POLYCHRONY TOOLSET

Espresso Team

June 7, 2011

Contents

The POLYCHRONY TOOLSET, is an Open Source development environment for critical/embedded systems. It is based on SIGNAL, a real-time polychronous dataflow language. It provides a unified model-driven environment to perform design exploration by using top-down and bottom-up design methodologies formally supported by design model transformations from specification to implementation and from synchrony to asynchrony. It can be included in heterogeneous design systems with various input formalisms and output languages.

SIGNAL language

SIGNAL is a specification and programing language for critical/real-time embedded applications. The main features of the SIGNAL languages are synchronized flows (flows + synchronization) and processes: a process is (recursively) a set of equations over synchronized flows describing both data and control.

The SIGNAL formal model provides the capability to describe systems with several clocks (polychronous systems) as relational specifications. Relations are mandatory to write partial specifications, to specify non-deterministic devices (for instance a non-deterministic bus), and to abstract the behavior of external processes (for instance an unsafe car driver). Using SIGNAL allows to specify an application, to design an architecture, to refine detailed components downto RTOS or hardware description. The SIGNAL model supports a design methodology which goes from specification to implementation, from abstraction to concretization, from synchrony to asynchrony. More details can be found in a short introduction to SIGNAL language
http://www.irisa.fr/espresso/Polychrony/introToSignal.php.

POLYCHRONY TOOLSET

The POLYCHRONY TOOLSET provides a formal framework:

- to validate a design at different levels, by the way of formal verification and/or simulation
- to refine descriptions in a top-down approach,
- to abstract properties needed for black-box composition,
- to assemble heterogeneous predefined components (bottom-up with COTS).
- to generate executable code for various architectures

The POLYCHRONY TOOLSET, contains three main components and an experimental interface to GNU Compiler Collection:

- The SIGNAL TOOLBOX, a batch compiler for the SIGNAL language, and a structured API that provides a set of program transformations. The SIGNAL TOOLBOX can be installed without the other components.
- The SIGNAL GUI, a Graphical User Interface to the SIGNAL TOOLBOX (editor + interactive access to compiling functionalities). SIGNAL GUI requires the SIGNAL TOOLBOX or an other component that redefines the SIGNAL TOOLBOX Syn and Sem APsI.
- The SME PLATFORM, a front-end to the SIGNAL TOOLBOX in the ECLIPSE environment (`http://www.eclipse.org`). SME PLATFORM requires the SIGNAL TOOLBOX or an other component that redefines the SIGNAL TOOLBOX Syn and Sem APIs.
- GCCST, a back-end to GNU Compiler Collection that generates SIGNAL programs (not available for download),

The POLYCHRONY TOOLSET, also provides:

- libraries of SIGNAL programs,
- a set of SIGNAL programs examples,
- user oriented and implementation documentations,
- facilities to generate new versions.

The POLYCHRONY TOOLSET is downloadable on the following web sites:

- The POLYCHRONY TOOLSET public web site: `http://www.irisa.fr/espresso/Polychrony`. This site, intended for users, contains downloadable executable versions of the software for differents platforms, user documentations, examples, etc.
- The INRIAGForge: `https://gforge.inria.fr`. This site, intended for developers, contains the whole sources of the environment and their documentation.
- The TOPCASED distribution site: `http://www.topcased.org`. This site provides the current reference version of SME PLATFORM, including the executable of the SIGNAL TOOLBOX.

The POLYCHRONY TOOLSET currently runs on Linux, MacOS, Windows systems; for more details see **Download** on the POLYCHRONY TOOLSET public web site. The POLYCHRONY TOOLSET public web site also provides user and implementation **Documentation**, SIGNAL**Libraries**, **Examples** and scientific **Publications**. Examples can be freely used. Libraries are distributed under GPL V2 licence.

1 POLYCHRONY TOOLSET structure

The global architecture of the POLYCHRONY TOOLSET is shown in figure 1. The POLYCHRONY TOOLSET contains the main components described below.

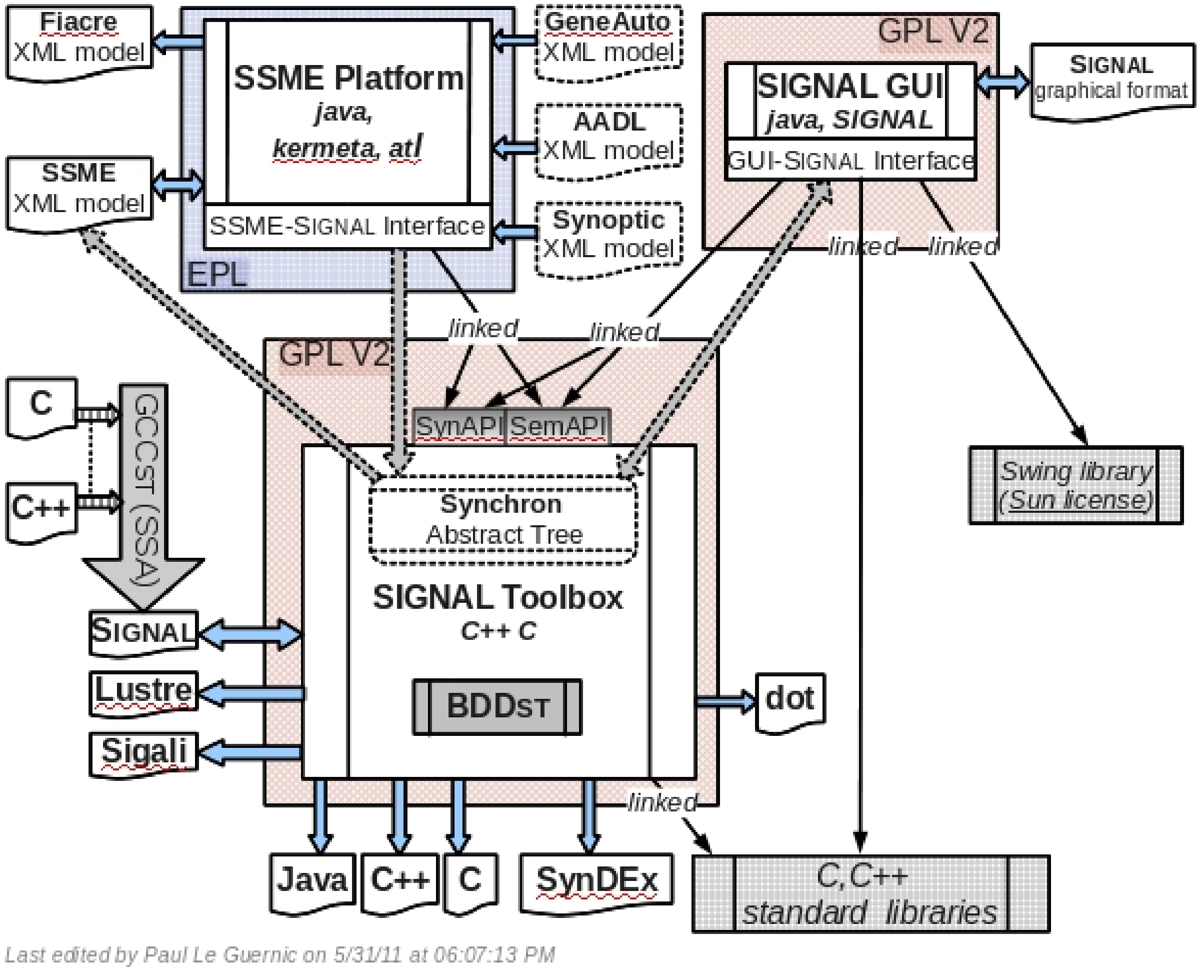


Figure 1: The Polychrony toolset high level architecture

- GCCST is an interface, developed in Espresso team, that allows programs written in C, C++ compiled by GNU Compiler Collection (`http://gcc.gnu.org/`) to be translated into SIGNAL programs. SSA (`http://gcc.gnu.org/onlinedocs/gccint/SSA.html`) is used as the intermediate format. A subset of SSA features is accepted (currently pointer free SSA programs). GCCST is a modified version of GNU Compiler Collection that is not yet distributed.
- The SIGNAL TOOLBOX, is written in C and C++; the SIGNAL TOOLBOX APIs are C++ classes. The SIGNAL TOOLBOX is distributed under GPL V2 licence. Those APIs provide ways to build SIGNAL abstract trees (via SynAPI) and to apply high level transformations over SIGNAL programs (via SemAPI); using those APIs makes SIGNAL an intermediate language (a semantic target).

The SIGNAL TOOLBOX accepts input programs written in SIGNAL and, via GCCST, a subset of each language compiled in SSA by the GNU Compiler Collection.
The SIGNAL TOOLBOX generates the result of applied transformations (including identity) in the following formats:

  - SIGNAL programs or SME models, after any transformation,
  - LUSTRE nodes, after any transformation that results in a *Lustré* SIGNAL process [1].
  - SIGALI systems for verification purpose, after an ultimate transformation that extracts the boolean abstraction of its argument,
  - java, C, C++ programs the "executable" target languages for *executable* SIGNAL *processes*,
  - SYNDEX for code distribution purpose.
  - Graphviz dot representation of a DCGraph.

  The SIGNAL TOOLBOX contains BDDST, a C program derived (by Espresso team members) from the Berkeley BDD package. It uses standard standard C, C++ libraries that are not provided in the SIGNAL TOOLBOX distribution.

- The SIGNAL GUI is a Graphical User Interface to the SIGNAL TOOLBOX (editor + interactive access to compiling functionalities). The SIGNAL GUI, is written in java and SIGNAL. It is dynamically linked to the SIGNAL TOOLBOX via a java-C++ interface. SIGNAL GUI requires a Swing library. An external format is provided. The SIGNAL GUI is distributed under GPL V2 licence.

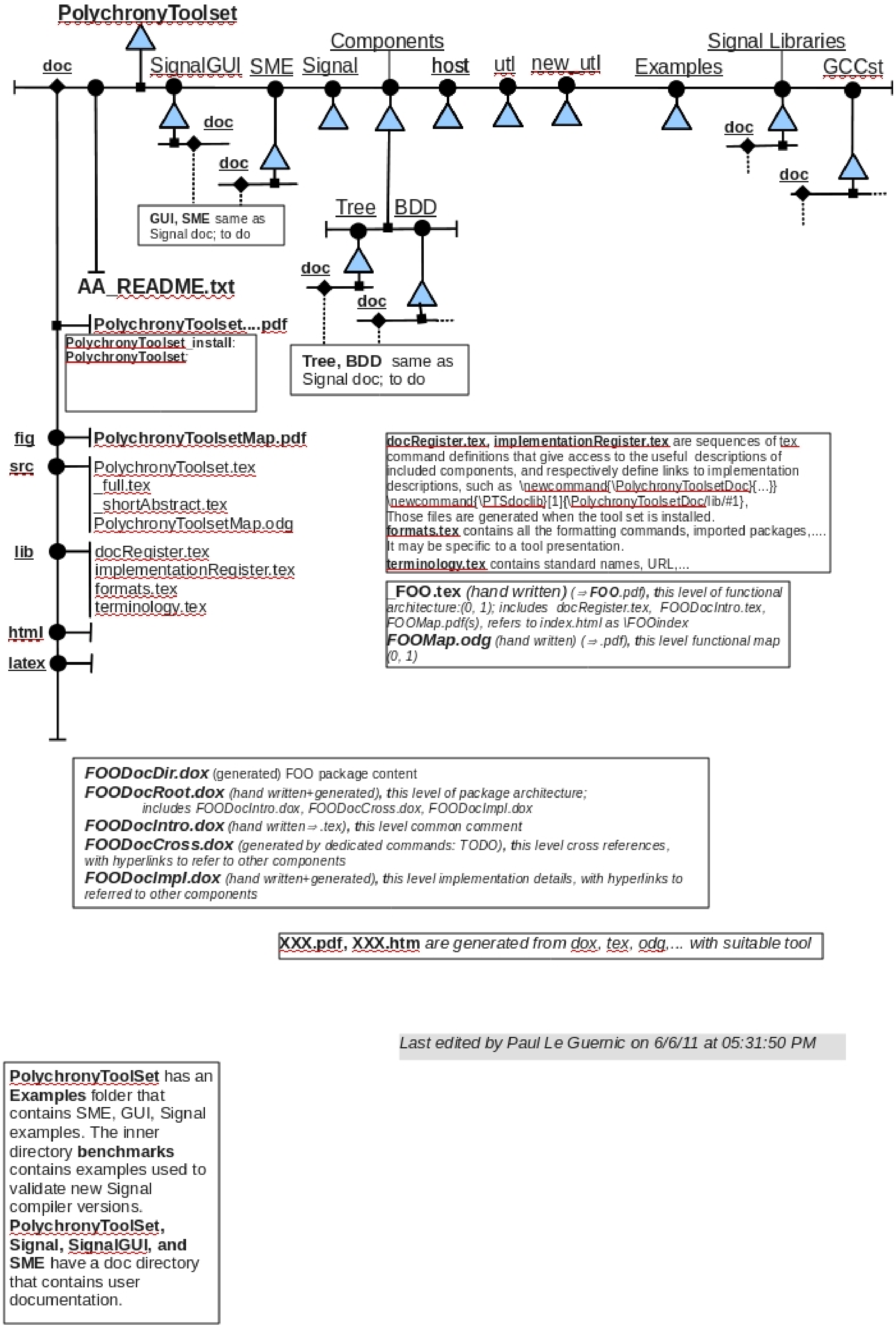
- The SME PLATFORM consists of a set of ECLIPSE plug-ins which rely on the ECLIPSE Modeling Framework (EMF). The platform is built over SME(SME is an acronym for SIGNAL Metamodel under ECLIPSE), a metamodel of the SIGNALlanguage extended with mode automata concepts. The SME PLATFORM is written in Java, Kermeta and ATL.

  A SME model can be translated to a FIACRE model for formal verification purpose. The SME PLATFORM is distributed under EPL licence. Using SME PLATFORM allows to built the SIGNAL description of an embedded system consisting of an application, described in GENEAUTO, distributed on an architecture described in AADL. The resulting SIGNAL program is used to co-simulate, co-verify,... the full (software and hardware) system. SYNOPTIC is a metamodel designed in the Sapcify project. SYNOPTIC, GENEAUTO, AADL translators are prototypes that do not belong to the SME distribution.

These components are structured int the installation directory as shown in figure 2

---

[1] A *Lustré* SIGNAL process is an *endochronous process*, the root of which is the clock of an external signal.
An *endochronous process* is a process the control of which is structured following a tree of boolean definitions (giving the clocks)



GUI, SME same as Signal doc; to do

Tree, BDD same as Signal doc; to do

docRegister.tex, **implementationRegister.tex** are sequences of tex command definitions that give access to the useful descriptions of included components, and respectively define links to implementation descriptions, such as \newcommand{\PolychronyToolsetDoc}{...}} \newcommand{\PTSdoclib}[1]{\PolychronyToolsetDoc/lib/#1},
Those files are generated when the tool set is installed.
formats.tex contains all the formatting commands, imported packages,....
It may be specific to a tool presentation.
terminology.tex contains standard names, URL,...

_FOO.tex *(hand written) (⇒ **FOO**.pdf), this level of functional architecture:(0, 1); includes docRegister.tex, FOODocIntro.tex, FOOMap.pdf(s), refers to index.html as \FOOindex*
FOOMap.odg *(hand written) (⇒ .pdf), this level functional map (0, 1)*

FOODocDir.dox (generated) FOO package content
FOODocRoot.dox *(hand written+generated), this level of package architecture;*
includes FOODocIntro.dox, FOODocCross.dox, FOODocImpl.dox
FOODocIntro.dox *(hand written ⇒ .tex), this level common comment*
FOODocCross.dox *(generated by dedicated commands: TODO), this level cross references, with hyperlinks to refer to other components*
FOODocImpl.dox *(hand written+generated), this level implementation details, with hyperlinks to referred to other components*

XXX.pdf, **XXX.htm** *are generated from dox, tex, odg,... with suitable tool*

Last edited by Paul Le Guernic on 6/6/11 at 05:31:50 PM

PolychronyToolSet has an **Examples** folder that contains SME, GUI, Signal examples. The inner directory **benchmarks** contains examples used to validate new Signal compiler versions. **PolychronyToolSet, Signal, SignalGUI, and SME** have a doc directory that contains user documentation.

Figure 2: The Polychrony toolset tree organization

2 GCCST

map . . . to be completed . . .

abstract . . . to be completed . . .

component short abstracts . . . to be completed . . .

component abstracts . . . to be completed . . .

3 SME PLATFORM

map . . . to be completed . . .

The SME PLATFORM is composed of several plug-ins under ECLIPSE giving access to:

- a reflexive editor: a tree view allowing to manipulate models conform to the SME metamodel;
- a graphical modeler based on the TOPCASED modeling facilities;
- a reflexive editor and an ECLIPSE view to create compilation scenarios;
- a direct connection to the POLYCHRONY TOOLSET services (compilation, formal verification, etc);
- a documentation and model examples.
- GENEAUTO, SYNOPTIC models.
- AADL models.
- a generator of FIACRE models for formal verification purpose.

The SME PLATFORM distribution includes the SIGNAL TOOLBOX linkable code.

component short abstracts . . . to be completed . . .

component abstracts . . . to be completed . . .

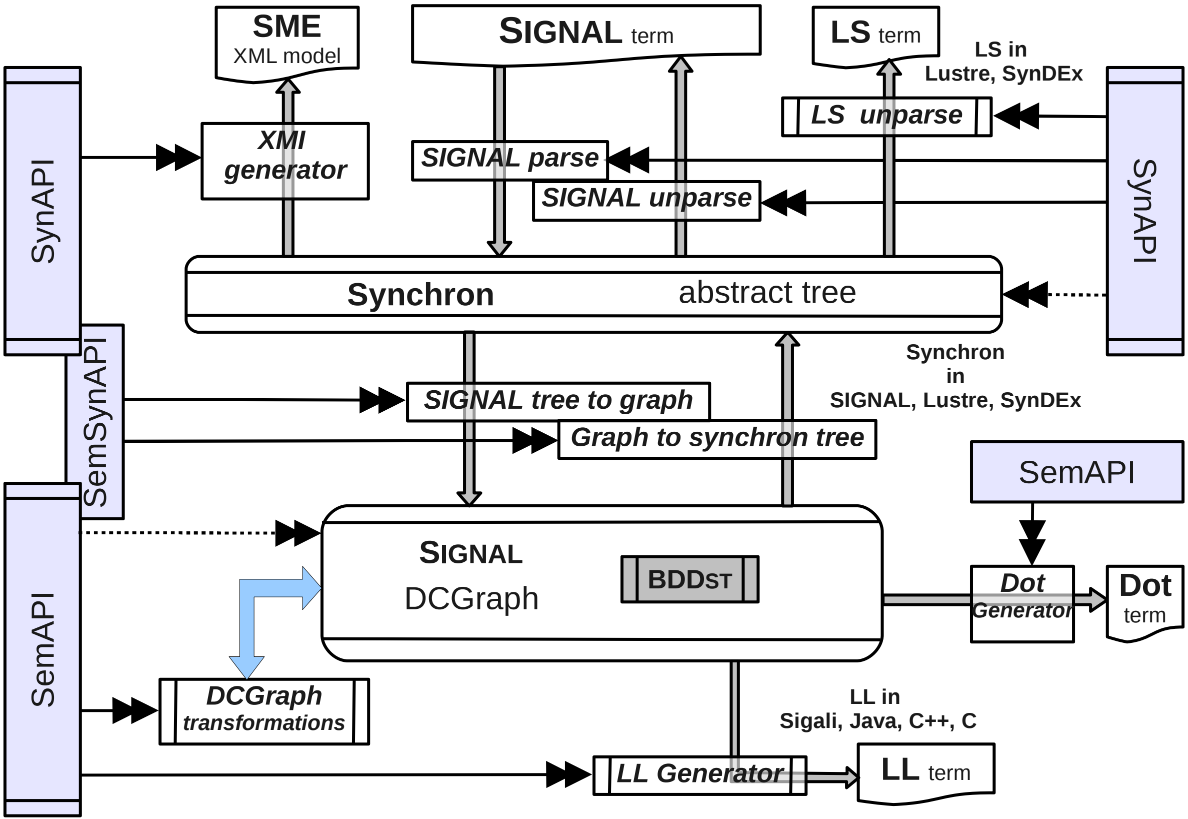
4 SIGNAL GUI

map . . . to be completed . . .

component short abstracts . . . to be completed . . .

component abstracts . . . to be completed . . .

5 SIGNAL TOOLBOX



Last edited by Paul Le Guernic on 11/26/10 at 12:26:40 PM

Figure 3: The Signal toolbox high level architecture

SignalDocIntro . . . to be completed . . .

The Signal toolbox contains the following components that are provided via APIs:

- the **Synchron abstract tree**, (in `Signal/package/SignalTreeManager`)
- the Signal **DCGraph**, (in `GraphVM/Graphdef`)
- the BDDst included in Signal **DCGraph**, (in `Components/BDD`) and (in `GraphVM/Graphdef/ClockManager/BDD`)
- the Signal **parser**, (in `Signal/package/Parsing`)
- the Signal **unparser**, (in `Signal/package/SignalTreeManager`)
- the **XMI generator**, (in `Signal/package/XMIGenerator`)
- the Lustre and SynDEx unparsers, (in `GraphVM/ExportTools/XGenerator`)
- the Signal**Tree to** Signal **DCGraph** translator, (in `Signal/package/Tree2Kernel`) and (in `Signal/package/Kernel2Graph`)
- the Signal **DCGraph to Synchron Tree** translator, (in `GraphVM/Printers/...`)

- a set of SIGNAL **DCGraph to** SIGNAL **DCGraph transformation**s, (in `GraphVM/...`)
- the dot (in `GraphVM/Printers`) code generator.
- the SIGALI, (in `GraphVM/ExportTools/SigaliGenerator`), JAVA, C++, C (in `GraphVM/CodeGenerators/XGenerator`) (in `GraphVM/Printers`) code generators.
- the host interface (not represented in figure figure 3) that makes the rest of the code OS/HW independent

component short abstracts . . . to be completed . . .

6 SIGNAL libraries

Libraries are provided in the POLYCHRONY TOOLSET distribution. There are:

- the following libraries of Signal programs
  - Apex contains the Signal definition of the Apex ARINC 653 library plus the associated C++ code.
  - Graphical contains the Signal definition of a graphical library.
  - Profiling contains the Signal definition of the morphism table and libraries used for profiling.
  - Sigali contains the Signal intrinsics for Sigali.
  - Std contains the standard Signal intrinsics (math, conversion, i/o, affine clocks).
- libraries used by downstream tools or programs (generated code...).
  - C contains a library provided for user program execution when generated code is C or C++.
  - Java contains a library provided for user program execution when generated code is Java.
  - Syndex contains SynDEx libraries used by programs generated from Signal2Syndex translation.
  - Utl contains data for the genMake tool (makefile generator for simulation).

7 SIGNAL Examples

Under construction