

# Aprendendo a utilizar a Ferramenta Modelsim

Professor: Lucas Cambuim (lfsc)



# 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.



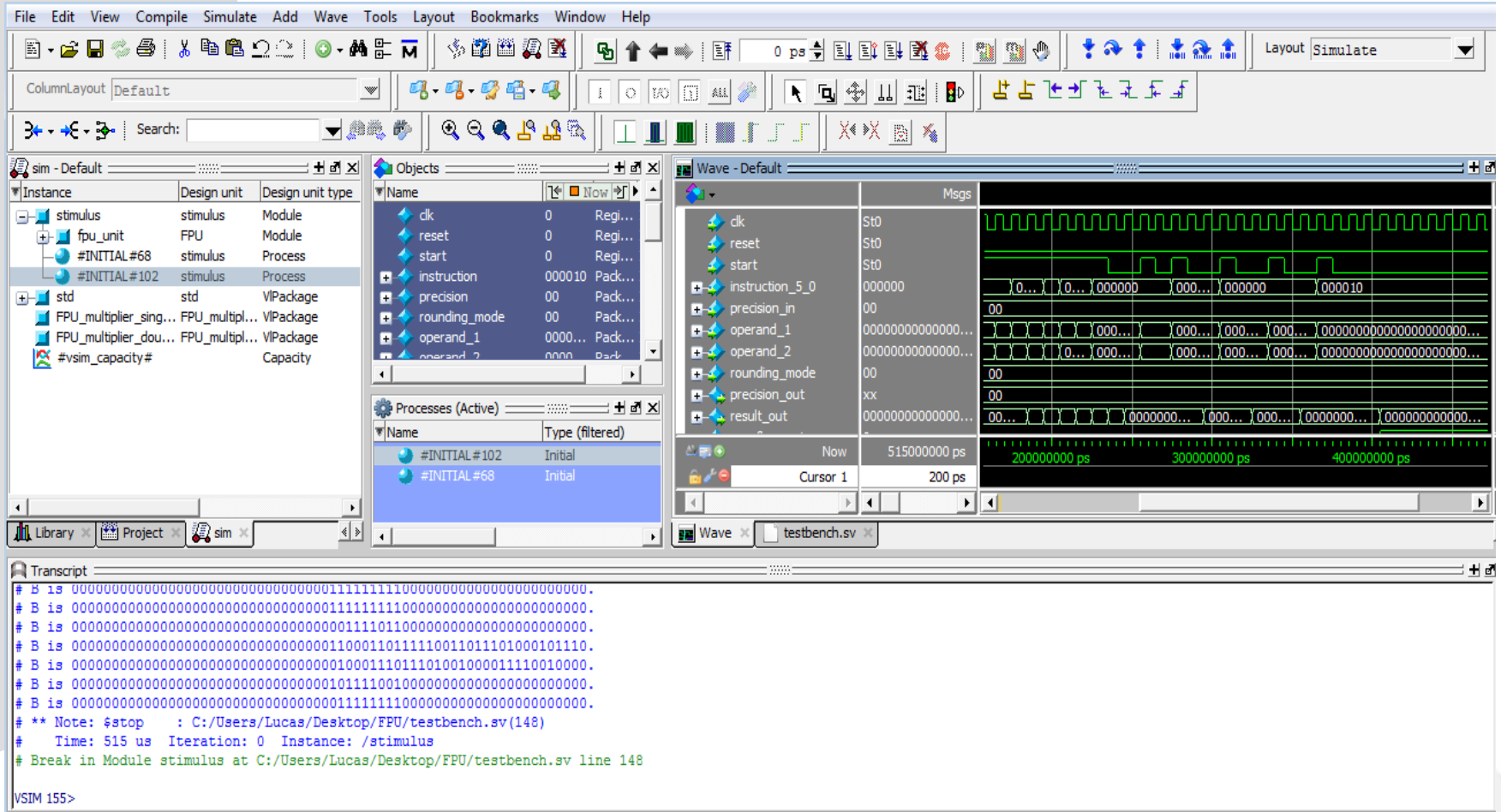
# Facilidades oferecidas pela ferramenta

- 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.

# Visão da Ferramenta

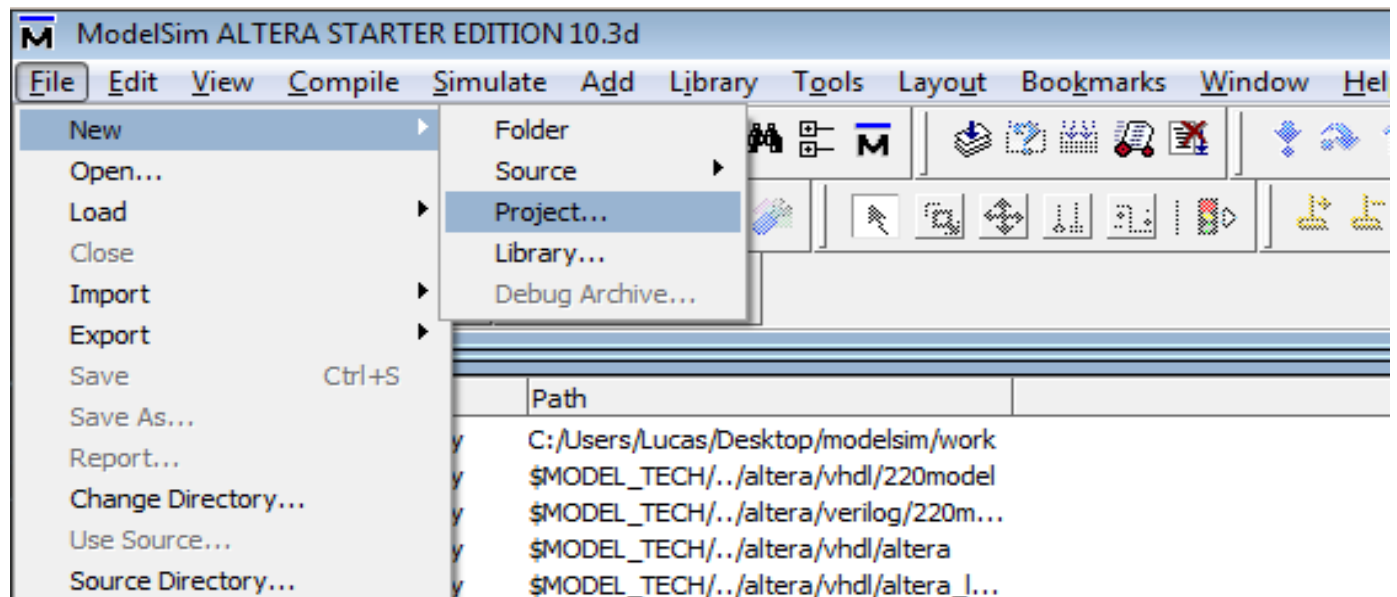


# Recursos importantes da Ferramenta

[illegible]

# Aprendendo a usar a ferramenta em etapas

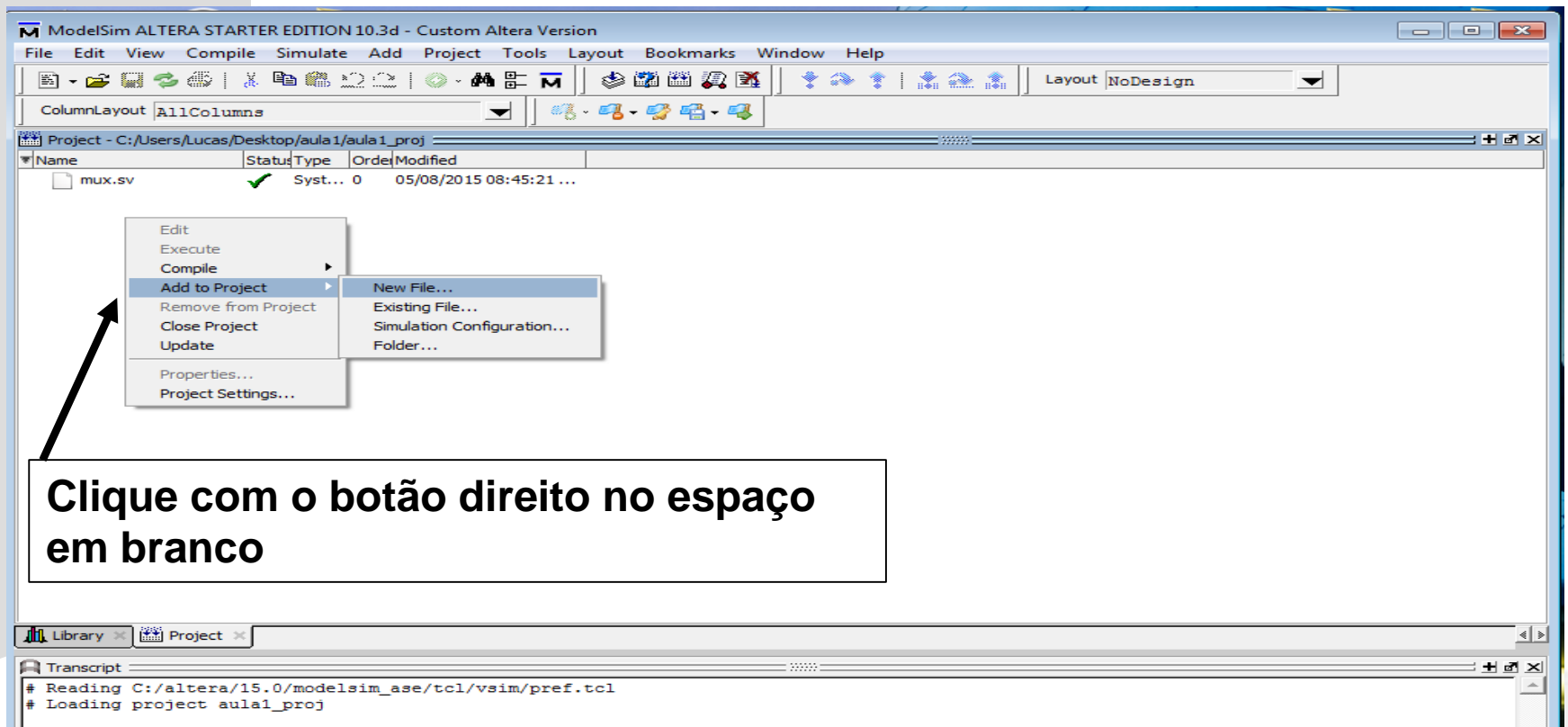
## 1) Criando um projeto (vá em File > new > Project)





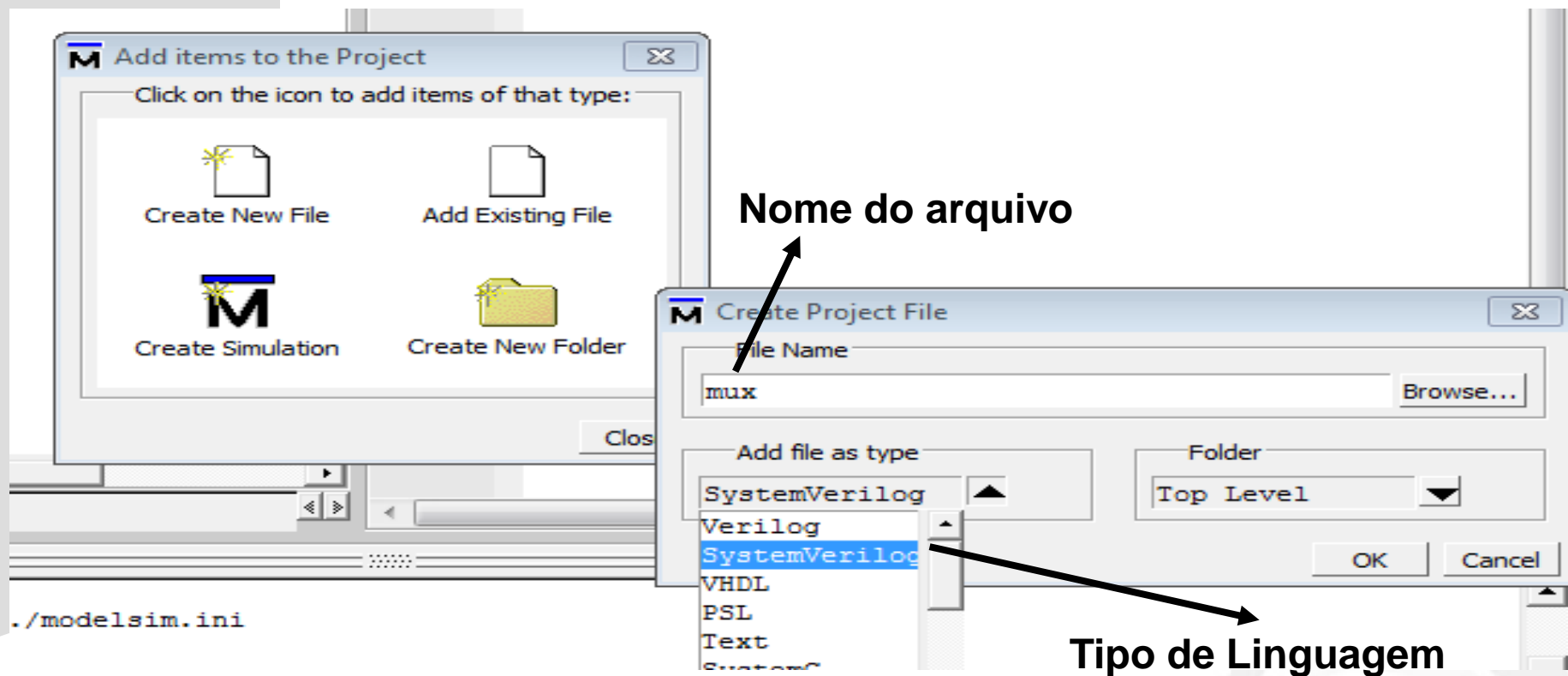
# Aprendendo a usar a ferramenta em etapas

## 2) Criando uma implementação em System Verilog



# Aprendendo a usar a ferramenta em etapas

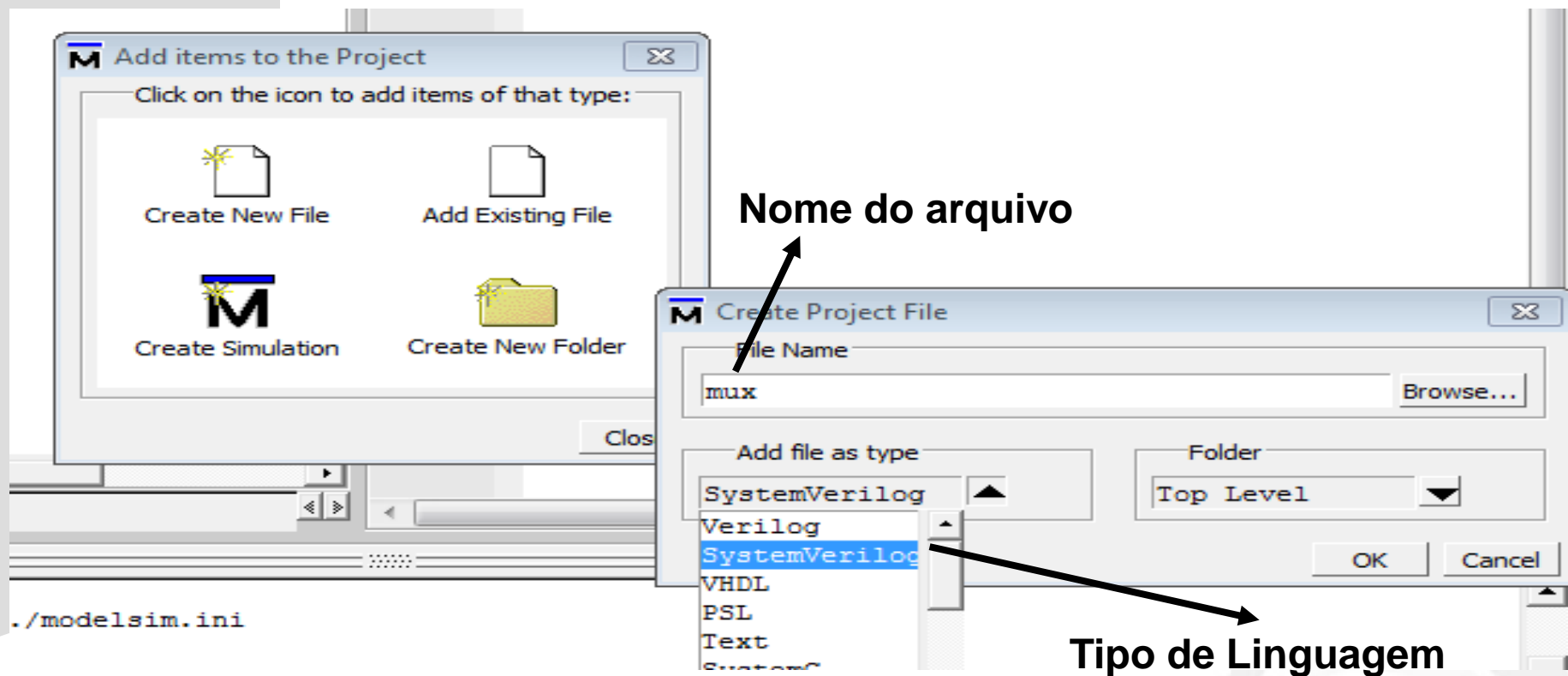
## 2) Criando uma implementação em System Verilog





# Aprendendo a usar a ferramenta em etapas

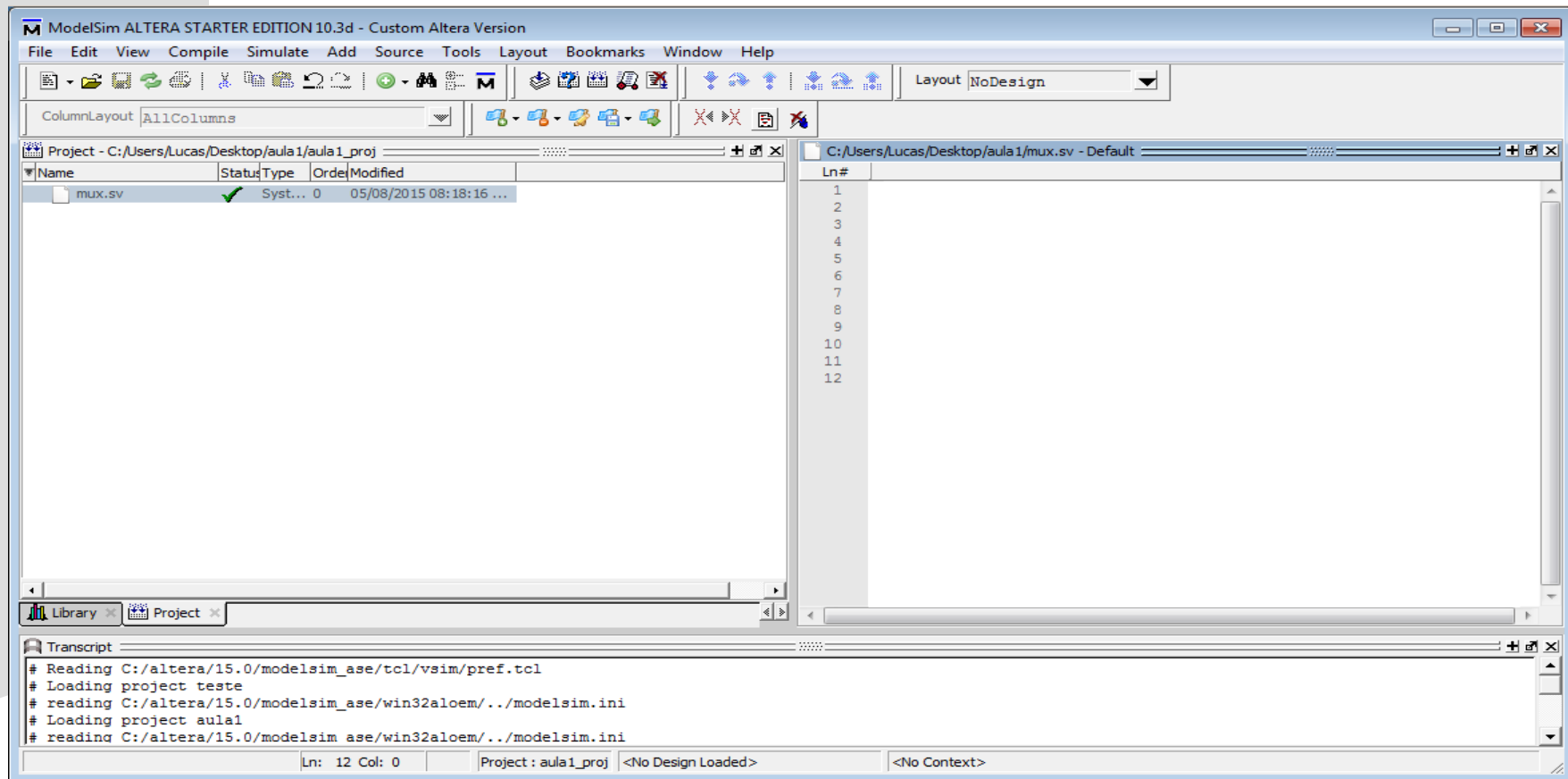
## 2) Criando uma implementação em System Verilog



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

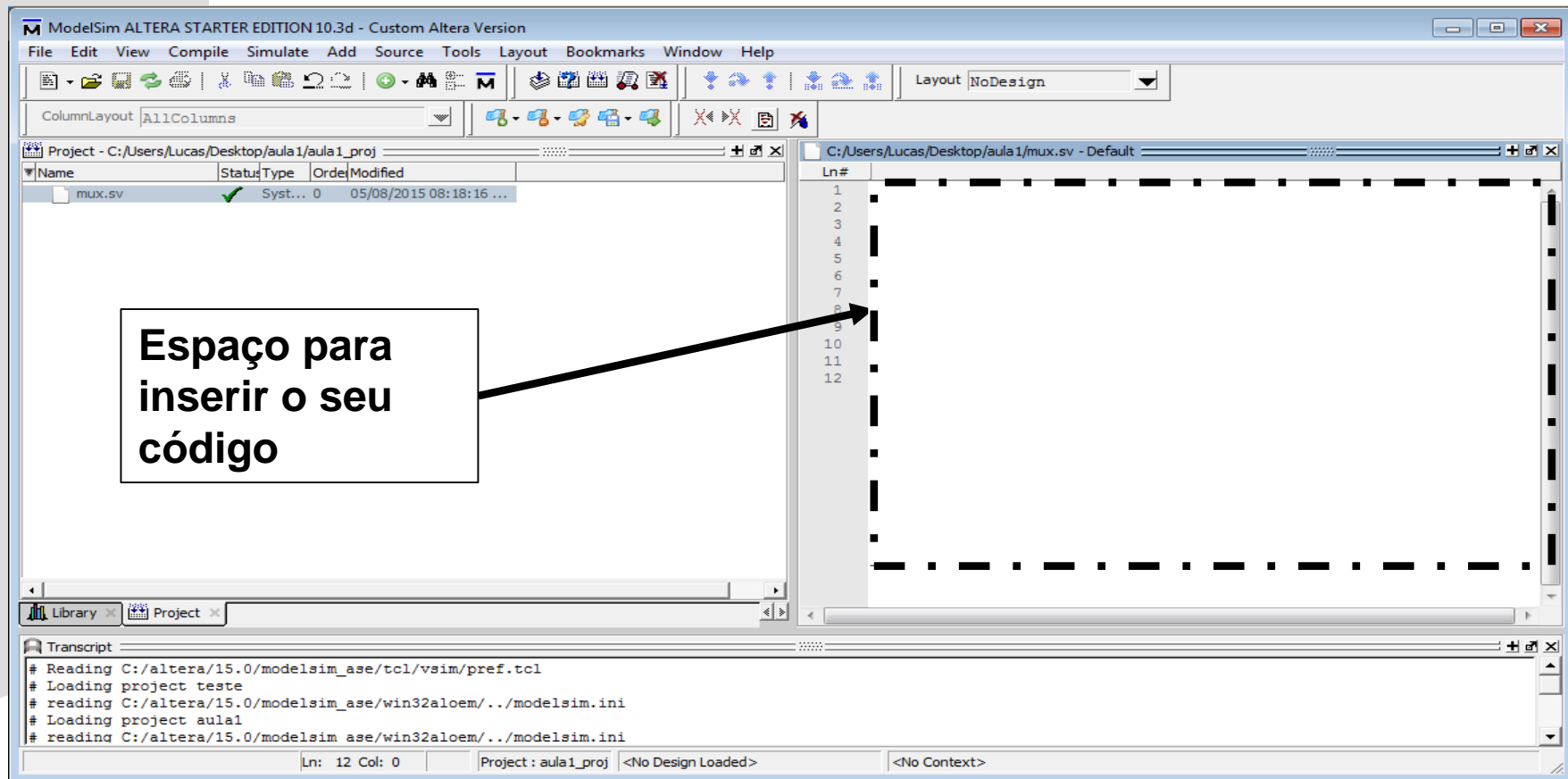
# Aprendendo a usar a ferramenta em etapas

## 2) Criando uma implementação em System Verilog



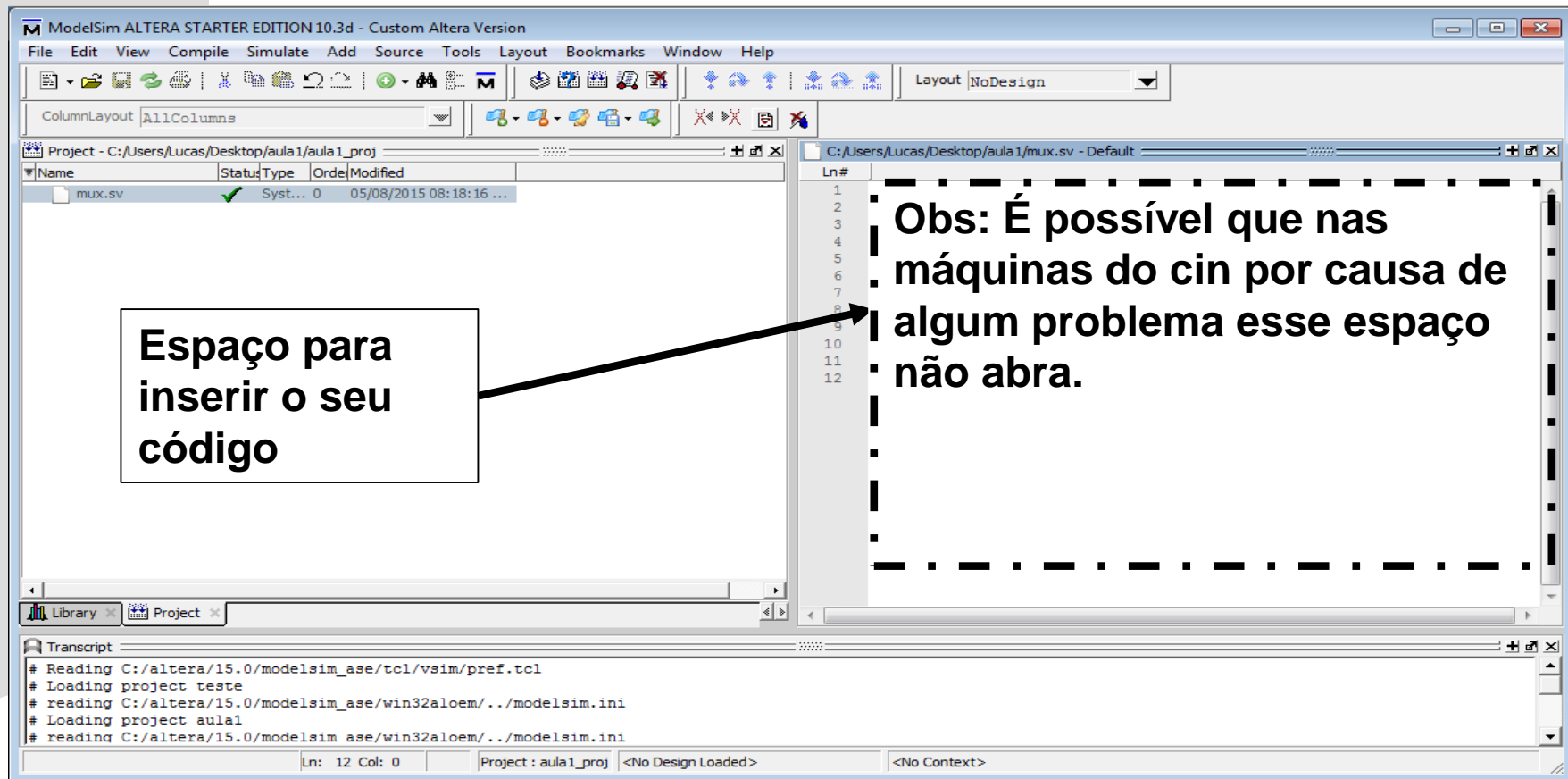
# Aprendendo a usar a ferramenta em etapas

## 2) Criando uma implementação em System Verilog



# Aprendendo a usar a ferramenta em etapas

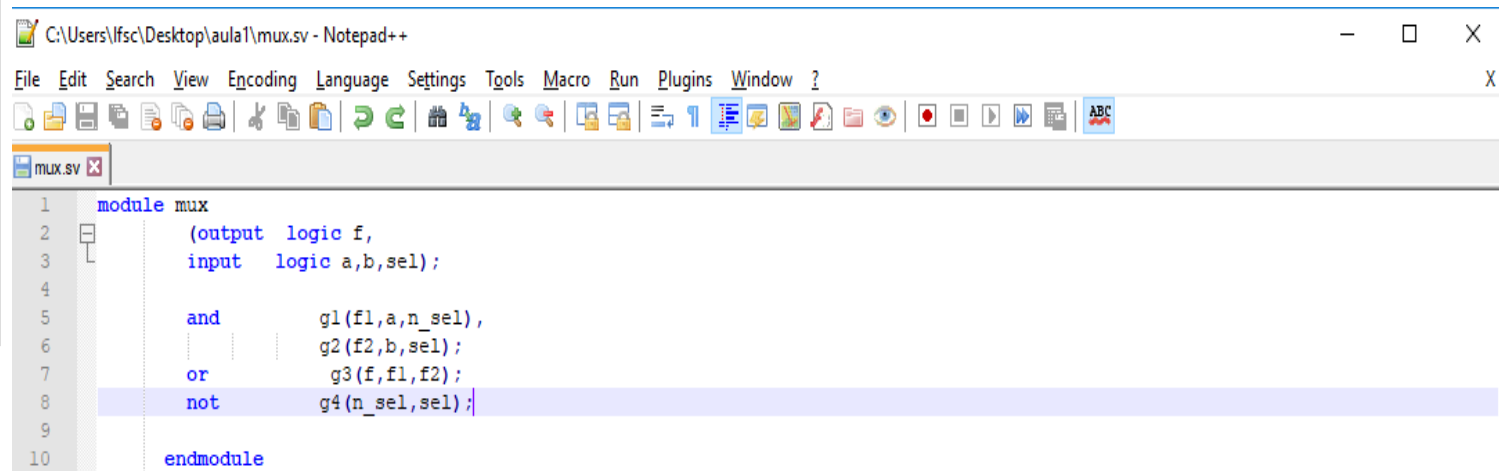
## 2) Criando uma implementação em System Verilog



# Aprendendo a usar a ferramenta em etapas

## 2) Criando uma implementação em System Verilog

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



The screenshot shows a Notepad++ window titled 'C:\Users\lfsc\Desktop\aula1\mux.sv - Notepad++'. The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, Window, and ?. The toolbar contains various icons for file operations, editing, and development. The code editor displays a System Verilog module named 'mux' with the following code:

```
1 module mux
2     (output logic f,
3      input logic a,b,sel);
4
5     and      g1(f1,a,n_sel),
6     ..      g2(f2,b,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
```

Copie este  
código  
exemplo e cole  
no arquivo  
mux.sv

# Aprendendo a usar a ferramenta em etapas

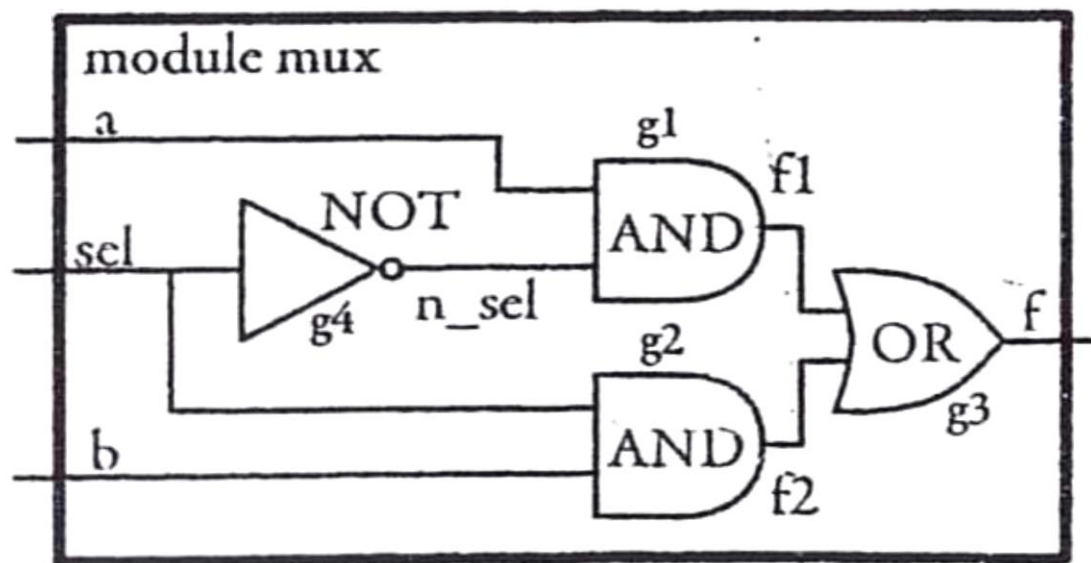
## 2) Criando uma implementação System Verilog

Nome do módulo  
*module mux*  
Portas do módulo  
(output logic f,  
input logic a,b,sel);

Portas (gates) primitivas  
são instanciadas e  
nomeadas  
*and* *g1(f1,a,n\_sel),*  
*g2(f2,b,sel);*  
*or* *g3(f,f1,f2);*  
*not* *g4(n\_sel,sel);*  
Conexões  
elétricas são  
nomeadas  
*endmodule*

# Aprendendo a usar a ferramenta em etapas

Visão em nível de portas lógicas desta implementação

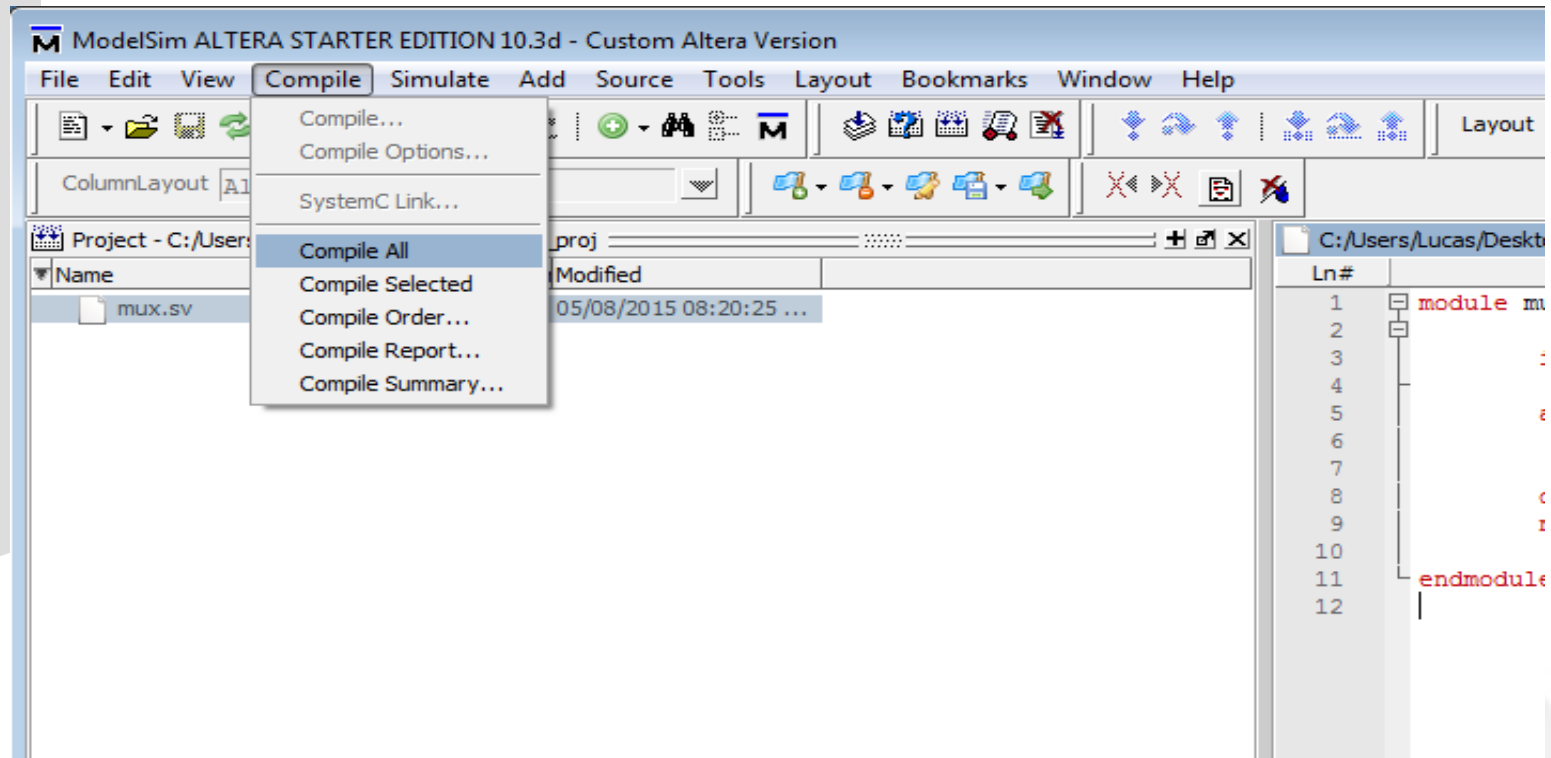


$$f = a \bullet \text{sel}' + b \bullet \text{sel}$$



# 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**

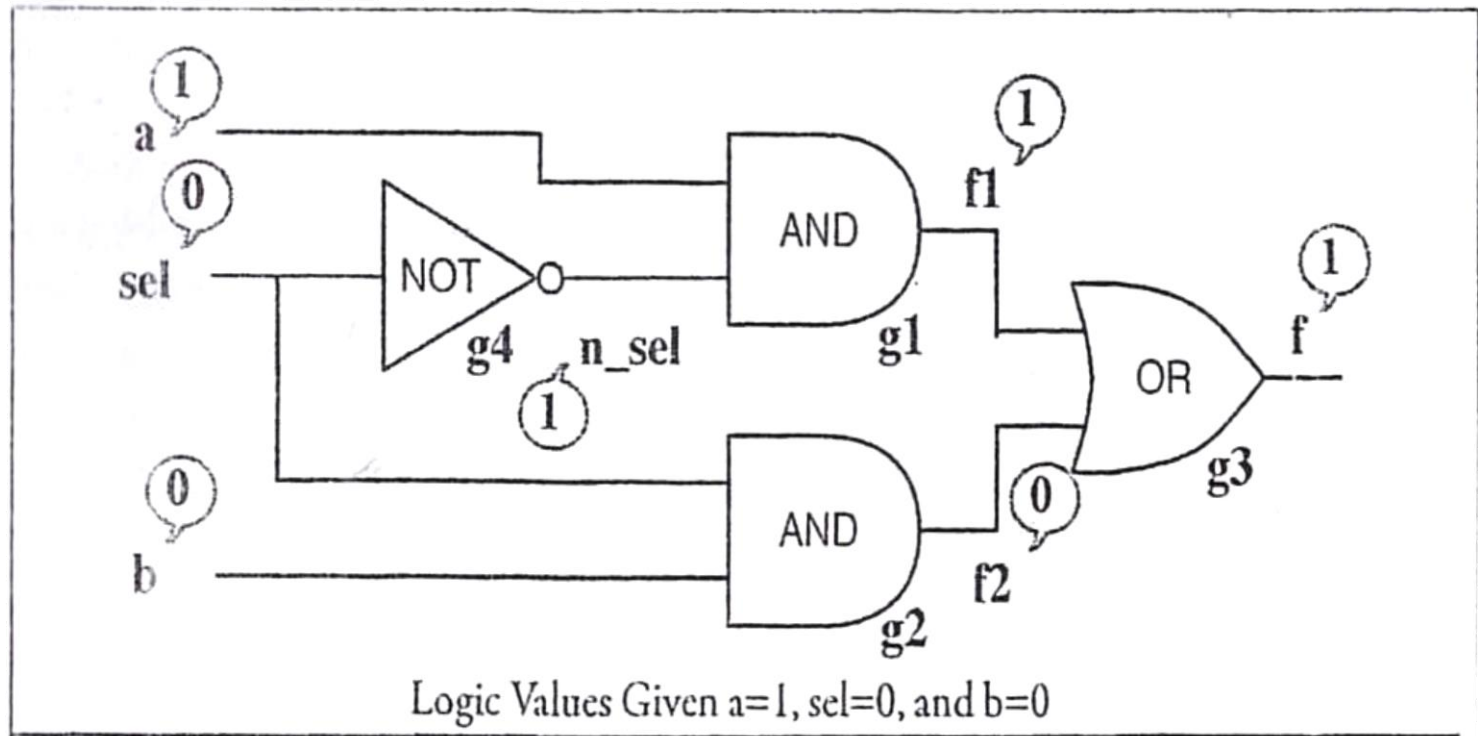
# Aprendendo a usar a ferramenta em etapas

O que falta para podermos simular o  
código  
?

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

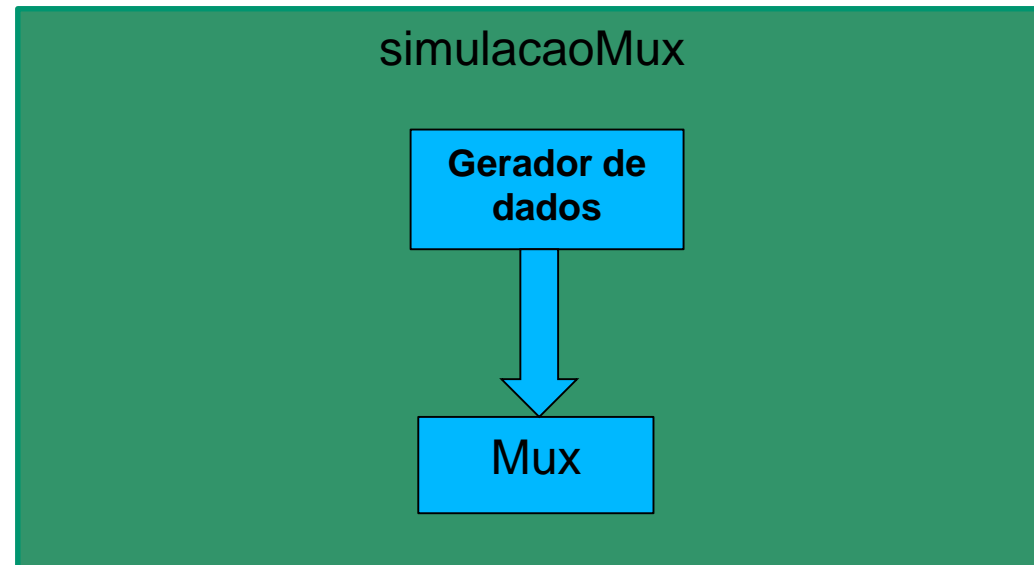
# Aprendendo a usar a ferramenta em etapas

Exemplo do comportamento do módulo para um dado de entrada



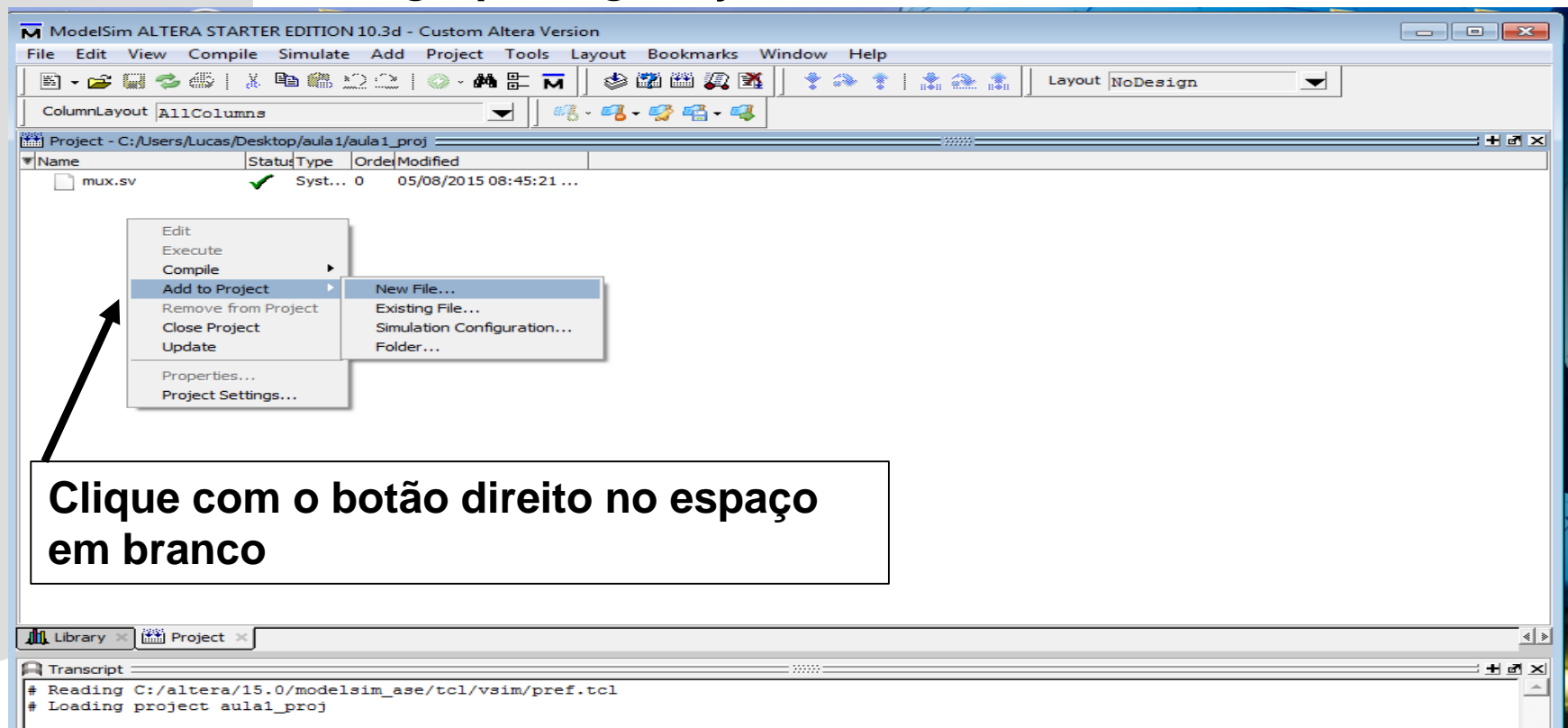
# Aprendendo a usar a ferramenta em etapas

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



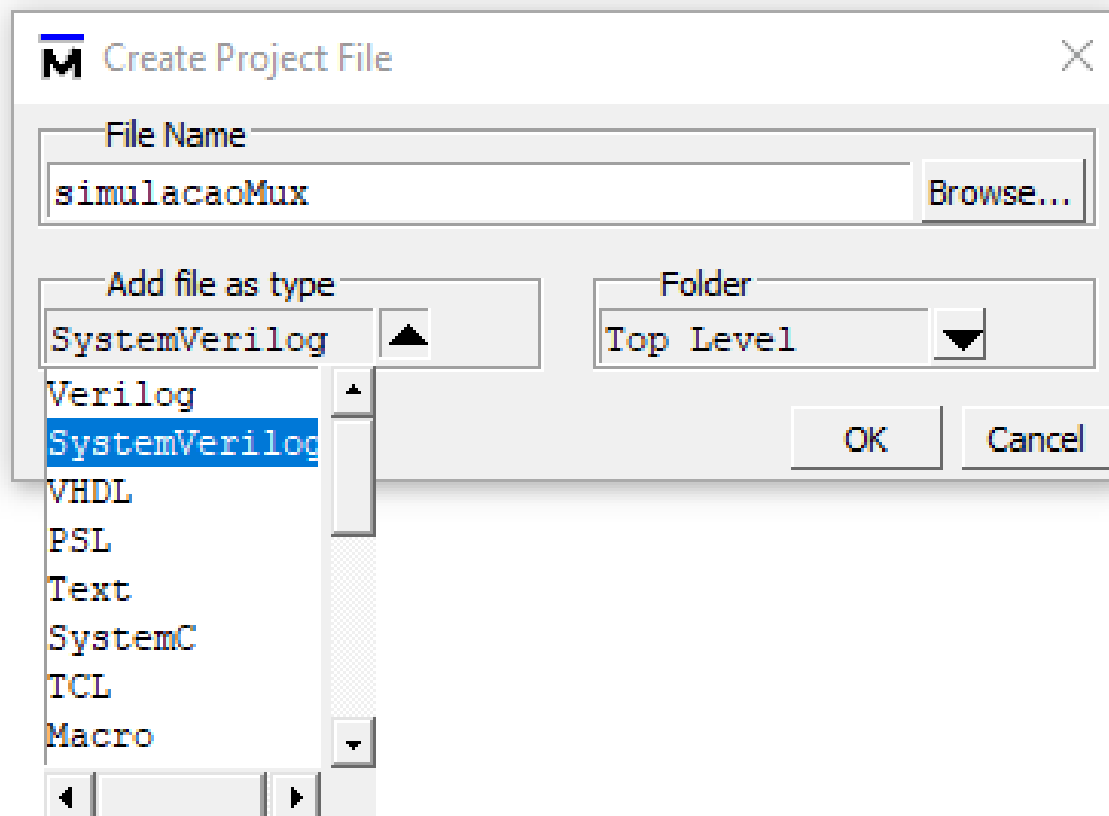
# Aprendendo a usar a ferramenta em etapas

## 4) Adicionando um novo arquivo .sv que contem 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 instancia 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;
```

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

```
  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

## 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.

```
initial begin
```

```
    $monitor($time,"a b sel = %b, muxOut = %b",  
count, muxOut);
```

```
    for(count = 0; count != 3'b111; count++) #10;
```

```
    #10 $stop;
```

```
end
```

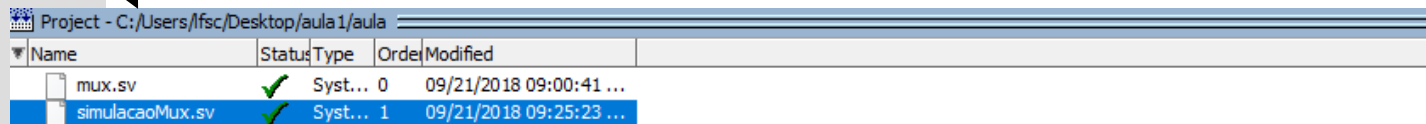
- **\$stop**

- Termina a simulação

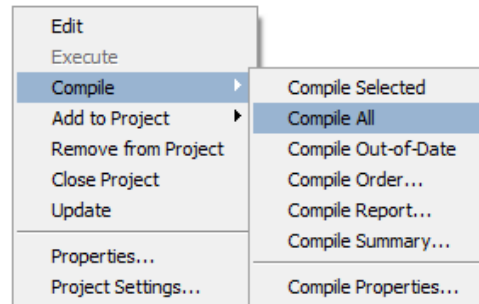
# Criando um módulo gerador

Dê um  
compile All

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

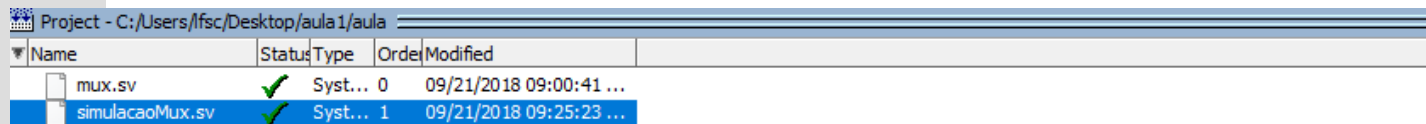


Name	Status	Type	Order	Modified
mux.sv	✓	Syst...	0	09/21/2018 09:00:41 ...
simulacaoMux.sv	✓	Syst...	1	09/21/2018 09:25:23 ...



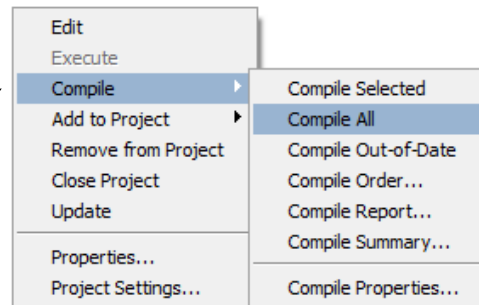
# Criando um módulo gerador

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



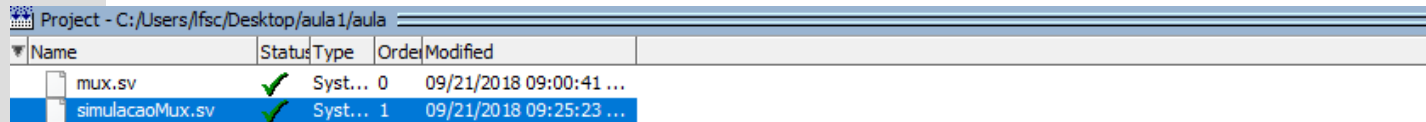
Name	Status	Type	Order	Modified
mux.sv	✓	Syst...	0	09/21/2018 09:00:41 ...
simulacaoMux.sv	✓	Syst...	1	09/21/2018 09:25:23 ...

**Clique com  
o botão  
direito >>  
Compile >>  
Compile All**



# Criando um módulo gerador

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

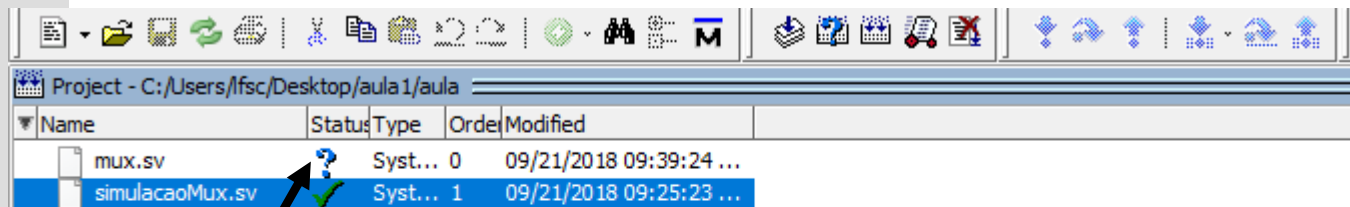


Name	Status	Type	Order	Modified
mux.sv	✓	Syst...	0	09/21/2018 09:00:41 ...
simulacaoMux.sv	✓	Syst...	1	09/21/2018 09:25:23 ...

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

# Criando um módulo gerador

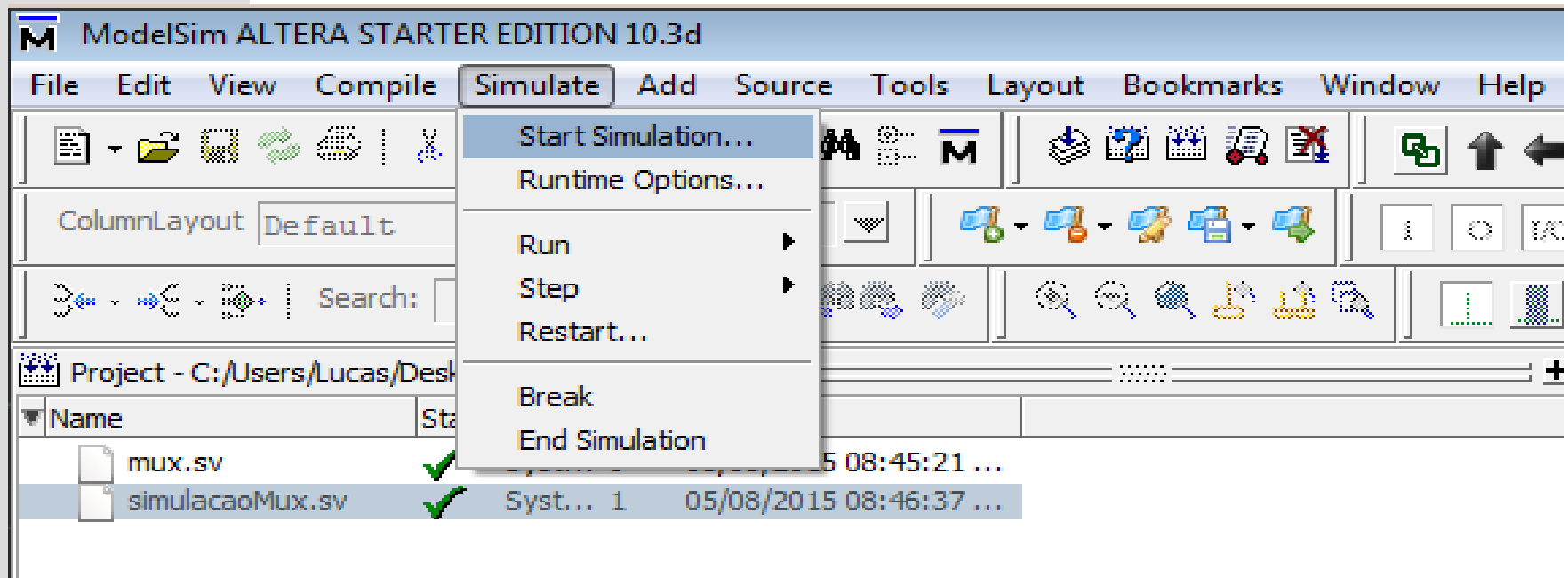
4) Adicionando um novo arquivo .sv que contem o código para geração de dados



Senão deverá  
aparecer uma  
interrogação ou  
x em vermelho

# Simulando o módulo com o gerador

## 5) Simulando o código



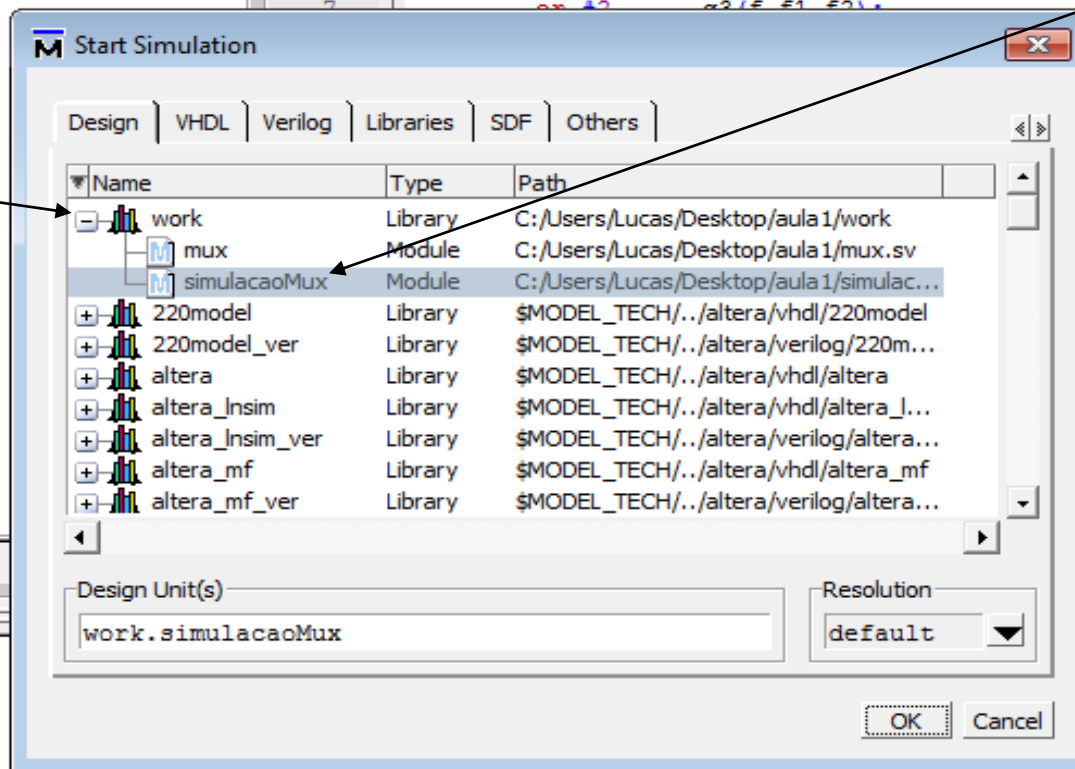


# Simulando o módulo com o gerador

## 5) Simulando o código

Ela está localizada no diretório work

Escolha o arquivo que irá gerar o sinais



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

# Simulando o módulo com o gerador

## 5) Simulando o código

The screenshot displays the VSIM6 simulation environment. The main window shows a project hierarchy for 'sim - Default'. The 'Instance' table lists the following components:

Instance	Design unit	Design unit type	Top Category	Visibility	Total coverage
simulacaoMux	simulacaoMux	Module	DU Instance	+acc=...	
dut	mux	Module	DU Instance	+acc=...	
g1	mux	Process	-	+acc=...	
g2	mux	Process	-	+acc=...	
g3	mux	Process	-	+acc=...	
g4	mux	Process	-	+acc=...	
#INITIAL#7	simulacaoMux	Process	-	+acc=...	
std	std	VPackage	Package	+acc=...	
#vsm_capacity#	Capacity	Statistics	-	+acc=...	

The 'Objects' window on the right shows the following objects:

Name	Value	Kind	Mode
count	xxx	Pack...	Internal
muxOut	x	Regi...	Internal

The 'Processes (Active)' window shows the following process:

Name	Type (filtered)	State	Order	Parent Path	Class Info
#INITIAL#7	Initial	Ready	13	/simulacaoMux	

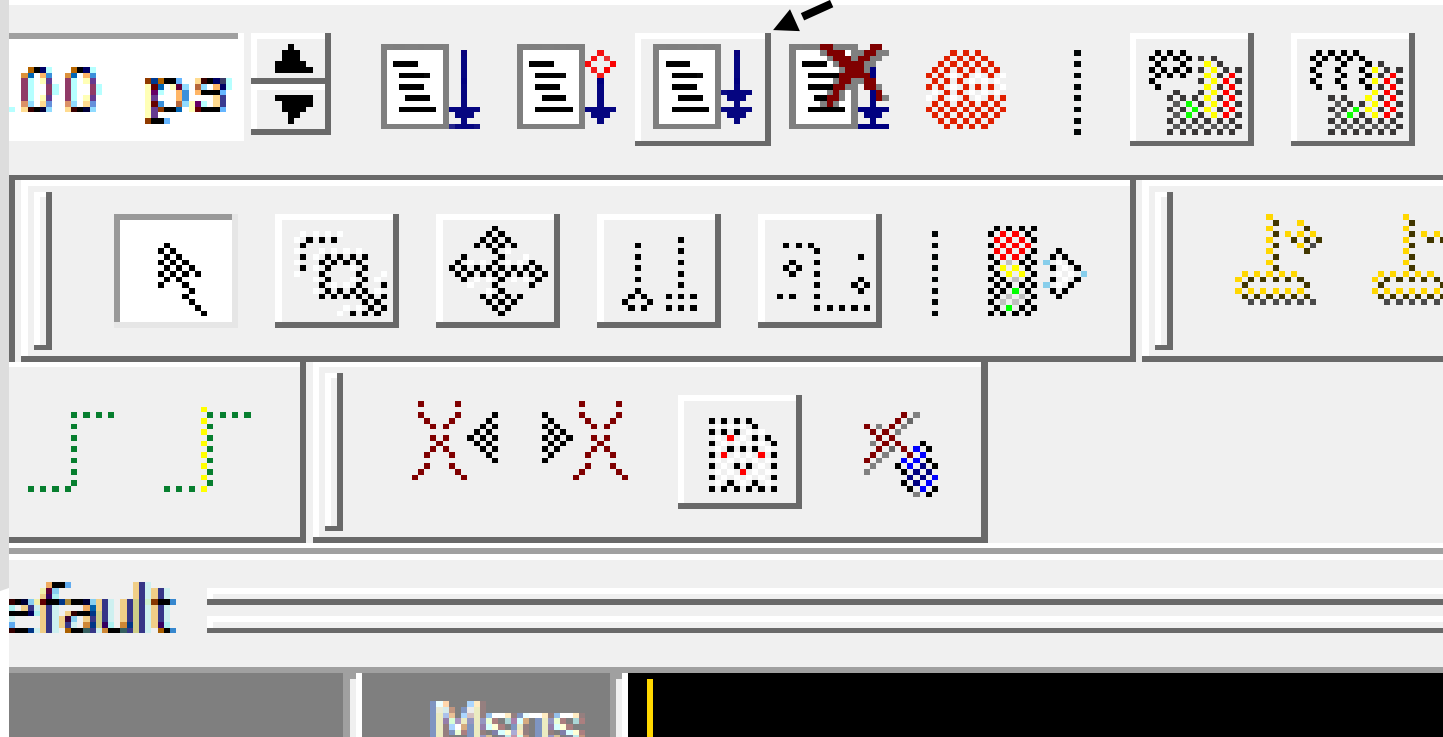
The 'Transcript' window at the bottom shows the following simulation log:

```
VSIM6> vsim -gui work.simulacaoMux
# End time: 09:52:15 on Sep 21, 2018, Elapsed time: 0:01:36
# Errors: 1, Warnings: 0
# vsim -gui work.simulacaoMux
# Start time: 09:52:15 on Sep 21, 2018
# Loading sv_std.std
# Loading work.simulacaoMux
# Loading work.mux
```

# Simulando o módulo com o gerador

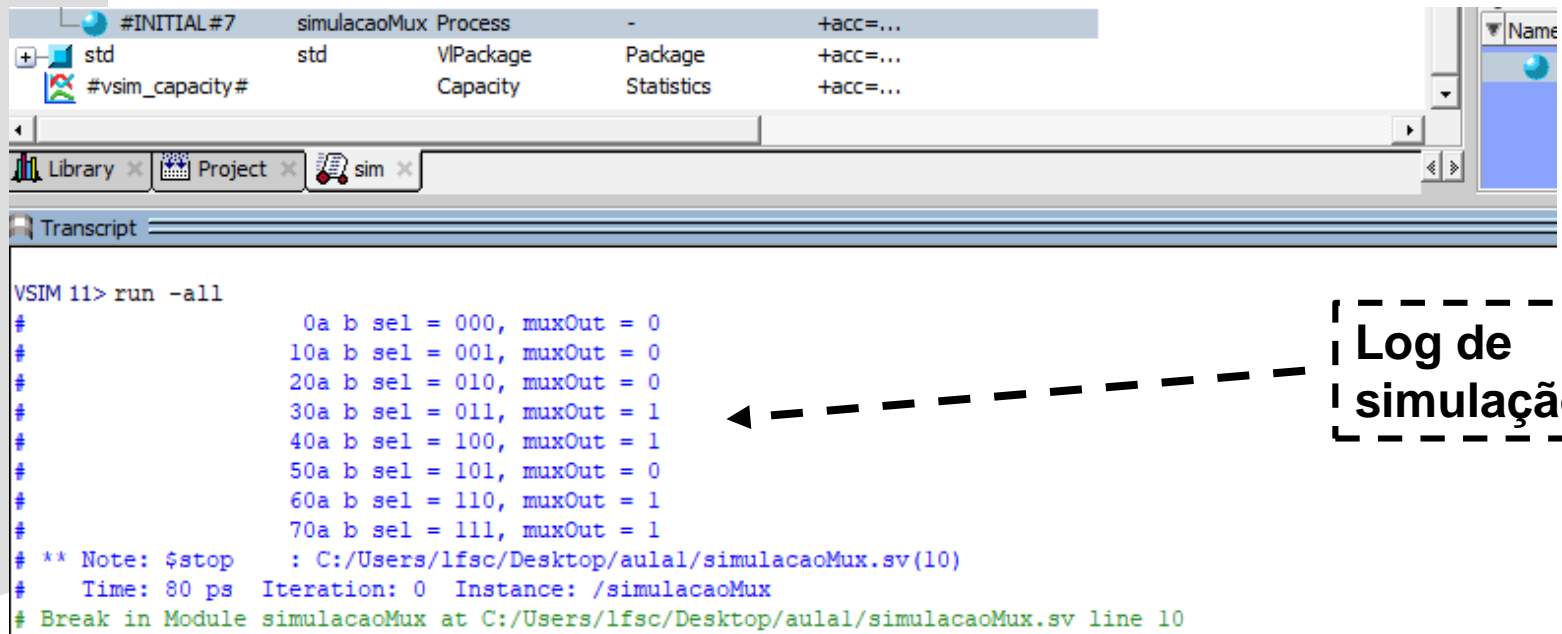
## 5) Simulando o código

Clicar  
(Run-all)



# Simulando o módulo com o gerador

## 5) Simulando o código



The screenshot shows a simulation window with a transcript of the simulation results. The transcript includes the command 'run -all' and a series of test vectors for a multiplexer. A dashed arrow points from a box labeled 'Log de simulação' to the transcript area.

```
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
```

Log de  
simulação

# Simulando o módulo com o gerador

## 5) Simulando o código

```
#          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
```

# Simulando o módulo com o gerador

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

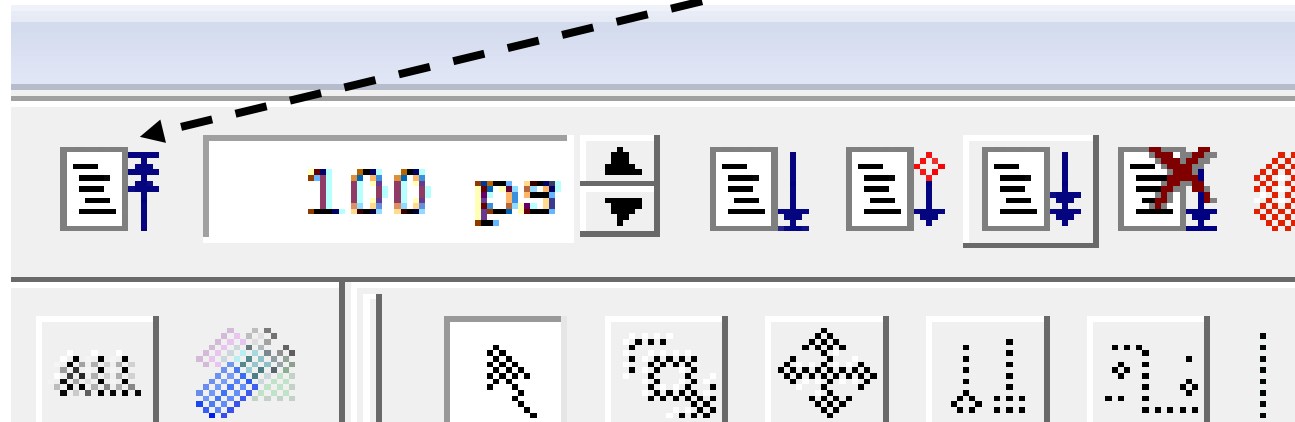
# Simulando o módulo com o gerador

**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

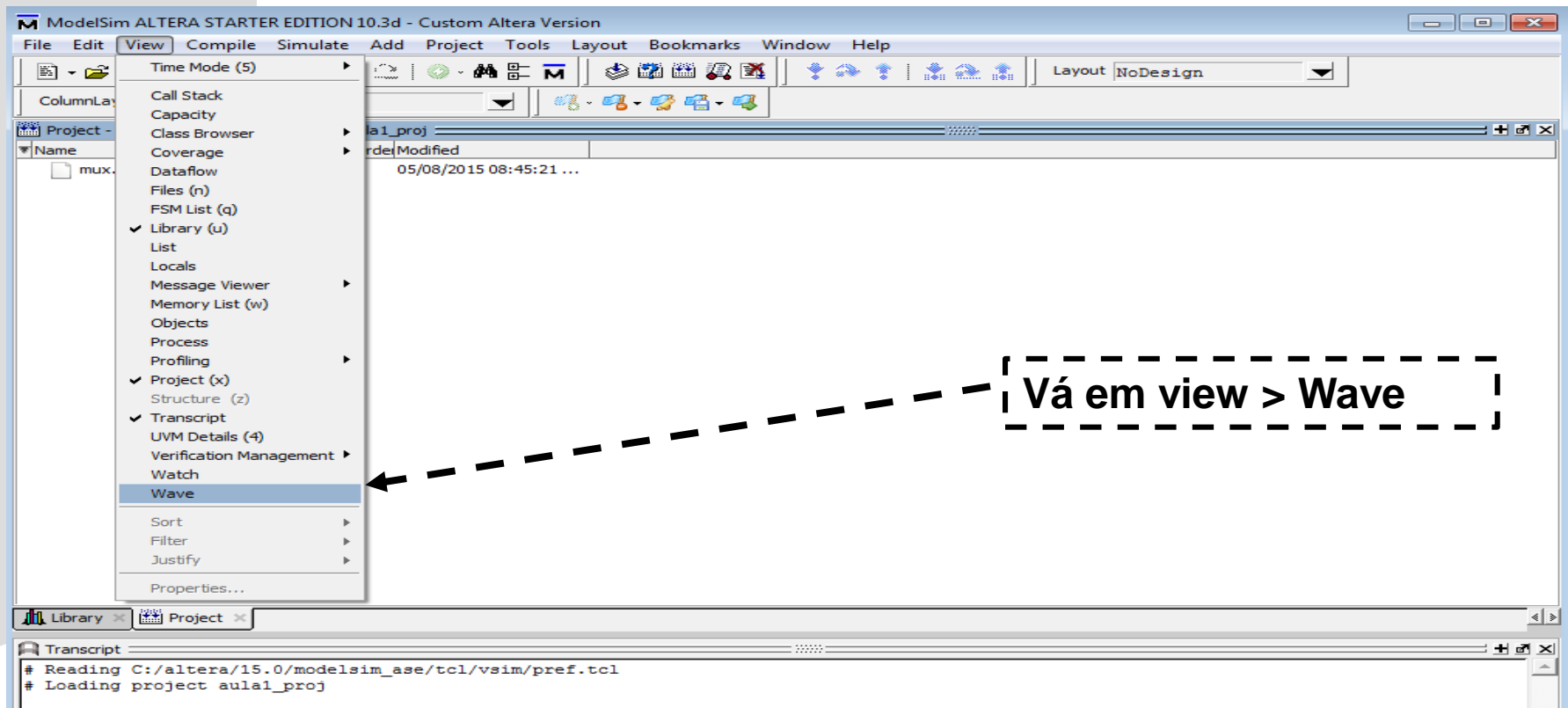
Clicar em  
*restart*





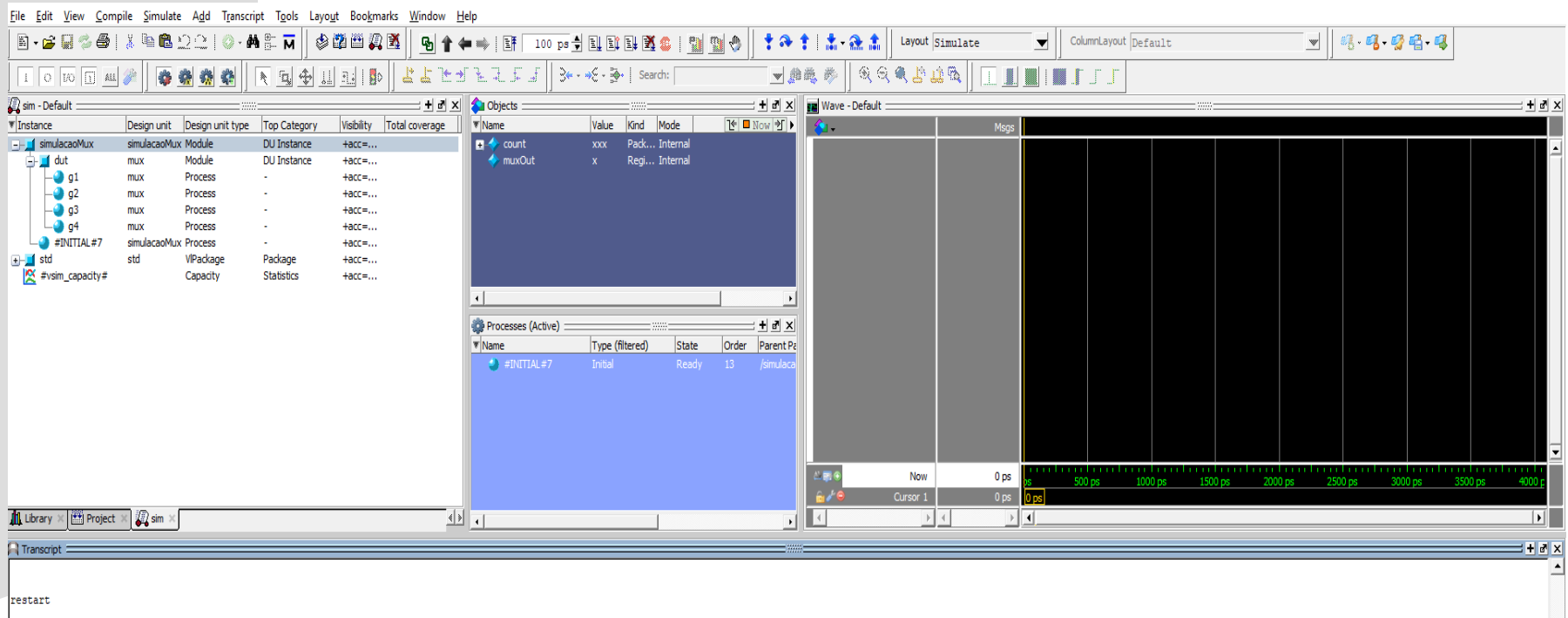
# Simulando com forma de onda

## 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

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

Layout Simulate ColumnLayout Default

sim - Default

Instance	Design unit	Design unit type	Top Category	Visibility	Total coverage
simulacaoMux	simulacaoMux Module	DU Instance	+acc=...		
dut	mux	Module	DU Instance	+acc=...	
g1	mux	Process	-	+acc=...	
g2	mux	Process	-	+acc=...	
g3	mux	Process	-	+acc=...	
g4	mux	Process	-	+acc=...	
#INITIAL#7	simulacaoMux Process	-	-	+acc=...	
std	std	VPackage	Package	+acc=...	
#sim_capacity#	Capacity	Statistics	+acc=...		

Objects

Name	Value	Kind	Mode
f	x	Reg...	Out
a	SBX	Net	In
b	SBX	Net	In
sel	SBX	Net	In
f1	SBX	Net	Internal
n_sel	SBX	Net	Internal
f2	SBX	Net	Internal

Processes (Active)

Name	Type (filtered)	State	Parent Pe
#INITIAL#7	Initial	Ready	13 simulacao

Wave - Default

Now 0 ps  
Cursor 1 0 ps

0 ps 500 ps 1000 ps 1500 ps 2000 ps 2500 ps 3000 ps 3500 ps 4000 ps

Selecione os sinais que deseja e arraste para o espaço ao lado

restart

# Simulando com forma de onda

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

The screenshot displays the ModelSim software interface. The top menu bar includes File, Edit, View, Compile, Simulate, Add, Transcript, Tools, Layout, Bookmarks, Window, and Help. The main workspace is divided into several panes:

- Instance:** A tree view showing the simulation hierarchy. The 'dut' (Design Under Test) is selected, showing its components: g1, g2, g3, g4, #INITIAL#7, std, and #vsm\_capacity#.
- Objects:** A table listing the objects in the simulation. The selected object is 'f', which is a Register (Reg) with an output (Out).
- Processes (Active):** A table showing the active processes. The selected process is '#INITIAL#7', which is in the 'Initial' state and is ready to execute.
- Wave - Default:** A window showing the waveform. The waveform is currently empty, with a time scale from 0 ps to 4000 ps. A dashed arrow points from the text box below to this window.

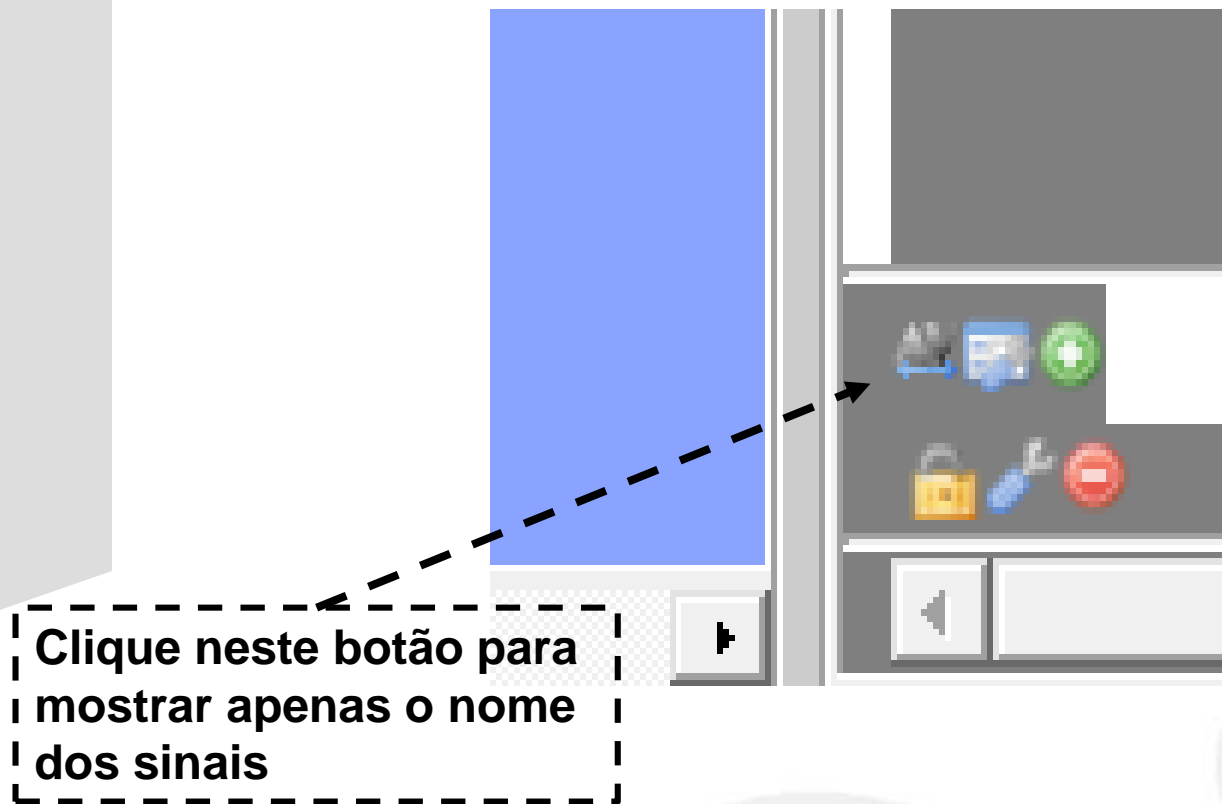
The Transcript window at the bottom shows the following commands:

```
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/se1  
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
```

Os sinais devem aparecer neste espaço

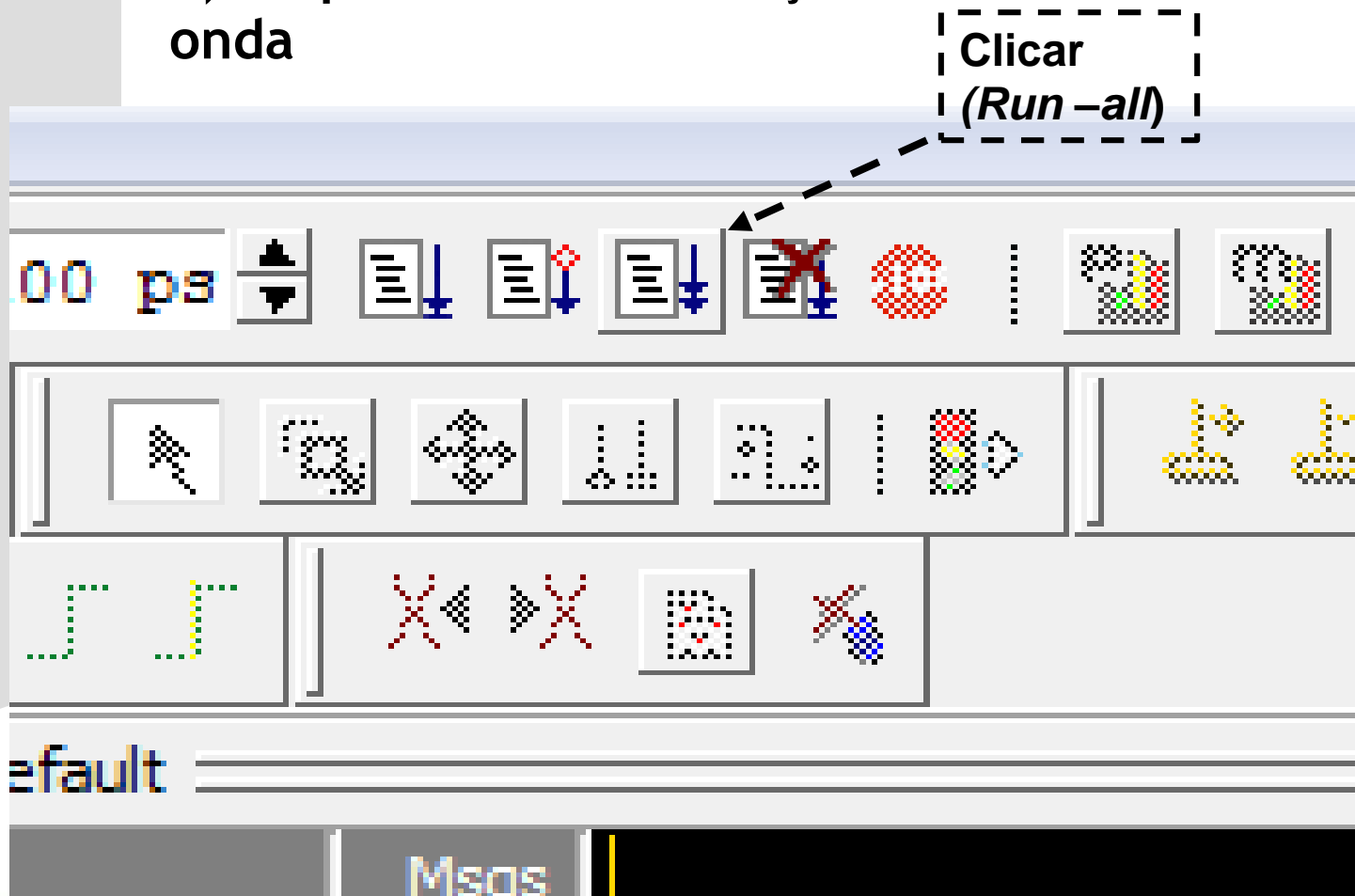
# Simulando com forma de onda

6) Repetindo a simulação e inserindo forma de onda



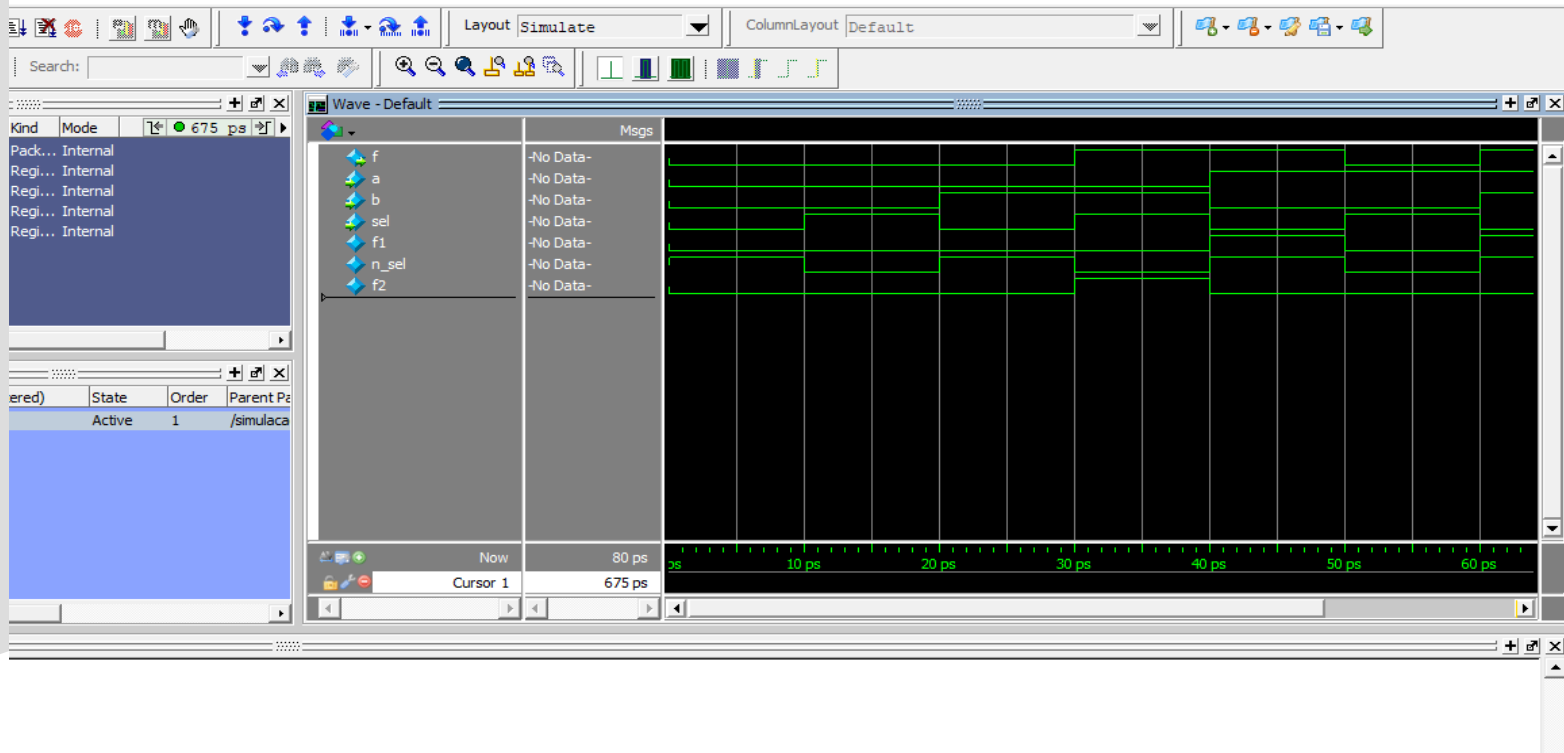
# Simulando com forma de onda

6) Repetindo a simulação e inserindo forma de onda



# Simulando com forma de onda

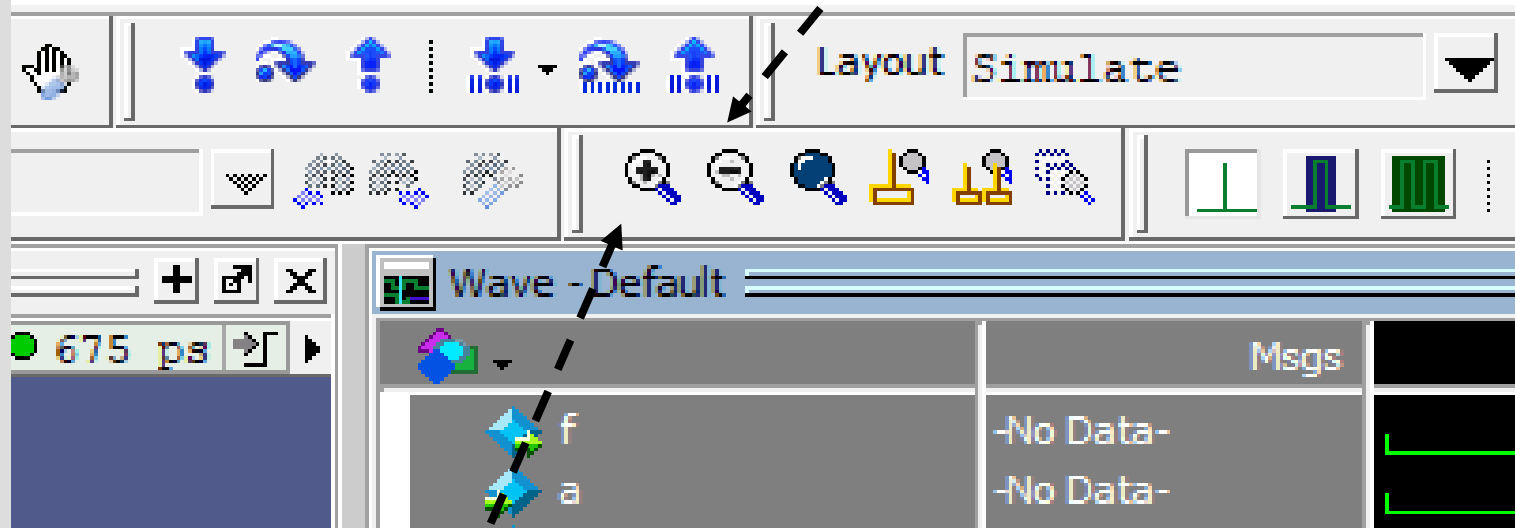
## 6) Repetindo a simulação e inserindo forma de onda



# Simulando com forma de onda

6) Repetindo a simulação e inserindo forma de onda

Para dar "Zoom out"  
aperte o botão O

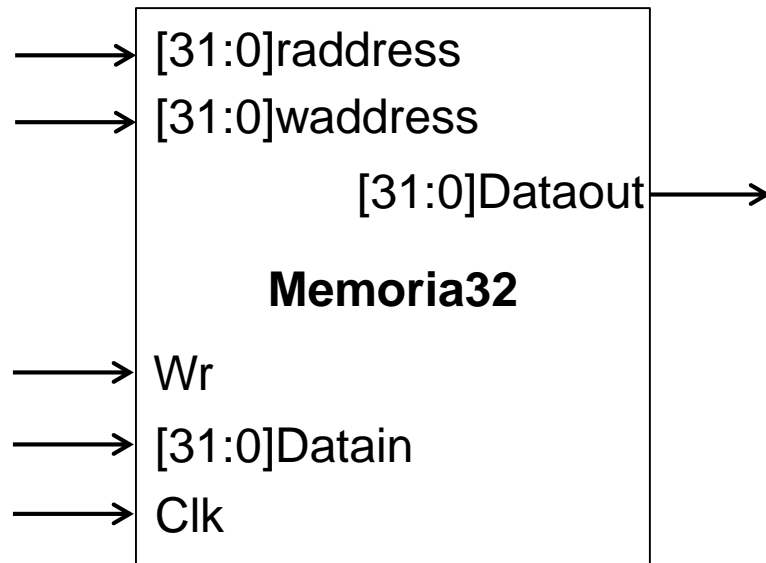


Para dar "Zoom in"  
aperte o botão I



# 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 = BIN;    -- The radix for data values
CONTENT
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 = BIN;    -- The radix for data values
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 = BIN;    -- The radix for data values
CONTENT              -- start of (address : data pairs)
```

BEGIN

000: 00000000;

001: 10100111;

002: 00011000;

003: 00100000;

004: 00000001;

005: 00001001;

006: 00111000;

007: 00100010;

END;

← Início do  
conteúdo em  
memória

← 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 = BIN;    -- The radix for data values
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 instruction.mif**

# Simulando uma memória de 32 bits

Baixe o arquivo “projeto.zip”

Localizado em:

[www.cin.ufpe.br/~if674/arquivos/2018.2/Projeto/projeto.zip](http://www.cin.ufpe.br/~if674/arquivos/2018.2/Projeto/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

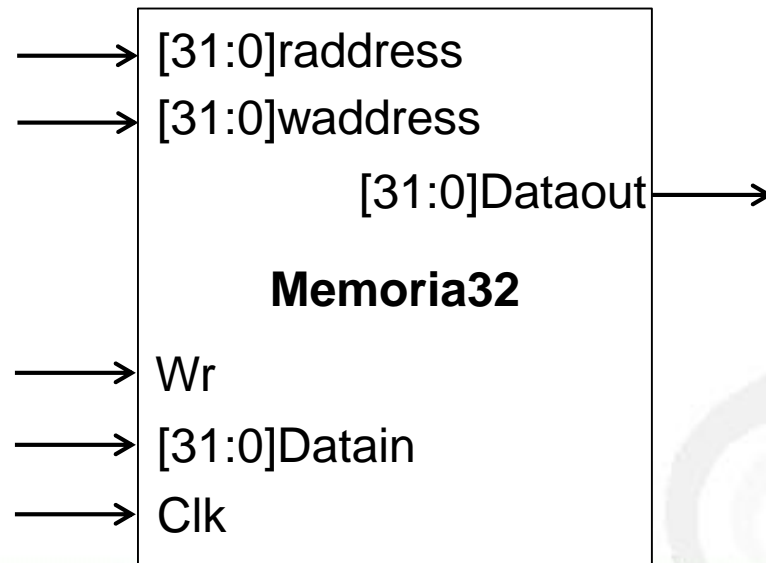
instruction.mif

para a pasta do seu projeto do modelsim.



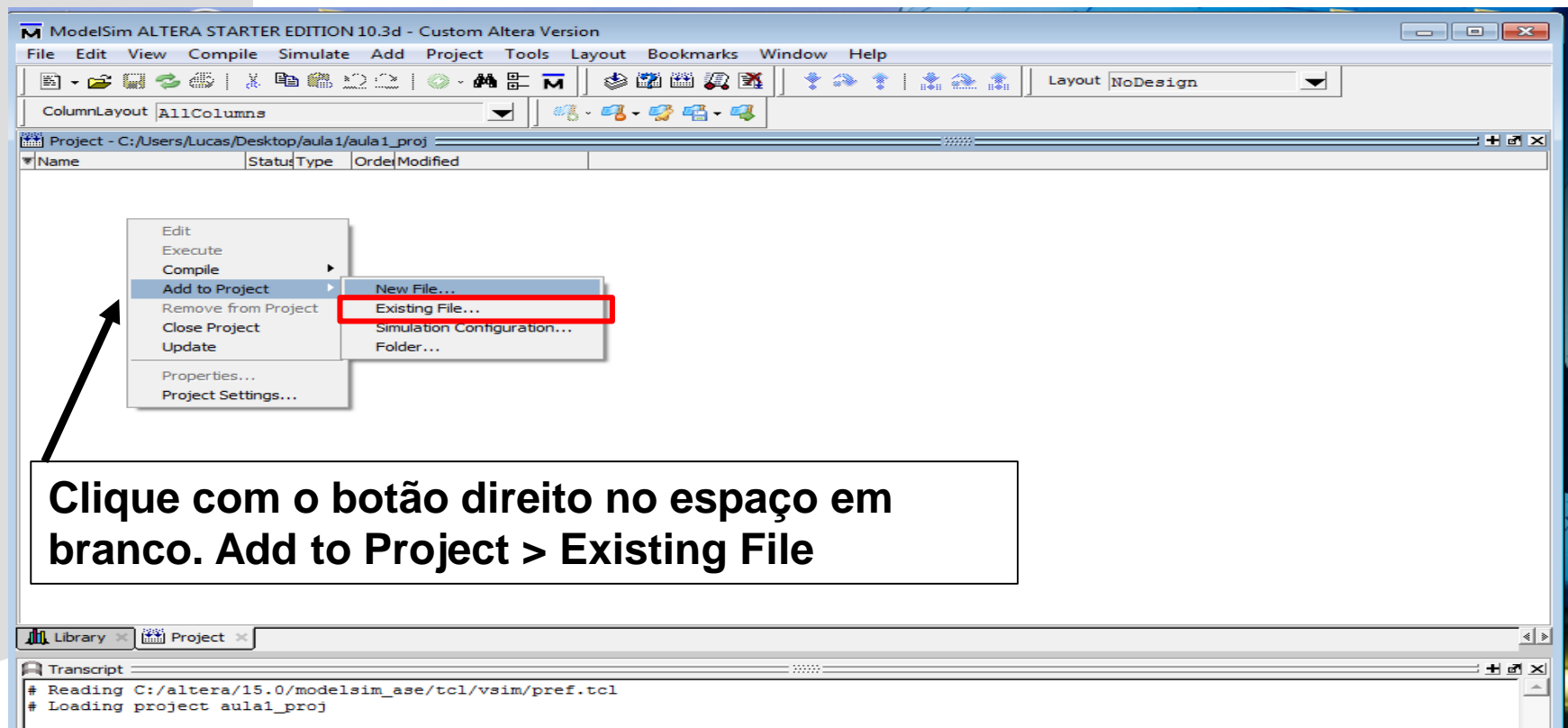
# Simulando uma memória de 32 bits

- O arquivo ramOnChip32.v contem 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:



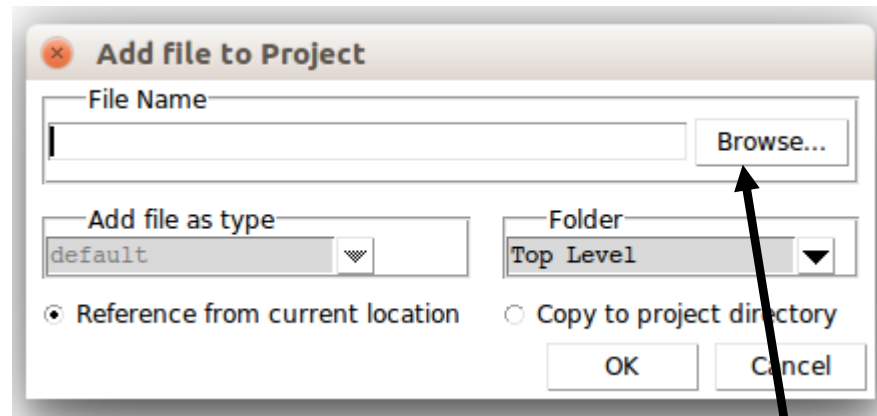
# Simulando uma memória de 32 bits

## Adicionando módulos existentes



# Simulando uma memória de 32 bits

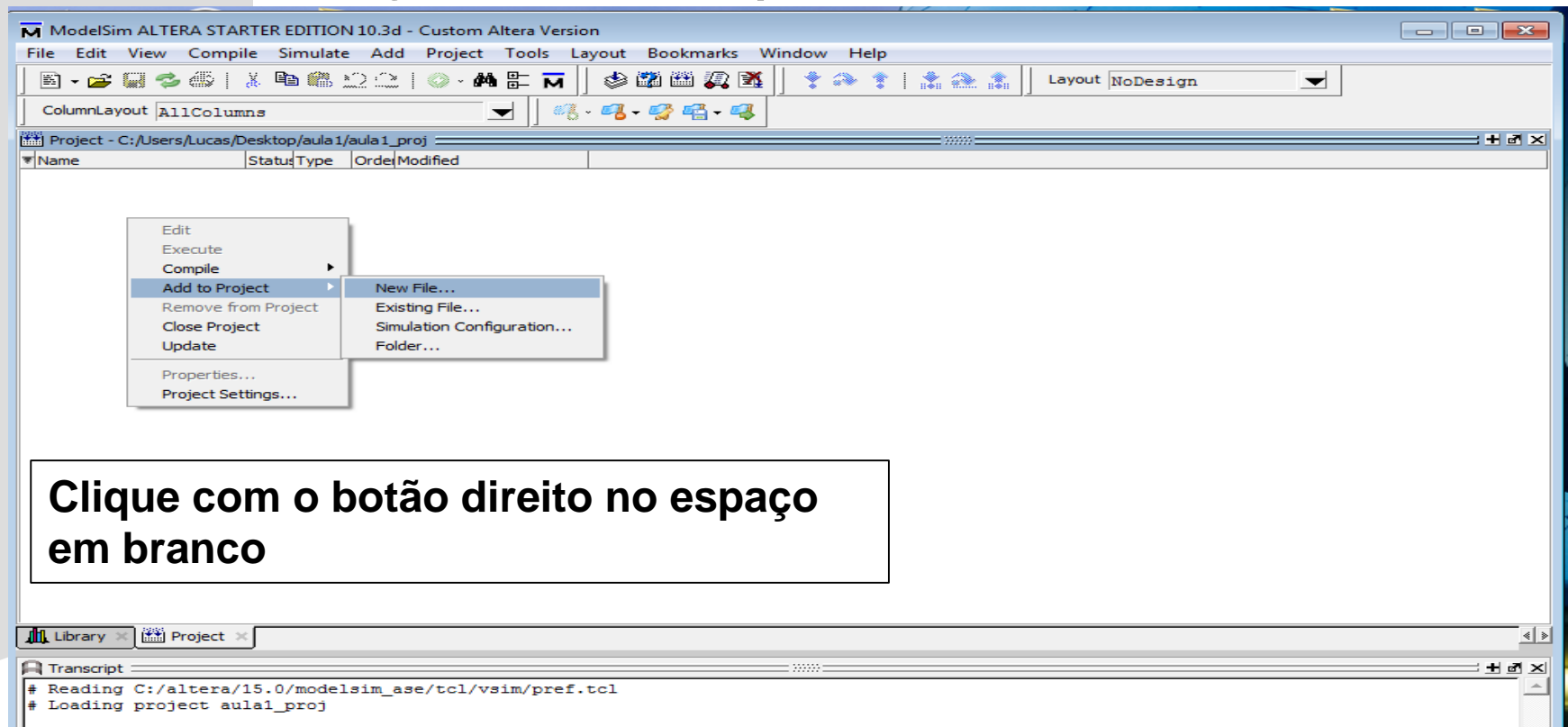
## Adicionando módulos existentes



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

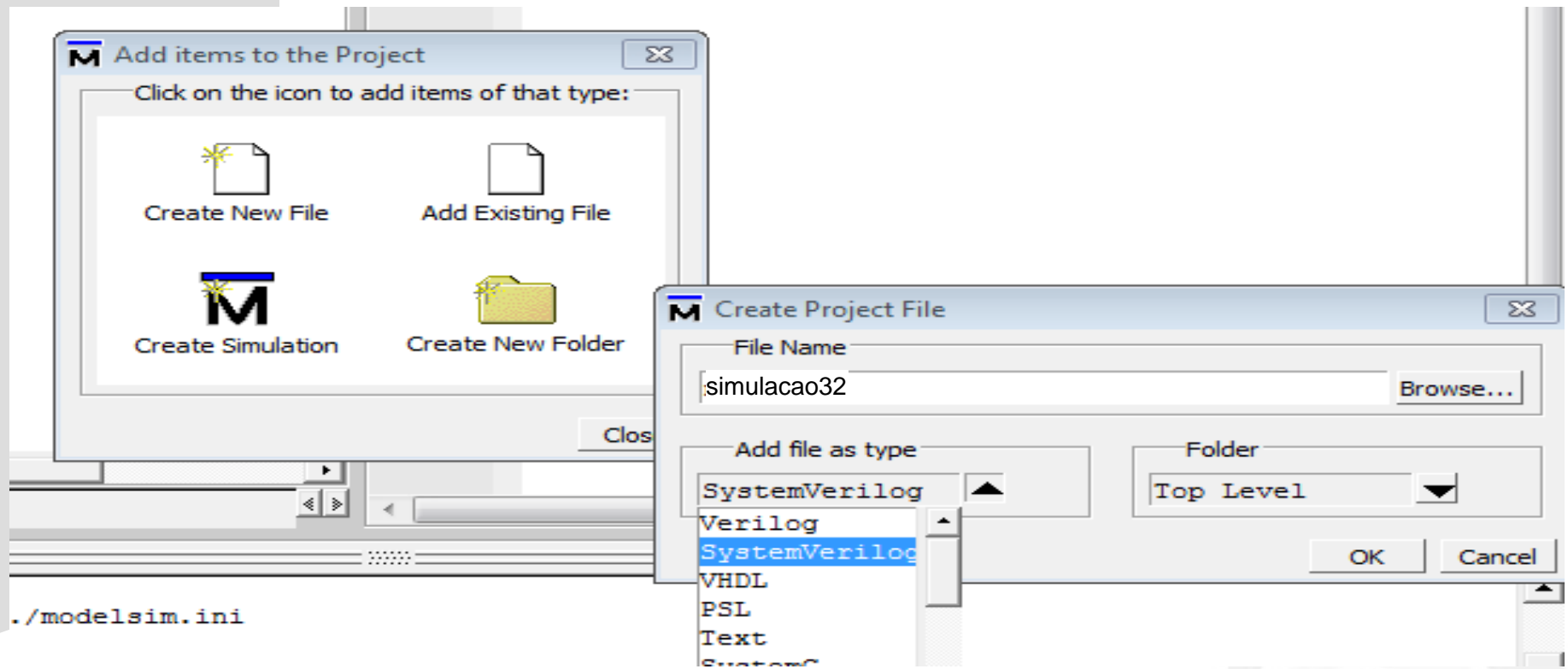
# Simulando uma memória de 32 bits

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



# Simulando uma memória de 32 bits

Dê o nome de simulacao32



# 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;
```

```
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;
```

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;
```

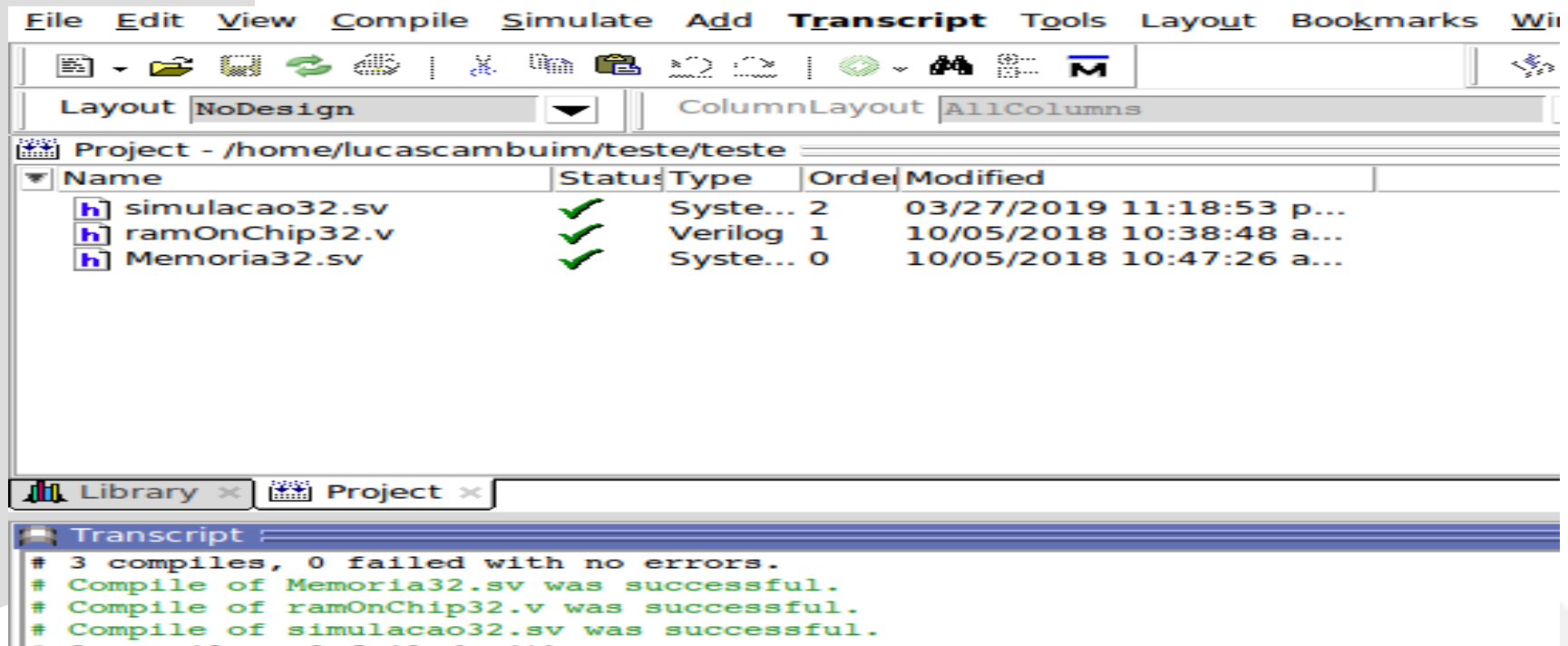
# Simulando uma memória de 32 bits

**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
end
endmodule
```

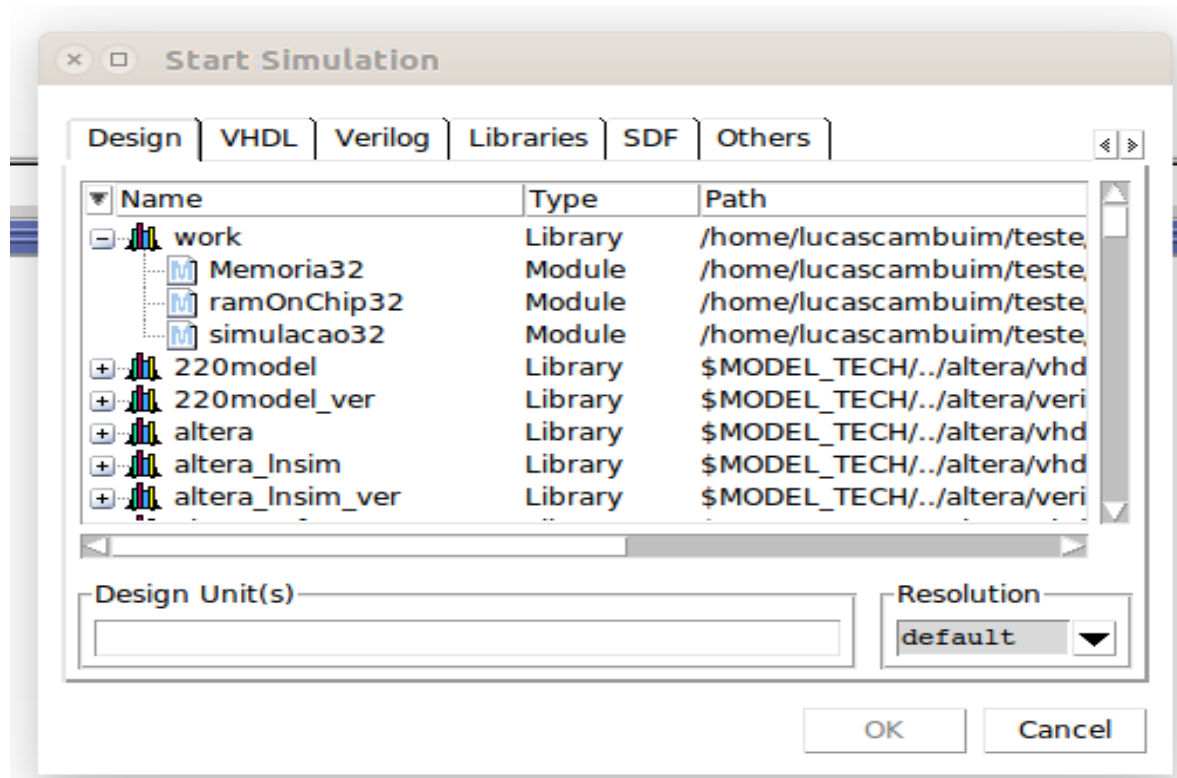
# Simulando uma memória de 32 bits

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



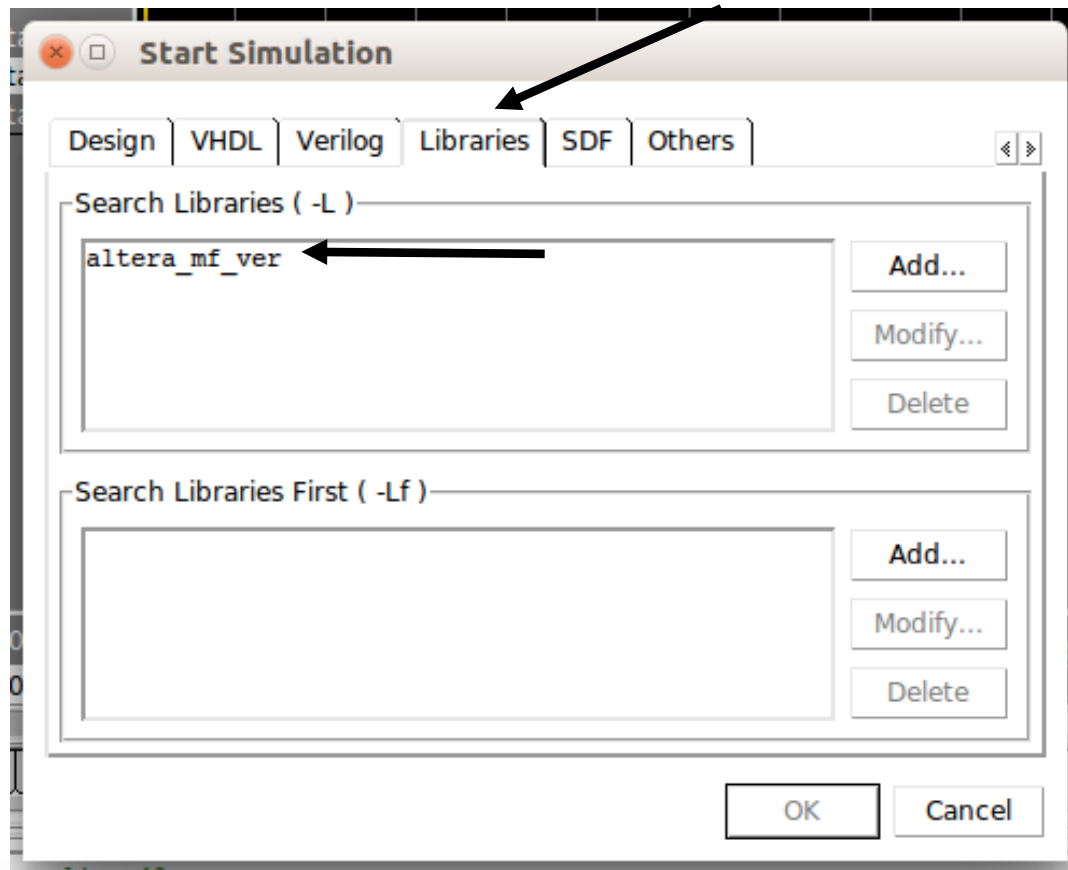
# Simulando uma memória de 32 bits

Em seguida vá em Simulate > Start Simulate



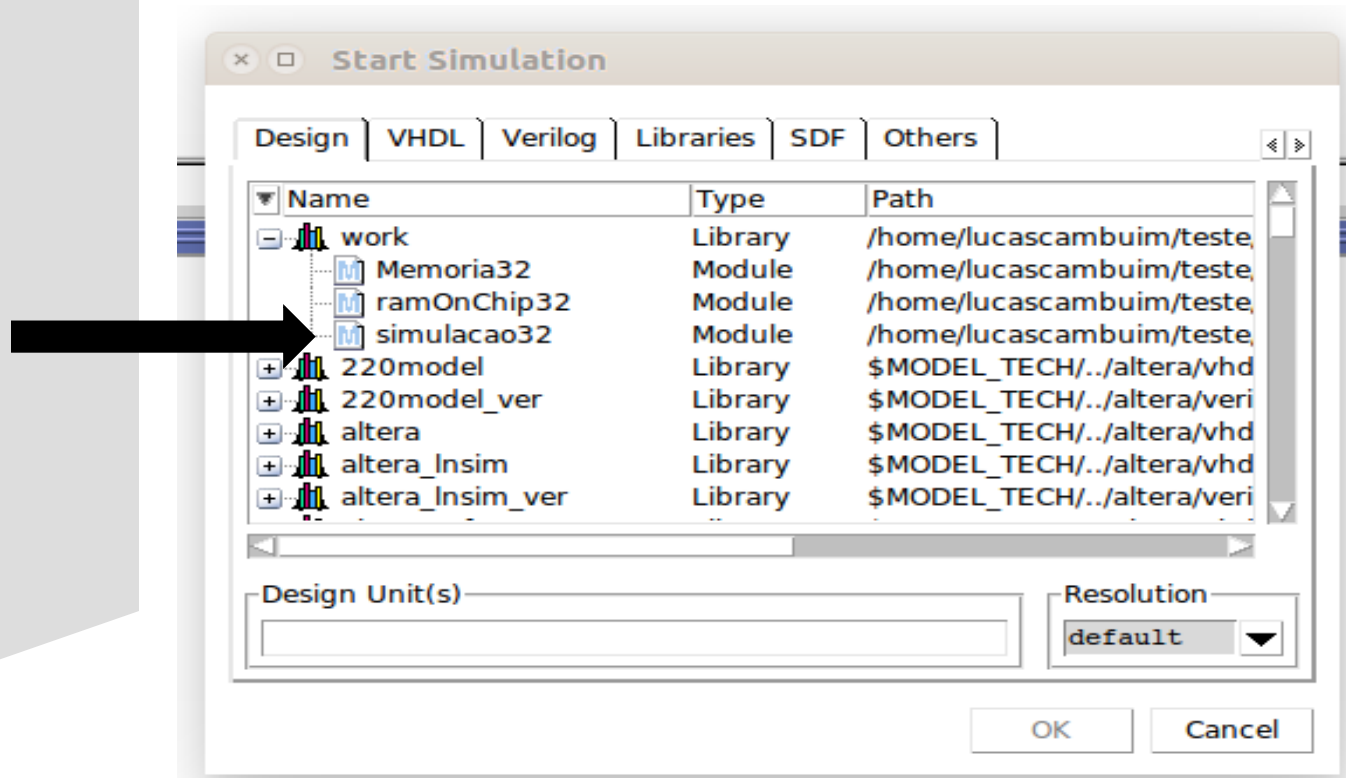
# Simulando uma memória de 32 bits

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



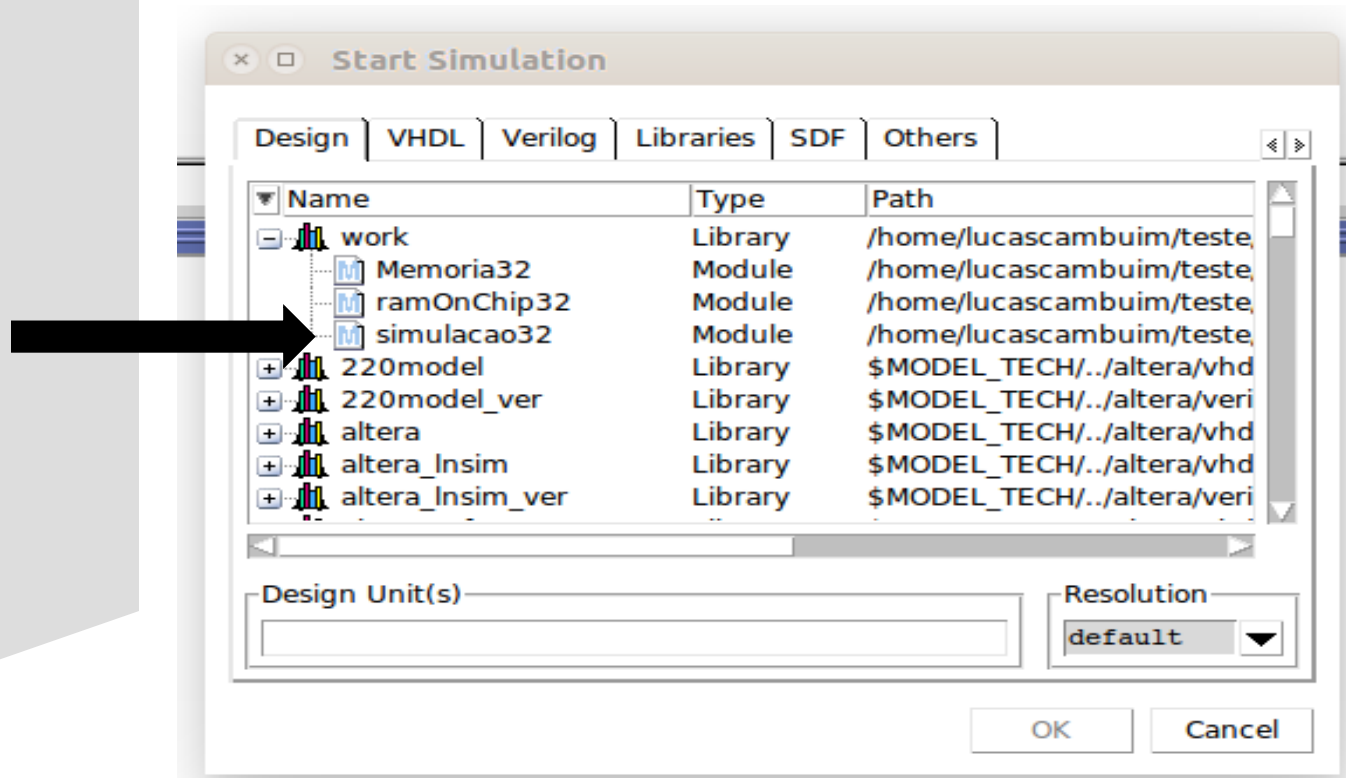
# Simulando uma memória de 32 bits

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

The screenshot displays a simulation tool interface with two main panels: 'Objects' and 'Wave - Default'.

**Objects Panel:** Lists various signals and parameters. The 'q' signal is highlighted at the bottom.

Name
CLKPERIOD
CLKDELAY
clk
nrst
rdaddress
wdaddress
data
Wr
q

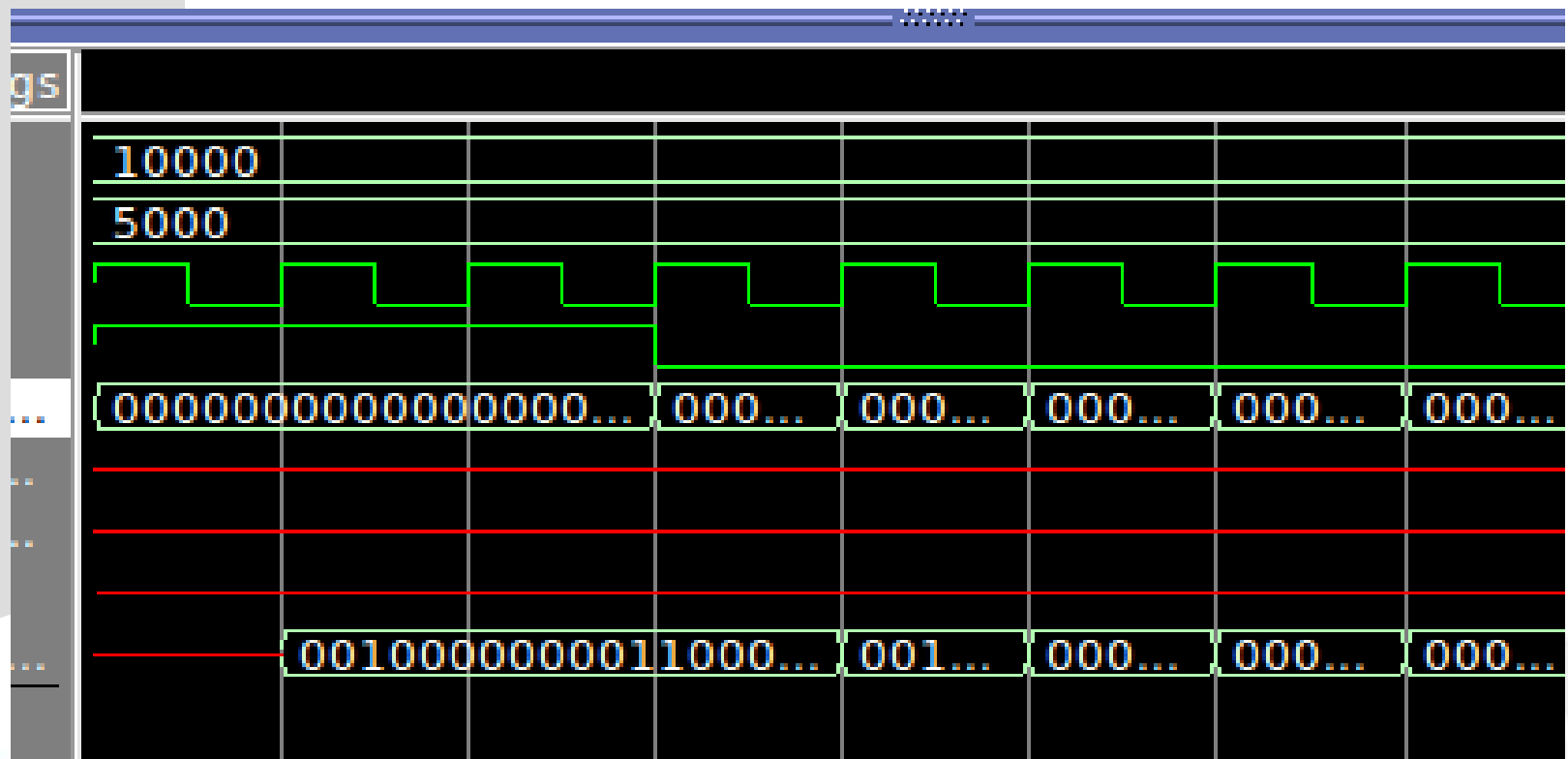
**Wave - Default Panel:** Shows a table of signal values over time. The 'q' signal is highlighted at the bottom.

Signal	Value
CLKPERIOD	10000
CLKDELAY	5000
clk	x
nrst	x
rdaddress	x
wdaddress	XXXXXXXXXX...
data	XXXXXXXXXX...
Wr	x
q	XXXXXXXXXX...

A curved arrow points from the 'q' signal in the 'Objects' panel to the 'q' signal in the 'Wave - Default' panel.

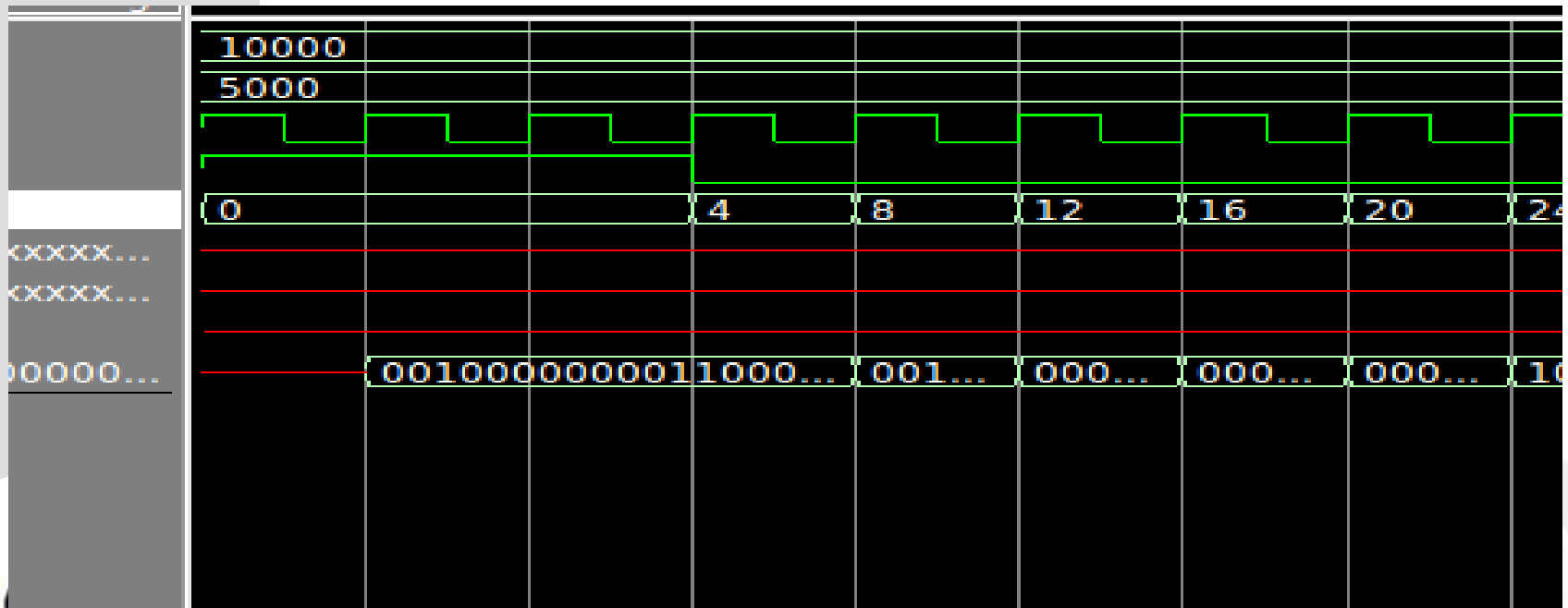
# Simulando uma memória de 32 bits

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



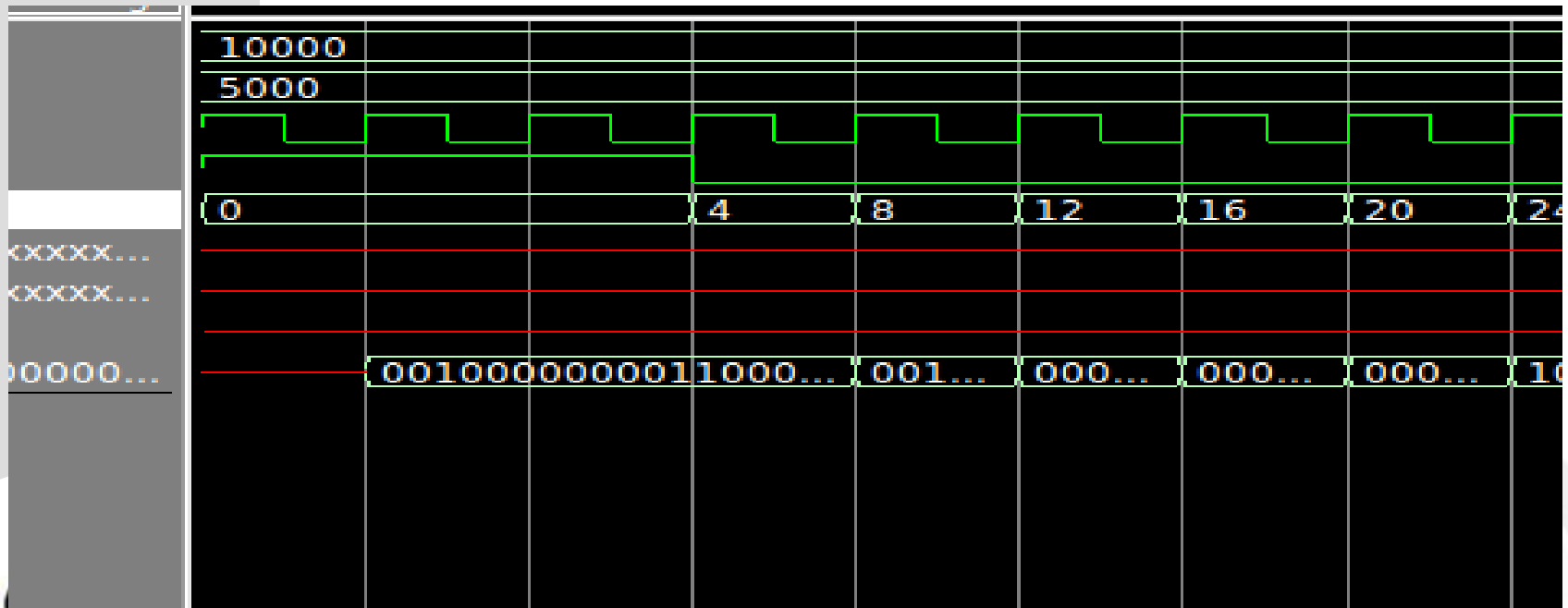
# Simulando uma memória de 32 bits

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 instruction.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 altera_mf_ver -L lpm_ver -L sgate_ver -L 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 -L lpm_ver -L sgate_ver -L altera_ver -novopt
work.simulacao32
```

Este nome tem que ser modulo top que gera os sinais

```
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
```

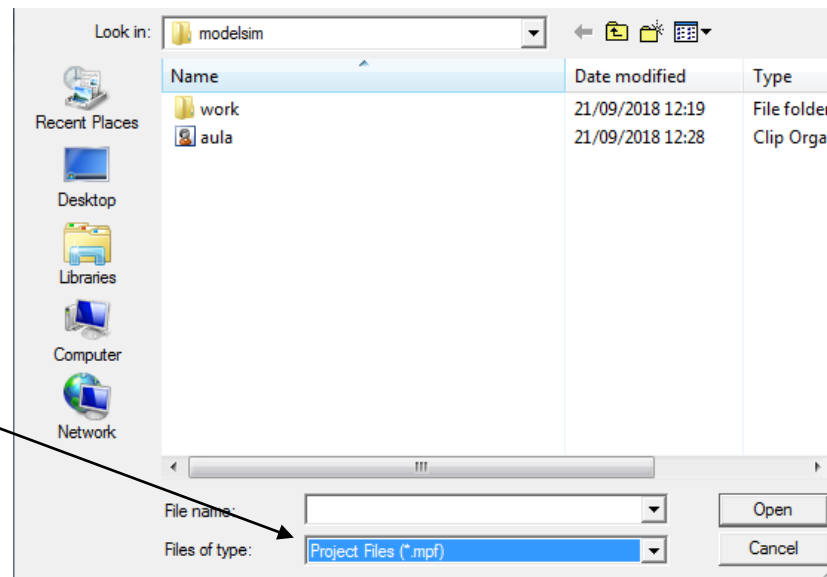
# Simulando a memória usando script

## 7) Usando memória e script do modelsim

### 7.1 Abra o modelSim

### 7.2 Clique em File > open

### 7.3 Localize o arquivo projeto/modelsim/projeto.mpf



Coloque para visualizar arquivos .mpf

# Simulando a memória usando script

## 7) Usando memória e script do modelsim

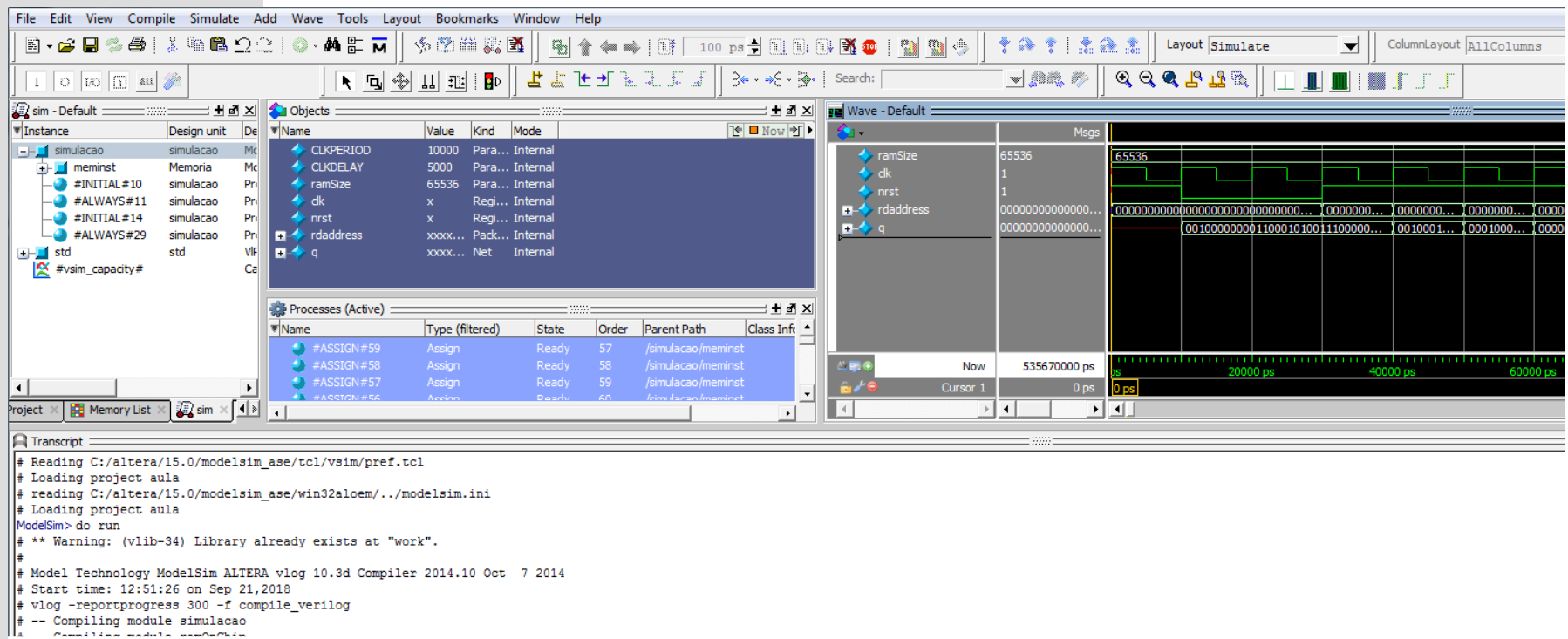
Digite no espaço “transcript” o seguinte comando:  
  
do runmemoria32

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

## 7) Usando memória e script do modelsim

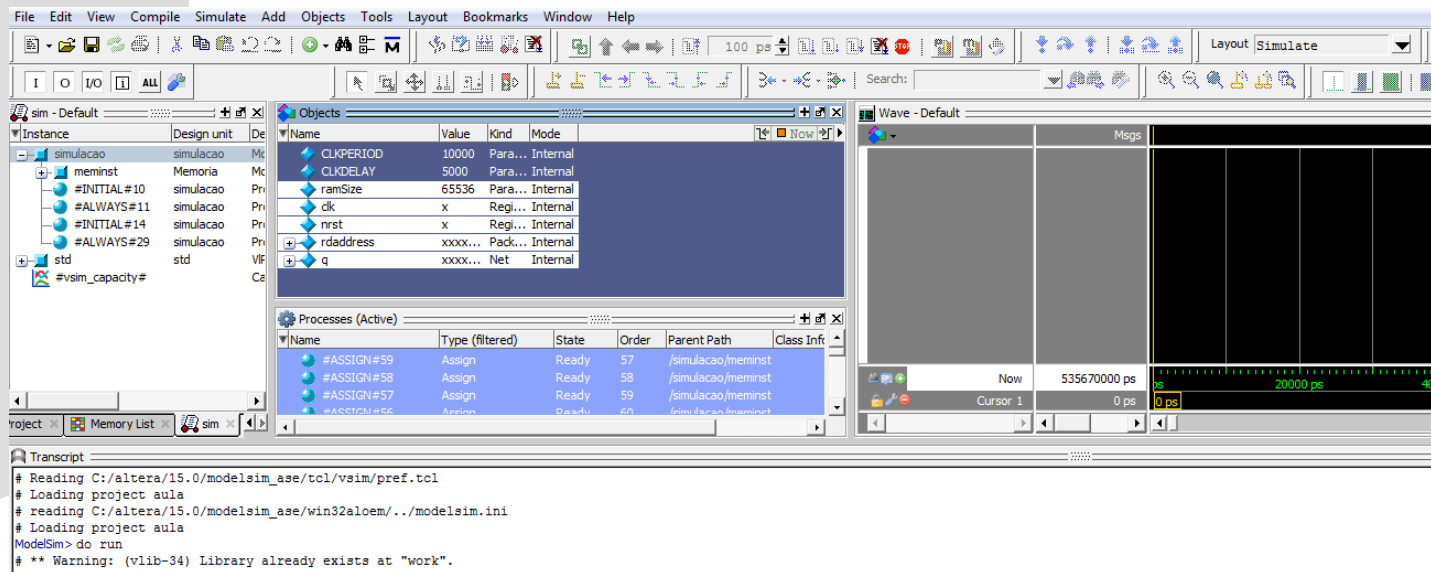


# Simulando a memória usando script

## 7) Usando memória e script do modelsim

Se quiser adicionar sinais ao script

### 1º) Adicione manualmente

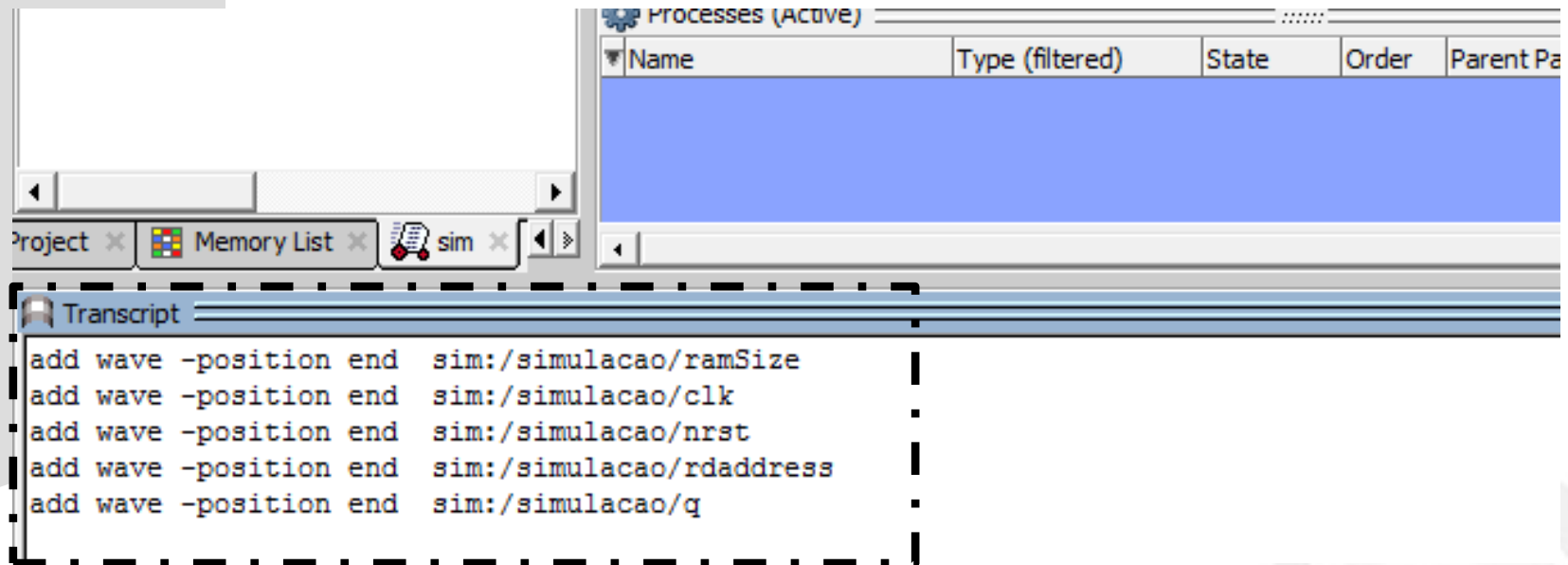


# Simulando a memória usando script

## 7) Usando memória e script do modelsim

Se quiser adicionar sinais ao script

2º) Copie os comandos





# 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
```

# Aprendendo a utilizar a Ferramenta Modelsim

Professor: Lucas Cambuim (lfsc)

