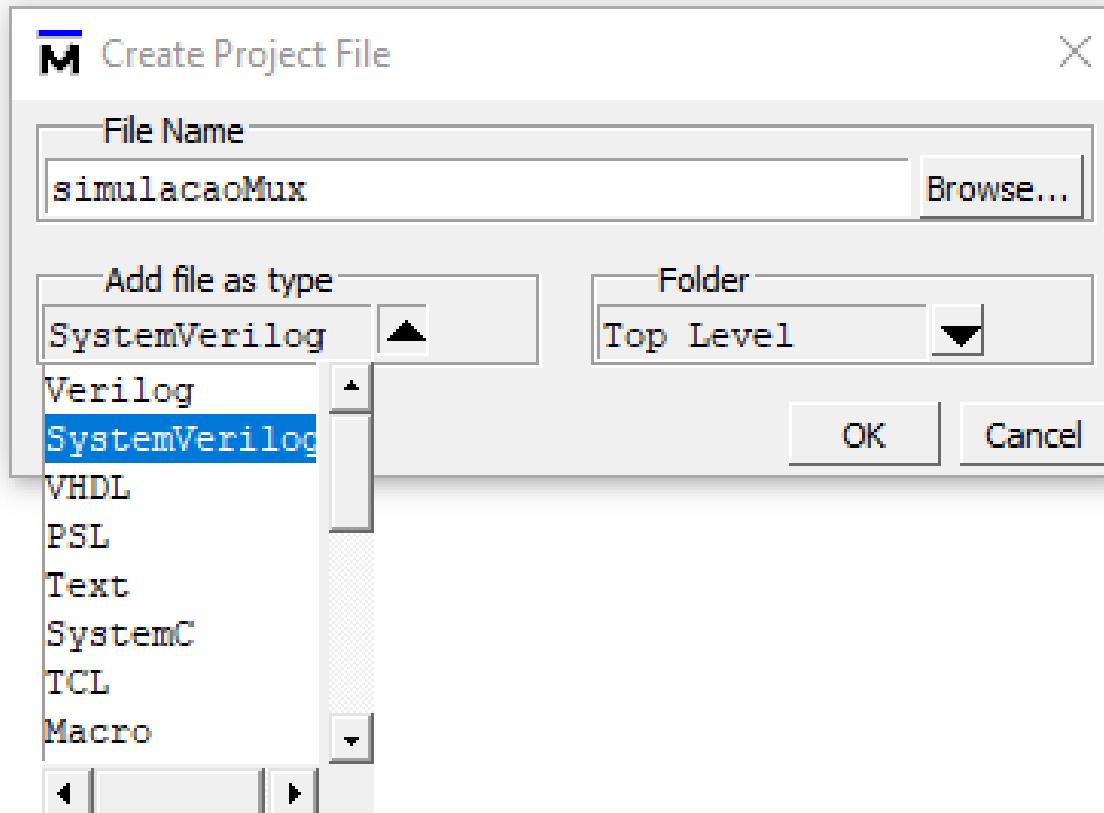
# Aprendendo a usar a ferramenta em etapas

4) Adicionando um novo arquivo .sv que contém o código para geração de dados de entrada



# Criando um módulo gerador

## 2) Criando uma implementação em System Verilog



# Criando um módulo gerador

4) Adicionando um novo arquivo .sv que contem o código para geração de dados

```
module simulacaoMux;  
    logic [2:0]count;  
    logic muxOut;
```

Cria uma instância do módulo **mux** e chamando de **dut**

```
        mux dut(.f(muxOut), .a(count[2]), .b(count[1]), .sel(count[0]));
```

```
initial begin  
    $monitor($time, "a b sel = %b, muxOut = %b", count, muxOut);  
    for(count = 0; count != 3'b111; count++) #10;  
    #10 $stop;  
end
```

```
endmodule: simulacaoMux
```

# Criando um módulo gerador

## 4) Adicionando um novo arquivo .sv que contem o código para geração de dados

```
module simulacaoMux;  
    logic [2:0]count;  
    logic muxOut;  
  
    mux dut(.f(muxOut), .a(count[2]), .b(count[1]), .sel(count[0]));  
  
    initial begin  
        $monitor($time, "a b sel = %b, muxOut = %b", count, muxOut);  
        for(count = 0; count != 3'b111; count++) #10;  
        #10 $stop;  
    end  
  
endmodule: simulacaoMux
```

Interliga os sinais externos com as entradas e saídas do módulo mux

# Criando um módulo gerador

## Entendendo o código de geração de dados

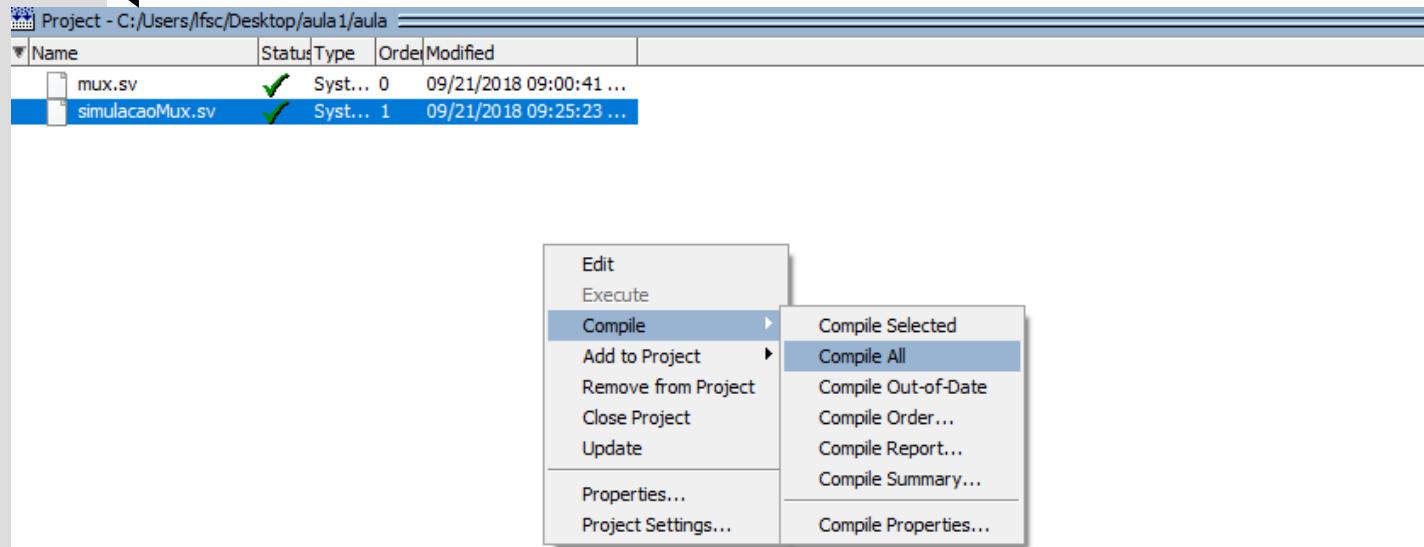
- **Initial**
  - É executado apenas uma vez no início da simulação. É tipicamente usado para inicializar variáveis e especificar formas de onda de sinais durante a simulação
- **\$monitor**
  - Tira um print dos dados toda vez que um de seus parâmetros mudam
- **#10**
  - Solicita que o simulador pare sua execução por 10 unidades de tempo.
- **\$stop**
  - Termina a simulação

```
initial begin
    $monitor($time,"a b sel = %b, muxOut = %b",
    count, muxOut);
    for(count = 0; count != 3'b111; count++) #10;
    #10 $stop;
end
```

# Criando um módulo gerador

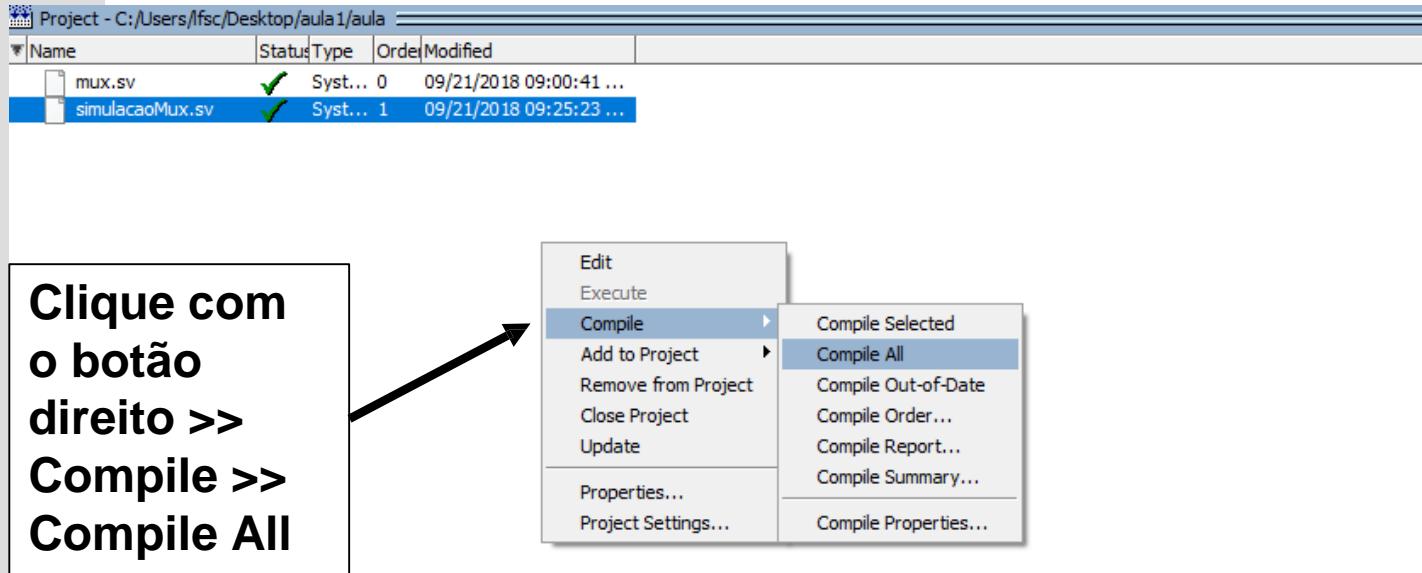
Dê um  
compile All

4) Adicionando um novo arquivo .sv que contem o código para geração de dados



# Criando um módulo gerador

4) Adicionando um novo arquivo .sv que contém o código para geração de dados



# Criando um módulo gerador

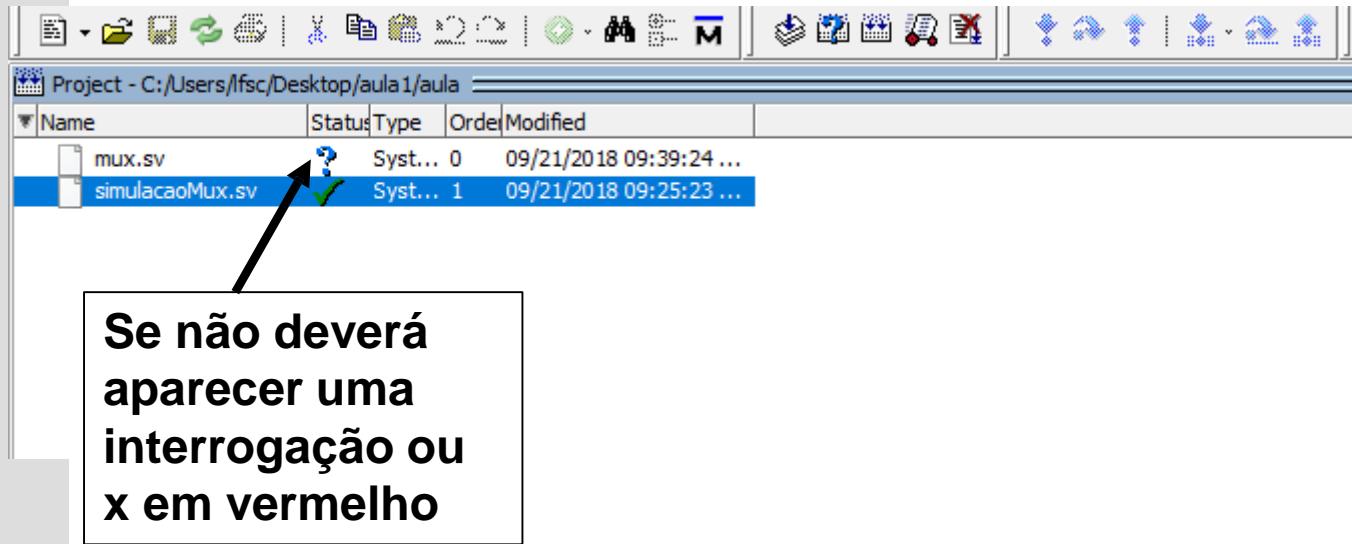
4) Adicionando um novo arquivo .sv que contem o código para geração de dados

Name	Status	Type	Order	Modified
mux.sv	✓	Syst...	0	09/21/2018 09:00:41 ...
simulacaoMux.sv	✓	Syst...	1	09/21/2018 09:25:23 ...

Se a sintaxe dos dois códigos estiverem corretas o *status* de ambos deverão aparecer verdes

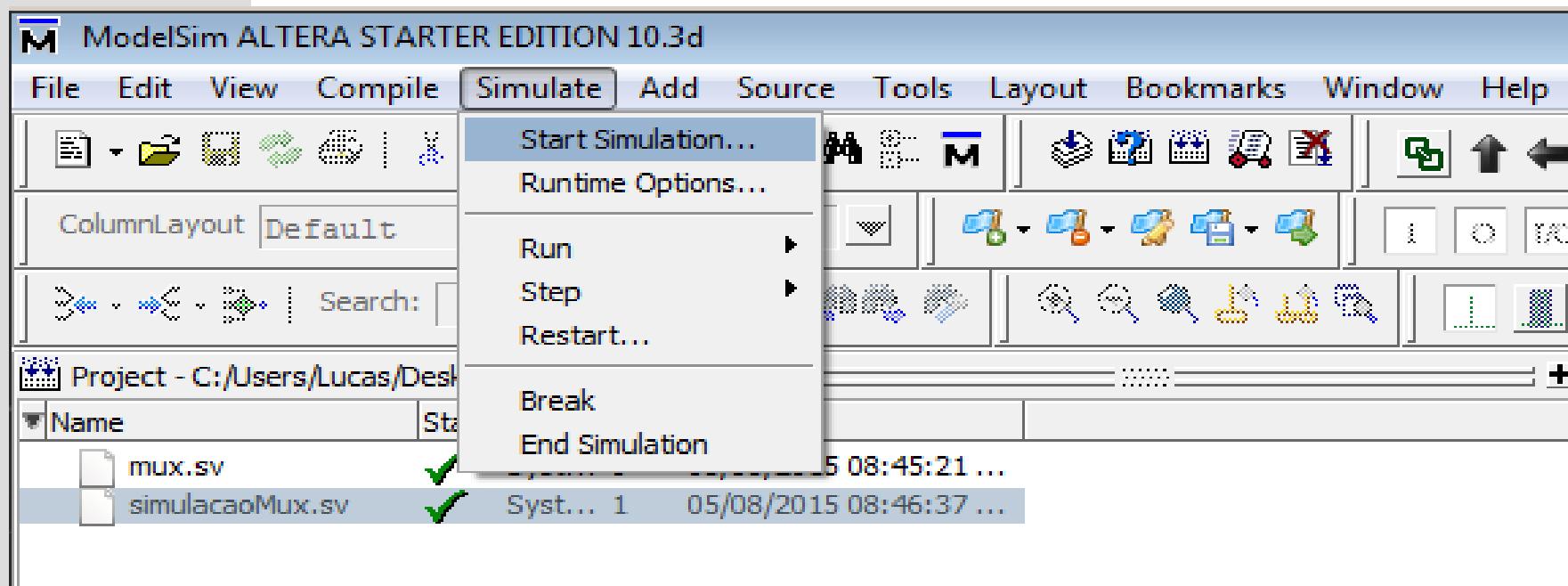
# Criando um módulo gerador

4) Adicionando um novo arquivo .sv que contém o código para geração de dados



# Simulando o módulo com o gerador

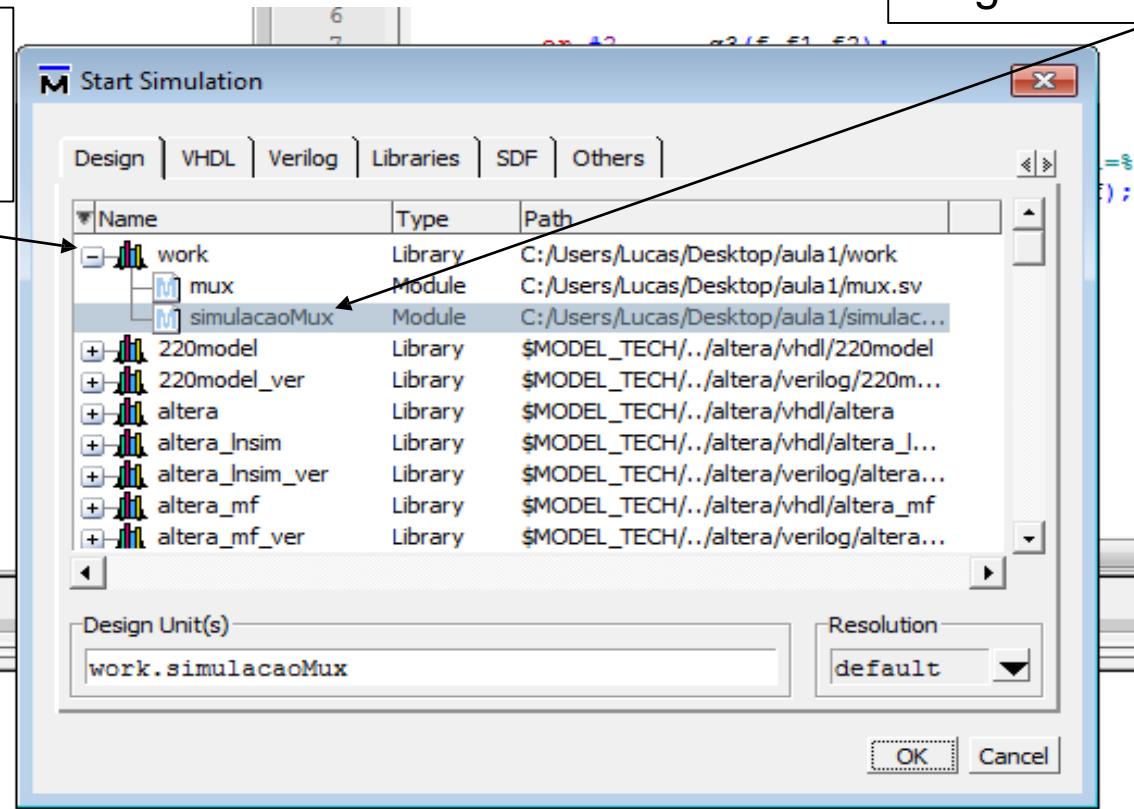
## 5) Simulando o código



# Simulando o módulo com o gerador

## 5) Simulando o código

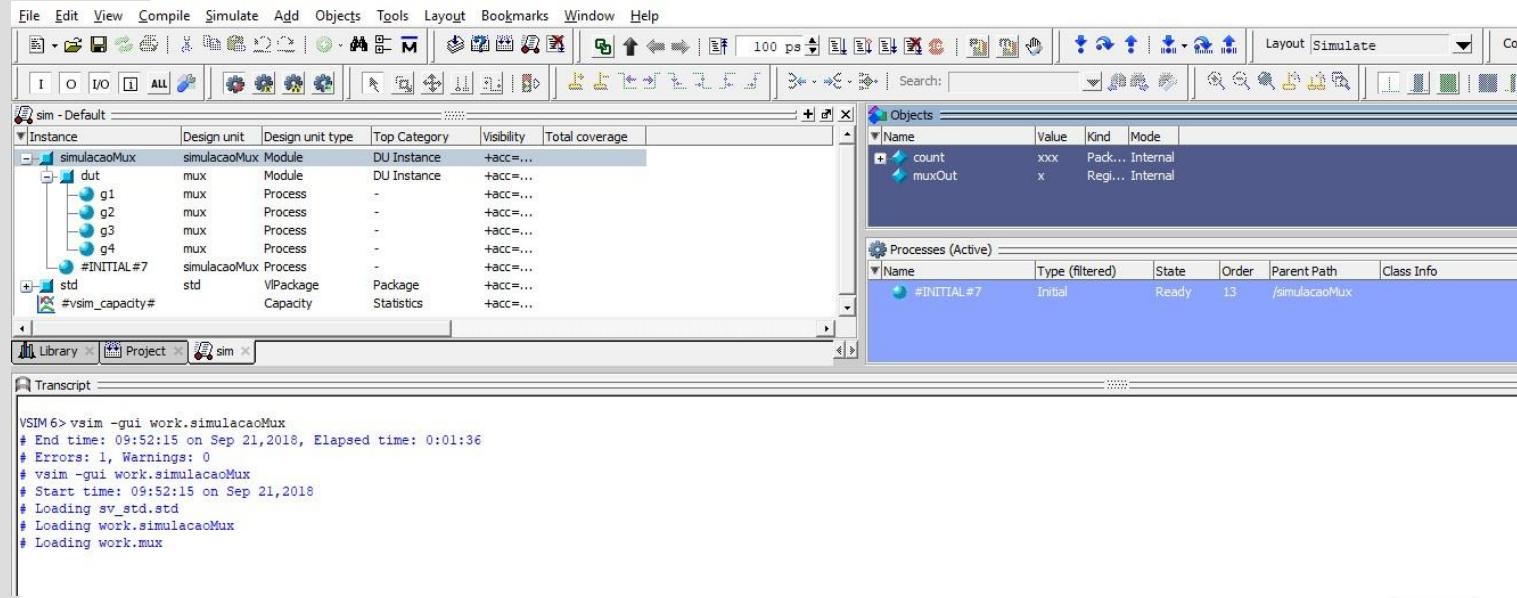
Ela está  
localizada no  
diretório work



Dica: É o modulo top que chama todos os outros módulos

# Simulando o módulo com o gerador

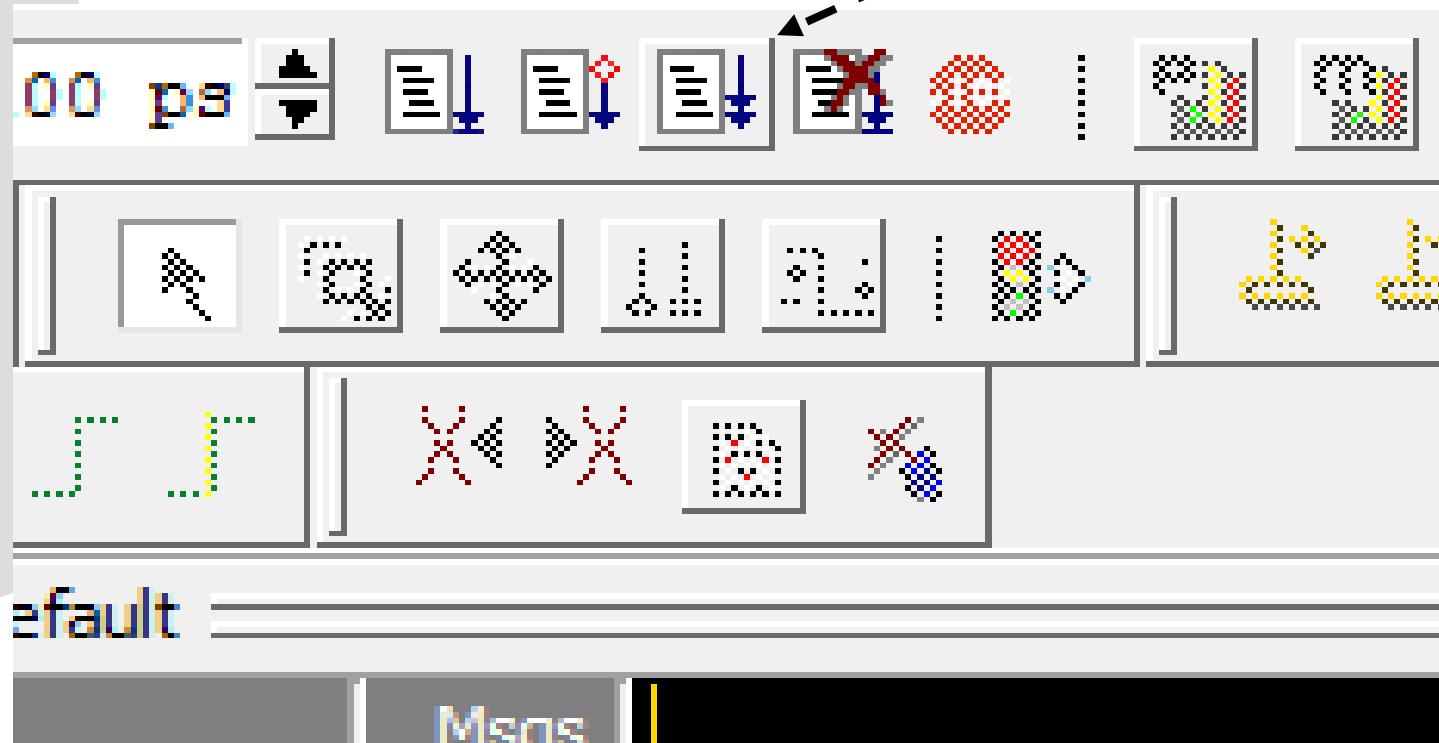
## 5) Simulando o código



# Simulando o módulo com o gerador

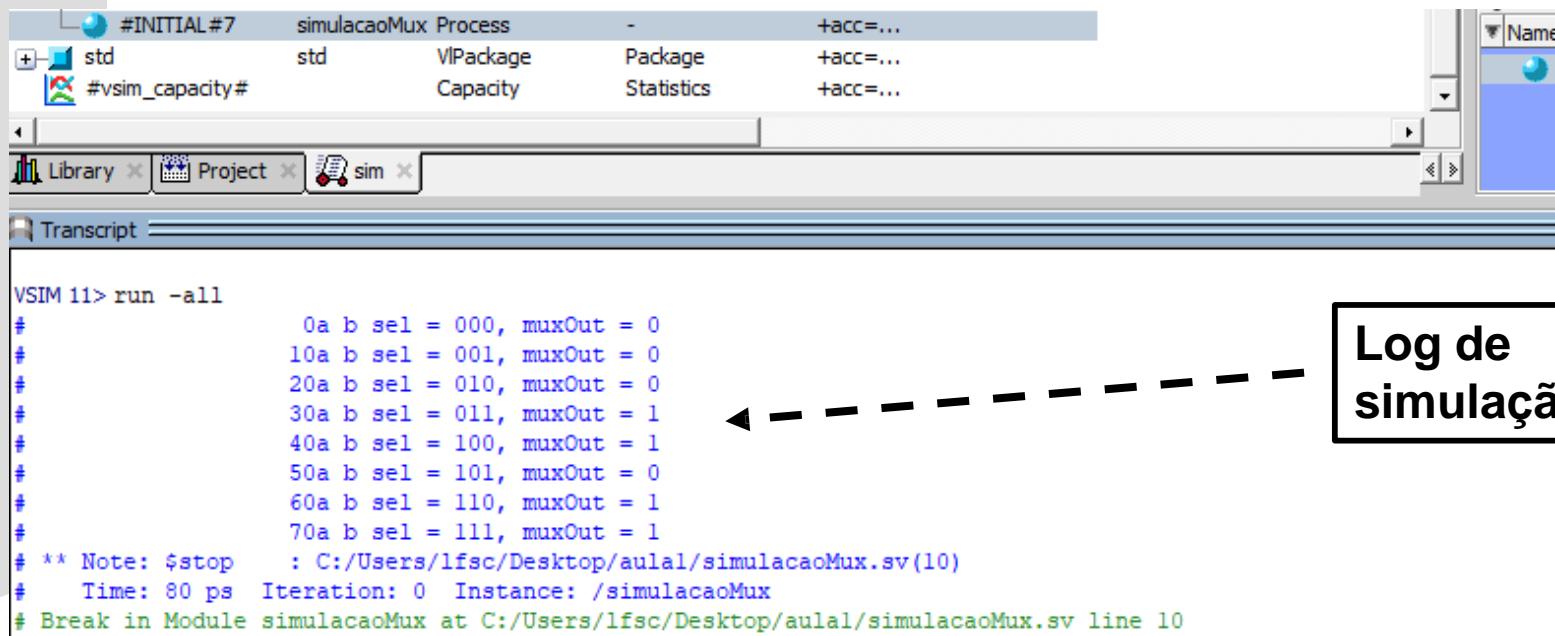
## 5) Simulando o código

Clicar  
(Run -all)



# Simulando o módulo com o gerador

## 5) Simulando o código



The screenshot shows the ModelSim simulation environment. The top part displays a tree view of the project structure under the root '#INITIAL#7'. The 'simulacaoMux' item is expanded, showing 'std' (VI Package), 'Capacity' (Statistics), and 'sim' (Process). Below the tree are tabs for 'Library', 'Project', and 'sim'. The bottom half of the window is a 'Transcript' window containing the following text:

```
VSIM 11> run -all
#
#          0a b sel = 000, muxOut = 0
#          10a b sel = 001, muxOut = 0
#          20a b sel = 010, muxOut = 0
#          30a b sel = 011, muxOut = 1
#          40a b sel = 100, muxOut = 1
#          50a b sel = 101, muxOut = 0
#          60a b sel = 110, muxOut = 1
#          70a b sel = 111, muxOut = 1
# ** Note: $stop : C:/Users/lfsc/Desktop/aulal/simulacaoMux.sv(10)
#   Time: 80 ps Iteration: 0 Instance: /simulacaoMux
# Break in Module simulacaoMux at C:/Users/lfsc/Desktop/aulal/simulacaoMux.sv line 10
```

A dashed arrow points from the text 'Log de simulação' to the transcript window.

**Log de simulação**

# Simulando o módulo com o gerador

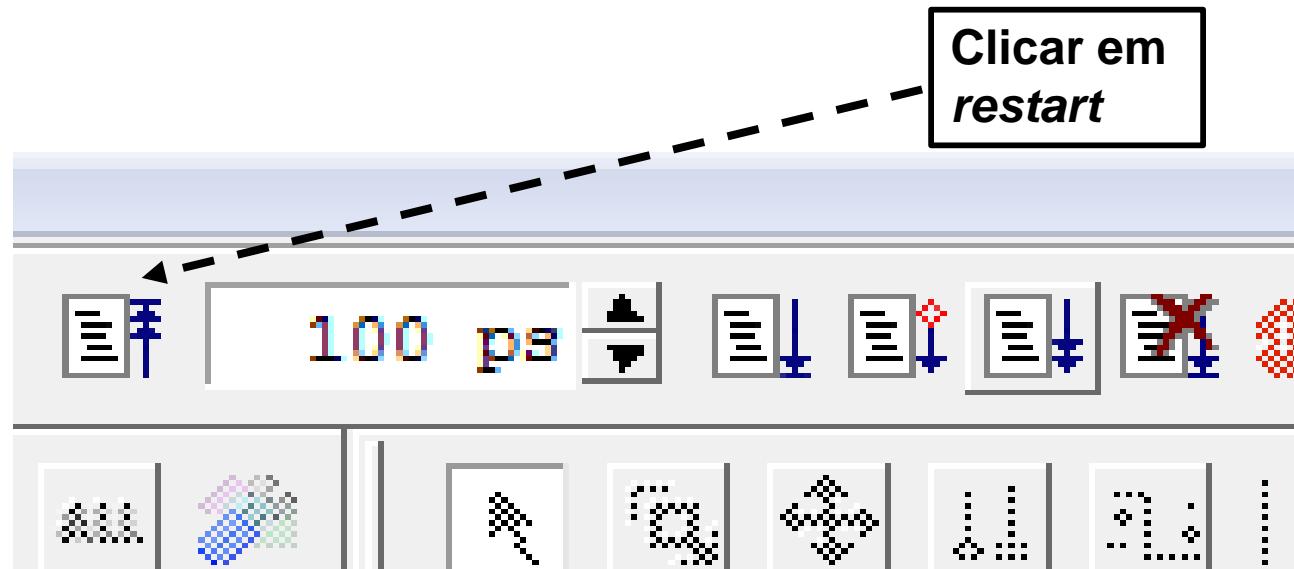
OBS: Sempre que você alterar o seu código você precisará compilar todos os arquivos modificados e simular novamente.

# Simulando o módulo com o gerador

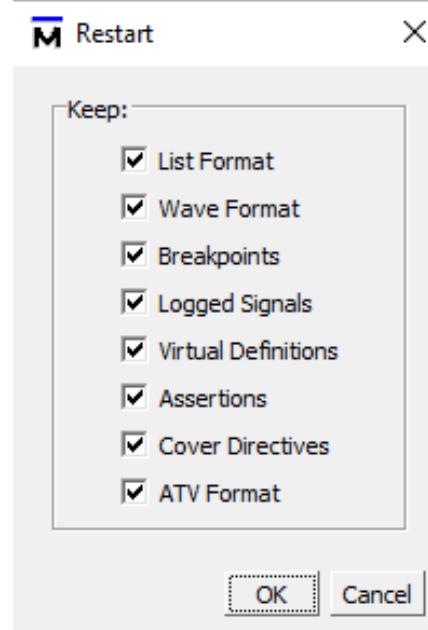
OBS: Se você alterar o arquivo e não compilar e em seguida simular, a simulação ocorrerá baseada na última compilação válida.

# Simulando com forma de onda

6) Repetindo a simulação e inserindo forma de onda (Waveform)



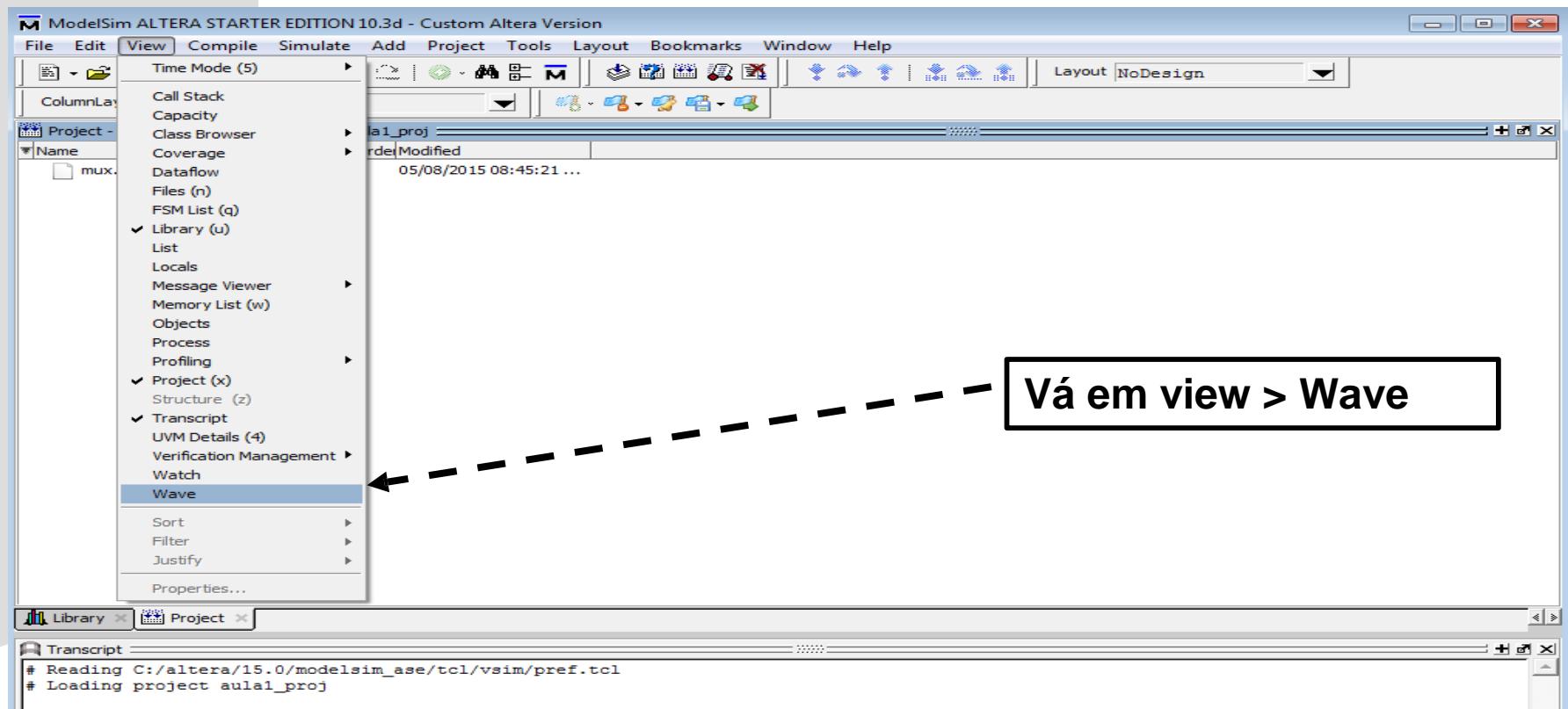
# Simulando com forma de onda



Clicar em “OK”

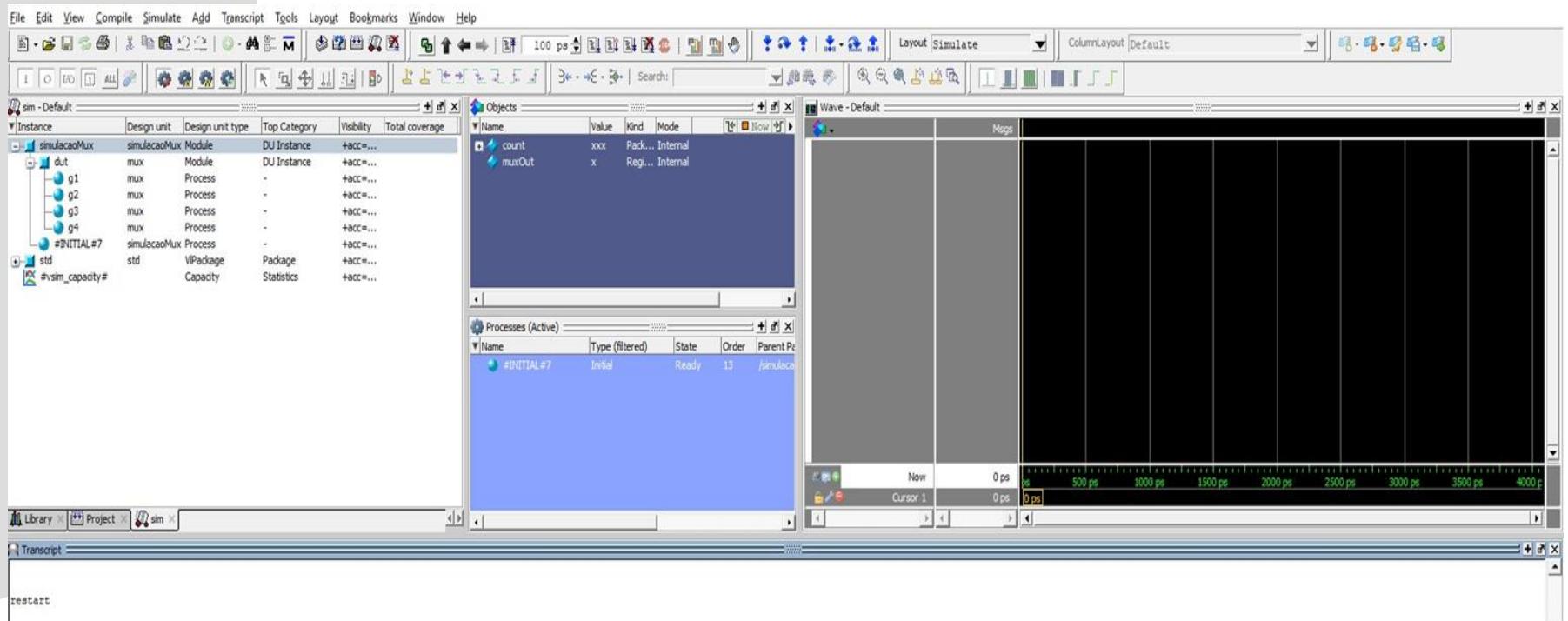
# Simulando com forma de onda

## 6) Repetindo a simulação e inserindo forma de onda



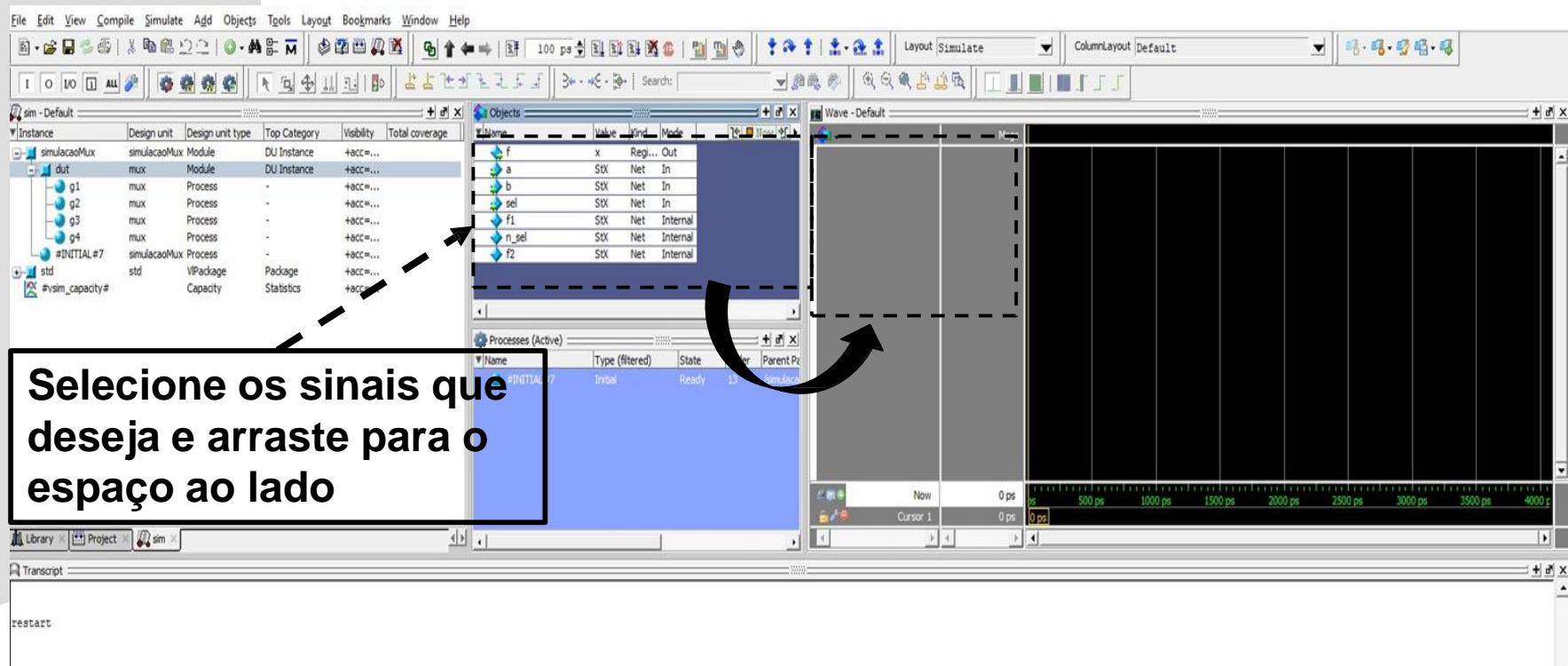
# Simulando com forma de onda

## 6) Repetindo a simulação e inserindo forma de onda



# Simulando com forma de onda

## 6) Repetindo a simulação e inserindo forma de onda



# Simulando com forma de onda

## 6) Repetindo a simulação e inserindo forma de onda

The screenshot shows the ModelSim simulation environment. The top menu bar includes File, Edit, View, Compile, Simulate, Add, Transcript, Tools, Layout, Bookmarks, Window, and Help. The main window has several panes:

- Objects** pane: Shows a tree view of the design units. Under **sim - Default**, there is a **dut** module containing processes **g1**, **g2**, **g3**, **g4**, and a **sel** process. There are also **mux** and **std** components.
- Wave - Default** pane: Displays waveforms for signals **f**, **a**, **b**, **sel**, **f1**, **n\_sel**, and **f2**. The time scale ranges from 0 ps to 4000 ps.
- Processes (Active)** pane: Shows a single process named **#INITIAL#7** in the Initial state.
- Script** pane: Contains the command history:

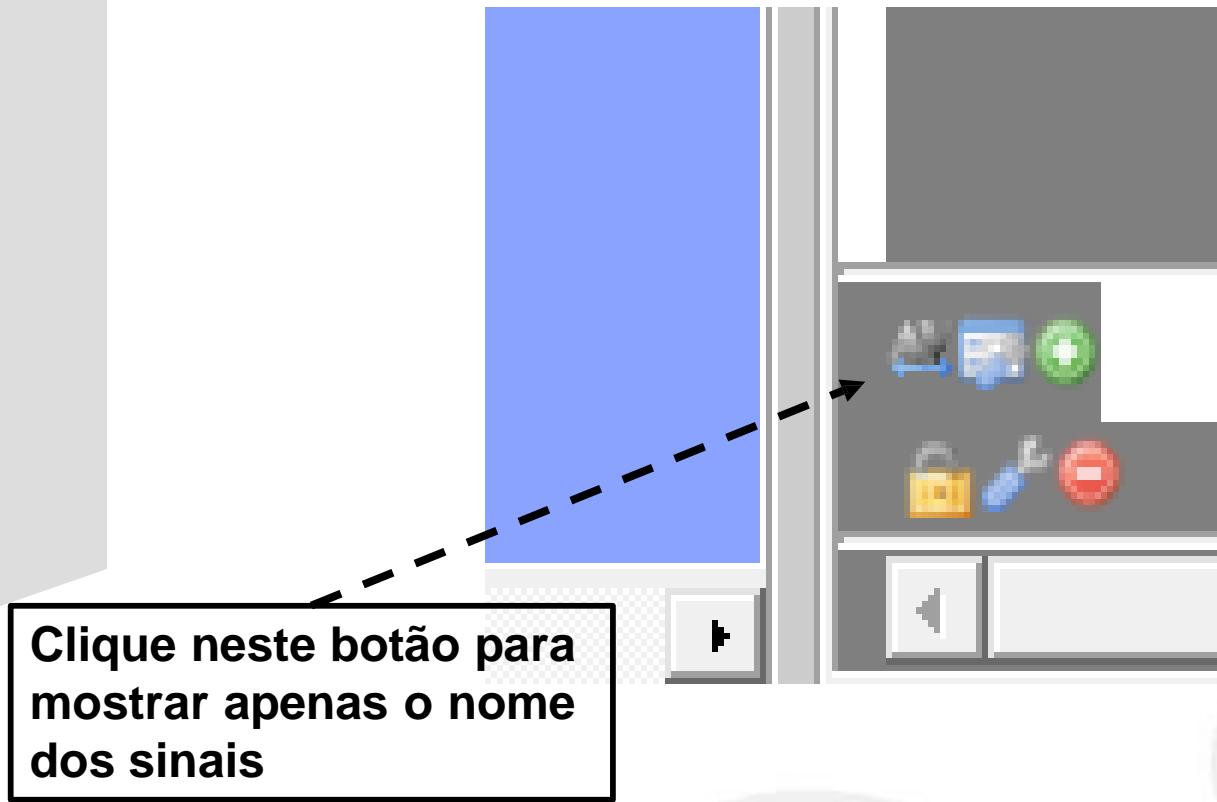
```
VSIM 13>
add wave -position end sim:/simulacaoMux/dut/f
add wave -position end sim:/simulacaoMux/dut/a
add wave -position end sim:/simulacaoMux/dut/b
add wave -position end sim:/simulacaoMux/dut/sel
add wave -position end sim:/simulacaoMux/dut/f1
add wave -position end sim:/simulacaoMux/dut/n_sel
add wave -position end sim:/simulacaoMux/dut/f2
```

A dashed arrow points from a callout box at the bottom to the Waveform pane.

**Os sinais devem  
aparecer neste espaço**

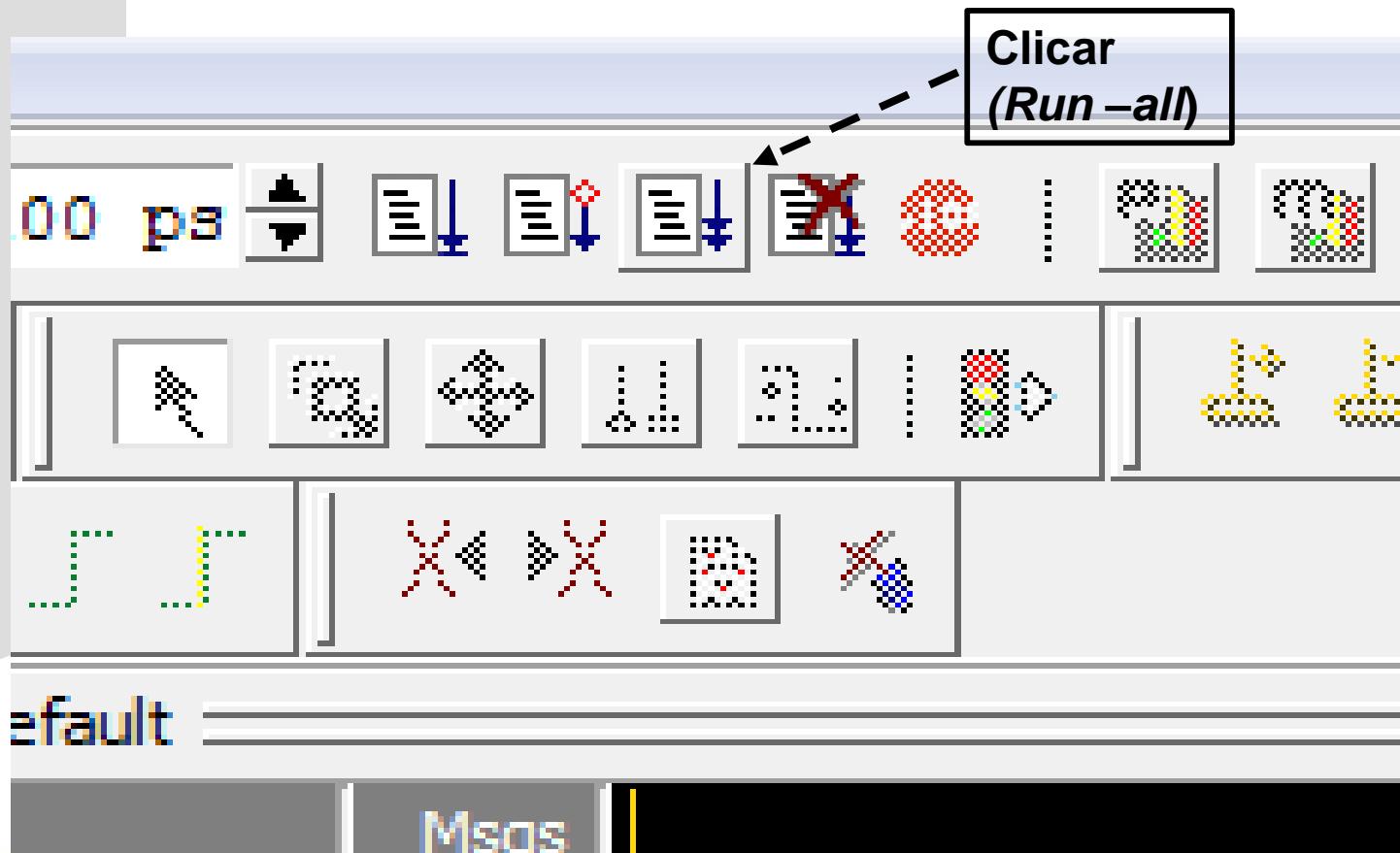
# Simulando com forma de onda

6) Repetindo a simulação e inserindo forma de onda



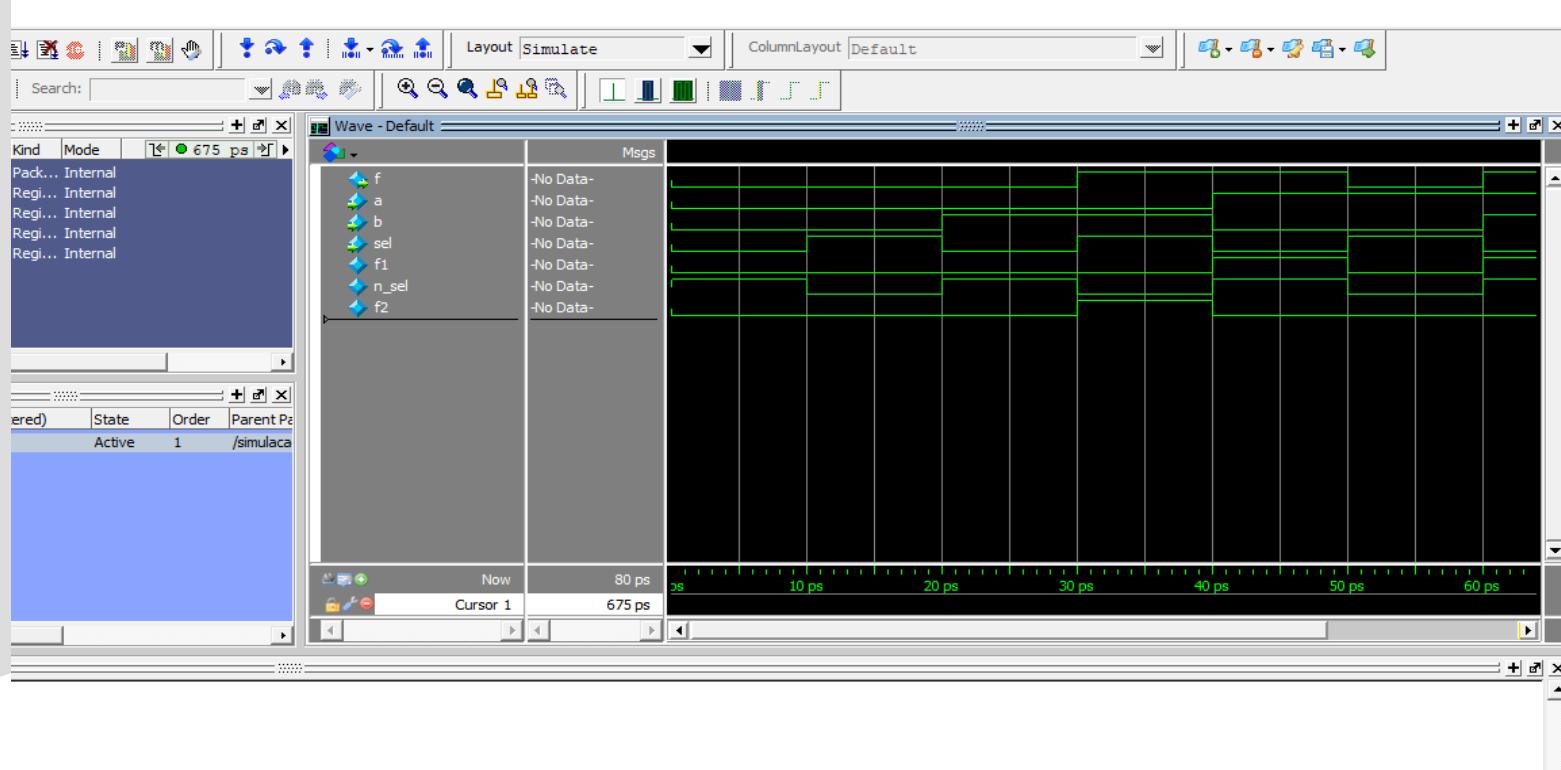
# Simulando com forma de onda

6) Repetindo a simulação e inserindo forma de onda



# Simulando com forma de onda

## 6) Repetindo a simulação e inserindo forma de onda



# Simulando com forma de onda

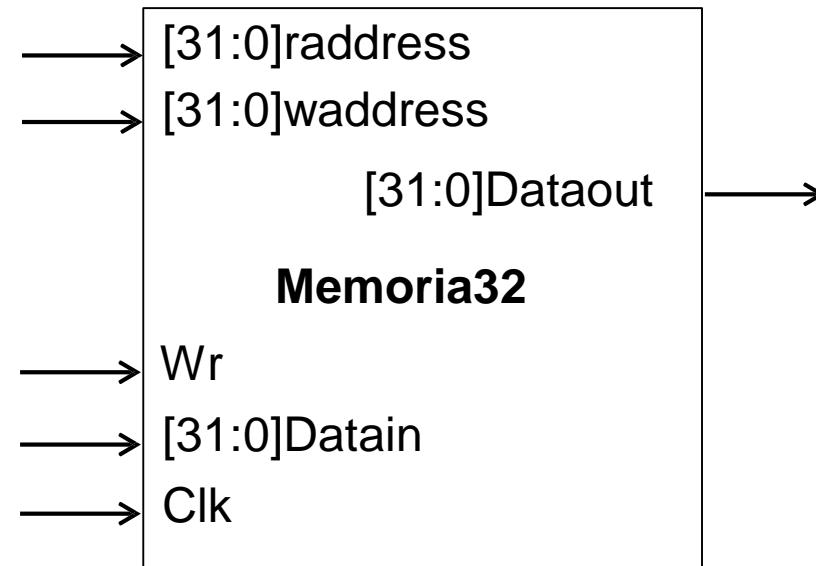
6) Repetindo a simulação e inserindo forma de onda



Para dar “Zoom in”  
aperte o botão I

# Simulando uma memória de 32 bits

Memória utilizada para armazenar as instruções



# Simulando uma memória de 32 bits

## Inicializando dados na memória

```
DEPTH = 8;          -- The size of memory in words  
WIDTH = 8;          -- The size of data in bits  
ADDRESS_RADIX = DEC; -- The radix for address values  
DATA_RADIX =        -- The radix for data values  
BIN; CONTENT        -- start of (address : data pairs)  
  
BEGIN  
000: 00000000;  
001: 10100111;  
002: 00011000;  
003: 00100000;  
  
004:00000001;  
005:00001001;  
006:00111000;  
007:00100010;  
END;
```

**Estrutura de código para inicializar os dados em memória.**

# Simulando uma memória de 32 bits

## Inicializando dados na memória

```
DEPTH = 8;          -- The size of memory in words      ← - - Quantidade de palavras
WIDTH = 8;          -- The size of data in bits       ← - - Largura da palavra
ADDRESS_RADIX = DEC; -- The radix for address values
DATA_RADIX =         -- The radix for data values
BIN; CONTENT        -- start of (address : data pairs)
BEGIN
000: 00000000;
001: 10100111;
002: 00011000;
003: 00100000;
004: 00000001;
005: 00001001;
006: 00111000;
007: 00100010;
END;
```

Maneira como deve ser descritos os endereços.  
DEC – decimal,  
BIN - Binário

Maneira como deve ser descritos os dados.  
DEC – decimal,  
BIN - Binário

# Simulando uma memória de 32 bits

## Inicializando dados na memória

```
DEPTH = 8;          -- The size of memory in words  
WIDTH = 8;          -- The size of data in bits  
ADDRESS_RADIX = DEC; -- The radix for address values  
DATA_RADIX =        -- The radix for data values  
BIN; CONTENT        -- start of (address : data pairs)
```

```
BEGIN               ←  
000: 00000000;  
001: 10100111;  
002: 00011000;  
003: 00100000;
```

Início do  
conteúdo em  
memória

```
004: 00000001;  
005: 00001001;  
006: 00111000;  
007: 00100010;
```

```
END;               ←
```

Fim do  
conteúdo em  
memória

# Simulando uma memória de 32 bits

## Inicializando dados na memória

```
DEPTH = 8;          -- The size of memory in words  
WIDTH = 8;          -- The size of data in bits  
ADDRESS_RADIX = DEC; -- The radix for address values  
DATA_RADIX =        -- The radix for data values  
BIN; CONTENT        -- start of (address : data pairs)  
  
BEGIN  
000: 00000000;  
001: 10100111;  
002: 00011000;  
003: 00100000;  
  
004: 00000001;  
005: 00001001;  
006: 00111000;  
007: 00100010;  
END;
```

**Esse código tem que ser salvo exatamente com o nome instructions.mif**

# Simulando uma memória de 32 bits

**Baixe o arquivo “projeto.zip”**

**Localizado em:**

<https://github.com/leamorim/Infra-de-Hardware-EC-CIn-UFPE/blob/master/Aula%20Modelsim/projeto.zip>

**E Descompacte**

# Simulando uma memória de 32 bits

Vá na pasta projeto/módulos e copiem os seguintes arquivos:

**ramOnChip32.v** e **Memoria32.sv**

para a pasta do seu projeto do modelsim.

# Simulando uma memória de 32 bits

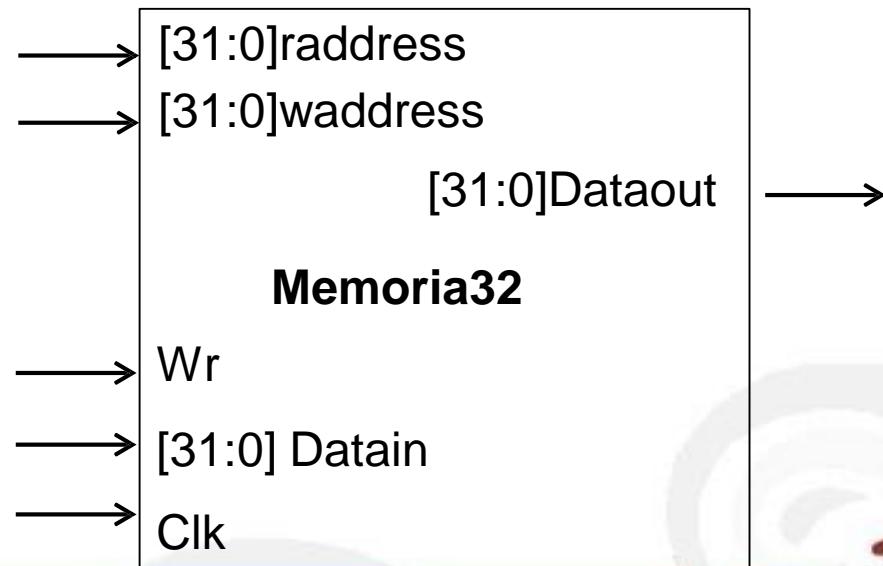
Também vá na pasta projeto/modelsim e copie o arquivo

**instructions.mif**

para a pasta do seu projeto do modelsim.

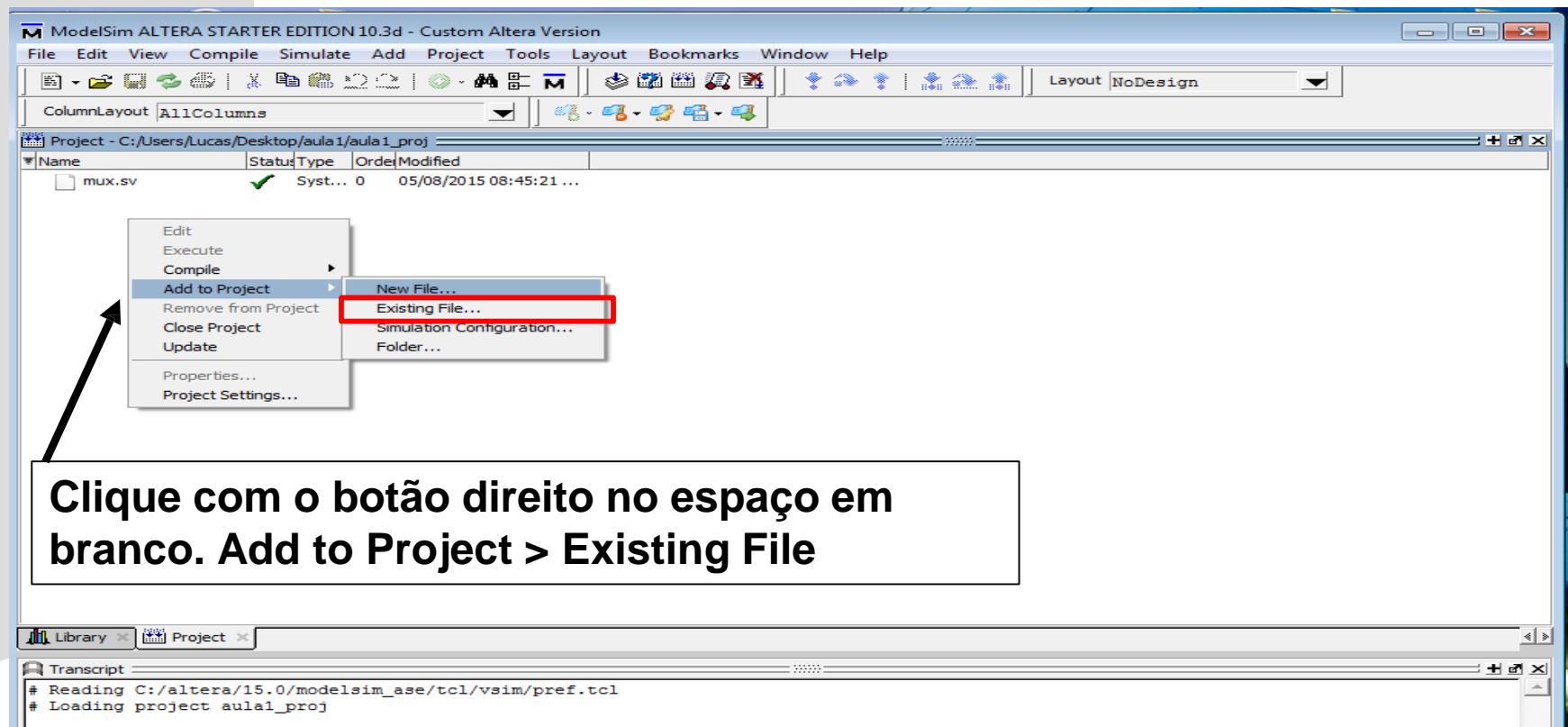
# Simulando uma memória de 32 bits

- O arquivo `ramOnChip32.v` contém um módulo genérico de memória
- O arquivo `Memoria32.sv` implementa sobre `ramOnChip32.v` o módulo de memória de 32 bits com as entradas e saídas mapeadas da seguinte forma:



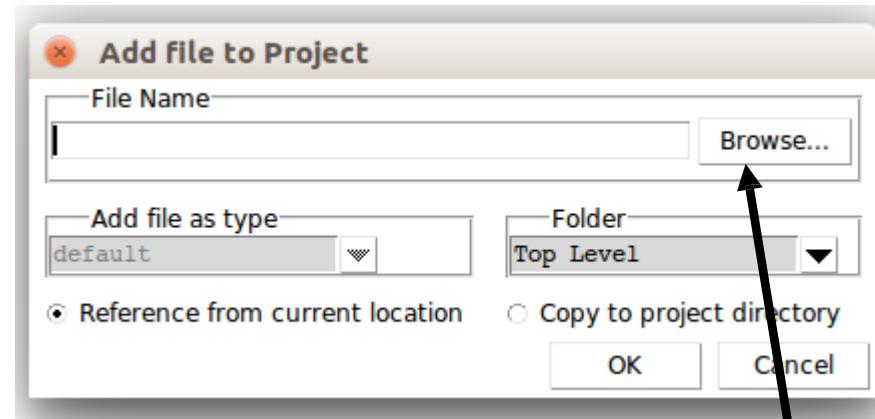
# Simulando uma memória de 32 bits

## Adicionando módulos existentes



# Simulando uma memória de 32 bits

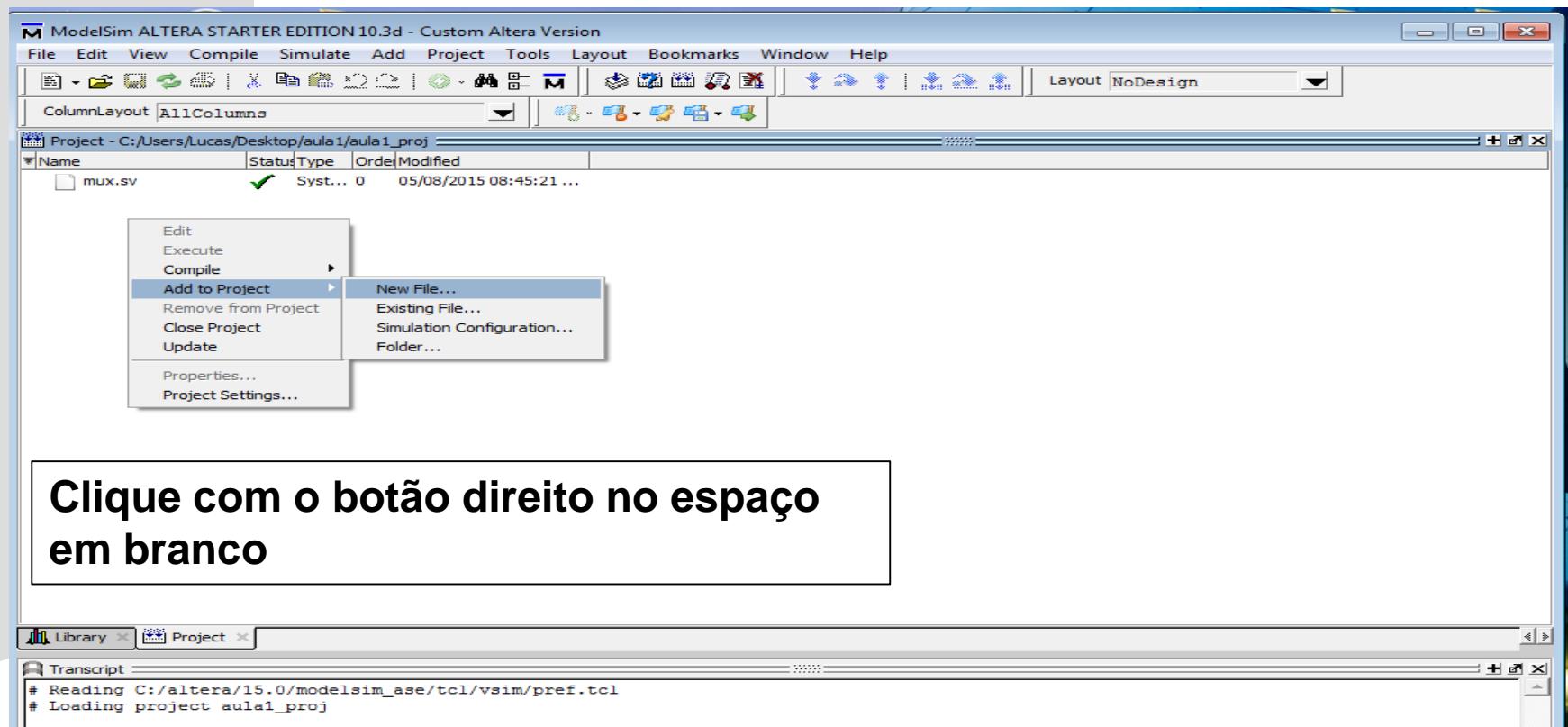
## Adicionando módulos existentes



**Selecione os dois arquivos ramOnChip32.v e  
Memoria32.sv**

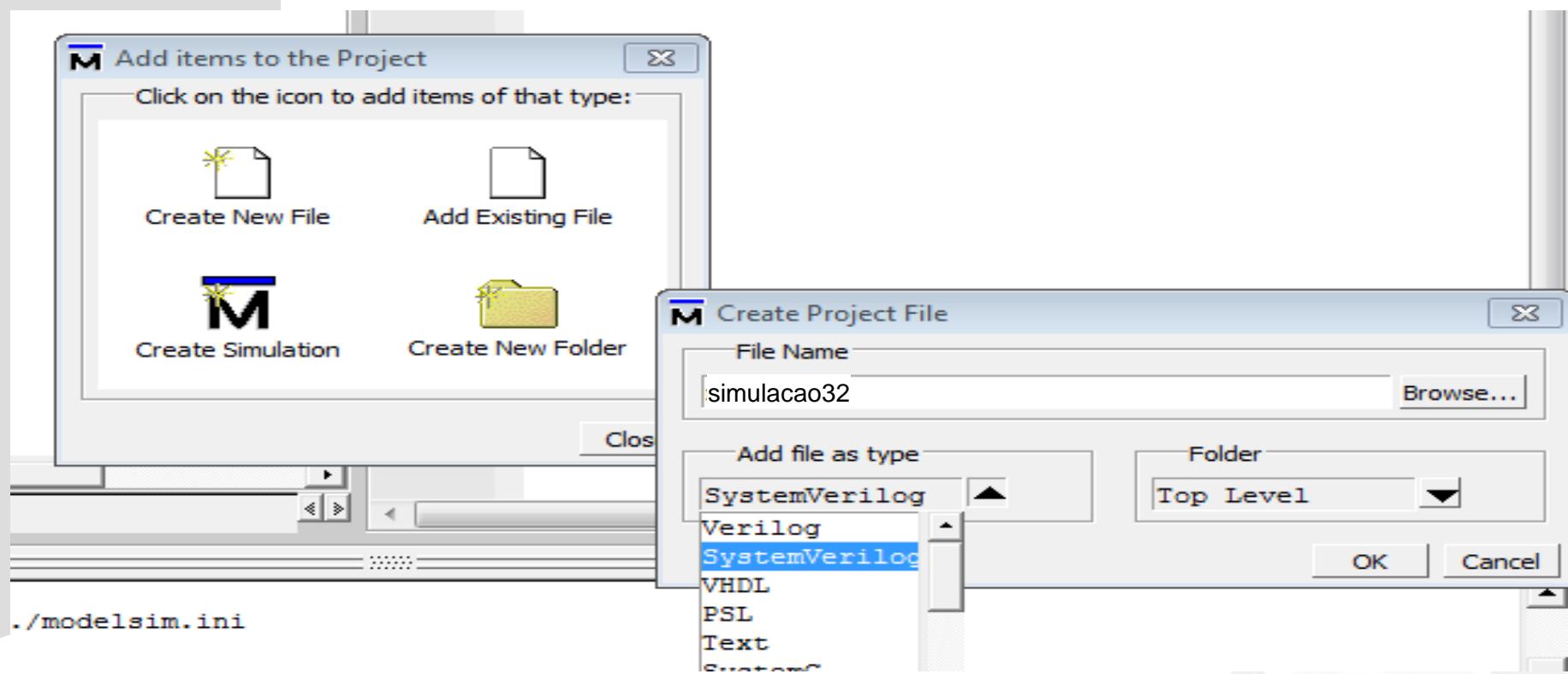
# Simulando uma memória de 32 bits

Agora crie um módulo novo que será aquele que irá gerar os dados para a



# Simulando uma memória de 32 bits

Dê o nome de simulacao32



# Simulando uma memória de 32 bits

Digite o seguinte código no arquivo **simulacao32**  
(também disponível em projeto/modulos)

```
`timescale 1ps/1ps

module simulacao32;

logic clk;
logic nrst;
reg [31:0]rdaddress;
reg [31:0]wdaddress;
reg [31:0]data;
reg Wr;
wire [31:0]q;
```

```
Memoria32 meminst(.raddress(rdaddress), .waddress(wdaddress),
.Clk(clk), .Datain(data), .Dataout(q), .Wr(Wr));
```

# Simulando uma memória de 32 bits

Digite o seguinte código no arquivo `simulacao32` (também disponível em projeto/modulos)

```
`timescale 1ps/1ps

module simulacao32;
    logic clk;
    logic nrst;
    reg [31:0]rdaddress;
    reg [31:0]wdaddress;
    reg [31:0]data;
    reg Wr;
    wire [31:0]q;
```

Criação de fios  
e registradores

```
Memoria32 meminst( .raddress(rdaddress), .waddress(wdaddress),
                    .Clk(clk), .Datain(data), .Dataout(q), .Wr(Wr));
```

# Simulando uma memória de 32 bits

Digite o seguinte código no arquivo simulacao32

```
`timescale 1ps/1ps

module simulacao32;
    logic clk;
    logic nrst;
    reg [31:0]rdaddress;
    reg [31:0]wdaddress;
    reg [31:0]data;
    reg Wr;
    wire [31:0]q;
```

Instanciando o módulo memoria32  
e dando o nome da instancia de  
meminst

```
Memoria32 meminst( .raddress(rdaddress), .waddress(wdaddress),
                    .Clk(clk), .Datain(data), .Dataout(q), .Wr(Wr));
```

# Simulando uma memória de 32 bits

Digite o seguinte código no arquivo simulacao32

```
`timescale 1ps/1ps

module simulacao32;
logic clk;
logic nrst;
reg [31:0]rdaddress;
reg [31:0]wdaddress;
reg [31:0]data;
reg Wr;
wire [31:0]q;
```

Conectando os fios e registradores externos as portas do módulo instanciado

```
Memoria32 meminst( .raddress(rdaddress), .waddress(wdaddress),
                    .Clk(clk), .Datain(data), .Dataout(q), .Wr(Wr));
```

# Simulando uma memória de 32 bits

Digite o seguinte código no arquivo simulacao32

```
//gerador de clock e reset
localparam CLKPERIOD = 10000;
localparam CLKDELAY = CLKPERIOD / 2;

initial begin
    clk = 1'b1;
    nrst = 1'b1;
    #(CLKPERIOD)
    #(CLKPERIOD)
    #(CLKPERIOD)
    nrst = 1'b0;
end

always #(CLKDELAY) clk = ~clk;
```

# Simulando uma memória de 32 bits

Digite o seguinte código no arquivo simulacao32

```
//realiza a leitura
always_ff @(posedge clk or posedge nrst)
begin
    if(nrst) rdaddress <= 0;
    else begin
        if(rdaddress < 64) rdaddress <= rdaddress + 4;
        else begin
            rdaddress <= 0;
            $stop;
        end
    end
end
endmodule
```

# Simulando uma memória de 32 bits

Em seguida vá em Compile>>compile All para compilar todos os módulos.

The screenshot shows a software interface for Verilog simulation. At the top is a menu bar with options: File, Edit, View, Compile, Simulate, Add, Transcript, Tools, Layout, Bookmarks, and Help. Below the menu is a toolbar with various icons. A toolbar dropdown shows "Layout NoDesign" and "ColumnLayout AllColumns". The main area displays a "Project - /home/lucascambuim/teste/teste" list with three items:

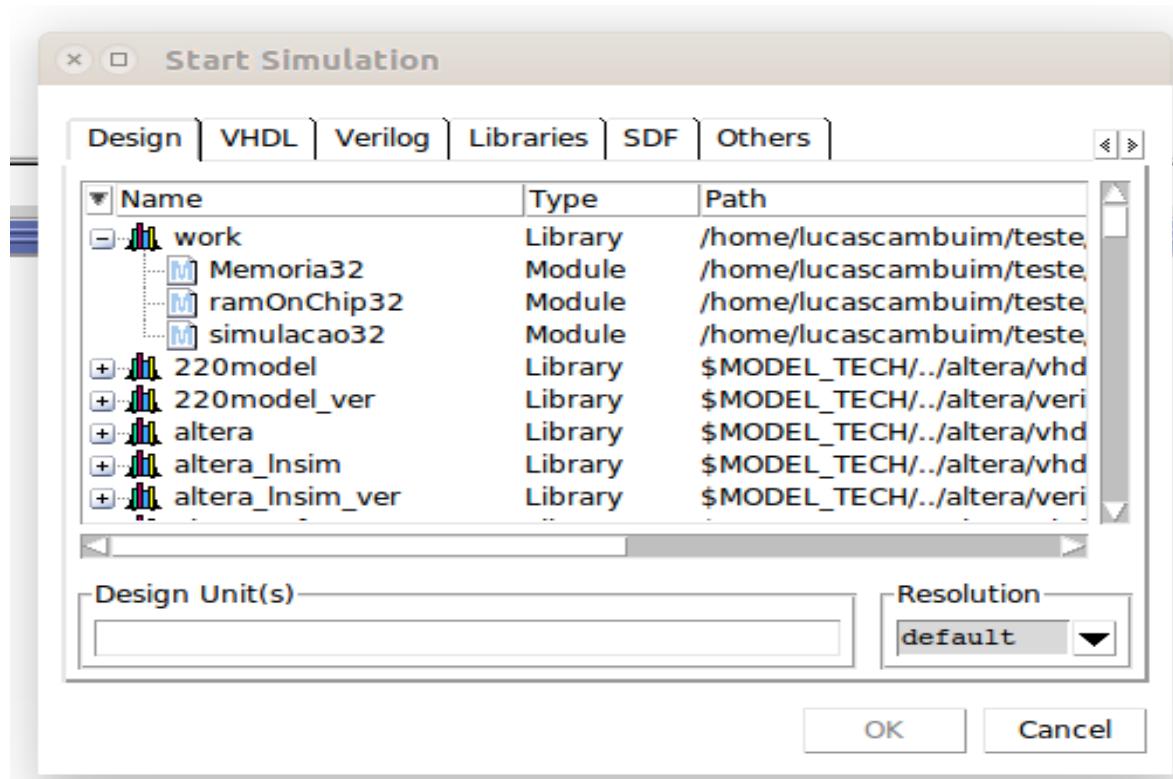
Name	Status	Type	Order	Modified
simulacao32.sv	✓	SystemVerilog	2	03/27/2019 11:18:53 p...
ramOnChip32.v	✓	Verilog	1	10/05/2018 10:38:48 a...
Memoria32.sv	✓	SystemVerilog	0	10/05/2018 10:47:26 a...

At the bottom, there are tabs for "Library" and "Project", with "Project" currently selected. The "Transcript" tab is open, showing the following text:

```
# 3 compiles, 0 failed with no errors.
# Compile of Memoria32.sv was successful.
# Compile of ramOnChip32.v was successful.
# Compile of simulacao32.sv was successful.
```

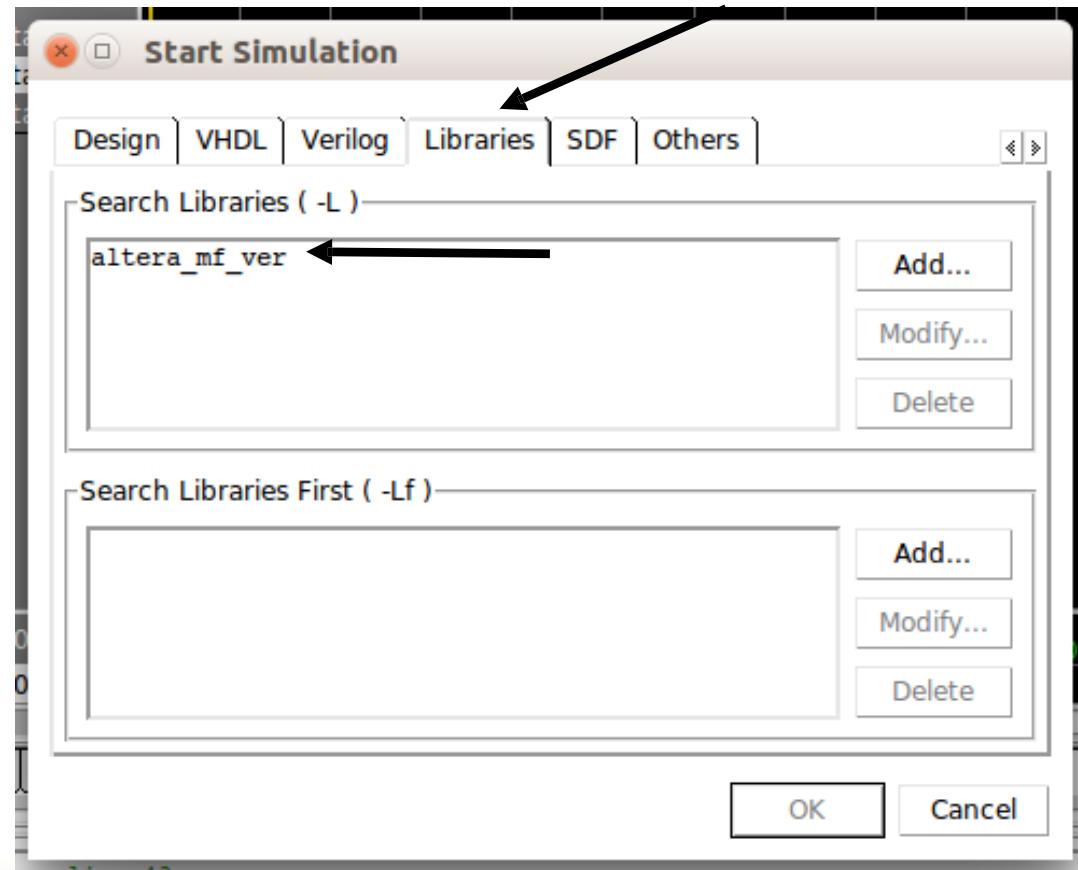
# Simulando uma memória de 32 bits

Em seguida vá em Simulate > Start Simulate



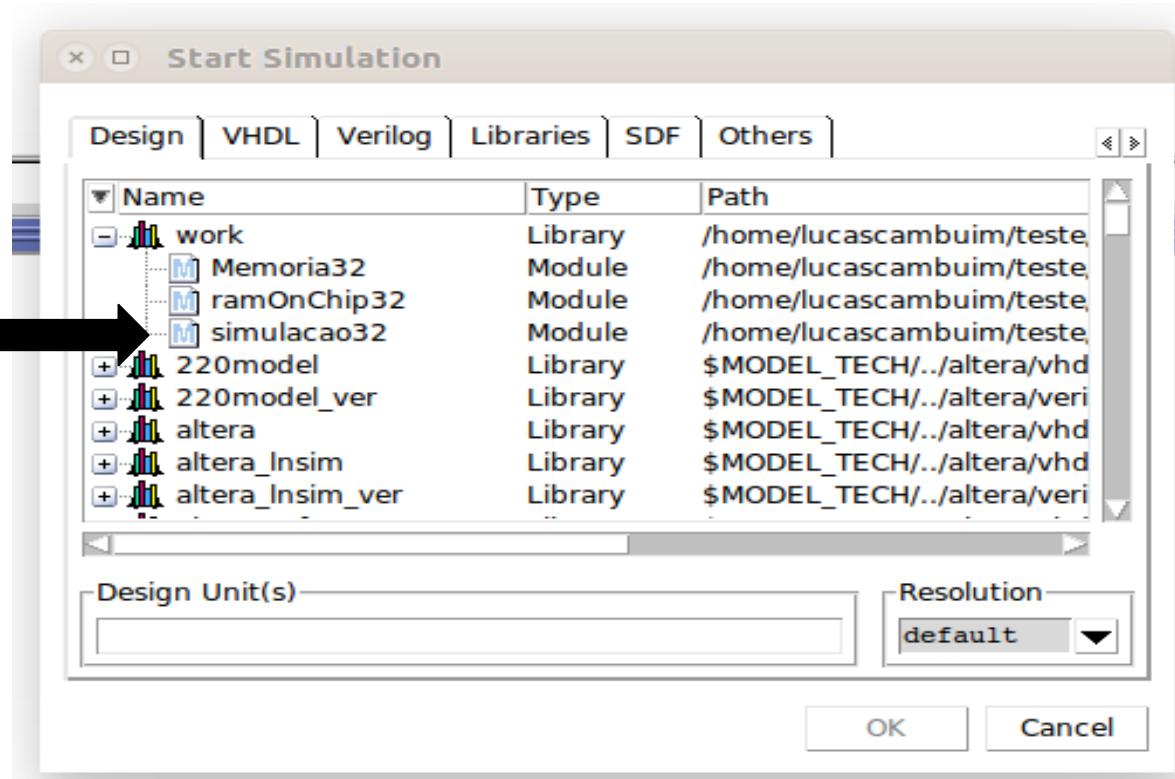
# Simulando uma memória de 32 bits

Na aba Libraries, adiciona a biblioteca “altera\_mf\_ver” para reconhecer a memória



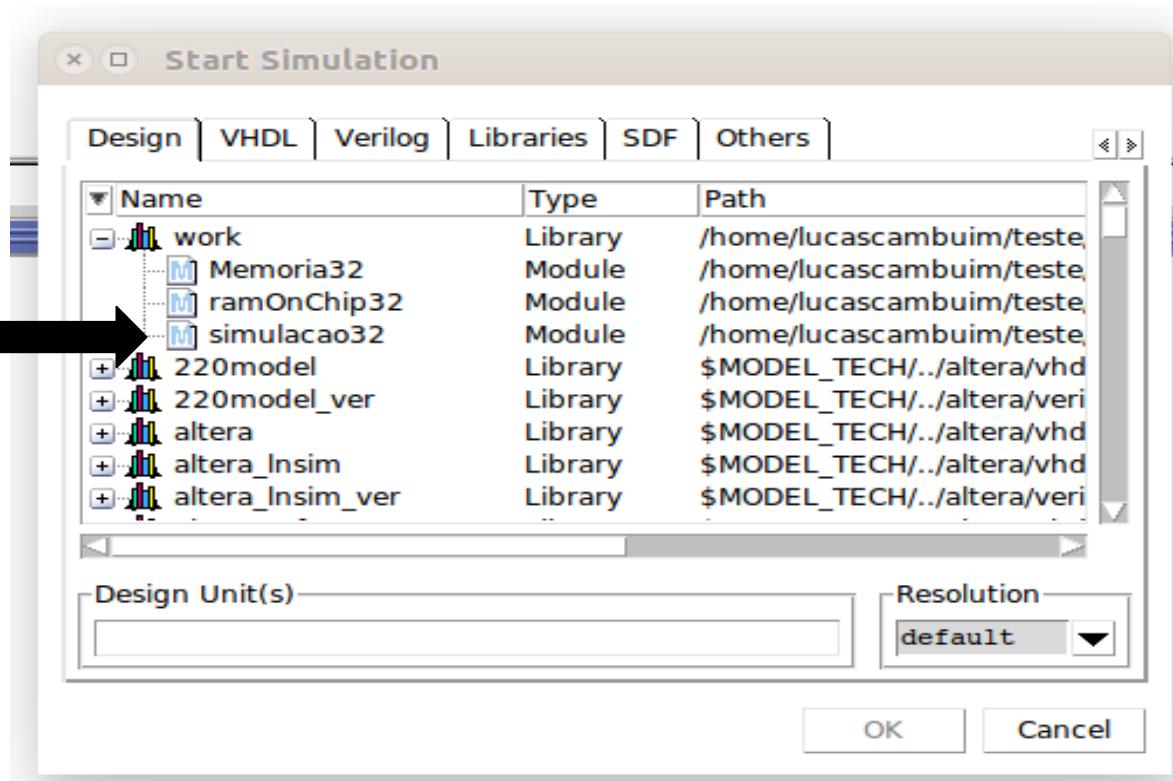
# Simulando uma memória de 32 bits

Na aba Design, selecione simulacao32



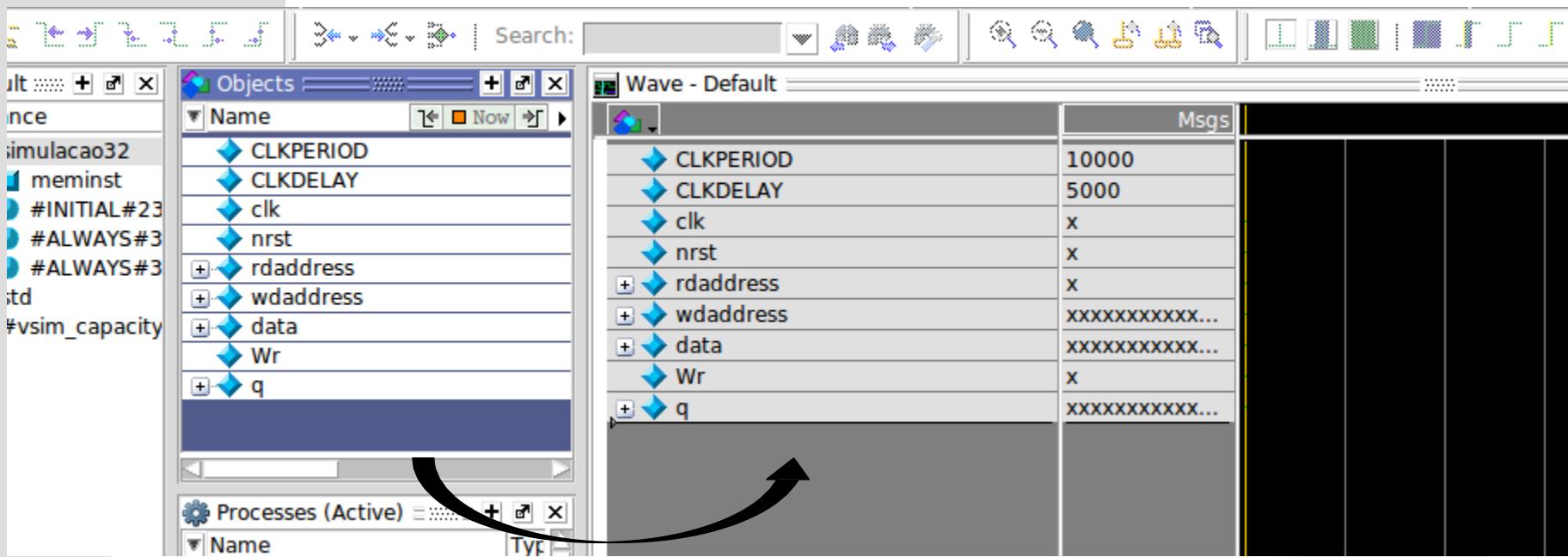
# Simulando uma memória de 32 bits

Na aba Design, selecione simulacao32



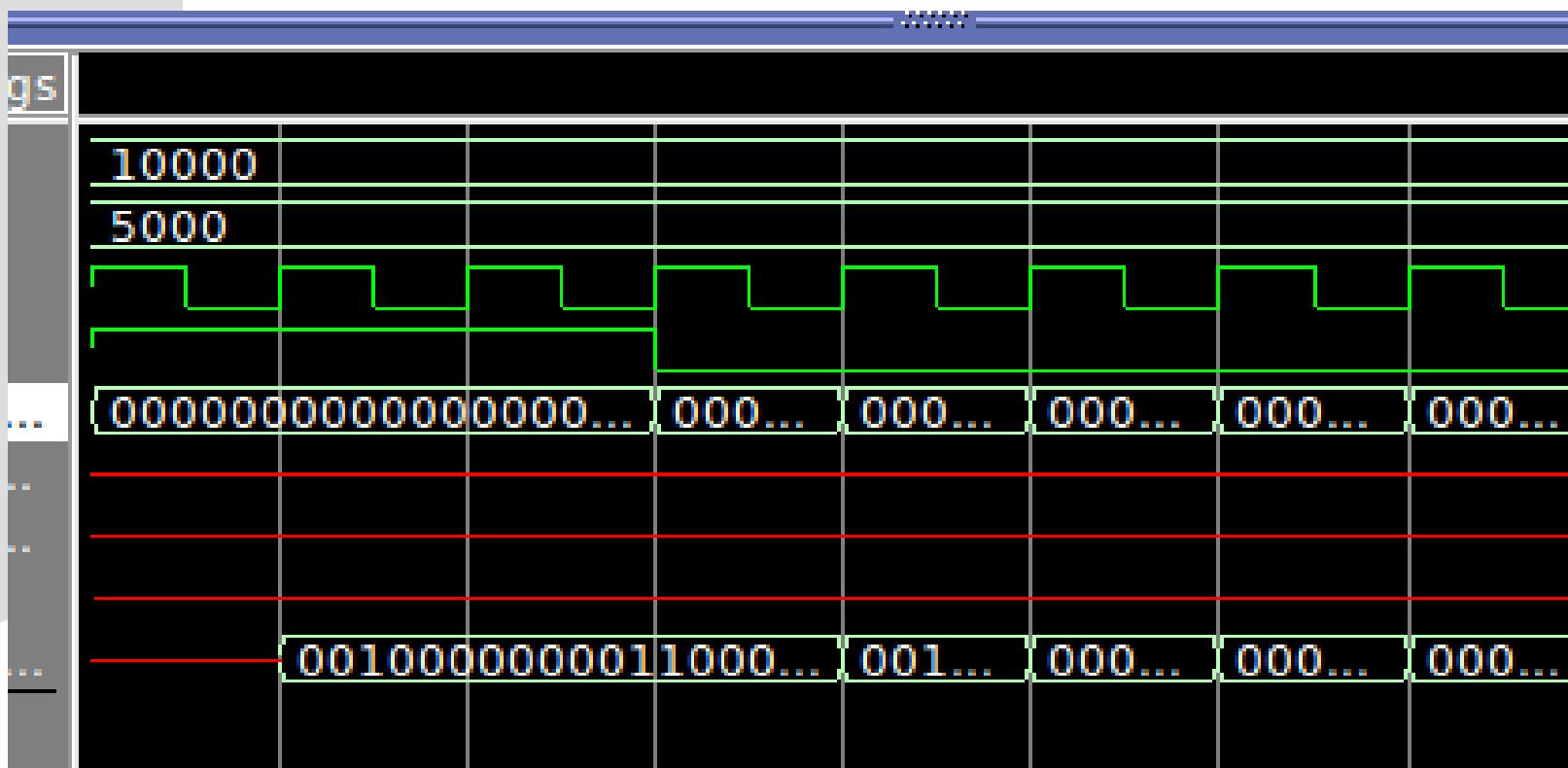
# Simulando uma memória de 32 bits

Na aba Design, selecione simulacao32



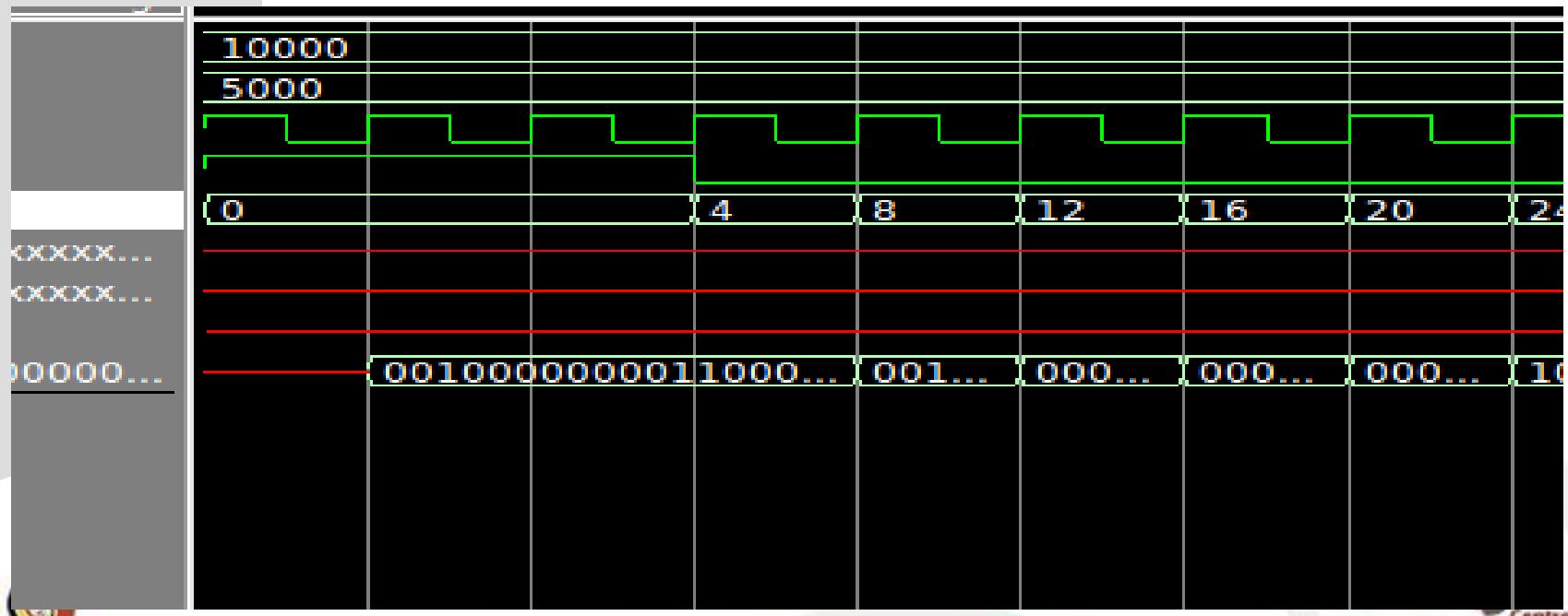
# Simulando uma memória de 32 bits

E por fim, clique em run -All e espera o simulador terminar



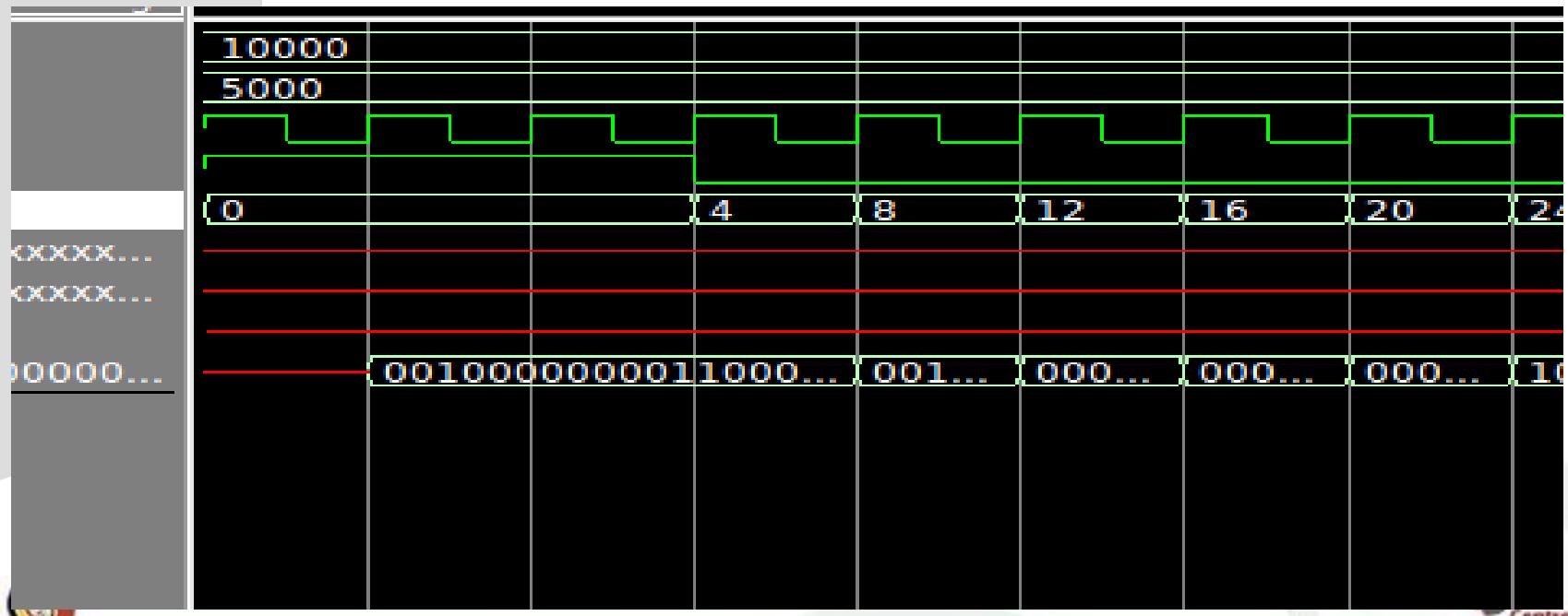
# Simulando uma memória de 32 bits

É possível mudar a forma do número apresentado.



# Simulando uma memória de 32 bits

Para isso, clique com o botão direito em cima do sinal desejado, vá em Radix e escolha decimal.



# Simulando uma memória de 32 bits

Verifique se a saída da memória está igual aos valores definidos no arquivo instructions.mif

# Simulando a memória usando script

## 7) Usando memória e script do modelsim

Vamos agora executar um script de modelsim que realiza todos os passos de compilação e simulação de maneira automática para simular a memória.

# Simulando a memória usando script

## 7) Usando memória e script do modelsim

Estrutura de arquivos do “projeto.zip”

```
.../projeto  
...../modulos  
...../ramOnChip32.v  
...../Memoria32.sv  
...../simulacao32.sv  
...../modelsim  
...../compile_verilog  
...../run  
...../instrucao.mif
```

# Simulando a memória usando script

## 7) Usando memória e script do modelsim

São dois arquivos de simulação:

a) **compile\_verilog**

Contém a localização dos arquivos verilog que pertence ao seu projeto.

Ex:

..../modulos/ramOnChip32.v

..../modulos/Memoria32.v

..../modulos/simulacao32.sv

# Simulando a memória usando script

## 7) Usando memória e script do modelsim

São dois arquivos de simulação:

b) runmemoria32

Contém os comandos para compilar e simular no modelsim.

# Simulando a memória usando script

```
vlib work
vdel -all -lib work
vlib work
vlog -f compile_verilog
vsim      -L          -L          lpm_ver   -L          sgate_ver  -L
           altera_mf_ver       altera_ver -novopt
work.simulacao32
add wave -position end sim:/simulacao32/CLKPERIOD
add wave -position end sim:/simulacao32/CLKDELAY
add wave -position end sim:/simulacao32/ramSize
add wave -position end sim:/simulacao32/clk
add wave -position end sim:/simulacao32/nrst
add wave -position end sim:/simulacao32/rdaddress
add wave -position end sim:/simulacao32/wdaddress
add wave -position end sim:/simulacao32/data
add wave -position end sim:/simulacao32/Wr
add wave -position end sim:/simulacao32/q
```

run -all

# Simulando a memória usando script

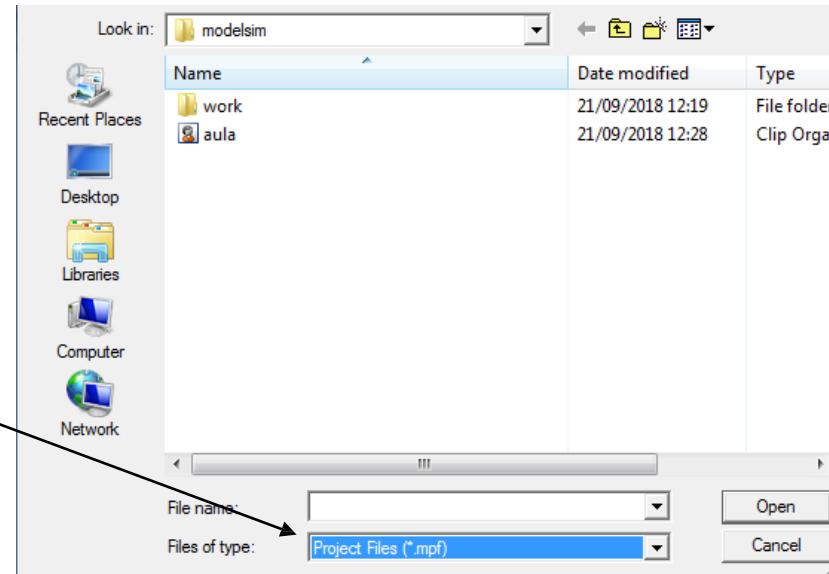
```
vlib work
vdel -all -lib work
vlib work
vlog -f compile_verilog
vsim      -L
          altera_mf_ver
work.simulacao32
add wave -position end sim:/simulacao32/CLKPERIOD
add wave -position end sim:/simulacao32/CLKDELAY
add wave -position end sim:/simulacao32/ramSize
add wave -position end sim:/simulacao32/clk
add wave -position end sim:/simulacao32/nrst
add wave -position end sim:/simulacao32/rdaddress
add wave -position end sim:/simulacao32/wdaddress
add wave -position end sim:/simulacao32/data
add wave -position end sim:/simulacao32/Wr
add wave -position end sim:/simulacao32/q
```

Este nome tem que ser modulo top que gera os sinais

# Simulando a memória usando script

## 7) Usando memória e script do modelsim

1. Abra o modelSim
2. Clique em File > open
3. Localize o arquivo projeto/modelsim/projeto.mpf



Coloque para visualizar  
arquivos .mpf

# Simulando a memória usando script

## 7) Usando memória e script do modelsim

Digite no espaço “transcript” o seguinte comando:

**do runmemoria32**

```
Transcript  
# Reading C:/altera/15.0/modelsim_ase/tcl/vsim/pref.tcl  
# Loading project aula  
# reading C:/altera/15.0/modelsim_ase/win32aloem/.../modelsim.ini  
# Loading project aula  
  
ModelSim> do runmemoria32
```

# Simulando a memória usando script

## 7) Usando memória e script do modelsim

The screenshot shows the ModelSim simulation environment. The top menu bar includes File, Edit, View, Compile, Simulate, Add, Wave, Tools, Layout, Bookmarks, Window, and Help. The main window displays several panels:

- Objects**: A table showing internal signals: CLKPERIOD (10000), CLKDELAY (5000), ramSize (65536), clk (x), nrst (x), rdaddress (xxxx...), and q (xxxx...). The 'Value' column shows their current or default values.
- Wave - Default**: A waveform viewer showing the simulation of the memory system. Traces include ramSize, clk, nrst, rdaddress, and q. The waveforms show the clock signal (clk) switching between 0 and 1, the address (rdaddress) being asserted, and the output q showing the result of the memory access.
- Processes (Active)**: A table listing four active processes: #ASSIGN#59, #ASSIGN#58, #ASSIGN#57, and #ASSIGN#56, all in Ready state.
- Transcript**: A text window showing the command-line history of the simulation setup.

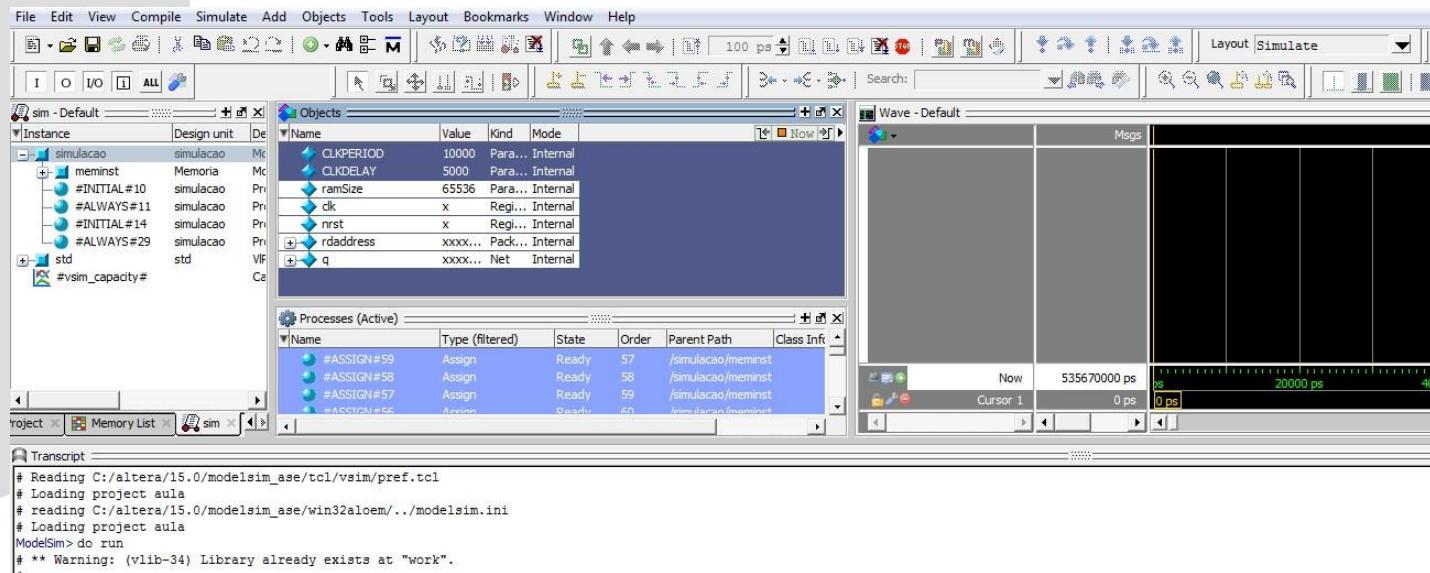
```
# Reading C:/altera/15.0/modelsim_ase/tcl/vsim/pref.tcl
# Loading project aula
# reading C:/altera/15.0/modelsim_ase/win32aloem/..../modelsim.ini
# Loading project aula
ModelSim> do run
# ** Warning: (vlib-34) Library already exists at "work".
#
# Model Technology ModelSim ALTERA vlog 10.3d Compiler 2014.10 Oct 7 2014
# Start time: 12:51:26 on Sep 21, 2018
# vlog -reportprogress 300 -f compile_verilog
# -- Compiling module simulacao
#   Compiling module vramctrl
```

# Simulando a memória usando script

## 7) Usando memória e script do modelsim

Se quiser adicionar sinais ao script

1º) Adicione manualmente

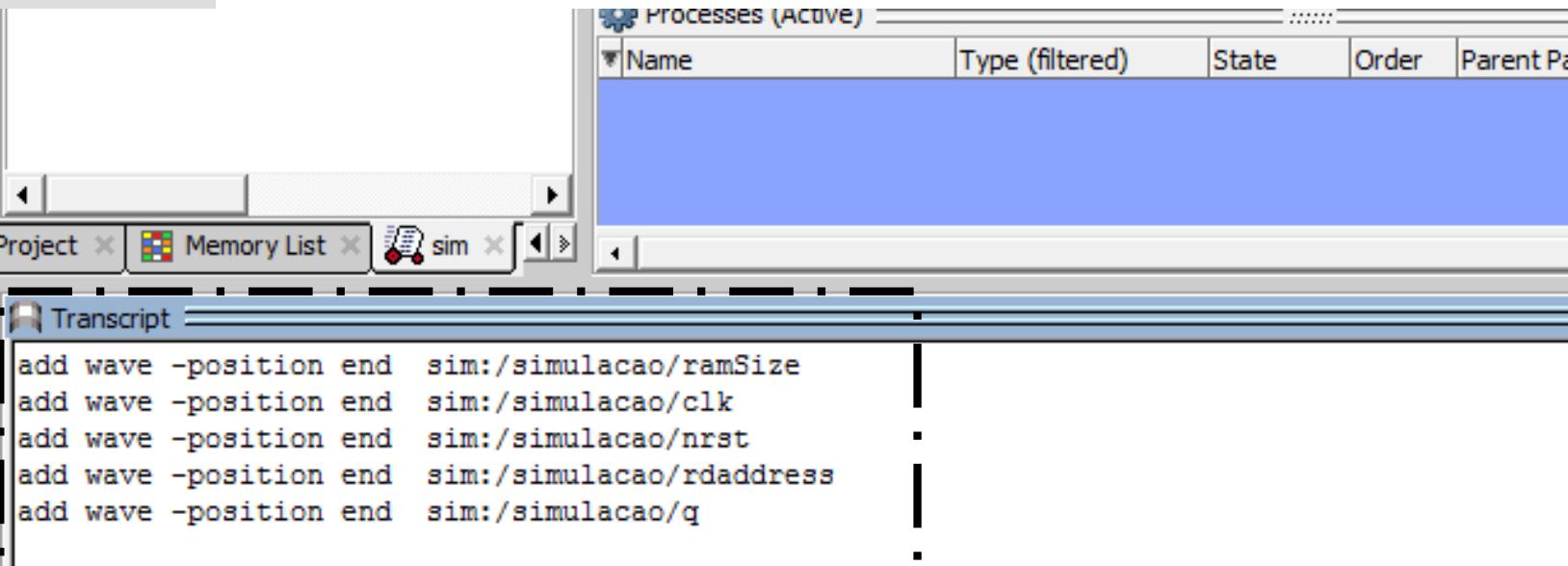


# Simulando a memória usando script

## 7) Usando memória e script do modelsim

Se quiser adicionar sinais ao script

2º) Copie os comandos



The screenshot shows the ModelSim interface with the following windows visible:

- Project**: Shows the project structure.
- Memory List**: Shows memory-related items.
- sim**: Shows the simulation process.
- Transcript**: Shows the command-line interface where the following commands were entered:

```
add wave -position end sim:/simulacao/ramSize
add wave -position end sim:/simulacao/clk
add wave -position end sim:/simulacao/nrst
add wave -position end sim:/simulacao/rdaddress
add wave -position end sim:/simulacao/q
```

# Simulando a memória usando script

## 7) Usando memória e script do modelsim

**Se quiser adicionar sinais ao script**

**3º) Cole no arquivo run**

...

```
vsim -L altera_mf_ver -L lpm_ver -L sgate_ver -L altera_ver -novopt  
work.simulação32
```

```
add wave -position end sim:/simulacao32/ramSize  
add wave -position end sim:/simulacao32/clk  
add wave -position end sim:/simulacao32/nrst  
add wave -position end sim:/simulacao32/rdaddress  
add wave -position end sim:/simulacao32/q
```

run -all

# Comandos para script

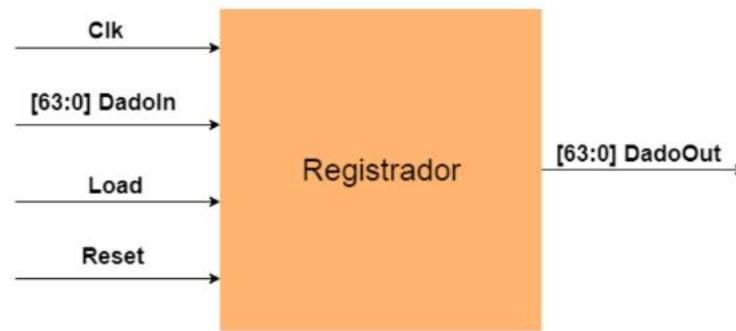
- Comandos e diretivas disponíveis no modelsim detalhados no seguinte PDF:
  - [https://www.microsemi.com/document-portal/doc\\_view/136364-modelsim-me-10-4c-command-reference-manual-for-libero-soc-v11-7](https://www.microsemi.com/document-portal/doc_view/136364-modelsim-me-10-4c-command-reference-manual-for-libero-soc-v11-7)

# Exercício

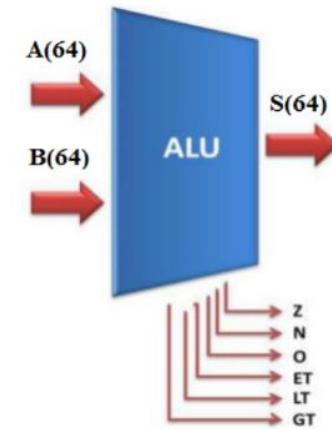
- Usando o módulo register.sv e o módulo ula64.vhd, disponível no arquivo projeto.zip, faça uma máquina de estados que incrementa o valor de uma instância de register.sv de 4 em 4 usando uma instância de ula64.vhd e salve o valor resultante na mesma instância de register.sv
  - Observação: O registrador leva um ciclo para ter o valor escrito disponível
  - É recomendado criar um outro arquivo para fazer as conexões

# Exercício - Observações

*Obs: o dado que entra em Data\_in só passa para Data\_out quando o registrador passa 1 ciclo de clock com o sinal **Load** ativo (Load = 1).*



Função	Operação	Descrição	Flags
000	S = A	Carrega A	Z, N
001	S = A + B	Soma	Z, N, O
010	S = A - B	Subtração	Z, N, O
011	S = A and B	And lógico	Z
100	S = A + 1	Incremento de A	Z, N, O
101	S = not A	Negação de A	Z
110	S = A xor B	OU exclusivo	Z
111	S = A comp B	Comparação	EG, GT, LT



# Aprendendo a utilizar a Ferramenta Modelsim



Professor: Lucas Cambuim (lfsc)  
Adaptado por: Lucas Amorim,  
Matheus Costa e Anderson  
Henrique

