Date: 4 July 2012

TOOL VERIFICATION CASES AND PROCEDURES

Title:

Reference: POLYCHRONY-VERIFIER/TVCP version 1

Status:

TABLE OF CONTENTS

1 Introduction

1.1 Purpose of the document

This document presents the tests cases to be performed for the qualification of Polarsys Polychrony Verifier component as described in the Generic Developer Tool Qualification Plan of Polychrony Verifier document ([R1]).

These tests cases verify the Generic Developer Tool Operational Requirements of Polychrony Verifier document ([R2]).

1.2 Applicable documents

Mark	Designation	Issuer	Reference	Issue
[A1]	Software consideration in airborne systems and equipment specification	RTCA	DO-178	C
[A2]	Software Tool Qualification Considerations	RTCA	DO-330	

1.3 Reference documents

Mark	Designation	Reference	Issue
[R1]	Generic Developer Tool Qualification Plan of Polychrony Verifier	OPEES DELIVERABLE D3.3.3 §7.2.2.1	Latest
[R2]	Generic Developer Tool Operational Requirements of Polychrony Verifier	OPEES DELIVERABLE D3.3.3 §7.2.3.1	Latest

1.4 Acronyms

TOR	Tool Operational Requirements
TQP	Tool Qualification Plan
TVR	Tool Verification Results

2 Polychrony tests overview

2.1 Verification kit

The Polychrony Verifier verification (Qualification?) kit contains the following files :

- A set of test directories (Test_polychrony_001_errSyntax, Test_polychrony_003_errDecl, Test_polychrony_005_errType, Test_polychrony_007_errForm, Test_polychrony_009_clockConstraints, Test_polychrony_011_cycles, Test_Polychrony_020_NoError, Test_polychrony_100_warnings), each one containing one or several signal file.
- A Reference directory, containing the same directories as listed above with, in addition, the directories and files (log files, c files and .h files) resulting from the C generation. These files are given for comparison purpose.

2.2 Test procedure

The test procedure consists in opening the downloaded Polychrony SSME platform or the AADL platform (that includes the SSME platform), then in importing the given Test_polychrony_Verifier project within the selected workspace. The project is composed of several directories containing Signal files to be tested.

To have the complete informations on the compiling, it is necessary to set some options: right click on a .SIG file, then select Polychrony/Misc options, then select “warnings”.

The test consists in trying to produce C code for each Signal program (right click on .SIG file, then select Polychrony/General Compilation/General Compilation; depending on the error, a table will open or not: if so, on “Code Generation” tab, select “C generation” and on “SIGNAL forms”, select “Signal LIS generation”).

Finally, generated files have to be compared with the given reference ones and C files generation status have to be checked.

Nota : In some cases, table for compilation option selection does not open because an error is raised during preliminary checks.

2.3 The Polychrony log files

Polychrony generates several log files. We recall that the compiling of a Signal program (FOO.SIG) consists in the following steps:

- The analyse of the syntax of the program (parsing). The syntax errors are returned to the user in the Eclipse console and in the file FOO.SIG_SYNTAX_ERRORS_LOG.

- The analyse of the form of the program (declaration of the objects (Signal, models,...). The errors are returned to the user in the file FOO_LIS.SIG (annotations on the original program).

- The analyse of the types of the program. The errors are returned to the user in the file FOO_LIS.SIG (annotations on the original program).

At this step the Signal program is represented by a DCG (Data Control Graph) that represents the synchronisations and the dependencies of the objects of the program after the instantiation of the referenced models. The compiling is then the application of a set of functionalities on the DCG that is modified. The applied functionalities depends on the objectives of the compiling (code generation with several techniques, export to other tools, analyse of the synchronisations,...). After each functionality, the DCG can be exported to the user as a new Signal program. The generated file depends on the level of the DCG:

  - FOO_BASIC_TRA.SIG (basic level) is the external representation of the DCG in which all the synchronisations of the program are explicited but not solved.

  - FOO_POLY_TRA.SIG (polychronous level) is the external representation of the DCG after the clock calculus (analyses of the synchronisations). The clocks are event signals (a boolean always true when it is present). The result of the clock calculus is a hierarchy of trees of clocks. When the hierarchy is not reduced to one tree, the program contains several master clocks (it is said polychronous).

  - FOO_ENDO_TRA.SIG (endochronous level) is the external representation of the program when the hierarchy of clocks is reduced to one master clock: when there are several master clocks in the Polychronous level DCG a parametrization of the clock system is used to produce only one tree of clocks, in this case the interface of the program is modified by adding of input signals.

  - FOO_BOOL_TRA.SIG (boolean level) is the external representation of the program in which the clocks (event in the previous levels) are represented by boolean signals (when the program is endochronous, it is possible to represent the absence of the signals by the “false” boolean value).

  - FOO_SCH_TRA.SIG (scheduled level) is the external representation of the program in which scheduling information are explicited (the reinforcing induced by the sequentialisation). The C/C++/Java code generators take this level of the DCG for the production of the code.

  - FOO_FLAT_TRA.SIG (flat level) is the external representation of the program in which the boolean hierarchy (boolean level DCG) is flattened, the hierarchy of clocks is reduced to one level. This level is used, for example, to export the Signal program to the Syndex Tool.

2.4 Traceability and expected results overview

The following table gives the traceability between test cases and upward requirements, as well as the expected generation status.

Test case	C files generation	Log file	Requirements	Comment
Test_polychrony_001_errSyntax / Test_polychrony_001_1	FAIL	Test_polychrony_001_1.SIG_SYNTAX_ERRORS_LOG	TOR_ Polychrony Verifier_Rq#001-1 TOR_ Polychrony Verifier_Rq#002-1 TOR_ Polychrony Verifier_Rq#020-1	Missing argument on a binary operator
Test_polychrony_001_errSyntax / Test_polychrony_001_2	FAIL	Test_polychrony_001_2.SIG_SYNTAX_ERRORS_LOG	TOR_ Polychrony Verifier_Rq#001-1 TOR_ Polychrony Verifier_Rq#002-1 TOR_ Polychrony Verifier_Rq#020-1	Missing ending ;
Test_polychrony_003_errDecl / Test_polychrony_003_1	FAIL	Test_polychrony_003_1_LIS.SIG	TOR_ Polychrony Verifier_Rq#003-1 TOR_ Polychrony Verifier_Rq#004-1 TOR_ Polychrony Verifier_Rq#020-1	Signal not declared
Test_polychrony_003_errDecl / Test_polychrony_003_2	FAIL	Test_polychrony_003_2_LIS.SIG	TOR_ Polychrony Verifier_Rq#003-1 TOR_ Polychrony Verifier_Rq#004-1 TOR_ Polychrony Verifier_Rq#020-1	Signal not declared (declared in a sub-process)
Test_polychrony_003_errDecl / Test_polychrony_003_3	FAIL	Test_polychrony_003_3_LIS.SIG	TOR_ Polychrony Verifier_Rq#003-1 TOR_ Polychrony Verifier_Rq#004-1 TOR_ Polychrony Verifier_Rq#020-1	Process and library not declared
Test_polychrony_003_errDecl / Test_polychrony_003_4	FAIL	Test_polychrony_003_4_LIS.SIG	TOR_ Polychrony Verifier_Rq#003-1 TOR_ Polychrony Verifier_Rq#004-1 TOR_ Polychrony Verifier_Rq#020-1	Signal double declaration
Test_polychrony_003_errDecl / Test_polychrony_003_5	FAIL	Test_polychrony_003_5_LIS.SIG	TOR_ Polychrony Verifier_Rq#003-1 TOR_ Polychrony Verifier_Rq#004-1 TOR_ Polychrony Verifier_Rq#020-1	Parameter not declared
Test_polychrony_003_errDecl / Test_polychrony_003_6	FAIL	Test_polychrony_003_6_LIS.SIG	TOR_ Polychrony Verifier_Rq#003-1 TOR_ Polychrony Verifier_Rq#004-1 TOR_ Polychrony Verifier_Rq#020-1	Type not declared
Test_polychrony_003_errDecl / Test_polychrony_003_7	FAIL	Test_polychrony_003_7_LIS.SIG	TOR_ Polychrony Verifier_Rq#003-1 TOR_ Polychrony Verifier_Rq#004-1 TOR_ Polychrony Verifier_Rq#020-1	Signal declaration in module instead of process
Test_polychrony_005_errType / Test_polychrony_005_1	FAIL	Test_polychrony_005_1_LIS.SIG	TOR_ Polychrony Verifier_Rq#005-1 TOR_ Polychrony Verifier_Rq#006-1 TOR_ Polychrony Verifier_Rq#020-1	disagreement between types of operands (string = integer + integer)
Test_polychrony_005_errType / Test_polychrony_005_2	FAIL	Test_polychrony_005_2_LIS.SIG	TOR_ Polychrony Verifier_Rq#005-1 TOR_ Polychrony Verifier_Rq#006-1 TOR_ Polychrony Verifier_Rq#020-1	Signal unexpected type (integers used with operand @)
Test_polychrony_005_errType / Test_polychrony_005_3	FAIL	Test_polychrony_005_3_LIS.SIG	TOR_ Polychrony Verifier_Rq#005-1 TOR_ Polychrony Verifier_Rq#006-1 TOR_ Polychrony Verifier_Rq#020-1	Signal unexpected type (integers used with operand |+)
Test_polychrony_005_errType / Test_polychrony_005_4	FAIL	Test_polychrony_005_4_LIS.SIG	TOR_ Polychrony Verifier_Rq#005-1 TOR_ Polychrony Verifier_Rq#006-1 TOR_ Polychrony Verifier_Rq#020-1	Parameter Unexpected type
Test_polychrony_005_errType / Test_polychrony_005_5	FAIL	Test_polychrony_005_5_LIS.SIG	TOR_ Polychrony Verifier_Rq#005-1 TOR_ Polychrony Verifier_Rq#006-1 TOR_ Polychrony Verifier_Rq#020-1	Signal unexpected type (integers computed from boolean operands)
Test_polychrony_005_errType / Test_polychrony_005_6	FAIL	Test_polychrony_005_6_LIS.SIG	TOR_ Polychrony Verifier_Rq#005-1 TOR_ Polychrony Verifier_Rq#006-1 TOR_ Polychrony Verifier_Rq#020-1	Signal used instead of clock
Test_polychrony_007_errForm / Test_polychrony_007_1	FAIL	Test_polychrony_007_1_LIS.SIG	TOR_ Polychrony Verifier_Rq#007-1 TOR_ Polychrony Verifier_Rq#008-1 TOR_ Polychrony Verifier_Rq#020-1	Signal multiple definition
Test_polychrony_007_e	FAIL	Test_polychrony_007_2_LIS.SIG	TOR_ Polychrony Verifier_Rq#007-1	Signal defined using

Test case	C files generation	Log file	Requirements	Comment
rrForm / Test_polychrony_007_2			TOR_ Polychrony Verifier_Rq#008-1 TOR_ Polychrony Verifier_Rq#020-1	partial definitions
Test_polychrony_007_errForm / Test_polychrony_007_3	FAIL	Test_polychrony_007_3_LIS.SIG	TOR_ Polychrony Verifier_Rq#007-1 TOR_ Polychrony Verifier_Rq#008-1 TOR_ Polychrony Verifier_Rq#020-1	Error on definition of a state variable
Test_polychrony_007_errForm / Test_polychrony_007_4	FAIL	Test_polychrony_007_4_LIS.SIG	TOR_ Polychrony Verifier_Rq#007-1 TOR_ Polychrony Verifier_Rq#008-1 TOR_ Polychrony Verifier_Rq#020-1	Not allowed use of a state variable
Test_polychrony_007_errForm / Test_polychrony_007_5	FAIL	Test_polychrony_007_5_LIS.SIG	TOR_ Polychrony Verifier_Rq#007-1 TOR_ Polychrony Verifier_Rq#008-1 TOR_ Polychrony Verifier_Rq#020-1	Non external action definition
Test_polychrony_007_errForm / Test_polychrony_007_6	FAIL	Test_polychrony_007_6_LIS.SIG	TOR_ Polychrony Verifier_Rq#007-1 TOR_ Polychrony Verifier_Rq#008-1 TOR_ Polychrony Verifier_Rq#020-1	Disagreement between function model and interface
Test_polychrony_007_errForm / Test_polychrony_007_7	FAIL	Test_polychrony_007_7_LIS.SIG	TOR_ Polychrony Verifier_Rq#007-1 TOR_ Polychrony Verifier_Rq#008-1 TOR_ Polychrony Verifier_Rq#020-1	Conflict between effective model and formal model
Test_polychrony_007_errForm / Test_polychrony_007_8	FAIL	Test_polychrony_007_8_LIS.SIG	TOR_ Polychrony Verifier_Rq#007-1 TOR_ Polychrony Verifier_Rq#008-1 TOR_ Polychrony Verifier_Rq#020-1	A main model with formal process is not yet implemented
Test_polychrony_009_clockConstraints / Test_polychrony_009_1	FAIL (*)	Test_polychrony_009_1_BOOL_TRA.SIG	TOR_ Polychrony Verifier_Rq#009-1 TOR_ Polychrony Verifier_Rq#010-1 TOR_ Polychrony Verifier_Rq#020-1	Clock Constraints. Log message is WARNING ! (compilation can be forced)
Test_polychrony_009_clockConstraints / Test_polychrony_009_2	FAIL (*)	Test_polychrony_009_2_BOOL_TRA.SIG	TOR_ Polychrony Verifier_Rq#009-1 TOR_ Polychrony Verifier_Rq#010-1 TOR_ Polychrony Verifier_Rq#020-1	Clock Constraints. Log message is WARNING ! (compilation can be forced)
Test_polychrony_009_clockConstraints / Test_polychrony_009_3	FAIL	Test_polychrony_009_3_BOOL_TRA.SIG	TOR_ Polychrony Verifier_Rq#009-1 TOR_ Polychrony Verifier_Rq#010-1 TOR_ Polychrony Verifier_Rq#020-1	Clock Constraints. Log message is WARNING ! (compilation can NOT be forced)
Test_polychrony_011_cycles / Test_polychrony_011_1	FAIL	Test_polychrony_011_1_BOOL_CYC.SIG	TOR_ Polychrony Verifier_Rq#011-1 TOR_ Polychrony Verifier_Rq#012-1 TOR_ Polychrony Verifier_Rq#020-1	Nor WARNING, nor ERROR in log file !
Test_polychrony_011_cycles / Test_polychrony_011_2	FAIL	Test_polychrony_011_2_BOOL_CYC.SIG	TOR_ Polychrony Verifier_Rq#011-1 TOR_ Polychrony Verifier_Rq#012-1 TOR_ Polychrony Verifier_Rq#020-1	Nor WARNING, nor ERROR in log file !
Test_polychrony_011_cycles / Test_polychrony_011_3	FAIL	Test_polychrony_011_3_BOOL_CYC.SIG	TOR_ Polychrony Verifier_Rq#011-1 TOR_ Polychrony Verifier_Rq#012-1 TOR_ Polychrony Verifier_Rq#020-1	Nor WARNING, nor ERROR in log file !
Test_polychrony_020_NoError / Test_polychrony_020_1	PASS	N/A	TOR_ Polychrony Verifier_Rq#020-1	Pb le fichier Test_polychrony_020_1_SCH_TRA est généré avec « WARNING: unused signals »
Test_polychrony_100_warnings / Test_polychrony_100_1	PASS	Test_polychrony_100_1_SCH_TRA.SIG	TOR_ Polychrony Verifier_Rq#011-1 TOR_ Polychrony Verifier_Rq#012-1 TOR_ Polychrony Verifier_Rq#020-1	Unused declared signal
Test_polychrony_100_warnings / Test_polychrony_100_2	PASS	Test_polychrony_100_2_SCH_TRA.SIG	TOR_ Polychrony Verifier_Rq#011-1 TOR_ Polychrony Verifier_Rq#012-1 TOR_ Polychrony Verifier_Rq#020-1	Not initialized delayed signal
Test_polychrony_100_warnings / Test_polychrony_100_3	PASS	Test_polychrony_100_3_SCH_TRA.SIG	TOR_ Polychrony Verifier_Rq#011-1 TOR_ Polychrony Verifier_Rq#012-1 TOR_ Polychrony Verifier_Rq#020-1	Safe is implemented as unsafe

*: Compilation can be forced selecting the option “force” on the Code Generation” tab. In this case, exceptions are added in the code.

3 Tests cases

For each test case, input .SIG file, generated log file and, when generation status is PASS, C file list are given here below.

3.1 Syntax error

3.1.1 Test_polychrony_001_1

3.1.1.1 Test objectives

This test case contains a syntax error: an argument is missing on a binary operator.

Thus, polychrony verifier shall detect this error (TOR_ Polychrony Verifier_Rq#001-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#002-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.1.1.2 Signal file

```
process Test_polychrony_001_1=
   %  Syntax error : * is a binary operator.
      The compiler tries to continue, but an error in found.
  %
     ( ? integer A ,B ;
       ! integer S ;
     )
  (| S := A * |)
```

3.1.1.3 Log file (`Test_polychrony_001_1.SIG_SYNTAX_ERRORS_LOG`)

The syntactic errors are printed in the Eclipse console and in the file `Test_polychrony_001_1.SIG_SYNTAX_ERRORS_LOG`:

```
/line 7 /column 14 /error  expression expected
  (| S := A * |)
```

3.1.1.4 C files

Not Generated.

3.1.2 Test_polychrony_001_2

3.1.2.1 Test objectives

This test case contains a syntax error: a *“end;”* is missing at the end of a declaration block.

Thus, polychrony verifier shall detect this error (TOR_ Polychrony Verifier_Rq#001-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#002-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.1.2.2 Signal file

```
process Test_polychrony_001_2=
  %  Syntax error : declaration requires a “end;” %
     ( ? integer A ,B ;
       ! integer S ;
     )
  (| S := A + Z
   | Z := A**2
   |)
  where
    integer Z
```

3.1.2.3 Log file (`Test_polychrony_001_2.SIG_SYNTAX_ERRORS_LOG`)

The syntactic errors are printed in the Eclipse console and in the file `Test_polychrony_001_2.SIG_SYNTAX_ERRORS_LOG`:

```
/line 12 /column 0 /error  `end' expected
```

3.1.2.4 C files

Not Generated.

3.2 Declaration error

3.2.1 Test_polychrony_003_1

3.2.1.1 Test objectives

This test case contains an error in object declaration: used signal A is not declared.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#003-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#004-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.2.1.2 Signal file

```
process Test_polychrony_003_1 =
     %Test : Signal not declared (A). Not allowed.
     %( ? integer AAA, B;
        ! integer C, D;
     )
   (| C := A+B
    | D := A-B
    |)
%Test_polychrony_003_1%;
```

3.2.1.3 Log file (Test_polychrony_003_1_LIS.SIG)

```
process Test_polychrony_003_1 =
```

```
      ( ? integer AAA, B;
        ! integer C, D;
      )%**ERROR: Signal A not declared in the process interface
      %
    (| C := A%**ERROR: Identifier not declared : A
            %+B%**ERROR: Missing argument:
            %
     | D := A%**ERROR: Identifier not declared : A
            %-B%**ERROR: Missing argument:
            %
    |)
  %Test_polychrony_003_1%;
```

3.2.1.4 C files

Not Generated.

3.2.2 Test_polychrony_003_2

3.2.2.1 Test objectives

This test case contains an error in object declaration: used objects S1 and S2 are not declared in the process interface.
Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#003-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#004-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.2.2.2 Signal file

```
process Test_polychrony_003_2= %  objects (S1,S2) not declared %
     ( ? integer A ,B ;
       ! integer S ;
     )
  (| S := A + S1
   | P{}
   | Z := A**S2 |)
  where
    integer Z;
    process P = (? X,Y; !S1, S2)
      (| S1 := X+Y
       | S2 := X*Y
       |)
  end;
```

3.2.2.3 Log file (Test_polychrony_003_2_LIS.SIG)

```
process Test_polychrony_003_2 =
     ( ? integer A, B;
       ! integer S;
     )%**ERROR: Signal S1 not declared in the process interface
     **ERROR: Signal S2 not declared in the process interface
     %
   (| S := A+S1%**ERROR: Identifier not declared : S1
            %%**ERROR: Missing argument:
            %
    | P{}%**ERROR: Identifier not declared : Y
      **ERROR: Identifier not declared : X
      **ERROR: Identifier not declared : S2
      **ERROR: Identifier not declared : S1
      %
    | Z := A**S2%**ERROR: Identifier not declared : S2
            %%**ERROR: Missing argument:
            %
    |)
   where
   integer Z;
   process P =
        ( ? X, Y;
          ! S1, S2;
        )
     (| S1 := X+Y
      | S2 := X*Y
      |)
   %P%;
```

```
    end
%Test_polychrony_003_2%;
```

3.2.2.4 C files

Not Generated.

3.2.3 Test_polychrony_003_3

3.2.3.1 Test objectives

This test case contains an error in object declaration: Process P and Q and library mLIB are not declared.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#003-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#004-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.2.3.2 Signal file

```
process Test_polychrony_003_3= %  P,Q: processes not declared
                                  mLib: library not declared %
     ( ? integer A ,B ;
       ! integer S ;
     )
  (| e::  P{}(A, #E)
   | f :: Q{}(B, #E)
   | e --> f
   | S := B**A
   |)
  where
    label e, f;
    use mLIB;
  end;
```

3.2.3.3 Log file (Test_polychrony_003_3_LIS.SIG)

```
process Test_polychrony_003_3 =
     ( ? integer A, B;
       ! integer S;
     )
   (| e :: P{}%**ERROR: Process P not declared
           %(A, #E)
    | f :: Q{}%**ERROR: Process Q not declared
           %(B, #E)
    | e --> f
    | S := B**A
    |)
   where
   label e, f;
   use mLIB;
   end
%Test_polychrony_003_3%;
```

3.2.3.4 C files

Not Generated.

3.2.4 Test_polychrony_003_4

3.2.4.1 Test objectives

This test case contains an error in object declaration: Signal S declared twice.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#003-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#004-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.2.4.2 Signal file

```
process Test_polychrony_003_4= %  Double declaration (S) %
     ( ? integer A ,B, S;
       ! integer S ;
     )
  (| S := B**A
   |)
  end;
```

3.2.4.3 Log file (Test_polychrony_003_4_LIS.SIG)

```
process Test_polychrony_003_4 =
     ( ? integer A, B, S;
       ! integer S;
     )%**ERROR: Double declaration of S
     %
   (| S := B**A |)
%Test_polychrony_003_4%;
```

3.2.4.4 C files

Not Generated.

3.2.5 Test_polychrony_003_5

3.2.5.1 Test objectives

This test case contains an error in object declaration: parameter N is not declared.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#003-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#004-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.2.5.2 Signal file

```
process Test_polychrony_003_5= %  parameter not declared %
     ( ? integer A ,B;
       ! [N]integer S ;
     )
  (| S := B**A
   |)
  end;
```

3.2.5.3 Log file (Test_polychrony_003_5_LIS.SIG)

```
process Test_polychrony_003_5 =
     ( ? integer A, B;
       ! [N%**ERROR: Identifier not declared : N
          **ERROR: Parameter not declared
         %]integer S;
     )
   (| S := B**A |)
%Test_polychrony_003_5%;
```

3.2.5.4 C files

Not Generated.

3.2.6 Test_polychrony_003_6

3.2.6.1 Test objectives

This test case contains an error in object declaration: typeT is not declared.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#003-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#004-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.2.6.2 Signal file

```
process Test_polychrony_003_6= % Error: Type not declared (T) %
     ( ? integer A ,B;
       ! integer S ;
     )
  (| S := B**U
   | U := A + B
   |)
  where
    T U;
  end;
```

3.2.6.3 Log file (Test_polychrony_003_6_LIS.SIG)

```
process Test_polychrony_003_6 =
     ( ? integer A, B;
       ! integer S;
     )
   (| S := B**U
    | U := A+B
    |)
   where
   T%**ERROR: Type not declared
   %U;
   end
%Test_polychrony_003_6%;
```

3.2.6.4 C files

Not Generated.

3.2.7 Test_polychrony_003_7

3.2.7.1 Test objectives

This test case contains an error in object declaration: labels are declared within a module.

Thus, polychrony verifier shall detect this **error** (TOR_Polychrony Verifier_Rq#003-1) and record it in a log-file (TOR_Polychrony Verifier_Rq#004-1). Moreover, C file generation shall not occur (TOR_Polychrony Verifier_Rq#020-1).

3.2.7.2 Signal file

```
module Test_polychrony_003_7 =
   %  a module cannot have labels declarations %
   label errLab;
   process P =
     ( ? integer A ,B;
       ! integer S ;
         )
     (| S := B** A
      %| errLab --> S%
      |)
end;
```

3.2.7.3 Log file (Test_polychrony_003_7_LIS.SIG)

```
module Test_polychrony_003_7 =
   label errLab;
   process P =
        ( ? integer A, B;
          ! integer S;
        )
      (| S := B**A |)
   %P%;
end %Test_polychrony_003_7%; %**ERROR: -- RESTRICTION --: a module cannot
have labels or local signals declaration
%
```

3.2.7.4 C files

Not Generated (only partial C code is generated corresponding to process P).

3.3 Type error

3.3.1 Test_polychrony_005_1

3.3.1.1 Test objectives

This test case contains an error in object type: signals C and D are declared as string and computed as integers.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#005-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#006-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.3.1.2 Signal file

```
process Test_polychrony_005_1 =
```

```
      ( ? integer A, B;
        ! string C, D;
      )
    (| C := A+B
     | D := A-B
     |)
  %Test_polychrony_005_1%;
```

3.3.1.3 Log file (Test_polychrony_005_1_LIS.SIG)

```
  process Test_polychrony_005_1 =
      ( ? integer A, B;
        ! string C%**ERROR: cf no-1
                %, D%**ERROR: cf no-2
                %;
      )
    (| C := A+B%**ERROR (ref-1): disagreement between types of operands
           %
     | D := A-B%**ERROR (ref-2): disagreement between types of operands
           %
     |)
  %Test_polychrony_005_1%;
```

3.3.1.4 C files

Not Generated.

3.3.2 Test_polychrony_005_2

3.3.2.1 Test objectives

This test case contains an error in object type:operator @ is used with integers operands, whereas it needs real ones.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#005-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#006-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.3.2.2 Signal file

```
process Test_polychrony_005_2=
     ( ? integer A, B;
       ! complex S1;
     )
  (| S1:=  A @ B |)
```

3.3.2.3 Log file (Test_polychrony_005_2_LIS.SIG)

```
process Test_polychrony_005_2 =
     ( ? integer A, B;
       ! complex S1%**ERROR: cf no-1
                 %;
     )
   (| S1 := A@B%**ERROR: unexpected type
               **ERROR (ref-1): disagreement between types of operands
               % |)
%Test_polychrony_005_2%;
```

3.3.2.4 C files

Not Generated.

3.3.3 Test_polychrony_005_3

3.3.3.1 Test objectives

This test case contains an error in object type:operator |+ is used with integers operands.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#005-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#006-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.3.3.2 Signal file

```
process Test_polychrony_005_3=
     ( ? integer A;
       ! integer S;
     )
  (| S:=  A |+ A |)
```

3.3.3.3 Log file (Test_polychrony_005_3_LIS.SIG)

```
process Test_polychrony_005_3 =
     ( ? integer A;
       ! integer S%**ERROR: cf no-1
                 %;
     )
   (| S := A |+ A%**ERROR (ref-1): disagreement between types of operands
            % |)
%Test_polychrony_005_3%;
```

3.3.3.4 C files

Not Generated.

3.3.4 Test_polychrony_005_4

3.3.4.1 Test objectives

This test case contains an error in parameter type: operator + is used with both integers and real operands.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#005-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#006-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.3.4.2 Signal file

```
process Test_polychrony_005_4=
     ( ? integer A ,B ;
       ! integer S ;
     )
  (| S := A + 6.9 +B
   |)
```

3.3.4.3 Log file (Test_polychrony_005_4_LIS.SIG)

```
process Test_polychrony_005_4 =
     ( ? integer A, B;
       ! integer S%**ERROR: cf no-1
                 %;
     )
   (| S := A+6.9+B%**ERROR: disagreement between types of the operands
           **ERROR: unexpected type
           **ERROR (ref-1): disagreement between types of operands
           % |)
%Test_polychrony_005_4%;
```

3.3.4.4 C files

Not Generated.

3.3.5 Test_polychrony_005_5

3.3.5.1 Test objectives

This test case contains an error in signal type: integer X is computed as a boolean.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#005-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#006-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.3.5.2 Signal file

```
process Test_polychrony_005_5 =
     ( ? boolean A, B, C;
       ! integer X;
     )
  (| X := (A  or B)  and not C |)
```

3.3.5.3 Log file (Test_polychrony_005_5_LIS.SIG)

```
process Test_polychrony_005_5 =
     ( ? boolean A, B, C;
       ! integer X%**ERROR: cf no-1
                 %;
     )
   (| X := (A or B) and (not C)%**ERROR (ref-1): disagreement between types
of operands
            % |)
%Test_polychrony_005_5%;
```

3.3.5.4 C files

Not Generated.

3.3.6 Test_polychrony_005_6

3.3.6.1 Test objectives

This test case contains an error in object type: signal is used instead of its clock.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#005-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#006-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.3.6.2 Signal file

```
process Test_polychrony_005_6=
  ( ? integer A;
      boolean B ;
    ! integer S ;
      )
  spec
  (| S ^= A when (A default B) |)

(| S := A**A |)
```

3.3.6.3 Log file (Test_polychrony_005_6_LIS.SIG)

```
process Test_polychrony_005_6 =
     ( ? integer A;
         boolean B;
       ! integer S;
     )
     spec (| S ^= A when (A default B)%**ERROR (ref-1): disagreement between
types of operands
              **ERROR: cf no-1
              % |)
   (| S := A**A |)
%Test_polychrony_005_6%;
```

3.3.6.4 C files

Not Generated.

3.4 Form error

3.4.1 Test_polychrony_007_1

3.4.1.1 Test objectives

This test case contains a form error : signals D is defined twice. A signal cannot have a double complete definition (See Signal V4 reference manual Chapter VI-1).

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#007-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#008-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.4.1.2 Signal file

```
process Test_polychrony_007_1 =
     % Test : Double definition of a signal (D). Not allowed.
```

```
      %( ? integer A, B;
         ! integer C, D;
       )
    (| C := A+B
     | D := A-B
     | D := A-B
     |)
  %Test_polychrony_007_1%;
```

3.4.1.3 Log file (Test_polychrony_007_1_LIS.SIG)

```
  process Test_polychrony_007_1 =
       ( ? integer A, B;
         ! integer C, D;
       )
    (| C := A+B
     | D := A-B
     | D := A-B%**ERROR: Multiple definitions for D
              %
     |)
  %Test_polychrony_007_1%;
```

3.4.1.4 C files

Not Generated.

3.4.2 Test_polychrony_007_2

3.4.2.1 Test objectives

This test case contains a form error : signal D is defined twice. It is defined by an equation of definition and by an equation of partial definitions. It is forbidden (See Signal V4 reference manual Chapter VI-1 (1-c)).

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#007-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#008-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.4.2.2 Signal file

```
process Test_polychrony_007_2 =
     % Test: D:= and D ::= (partial definition) are not allowed for the
definition of a signal D.
     %( ? integer A, B;
        ! integer C, D;
      )
   (| C := A+B
    | D := A-B
    | D ::= A-B
    |)
%Test_polychrony_007_2%;
```

3.4.2.3 Log file (Test_polychrony_007_2_LIS.SIG)

```
process Test_polychrony_007_2 =
     ( ? integer A, B;
       ! integer C, D;
      )
   (| C := A+B
    | D ::= A-B
    |)
%Test_polychrony_007_2%;
```

3.4.2.4 C files

Not Generated.

3.4.3 Test_polychrony_007_3

3.4.3.1 Test objectives

This test case contains a form error in state variable definition. A state variable can be defined exclusively by equations of partial definition (See Signal V4 reference manual Chapter VI-1 (1-d)).

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#007-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#008-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.4.3.2 Signal file

```
process Test_polychrony_007_3  =
     ( ? integer A, B, C;
       ! integer S;
     )
  (| X := A + 3 %  NOT ALLOWED FOR A STATE VAR %
   | S:=  B *C
   |)
  where
    statevar integer X;
  end
```

3.4.3.3 Log file (Test_polychrony_007_3_LIS.SIG)

```
process Test_polychrony_007_3 =
     ( ? integer A, B, C;
       ! integer S;
     )
   (| X := A+3%**ERROR: Error on definition of the state variable X
            %
    | S := B*C
    |)
   where
   statevar integer X;
   end
%Test_polychrony_007_3%;
```

3.4.3.4 C files

Not Generated.

3.4.4 Test_polychrony_007_4

3.4.4.1 Test objectives

This test case contains a form error in use of a state variable. A state variable can be defined exclusively by equations of partial definition. These equations define the next values of a state variable. The last defined value, which is the only one that can be accessed at every instant, is referred to via the special notation X? (See Signal V4 reference manual Chapter VI-1 (1-d)).

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#007-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#008-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.4.4.2 Signal file

```
process Test_polychrony_007_4 = %  errors x::= for a state var %
     ( ? integer A, B, C;
       ! integer S;
     )
  (| X ::= A+3
   | S:=  X + B *C  %  NOT ALLOWED FOR A STATE VAR, USE X? (previous value)%
   |)
  where
    statevar integer X;
  end
```

3.4.4.3 Log file (Test_polychrony_007_4_LIS.SIG)

```
process Test_polychrony_007_4 =
     ( ? integer A, B, C;
       ! integer S;
     )
   (| X ::= A+3
    | S := X+(B*C)%**ERROR: Constant or state variable not allowed
          %
    |)
   where
   statevar integer X;
   end
%Test_polychrony_007_4%;
```

3.4.4.4 C files

Not Generated.

3.4.5 Test_polychrony_007_5

3.4.5.1 Test objectives

This test case contains a form error: an action must be defined as external.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#007-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#008-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.4.5.2 Signal file

```
process Test_polychrony_007_5= % Error: An action must be an external %
     ( ? integer A ,B;
       ! integer S ;
     )
  (| S := P(A,B)
   |)
  where
    action P = (? U,V; ! X;)
      (| X := U+V |)
  end;
```

3.4.5.3 Log file (Test_polychrony_007_5_LIS.SIG)

```
process Test_polychrony_007_5 =
     ( ? integer A, B;
       ! integer S;
     )
   (| S := P(A, B) |)
   where
   action P =
        ( ? U, V;
          ! X;
        )
      (| X := U+V |)
   %P%; %**ERROR: An action must be external
   %
   end
%Test_polychrony_007_5%;
```

3.4.5.4 C files

Not Generated.

3.4.6 Test_polychrony_007_6

3.4.6.1 Test objectives

This test case contains a form error: function model does not corresponds to its interface.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#007-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#008-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.4.6.2 Signal file

```
process Test_polychrony_007_6= % Error: Conflicting between interface and
model declaration  %
     ( ? integer A ,B;
       ! integer S ;
     )
  (| S := P(A,B)
   |)
  where
    function P = (? V; ! X;)
      (| X := V+V |)
  end;
```

3.4.6.3 Log file (Test_polychrony_007_6_LIS.SIG)

```
process Test_polychrony_007_6 =
     ( ? integer A, B;
       ! integer S;
     )
   (| S := P(A, B)%**ERROR: Conflicting number of input signals
           % |)
   where
   function P =
        ( ? V;
          ! X;
        )
      (| X := V+V |)
   %P%;
   end
%Test_polychrony_007_6%;
```

3.4.6.4 C files

Not Generated.

3.4.7 Test_polychrony_007_7

3.4.7.1 Test objectives

This test case contains a form error: there is a conflict between effective model and formal model. In Signal (See chapter XI, Signal V4 reference manual for more details), it is possible to declare model types (`type node TT_P=...`) that represents the interface of a model. A model type can be (like for the model) a function, a node, a process, or an action. Such a model type can be used to specify formal process models as formal parameters of process models (`process FOO = TT_P..., node FOO1 = TT_P...`). The compiler must verify that the class of the model (node, function, process, action) is the same than the class of the model type. For the example, the declaration of the model FOO (a process) is erroneous (the model type is declared as node), the declaration of the model FOO1 is correct.

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#007-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#008-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.4.7.2 Signal file

```
process Test_polychrony_007_7 =(? integer x; ! integer y, z;)  % ERROR:
Conflicting between effective model and formal model  %
  (| z := Sampling {FOO1, 78}(y, ^x)
   | y := Sampling {FOO, 56} (x, ^x)
   |)
  where
    type node TT_P=(?integer a; ! integer b;);
    type monType = external;
     process Sampling = { process TT_P P;  integer PP;}
          (?a; event h; !b ;)
           (|  (| P{}
               | cell_a ^= input_a ^+ h
               | cell_a := input_a default cell_a $1
               | a := cell_a when h
               |) where cell_a; a end
            | input_a := a
            |)
          where
            input_a;
          end ;

     process FOO = TT_P (| b:= 2*a |);
     node FOO1 = TT_P (| b:= a+a-1 |);
   end ;
```

3.4.7.3 Log file (Test_polychrony_007_7_LIS.SIG)

```
process Test_polychrony_007_7 =
     ( ? integer x;
       ! integer y, z;
     )
   (| z := Sampling{FOO1, 78}(y, ^x)
    | y := Sampling{FOO, 56}(x, ^x)
    |)
   where
   type node TT_P = ( ? integer a;
                      ! integer b;
```

```
   type monType = external;
   process Sampling =
        { process TT_P P;%**ERROR: Conflicting between type model and formal
model
          %
          integer PP; }
        ( ? a;
          event h;
        ! b;
        )
      (| (| P{}
          | cell_a ^= input_a ^+ h
          | cell_a := input_a default (cell_a$1)
          | a := cell_a when h
          |) where cell_a;
                  a; end
       | input_a := a
       |)
      where
      input_a;
      end
   %Sampling%;
   process FOO =
        TT_P
      (| b := 2*a |)%**ERROR: Conflicting between interface and model
declaration
      %
   %FOO%;
   node FOO1 =
        TT_P
      (| b := (a+a)-1 |)
   %FOO1%;
   end
%Test_polychrony_007_7%;
```

3.4.7.4 C files

Not Generated.

3.4.8 Test_polychrony_007_8

3.4.8.1 Test objectives

This test case contains a form error: the compiling of a module is the compiling of the public processes declared in the module: by default a module is public, the keyword private restricts the visibility of a model. When a process is parametrized by a model (for example, **process** Sampling = { **process** TT_P P; **integer** PP;} in the test), an error is currently detected and the program is rejected.
To compile it, some information must be known such that the dependencies, the clock relations between the inputs, outputs and the protocol used for the implementation (like for the separated compiling).

Thus, polychrony verifier shall detect this **error** (TOR_ Polychrony Verifier_Rq#007-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#008-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.4.8.2 Signal file

```
 module Test_polychrony_007_8 = %**Restriction: A main model with formal
process is not yet implemented.%
      type process TT_P=(?a; !b;);
      type monType = external;
      process Sampling = { process TT_P P;  integer PP;}
            (?a; event h; !b;)
        (|  (| P{}
             | cell_a ^= input_a ^+ h
             | cell_a := input_a default cell_a $1
             | a := cell_a when h
             |) where cell_a; a end
          | input_a := a
          |)
        where
          input_a;
        end ;

      process FOO = TT_P (| b:= 2*a |);

      process myMain=(? integer x; ! integer y;)
        (| y := Sampling {FOO, 56} (x, ^x) |)
      ;
end ;
```

3.4.8.3 Log file (Test_polychrony_007_8_LIS.SIG)

```
module Test_polychrony_007_8 =
   type process TT_P = ( ? a;
                         ! b;
                       );
   type monType = external;
   process Sampling =
        { process TT_P P;
          integer PP; }
        ( ? a;
            event h;
          ! b%**WARNING: Signal b not defined in the graph (it will be
defined by b := b)
%;
        )
      (| (| P{}
```

```
            | cell_a ^= input_a ^+ h
            | cell_a := input_a default (cell_a$1)
                                    %**WARNING: XZX_34 should be initialized
                                    **WARNING: XZX_89 should be initialized
                                    %
            | a := cell_a when h
            |) where cell_a;
                     a; end
          | input_a := a
          |)
        where
        input_a;
        end
    %Sampling%; %**ERROR: A main model with formal process is not yet
implemented.%
    process FOO =
         TT_P
        (| b := 2*a |)
    %FOO%;
    process myMain =
         ( ? integer x;
           ! integer y;
         )
        (| y := Sampling{FOO, 56}(x, ^x) |)
    %myMain%;
end %Test_polychrony_007_8%;
```

3.4.8.4 C files

Not Generated.

3.5 Clock constraints

3.5.1 Test_polychrony_009_1

3.5.1.1 Test objectives

This test case contains a clock constraint: both operands of + operator do not have the same clock (A and (A when A>0). To prove the equality, it is necessary to prove that A >0, that is impossible without more information on the environment. This example is a program with a master clock (clock of A).

Thus, polychrony verifier shall detect this constraint (TOR_ Polychrony Verifier_Rq#009-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#010-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.5.1.2 Signal file

```
process Test_polychrony_009_1  =
     ( ? integer A;
       ! integer X;
     )
  (| X :=  A + (A when A>0) |)
```

3.5.1.3 Log file (Test_polychrony_009_1_BOOL_TRA.SIG)

```
process Test_polychrony_009_1_BOOL_TRA =
     ( ? integer A;
       ! integer X;
     )
   pragmas
      Main
   end pragmas
   (| (| Tick := true
       | when Tick ^= A ^= X
       | ACT_Tick{}
       |)
    | (| (| Tick ^= when C_CLK |) |)%**WARNING: Clocks constraints
      %
    |)
   where
   boolean Tick;
   process ACT_Tick =
        ( ? boolean Tick;
            integer A;
          ! integer X;
        )
      (| when Tick ^= C_CLK
       | (| X := A+A
          | C_CLK := A>0
          |)
       |)
      where
      boolean C_CLK;
      end
   %ACT_Tick%;
   end
%Test_polychrony_009_1_BOOL_TRA%; %(| Tick := true when Tick
                                    | C_CLK_7 := not (A>0)
                                    |)**WARNING: unused signals
```

```
%
```

3.5.1.4 C files

Not Generated.

3.5.2 Test_polychrony_009_2

3.5.2.1 Test objectives

This test case contains a clock constraint : both operands of + operator do not have the same clock (A and (A when C)) or this clock equality is not proven. This example is a program with several master clocks (clock of A, clock of C).

Thus, polychrony verifier shall detect this constraint (TOR_ Polychrony Verifier_Rq#009-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#010-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.5.2.2 Signal file

```
process Test_polychrony_009_2 =
     ( ? integer A;
         boolean C;
       ! integer X;
     )
  (| X :=  A + (A when C) |)
```

3.5.2.3 Log file (Test_polychrony_009_2_BOOL_TRA.SIG)

```
process Test_polychrony_009_2_BOOL_TRA =
     ( ? integer A;
         boolean C, C_C, C_X;
       ! integer X;
     )
   pragmas
      Main
   end pragmas
   (| (| Tick := true
       | when Tick ^= C_C ^= C_X
       | ACT_Tick{}
       |)
    | (| (| when C_X ^= when C_CLK_29 |) |)%**WARNING: Clocks constraints
      %
    |)
   where
   boolean Tick;
   process ACT_Tick =
        ( ? boolean Tick;
            integer A;
            boolean C, C_C, C_X;
          ! integer X;
        )
      (| when Tick ^= C_57 ^= C_CLK_29
       | (| when C_C ^= C |)
       | (| when C_X ^= A ^= X
          | ACT_C_X{}
          |)
       | (| C_57 := left_tt ( C,C_C )
          | C_CLK_29 := C_X and C_57
          |)
       |)
      where
      boolean C_57, C_CLK_29;
      process ACT_C_X =
```

```
            ( ? integer A;
                boolean C_X;
              ! integer X;
            )
          (| (| X := A+A |) |)
       %ACT_C_X%;
       end
   %ACT_Tick%;
   end
%Test_polychrony_009_2_BOOL_TRA%; %(| Tick := true when Tick
                                    | C_CLK_24 := not C_C
                                    | C_CLK_26 := not C_X
                                    | C_CLK_20 := not C
                                    |)**WARNING: unused signals

%
```

3.5.2.4 C files

Not Generated.

3.5.3 Test_polychrony_009_3

3.5.3.1 Test objectives

This test case contains a clock constraint: the system is not deterministic. There are several solutions for fixing the clock of the signal x.

Thus, polychrony verifier shall detect this constraint (TOR_ Polychrony Verifier_Rq#009-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#010-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.5.3.2 Signal file

```
process Test_polychrony_009_3 =
  ( ? integer a ;
      event h;
    ! integer x; )

  (| y := x when h
   | u:= y+z
   | z:= u$1 init 0
   | x := a default u
   |)

 where
   integer z, u, y;
end
```

3.5.3.3 Log file (Test_polychrony_009_3_BOOL_TRA.SIG)

```
process Test_polychrony_009_3_BOOL_TRA =
     ( ? integer a;
         boolean C_a, h, C_y;
       ! integer x;
     )
   pragmas
      Main
   end pragmas
   (| (| Tick := true
       | when Tick ^= C_a ^= h ^= C_y
       | ACT_Tick{}
       |)
    | (| (| when C_y ^= when C_CLK_69 |)
       | (| when C_x ^= when C_CLK_75 |)
       | (| when C_CLK_71 ^= when C_CLK_77 |)
       |)%**WARNING: Clocks constraints
      %
    |)
   where
   boolean Tick;
   process ACT_Tick =
        ( ? boolean Tick;
            integer a;
            boolean C_a, h, C_y;
          ! integer x;
        )
      (| when Tick ^= C_x ^= C_CLK_69 ^= C_CLK_71 ^= C_CLK_75 ^= C_CLK_77
```

```
        | (| when C_a ^= a |)
        | (| when C_y ^= u
           | ACT_C_y{}
           |)
        | (| when C_x ^= x
           | ACT_C_x{}
           |)
        | (| C_x := C_a or C_y
           | C_CLK_69 := C_x and h
           | C_CLK_71 := (not C_a) and C_y
           | C_CLK_75 := C_a or C_CLK_69
           | C_CLK_77 := (not C_a) and C_CLK_69
           |)
        |)
      where
      boolean C_x, C_CLK_69, C_CLK_71, C_CLK_75, C_CLK_77;
      integer u;
      process ACT_C_y =
           ( ? integer x;
               boolean C_y;
             ! integer u;
           )
         (| when C_y ^= z ^= y
          | (| z := u$1 init 0
             | u := y+z
             | y := x when C_y
             |)
          |)
         where
         integer z, y;
         end
      %ACT_C_y%;
      process ACT_C_x =
           ( ? integer a, u;
               boolean C_a, C_x, C_CLK_71;
             ! integer x;
           )
         (| (| x := (a when C_a) default (u when C_CLK_71) |) |)
      %ACT_C_x%;
      end
   %ACT_Tick%;
   end
%Test_polychrony_009_3_BOOL_TRA%; %(| Tick := true when Tick
                                    | C_CLK_61 := not C_a
                                    | C_CLK_63 := not h
                                    | C_CLK_65 := not C_y
                                    |)**WARNING: unused signals

%
```

3.5.3.4 C files

Not Generated.

3.6 Cycle detection

3.6.1 Test_polychrony_011_1

3.6.1.1 Test objectives

This test case contains a computation cycle : x is used to compute u which is used to computed x .

Thus, polychrony verifier shall detect this cycle (TOR_ Polychrony Verifier_Rq#011-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#012-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.6.1.2 Signal file

```
process Test_polychrony_011_1 =  %  cycle: x -> u -> x %
  ( ? integer a,b;
    ! integer x;
      )
  (| x := a + u
   | u := x ** b
   |)
  where
    integer u;
  end;
```

3.6.1.3 Log file (Test_polychrony_011_1_BOOL_CYC.SIG)

```
process Test_polychrony_011_1_BOOL_CYC =
     ( )
   (| (| x := a+(x**b) |)
    | (| {x --> x} when Tick |)
    |)
%Test_polychrony_011_1_BOOL_CYC%;
```

3.6.1.4 C files

Not Generated.

3.6.2 Test_polychrony_011_2

3.6.2.1 Test objectives

This test case contains a computation cycle : in this example, b depends on a when c is true, and a depends on b when c is false. At an instant, c can be true or false, so only one depedence is effective. This kind of cycle is called a « false circuit ». The signal compiler detects such cycle but it does not generate code (restriction).

Thus, polychrony verifier shall detect this cycle (TOR_ Polychrony Verifier_Rq#011-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#012-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.6.2.2 Signal file

```
process Test_polychrony_011_2 =
%  a «false» cycle: a->b->a, but at each instant only one dependence is
effective. This example is currently rejected for the code generation.%
  ( ? integer a,b;
      boolean c;
    ! integer x;
      )
  (| x := (a+b) when c
   | { a --> b } when c
   | { b --> a } when not c
   |)
```

3.6.2.3 Log file (Test_polychrony_011_2_BOOL_CYC.SIG)

```
process Test_polychrony_011_2_BOOL_CYC =
     ( )
   (| (| a := ~inputenv()
       | b := ~inputenv()
       |)
    | (| {a --> b} when C_CLK_434
       | {b --> a} when C_x
       |)
    |)
%Test_polychrony_011_2_BOOL_CYC%;
```

3.6.2.4 C files

Not Generated.

3.6.3 Test_polychrony_011_3

3.6.3.1 Test objectives

This test case contains a computation cycle: when a signal D (not an input signal) is not defined in a program, the compiler adds the equation D:=D. When this signal is useful for a computation, or it is an output signal, the added definition will not be removed from the program and the compiler will detect a cycle when the code generation is required by the user.

Thus, polychrony verifier shall detect this cycle (TOR_ Polychrony Verifier_Rq#011-1) and record it in a log-file (TOR_ Polychrony Verifier_Rq#011-1). Moreover, C file generation shall not occur (TOR_ Polychrony Verifier_Rq#020-1).

3.6.3.2 Signal file

```
process Test_polychrony_011_3=
% WARNING:  When a signal (not an input) is not defined in the program (D)
is defined by D:=D. But for this example, a cycle will be detected if the
code generation is asked.%
     ( ? integer A ,B ;
       ! integer S, D;
     )
  (| S := A + B |)
```

3.6.3.3 Log file (Test_polychrony_011_3_BOOL_CYC.SIG)

```
process Test_polychrony_011_3_BOOL_CYC =

   ( )

  (| (| D := D |)

   | (| {D --> D} when C_D |)

   |)

%Test_polychrony_011_3_BOOL_CYC%;
```

3.6.3.4 C files

Not Generated.

3.7 No detected error case

3.7.1 Test_polychrony_020_1

3.7.1.1 Test objectives

This test case contains no error.

Thus, C code generation shall occur (TOR_ Polychrony Verifier_Rq#020-1) and log file shall contain no error message. C code concordance with the reference ones shall be checked.

3.7.1.2 Signal file

```
process Test_polychrony_020_1 =
     ( ? real a;
```

```
      real b;
      real c;
    ! real x2;
      real x1;
      boolean oc;
  )
(| (| OK := a/=0.0
    | (err1, a1, x2) := SecondDegree(OK, a, b, c)
    | (a2, err2) := FirstDegree(when (not OK), b, c)
    | (| x1 := a1 default a2
       | oc := (when err1) default err2
       |)
    | a ^= b ^= c
    |)/ err2, a2, err1, a1, OK |)
where
process FirstDegree =
     ( ? boolean Compute;
         real b;
         real c;
       ! real x;
         boolean err2;
     )
   (| (| (| bb1 := bb when (bb/=0.0)
          | cc1 := cc when (bb/=0.0)
          | x := -(cc1/bb1)
          |)/ cc1, bb1
       | err2 := (cc/=0.0) when (bb=0.0)
       | (| bb := b when Compute
          | cc := c when Compute
          |)
       |)/ cc, bb |)
   where
   bb1, cc1;
   real bb, cc;
   end
%FirstDegree%;
process SecondDegree =
     ( ? boolean OK;
         real a;
         real b;
         real c;
       ! boolean err;
         real x21;
         real x2;
     )
   (| (| (nul, err, d) := Discriminant{}(bb, cc)
       | (| aa := a when OK
          | bb := (b when OK)/aa
          | cc := (c when OK)/aa
          | c ^= b ^= a
          |)/ aa
       | x := (-(bb when nul))/2.0
       | (stable, x1, x2) := twoRoots{epsilon}(d, bb)
       | (| x2 := x
          | x21 := x1 default x2
          |) where real x2; end
```

```
    | c ^= when stable
    |)/ stable, x1, x, cc, bb, d, nul |)
where
constant real epsilon = 0.00001;
process Discriminant =
     ( ? real b;
         real c;
       ! event nul;
         boolean err;
         real d;
     )
   (| dd := (b*b)-(4.0*c)
    | d := dd when (dd>0.0)
    | nul := when (dd=0.0)
    | err := when (dd<0.0)
    |)
   where
   real dd;
   end
%Discriminant%;
process twoRoots =
     { real eps; }
     ( ? real discr;
         real b;
       ! boolean stable;
         real x1;
         real x2;
     )
   (| (| (stable, d) := rac{eps}(discr)
      | bb := (b cell (^d) init 0.0) when (^d)
      | x2 := -((bb-d)/2.0)
      | x1 := -((bb+d)/2.0)
      |)/ bb, d |)
   where
   process rac =
        { real epsilon; }
        ( ? real x;
          ! boolean stable;
            real y;
        )
      (| (| (| mx := x cell (^yy) init 0.0
             | next_yy := (((yy+(mx/yy))/2.0) when biterate) default yy
             | yy := (x/2.0) default (next_yy$1 init 1.0)
             |)/ mx
         | (| biterate := (^x) default (not stable)
             | next_stable := abs(next_yy-yy)<epsilon
             | stable := next_stable$1 init true
             | yy ^= stable
             | y := yy when (next_stable and biterate)
             |)/ next_stable
         |)/ biterate, next_yy, yy |)
      where
      process abs =
           ( ? x;
             ! s;
           )
```

```
                (| s := (x when (x>=0.0)) default (-x) |)
            %abs%;
            real yy, next_yy, mx;
            boolean next_stable, biterate;
            end
         %rac%;
         real d;
         bb;
         end
      %twoRoots%;
      event nul;
      real d, bb, aa, cc, x;
      x1, stable;
      end
   %SecondDegree%;
   boolean OK, err1;
   real a1, a2;
   err2;
   end
%Test_polychrony_020_1%;
```

3.7.1.3 Log file (Test_polychrony_020_1_LIS.SIG)

```
process Test_polychrony_020_1 =
     ( ? real a;
         real b;
         real c;
       ! real x2;
         real x1;
         boolean oc;
     )
   (| (| OK := a/=0.0
       | (err1, a1, x2) := SecondDegree(OK, a, b, c)
       | (a2, err2) := FirstDegree(when (not OK), b, c)
       | (| x1 := a1 default a2
          | oc := (when err1) default err2
          |)
       | a ^= b ^= c
       |)/ err2, a2, err1, a1, OK |)
   where
   process FirstDegree =
        ( ? boolean Compute;
            real b;
            real c;
          ! real x;
            boolean err2;
        )
      (| (| (| bb1 := bb when (bb/=0.0)
             | cc1 := cc when (bb/=0.0)
             | x := -(cc1/bb1)
             |)/ cc1, bb1
         | err2 := (cc/=0.0) when (bb=0.0)
         | (| bb := b when Compute
            | cc := c when Compute
            |)
         |)/ cc, bb |)
      where
```

```
   bb1, cc1;
   real bb, cc;
   end
%FirstDegree%;
process SecondDegree =
     ( ? boolean OK;
         real a;
         real b;
         real c;
       ! boolean err;
         real x21;
         real x2;
     )
   (| (| (nul, err, d) := Discriminant{}(bb, cc)
       | (| aa := a when OK
          | bb := (b when OK)/aa
          | cc := (c when OK)/aa
          | c ^= b ^= a
          |)/ aa
       | x := (-(bb when nul))/2.0
       | (stable, x1, x2) := twoRoots{epsilon}(d, bb)
       | (| x2 := x
          | x21 := x1 default x2
          |) where real x2; end
       | c ^= when stable
       |)/ stable, x1, x, cc, bb, d, nul |)
   where
   constant real epsilon = 0.00001;
   process Discriminant =
        ( ? real b;
            real c;
          ! event nul;
            boolean err;
            real d;
        )
      (| dd := (b*b)-(4.0*c)
       | d := dd when (dd>0.0)
       | nul := when (dd=0.0)
       | err := when (dd<0.0)
       |)
      where
      real dd;
      end
   %Discriminant%;
   process twoRoots =
        { real eps; }
        ( ? real discr;
            real b;
          ! boolean stable;
            real x1;
            real x2;
        )
      (| (| (stable, d) := rac{eps}(discr)
          | bb := (b cell (^d) init 0.0) when (^d)
          | x2 := -((bb-d)/2.0)
          | x1 := -((bb+d)/2.0)
```

```
            |)/ bb, d |)
        where
        process rac =
             { real epsilon; }
             ( ? real x;
               ! boolean stable;
                 real y;
             )
           (| (| (| mx := x cell (^yy) init 0.0
                  | next_yy := (((yy+(mx/yy))/2.0) when biterate) default yy
                  | yy := (x/2.0) default (next_yy$1 init 1.0)
                  |)/ mx
              | (| biterate := (^x) default (not stable)
                 | next_stable := abs(next_yy-yy)<epsilon
                 | stable := next_stable$1 init true
                 | yy ^= stable
                 | y := yy when (next_stable and biterate)
                 |)/ next_stable
              |)/ biterate, next_yy, yy |)
           where
           process abs =
                ( ? x;
                  ! s;
                )
              (| s := (x when (x>=0.0)) default (-x) |)
           %abs%;
           real yy, next_yy, mx;
           boolean next_stable, biterate;
           end
        %rac%;
        real d;
        bb;
        end
     %twoRoots%;
     event nul;
     real d, bb, aa, cc, x;
     x1, stable;
     end
  %SecondDegree%;
  boolean OK, err1;
  real a1, a2;
  err2;
  end
%Test_polychrony_020_1%;
```

3.7.1.4 C files

The following files are generated :

- Test_polychrony_020_1_body.c
- Test_polychrony_020_1_body.h
- Test_polychrony_020_1_externals.h
- Test_polychrony_020_1_externalsProc.h
- Test_polychrony_020_1_io.c
- Test_polychrony_020_1_main.c
- Test_polychrony_020_1_types.h

They are to be compared with the ones furnished within the verification kit.

Due to the method used for parameter naming, some differences in parameter name suffixes may be observed (eg. XZX_224 instead of XZX_266). This is not to be considered as an error.

3.8 Warnings

3.8.1 Test_polychrony_100_1

3.8.1.1 Test objectives

This test case contains a warning : input signal B is not used.

This is not an error as all process definitions ans clocks can be computed, thus C file generation shall occur (TOR_ Polychrony Verifier_Rq#020-1).

3.8.1.2 Signal file

```
process Test_polychrony_100_1=  %  **WARNING: Input B unused in the process
body %
     ( ? integer A ,B;
       ! integer S;
     )
  (| S := A + A |)
```

3.8.1.3 Log file (Test_polychrony_100_1_LIS.SIG)

```
process Test_polychrony_100_1 =
     ( ? integer A, B;
       ! integer S;
     )%**WARNING: Input B unused in the process body
     %
   (| S := A+A |)
%Test_polychrony_100_1%;
```

3.8.1.4 C files

The following files are generated :

- Test_polychrony_100_1_body.c
- Test_polychrony_100_1_body.h
- Test_polychrony_100_1_externals.h
- Test_polychrony_100_1_externalsProc.h
- Test_polychrony_100_1_io.c
- Test_polychrony_100_1_main.c
- Test_polychrony_100_1_types.h

They are to be compared with the ones furnished within the verification kit.
Due to the method used for parameter naming, some differences in parameter name suffixes may be observed (eg. XZX_224 instead of XZX_266). This is not to be considered as an error.

3.8.2 Test_polychrony_100_2

3.8.2.1 Test objectives

This test case contains a warning : delayed signal (A$1) is not initialized.

This is not an error as all process definitions ans clocks can be computed, thus C file generation shall occur (TOR_ Polychrony Verifier_Rq#020-1).

3.8.2.2 Signal file

```
process Test_polychrony_100_2 =
  % WARNING:  When a signal is not initialized (for a delay), the Signal
default value is used (Signal Manual reference) %
     ( ? integer A ;
       ! integer S;
     )
  (| S := A $ 1 |)
```

3.8.2.3 Log file (Test_polychrony_100_2_LIS.SIG)

```
process Test_polychrony_100_2 =
     ( ? integer A;
       ! integer S%**WARNING: S should be initialized
                 %;
     )
   (| S := A$1 |)
%Test_polychrony_100_2%;
```

3.8.2.4 C files

The following files are generated :

- Test_polychrony_100_2_body.c
- Test_polychrony_100_2_body.h
- Test_polychrony_100_2_externals.h
- Test_polychrony_100_2_externalsProc.h
- Test_polychrony_100_2_io.c
- Test_polychrony_100_2_main.c
- Test_polychrony_100_2_types.h

They are to be compared with the ones furnished within the verification kit.
Due to the method used for parameter naming, some differences in parameter name suffixes may be observed (eg. XZX_224 instead of XZX_266). This is not to be considered as an error.

3.8.3 Test_polychrony_100_3

3.8.3.1 Test objectives

This test case contains a warning: **safe** is implemented as **unsafe(**see XI–1 Classes of process models of the Signal V4 reference manual). A Signal process model establishes a designation between a name and a set of parameterized equations; any reference to this name is formally replaced by the designated equations. Such a model can be attributed by one of the following keyword: safe, deterministic or unsafe(default):

- A process is said **safe** if it is an iteration of function (on the inputs). Two different instantiations of a safe process with the same input values will provide the same results. Such a process is memoryless. It cannot call external processes that are not safe.
- A process is said **deterministic** automaton, if it is a function of sequences, from initial states, trajectories of the inputs and trajectories of the clocks of the outputs (considered, in some sense, as inputs), into trajectories of the outputs. This corresponds to the notion of deterministic process (on the inputs). Its only possible “side effects” are changes to its private memory. Two different instantiations of a deterministic automaton process with the same sequences of input values (and output clocks), and in the same initial conditions, will provide the same sequences of outputs. It cannot call external processes that are not safe. Any safe process is deterministic automaton.
- A process is **unsafe** in all other cases.

This is not an error as all process definitions ans clocks can be computed, thus C file generation shall occur (TOR_ Polychrony Verifier_Rq#020-1).

3.8.3.2 Signal file

```
process Test_polychrony_100_3=
         % Warning: -- RESTRICTION -- : Safe is implemented as unsafe. %
     ( ? integer A ,B;
       ! integer S ;
     )
  (| S := P(A,B)
   |)
  where
    process P = (? U,V; ! X;)
    safe
      (| X := U+V |)
  end;
```

3.8.3.3 Log file (Test_polychrony_100_3_LIS.SIG)

```
process Test_polychrony_100_3 =
     ( ? integer A, B;
       ! integer S;
     )
   (| S := P(A, B) |)
   where
   process P =
        ( ? U, V;
          ! X;
        )
        safe%**WARNING: -- RESTRICTION -- : Safe is implemented as unsafe. %
      (| X := U+V |)
   %P%;
   end
```

```
%Test_polychrony_100_3%;
```

3.8.3.4 C files

The following files are generated :

- Test_polychrony_100_3_body.c
- Test_polychrony_100_3_body.h
- Test_polychrony_100_3_externals.h
- Test_polychrony_100_3_externalsProc.h
- Test_polychrony_100_3_io.c
- Test_polychrony_100_3_main.c
- Test_polychrony_100_3_types.h

They are to be compared with the ones furnished within the verification kit.
Due to the method used for parameter naming, some differences in parameter name suffixes may be observed (eg. XZX_224 instead of XZX_266). This is not to be considered as an error.

REVISION HISTORY

Rev.	*Date*	*Prepared by*	*Reviewed by*	*Approved by*	*Reason for change*
1	July 04th, 2012	T. Le Lagadec	M. Heitz	E. Bonnafous	Initial release
2	July 17th, 2012	L.Besnard			Adding explanations

FILES

Software packages	*User files*
Windows 2000	
Word 2000	*Template Document:* POLYCHRONY-VERIFIER-TVCP-001draft_LB.doc

DISTRIBUTION

This document is available in computerised form on a server.

Accordingly, it is not formally distributed in hard-copy form.

Should a printed out copy of this document be used, make sure that you indeed have the latest applicable version by consulting the server.