SCiPX: Developer's Guide

Jean-François Le Tallec

Abstract

The purpose of this document is to describe the internals of the SCiPX tool, to give some advices and prerequisites to be able to modify/extend it.

1 Introduction

SCiPX stands for SystemC to iP-Xact extraction tool. It takes as an input a SystemC-TLM [IEE05, Ayn09] design and produces the architectural view of this design in the IP-Xact [IPX] format linked to IP-Xact components definitions. It can also be seen as a partial IP-Xact packager for components. To do so, SCiPX relies on two different approaches. The first one uses the source code analysis tool Doxygen to produce the footprint of a SystemC components in the xml format. The second one uses a slightly modified version of SystemC 2.2 to stop the execution of a SystemC program, built from the SystemC design, exactly when the elaboration phase of the design is finished. Then, SCiPX merges the different informations in order to produce an IP-Xact model of the design and the different Components models used by the design. Further information about inner techniques can be found in the first paper about SCiPX [LTDS+11] or in the French thesis [LT12].

2 The tool's compilation

SCiPX tool relies on different already existing tool/library to be able to process the whole analysis. The installation is fully handled thanks to bash script and is detailled in the "Getting Started" document. However, the developer should want to be aware of the different installation steps. The main script *install.sh* invokes the *install-lib.sh* script (available in the script directory). The installation script extracts tar-ball files available in the external directory then compiles and installs them in the "locallib" directory when needed. The Makefile.root file is created during the process. In the end, this file contains path to all needed libraries. If the installation fails at some point, the file can be corrupted (lack of specific path). If everything goes well during the installation, Makefile.root should contains the following Environment variables: ROOT_DIR, SYSTEMCINST, DOXYGENINST, TLM2INST, SCIPXINST, XSDCXXINST. All these paths will be sourced by the standard "Makefile.global" (in the example

directory) to be able to compile a SystemC program. SYSTEMCINST is the path to the slightly modified SystemC 2.2 library. Modifications are detailed in the section 3. DOXYGENINST is the path to a specific version of Doxygen tool. It will be used to perform static source code analysis. For that purpose, the Doxygen tool is compiled plus a specific add-on intended to ease the exploitation of Doxygen output. TLM2INST is the path to the template tlm 2.0 library, SCIPXINST is the path to the SCiPX source code and XSDCXXINST is xml to C++ parser/serializer library. The figure 1 highlights dependencies between both of them.

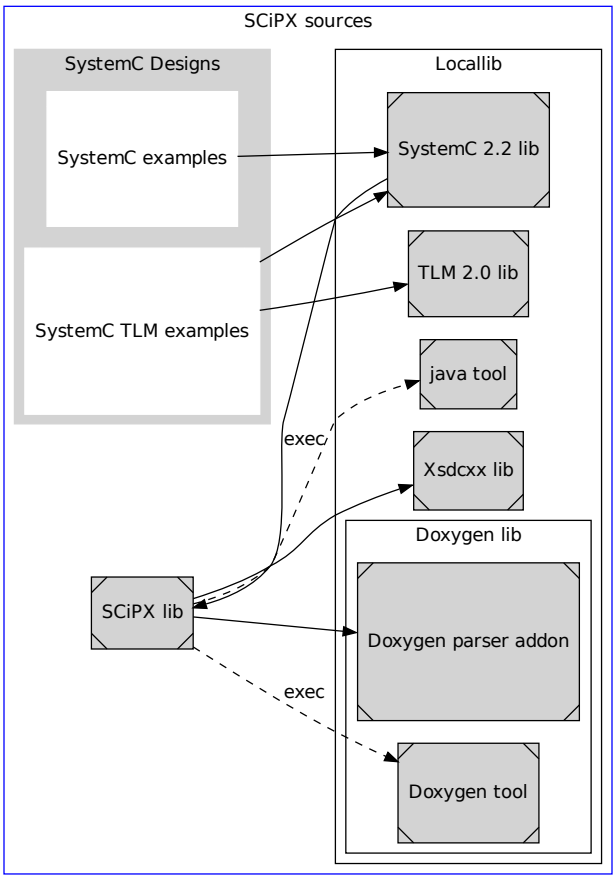

Figure 1: Libs dependencies in SCiPX

3 The modified SystemC 2.2 version

As introduced in section 1, SCiPX links every SystemC program to a slightly modified version of SystemC. In this section, we describe each modification and the motivation to do such a modification.

The first modification is not so important but is needed to be able to compile SystemC library with not-so-old gcc versions.

```
#include "string.h"
#include "cstdlib"
```

have been added to the file *$(SystemC Directory)/src/sysc/utils/sc_utils_ids.cpp*.

All the following modifications intend to ease the access to specific structures during the execution of a SystemC program. The most important one is to be able to stop the execution after the elaboration phase. For that purpose, an external call to the main SCiPX function has been added in the following file : *$(SystemC Directory)/src/sysc/kernel/sc_simcontext.h/cpp*. This function call intends to start the SCiPX analysis, feeding the dynamic analysis part with a pointer to the internal SystemC structure *sc_simcontext*. This specific structure owns references to all objects built during the elaboration phase. Another step to ease the SCiPX analysis was to add specific methods to be able to access private members (i.e. modules, ports, exports and channels). Same kind of methods was added to the *sc_module*, *sc_port*, *sc_export* and *sc_prim_channel* classes. To sum up, all these modifications do not change class's structures and only intends to ease the navigation throw the whole design at runtime.

4 The SCiPX Internals

4.1 scipx_callback.cpp

The main SCiPX executed function is the *scipx_callback(sc_simcontext*)* function. Such a call appears once the whole design has been instantiated (i.e. the design remains the same till the end of the execution). It will process the full analysis *via* the *scipxAnalysis* object. Once done, it returns to the original behavior of the SystemC program if the user choose to start the simulation.

4.2 scipxAnalysis.h/cpp

The *scipxAnalysis* object is responsible for invoking successively each step of the SCiPX analysis. The first step is to collect all module's types used in the design. This step is done by navigating into the *sc_simcontext* object and is done by an *scipxModuleTypeHandler* object. This list of types will feed the static analysis to be able to retrieve public member names of each component. The next step is to launch the Doxygen tool on the SystemC source code with respect to a configuration file which is located in the *external* directory. Once done, the analysis result is processed by the *scipxDoxyFileParsing* object to extract static information of found types. Thanks to these information, the *scipxGenerateOffsetCode* object

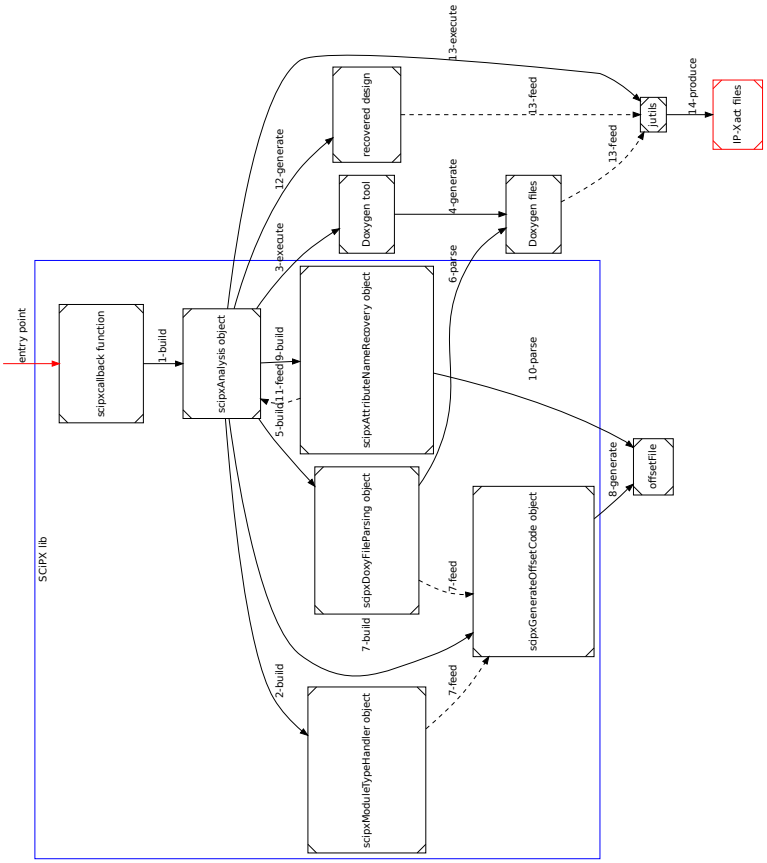

Figure 2: SCiPX internal dependencies

produces a C++ file in order to generate the offset of each public member of each classes. So the next step is compiling and executing this code to generate the mapping table of each classes/public members/offset. Throw this step, the offsetFile is created. The offsetFile is a formatted text file needed to feed the *scipxAttributeNameRecovery* object. This object is an interface to the offset mapping. It is then used by the master function *buildsystemStruct* of the *scipxAnalysis* object to produce merge static and dynamic informations. The whole produced design is then serialized and lastly processed by the *jutils* to finally produced the IP-Xact design and components files. The whole flow is depicted is the figure 2

4.3 scipxDoxyFileParsing.h/cpp

The scipxDoxyFileParsing class is the key point to navigate throw static information. As explain in section 4.2, the first important thing this class is in charge is to retrieve public member names in order to easily identify port/export names. The second is to provide an interface to the static information in order to retrieve, in a generic way, from which closest mother class is derived a specific port/export and then to be able to build different set of port/export (i.e. identify directed flows, for more explanation cf. [LT12]).

4.4 scipxGenerateOffsetCode.h/cpp, scipxAttributeNameRecovery.h/cpp

This class intends to generate generic C++ plain text with respect to the components' list of the analyzed design and the public members' list of each component. It then produces the *fakemain.cpp* file which is a concatenation of the original *main.cpp* file plus a newly generated *main* function. The concatenation is explain by the fact that we are going to ask each class the offset of all their public attribute. It then keeps all needed dependencies/includes needed without interfering as the *sc_main* is never called in *fakemain.cpp* and the default *main* function became the one we gave instead of the SystemC one. This *main* function, once compiled/executed, generates a plain text *offsetFile*. The *scipxAttributeNameRecovery* class intends to parse the generated *offsetFile*. It creates a *scipxClassDescriptor* object for each class to ease the mapping interrogation. This two classes are strongly connected. As long as a developer want to modify the *fakemain.cpp* generation, a change to *scipxAttributeNameRecovery* class could be necessary to stay compatible. In other words, if the *offsetFile* format is changed, *scipxAttributeNameRecovery* will not be able to parse it anymore.

4.5 systemCXMI.h/cpp, XMI.h/cpp

These two classes have been automatically generated thanks to xsdcxx tool. Thus, it must not be changed.

References

[Ayn09] Aynsley, John. OSCI TLM 2.0 Language Reference Manual, July 2009.

[IEE05] IEEE Standards Association. *Open SystemC Language Reference Manual*. Open SystemC Initiative, 2005. IEEE Std. 1666–2005.

[IPX] IP-Xact standard IEEE 1685. www.accellera.org/downloads/ieee.

[LT12] Jean-François Le Tallec. *Model extraction for System On Chip design*. PhD thesis, Université de Nice Sophia-Antipolis, France, 2012.

[LTDS+11] Jean-François Le Tallec, Julien DeAntoni, Robert Simone, Benoit Ferrero, Mallet Frdric, and Laurent Maillet-Contoz. Combining SystemC, IP-Xact and UML-Marte in model-based SoC design. In *Proc. of the 2011 Workshop on Model Based Engineering for Embedded Systems Design*, MBED'2011, Grenoble, France, March 2011.