Efficient Automatic Visualization of SystemC Designs

Daniel Große Rolf Drechsler Lothar Linhard Gerhard Angst

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{grosse, drechsle}@informatik.uni-bremen.de

Concept Engineering GmbH
Bötzinger Str. 29
79111 Freiburg, Germany
{lothar, gerhard}@concept.de

Abstract

Complex designs can only be understood, if the underlying information is provided in a concise way. In this context visualization is becoming an essential part of system design, understanding and debugging. In this paper we present an approach to visualization of designs described in SystemC. For the visualization an industrial tool including many features, like e.g. schematic viewing, cross-probing between schematic view and source code view, critical path fragment navigation or object search, is used by an API. The techniques how to extract the information from SystemC are described and examples are provided.

1 Introduction

While classical approaches to circuit design make use of hardware description languages (HDLs), like VHDL or Verilog, there is a strong interest in C-like description languages. These languages allow for fast simulation in an early stage of the design process. Furthermore, hardware/software co-design can be performed in the same system environment. One of the most popular languages of this type is SystemC. As a C++ class library SystemC enables modeling of systems at different levels of abstraction starting at the functional level and ending at a cycle-accurate model. The well-known concept of hierarchical descriptions of systems is transferred to SystemC by modeling a module as a C++ class [GLMS02, MRR03, S02].

Using the top down design methodology a system level description has to be refined down to implementation. During this process verification plays an important role. Verification can be divided in simulation based and formal methods. In case of SystemC, examples of the simulation based techniques are [RHKR01, FRS02]. First formal approaches to check the behavior of a circuit description in SystemC have been reported in [DG02, GD03]. For the verification process of today's complex systems and to understand these systems their visualization is important. Particularly with regard to debugging a concise visualization methodology has to be developed.

In this paper an approach is presented to automatically extract structural information of a SystemC design and then transfer it to a commercial visualization tool[1] (using an API). This allows for an easy design understanding. With the presented technique it is possible to "navigate" through the hierarchy of the complete SystemC design. Furthermore the corresponding source code of any SystemC module is linked to the graphical module representation.

[1] www.concept.de

The paper is structured as follows: In Section 2 a short introduction to SystemC is given. Section 3 describes the visualization tool and shows an example how to use its API. Then the technique to extract the design data of a system described in SystemC is discussed in Section 4. Case studies are given in Section 5. The conclusions and an overview of future work are provided in Section 6.

2 Introduction to SystemC

The main features of SystemC for modeling a system are described and a simple circuit example is given. The SystemC design methodology is based on the following:

- Modules are the basic building blocks for partitioning a design. A module can contain processes, ports, channels and other modules. Thus, a hierarchical design description becomes possible.
- Communication is realized with the concept of interfaces, ports and channels. An interface defines a set of methods to access channels. Through ports a module can send or receive data and access channel interfaces. A channel serves as a container for communication functionality, e.g. to hide communication protocols from modules.
- Processes are used to describe the functionality of the system, and allow expressing concurrency in the system. They are declared as special functions of modules and can be sensitive to events, e.g. an event on an input signal.
- Hardware specific objects are supplied like e.g. signals, which represent physical wires, clocks, and a set of data-types useful for hardware modeling.

Besides this, SystemC provides a simulation kernel. The functionality is similar to traditional event-based simulators. Note that a SystemC description can be compiled with a standard C++ compiler to produce an executable specification. The output of a system can be textual, using C++ routines like `cout` for instance, or waveforms. Interfacing other software is also possible [CRAB01].

As an example to show how a circuit can be modeled in SystemC a scalable bus arbiter has been chosen [McM93]. This circuit controls the access of n clients to a bus and combines priority arbitration with a round robin technique. This guarantees that every client has access to the bus. The circuit consists of n cells, one for each client. The description is hierarchical. On the lowest layer it is modeled at gate level. In Figure 1 the SystemC code of a single arbiter cell is shown. The cell is based on basic gates, like AND and OR. In the cell description positional port binding is used in which the gates have been declared so that inputs are followed by outputs. The top level description of the arbiter is not shown. This example is also used in the following to demonstrate the main features of our visualization tool.

3 Visualization Tool

In this section, first a brief description of the motivation of the project is given. Then some details about the visualization tool that is used to display SystemC designs are given.

```
SC_MODULE(Cell) {
  sc_in_clk TICK;
  sc_in<bool> req_in;
  sc_in<bool> tok_in;
  sc_in<bool> gra_in;
  sc_in<bool> ove_in;
  sc_out<bool> ack_out;
  sc_out<bool> gra_out;
  sc_out<bool> tok_out;
  sc_out<bool> ove_out;

  FlipFlop *W,*T;
  AndGate  *A1,*A2,*A3,*A4;
  OrGate   *O1,*O2,*O3;
  NotGate  *N;
  sc_signal<bool> a,b,c,d,e,f;

  SC_CTOR(Cell) : TICK("clk"), req_in("req_i"), tok_in("tok_i"),
  gra_in("gra_i"), ove_in("ove_i"), ack_out("ack_o"), gra_out("gra_o"),
  tok_out("tok_o"), ove_out("ove_o") {
    W = new FlipFlop("Wait"); (*W)(TICK,b,c);
    T = new FlipFlop("Token"); (*T)(TICK, tok_in, tok_out);
    A1 = new AndGate("AndGate1"); (*A1)(a, req_in,  b);
    A2 = new AndGate("AndGate2"); (*A2)(c, tok_out, d);
    A3 = new AndGate("AndGate3"); (*A3)(f, gra_in,  gra_out);
    A4 = new AndGate("AndGate4"); (*A4)(req_in, e,  ack_out);
    O1 = new OrGate("OrGate1"); (*O1)(c, tok_out, a);
    O2 = new OrGate("OrGate2"); (*O2)(d, ove_in, ove_out);
    O3 = new OrGate("OrGate3"); (*O3)(d, gra_in,  e);
    N = new NotGate("NotGate"); (*N)(req_in, f);
  }
};
```

Figure 1: SystemC description of a single arbiter cell

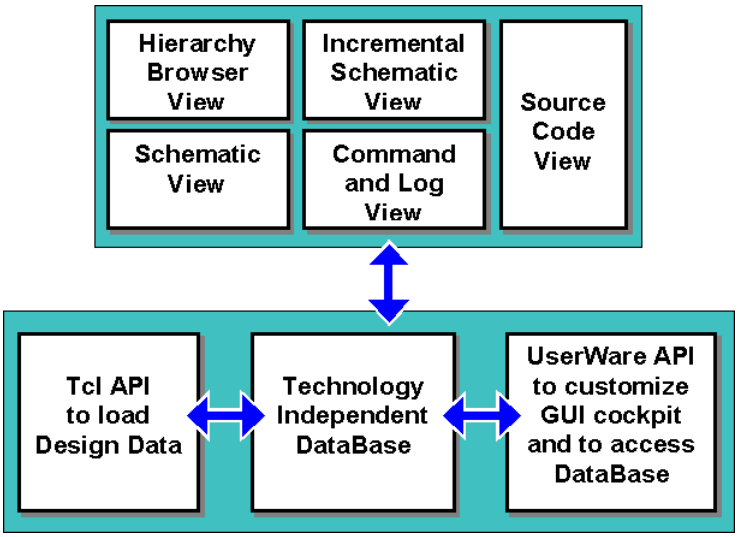

Figure 2: Architecture of Concept Engineering's customizable GUI cockpit

3.1 Motivation and Background of the Project

The company Concept Engineering develops schematic generation and viewing technology for use with logic synthesis, verification, test automation and physical design tools since 1990. Concept's existing visualization technology already supports automatic schematic generation at the transistor, gate, register-transfer, block and system level. In the first place the visualization products are used at gate level and below. Endorsement of SystemC by multiple EDA and semiconductor companies is now starting to create more demand for advanced system level visualization and debugging technology and as a result Concept Engineering has partnered with the University of Bremen to explore new technologies to accelerate the development and debugging of SystemC based designs.

At the project start the following tools were available:

Nlview™ Widgets: Nlview Widgets integrate with the most popular GUI development environments, including Tcl/Tk, Java, MFC, ActiveX and Perl. Nlview Widgets relieve EDA tool designers of the tedium and productivity loss of having to develop their own schematic generation software by providing a set of robust and flexible software components for automated schematic generation and viewing. By adopting Nlview Widgets, developers can quickly realize GUIs for EDA products with shorter design cycles and lower development and maintenance costs.

SpiceVision™: SpiceVision is an interactive visualization tool that helps chip designers debug and analyze SPICE circuits and models.

GateVision®: GateVision is a stand-alone graphical netlist analyzer that allows intuitive design navigation, schematic viewing, logic cone extraction, interactive logic cone viewing, and design documentation.

While all these tools have very strong visualization engines, there is no dedicated interface to system level descriptions, like SystemC.

3.2 Description of Tool Features

The proposed approach described in this paper is based on a language and technology independent visualization cockpit. This visualization cockpit provides a full featured GUI with different cross-linked design views and with a technology independent data base. A powerful UserWare API allows the customization of the GUI cockpit to meet specific debugging demands. The architecture of the customizable GUI cockpit is shown in Figure 2.

As a first step design data is loaded into the visualization system using a Tcl based API. Typical design objects stored in the GUI data base are for example: components, instances, ports, nets and object attributes. The data base is technology independent and is capable to store data for all common design levels e.g. transistor level, gate level, RT level and system level. All data base objects support source code attributes so that each data base object can be cross-probed to one or more source code regions. The multi-view GUI cockpit in combination with the data base and the source code attributes provides an easy-to-use and customizable platform for technology independent and language independent graphical debugging. Some example code fragments to load a circuit structure into the data base are shown in the following:

```
# hierarchy information generated from SystemC description
set db [ zdb new ]
set p ~/systemc/simvis

#---- FLIPFLOP ----
$db load primitive FlipFlopM DFF
$db load port clock input -spos $p/FlipFlop.h 98 116
$db load port in input -spos $p/FlipFlop.h 117 134
$db load port out output -spos $p/FlipFlop.h 135 154

#---- ANDGATE ----
$db load primitive AndGateM AND
$db load port in1 input -spos $p/AndGate.h 129 147
$db load port in2 input -spos $p/AndGate.h 148 166
$db load port out output -spos $p/AndGate.h 167 186

#....

#---- MODULE ----
$db load module Cell
$db load port TICK input -spos $p/Cell.h 181 198
$db load port req_in input -spos $p/Cell.h 199 220
$db load port tok_in input -spos $p/Cell.h 221 242
# ....
$db load inst Wait FlipFlopM
$db load inst Token FlipFlopM
$db load inst AndGate1 AndGateM
# ....
$db load net net8 -pin Wait clock -pin Token clock -port TICK
$db load net signal_1 -pin Wait in -pin AndGate1 out
#....
```

```
#---- TOP MODULE ----
$db load module topmodule -top
$db load inst cell0 Cell
$db load inst cell1 Cell
#....

Main:DataBaseChanged $db
```

A one-to-one correspondence to the SystemC arbiter cell shown in Figure 1 can be seen. First the basic gates are declared using the `load primitive` command followed by the port declaration. Notice that with the `-spos` attribute a source code reference is given (in this example a reference to the byte positions where the considered port is declared). Then the arbiter cell is declared with the necessary instances and the appropriate connections. Below the comment "top module" the instances of different cells are shown.

Once the design data is loaded into the data base, the GUI cockpit automatically provides the usual debugging features such as: schematic viewing, critical path fragment navigation, object search, design documentation and cross-probing between the design views and source code view. These built-in debugging features are provided without additional coding just by loading the data base of the debug cockpit. As shown in the example code, each design object loaded into the data base can hold source code location attributes that provide detailed information where the corresponding object is defined in the SystemC code. This allows cross-probing of design objects from different design views such as schematic view or schematic fragment view to the SystemC source code view. In addition the cockpit data base and the GUI views also support hierarchical design descriptions. By this interactive design navigation, search operations, schematic rendering and other operations is enabled on all levels of hierarchy, from complete top level overview to all sub-circuit levels. The hierarchy tree view provides an easy-to-understand overview about the design structure.

In addition to the "usual" debug features the UserWare API allows the customization of the debugging environment. For example users can write design rule checkers or other user-defined tool functions. An example of the userware feature is given in the case studies. For an impression on the capabilities of the visualization tool the graphical view of one arbiter cell is shown in Figure 3.

4 Link to Visualization Tool

This section describes the usage of the integrated simulation kernel of SystemC to explore a SystemC design allowing to load the design data into the visualization tool. Starting with an overview of the proposed method the approach is discussed in detail.

4.1 Overview

The simulation kernel of SystemC is facilitated to extract the information needed in the visualization tool. Necessary procedures are compiled together with the design and the simulation kernel as shown in Figure 4.

By executing the design the extraction method is started. So the method is able to query the kernel. Only at top level some extra code has to be added to the design: After design instantiation

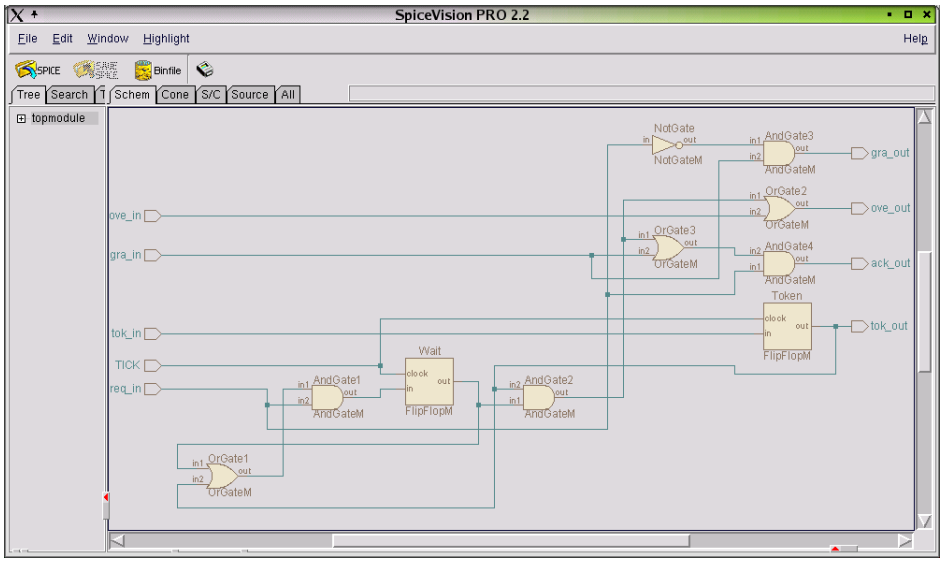

Figure 3: Gate level view of arbiter cell

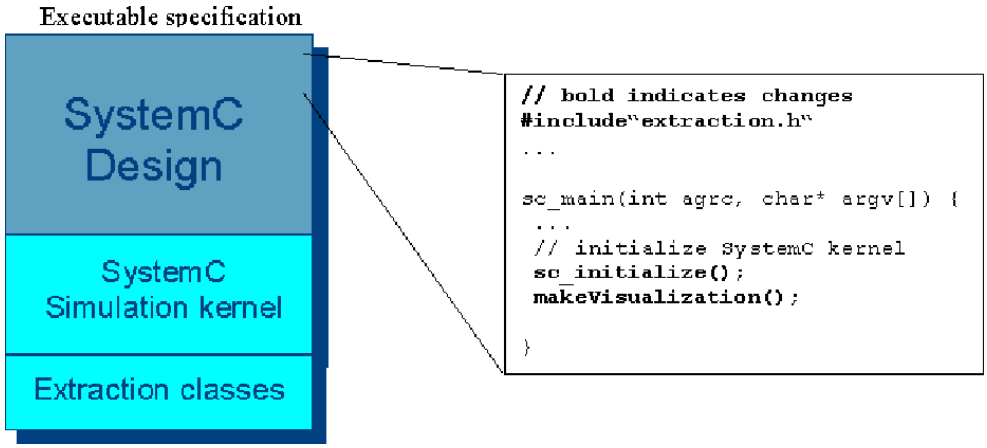

Figure 4: Connection to the simulation kernel of SystemC

and initialization of the kernel (see right hand side of Figure 4) the extraction algorithm is called (`makeVisualization()`). Normally the method `sc_initialize()` is called by `sc_start` directly, but here it has to be executed separately in order to set up all connections of the design correctly before the extraction algorithm is started. Of course the new classes have to be included in the top level file.

The recursive extraction algorithm is shown in pseudo code in Figure 5. The algorithm explores the SystemC design top down. Starting from a list of all modules instantiated at top level it identifies for each module its child objects. These objects are filtered and for every sub-module the algorithm is called recursively. This means that if a fixed module m is considered, all sub-modules of m and their interconnections have already been handled. In steps 2(a), (b), (c), 3 of the extraction algorithm the API of the visualization tool is used to load the data base of the visualization tool with the SystemC design (see Section 3).

Next, the kernel methods used by the extraction algorithm are discussed.

1. Get a list l of all top level modules
2. For each module m of l do
   (a) Get list c of child objects (sub-modules, ports, ...) of m
   (b) Call recursively 2. with l equals the list of sub-modules in c
   (c) Declare module m and its ports
   (d) Instantiate every sub-module in c
   (e) Create connections from current ports to ports of sub-modules in c and interconnections of sub-modules
3. Create top level connections

Figure 5: Sketch of the extraction algorithm for SystemC designs

4.2 Acquisition of Design Data

If the executable specification of a SystemC design - extended with the proposed classes - is started, the SystemC scheduler, that is a part of the simulation kernel, performs the initialization phase. After this phase the extraction method is executed. In this situation the structure of the design is determined, i.e. all modules and channels have been instantiated and are registered to the simulation kernel. Now the extraction algorithm can call methods of the kernel to collect information. The list of all instantiated modules needed in step 1 of the algorithm is available via the object manager of SystemC. The object manager provides the methods `first_object()` and `next_object()`. The following code shows how the list l is filled:

```
sc_object_manager* objm = sc_get_curr_simcontext()->get_object_manager();
sc_object* sop = objm->first_object();
while (sop) {
  if (strcmp(sop->kind(),"sc_module") == 0) {
     // add sop to list l
     ...
  }
  sop = objm->next_object();
}
```

The `kind()` method of a SystemC object allows to distinguish the type of the object. Notice that all this is part of the SystemC simulator. The child objects of a module can be determined by method `get_child_objects()` (necessary in step 2(a)). Because ports correspond to interfaces it is possible to extract the connections of a module with its surrounding components by calling the method `get_interface()`. Hash tables are employed as data structures to store interconnections within and beyond the current level of the considered module.

4.3 Link to Source Code of a SystemC Design

Up to now the described approach can only use information of a SystemC design which is available through the simulation kernel. Here arises a problem with the port names of a SystemC module.

```
SC_MODULE(name) {
  // declare ports, signals, member functions, sub-modules
  SC_CTOR(name) {
   // body of constructor
   // process declaration, sensitivities
   // instantiation of sub-modules
  }
};
```

Figure 6: A SystemC module

It has to be distinguished between names given in the source code and names passed during instantiation. The instantiation names are available due to according method calls but may not be given. (In this case the kernel automatically generates names of the type `port_i`.) Therefore the port names given in the source code are very important. In the current implementation a simple parser resolves the port names for a given module out of the corresponding file. Another important reason for the link to the source code is its annotation to the graphical representation of the corresponding module. The source code link which has to be available in the extraction algorithm has been realized as follows: Consider a SystemC module as shown in Figure 6. By changing the macro `SC_CTOR` (using the standardized macros `__FILE__` and `__LINE__`) and the definition of the class `sc_module` it is possible to obtain the filename where a module is declared because the preprocessor of the C++ compiler replaces the latter macros during the compilation of the design. The class `sc_module` of SystemC has been extended with two member variables (which store the filename and the line number of the constructor definition) and a virtual method which is called at the end of elaboration time, i.e. just before simulation. This virtual method uses the two mentioned macros to set the values of the variables. By this, during the extraction process all essential information is available.

With the presented technique it is possible to efficiently extract the structure of a SystemC design for visualization. The extraction process considers components of the design only once and also only the active parts. This is due to the fact that the extraction algorithm is compiled together with the design and thus is able to obtain the design information from the simulation kernel of SystemC.

5 Case Studies

In this section two case studies are shown and some of the main features of the visualization tool are discussed by examples. The tool is platform independent and can be run under Windows, Sun Solaris, HP-UX and Linux.

5.1 A Scalable Arbiter

Screen shots for the scalable arbiter from Section 3 are given in Figure 7 and 8. In this case an arbiter consisting of 5 cells is shown. The hierarchy structure is shown in a “directory-like” manner in the hierarchy tree window on the left side of the tool. This allows to easily navigate through the design and provides an easy understanding. The user has a search function and can e.g. search for names of cells.

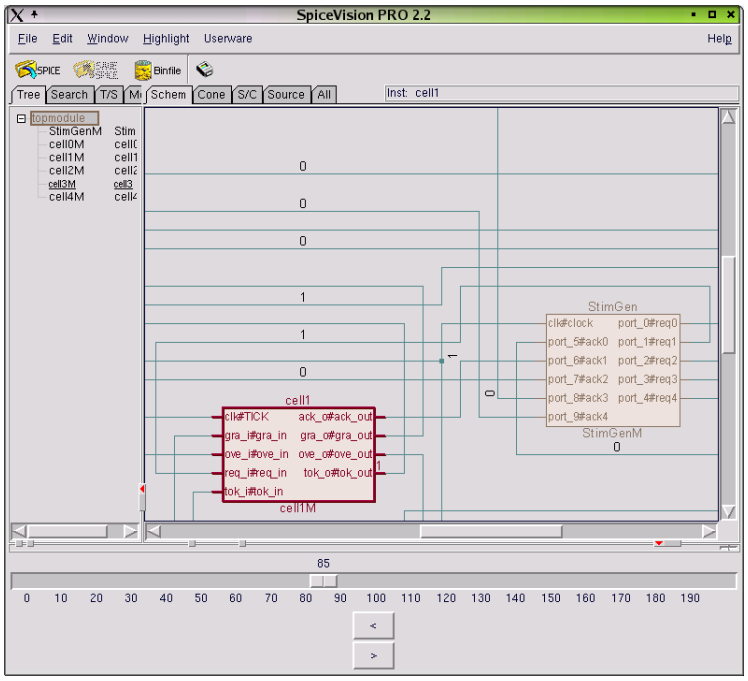

Figure 7: System level view of arbiter with simulation data

The corresponding components of SystemC modules are shown on the right hand side of the figures. While Figure 7 shows a part of the top level view of the arbiter, Figure 8 provides an idea on displaying hierarchies. By using boxes the user gets a direct understanding when signals go from one module to the next. Here the cone extraction view has been selected to focus only on parts of the current cell. By a simple drag-and-drop mechanism for each cell the corresponding source code can be shown. With selecting the tab *Source* the SystemC source code is displayed.

5.2 RISC CPU

As a second example the RISC CPU available from the SystemC web page is considered. This is an example, where the gate level description is not available. Thus the basic components at the lowest level are simply displayed as empty boxes. The corresponding system level building blocks of the CPU are shown as symbol boxes in the schematic view window. A screen shot is given in Figure 9. In the lower part the source of the selected module `floating` can be seen. Here also the flexibility of the cockpit is demonstrated. By a simple additional command the source code is shown in parallel to the graphics.

Other options are to split the screen or to annotate simulation values. This has been done in a prototype environment with application in verification (see Figure 7). In this context it is often important to track a signal through the design. Technically the annotation with simulation traces has been realized by exporting which signals are traced (by the SystemC method `sc_trace`) during the simulation. This is necessary to match these signals with the corresponding ones in the visualization tool. A simple Tcl script parses the vcd waveform file and transmits the trace values to the visualization tool.

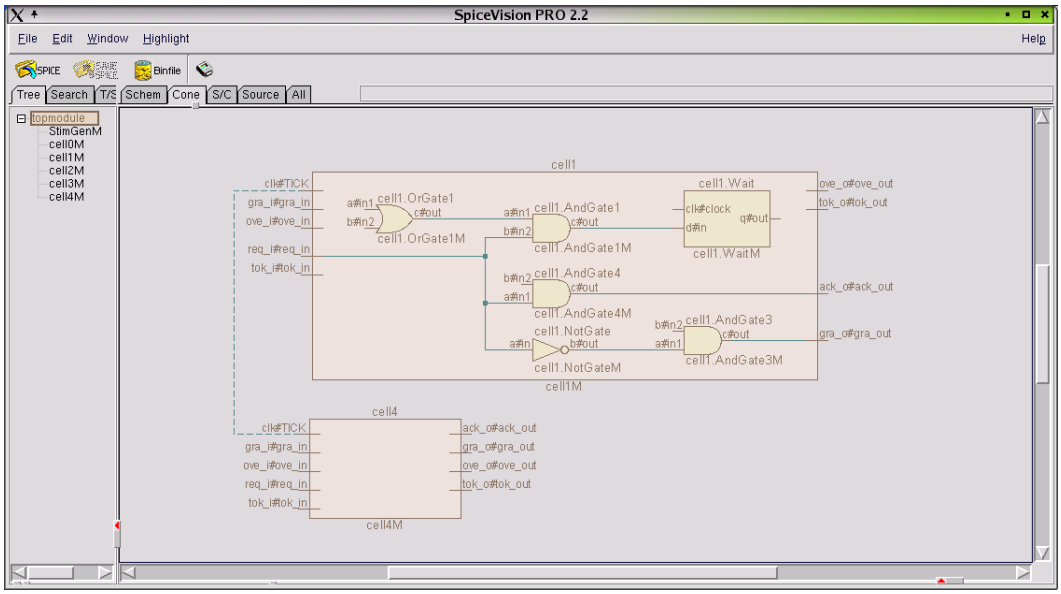

Figure 8: Hierarchy of arbiter

6 Conclusions and Future Work

In this paper a visualization environment for designs described in SystemC has been presented. The data needed during displaying is extracted at run time. For this, slight modifications of the SystemC simulation kernel are needed. The tool is very flexible and uses a simple parser. Information on input and output names of modules can be extracted efficiently without a detailed parsing step of the whole design.

System visualization is an important step in understanding and debugging of designs. For this, it is focus of future work to integrate the visualization tool in a debugging environment. By the annotation of simulation values a first step in this direction has been done.

References

[CRAB01] L. Charest, M. Reid, M. Aboulhamid, and G. Bois. A methodology for interfacing open source SystemC with a third party software. *Design, Automation and Test in Europe*, pages 16–20, 2001.

[DG02] R. Drechsler and D. Große. Reachability analysis for formal verification of SystemC. In *EUROMICRO*, pages 337–340, 2002.

[FRS02] F. Ferrandi, M. Rendine, and D. Scuito. Functional verification for SystemC descriptions using constraint solving. In *Design, Automation and Test in Europe*, pages 744–751, 2002.

[GD03] D. Große and R. Drechsler. Formal verification of LTL formulas for SystemC designs. In *IEEE International Symposium on Circuits and Systems*, pages IV:748–IV:751, 2003.

[GLMS02] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

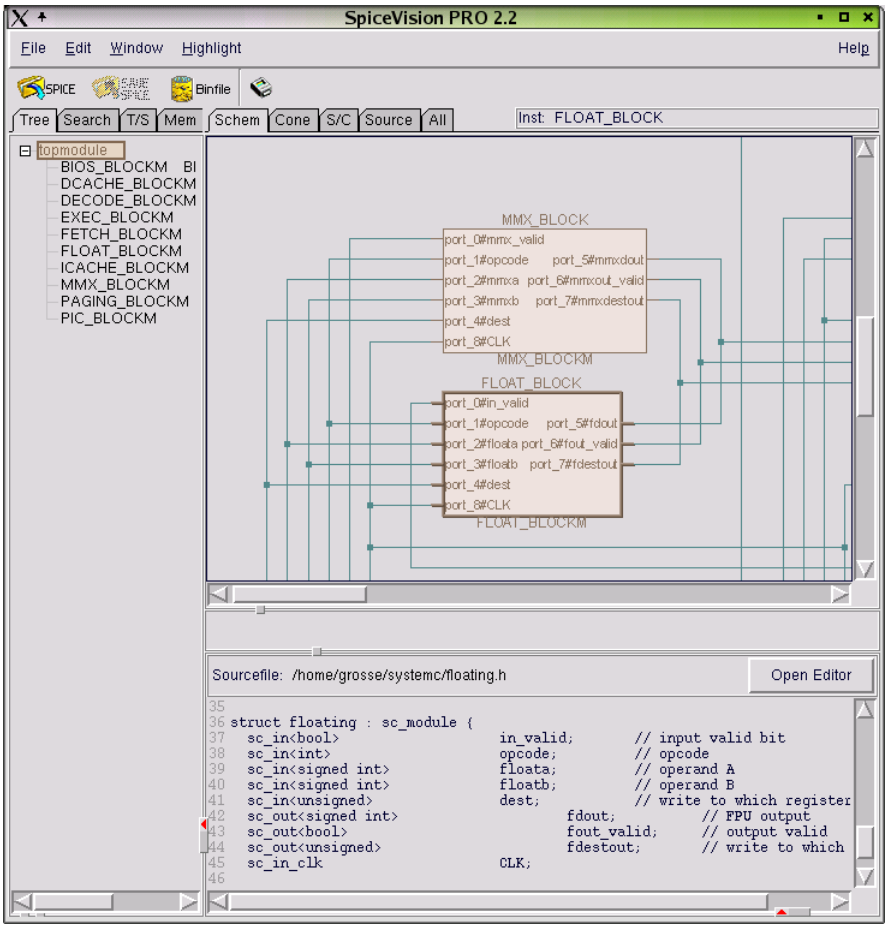

Figure 9: RISC CPU

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publisher, 1993.

[MRR03] W. Müller, W. Rosenstiel, and J. Ruf. *SystemC Methodologies and Applications*. Kluwer Academic Publishers, 2003.

[RHKR01] J. Ruf, D. W. Hoffmann, T. Kropf, and W. Rosenstiel. Simulation-guided property checking based on multi-valued ar-automata. In *Design, Automation and Test in Europe*, pages 742–748, 2001.

[S02] Synopsys Inc., CoWare Inc., and Frontier Design Inc., http://www.systemc.org. *Functional Specification for SystemC 2.0*.