

Reactive System Programming

Esterel (Part 1)

SF7 (EPU SI3) / E2 (Master STIC)

October 30, 2005

1 Introduction to the Esterel Studio Environment

Esterel Studio (ES, in what follows) is used to model, simulate, and verify electronic devices and *embedded systems*. The current version is ES v5.3. Esterel Studio is a collection of licensed modules. It can be used only in licensed sites. As an academic user you can get a license for a simpler version (visit [//www.esterel-technologies.com](http://www.esterel-technologies.com), download section). From there, you may download a compiler for ESTEREL v5_91 and related software, running under Linux.

Inputs to ES are either textual (ESTEREL code) or graphical (SYNCCHARTS). In this lab we consider the former and more precisely ESTEREL v7.3 (these programs must be adapted to run on your private installation).

Several ESTEREL files and documents are available at [//www.i3s.unice.fr/~andre/CAdoc](http://www.i3s.unice.fr/~andre/CAdoc) (direct access to this directory). Copy `td3St.zip`. `td3Stv5.zip` is an archive containing ESTEREL v5-91 code.

Remark 1 most screen captures have been made with ES-5.2. There may be slight difference with the ES-5.3 screens.

Remark 2 your windows environment variables INCLUDE, LIB, and PATH may have to be modified.

Launch ES (`C:\Program Files\Esterel Technologies\EsterelStudio\bin\estudio.exe`).

1.1 Starting a new project

1. Create a project from scratch:

- Either click on `File>New` or click on the `New File` button.
- Browse to select a location for the new project.
- Type the name of the project (Here, `ABRO`). A new file `ABRO.etp` is created.
- Copy file `ABRO.str1` into the project directory.
- Click `Save`.

2. Add ESTEREL files to your project:

- Right-click the `Model` folder in the Tree Pane (on the left side).

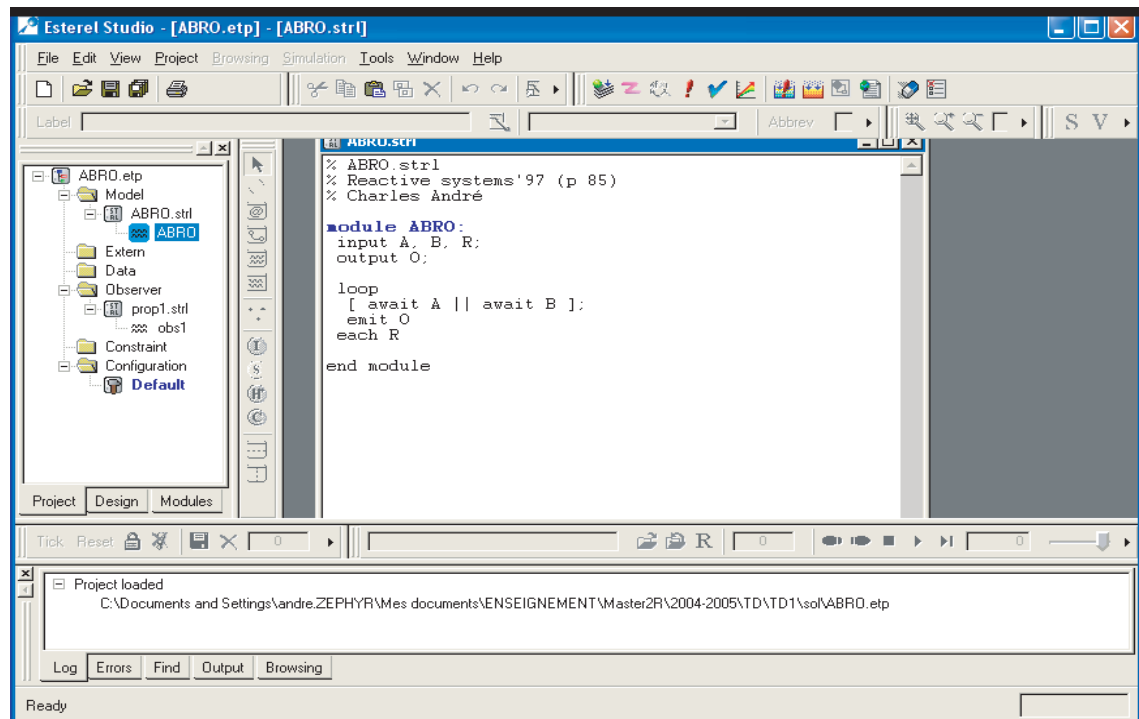


Figure 1: Esterel Studio Workspace.

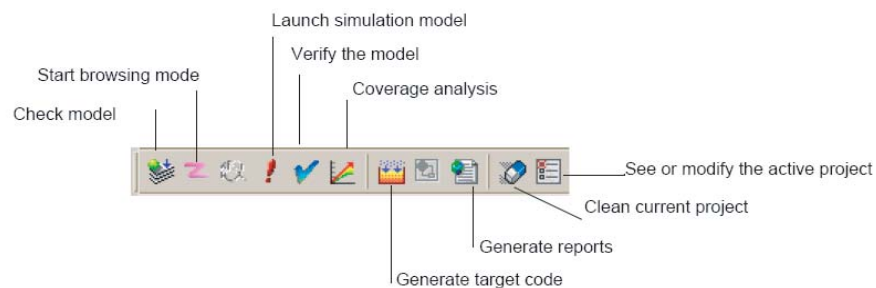


Figure 2: Project toolbar.

- Select Insert a file and type ABRO.strl.
- Save.

3. To edit a file, click on its icon in the Tree Pane.

1.2 Customizing

1. Click on Project>Active configuration.
2. Click the General tab settings.
3. Select EsterelV7 as the Project language. From now on, use ESTEREL v7 syntax only.
4. Fill in the Main Module entry. Here, set to ABRO.
5. In the Code Gen tab, check the Target Language. It should be C ANSI.
6. Close the dialog window (click OK).
7. Click on Save All.

Remark : Most commands are reachable from the menu, as explained above, and by clicking on icons (see Fig. 2 and the in-line Help).

1.3 Simulating Model Behavior

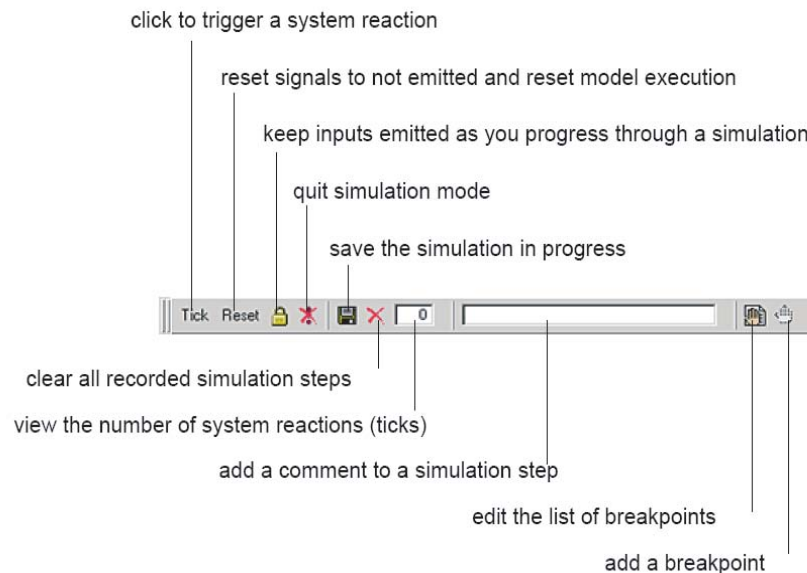


Figure 3: Esterel Studio Simulation Toolbar.

1.3.1 Interactive Simulation

1. Checking syntax:

- Click on `Project>Check model`.
- Read messages in the Console pane.
- Modify your program if needed.

2. Launching the simulation:

- Click the `Simulation` button (exclamation mark icon) or click `Project>Simulate`.
- The `Inputs` and the `Outputs` panels appear automatically.

3. Running the simulation

- In the `Inputs` panel all system inputs are listed. The default color is blue, which means that the signal is absent. Click on the signal name to make it present, it then turns to red.
- Click `Tick` in the simulation control toolbar.
- Observe the reaction: signals changing color in the `Outputs` panel; signals and control points highlighted in the source code pane.

4. Setting signal activation modes:

- Right-Click the signal name in the `Inputs` pane.
- Try the various modes.

5. Quit simulation mode: Click on the crossed exclamation mark icon.

1.4 Scenarios

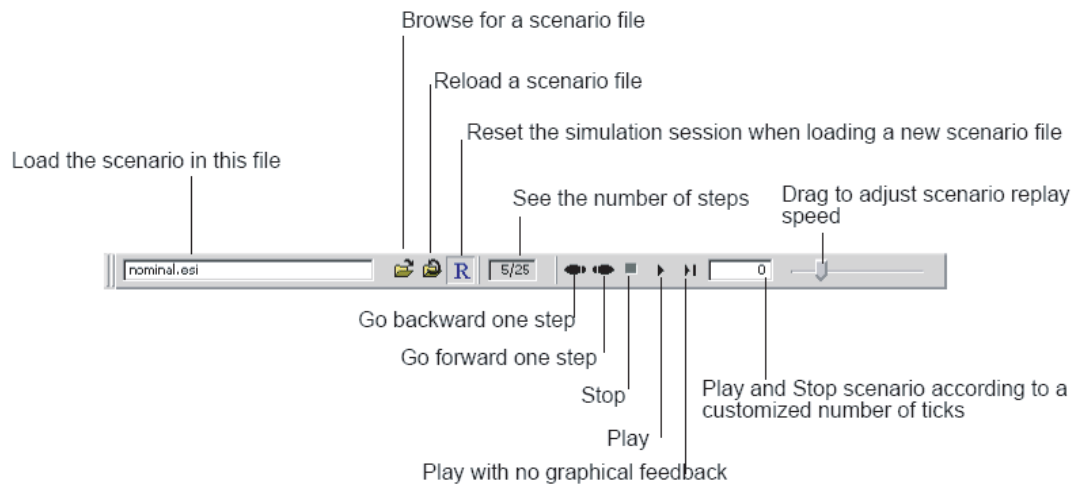


Figure 4: Simulation recorder toolbar.

While running a simulation you may record your scenario (Toolbar in Fig. 4).

1. Running a scenario:

- Apply a scenario that covers the main functionalities of your application.
- The initial state can be restored at any time by clicking the Reset icon.

2. Recording a scenario: before leaving your simulation

- Click the Disket icon in the simulation control toolbar.
- Give a name. Here, ABRO.esi
- Quit the simulation mode.

3. Replaying a scenario:

- Launch the simulation mode.
- In the Simulation control toolbar click the Open Folder icon.
- Select a recorded scenario.
- Now the scenario can be replayed step-by-step (Step icon) or automatically.

4. Displaying waveforms

- Click on the timewaves icon in the Simulation tool bar.
- Observe the waveforms.
- Note that waveforms are displayed using GTKware, a free software.

2 Variations on a simple example

Goal : understanding various forms of preemptions.

We consider an increasingly sophisticated “Clock”.

2.1 Basic Clock

The basic version receives a (periodic) input signal **Pulse**, and generates output signals **Tick** and **Tock**.

Caution : don't confuse **Tick** (a user's defined signal) with **tick** (the predefined ESTEREL signal).

```
// basicClock.str1
// never ending tick – tock
// Charles Andre
// October 2006

module basicClock :
  input Pulse ;
  output Tick , Tock ;

  loop
    await Pulse ; emit Tick ;
    await Pulse ; emit Tock
  end loop
end module
```

Question a: Compile and simulate this program. Observe the control points (highlighted keywords) in the animated source code.

Question b: Explain why **Pulse** has no effect at the very first instant. Modify the code to take account of a possible presence of **Pulse** at the first instant.

2.2 Preemption

The above system is not especially interesting and it does it for long! We add a signal **Hush** that stops the clock.

2.2.1 Strong abortion

```
// basicClock1.str1
// tick – tock + strong abort
// Charles Andre
// October 2006

module basicClock1 :
  input Pulse , Hush ;
  output Tick , Tock ;

  abort
    loop
      await Pulse ; emit Tick ;
      await Pulse ; emit Tock
    end loop
  when Hush
end module
```

Question a: Compile and simulate this program.

The same behavior can be obtained by re-using the `basicClock` module:

```
// basicClock2.str1
// tick - tock + strong abort + run
// Charles Andre
// October 2006
```

```
module basicClock2:
  input Pulse, Hush;
  output Tick, Tock;

  abort
    run basicClock
  when Hush
end module
```

Question b: Compare the behavior of the two programs: run them concurrently and add an *observer* module that emits an *Alarm* signal when the outputs differ (Fig. 5).

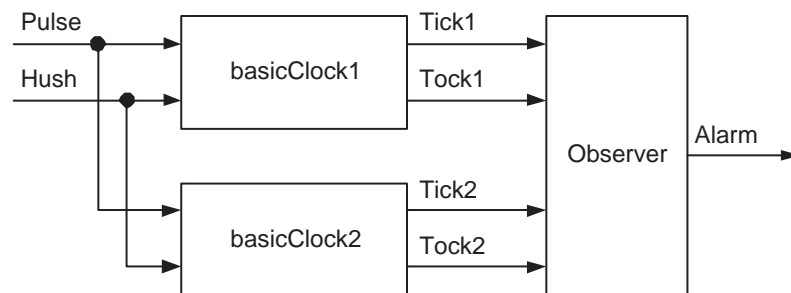


Figure 5: Comparing two programs.

2.2.2 Weak abort

Now, when *Hush* occurs, we let the aborted part complete its current reaction, for instance react to a *Pulse*.

Question a: Compile and simulate `basicClock3`.

```
// basicClock3.str1
// tick - tock + weak abort + run
// Charles Andre
// October 2006
```

```
module basicClock3:
  input Pulse, Hush;
  output Tick, Tock;

  weak abort
    run basicClock
  when Hush
end module
```

Question b: The weak abort construct is derived from a trap construct. `basicClock3b` is the trap-based version of the clock. Visualize signals and *exceptions* for a scenario. As you did in Section 2.2.1, show that `basicClock3` and `basicClock3b` have the same behavior.

```
// basicClock3b.str1
// tick - tock + trap + run
// Charles Andre
// October 2006
```

```
module basicClock3b :
  input Pulse , Hush;
  output Tick , Tock;

  trap Purr in
    run basicClock
    ||
    await Hush;
    exit Purr
  end trap
end module
```

2.2.3 Model Checking

Observing the same behavior of two modules on a given scenario is not enough to prove their equivalence.

ES supports *formal verification* tools. Use formal verification to:

- Detect corner-case bugs in the design which are hard to find using standard simulation-based verification.
- Prove rigorously that the design fulfills its main and critical functional requirements.

We apply the latter to prove that `basicClock3` and `basicClock3b` are equivalent.

1. Create a new project `equiv`
2. Load files `basicClock.str1`, `basicClock3.str1`, `basicClock3b.str1`, `eqTest.str1` (see Fig. 6).
3. Right-click the Observer folder in the Tree Pane.
4. Select New>File.
5. Type the observer module (`eqObserver.str1`). This module takes `Tick1`, `Tock1`, `Tick2`, `Tock2` as input signals, and `eqViolation` as an output signal. It emits `eqViolation` whenever `Tick1 ≠ Tick2` or `Tock1 ≠ Tock2`.
6. Save all.
7. Click Project>Active configuration. Select the Verification tab.
8. Select the Use built-in verifier (BDD) and the Global evaluation option. Click on the Apply button.
9. Select the Environment tab. Double-click on the observer file name to enable it. Click on the Apply button.

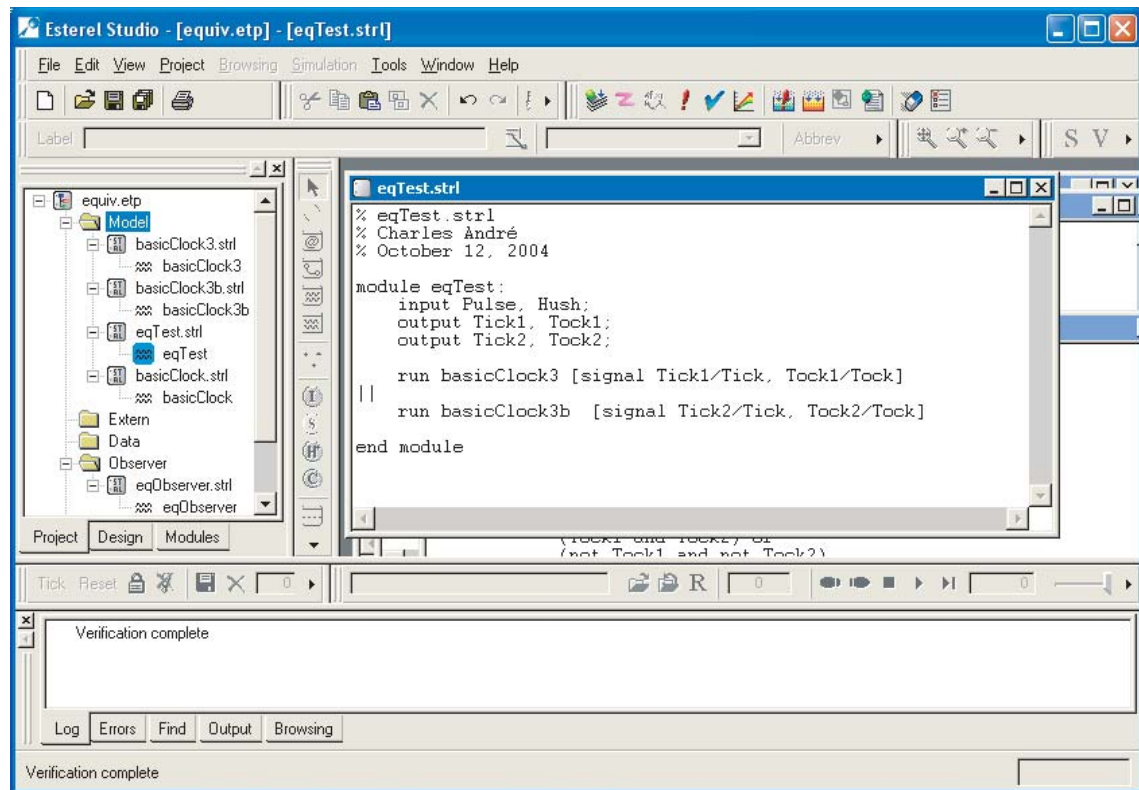


Figure 6: Workspace for the verification.

10. Click OK to close.
11. Click Project>Verify. After a compilation phase, a new window appears. Click the External Observers tab.
12. Right-click on the signal name (eqViolation). Choose Possibly present?.
13. Click on the Verify button.
14. Check the diagnostic in the Result pane. If the property holds then the status of eqViolation is AlwaysAbsent. If the property is violated then a counter-example scenario is built; the associated file is given in the Counter-example scenario column.
15. Click the Close button.

See Fig. 7.

2.3 Displaying time

A clock should give time! In this section we add a display functionality.

The new clock displays Hours, Minutes, and Seconds (output signals Hd, Md, Sd, the type of which is integer). The pure input signal C is provided by an external generator whose frequency is 1 Hz.

The clock module consists of 3 instances of a modulo counter (See Fig. 8).

Question a: Write and test a module ModCounter for a modulo(N) counter.

Question b: Assemble modules in a new clock module.

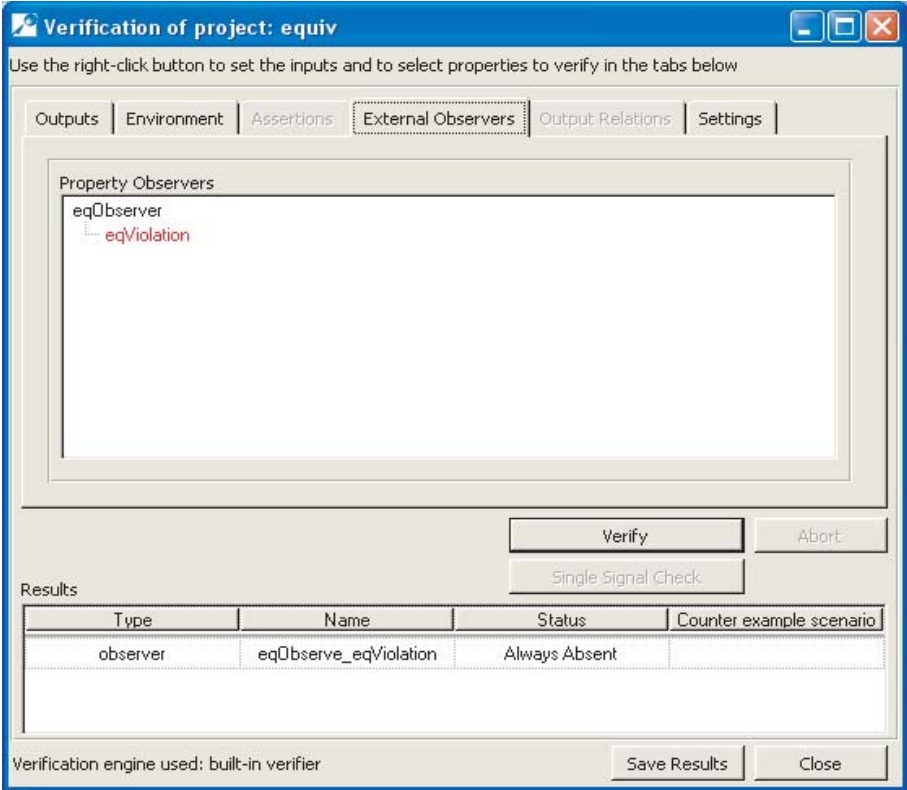


Figure 7: Verification window.

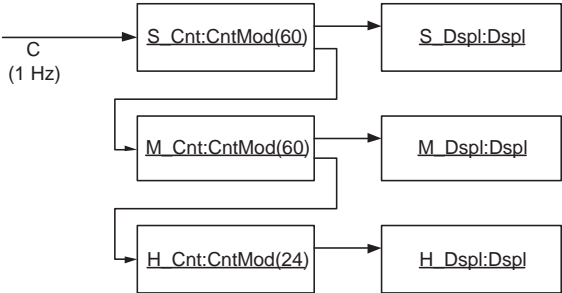


Figure 8: Structure of the new clock module.