

An Open-Source Tool for SystemC to Verilog Automatic Translation

Javier Castillo, Pablo Huerta, José Ignacio Martínez

Universidad Rey Juan Carlos, Madrid, Spain,
javier.castillo@urjc.es,
pablo.huerta@urjc.es,
joseignacio.martinez@urjc.es,
<http://www.escet.urjc.es/~jmartine>

Abstract. As the complexity of electronic systems increases, new ways for describing these systems are proposed. One actual trend involves the use of system level languages that allows the description of the whole system in a higher abstraction level. This type of methodology helps a designer to obtain an appropriate Hw-Sw partition, where the Sw is compiled to the target platform and the Hw is refined to bring it down to a lower level of abstraction in order to be synthesized. This last step usually requires the use of a translation tool that from a description of the system in a system level modeling language, converts it to an equivalent one in a standard Hardware Description Language, usually Verilog or VHDL. This works presents a tool that from a SystemC RTL description generates its equivalent Verilog code ready to be synthesized by any standard Verilog Synthesis Tool.

1 Introduction

The increasing complexity of the electronic systems has made necessary the exploration of new solutions in order to reduce its development time.

One of this solutions is to use new description languages [1][2][3][4] which allow the designer to describe the system in a higher level of abstraction.

From this high level description of the system, a tool must provide a flow to reach the final silicon implementation. Following a traditional hardware-software codesign flow [5] (Fig.1), the system level description is profiled, and an appropriate hardware-software partition is proposed. Then the software has to be compiled to the targeted microprocessor. The hardware high level implementation can be directly converted into a working hardware using a proprietary synthesizer, like Handel-C or CatapultC. Another approach consist of rewriting it to obtain a lower and more detailed level of abstraction appropriate for hardware synthesis tools. This step is commonly made by hand with the problems it represents (time cost, prone to errors ...).

This work presents an open-source tool that taking as input a hardware module described using a high level description language, SystemC, gives as a result an equivalent description in Verilog. This Verilog description can be synthesized using any standard RT synthesis tools.

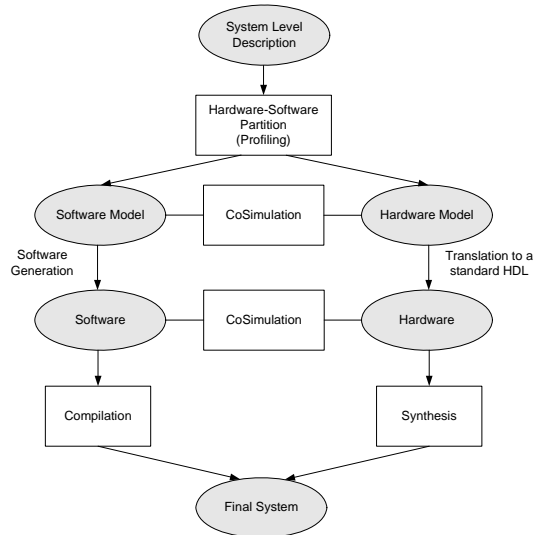


Fig. 1 Traditional Hardware-Software CoDesign Flow

2 SystemC

2.1 SystemC Overview

Nowadays hardware is usually described using HDLs such as VHDL and Verilog. However, software and system designers develop their code in C/C++. Hardware/software codesign becomes a very hard task due to the use of different languages at each abstraction level, or even in the same level (IP exchange). In this context, a single language that can be used in all the design stages is needed.

SystemC is a library of classes for C++ and a simulation kernel that provides all the features needed to describe a system in all its abstraction levels and a reference platform for IP exchange. SystemC also provides a library called the SystemC Verification Standard (SCV) [6]. This library provides classes and methods to build a verification methodology called Transaction Model Style (TLM) based on the use of transactors [7]. This verification methodology can be used with designs made with other HDLs. Many EDA vendors provide tools that allow SystemC to be mixed with different languages.

2.2 SystemC Design Flow

Estimations say that the main bottleneck of a traditional design flow is the verification stage [8]. It is considered that at least 60% of the design effort is made in the verification stage. Traditional verification schemes (Fig. 2) also have the problem that the System Level

The translator inputs are two files, called header and implementation that describe a module. Only one module can be declared in each header file.

The translation process is divided in two steps. In the first step the implementation file is parsed. In the second step the information from the first step is combined with the information obtained by parsing the header file to get the final Verilog translation.

3.1 Implementation File Parsing

In this first step the translator takes the module implementation file and uses the translatable RT subset defined in [13]. The implementation file contains the implementation of the module and all the auxiliary functions required. The tool converts all the SystemC code and functions inside the processes into Verilog. The local variables inside the modules and their references are renamed, adding to the names the name of the process in which they are declared. This is because Verilog doesn't support local variables inside the process, whilst in SystemC is not unusual to have different processes using local variables with the same name, for instance, `aux`

Another task that is carried out in this first step is to convert the structures data type into something Verilog can understand because it is a data type not supported. Structures are translated into local variables changing the name of each element of the structure for the same name plus the structure's own name. All the references to the structures inside the process code are also changed to the new names.

Translating SystemC assignments to Verilog can be difficult because SystemC can not distinguish between signal blocking and non-blocking assignments. In SystemC the only way to assign a value to a `sc_signal` is to use the `.write()` method of the `sc_signal` class. The first thing to analyze is that if this method produces a blocking or a non-blocking assignment to the signal. A simple test to understand the behavior of the `.write()` method was designed:

```
sc_signal < sc_uint < 8 > >a,b,c;

void test::test ()
{
  if (reset.read ())
  {
    a.write(1);
    b.write(2);
    c.write(3);
  }
  else
  {
    a.write(c.read());
    b.write(a.read());
  }
}}
```

Running this simple piece of code shows that `a` is 3 and `b` is 1 after the first cycle signal. That means that the `.write()` method has a non-blocking behavior and all assignments to a `sc_signal` in the Verilog translation should use the non-blocking assignment operator (`<=`).

- Instantiated Modules
 - o Port Bindings
- Enumerated Types
 - o Enumerated members
- Auxiliary Functions
 - o Inputs and outputs for each function

When the parser has finished collecting information, the generation of the Verilog file can start.

After writing the module name, all the ports with its associated types have to be written. The translator has to decide for each output if it has an associated register or not. If the output is directly connected to an instance, the output should not have an associated register. If not, this output will be written to from a process and should have the associated register. This is very easy to analyze just looking at the instances list and the bindings of each instance.

After the ports are declared the enumerated data types have to be translated. This feature was incorporated at the request of many users and has been included in version 0.2 to ease the design of state machines. Verilog doesn't allow enumerated data types but as in the case of structures it is not difficult to mimic its behavior, using parameters in this case. For example the SystemC enumerated variable:

```
enum {S0,S1,S2,S3} state;
```

is translated into:

```
parameter S3=0, S2=1, S1=2, S0=3;
reg [1:0] state;
```

The translator automatically calculates the register length to fit the number of elements of the enumerated data type. It is necessary to be careful with enumerated types because using the same element name in more than one enumerated type will produce an error.

The signals used by the module are declared next. It is compulsory to decide if the SystemC *sc_signal* should be translated into a Verilog *wire* or *reg* signal type. The way of doing this is to look at the file_writes.sc2v file generated during the implementation file parsing. If the signal has been written inside any process, the signals will be translated into a *reg* type. If not, the signal was declared to be connected to an instance and it is translated into a *wire*.

After finishing with all the signals and ports, the next step is to write the hierarchy instantiations. This step requires to read the list with all the information about the instances and its bindings and write the equivalent code to the Verilog file.

Next, the file_defines.sc2v which has all the preprocessor directives (defines, conditional compilation directives and macros) is read and written to the translation.

Finally the functions and processes code are written down in the translation from the files generated during the implementation file parsing, concluding with a Verilog equivalent file ready to be simulated and/or synthesized.

References

1. OSCI : SystemC 2.1 : <http://www.systemc.org>
2. Celoxica : Handel-C Language Reference Manual : 2005
3. Xilinx : Xilinx System Generator for DSP
4. Pellerin D., Thibault S. : Practical FPGA Programming in C. Prentice Hall, April 2002
5. Chiodo M., Giusto P., Hsieh H., Jurecska A., Lavagno L., Sangiovanni-Vincentelli A. : Hardware-Software Codesign of Embedded Systems. In IEEE Micro, August 1994, 26-36
6. OSCI : SystemC Verification Standard Specification : 2002
7. Yuri E., Tatsuda J., Khan N., Dietrich C.: Transaction-based simulation using SystemC/SCV, <http://www.eetasia.com/>, March 2005
8. The International Technology Roadmap For Semiconductors, 2001 Edition.
<http://public.itrs.net/Files/2001ITRS/home.htm>
9. Castillo J., Huerta P., Martinez J.I. : SystemC Design Flow for a DES/AES CryptoProcessor, Proceedings of WSEAS'04. Athens, 2004
10. Forte : Cynthesizer Datasheet: http://www.forteds.com/products/cynthesizer_datasheet.pdf, 2004
11. Synopsys : Cocentric System Studio Datasheet :
http://www.synopsys.com/products/cocentric_studio/cocentric_studio.html
12. Levine J., Mason T., Brown D. : lex & yacc, Second Edition. Oreilly, October 1992
13. Synopsys : Describing Synthesizable RTL in SystemC, 2002
14. www.gdhsw.escet.urjc.es