Visualization of SystemC Designs

Christian Genz
University of Bremen
28359 Bremen, Germany
genz@informatik.uni-bremen.de

Rolf Drechsler
University of Bremen
28359 Bremen, Germany
drechsle@informatik.uni-bremen.de

Gerhard Angst
Concept Engineering GmbH
79111 Freiburg, Germany
gerhard@concept.de

Lothar Linhard
Concept Engineering GmbH
79111 Freiburg, Germany
lothar@concept.de

***Abstract*— The open source library SystemC aids the composition of system level designs. The library offers a wide variety of datatypes and a simulation kernel. Both components help to describe VLSI designs very close to the RT level or at the more abstract system level. Since SystemC is based on C++, it allows the integration of several techniques into the model which do not contribute to the model itself, e.g. for verification, system exploration or debugging. But especially for debugging it is helpful to hide all these additional elements.**

This work addresses the problem of SystemC visualization for debugging backends. Besides displaying a system's behavior and its structure, our approach also excludes non-relevant data which is part of the system description, but does not define its behavior or its structure.

I. Introduction

Since Moore's law is still valid, modern circuits and their specifications tend to grow in size and complexity. One reason for this is the increasing number of integrated transistors, caused by a growing functionality. Another reason is the growing integration of software implementations in hardware architectures. It is not surprising that especially embedded systems as in PDA's or cell phones benefit from (re)using large software parts. To handle this complexity, industry and academia often use a homogeneous model for the development process. The system level language SystemC supports such homogeneous system models by offering hardware description capabilities and an interface to arbitrary software algorithms.

But being embedded into C++, SystemC hardly restricts the way a system is implemented. SystemC datatypes and their derivates have a precise meaning for such implementations. E.g. there is no mechanism to separate an integrated testbench from the behavior of the model by declaration. As a result the complexity of a design increases again. Especially in groups where many developers interact, these designs have to be understandable to be shared.

In this work[1] we present a visualization approach for SystemC, based on the analysis tool from [1]. The implementation is made up of two tools that interact with each other. The first one is *ViSyC*, a statical analysis tool for SystemC. It computes an *Intermediate Representation* (IR) of the behavior and the structure that is free of non-relevant code. This means that we only pay attention to operations that contribute to the computation of a value that is propagated to a port. The second tool named RTLVision is an interactive *Graphical User Interface* (GUI) by Concept Eng.. By obtaining an IR from *ViSyC*, RTLVision is able to display the respective SystemC model.

The paper is structured in the following way: Section II discusses strategies of other approaches as well as their advantages and drawbacks. Section III gives a short overview of SystemC, including its usage and its benefits. Section IV contains the main contribution of this work. Therefore the section is subdivided into four parts. The first part of the section summarizes the architecture of our approach. The second part introduces the GUI. The following two parts go into the details of analysis and visualization of *ViSyC*. Finally, Section V concludes the paper and gives a perspective on future work.

[1]This research work was supported in part by the German Federal Ministry of Education and Research (BMBF) in the Project HERKULES under the contract number 01M3082.

II. Related Work

There are not many other approaches that deal with SystemC visualization. They all use different mechanism for the analysis, as well as for the visualization. In the following both components of each approach are used for an evaluation and a comparison to our work.

One of the first approaches working on SystemC visualization is [2]. Here a modified SystemC kernel is used to collect hierarchy information from the executed model. Since each instance that is derived from `sc_object` is listed in the SystemC kernel, all modules and their connections are known after the evaluation phase. In a second step the extracted hierarchy is transfered to a GUI. The disadvantage of the work is the incompleteness of the hierarchy and the lack of behavioral information.

An approach from the University of Luebeck [3] adopted the basic idea behind [2]. While Große et al. modified the SystemC library directly, the implementation of [3], named gSysC, is a library on top of SystemC. This library holds wrapper functions to expand SystemC by a small set of debugging functions. Via an external GUI, these functions allow to controll the simulation run. The visualized hierarchy is a flat one. All modules including their interconnections, are shown in a single layer. Equivalent to [2] there is no behavior extracted by this approach.

Complementary to gSysC, SystemCXML [4] is an approach that does not rely on an introspection technique at runtime. The extraction of a hierarchy is done via Doxygen. Doxygen parses the SystemC sources and produces XML tags for all module declarations. These tags are used to form an IR of the model. From this IR again a graph description is generated that can be viewed with free tools, e.g. Graphviz. The approach is only able to partially extract the structure. It is not aware of any behavior defined by the processes of the model, and it does not support its visualization.

Another interesting approach is LusSy [5]. LusSy is a project that integrates a backend for visualization and a frontend for analysis of SystemC, i.e. PINAPA [6]. Similar to SystemCXML the visualization is realized as a graph. But differing from other presented work, the analysis frontend PINAPA uses a combination of static analysis and runtime evaluation. The IR, extracted with PINAPA, is used to generate automatons that hold the behavior of the parsed model. But as this automatons consist of states and labeled transitions to represent the semantic of all processes inside a model, the structure is neglected. Hence, LusSy is not appropriate to visualize SystemC models.

Except for LusSy, all approaches listed are unable to obtain a complete hierarchy from a given SystemC model. They do not even collect behavioral information. Another disadvantage is the lack of positional information that would allow to localize structural and behavioral components of the IR. Without this ability features like crossreferencing cannot be implemented.

The visualization engines of the listed approaches are rather simple and have very limited functionality. Even with an analysis that produces an IR with detailed positional information, there would be no bidirectional relation between the source code and the displayed components.

III. SystemC

The system description language SystemC[2] provides hardware constructs, implemented in a C++ class library. The hardware models specified using SystemC can be compiled on a large number of supported architectures using a standard C++ compiler. The compiled executables can be cycle accurate simulations as well as untimed algorithmic descriptions of the given design. The executable specifications can be used for evaluation, debugging and refinement purposes without the usage of a commercial simulator. Depending on the abstraction level the simulation speed can be tremendously higher compared to a functional equivalent HDL model. Because of its unrestricted C++ conformance each SystemC model can be combined with other software libraries. This allows system engineers to take advantage of HW/SW Co-Design and to refine their SoC designs with a high level of flexibility. Another benefit of SystemC, coming with its C++ conformance, is a wide range of abstraction levels that can be used to simplify huge system designs. Complex communication protocols and control logic can easily be separated from functional parts of the specification. For this reason SystemC offers techniques that can raise or lower the level of abstraction. The TLM library implements such a technique to support SystemC's efficient refinement methodology.

SystemC combines features that are typical for an HDL, like concurrency as it appears in hardware, with software paradigms, like object orientation. Those features distinguish SystemC from VHDL, Verilog and SystemVerilog and enable system description capabilities. SystemC also allows arbitrary memory access using pointers and dynamic memory allocation. Even real polymorphism and the concept of virtual functions that binds overloaded class members to function pointers, is applicable in system descriptions.

IV. Tools and Implementation

A. Basic Principles

Because static analysis and the rendering of a SystemC model to an interactive GUI with multiple layers are tasks with a challenging complexity, the functionality of the current work is split into two tools. After *ViSyC* extracted the IR, the GUI can display the model. The communication among these tools is realized with two protocols. One of these protocols holds the IR and is unidirectional. The other one is a bidirectional TCP/IP connection. It is suitable for optional control by user interactions.

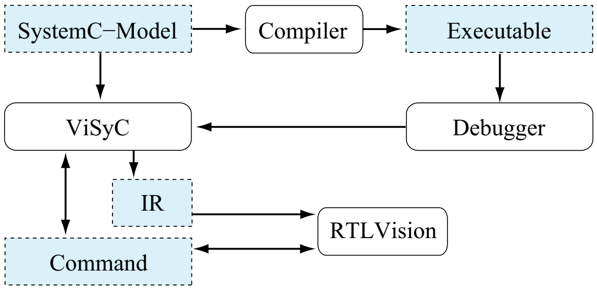

Fig. 1. Architecture of the approach

As Figure 1 shows, the communication between *ViSyC* and RTLVision can be used to catch user commands from a debugger backend that is running a simulation. With this commands *ViSyC* can instruct the GUI to switch to specific parts of the design and to update signal values during execution.

B. RTLVision

RTLVision is a tool from Concept Engineering GmbH[3] for the purpose of graphical system exploration. It enables the designer to navigate through a database interactively.

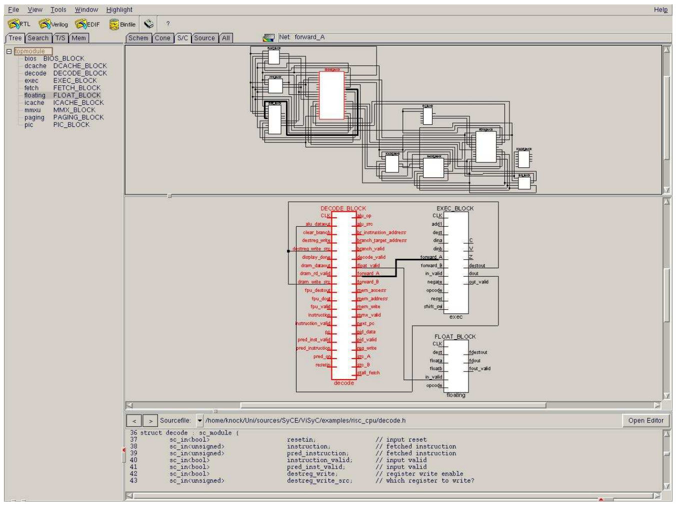

Fig. 2. RTLVision

As can be seen in Figure 2 RTLVision owns a set of views which contain different details of the model to complement each other. Besides the different views, RTLVision has many features, among them are:

- crossprobing
- critical path fragment navigation
- module exploration
- RTL operator symbols

A main difference to the visualization in [1] is the usage of RTL operators, offered by RTLVision. They are used to directly map expressions given by the behavior of the model, instead of mapping black boxes or complex modules that implement the corresponding operators.

[2] www.systemc.org

[3] www.concept.de

C. Analysis

The SystemC analysis tackles the problem of extracting the hierarchy given by a model. C++ does not offer sufficient reflexion capabilities to support programmers by obtaining this hierarchy of all objects during execution. But SystemC implements a technique, called data introspection, to provide this support. As explained in Section II there are many restrictions and problems caused by this technique. That is the reason why an analysis frontend for IR extraction has been implemented in this work.

As mentioned RTLVision integrates features like crossprobing and RTL operators. In order to support all these features during visualization a number of requirements has to be fullfilled by the IR:

1) The complete hierarchy of the model at runtime, neglecting the SystemC library, must be extracted.
2) The complete behavior defined by the processes of the model must be extracted.
3) The positional information of the extracted content has to be available and correct.

Especially the third requirement may sound trivial. But the positional information needed by our visualization backend does not only consist of lines and filenames. The information additionally includes byte offsets which are not reproducible after the expansion of include directives or macro substitutions.

Our analysis frontend consists of a parser that is implemented using PCCTS [7] and an interpreter. The parser generates a static hierarchy of the given SystemC model. The hierarchy consists of a symbol table that is linked to an *Abstract Syntax Tree* (AST) which is a direct acyclic graph. While the symbol table represents the structure of user defined types and functions, the AST describes the behavior of the program. In a following step, the static hierarchy is used as an input for the interpreter. An interpretation simulates the SystemC evaluation phase that instantiates and interconnects modules, starting at `sc_main`.

The result of the interpreter run is an IR that holds a hierarchical description of the memory at runtime. The IR delivers all object instances, independent of their relation to SystemC. Thus, interconnected modules and local variables are stored in the same structure.

D. Visualization

The aim of the visualization phase is to produce an intermediate data object that transfers relevant information of the model to the GUI. While SystemC is embedded into C++, the sources often contain many lines of code that are meaningless or even confusing to the user. Source code passages where tracefiles are handled, code that produces debugging output, stimuli generators or algorithms for time measurement should not be included in the displayed image of the model. Hence, any type of code that does not directly belong to the SystemC model has to be excluded from the visualization output.

The intermediate data object is given by a binary database that facilitates the GUI to interpret even huge sets of data (e.g. gate level) very quickly. This database contains a behavioral and the hierarchical description of a structure to represent the model that is to be displayed. The components that enable an arbitrary description of the hierarchy are listed below:

- modules: defining behavioral blocks
- nets: representing signals and buses
- ports: declaring module interfaces
- flipflops: for storage

There are a lot of other components that are accepted by the GUI for the description of cuircuits. But in order to reduce the complexity of a system description to an abstract level, we only use a subset of the available components for block oriented structures.

The behavior of a model is specified using modules. These connect their input and output ports with subordinated modules and other behavioral elements that are listed in the following:

- gates: and, or, xor, not, ...
- arithmetical operators: adder, multiplier, divisor, ...
- relational operators: equal to, greater or equal, ...
- RTL blocks: multiplexer, encoder, decoder, ...

The connection among modules can be established by nets. One net is connected to a number of ports, greater or equal to two. Not only ports, but also fan-ins and fan-outs of gates, operators and RTL blocks are interconnected by nets.

Unlike the simulation of a SystemC model the visualization process does not start with the function `sc_main`. To make sure all non-relevant details of the model are neglected in the output, the generation of the database starts from a virtual top module. This top module is generated after the analysis process has finished and builds the outermost layer of the module hierarchy. More precisely, it includes all instances that have not been created inside another module. Similar to a C/C++ scope the top module holds a list of module instances, arranged accordingly to the order in which they are allocated during the interpretation phase.

In order to map the required structure to the database, all these instances are traversed recursively. But the SystemC instances cannot be directly mapped to behavioral blocks in the database. Thus the following technique has to be applied to each single module in the hierarchy:

1) Identify the ports:
   All member variables inside the module are checked for their datatype. In case the type is derived from `sc_port` (e.g. `sc_in`, `sc_out` or `sc_inout`) a port is generated that obtains its direction by the type. The template type `T` from `sc_in<T>` is not needed here.
2) Identify interconnected instances:
   Each port inside a valid SystemC program is connected to a signal or another port. Consequently these connections lead to all adjacent instances of the currently observed one. If a connected instance is found that has been identified already, the traversal is stopped.
3) Identify interconnections:
   The recursive algorithm from above is started for each identified port P_i within an instance. Once it terminates, all ports and signals that are assigned to P_i are known. While signals are directly mapped to nets, a port demands an extra net to be created, connecting this port and P_i.

The way the behavior is examined is very similar to the recursive way the structure is processed. Defining behavior in SystemC is inherently attached to the choice of a process type. As all processes are registered in `SC_CTOR`, the names of all process functions can be obtained by finding all usages of `SC_METHOD`, `SC_THREAD` and `SC_CTHREAD`. When traversing the AST of such a function block for visualization purpose, we neglect all declarative statements. These have been stored in a symbol table during analysis already. Thus

```
if ( expr )
    then_stmt;
else
    else_stmt;
```

```
switch ( expr )
{
    case 0:
        else_stmt;
        break;
    default:
        then_stmt;
}
```

Fig. 3. Transferring `if` to `switch` statement

only three kinds of statements remain: expression statements, loop statements and conditional statements.

An expression statement that contains nothing but a concatenation of expressions, is the simplest statement to be visualized. Each operator is mapped directly into the database while traversing the AST in reverse order. This is done by observing the operands first. An operator cannot be written to the database until all its operands are processed. For this reason the visualization of a module behavior starts at its output ports. Since the usage of overloaded operators cannot be directly mapped, they are handled as function calls. Function calls again are treated like modules since they have the potential to use controll-flow statements. Conversions are not displayed since they can change the semantics of operators.

Statements that do affect the control flow of a process are not as easy to handle as operators. The GUI deals with data flow elements only which means that iterations and conditional statements have to be transfered into expressions. To simplify this process, all `if` statements are encoded as `switch` statements as shown in Figure 3.

The visualization of a `switch` statement is done in four phases:

1) All statements that are potentially executed have to be pure expressions without the use of loops, recursions or conditions. All these mechanisms are replaced by an equivalent expression.
2) There has to be a statement for each possible evaluation of the condition. If a `default` path is missing, it is going to be attached. In case the condition is true all variables that receive no new value are assigned to their current values in the new `default` path.
3) Each `case` statement is visualized recursively.
4) Each variable assignment, in a `case` statement is connected to a multiplexer. The multiplexer uses the value of the condition to decide which path is chosen for the assignment. If there are less assignments for a variable than possibilities to evaluate for the condition, then all unhandled cases are linked to the current value of the variable.

As usually done in Verilog or VHDL, loops are unrolled. The result of this technique is given by m copies of the behavior described inside the loop. The value m has to be static, as it must be known without executing the model. All copies are tied together by giving their results of variable v as the initial values of v in the next copy.

Especially the described techniques for visualizing control-flow statements are analog to the synthesis of HDL specifications. But they provide the mapping of a system model to a display before the model can be synthesized (shown in Figure 4). The mapping again helps system designers to evaluate their models. Since the complete model is visualized and elementary features like path tracing can be controlled from a backend, our approach offers a bidirectional intuitive interface to other development phases like debugging.

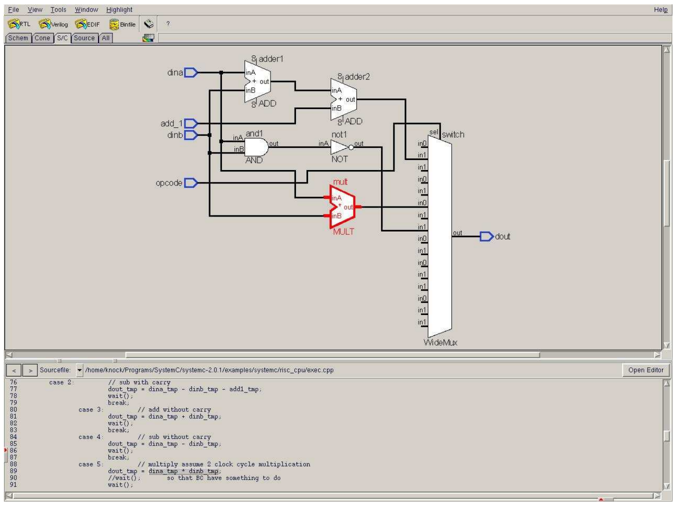

Fig. 4. Wordlevel view

V. Conclusion

In this paper we presented an approach for visualization of complete SystemC designs at different levels of abstraction. The approach uses the tools *ViSyC* and RTLVision. The described tools for analysis and visualization support features like crossprobing, path fragment navigation and data hiding of non-relevant details. The approach does not rely on a modified SystemC library and is not restricted by the data introspection of SystemC.

Future work includes communication facilities for debugging backends. Also the type conversions are left for future work. This will support system level developers in testing and correcting SystemC designs at runtime.

References

[1] C. Genz and R. Drechsler, “System exploration of SystemC designs,” in *IEEE Annual Symposium on VLSI*, 2006, pp. 335–340.

[2] D. Große, R. Drechsler, L. Linhard, and G. Angst, “Efficient automatic visualization of SystemC designs,” in *Forum on Specification and Design Languages*, 2003, pp. 646–657.

[3] C. Eibl, C. Albrecht, and R. Hagenau, “gSysC: A graphical front end for SystemC,” in *European Conference on Modelling and Simulation*, 2005, pp. 257–262.

[4] D. Berner, H. Patel, D. Mathaikutty, J.-P. Talpin, and S. Shukla, “SystemCXML: An extensible SystemC front end using XML,” Virginia Polytechnic Institute and State University, Tech. Rep. 06, 2005.

[5] M. Moy, F. Maraninchi, and L. Maillet-Contoz, “LusSy: A toolbox for the analysis of systems-on-a-chip at the transactional level,” in *International Conference on Application of Concurrency to System Design*, 2005, pp. 26–35.

[6] M. Moy, F. Maraninchi, and L. Maillet-Contoz, “PINAPA: An extraction tool for SystemC descriptions of systems-on-a-chip,” in *ACM international conference on Embedded software*, 2005, pp. 317–324.

[7] T. Parr, *Language Translation using PCCTS and C++: A Reference Guide*. Automata Publishing Company, 1997.