ABSTRACT
Nowadays SystemC plays an important role in digital system design. This C++ class library provides the necessary constructs to model system architecture including hardware timing, concurrency, and reactive behavior missing in standard C++. The SystemC framework offers also simulation kernel to simulate SystemC Models but without a graphical user interface (GUI). This framework can be in some way integrated into an existing tool using then its GUI, as it was done by leading EDA (Electronic Design Automation) companies, but there is also the possibility to extend the framework itself. This approach was used and is presented in this paper. We propose the extensions of SystemC library to enable graphical representation of user’s structural model and graphical presentation of simulation results in conjunction with the model schematic visualization. The resulting schematic view is clear, and easy to understand. Together with built-in simulator it is a powerful tool for structure verification and debugging.

Categories and Subject Descriptors
B.6.3 [Logic Design]: Design Aids – hardware description languages, simulation, verification.
B.5.2 [Register-Transfer-Level Implementation]: Design Aids – hardware description languages, simulation, verification.

General Terms
Design, Languages, Verification.

Keywords
C++, digital systems, discrete event simulation, hardware description language, hardware design, hardware design visualization, hardware verification, SystemC, system modelling.

1. INTRODUCTION
SystemC is an ANSI standard C++ class library for system and hardware design. The general purpose of SystemC is to provide a C++-based standard for designers and architects who need to address complex systems that are a hybrid between hardware and software [5]. Just like Hardware Description Languages (HDLs) e.g. VHDL or Verilog, SystemC can be used for hardware design. However, SystemC provides also a mechanism for managing complex systems involving large numbers of components including software which is not available in traditional HDLs.

The C++ language was chosen not only because it provides the possibility to extend the language without adding new syntactic constructs - through classes, but also to allow designers to use the familiar C++ development tools to develop both hardware and software components as well as interfaces [7]. Many EDA (Electronic Design Automation) companies integrated SystemC support into their EDA tools like for example System Studio or Discovery Verification Platform by Synopsys, Active-HDL by Aldec or ModelSim by Mentor Graphics. Most of the EDA tools provide user-friendly graphical user interface (GUI) and often also supporting tools like structure editor, visualization tool or test bench generator. However, this is not the case of common C++ development environments like e.g. Microsoft’s Visual Studio. A light-weight cycle-based simulation kernel, OSCI reference simulator, is an integral part of SystemC framework. It offers fast simulation but without any GUI and restricted simulation control. Standard simulation output is also not available, so the users have to program in C++ language not only the test benches but also the way the simulation results should be displayed. Neither there is a possibility to display structural SystemC description in a graphical way.

To overcome these drawbacks of common C++ development environments a solution opens up based on the same principle – the extension of C++ by means of SystemC library. The idea was to extend the SystemC library by means of new templates, classes and functions to support GUI, visualization of SystemC model structure and simulation.

2. RELATED WORK
Before designing the new features of SystemC framework the available software tools have been studied. There are several commercial and non-commercial environments supporting SystemC.

For example well known and popular simulator ModelSim by Mentor Graphics [6] offers all necessary functions. It is easy to use and the simulation outputs are clearly shown as waveforms of selected signals. As an alternative, the Aldec’s simulator Riviera-PRO or VCS functional verification tool by Synopsys can be
chosen, which have much in common with ModelSim. Most of them are able to simulate also mixed-language designs. Many commercial environments (e.g. Visual Elite, System Studio, Active-HDL) integrate other tools with functional simulator including structural editors and visualization tools.

Visual SC Designer by Oustech [8] is an impressive plug-in for Microsoft Visual Studio. This commercial extension of traditional C++ development environment corresponds to one of the basic SystemC ideas. Menus, toolbars and schematic model view simplify SystemC design creation as much as possible. Visual SC Designer is well integrated into the development environment and easy to use. However, it provides only the basic visualization facilities and the extension possibilities are very limited.

Several non-commercial approaches to SystemC GUI development have been published. They are based on open SystemC library using one of the two approaches - SystemC library modification and extension (e.g. [3][9]) or SystemC library interfacing with external GUI [1][4][10][11].

One of the first SystemC graphical user interfaces was based on Qt GUI library for C++. This was tightly-coupled with SystemC simulation kernel to enable simulation control. An interesting tool represents SyCe [3] – an integrated environment for SystemC system design, developed at the University of Bremen. It enables schematic visualization of highest level of hierarchy, or module structure respectively, cross-probing between schematic view and source code, as well as some other useful functions.

A different approach was adopted by Eibl et al. from University of Lübeck [1][4]. The gSysC library based on Qt graphical library was loosely-coupled with SystemC by means of wrapping functions to allow for visualization and simulation control of SystemC models. However, to visualize the components they have to be registered with gSysC library which complicates the visualization. The ports and signals values are displayed in a table manner which is not the best solution.

None of the above mentioned approaches offers the schematic model view combined with simulation. This was the reason why we decided to implement this requirement to our solution.

SystemC is completely defined by IEEE 1666 standard [5]. There are three basic structural components:

- Modules are the main structural building blocks that may contain other components.
- Ports, as module interfaces, are providing connection to another component.
- Channels, as module interfaces, are providing connection to another component.

Behaviour of components is described by means of processes: methods and threads. These may contain any kind of C++ code. Binding structural and behavioural components together we are able to create various systems.

The designed solution, called SystemC+Visualizer, offers the combination of SystemC model structure visualization and simulation results visualization. It was designed as a static library. That means to be functional the SystemC design has to be compiled and linked together with our library generating thus the the unique SystemC executable for each user’s design.

Another popular and widely used method for source code processing is parsing of user files, creation of intermediate files like XML (Extensible Markup Language) and simulation [2]. This is more suitable for languages that cannot be simply executed. Such a method is not so efficient for SystemC. User’s C++ files can be easily transformed into self-simulate executables while an interpreter is needed to simulate parsed XML files. Also we could not be quite sure that all required data would be extracted by a parser. A corrupted structural model might then appear.

That is why we decided to avoid parsing. Therefore we based our framework on an existing SystemC library and we built in the structural model into it. This effort should result in a reliable, easy to use tool which is fully compatible with SystemC standard.

3. FEATURES

The SystemC+Visualizer, as our framework was named, offers two main features: a visualization of SystemC models and a simulation. It is designed as a static library, so users are expected to compile and link their SystemC designs together with our library. That means for each user’s design the unique SystemC executable is created.

3.1 Compatibility mode

Since we cannot be quite sure that every SystemC design will be compatible with our extensions, the possibility to run a SystemC design without the extensions is required. That is why the extensions were implemented as an optional feature. The user can choose which version of SystemC will be used: the official OSCI (Open SystemC Initiative) release [7] or our modified version with graphical user interface (GUI) and schematic view. Without any change in user’s code an original SystemC application will start. To use the extended version it is just necessary to call vis_enable() function prior to the first sc_start(...) function call.

3.2 Visualization

The transformation of SystemC model into the graphical model is fully automatic. It is executed at the application start-up. Depending on the model size this takes a few seconds. When the application appears, the top components are shown. The fragment of a simple SystemC model and its graphical representation is given in Figure 1.

The structural components are represented as follows:

- Modules as rectangles. The user can change their position and size.
- Ports as small fixed size rectangles with arrow inside according to the data flow.
- Channels as lines between ports. The nodes are used for channel shaping - they can be moved, added or removed according to the user’s needs.

The object labels match with the object names in user’s source code. If not set the unique names are generated. Another important feature is a layout algorithm. The implemented algorithm places components in a way to avoid conflicts and to
make schematic view well arranged and easy to understand. The currently implemented layout algorithm can be called “bus layout” since it places all the channels into the middle of the schematic view, creating thus a kind of “bus”. The channels are surrounded by two rows of modules – one on the top and the other one at the bottom. Although the generated layout is usually well organized and easy to read there is a possibility to rearrange the modules and channels manually. Of course the interconnections themselves can’t be altered. The visualization algorithms and data structures are integrated into the designed Visualizer component.

3.3 Simulation
For the simulation purpose the default SystemC simulation kernel called OSCI reference simulator is used. However, a few changes have been designed and implemented into it. The break points were inserted into the simulation loop to enable simulation control. Run/Continue and Stop commands are the control commands available. All simulator outputs are recorded into a data structure. The stored values referring to the selected discrete time are presented in the schematic view. The ports and channels are coloured according to the value and the equivalent alphanumeric value is shown (See Table 1). For example, an unspecified value (often represented as an error e.g. bus conflict) on the port is represented by red coloured port and ‘X’ character.

<table>
<thead>
<tr>
<th>Value</th>
<th>Color</th>
<th>Alphanumeric sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>No value (simulation turned off)</td>
<td>Black</td>
<td></td>
</tr>
<tr>
<td>High impedance</td>
<td>Blue</td>
<td>‘Z’</td>
</tr>
<tr>
<td>Low</td>
<td>Green</td>
<td>‘0’</td>
</tr>
<tr>
<td>High</td>
<td>Light orange</td>
<td>‘1’</td>
</tr>
<tr>
<td>Other value</td>
<td>Dark green</td>
<td>‘value’</td>
</tr>
<tr>
<td>Unspecified value</td>
<td>Red</td>
<td>‘X’</td>
</tr>
</tbody>
</table>

3.4 Navigation through the structure and time
When the application starts only the top level of the design is visible. To show inner blocks of the module the user is expected to double-click on it. To return to the upper level the user shall press the Backspace key or click the appropriate toolbar button. The users can navigate through the whole structure, but just one level is shown at the moment.

There is a timeline placed at the bottom of the application window that enables discrete time selection. According to the selected time, the values are shown in the schematic view. The time zoom function is implemented too.

3.5 Reliable output
The usage of the existing SystemC structural model and behavioural model executed under OSCI reference simulator is a guarantee of the output correctness. The user can be sure that the displayed schematic view reflects the SystemC IEEE standard such as the outputs of simulation.

3.6 Platform independent components
In fact the SystemC is platform independent. We are trying not to break this quality. The newly programmed code is platform independent too. The GUI and the input-output system are built using GLUT – the OpenGL utility toolkit. There are the versions of GLUT for Microsoft Windows as well as for Unix-like systems.

4. ARCHITECTURE
The SystemC+Visualizer was implemented as a static library. That means the user’s SystemC design files shall be compiled and linked together with our library. Our framework is made up of mostly unchanged SystemC library, the newly designed Visualizer and the graphical library GLUT (See Figure 2).

At the application start-up the Elaboration module is executed. It results in the creation of SystemC model containing structural and/or behavioural model. The model was extended in order to store the Simulation records. These can be exported into a file. The original SystemC library already contains a function to export
the simulation results into the VCD file, which is an unwritten standard for simulation results storage. It was used as the export function basis. The structural model is shared with the Visualizer that converts it into a Graphical model. Every module, port and channel is represented as a graphical object like rectangle, line etc. The generated graphical model shall be observed using a GLUT based User interface. The layout of the whole schematic view can be saved into a file. The visible schematic view is exportable as a bitmap picture.

The Visualizer also provides an interface between SystemC and GUI. The GLUT module provides not only the presentation layer but also the input-output system through which the OSCI simulator can be started. All the simulation outputs are recorded and returned back as a part of schematic view.

5. DATA STRUCTURES
As part of the design a lot of data structures were designed for the application, starting from simple variables up to the complex structures that handle the simulation records and the graphical model.

5.1 Simulation records
The simulation records are integrated into the SystemC structural model. Every port or channel contains one instance of the vis_log class template. A class template is a C++ feature allowing us to record any kind of data including the user defined data types. To present the recorded data these must be converted into a string so if this is not possible the user data are recorded but cannot be shown in schematic view. In fact the vis_log is derived from the vis_log_base class template and these are specialized for type bool, sc_logic and others. For simplification, a very general version of vis_log is described and given in Figure 3.

Every vis_log contains twin arrays. For simulation outputs m_value is used. The simulation time when the values have been recorded is stored at m_time. The arrays are instances of std::vector that simplifies array reallocation. The appropriate data type for the m_value vector is fully automatically selected by the compiler. During the simulation millions of samples can be created. To reduce this amount only the value changes are stored. The last parameter, m_now is an index of currently used record. This accelerates the repetitive access to the same value.

There were some functions, defined to manipulate data. The constructor and insert are clear to understand. A value_type is used for data type resolution. A jump_to provides search for current record according to the simulation time. The rest of functions are used to access the stored data. Their output shall be unformatted data, formatted data (string) or colour code according to the value (See Table 1).

5.2 Graphical model
The graphical model is made up of graphical objects related to the modules, ports and channels. For every module an instance of module_ext (module extensions) is created. It contains module position, dimensions, colour and more. The port parameters like
template <class DT> class vis_log {

protected:
    std::vector <DT> m_value;
    std::vector <sc_dt::uint64> m_time;
    unsigned int m_now;

public:
    vis_log () ;
    bool insert
        (sc_dt::uint64 time, DT value);
    unsigned int jump_to
        (sc_dt::uint64 time);
    DT value_now () ;
    char* string_now () ;
    char* value_type();
    short color_code_now();
};

Figure 3. A very general version of the simulation record data structure – class vis_log.

parent module name, relative position inside the module or rotation are stored at the port_ext. As mentioned before, the channels are like the wires so a channel equivalent graphical object is the wire_ext. It contains at least two wire nodes. The first for the wire begin and the second for its end. The wire is shown as a line leading from one port to the other through the nodes. The wire_ext is only the envelope with a few parameters. The nodes carry all the necessary information including the node position. Binding all these graphical objects together the graphical model equivalent to the SystemC model appears. The graphical model does not contain any variables and none of the software bindings where read and write functions are used to manipulate port or signal data.

6. RESULTS AND EXPERIENCE
The designed framework successfully passed all the tests made up of various SystemC models from simple to more complex circuits, such as CPUs. These models are specified within the original SystemC library or were created for the testing purpose.

6.1 Output correctness
The displayed schematic model and the simulation outputs fully meet our expectations. Binary, integer and floating point values were correctly recorded and displayed. Widely used Mentor Graphics ModelSim simulator was used as a reference. The Exported simulation results from SystemC+Visualizer to VCD file were either displayed directly in GTKWave or converted into the WLF file, displayed in a waveform view and compared to the original ModelSim output. The results were identical.

6.2 Memory consumption
The dependence of memory consumption on the model size was compared for SystemC+Visualizer and the OSCI reference simulator. The basic testing model included 46 modules, 177 ports and 138 signals. The other four models were the 2, 4, 8 and 16 times greater.

The measured data are shown in Table 2 and graphically represented in Figure 4. The amount of used memory in both cases increases linearly depending on the size of the model. The SystemC+Visualizer requires about 18 MB for graphical user interface, graphical model, and other extensions. It is considerably larger amount and grows faster because it contains all the simulation results. However, the increase in memory consumption is very slow. Therefore to fill up one gigabyte the model size of about 50,000 modules 200 000 ports and 150 000 signals, including the simulation data, would be needed.

6.3 Simulation speed
The simulation duration dependence on the model size for SystemC+Visualizer was investigated and compared to the OSCI simulator. The measured values are shown in Table 2. As in the previous case, there is a linear dependence. Compared to the OSCI simulator the simulation is on average eight times slower. This is due to the systematic recording of all outputs and particularly because of the graphical user interface presence. This was shown by an experimental configuration where GUI was not used and the simulation was running three times faster. Compared to OSCI simulator it was only 2.5 times slower.

Table 2. Dependence of memory consumption and simulation time on model size

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Simulator</th>
<th>SystemC Model Size (Basic size: 46 modules, 177 ports, 138 signals)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Basic size</td>
<td>2x</td>
</tr>
<tr>
<td>Memory Consumption</td>
<td>SCV</td>
<td>20 352</td>
</tr>
<tr>
<td>(KB)</td>
<td>OSCI</td>
<td>1 104</td>
</tr>
<tr>
<td></td>
<td>SCV</td>
<td>18</td>
</tr>
<tr>
<td>Simulation Time</td>
<td>OSCI</td>
<td>3</td>
</tr>
</tbody>
</table>

LEGEND:
SCV – SystemC+Visualizer 2.0
OSCI – OSCI Reference Simulator 2.1v1
7. CONCLUSIONS
In this paper a framework for SystemC design schematic view and simulation was presented. It is based on the SystemC library that is slightly modified and extended by the specially designed and implemented Visualizer component and the graphical library GLUT. The new data structures were proposed to handle the simulation records and the graphical model. The functions for schematic view manipulation, simulation control and for simulation records access were also proposed and implemented. It is important to emphasise that the output correctness is guaranteed since it is based on the official SystemC library and uses the original OSCI simulator.

However the SystemC library has been modified, there is always the possibility to access the original version by activating the compatibility mode. Another solution, using parser and intermediate XML files, has been proposed but it is not that effective and reliable.

The proposed application called SystemC+Visualizer is supposed to be used for SystemC design debugging, verification and structure checking. The possibility to export the recorded simulation values into the VCD file we consider another important contribution. Thanks to this the simulation results can be further processed, for example displayed in the waveform view using an external viewer (e.g. GTKWave). In this way a designer can have to his disposition two different views of simulation results at the same time which can substantially ease the debugging process. Especially when locating an issue in structural description the possibility to easily move across the layers of structure together with the values differentiation by colours can speed up the error localization.

Our framework is distributed under the specific SystemC license, meaning it is free for non commercial purposes. In accordance with the license the product remains open for further development. Further adjustments will be focused on increasing the speed of simulation, export speed and the visualization of non-structural elements (e.g. local variables). The algorithm of model objects deployment can also be improved.

The number of SystemC objects is limited only by the amount of available memory. However, we advice the users not to use more than hundred objects per design level. Otherwise the application performance will be rapidly reduced. Thanks to the mentioned characteristics the application is particularly suitable for students and/or beginners in SystemC design. However, it can also be useful in practice especially for structural model debugging. We hope that SystemC+Visualizer will help to propagate the progressive modelling language SystemC.

8. ACKNOWLEDGMENTS
This work was partially supported by Slovak Science Grant Agency (VEGA 1/0649/09 “Security and reliability in distributed computer systems and mobile computer networks”).

9. REFERENCES


