Signal tool box

This section describes the **signal** command.

SYNOPSIS

- **signal -h, signal -vers**
- **signal [OPTIONS]**

- ***signalsh [-com=CFILE] [-d=dirname]***

DESCRIPTION

- **signal -h** displays a summary of this documentation.
- **signal -vers** displays Signal compiler current version.
- **signal INFILE [OPTIONS]** according to other given options, compiles a Signal *process* or *module* named **FOO**, defined in **INFILE**

- ***signalsh [-com=CFILE] [-d=dirname]*** *instead of using compilation options, reads the compilation scenario in standard input or in a file.*

ENVIRONMENT

The following environment variables are assumed to be defined.

- **$pKDoc_ROOT** is not required by the **signal** command. Its value is the directory that contains access to Polychrony full documentation, including this "Polychrony_SignalToolbox_use" file.
- **$Signal_bin** is required by the **signal** command. Its value is the directory that contains files required by the compiler.
- **$SignalLib_Std** is required by the **signal** command when the compiled program has occurrences of standard Signal process. Its value is the directory that contains standard Signal library
- **$SIGNAL_LIBRARY_PATH** is required by the **signal** command when the compiled program has occurrences of Signal process defined in a library. Its value is the list of directories that contain standard and user defined Signal library.

These variables are set by "source $pK_ROOT/PolychronyToolset_setup", where the value of "$pK_ROOT" is the directory that contains the root of the installation of Polychrony tool box.

EXAMPLES

When the file FIII.SIG contains an executable process FOO, and WD is the current directory path:

- **signal FIII.SIG** compiles FOO. Diagnosis is displayed.to the standard output.
- **signal FIII.SIG -c** compiles FOO. The following C code files are generated in the directory WD/FOO:
  - FOO_XXX.h,, where XXX belongs to {externalsProc, externals, types, body},
  - FOO_YYY.c, where YYY belongs to {main, body,io}

- **signal FIII.SIG -tra -d=tmp** compiles FOO. The following Signal code files FOO_BASIC_TRA.SIG and FOO_POLY_TRA.SIG are generated in directory WD/tmp.

OPTIONS

General rules

- An option identification starts with “-” character immediately followed by a letter or a sequence of letters
- When an option has complementary sub-options, each of them is identified by “--” string immediately followed by a letter or a sequence of letters
- When an option (a sub-option) has a value this value immediately follow the option (sub-option) identification. The value of an option is made of the character “=” immediately followed by a string
- The ordering of independent options is free and meaningless.
- Unless otherwise specified, the ordering of elements in value, sub-options in option is free and meaningless
- When a *cumulative option* has several occurrences in the command line, the value of the option is the upper bound of its occurrences. A cumulative option is an option with non exclusive values.
- **INFILE** argument must have one and only one occurrence in the command line. **Deprecated**
- When an optional *non cumulative* option has several occurrences in the command line all occurrences but the last are ignored. **Deprecated**
- **A non cumulative argument, including INFILE, cannot have several occurrences in the command line.**

Notation

Let **XX** be an absolute path

- for a relative path **PATH** path(XX, PATH) refers to PATH in XX
- for an absolute path file **PATH** path(XX, PATH) is PATH

Let **WD** be the absolute path of the current directory, unless otherwise specified

- path(PATH)=path(WD,PATH)

Options Controlling the Input

INFILE Deprecated

The Signal source file is path(INFILE)

- INFILE has neither implicit nor mandatory suffix. "SIG" is the recommended suffix.
- path(INFILE) contains a unique Signal Module or Process named **FOO**, in accordance with the Signal syntax
- When path(INFILE) contains a Signal Module FOO, all compiled processes in FOO are compiled with the same OPTIONS

-par=PARF Deprecated

path(**PARF**) is the Signal static parameter list for instantiating **FOO**

- PARF has neither implicit nor mandatory suffix . "PAR" is the recommended suffix.
- path(PARF) contains a unique list of actual static parameters in accordance with the Signal syntax

- if **FOO** has static formal parameters this argument is mandatory, otherwise it is optional

[-s=]INFILE [--par=PARF | --mod :name1 :name2 ...] NEW NOTATION

This option is required: it indicates the source file to be compiled.
The sub-options --**par** and --**mod** are exclusive.
The Signal source file is path(INFILE)

- INFILE has neither implicit nor mandatory suffix. "SIG" is the recommended suffix.
- path(INFILE) contains a unique Signal Module or Process named **FOO**, in accordance with the Signal syntax

When ***--par=PARF*** is present, path(INFILE) contains a process FOO, ***path(PARF)*** is the Signal static parameter list for instantiating FOO

- PARF has neither implicit nor mandatory suffix . "PAR" is the recommended suffix.
- path(PARF) contains a unique list of actual static parameters in accordance with the Signal syntax
- if **FOO** has static formal parameters this argument is mandatory, otherwise it is optional

When --***mod :name1 :name2 ...*** is present, path(INFILE) contains a module FOO, defining *public* processes named ***name1, name2 ...*** if one of those listed processes is private or requires static parameters the compilation fails.

- --***mod*** with an empty list of names compiles all public processes. They must all be free of static parameter declaration

When none of these sub-options is present:

- if path(INFILE) contains a process FOO, FOO is compiled without actual static parameters
- if path(INFILE) contains a model FOO, all public processes free of static parameter declaration are compiled.

Options Controlling the Output

Fatal syntax error

Initial syntax errors are written to the standard error file.

-war

display warnings to standard output, or in FOO_LIS.SIG file if -lis is present

-v

- When -**v** is absent a succinct compilation log is displayed to the standard output
- When -**v** is present, a detailed compilation log is displayed to the standard output

-tm
enable the traceability mode

-d=DIR

If this option is present, output files are created in directory PATH=path(DIR), otherwise PATH=path(FOO). **Deprecated**
If this option is present, the *implicit output directory* is PATH=path(DIR), otherwise PATH=path(FOO)

Options Controlling the kind of output files

-lis

This option creates a Signal file PATH/FOO_LIS.SIG containing the pretty printed FOO definition annotated with diagnosis messages **Deprecated**

-tra. Deprecated

This option creates a Signal process file PATH/FOO_XXX_TRA.SIG containing the result of

- XXX = BASIC elementary clock reduction
- XXX = POLY (BASIC) clock static resolution (maximal triangularisation)
- XXX = ENDO (POLY) endochronous parametrization (single master clock)
- XXX = BOOL (ENDO) event to Boolean clock transformation
- XXX = SCH equation sequencing

-tra[=[[bB][fF][iI][lL][mM][pP][sS][tT]]][--eE][--d=DIR][--t=OUTFILE] **NEW NOTATION ?**

This option is a *cumulative option*. A -tra option value cannot be repeated neither in the same **-tra** option nor in distinct **-tra** options. Two occurrences of the same letter is considered as a repetition even if one of them is a capital letter and the other one a small letter. Letter ordering is free and meaningless.
The sub-option **--t=OUTFILE** can only be present if the value of the **-tra** option is a singleton.
The following shortcuts are available:

- **-lis [--d=DIR][--t=OUTFILE]** stands for **-tra=l [--d=DIR][--t=OUTFILE]**
- **-tra [--d=DIR][--t=OUTFILE]** stands for **-tra=p [--d=DIR][--t=OUTFILE]**
- **-tra=:a[--d=DIR], tra=:A[--d=DIR]** respectively stands for
  - **-tra=liptbfms[--d=DIR]** and
  - **-tra=LIPTBFMS[--d=DIR]**
- **-ftra [--d=DIR][--t=OUTFILE]** stands for **-tra=b --E [--d=DIR][--t=OUTFILE]** when expressions are canonical.

This option creates a set of files. Each file contains a Signal term that is either a Signal process or a Signal module according to the content of path(FILE). All files but FOO_LIS.SIG (see below), or FILE if **--t=OUTFILE** is present, contain a syntactically correct Signal term. When a syntactic error occurred, FOO_LIS.SIG, or FILE if the -tra option value is "**l**" or "**L**", may contain an incorrect Signal term otherwise it contains a syntactically correct Signal term.

--[eE]

- When this sub-option of the -tra option is absent a Signal process,is structured in sub-process models showing the clock hierarchy.
- When **--e** is present all process models that are not extern or specified as "*unexpanded*" are expanded
- When **--E** is present all process models that are not extern are expanded.

--d=DIR: if **--d=DIR** is present the output directory for files related to the option values listed before DIR is PATH =path(DIR), otherwise the *output directory* PATH is the *implicit output directory*.

--t=OUTFILE

If **--t=OUTFILE** is present with value Z of the -tra option then the *output file* for Z value is TARGETFILE=path(PATH, OUTFILE), where PATH is the output directory as defined above. If the output file for Z value is not defined by **–t** sub-option then the output file for Z value is the *implicit output file* defined according to each option letter value **Z**. For all capital letter values Z but L, the implicit scheduling of FOO is displayed, otherwise implicit scheduling is not. The following files can be generated according to the value of the **-tra** option and provided that the transformations the result of which is displayed are required by *action options*

- Z=**l**, Z=**L**, the implicit output file is PATH/FOO_LIS.SIG; the output file contains the pretty printed FOO definition annotated with diagnosis messages. If the syntactic analysis of FOO fails to complete, PATH/FOO_LIS.SIG is generated only when Z=L
- Z=**i**, Z=**I**, the implicit output file is PATH/FOO_BASIC_TRA.SIG. The initial process is reduced to *Signal basic language*; Elementary clock reductions have been processed..
- *Z=**p**, Z=**P***, the implicit output file is PATH/FOO_POLY_TRA.SIG. The Signal basic term is reduced by static clock resolution to *clock reduced* Signal term
- *Z=**t**, Z=**T***, (t for tree) the implicit output file is PATH/FOO_ENDO_TRA.SIG. Each clock reduced Signal term is transformed by endochronous parametrization; to an *endochronous term:* each process has a unique master clock.
- *Z=**b**, Z=**B***, the implicit output file is PATH/FOO_BOOL_TRA.SIG. The endochronous Signal term is transformed by event to Boolean clock transformation. to a *Boolean clocked* Signal term: all clocks are Boolean clocks.
- *Z=**f**, Z=**F***, the implicit output file is PATH/FOO_FLAT_TRA.SIG. The *Boolean clocked* Signal term is transformed by Boolean clock completion to a *monochronous control* signal term: all Boolean clocks are made synchronous
- *Z=**m**, Z=**M***, the implicit output file is PATH/FOO_MONO_TRA.SIG *The FLAT process is transformed by signal completion; all signals are made synchronous*
- *Z=**s**, Z=**S***, the implicit output file is PATH/FOO_SCH_TRA.SIG. The *Boolean clocked* Signal term is transformed by equation total sequencing;

-sme[=[[bB][fF][iI][lL][mM][pP][sS][tT]]][--eE][--d=DIR][--t=OUTFILE] **NEW NOTATION**

This option is a *cumulative option* equivalent to **-tra**; the differences are listed below:

- output files are suffixed by ".sme" instead of ".SIG";
- output files contain SME EMF models representing Signal processes and/or Signal models;
- when **Z=l** or **Z=L**, the created file is **PATH/FOO.sme** instead of **PATH/FOO_LIS.sme**
- **-sme** is a short cut for **-sme=l**.

-ssme[=[[bB][fF][iI][lL][mM][pP][sS][tT]]][--eE][--d=DIR][--t=OUTFILE] **NEW NOTATION**

This option is a *cumulative option* equivalent to **-sme**; there are two differences:

- output files are suffixed by ".ssm" instead of ".sme"
- output files contain SSME EMF models instead of SME EMF

-z3z[--d=DIR][--t=OUTFILE]

This option creates a SIGALI file.

--***d=DIR*** defines PATH as it does in **-tra** option.

--t=OUTFILE
If **--t=OUTFILE** is present the *output file* is TARGETFILE=path(PATH, OUTFILE), where PATH is the output directory as defined above. If the output file is not defined then the output file is the *implicit output file* path(PATH,FOO_F3Abstraction.z3z)

-syndex[--d=DIR][--t=OUTFILE]

This option creates a SynDEx file.
The output directory is defined as it is in **-z3z** option with path(PATH,FOO.sdx) as implicit output file.

Graph Options
-dr UPWARD normalization of delayed signals
-crew=[b][d]
b: event normalization of Boolean clock expressions
d: when/default definitions are rewritten (operand reduced to a signal)
-depth=n tuning of Boolean reduction algorithm (for wizards only),
n is an integer (by default, n=5)
-su signal syntactic equivalence reduction
-s=n variable substitution
n is an integer
definition is substituted to each signal that has at most n occurrences
-ec[:i] creates a file FOO_EXCLUSIVE_CLOCKS.SIG containing the exclusive clocks pairs
-ec is equivalent to -ec:0
-ec:0 only the clocks of the signals are selected
-ec:1 the clocks of the signals are considered + the clocks referenced in the expressions
-ec:2 all the clocks are considered
-ds replaces the default definitions by partial definitions (default splitting)
-pdg replaces partial definitions by default expressions (partial definitions grouping)

Abstraction Options
-spec creates a file FOO_ABSTRACT.SIG containing the result of FOO interface abstraction

Partition Options
-clu code separation wrt input predecessors equivalence
-sbo Boolean nodes isolation
-sso state variables isolation

Code generation Options
Recall: the code is generated in the sub-directory FOO.
For simulation, go to this sub-directory, then you can use the genMake commands (see below).

-force forces the code generation by adding ``exceptions" for constraints
-check generates code for assertions
Options for optimization on delays: by default, optimization are applied. They can be avoided by the
following options:
-udo the delayed signal and the original one are in independent variables (the default is to manage
the delayed signal and the original one in the same memory when it is possible)
-umd the values are copied from the delayed variable to the original one (for non scalar

types,
the default is to manage the delayed signal and the original one in a circular area)

-c[:[i][m]]
c: creates FOO.c, FOO.h, and files of
FOO_ext.h, FOO_undef.log, FOO_type.h, that are needed in FOO.c
c:i creates FOO_io.c and files of
FOO_undef.log, FOO_type.h, that are needed in FOO_io.c
c:m creates FOO_main.c
c:im, c:mi creates files created by c:i and c:m
c creates files created by c: and c:im

where (ANSI C code)
FOO_externalsProc.h:
created if some external functions are referred to in FOO;
contains interface of those functions
FOO_externalsUNDEF_LIS.h:
created if some referred to types or constants are used
and not defined in FOO; in this case FOO_externalsUNDEF.h must be provided
FOO_types.h:
contains C types corresponding to SIGNAL types
FOO_body.c:
contains code associated with each step and the scheduler
FOO_body.h:
contains interface of functions generated in FOO_body.c
FOO_io.c:
contains input output functions associated with interface of FOO
FOO_main.c:
contains main C program
-c++[:[i][m]] same as -c[:[i][m]] with C++ code
-java [:[i][m]] mostly the same as -c[:[i][m]] with java code
FOO.java:
contains code associated with each step and the scheduler
FOO_io.java:
contains input output functions associated with interface of FOO
FOO_main.java:
contains main Java program
-threads (associated with C option) in this case a thread is generated for each cluster and the scheduling is managed using semaphores

-profiling (not fully implemented)
creates files for cost performance evaluation

Inter-format translations:
-bC (for Boolean Control) produces the SIGNAL code without "events"; the hierarchy of clocks is completed by Boolean inputs
-fl produces the SIGNAL code in which the hierarchy of clocks is reduced (flattened) to one level
-sD (serialized Data) produces the SIGNAL code in which the serializations are explicit

(*) Level formats:
* BASIC the control is explicit but unsolved
* POLY the control is a forest of trees of event clocks, each tree is hierarchic
* ENDO the control is a forest reduced to one tree of event clocks, the tree is hierarchic
* BOOL the control is a forest reduced to one tree of Boolean clocks, the tree is hierarchic
* FLAT the control is a forest reduced to one tree of Boolean clocks, the tree is flattened
* SCH BOOL level in which the scheduling (static or partial) is explicit

SEE ALSO
genMake -h

Notes:
- on Windows, to call the compiler with an option with the = character, it is necessary to use quotes
Examples: signal -tra P.SIG "-par=P.PAR" "-d=mydir"

Signal language levels

Signal Full language

Signal Basic language

Signal Kernel language

Signal canonical language

Signal Control language

Sketch of transformations

Elementary clock reduction

Static clock resolution

(maximal triangularisation)

Endochronous parametrization

Event to Boolean clock transformation

Boolean clock completion

Signal completion

The signal completion of a signal S is its signal completion up to H, where H is the context clock of the process that contains the declaration of S.

The completion of a signal S up to a clock H that contains ^S results in a signal S' synchronous with

H.

- when S is defined, S' is equal to S
- otherwise, when S' is defined
  - when S is declared with a default value V, V is the value of S',
  - otherwise V is the previous value of S'.

Remark*: since in the signal completion P' of a process P each signal S holds the same value in P and in P' when it is defined in P, giving other completion mode seem useless.*

Some Signal concepts

Module

A module contains a list of *private* and *public*" processes. The module can be used as a source code provider, or it can be used as a set of compiled processes.

A compiled public process cannot have static formal parameters. When a process IP1 in a module that has static formal parameters f1...fn is designed for being used as a compiled process in a library, a process P1 without static parameters must be defined to instantiate IP1 with actual parameters declared in the embedding module. The *use* clause allows to compile IP1 in various context, with different actual parameters.

Context clock

Context clock of a process

The context clock of a process that is not a sub process of a nesting process is the upper bound of all signals declared in this process