Checking Temporal Properties in SystemC Specifications*

Axel Braun, Joachim Gerlach, Wolfgang Rosenstiel
{abraun, gerlach, rosenstiel}@informatik.uni-tuebingen.de

University of Tübingen
Sand 13, 72076 Tübingen, Germany

1 Motivation

Today's system designs consist of multiple architectural components, software as well as hardware. The ability to specify and verify these systems at a high level of abstraction is a key competence to cope with the increasing design complexity. C/C++-based approaches on system specification and design are becoming more and more important. They provide a common platform for system designers, hardware and software engineers, and allow a high-performant simulation of system's behavior during the whole design process. The leading approach for C++-based system specification is SystemC [1], which is on the step of becoming a de facto standard in industrial system-level design. Generally, within SystemC the testbench of a design will also be specified in SystemC, which results in a tight coupling of the design and the corresponding test environment. On the other hand, only rudimentary testbench support is given in SystemC and sophisticated features of today's testbench environments are missing. The checking of temporal properties is one of the core requirements in the area of functional verification and is not supported by the standard SystemC language. This paper addresses that important problem. It shows strategies for checking temporal properties in a SystemC design in terms of an easy-to-understand application, a traffic light controller.

The paper is organized as follows: Section 2 shows strategies for providing mechanisms for temporal property checking to SystemC. In section 3 the benchmark design, a traffic light controller, which is used to illustrate and evaluate the property checking strategies, is introduced. Section 4 shows the application of the strategies to the benchmark design, and section 5 summarizes the experimental results.

2 Strategies for Temporal Property Checking

There are different strategies to provide extended testbench features to SystemC: They can be implemented directly within the SystemC language, which results in an add-on library to be linked to a SystemC design (in the same way as the SystemC library itself). By this, extended testbench features can be directly used in a SystemC testbench. Following this strategy, an add-on library for temporal property checking is developed for SystemC (see figure 1). This library includes functions for the checking of temporal properties specified in a temporal logic calculus. With these functions, the traces of a simulation can be checked against a specification of a temporal property given in a Finite Linear Temporal Logic (FLTL) [3]. FLTL is a linear time temporal logic that interprets formulas over finite traces and supports quantitative timed operators. This strategy is efficient and keeps the communication effort low, because it is handled completely inside SystemC.

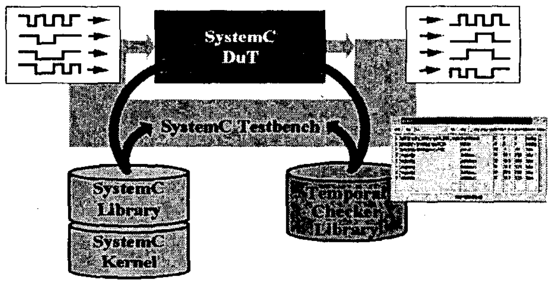
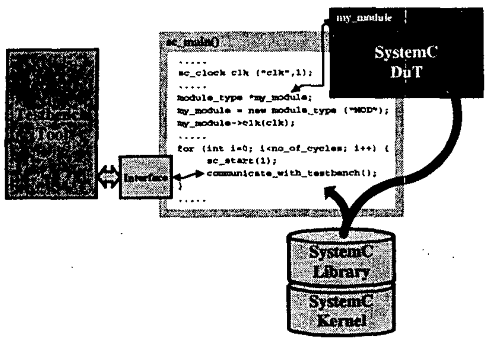


Figure 1: Add-on Library Strategy

Another strategy to provide extended testbench features to SystemC is to interface SystemC with an existing external testbench environment (in our approach, TestBuilder [2]). In this approach, the SystemC design and testbench environment are connected by an (automatically generated) interface module. The interface module exchanges data between the SystemC simulation and the testbench environment. Following this strategy, advanced temporal checking features of the testbench environment can be applied to the SystemC design (see figure 2).

*This work has been partially by DFG (Deutsche Forschungsgemeinschaft) in the Priority Programs Embedded Systems and Rapid Prototyping under Ro1030/4-2 and 5-2.



```
.....
sc_clock clk ("clk",1);
.....
module_type *my_module;
my_module = new module_type ("MOD");
my_module->clk(clk);
.....
for (int i=0; i<no_of_cycles; i++) {
     sc_start(1);
     communicate_with_testbench();
}
.....
```

Figure 2: Interfacing Strategy

3 SystemC Traffic Light Controller

The strategies outlined in section 2 for providing temporal property checking features to SystemC are demonstrated in terms of an easy-to-understand example given by a traffic light controller for a pedestrian crossing. Two pairs of traffic lights are controlled: The cars' traffic lights for the road and the pedestrians' traffic lights for road crossing. Each pair of traffic lights is controlled by a separate SystemC thread. Pedestrians can request green light for crossing the road by pushing the request button. Figure 3 visualizes the scenario.

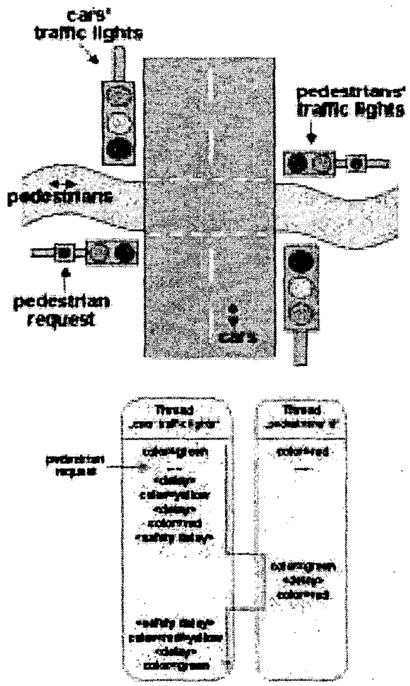


Figure 3: Traffic Light Controller

The pedestrian request is simulated by a SystemC thread. By default the cars' traffic lights are green. If a pedestrian request (`ped_req`) is notified, the cars' traffic lights thread remains green for 5 time steps and then turns to yellow for another 5 time steps and then to red afterwards. After a safety time period of 5 time steps, it notifies the pedestrians' traffic lights thread via signal (`tl_ped_green`) to switch from red to green. After a green phase (20 time steps), the pedestrians' traffic lights turns back to red, and the pedestrians' traffic lights thread notifies the cars' traffic light thread to switch back to green via signal `tl_ped_red`. Therefore, the cars' traffic light thread remains red for a safety phase (5 time steps) and then turns back to red-yellow and finally to green. The cutout of a waveform trace of such a cycle is given in Figure 4, depicting all relevant timing information. This traffic light controller application is quite simple but gives a good example for the following reasons: The SystemC code is short and easy-to-understand but contains some „non-trivial" features that are important from a testbench point of view, and the application allows to specify evident temporal properties to be checked against the SystemC design.

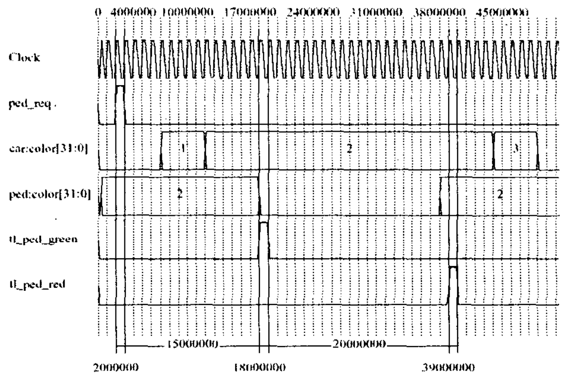


Figure 4: Waveform Trace of a Traffic Light Controller Cycle

4 Checking Temporal Properties

Both stategies for providing temporal property checking features to SystemC outlined in section 2, the add-on library approach as well as the interfacing approach, were implemented. This sections demonstrates the application of these strategies to the benchmark design of section 3. A temporal property check is done using the following set of properties to be proven against the traffic light controller design.

- Property-1: *„pedestrians' traffic lights become green within 15 time steps from pedestrian request"*
- Property-2: *„20 time steps after the pedestrian green release signal, the red acknowledge signal must occur"*

- Property-3: *„the acknowledgement signal that pedestrian traffic lights are switched back to red, must occur exactly after 37 time steps after a pedestrian request arrives“*
- Property-4: *„pedestrian green phase request occurs 10 time steps after a pedestrian request“*

Regarding figure 3, it can be seen that for the traffic light controller, the properties -1, -2, and -3 hold. Property-4 does not hold, because this condition will never be satisfied by the model: The request for a green phase at the pedestrian's traffic light will not arrive before 15 time steps after the pedestrian request. This temporal property will be used to show the behavior of both approaches of an assertion that is not valid. The preparing steps to set up a temporal check differ significantly for the strategies of section 2. In the following, the preparing steps are shown in detail for both strategies in terms of property3 and -4. The results of the application of both strategies to the properties -1 to -4 are compared and discussed in section 5.

4.1 Add-on Library Approach

For the add-on library approach, the temporal properties have to be expressed in terms of FLTL formulas. Those formulas can be checked using extended SystemC functions provided by the temporal checking add-on library. This requires some effort in coding the temporal properties to FLTL, but minimum changes to the original SystemC testbench. Figure 5 shows parts of the SystemC design, extended by calls for functions of the temporal checking library (printed in bold letters). In our example, the temporal check is located in the top level module of the SystemC design. Nevertheless, the assertions can be placed everywhere in the design code. After the initialization function of the temporal checking facility (`sc_init_check`) specifying the positive clock edge to be the linear time basis for the check, and the internal cache size for the FLTL formulas, assertions can be placed in the code. The `sc_check` statement takes a FLTL formula to be checked during the simulation run.

In figure 5, the reactivity of the `ped_red` signal (acknowledgement signal that the pedestrian traffic light is red again) to the `ped_req` signal (pedestrian request for green phase) is checked. It must be ensured that the acknowledgement occurs exactly 37 time steps after the request. The example additionally shows the method for accessing signals: The pedestrian request signal, that is defined within the top-level module, can be accessed directly, whereas the `ped_req` acknowledgement signal, that is defined in the traffic light controller sub-module `tl`, can be reached straightforward using the '`->`' operator.

```
int sc_main(int ac, char *av[])
{
  ...
  //Signals
  sc_signal<bool> ped_req;
  //Clock
  sc_clock clk("clock",1,0.5);
  ...
  // Initialize temporal checker tool
  sc_init_check(clk.pos(),100000);
  ...
  //Modules
  tl_ctrl *tl;
  tl_tb   *tl_t;
  tl    = new tl_ctrl("tl_ctrl");
  (*tl)(ped_req,clk);
  tl_t = new tl_tb("tl_tb");
  (*tl_t)(ped_req,clk);
  sc_check(G (prop(ped_req) >
          X (37, prop(tl->tl_ped_red))));
  ...
  sc_start(20000);
  ...
  // stop temporal checking
  sc_quit_check();
  return 0;
}
```

Figure 5: Temporal Checker Code

To invoke the temporal checking feature not only for valid situations, we also applied property-4, which does not hold. The feedback of the temporal checker tool is given in Figure 6: The `ped_green` signal does not occur 10 time steps after the arrival of the pedestrian request signal, so a failure message is generated. The traffic light model reports its current state by displaying the status for red, yellow and green `[re] [ye] [gr]` for the road or simply red and green `[re] [gr]` for pedestrians.

```
...
2262     Pedestrian request
2268     Traffic light (car):     [  ][ye][  ]
2273     Traffic light (car):     [re][  ][  ]
2279     Traffic light (ped):     [  ][gr]
sc_check:tl_main.cc:49:0:G ((prop({0})>
         X (prop(tl_ctrl.{2})))):false:2.3e-06 sc_sec
SimErr
2315     Traffic light (ped):     [re][  ]
2321     Traffic light (car):     [re][ye][  ]
2326     Traffic light (car):     [  ][  ][gr]
...
```

Figure 6: Temporal Checker Error Feedback

4.2 Interfacing Approach

For the interfacing approach, the temporal properties have to be specified within a TestBuilder test environment using the mechanisms provided by TestBuilder. The test environment has to be interfaced to the SystemC design. This is supported by an automated interface generator, which builds a corresponding shell around the SystemC design that allows TestBuilder to access to internal signals of the SystemC design, for example. This approach allows to use advanced test features of TestBuilder, but requires some interfacing effort. In a first step the sc_main function is automatically generated by parsing the SystemC description of the traffic light example. The second step is to specify the temporal properties to be checked. In opposite of the add-on library approach described above, this specification has to be done in a separate module for the TestBuilder C++ environment. This code is compiled and linked together with the automatically generated interface module and the original SystemC design. Figure 7 depicts the code for the temporal properties written in TestBuilder style. The checks are formulated similarly to the temporal checker tool after the initialization. The temporal property is stated by specifying the appropriate parameters of a property object `tbvPropertyT`. The actual property is specified using the `setType()` function and the `setFulfill()` method (last line in figure 7).

```
void tbvMain()
{
  basicTvmT *tvmP;
  // begin test
  tbvOut << "C++: Starting test 'tl_test' at t="
         << tbvGetInt64Time()
         << " " << tbvGetSimTimeUnitP() << endl;
  tvmP = (basicTvmT*)tbvTvmT::getTvmByInstanceNameP("tl
...
tbvPropertyT check4("red ackn within 38 steps from ped.
check4.setType(tbvPropertyT::EVENTUALLY);
check4.setEnableTrigger(edge);
check4.setEnable(tvmP->ped_req());
check4.setWindowEnd(cycle * 38);
check4.setDisableTrigger(edge);
check4.setDisable(tbvExpressionT(FALSE));
check4.setFulfillTrigger(edge);
check4.setFulfill(tvmP->ped_red());
...
// let the setup functions finish
tbvWait(1);
// enable constraint checking
...
check4.setActive(TRUE);
...
```

Figure 7: TestBuilder Code

Additionally, the conditions where and when this (temporal) property has to be checked are specified. This is done by fixing conditions for observation windows. Within these observation windows the temporal property is checked during simulation run. The observation window starts exactly when a pedestrian request signal (`ped_req`) occurs and ends 38 time steps after this event, and the temporal property is checked within this period. Compared to the add-on library approach, the specification of the observation window corresponds to an appropriate placing of a temporal checking library function in the SystemC code.

In case of the check of property 4, the designer gets a `'PROPERTY_FAILURE'` message from TestBuilder during the simulation is running, similar to the temporal checker tool. Figure 8 shows such a feedback. The specification of the observation window ensures that every occurrence of this failure is reported.

```
...
4495     Pedestrian request
4525     Traffic light (car):     [  ][ye][  ]
4550     Traffic light (car):     [re][  ][  ]
4580     Traffic light (ped):     [  ][gr]

*** WARNING From TestBuilder, exceptionType= 'PROPERTY_
Property 'red within 38 time steps from pedestrian requ
         TVM 'top_sc.tl_test' failed at time 4690.
*** End of message
SimErr
4760     Traffic light (ped):     [re][  ]
4790     Traffic light (car):     [re][ye][  ]
4815     Traffic light (car):     [  ][  ][gr]
...
```

Figure 8: TestBuilder Error Feedback

5 Results

Add-on library approach and interfacing approach were compared in terms of simulation speed, handling and usability. Both approaches are functional verification methods based on simulation, in case of SystemC based on simulation by an executable specification. Table 1 shows the simulation performance of both approaches for the properties specified in section 4.

Used temporal property	Temporal Checker	TestBuilder
Without Check	0.132	0.132
Property-1	0.181	0.590
Property-2	0.169	0.666
Property-3	0.169	0.638
Property-4	0.144	0.788

Table 1: Execution Times for Temporal Checks

The interfacing approach introduces more communication and much more effort for synchronization than the add-on approach, which integrates the temporal checking features directly in SystemC. This results in a 1.26 times (average) higher run time in case of the add-on library approach and a 5.08 times (average) higher run time in case of the interfacing approach for the check of a temporal property. The differences between the four temporal properties checked in our experiments are similar within both strategies. The complexity of all of the four temporal properties is also quite similar.

Concerning the handling and ease-of-use, both approaches show advantages and disadvantages. The add-on library approach allows a simpler specification of the temporal properties compared to TestBuilder. All specifications of the add-on library approach are done within the SystemC code of the design and they can be located exactly at the position in the design code where they have to be checked. Whereas TestBuilder is separated from the code of the design and requires a separate specification. This separation also requires the coding of additional conditions where and when a temporal property is being checked. If a large number of complex temporal properties have to be checked, the mixture of temporal verification code and design code may lead to an entirely more complex code.

In opposition to that, the TestBuilder approach requires a very careful specification of all conditions when a certain temporal property is being checked. If one of those conditions, e.g. the observation window, is incorrect, the related property could be never checked, possibly without any notice to the designer.

6 Conclusion

In this paper two basic strategies for providing temporal checking features to SystemC were analyzed. Therefore, an add-on library approach, which results in a temporal checker tool that allows the observation of temporal properties within a SystemC design, was implemented. Secondly, an interfacing approach between SystemC and an existing testbench environment, TestBuilder, was implemented, which makes the advanced temporal checking features of TestBuilder available for the verification of a SystemC design. Both temporal property checking approaches were compared in terms of an easy-to-understand benchmark design, given by a traffic light controller.

As our experiments show, the add-on library approach is about four times faster concerning simulation speed than a corresponding check using the interfacing approach. Both strategies differ in terms of handling and application: Using the temporal checker approach, the properties can directly be placed at a particular position in the SystemC design code, whereas the interface approach requires a separate and more complex specification of the design parts where the property has to be checked.

7 References

[1] Open SystemC Initiative: *"SystemC Version 2.0 beta-1"*, User's Guide, http://www.systemc.org, 2001.

[2] Cadence Design Systems, Inc.: *"TestBuilder User Guide"*, Product Version 1.0, March 2001.

[3] J.Ruf, D. W. Hoffmann, T. Kropf and W. Rosenstiel: *"Simulation-Guided Property Checking Based on Multi-Valued AR-Automata"*, In Proceedings of Design Automation and Test in Europe (DATE), Munich, March 2001.