



# Esterel Technologies Training Material

Lab #1 - A Simple RAM  
For Academic Program Only

# Restricted Usage Conditions

- This material is provided to help academics build Esterel Studio and Esterel language training courses
- Usage is exclusively restricted to teaching activities for university and training institution students
- No industrial usage authorized
- Please, give us your feed-back, tell us about further needs and share other training material and labs.



**Sylvie Aldebert**

Academic Program Coordinator

Esterel Technologies

TWINS 1 - 679 avenue Julien Lefèbvre

06270 VILLENEUVE-LOUBET

FRANCE

**Phone:** +33 (0)4 92 02 40 40

**Phone:** +33 (0)4 92 02 40 64

**Email:** [academic@esterel-technologies.com](mailto:academic@esterel-technologies.com)



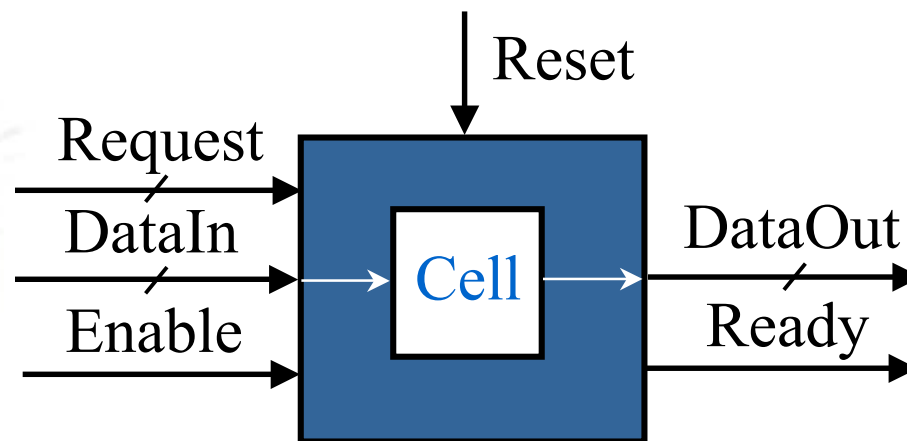
# A Simple RAM Esterel Language Version

## Simple RAM Specifications

The Simple RAM is implemented using one memory cell

The Simple RAM functions:

- if the **WriteReq** signal is present, write the data present on the **DataIn** bus in the memory cell
- if the **ReadReq** signal is present, read the content of the memory cell and send it through the **DataOut** bus
- read and write requests may occur **simultaneously**.  
In this case the read data will be the written one.



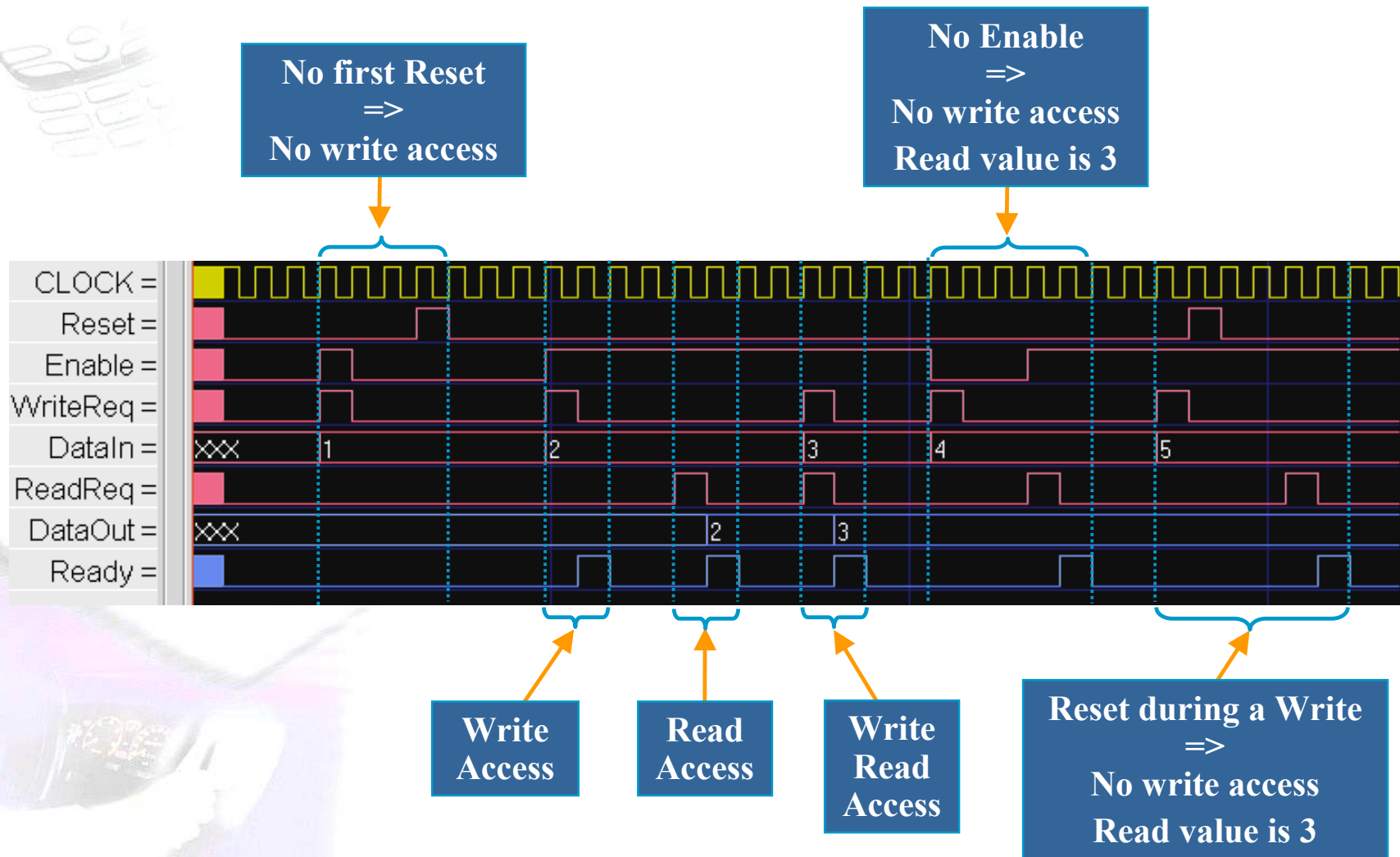
## Simple RAM Specifications

- ✚ The **Ready** signal is sent at the end of each access
- ✚ Each read or write is executed *one cycle* after the corresponding request
- ✚ These accesses are *suspended* when the **Enable** signal is absent
- ✚ Each access is immediately *interrupted* when the **Reset** signal is present
- ✚ The memory can only be accessed after a first occurrence of **Reset** (boot sequence)



# Simple RAM (3/7)

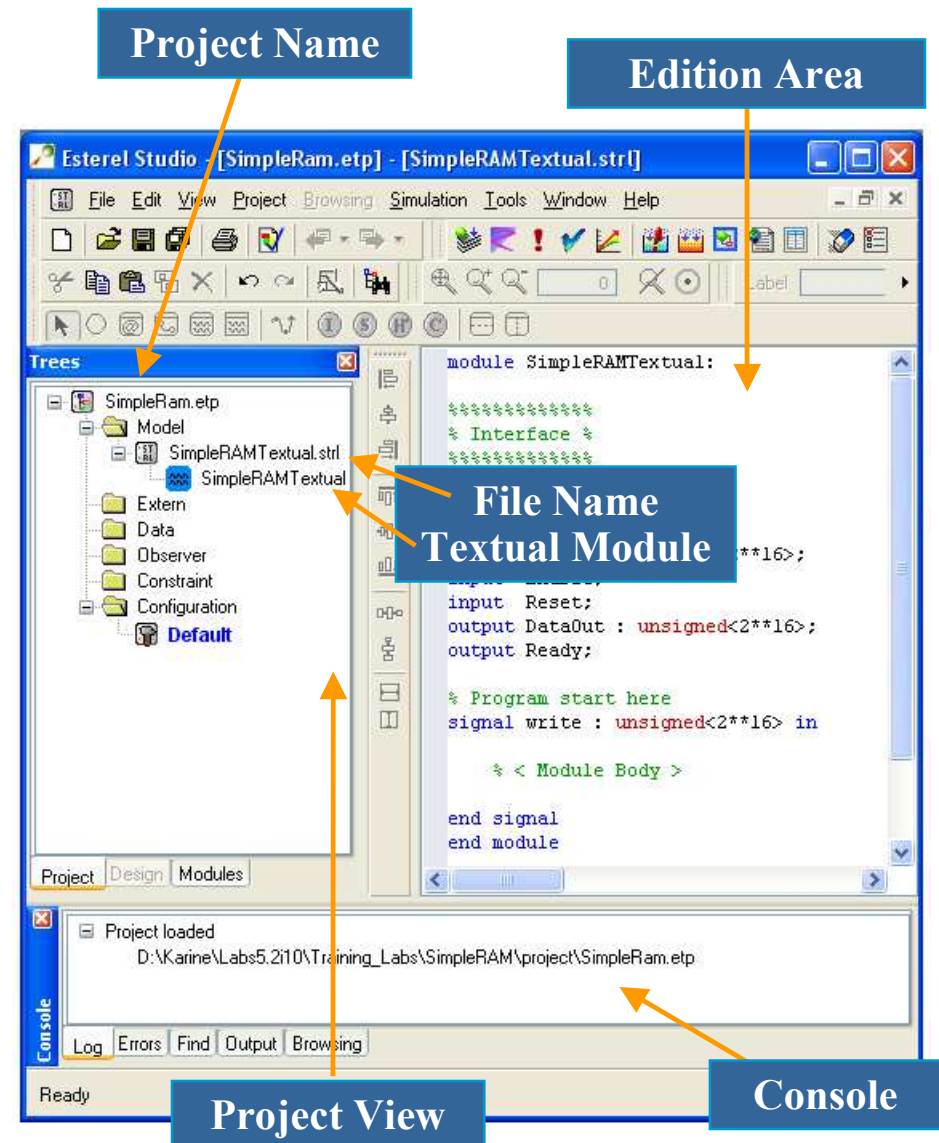
## Simple RAM Protocol



# Simple RAM (4/7)

## Exercise 1

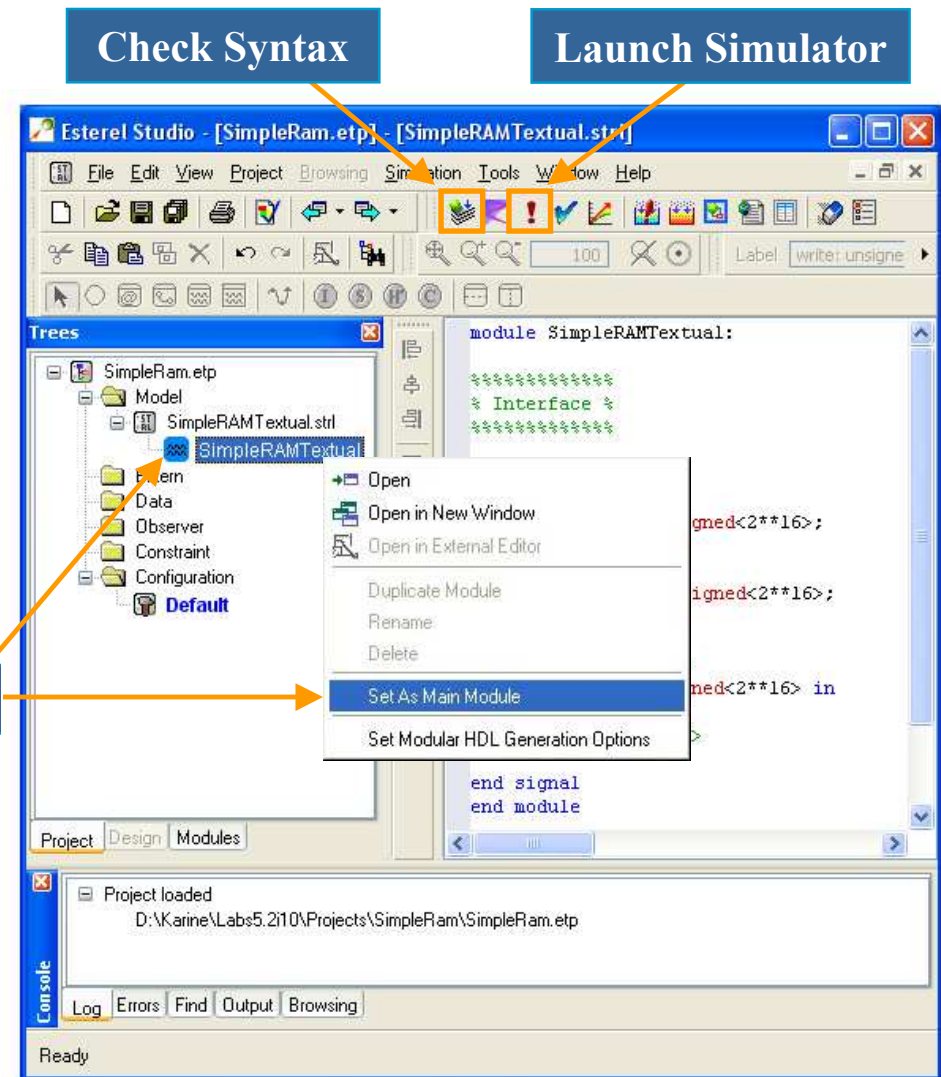
1. Launch  
Esterel Studio V5.2
2. Open the existing project  
*SimpleRAM.etp*  
using *File* → *Open Project*  
menu
3. Define the Simple RAM  
behavior using the  
predefined interface in the  
*SimpleRAMTextual* module





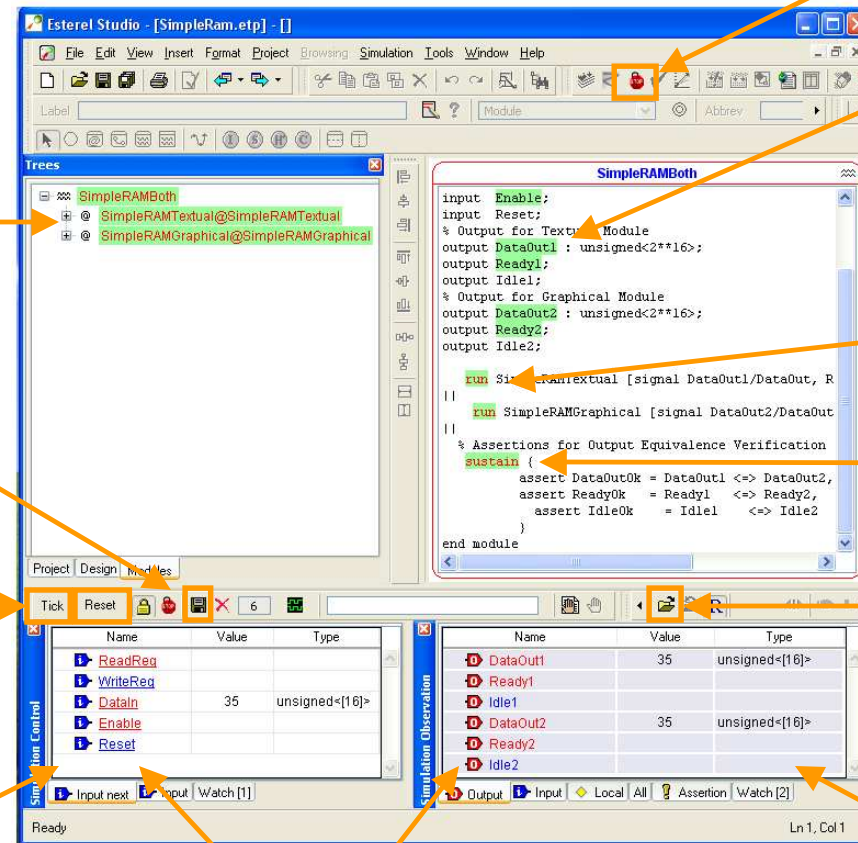
# Simple RAM (5/7)

4. Declare the *SimpleRAMTextual* module as **main module**
5. Check the **syntax** of your code
6. **Simulate** the behavior of your code



# Simple RAM (6/7)

## Esterel Studio graphical Simulator



Active module

Save the current  
scenario

Set a new instant  
or reset the  
simulation

Input Area

Red = Present  
Blue = Absent

Stop Simulator

present signals  
in green

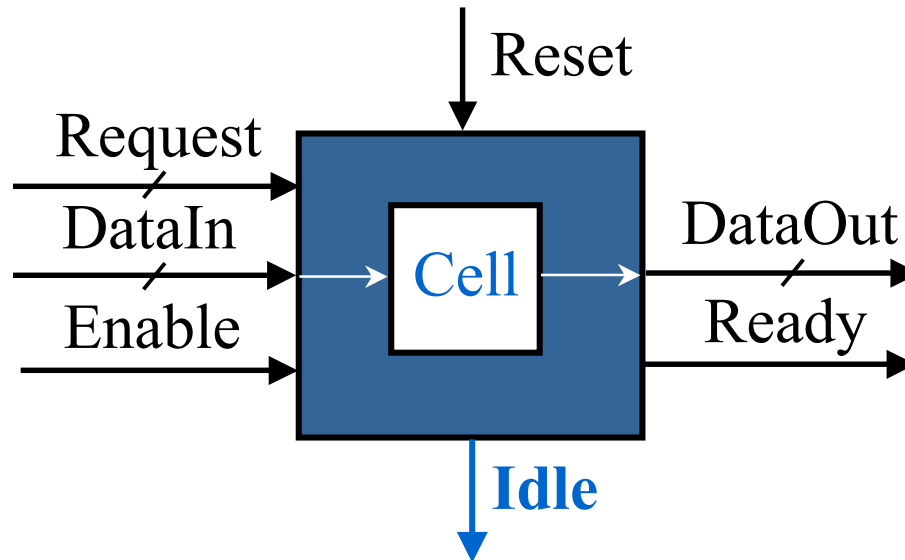
Active  
statements in  
green

Blocked Control  
in red

Play an existing  
scenario

Output Area

## Exercise 2



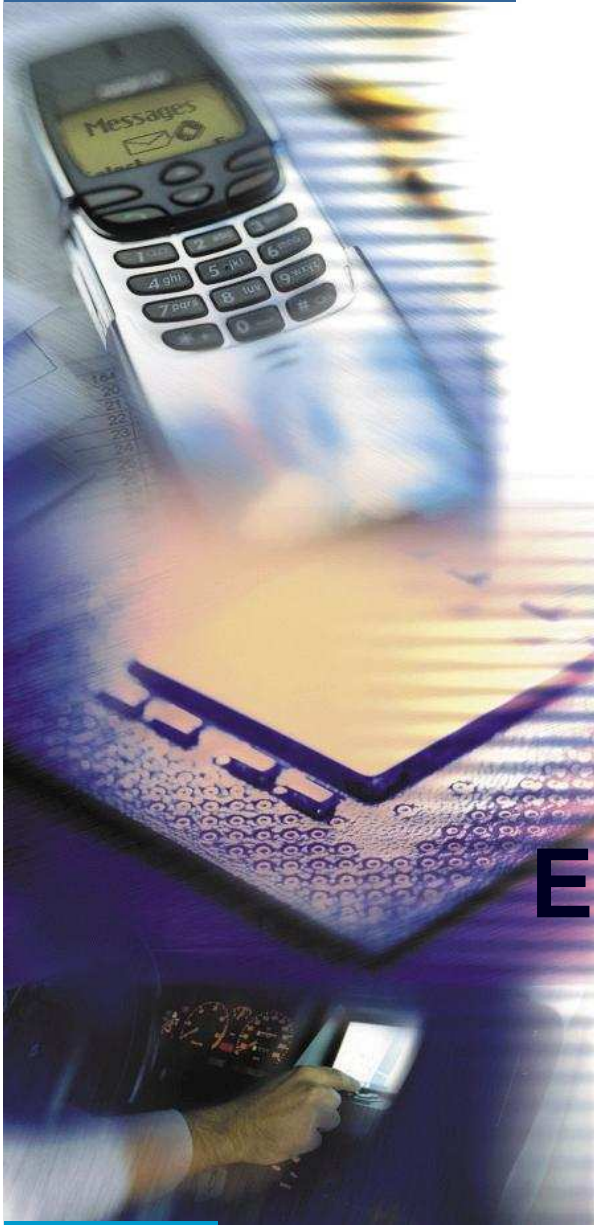
1. Modify the *SimpleRAMTextual* module to add the following feature:
  - + The *Idle* output signal must be maintained present as long as there is no ongoing access
2. Check the **syntax** of your code
3. **Simulate** its behavior



# Lab Solutions

## A Simple RAM

### Esterel Language Version



## Exercise 1: The Simple RAM without Idle

```
module SimpleRAMTextual:

  // Interface
  input ReadReq;
  input WriteReq;
  input DataIn : unsigned<[16]>;
  input Enable;
  input Reset;
  output DataOut : unsigned<[16]>;
  output Ready;

  signal write : unsigned<[16]> in    // local signal to store the data
    <module body>
  end signal

end module
```

## Correction - Simple RAM (2/5)

```
await Reset;
loop
  await not Reset;
  abort
  suspend
  // Write Access
  loop
    await immediate WriteReq;
    pause;
    emit { ?write <= ?DataIn,
          Ready }
  end loop
  ||
  // Read Access
  loop
    await immediate ReadReq;
    pause;
    emit { ?DataOut <= ?write,
          Ready }
  end loop
  when not Enable
  when Reset
end loop
```



## Exercise 2: The Simple RAM with Idle

```
module SimpleRAMTextual:

// Interface
input ReadReq;
input WriteReq;
input DataIn : unsigned<[16]> ;
input Enable;
input Reset;
output DataOut : unsigned<[16]> ;
output Ready;
output Idle;

signal write : unsigned<[16]> , IdleWrite, IdleRead in

    <module body>

end signal
end module
```

# Correction - Simple RAM (4/5)

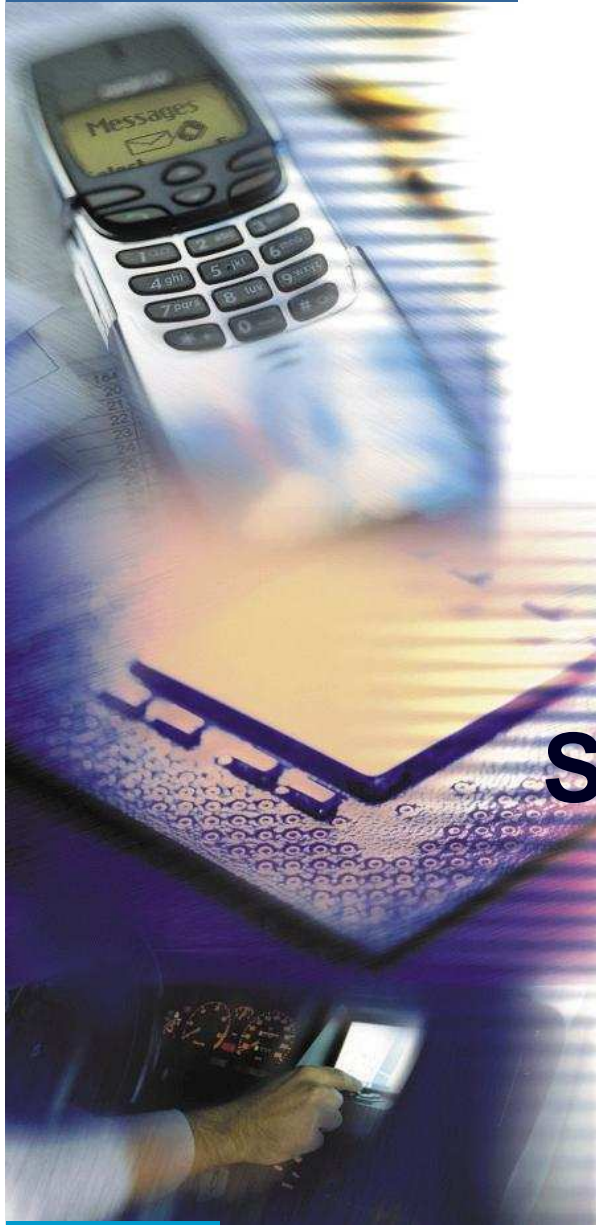
```
...  
// Write Access  
loop  
  abort  
    sustain IdleWrite  
  when immediate WriteReq;  
  pause;  
  emit { ?write <= ?DataIn,  
        Ready }  
end loop  
  
||  
// Read Access  
loop  
  abort  
    sustain IdleRead  
  when immediate ReadReq;  
  pause;  
  emit { ?DataOut <= ?write,  
        Ready }  
end loop  
...
```

# Correction - Simple RAM (5/5)

```
await Reset;  
loop  
  await not Reset;  
  abort  
    suspend  
      <Write Access>  
      ||  
      <Read Access>  
  when not Enable  
    ||  
    sustain Idle <= IdleWrite and IdleRead  
  
  when Reset  
end loop
```



# A Simple RAM Safe State Machine Version





## Exercise 3: The Graphical version

- ✚ The goal is now to create a graphical version of the previous Simple RAM including Idle signal management
- ✚ The graphical version must have the **same behavior** as the textual version

## Esterel Studio Features

- ✚ Intuitive Specification Interface
- ✚ Interactive Graphical Simulator
- ✚ Formal Behavior Verification
- ✚ Automatic Testbench Generator
- ✚ Code Generator – Optimizers
- ✚ Automated Document Generation

## ► Formal Specification & Design Capture

- ▼ Spots ambiguities at the specifications & design levels
- ▼ Hierarchical description of complex systems

## ► Includes Powerful FSM Semantics

- ▼ Parallelism
- ▼ Hierarchy
- ▼ Preemption
- ▼ Communication
- ▼ Priorities
- ▼ Determinism



The diagram illustrates a hierarchical state machine model. It features a main module labeled 'MAIN\_MODULE' which contains a state machine. This module is composed of several states and transitions. A sub-module, labeled 'Graphical Macrostate', is embedded within the main module. This sub-module contains its own state machine, including a state labeled 'Instance1@Module\_A'. The diagram is annotated with various labels and arrows pointing to specific components:

- multiple initial transitions**: Points to the initial state of the main module.
- Initial connector**: Points to the initial state of the main module.
- local signals**: Points to the 'Local1' signal in the main module and 'Local2' in the macrostate.
- graphical macrostate (hierarchy)**: Points to the 'Graphical Macrostate' sub-module.
- input (trigger)**: Points to the 'Input1 / Output1' transition in the main module.
- output (effect)**: Points to the 'Output1' output in the main module.
- state**: Points to a state in the main module.
- transition**: Points to a transition in the main module.
- priorities**: Points to the priority values '<1>' and '<2>' on transitions.
- multiple effects**: Points to the 'Output3, Local1' output in the main module.
- parallelism**: Points to the parallel structure of the macrostate sub-module.
- normal termination transition**: Points to the 'EndMacrostate' transition in the macrostate.
- run module (reusability)**: Points to the 'Instance1@Module\_A' sub-module.
- preemption**: Points to the 'StartModuleA' transition in the macrostate.
- terminal state**: Points to the final state of the macrostate.

# Esterel Studio Editor – The States

The screenshot shows the Esterel Studio Editor interface with several annotations:

- state**: Points to the 'Simple state' icon in the toolbar.
- initial connector**: Points to the 'Initial connector' icon in the toolbar.
- type of state**: Points to the dropdown menu showing state types: Simple state, Simple state, Run module, Graphical macrostate, and Textual macrostate.
- state name**: Points to the 'Label' field containing 'State1'.
- terminal state**: Points to the 'Terminal' checkbox in the Properties dialog.

The main workspace displays a state diagram with a state named 'State1' and an initial connector. The Properties dialog is open, showing the configuration for 'State1'.

**Properties Dialog:**

- Attributes tab selected.
- Name: State1
- Terminal: ☒ Process: Simple State
- OnInside Action: /
- Also execute OnInside when entering st: ☐
- OnEntry Action: /
- OnExit Action: /

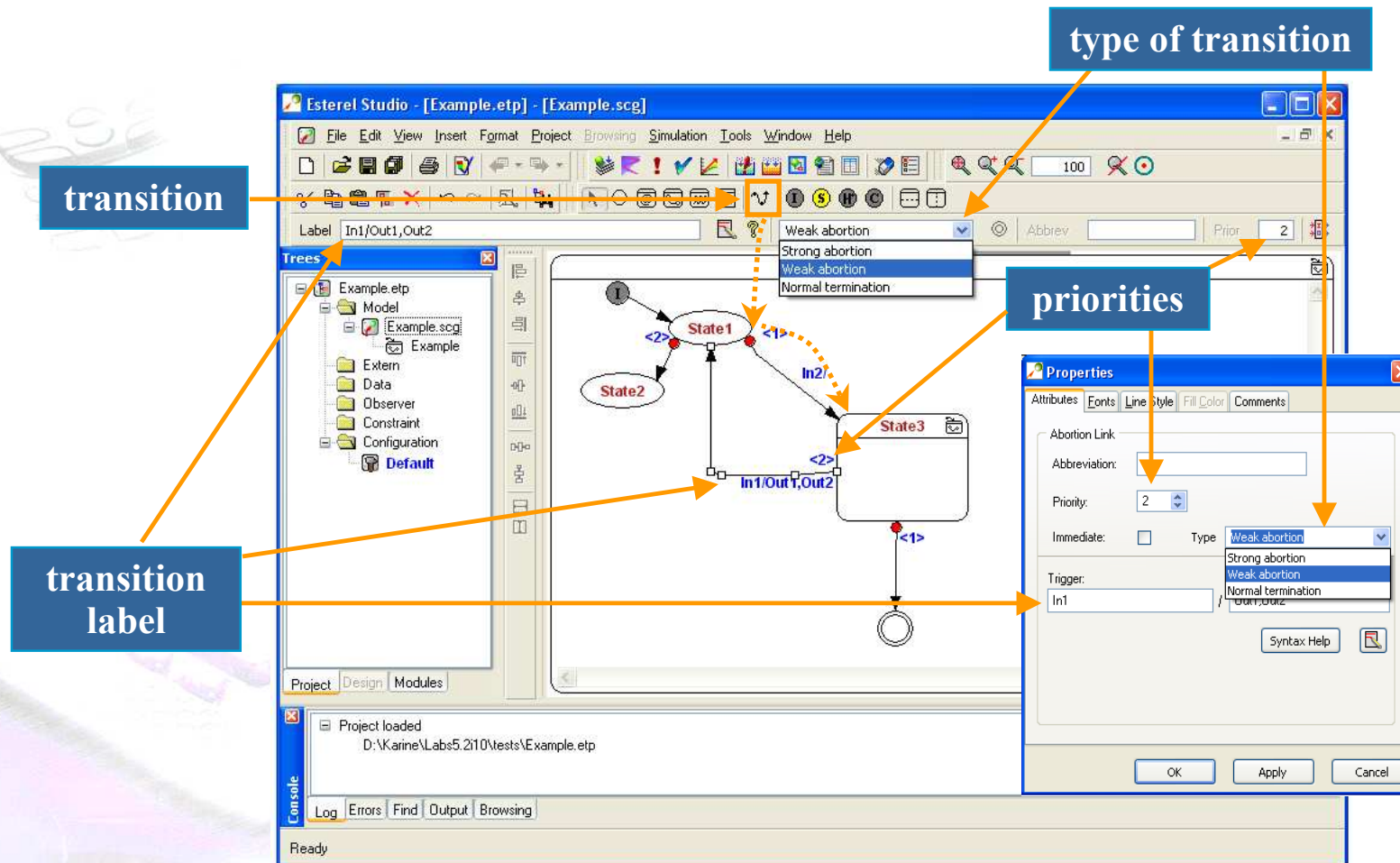
**Tree View:**

- Example.etp
- Model
- Example.scg
- Example
- Extern
- Data
- Observer
- Constraint
- Configuration
- Default

**Console:**

- Project loaded
- D:\Karine\Labs5.2110\tests\Example.etp

# Esterel Studio Editor – The Transitions



# Esterel Studio Editor – Architecture

Run  
Module

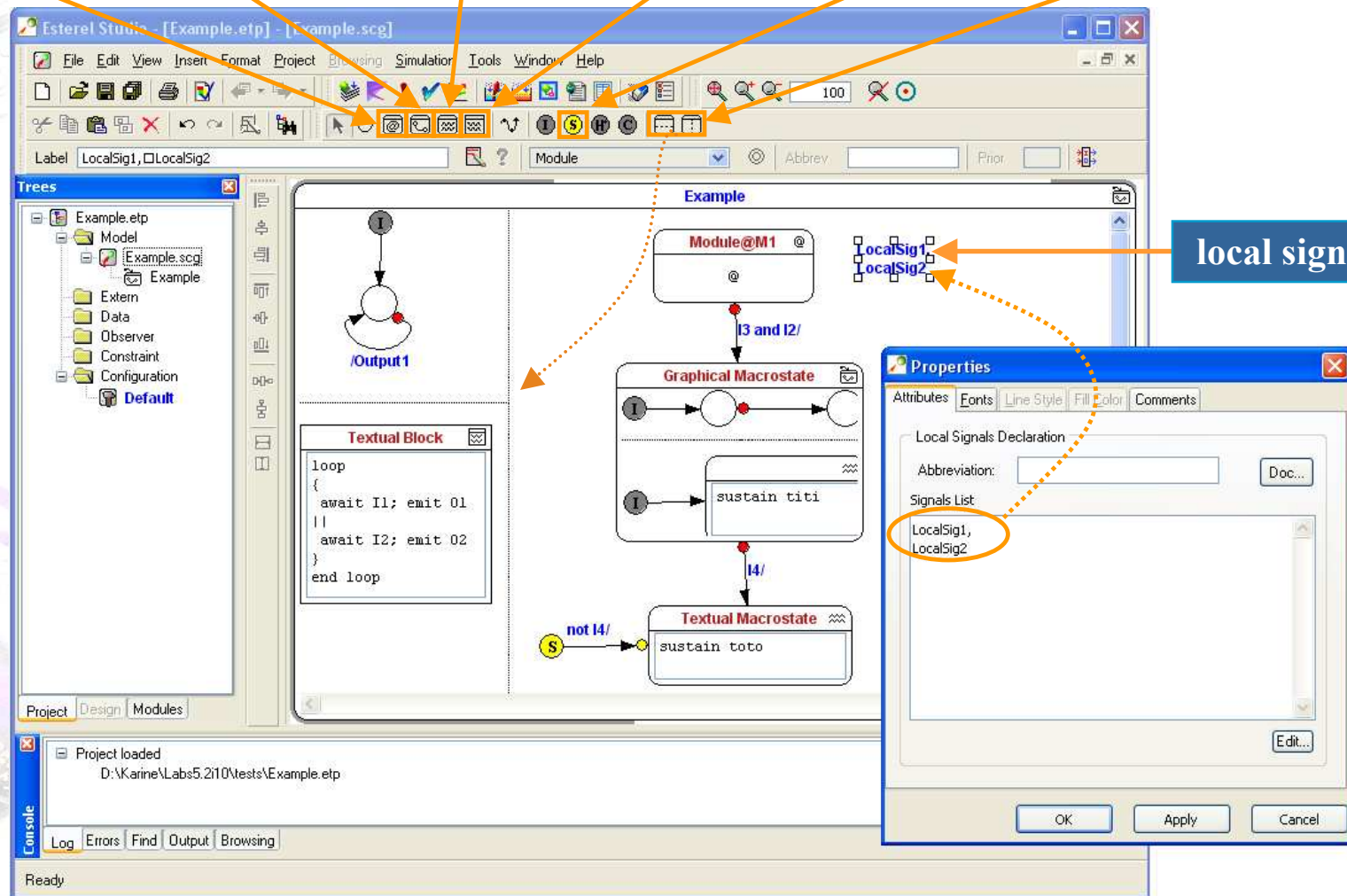
Graphical  
Macrostate

Textual  
Macrostate

Textual  
Block

Suspend  
Connector

Parallelism  
indicators



# Simple RAM (2/5)

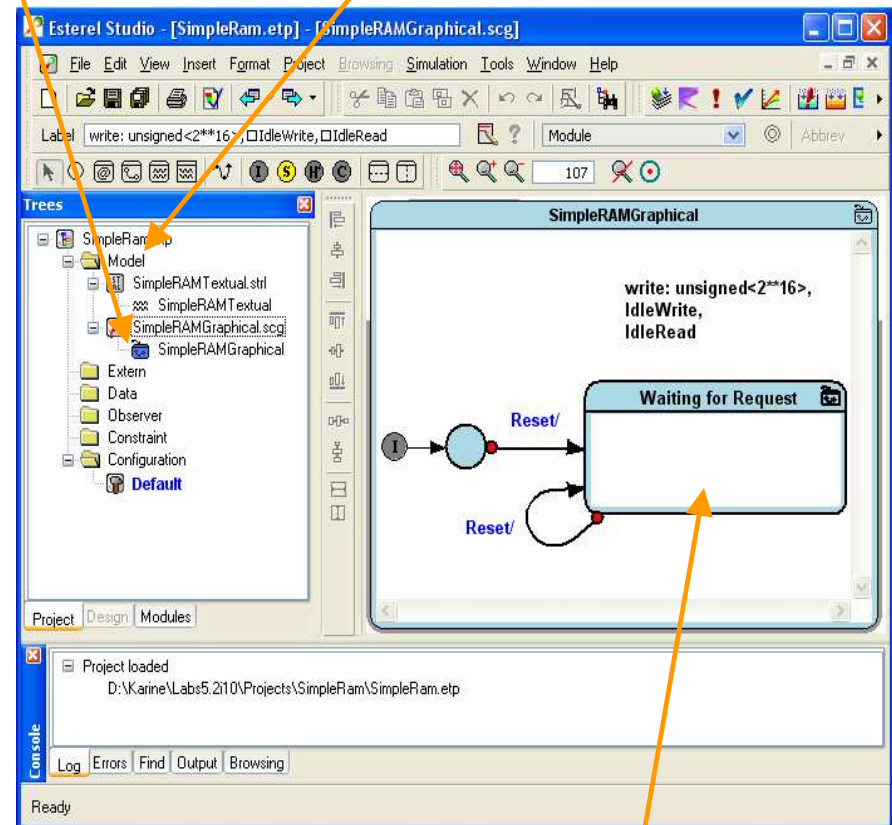


## Exercise 3

1. Right-Click on *Model* folder to insert the **existing file** named *SimpleRAMGraphical.scg*
2. Define the *SimpleRAMGraphical* module as **main module**
3. **Define** the Simple RAM Safe State Machine (SSM)
4. **Simulate** its behavior

New Main Module

Right-Click/  
Insert File

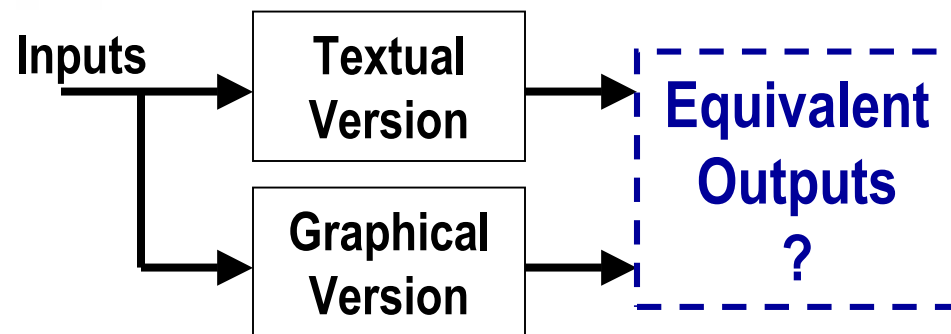


Determine the content of  
this graphical macrostate

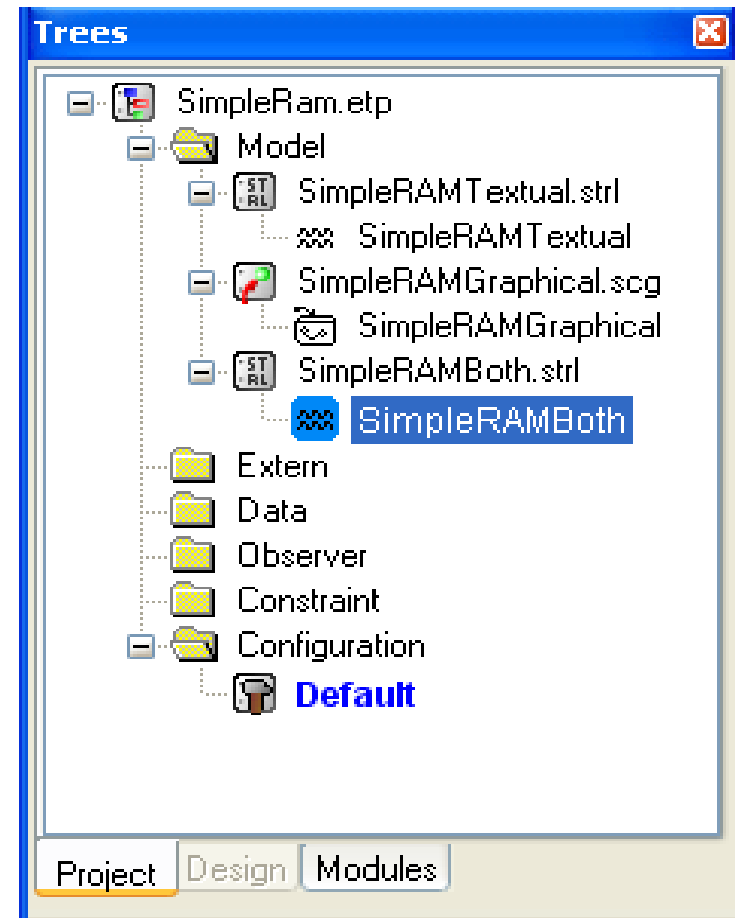


## Exercise 4

### Comparison between Textual and Graphical



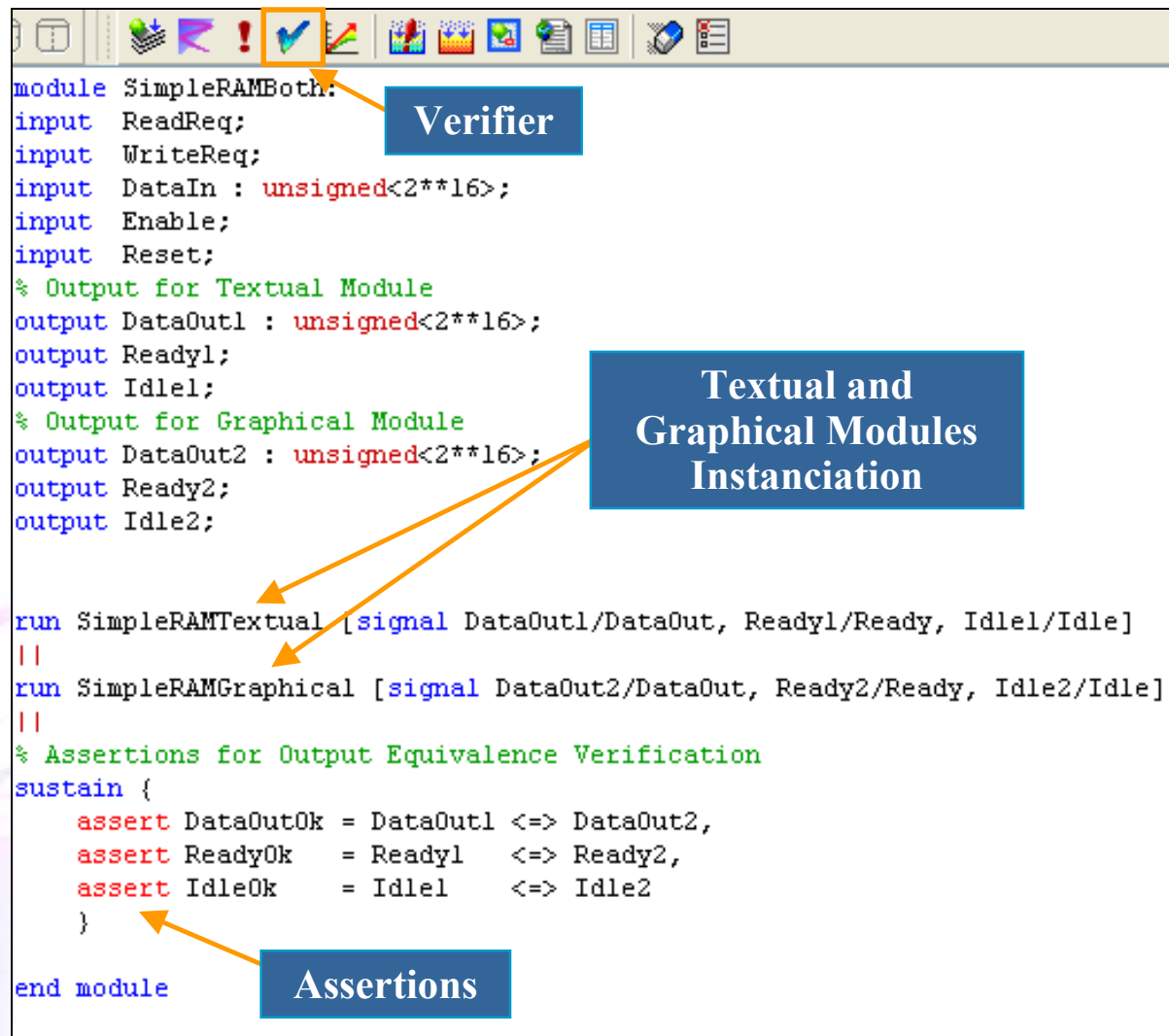
1. Right-Click on *Model* folder to insert the **existing file** named *SimpleRAMBoth.strl*
2. Define the *SimpleRAMBoth* module as **main module**





# Simple RAM (4/5)

3. **Verify** that all the predefined **assertions** are always true



```
module SimpleRAMBoth:
input  ReadReq;
input  WriteReq;
input  DataIn : unsigned<2**16>;
input  Enable;
input  Reset;
% Output for Textual Module
output DataOut1 : unsigned<2**16>;
output Ready1;
output Idle1;
% Output for Graphical Module
output DataOut2 : unsigned<2**16>;
output Ready2;
output Idle2;

run SimpleRAMTextual [signal DataOut1/DataOut, Ready1/Ready, Idle1/Idle]
||
run SimpleRAMGraphical [signal DataOut2/DataOut, Ready2/Ready, Idle2/Idle]
||
% Assertions for Output Equivalence Verification
sustain {
  assert DataOutOk = DataOut1 <=> DataOut2,
  assert ReadyOk  = Ready1   <=> Ready2,
  assert IdleOk   = Idle1    <=> Idle2
}

end module
```

**Verifier**

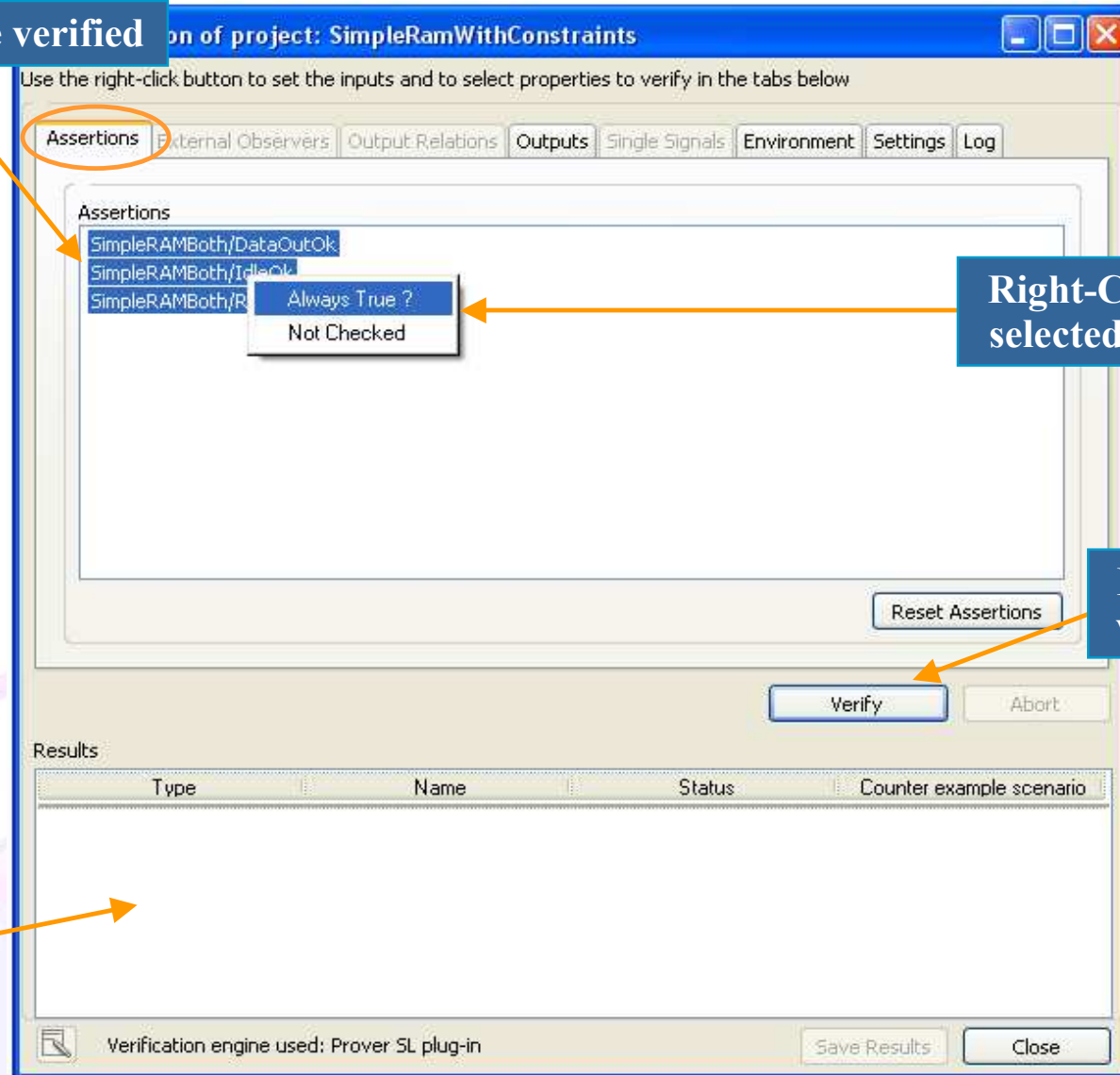
**Textual and Graphical Modules Instanciation**

**Assert all the time!**

**Assertions**

# Simple RAM (5/5)

Assertions to be verified



Right-Click on the  
selected assertions

Launch the  
verification

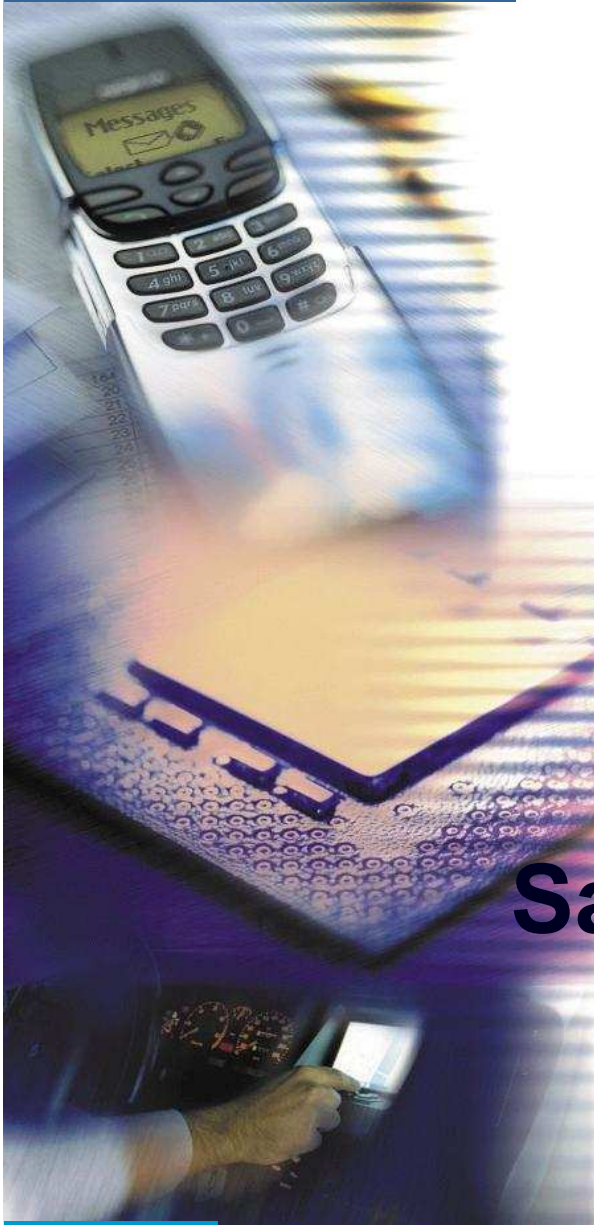
Result Area



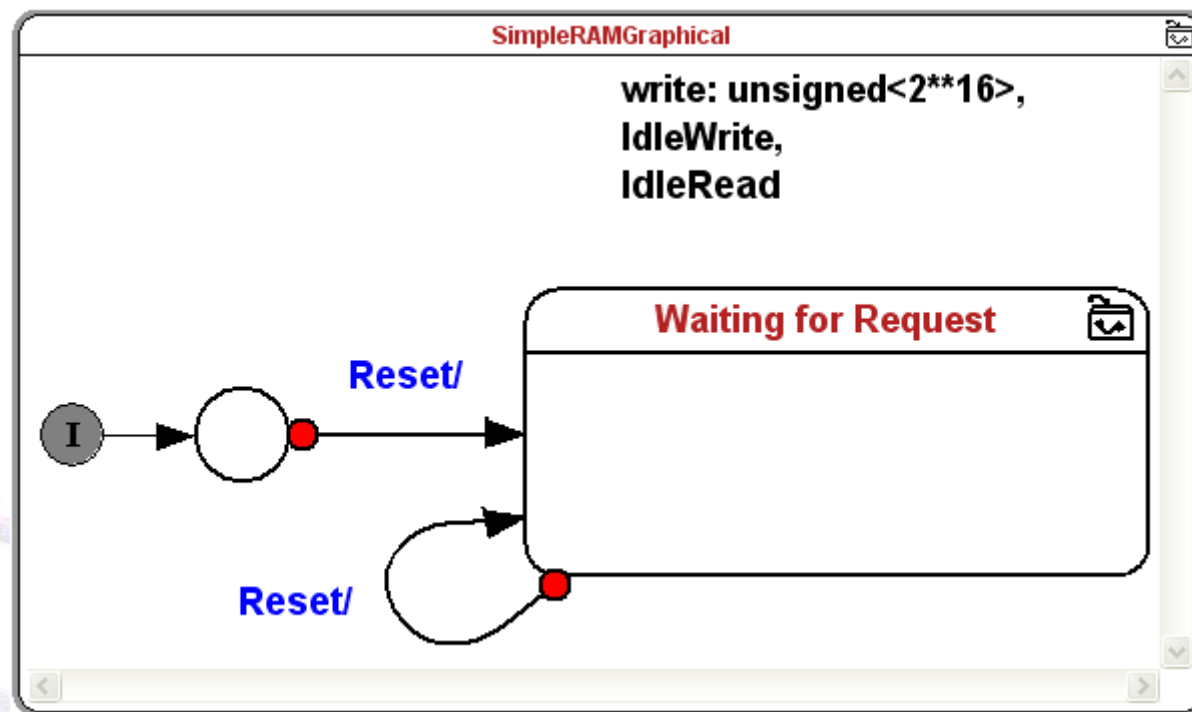
# Lab Solutions

## A Simple RAM

### Safe State Machine Version



## Exercise 3: The Graphical Simple RAM



# Correction - Simple RAM (2/2)

