

# *gSysC*: A GRAPHICAL FRONT END FOR SYSTEMC

Christian J. Eibl, Carsten Albrecht, and Rainer Hagenau  
Institute of Computer Engineering  
University of Lübeck  
Ratzeburger Allee 160, D-23538 Lübeck, Germany  
Email: {eibl, albrecht, hagenau}@iti.uni-luebeck.de

## KEYWORDS

SystemC, GUI, Simulation Controller

## ABSTRACT

Nowadays, hardware development bases on high-level methods with appropriate tool support. SystemC, a C++ class library, provides a high-level interface to model and simulate hardware designs on different levels. Unfortunately, there is no graphical interface included for demonstration, debugging, or educational purposes. *gSysC* presented here forms a GUI to SystemC. It allows the programmer to watch the interaction of the simulated design parts and provides more runtime control features such as single-step simulation or breakpoints.

## 1 INTRODUCTION

Simulation is a state-of-the-art process to test, verify, and profile newly designed hardware models. Generally, it provides exhaustive views and in-depth analysis of crucial, unapproachable system parts. Nowadays, an increasing number of hardware design tools using hardware description languages are available. But there is a demand for higher-level methods of modelling and simulating, especially for hybrid hardware/software systems. SystemC [SystemC 2002] brought out

by a pool of companies is a C++ class library that allows simulation of systems compounded of modules modelled on varying abstraction levels. It backs the top-down design methodology so that each module can be iteratively redesigned. Unfortunately, SystemC models can only be analysed by trace and log files. Visualisations of simulated modules and their interaction is not provided but they can be of course introduced utilising any library for graphical output. A fixed graphical user interface (GUI) would help to benefit even more of SystemC simulation models. Especially in the areas of presentation, demonstration, and education, visual support would be helpful.

In the following, similar systems are introduced. Their strengths and weaknesses are shortly discussed and, in contrast to them, the benefit of *gSysC* is explained. Moreover, the main details are shown presenting first the concept, further on, its implementation and way of application and a case study utilising a simple model of a CPU with cache connected to a RAM module via a bus.

## 2 RELATED WORK

Currently, there are both commercial and non-commercial approaches to a GUI for SystemC. Commercial approaches generally integrate SystemC into their hardware design environments. It is used for fast functional analy-

sis and verification. For example, Prosilog Magillem [Prosilog 2005] allows the user to automatically generate SystemC as well as Verilog and VHDL code for the graphical hardware design. Moreover, simulators such as the Incisive Unified Simulator of Cadence support SystemC as well. They are able to simulate mixed-language designs so that test benches can be easily defined using SystemC. But the simulation engine is a proprietary one.

A GUI based on the open SystemC library is described in [Charest et al. 2001, Reid et al. 2001b]. It is a self-made, Qt-based front end. Qt is a platform independent GUI library for C++ [Trolltech 2002]. The focus is laid on the graphical run-time observation of signals. The interconnection of GUI and SystemC is implemented by adapting the SystemC library so that the GUI is notified of signal-value alterations by the simulation engine. The GUI provides a simulation controller for configuration, initialisation, and run-time control. In contrast to this system, the GUI presented in [Große et al. 2003] shows the full system architecture using the interactive visualisation tool SpiceVision<sup>TM</sup>. All information on system model and run-time signal values is extracted by a modification of the SystemC library.

So, both visualisation systems have a firm bond to the SystemC library because of the necessary modifications. Since 1999, several updates and revisions of SystemC were released so that both GUIs require high effort to keep them up to date.

### 3 CONCEPT OF *gSysC*

In contrast to these tightly-coupled approaches, *gSysC* provides a loosely-coupled GUI for SystemC. It is based on Qt as well and can be fully removed by a compiler flag without changing the code.

The main goals of *gSysC* are to be independent of different releases, the programmer's permission, and control of what parts are shown, and an

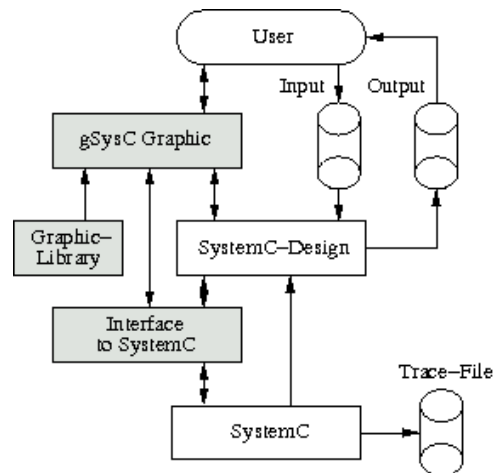


Figure 1: Programmer's View of SystemC and *gSysC*.

option of removing the visualisation. Additionally, because of the high portability of SystemC, the GUI should be supported by most platforms supporting SystemC.

Figure 1 shows the interfaces between user, SystemC model, SystemC simulation kernel represented by white boxes and the *gSysC* extensions in grey boxes. The user interface of SystemC is limited to reading a configuration at the simulation start and writing textual information or signal traces to the console or hard disk. The user cannot interact with the running simulation. The SystemC design is the software model of the simulated hardware and SystemC represents the simulation kernel and library. *gSysC* based on the graphic library provides besides graphical presentations of the design run-time access to the simulation. It introduces a simulation controller shown in figure 2 that provides single-step simulation, simulation of a certain number of clock cycles, or conditional break points, e. g. the simulation halts at a certain signal value. For an automatic and continuous simulation the time delay of each cycle can be determined. Further on, the GUI shows the simulated models with its modules and allows the user to browse the different levels and properties of modules and signals.

The GUI is attached to the simulation program by preprocessor macros. They register

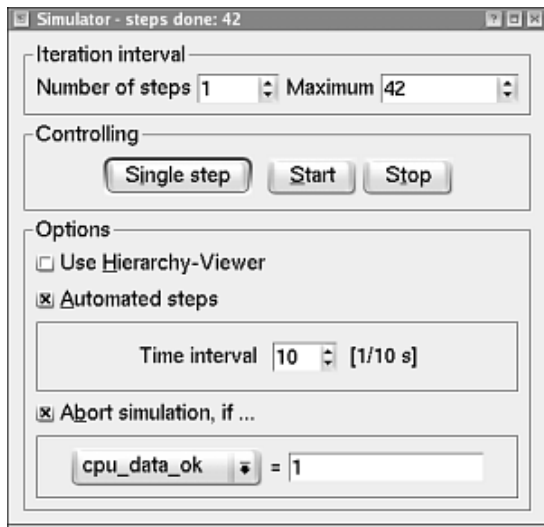


Figure 2: Simulation Controller.

each module and signal ports as well as their relation to others. Especially for demonstration purposes, the programmer can focus on important parts and leave out less important ones. The macros include code to indicate any value changes to the GUI kernel. The intervals of indication can be configured as a number of clock cycles. The SystemC-equivalent port classes in the lower layer are derived from them and apply the SystemC classes. *gSysC* takes control of the simulation system by overloading SystemC's control functions such as `sc_start()`. A base set of library methods can be used to show certain details within a simulation run, e.g. fill level of buffers. Due to the variety of simulation opportunities the programmer can enlarge this set by own application-specific extensions.

In case of an almost bug-free SystemC simulation model, *gSysC* can be removed at compile time by setting a define flag. Then, an unmodified SystemC simulation is created. This is suitable for long-term runs in particular.

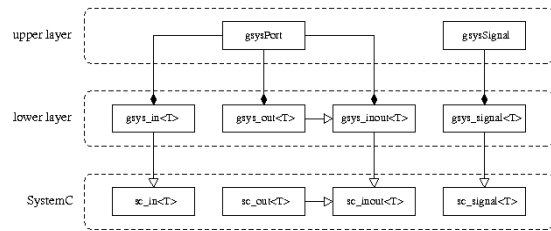


Figure 3: Architecture of the Interface Between *gSysC* and SystemC.

## 4 IMPLEMENTATION AND APPLICATION

The implementation of *gSysC* has to perform a balancing act between SystemC, Qt and its goals previously defined. The avoidance of additional programmer's code for visualisation leads to a hierarchical structure for the interface between SystemC and *gSysC*. Nevertheless, applying *gSysC* to SystemC models requires some additional code lines.

### 4.1 Interface Between *gSysC* and SystemC

The interface architecture is shown in figure 3 including all classes and their relations. It is divided into two layers. In the lower layer, which is next to SystemC, port and signal classes of SystemC are derived in order to receive new values written on them. In the upper layer there are port and signal equivalents of *gSysC* for processing purposes in *gSysC*. In addition to the values, the *gSysC* classes provide information and functions for visualisation. The included functions perform port and signal highlighting, emphasize the position and the connected neighbour modules, and open property information windows. The lower layer contains all classes derived from SystemC. These wrapper classes are necessary to achieve a data-type independent implementation and to get the opportunity to easily use sets and lists, even with ports and signals of different data types. Access to the SystemC layer is only performed by the derived *gSysC* classes so that all

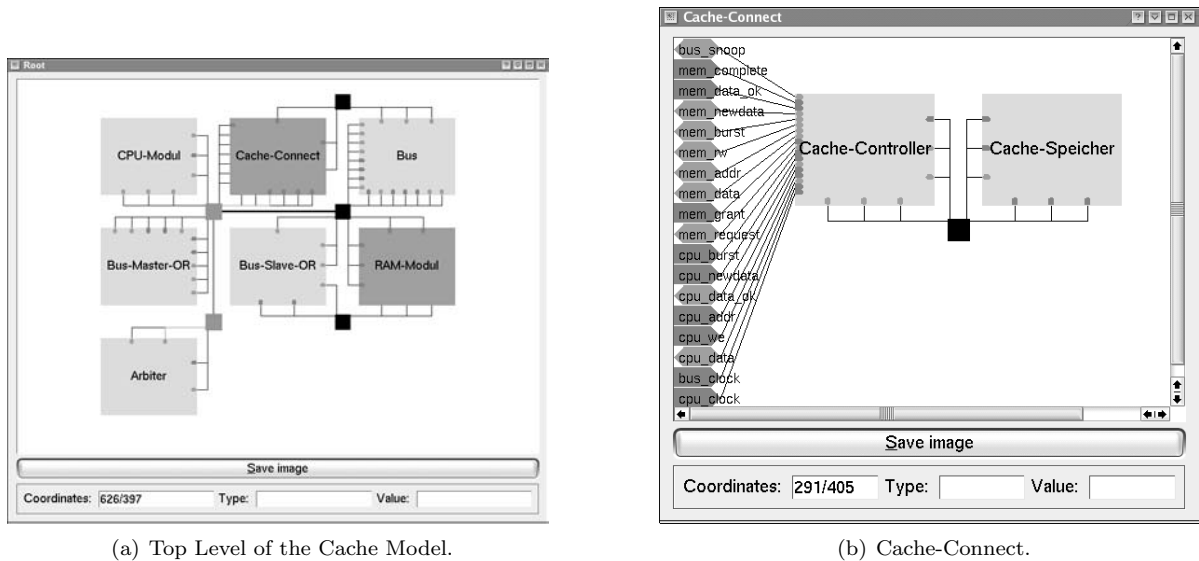


Figure 4: SystemC Model Visualisation of *gSysC*.

value changes can be tracked.

## 4.2 Registration

Hardware modules may have a huge number of interfaces for control and data exchange purposes. So, there are a couple of reasons to leave out some ports and signals respectively in the visualisation. Fewer observed objects ease the overview of the demonstrated design, allow easier debugging by concentrating on the chosen view, and speed up the simulation. All tracked parts of the design must be registered for the visualisation. So, the registration can become a clumsy procedure, especially for large designs. The programmer has to add a code line per module or port that activates *gSysC* features for this object and integrates it into the hierarchical structure of the design. The provided macros reduce the effort to a minimum:

- `REG_MODULE(module, name, parent)` registers the module on the subsequent level of `parent`. Root-level modules are indicated by the `NULL` pointer. The `name` is used in the visualisation.

- `REG_PORT(port, module, signal)` activates the visualisation of the used ports in the SystemC design. The `port` of `module` is connected to the `signal`. More distinguished macros such as `REG_IN_PORT(port, module, signal)`, `REG_OUT_PORT(port, module, signal)`, and `REG_INOUT_PORT(port, module, signal)` including the direction of the ports are available as well.
- `RENAME_SIGNAL(object, name)` and `RENAME_PORT(object, name)` allow the programmer to give signals and ports self-documenting names.

Assuming that all modules, ports, and signals are supposed to be visualised, the registration process could be performed in a preprocessing step. But, at the moment, there has not yet been provided any support tool.

## 5 CASE STUDY

The hardware design of this SystemC model consists of a CPU directly connected to a memory

cache and bus modules. The cache sends to and receives data from the RAM module using a simple bus. Figure 4(a) shows all modules of the top level model. Here, one can see how the modules with ports and interconnections are displayed. The place and route strategy is simplified using clustered signals with central crossing points. During the simulation used signals, ports, and modules may be highlighted. Figure 4(b) discloses the module 'Cache-Connect' of figure 4(a). It is made of two linked modules unseen on the top level view and has a number of in and out ports shown on the left side. If the view of a module is opened its interior behaviour is highlighted as well during the simulation.

The simplified program code of the SystemC model presented here shows the important parts of applying *gSysC*:

```
#include "gsysc.h"
#include ...

int sc_main(int argc, char* argv[])
{
    sc_clock cpu_clk("CPU-Clock");

    // signal declarations
    sc_signal<sc_bv<32> > addr_sig;
    sc_signal<bool> we_sig;
    ...
    cache_connect* c;
    ...

    REG_MODULE(c, "Cache-Connect", NULL);
    REG_MODULE(c->ctrl, "CController", c);
    REG_MODULE(c->memory, "CMemory", c);

    sc_signal<bool> bus_clk_sig;

    bus_bus b("bus");
    b.m_dt(or_mb_dt);
    b.clk(bus_clk_sig);

    REG_MODULE(&b, "Bus", NULL);
    REG_INOUT_PORT(&b.m_dt, &b, &or_mb_dt);
    ...

    c->cpu_clk(cpu_clk);
    c->bus_clk(bus_clk_sig);
}
```

Port name	Value	ID
1 cpu_addr	00000000000000000000000000000000	136372760
2 cpu_newdata	0	136373616
3 mem_request	000	136374400
4 mem_burst	011	136376704
5 mem_complete	0	136378104
6 mem_grant	0	136374904
7 mem_newdata	1	136377136
8 mem_data_ok	0	136377712
9 mem_rw	1	136376152
10 mem_addr	00000000000000000000000000000000	136375688
11 cpu_burst	001	136374008

mem\_data Add Remove

Figure 5: Table of Signals for Value Tracking.

```
REG_IN_PORT(&c->cpu_clk, c, &cpu_clk);
REG_IN_PORT(&c->bus_clk, c, &bus_clk_sig);

bus_master_or mor("master_or");
REG_MODULE(&mor, "Bus-Master-OR", NULL);
...

return 0;
}
```

First, clock and signals for module interconnection are declared. Then, the modules are defined and their ports are connected to the signals. Last, the module and its ports are registered. Here, `cache_connect` is built of a controller and memory so that these modules are registered for a subsequent level. These three steps are done for all modules. After the declaration, connection, and registering phases, the usual SystemC code which is not shown here is required.

One can keep an eye on port and signal values utilizing their property windows or watching the full port/signal list. A snapshot of this list is shown in figure 5.

In combination with the existing opportunities such as VCD trace files and text messages, SystemC becomes with *gSysC* a more powerful tool for debugging and functional verification. Nevertheless, the costs of *gSysC* are partially high. The simulation run-time can be slowed down about 50 per cent depending on used options. Because of the opportunity to remove *gSysC* from the SystemC-model code, long-term simulations without GUI are performed without

any performance loss.

## 6 CONCLUSION

*gSysC* is a GUI extension for SystemC based on Qt, a platform independent GUI library for C++. The shown extension does not alter the SystemC library including the simulation kernel. The features for graphical representation are introduced by macros and redefined functions overloading but calling the ones provided by SystemC. As expected, the performance decreases by about 50 per cent in several cases. The library is open source and can be found at the web pages of the Institute of Computer Engineering, University of Lübeck ([www.itl.uni-luebeck.de](http://www.itl.uni-luebeck.de)).

## REFERENCES

- [Cadence 2005] Cadence Design Systems Inc., *Incisive Unified Simulator*, Datasheet 5418C 04/05, 2005.
- [Charest et al. 2001] Charest, L.; M. Reid; E.M. Aboulhamid; and G. Bois. 2001. *Methodology for Interfacing Open Source SystemC with a Third Party Software*. Proceedings of Design Automation and Test in Europe Conference & Exhibition, 16-20, Munich, Germany.
- [Eibl 2004] Eibl, C.J. 2004. *gSysC – Visualisierung von SystemC-Projekten*. Student Project, Institute of Computer Engineering, University of Lübeck, Germany.
- [Große et al. 2003] Große, D.; R. Drechsler; L. Linhard; and G. Angst. *Efficient Automatic Visualization of SystemC Designs*. Forum on Specification & Design Languages, Frankfurt, Germany.
- [Prosilog 2005] Prosilog, *Magillem*, Product Brief, 2005.
- [Reid et al. 2001b] Reid, M.; L. Charest; M. Aboulhamid; G. Bois; and A. Tsikhanovich, 2001. *Implementing a*

*Graphical User Interface for SystemC*. Proceedings of the 10th International HDL Conference, 224-231, Santa Clara, CA, USA.

[SystemC 2002] Open SystemC Initiative (OSCI): *SystemC Version 2.0.1 User's Guide*. Technical Report, 2002.

[Trolltech 2002] Trolltech AS. *Qt 3.1*. Whitepaper, 2002.