Appendix A

FDC Language

The following chapter defines the syntax and the (informal) semantics of the FDC language. FDC specifications are the common configuration and implementation approach in the REGATTA analysis framework. They are used to adapt the generic framework components to a concrete language. Moreover, these specifications allow to define various analyses, e.g. implementing arbitrary coding checks.

1 FDC SYNTAX

The notation of FDC specifications is geared to the syntax of the *State Machine Compiler* [Rapp08]. In Figure A.1 an EBNF of the FDC language is given. This allows to describe abstract and configurable analysis tasks.

2 FDC SEMANTIC

The informal semantic is specified as follows:

- A FDC specification divides into seven sections:
  - The *require* section contains a list of C++ class header files to be included.
  - The *import* section contains a list of imported variables.
  - The *variable* section consists of any number of locally used variables.

```
fdc_spec      ::= {fdc_desc}+
fdc_desc      ::= t_FDC ident '{' {require_decl}* {import_decl}* {var_decl}*
                                  [prior_decl] [init_decl] {func_decl}*
                                  {state}+ '}'
require_decl  ::= t_REQUIRE <include_path>
import_decl   ::= t_IMPORT type ident ';'
var_decl      ::= t_VAR var_type ident ';'
prior_decl    ::= t_PRIORITY number ';'
init_decl     ::= t_INIT '{' action '}'
func_decl     ::= t_FUNCTION ident '{' action '}'
state         ::= state_name [entry] [exit] '{' {trans}+ '}'
trans         ::= proxy [ '[' guard ']' ] state_name [ '{' action '}' ]
proxy         ::= ident
state_name    ::= ident | t_NIL
entry         ::= t_ENTRY '{' action '}'
exit          ::= t_EXIT '{' action '}'
type          ::= t_INT | t_STRING | t_DOUBLE | t_BOOL
ident         ::= [a-zA-Z_][a-zA-Z0-9_]+
var_type      ::= // any correct C++ type
action        ::= // correct C++ code + Regatta API
guard         ::= // correct C++ Boolean expression
```

Figure A.1: EBNF syntax of the FDC language

  * The *priority* section specifies the particular priority of the FDC specification.
  * The *init* function represents the FDC initialization routine.
  * The *function* section defines C-like functions which can be called from any action code.
  * The *FSM* section describes the underlying FSM used to control the analysis steps.

- Each `require` statement specifies the name of a header file to be included. It is used if the action code references functionality that is not included per default by the C++ compiler. Include files are searched in all REGATTA source code paths and the standard include paths.
- The `import` statement declares a variable that has to be externally set in the FDC XML configuration file with respect to a concrete analysis task.
  NOTE: Imported variables are read only in all action code blocks otherwise the compiler reports an error.
- A `var` statement declares a local variable that can be read or written in any action code block of the FDC specification.
- Allowed types in an `import` and `var` declaration must satisfy C++ syntax for type declarations.

- The priority of an FDC can be specified by the `priority` statement. This statements defines a call sequence if two FDCs have registered for the same elements of the created EST (Definition 4). As higher the given priority value is as later the FDC is called.
- The `init` function is executed just before syntax analysis starts. It may contain arbitrary initialization code for the FDC using correct C++ syntax.
- A `function` declaration defines a function with no return value and an empty argument list. It may contain any correct C++ code and is called within the transition action code segments like a usual C function.
- The actual FSM specification consists of any number of states and an arbitrary number of transition between these states.
  * A *state* specifies any number of transitions that switch the FSM into a new state. The special state name `nil` is reserved for a transition leading back to the current state. A state may have one `entry` and one `exit` function. Code in the `entry` function is executed when the state is newly entered. The `exit` code block is evaluated when the current state is left. The lexically first state implicitly represents the initial start state.
  * A *transition* consists of a *proxy symbol*, a *guard*, and a *target* state. A proxy symbol is a wildcard for elements of the analysis alphabet of the given context-free grammar. Elements of a proxy symbol trigger the particular transition. The target state has to be the name of another existing state inside the FDC specification. A conditional transition can be implemented by using an optional guard. The guard is evaluated during FDC execution and only if it evaluates to *true* the transition is triggered. A guard must be a valid C++ Boolean expression and may contain queries of the current state of another FDC specification. Such a query has to meet the following syntax:

```
// <FDC name>. name of the target FDC
// <queried state>. existing state in the target FDC
<FDC name>->getState() == <queried state>
```

  * An action code block has to contain any correct C++ code. Per default, the user has the full access to the REGATTA API using two implicit variables: `AM_AnalysisMngr* am` which allows to access any framework components, and `FE_Data* data` that refers to the current language-specific EST element. The fol-

lowing example sketches the access to a meaningful token (see Section 1.1 on page 34):

```
function check
{
   // retrieve the next meaningful token
   FE_Token* tok = data->getNextLRT();
   if (!tok) {
      return;
   }
   std::count << "Token name: "
              << tok->getName() << std::endl;
}
```

Appendix B

Debug Pattern Catalog

At first, this chapter describes the general description format used for the debug pattern catalog. Subsequently, two debug patterns are introduced, exemplarily.

1 GENERAL FORMAT

Each debug pattern is described in the following common format. Such a format is already used by Gamma et al. [GHJV95] to describe their object-oriented design pattern catalog:

- *Pattern name.* The format description is introduced by a short and intuitive name for the debug pattern.
- *Motivation*. The existence of the debug pattern is motivated in this paragraph. Here, a representative problem is illustrated.
- *Symptom*. This paragraph describes the failure symptom which is typical for the error the pattern is aimed at.
- *Assumption*. The assumption specifies the problem or certain conditions, i.e. a specific architecture or component composition in the design, that probably causes the observed symptom.
- *Participants*. All participating system-level debugging commands and their specific responsibilities in the pattern context are described in this paragraph.
- *Debug procedure.* The debug procedure is described by a flowchart to formally document the necessary steps to isolate the error.

- *Example*. This paragraph sketches a typical situation where the pattern helps the designer to find and correct an error.
- *Related patterns*. Related patterns also match the observed failure symptom. These patterns should be tried if the failure-causing defect could not be found using the current pattern.

2 COMPETITION PATTERN

Motivation

A typical SystemC design consists of different communicating processes that are running in parallel and that are synchronized over simulation time. The execution order of processes is not determined during a simulation delta cycle. This introduces nondeterminism into the simulation process and may lead to unpredictable and wrong design behavior. Examples for such wrong design behavior are race conditions or deadlocks.

Symptom

The design is simulated (with the same inputs) and unexpectedly produces different output or erroneous results, possibly in some minor cases. This situation could be caused by a nondeterministic execution of parallel processes in the same delta cycle or by an erroneous process synchronization.

Assumption

There are at least two processes concurrently competing for the same (shared) resource such as a signal or a bus.

Participants

dp_sense	return all processes triggered by the same event
lpt_rx	review the sensitivity list of the given process
lst	explore the source code of a process
lse_rx	get the correct hierarchical event name

Debug Procedure

Figure B.1 shows the debug procedure of the COMPETITION pattern.

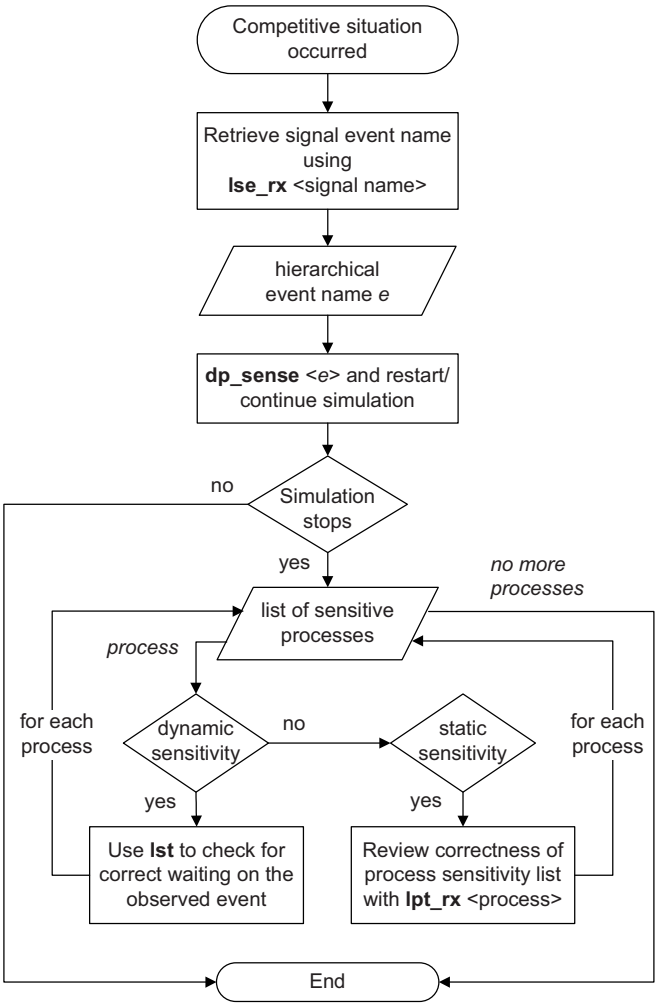

Figure B.1: COMPETITION pattern flowchart

Example

Problem. Figure B.2 sketches a situation where two thread processes *top.bfm._fsm_update* and *top.bfm._fsm_rst* write to the same signal *top.ctrl_w*. Both threads are activated by the ready signal *top.rdy_1*. Since the SystemC simulation kernel does not define a deterministic order of thread activations inside a delta cycle, a race condition can occur on signal *top.ctrl_w*.

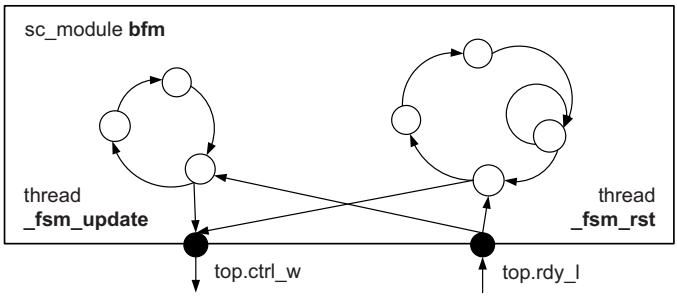

Figure B.2: Exemplary race condition

Debug procedure. Assume that the hierarchical event name of the signal causing the problem is already known. Thus, *dp_sense* is called with it and the simulation is restarted:

```
(gdb) dp_sense "top.rdy_l.m_negedge_event"
*** COMPETITION debug pattern activated
*** restart/continue simulation using run/continue
(gdb) run
```

When the simulation stops *dp_sense* reports two thread processes sensitive to the observed event. For convenience reasons, the debugging environment additionally adds breakpoints at the sensitive processes:

```
*** dp_sense 'top.rdy_l.m_negedge_event'
*** Check competitive situation between sensitive processes
  in module top.bfm
      top.bfm._fsm_update  <static>
      top.bfm._fsm_rst  <static>

*** breakpoints in sensitive processes
  breakpoint at top.bfm._fsm_update
  breakpoint at top.bfm._fsm_rst
```

Knowing that thread *_fsm_update* is correctly activated by the ready signal, the sensitivity list of the statically triggered thread *_fsm_rst* is investigated using *lpt_rx*:

```
(gdb) lpt_rx "top.bfm._fsm_rst"
process top.bfm._fsm_rst sensitive to
    <static> top.fsm_rst_l.m_negedge_event
    <static> top.rdy_l.m_negedge_event
    <dynamic> top.write_tx.m_value_changed
```

The command shows that the sensitivity list falsely includes the ready signal which turns out to be an environment defect.

Related Patterns

LIVELOCK

3 TIMELOCK PATTERN

Motivation

Event-based communication between concurrent processes can often lead to a lock condition. Then, a process is caught in an infinite loop and thus never waits for an event.

Symptom

The simulation infinitely loops or at least appears to do so. Additionally, the simulation time does not proceed.

Assumption

Due to design specification knowledge, the user suspects one or more processes causing the lock.

Participants

dp_timelock	manually look for hanging processes
lst	explore the source code of a process
dstep	proceed simulation step-wise

Debug Procedure

Figure B.3 shows the debug procedure of the TIMELOCK pattern.

Example

Problem. Figure B.4 illustrates a system where a device (*top.i_device*) communicates with a host (*top.i_host*) over a bus. Device and host exchange data using the two signals *top.req_data* and *top.resp_data*. Available data packages are indicated by two ready signals *top.req_ready* and *top.resp_ready*, respectively. The user models the signal interaction in a faulty way. So, the simulation locks.

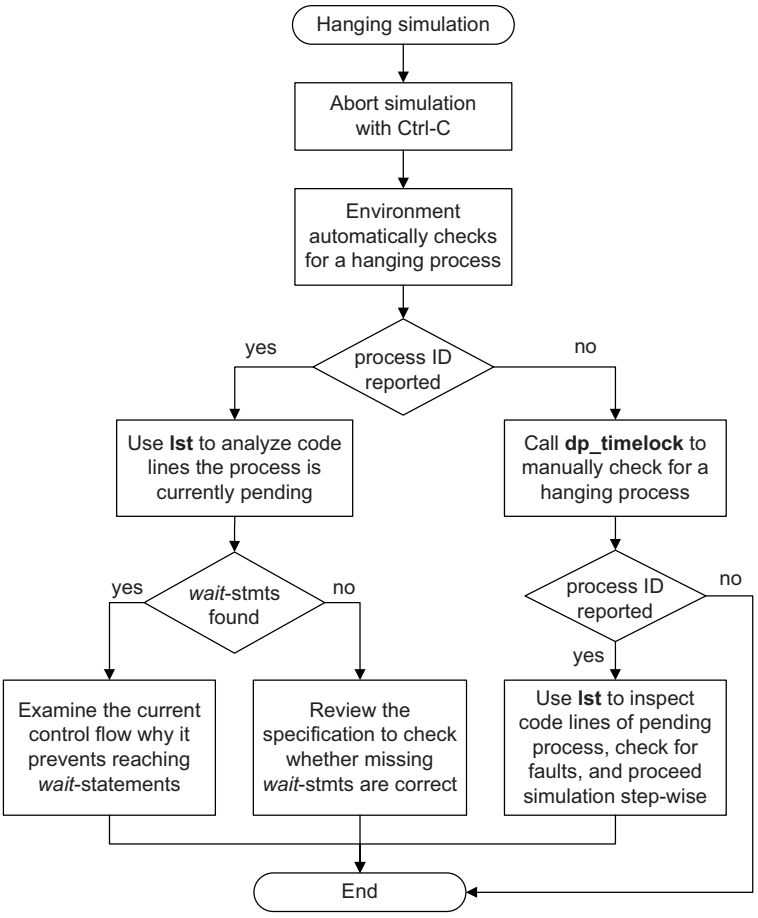

Figure B.3: TIMELOCK pattern flowchart

Debug procedure. The user aborts the simulation by pressing Ctrl-C. It is assumed, that the debugger does not report any process ID. Hence, the *dp_timelock* command is called to manually check for a hanging process:

```
Program received signal SIGINT, Interrupt.
(gdb) dp_timelock
*** following process seems to hang
top.i_device._rx_tx
*** TIMELOCK debug pattern activated
*** call lst 'top.i_device._rx_tx' to examine source code of
pending process
```

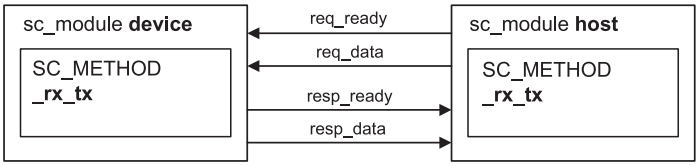

Figure B.4: Exemplary TIMELOCK pattern

The source code of the hanging process is investigated to find the root cause of the observed failure:

```
(gdb) lst 'top.i_device._rx_tx'
--lst: list active source of [c]thread/method---
process top.i_device._rx_tx is currently
 at /home/hld/project/tb/src/device.cpp:254
in device::_rx_tx
254          void device::_rx_tx() {
255              process_req_data(req_data.read());
256              resp_data.write(gen_resp_data());
257              resp_ready.write(~resp_ready.read());
258
```

At first glance, no error is found. Thus, the simulation is proceeded in step-mode using multiple *dstep* commands:

```
(gdb) dstep
*** setting breakpoint(s) at next delta cycle
*** runnable process(es) at delta cycle 3563, @14 ns
   break @ 'host::_rx_tx()' of 'top.i_host._rx_tx'
*** type <continue> to visit breakpoints
(gdb) cont
```

After some more *dstep* commands, the debugger log shows that the processes *top.i_host._rx_tx* and *top.i_device._rx_tx* become mutually active, invoking each other without any simulation time progress. A code review of the affected modules identifies a lack of time-consuming statements within the thread loop.

Related Patterns

DEADLOCK, LIVELOCK

References

[Actis] Actis Design. AccurateC: Static C++ Code Analysis for SystemC. *Actis Design*, Portland, 2008.

[AB+07] A. Aiken, S. Bugrara, I. Dillig, T. Dillig, B. Hackett, and P. Hawkins. An Overview of the Saturn Project. In *Workshop on Program Analysis for Software Tools and Engineering*, pp. 43–48, 2007.

[ABL02] G. Ammons, R. Bodik, and J.R. Larus. Mining Specifications. *ACM SIGPLAN Notices*, Volume 37, Issue 1, pp. 4–16, 2002.

[ABS03] G. Agosta, F. Bruschi, and D. Sciuto. Static analysis of transaction-level models. In *Conference on Design Automation*, pp. 448–453, 2003.

[AF02] T. Arts and L.-A. Fredlund. Trace Analysis of Erlang Programs. In *ACM SIGPLAN Workshop on Erlang*, pp. 16–23, 2002.

[AP+07] N. Ayewah, W. Pugh, J.D. Morgenthaler, J. Penix, and Y. Zhou. Evaluating Static Analysis Defect Warnings on Production Software. In *Workshop on Program Analysis for Software Tools and Engineering*, pp. 1–8, 2007.

[ARL00] D. Abts, M. Roberts, and D.J. Lilja. A Balanced Approach to High-level Verification: Performance Trade-offs in Verifying Large-scale Multiprocessors. In *International Conference on Parallel Processing*, pp. 505–510, 2000.

[ARM] ARM Ltd. RealView Development Suite Homepage. [Online], http://www.arm.com, accessed October 2008.

[Ash06] P. Ashenden. *The Designer's Guide to VHDL*. Morgan Kaufmann, 3rd revised edition, 2006.

[ASU03] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers*. Addison-Wesley Longman, 2003.

[Bra83] D. Brand. Redundancy and Don't Cares in Logic Synthesis. *IEEE Transactions on Computers*, 32(10):947–952, 1983.

[BB+07] J. Bormann, S. Beyer, A. Maggiore, M. Siegel, S. Skalberg, T. Blackmore, and F. Bruno. Complete Formal Verification of TriCore2 and Other Processors. In *DVCon*, 2007.

[BDBK08] D. C. Black, J. Donovan, B. Bunton, and A. Keist. *SystemC: From the Ground Up*. Spinger, 2nd edition, 2008.

[BFF05] N. Bombieri, A. Fedeli, and F. Fummi. On PSL Properties Re-use in SoC Design Flow Based on Transaction Level Modeling. In *International Workshop on Microprocessor Test and Verification*, pp. 127–132, 2005.

[BHJM07] D. Beyer, T.A. Henzinger, R. Jhala, and R. Majumdar. The Software Model Checker Blast: Applications to Software Engineering. *International Journal on Software Tools for Technology Transfer*, 9(5–6):505–525, 2007.

[BKS08] N. Blanc, D. Kroening, and N. Sharygina. Scoot: A Tool for the Analysis of SystemC Models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 467–470, 2008.

[BMP07] B. Bailey, G. Martin, and A. Piziali. *ESL Design and Verification: A Prescription for Electronic System Level Methodology.* Morgan Kaufmann, 2007.

[BNL05] D. Beyer, A. Noack, and C. Lewerentz. Efficient Relational Calculation for Software Analysis. *IEEE Transactions on Software Engineering*, 31(2):137–149, 2005.

[BP+05] D. Berner, H. Patel, D. Mathaikutty, J.-P. Talpin, and S. Shukla. SystemCXML: An Extensible SystemC Front End Using XML. In *Forum on Specification and Design Languages*, pp. 405–408, 2005.

[BR02] T. Ball and S.K. Rajamani. The SLAM Project: Debugging System Software via Static Analysis. In *Symposium on Principles of Programming Languages*, pp. 1–3, 2002.

[Cir04] Collett International Research. *IC/ASIC Functional Verification Study*, 2002/2004.

[CoWare] CoWare Inc. Platform Architect Homepage. [Online], http://www.coware.com, accessed July 2008.

[CD+05] Xi Chen, A. Davare, H. Hsieh, A. Sangiovanni-Vincentelli, and Y. Watanabe. Simulation Based Deadlock Analysis for System Level Designs. In *Design Automation Conference*, pp. 260–265, 2005.

[CDT] Eclipse CDT Project Homepage. [Online], http://www.eclipse.org/cdt, accessed September 2008.

[CF+93] J. Cuny, G. Forman, A. Hough, J. Kundu, C. Lin, L. Snyder, and D. Stemple. The Ariadne Debugger: Scalable Application of Event-Based Abstraction. In *Workshop on Parallel & Distributed Debugging*, pp. 85–95, 1993.

[CGP00] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model checking*. MIT Press, Cambridge, MA, 2000.

[CRAB01] L. Charest, M. Reid, E. Aboulhamid, and G. Bois. A Methodology for Interfacing Open Source SystemC with a Third party Software. In *Design, Automation and Test in Europe*, pp. 16–20, 2001.

[CS06] C. Csallner and Y. Smaragdakis. Dynamically Discovering likely Interface Invariants. In *International Conference on Software Engineering*, pp. 861–864, 2006.

[CS+06] E. Cheung, P. Satapathy, Vi Pham, H. Hsieh, and Xi Chen. Runtime Deadlock Analysis of SystemC Designs. In *IEEE International High-Level Design Validation and Test Workshop*, pp. 187–194, 2006.

[Das00] M. Das. Static Analysis of Large Programs (Invited Talk) (Abstract only): Some Experiences. In *ACM/SIGPLAN Workshop Partial Evaluation and Semantics-Based Program Manipulation*, p. 1, 2000.

[Det96] D.L. Detlefs. An overview of the extended static checking system. In *Formal Methods in Software Practice*, pp. 1–9, 1996.

[Don92] C. Donnelly. *Bison: The Yacc – Compatible Parser Generator*. Free Software Foundation, 1992.

[DAC99] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in Property Specifications for Finite-state Verification. In *International Conference on Software Engineering*, pp. 411–420, 1999.

[DE88] M. Ducassé and A.-M. Emde. A Review of Automated Debugging Systems: Knowledge, Strategies and Techniques. In *International Conference on Software Engineering*, pp. 162–171, 1988.

[DF04] R. Drechsler and G. Fey. Design Understanding by Automatic Property Generation. In *Workshop on Synthesis And System Integration of Mixed Information Technologies*, pp. 274–281, 2004.

[DG+05] A. Dahan, D. Geist, L. Gluhovsky, D. Pidan, G. Shapir, Y. Wolfsthal, L. Benalycherif, R. Kamidem, and Y. Lahbib. Combining System Level Modeling with Assertion based Verification. In *International Symposium on Quality of Electronic Design*, pp. 310–315, 2005.

[DG02] R. Drechsler and D. Große. Reachability Analysis for Formal Verification of SystemC. In *Euromicro Symposium on Digital System Design*, pp. 337–340, 2002.

[DLS02] M. Das, S. Lerner, and M. Seigle. ESP: Path-Sensitive Program Verification in Polynomial Time. In *Conference on Programming Language Design and Implementation*, pp. 57–68, 2002.

[DSG03] F. Doucet, S. Shukla, and R. Gupta. Introspection in System-Level Language Frameworks: Meta-Level vs. Integrated. In *Design, Automation, and Test in Europe*, pp. 382–387, 2003.

[EA03] D. Engler and K. Ashcraft. RacerX: Effective, Static Detection of Race Conditions and deadlocks. In *ACM SIGOPS Operating Systems Review*, Volume 37, Issue 5, pp. 237–252, 2003.

[EAH05] C. Eibl, C. Albrecht, and R Hagenau. gSysC: A Graphical Front End for SystemC. In *European Conference on Modelling and Simulation*, pp. 257–262, 2005.

[EC+01] D. Engler, D.Y. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code. *ACM SIGOPS Operating Systems Review*, Volume 35, Issue 5, pp. 57–72, 2001

[Fos06] H. Foster. Seven Habits of Effective Formal Verification Planning. In *SOC-central*, [Online], http://www.soccentral.com, accessed May 2008.

[FD04] G. Fey and R. Drechsler. Improving Simulation-Based Verification by Means of Formal Methods. In *Asia and South Pacific Design Automation Conference*, pp. 640–643, 2004.

[FF01] C. Flanagan and S.N. Freund. Detecting Race Conditions in Large Programs. In *Workshop on Program Analysis for Software Tools and Engineering*, pp. 90–96, 2001.

[FGP07] M. Fujita, I. Gosh, and M. Prasad. *Verification Techniques for System-Level Design*. Morgan Kaufmann Series in Systems on Silicon, 2007.

[FKB08] R. Fechete, G. Kienesberger, and J. Blieberger. A Framework for CFG-Based Static Program Analysis of Ada Programs. In *Ada-Europe International Conference on Reliable Software Technologies*, pp. 130–143, June 2008.

[FKL03] H. Foster, A. Krolnik, and D. Lacey. *Assertion-Based Design*. Kluwer, 2003.

[FTA02] J.S. Foster, T. Terauchi, and A. Aiken. Flow-Sensitive Type Qualifiers. In *Conference on Programming Language Design and Implementation*, pp. 1–12, 2002.

[Ghe06] F. Ghenassia. *Transaction-Level Modeling with SystemC*. Kluwer Academic Publishers, 2006.

[God97] P. Godefroid. Model checking for programming languages using VeriSoft. In *Symposium on Principles of Programming Languages*, pp. 174–186, 1997.

[GCOV] GNU test coverage program gcov. [Online], http://gcc.gnu.org/onlinedocs/gcc/Gcov.html, accessed June 2008.

[GD03] D. Große and R. Drechsler. Formal Verification of LTL Formulas for SystemC Designs. In *IEEE International Symposium on Circuits and Systems*, pp. 245–248, 2003.

[GD04a] D. Große and R. Drechsler. Checkers for SystemC Designs. In *International Conference on Formal Methods and Models for Codesign*, pp. 171–178, 2004.

[GD04b] G. Fey and R. Drechsler. Improving Simulation-Based Verification by Means of Formal Methods. In *Asia and South Pacific Design Automation Conference*, pp. 640–643, 2004.

[GDB] R. Stallman, R. Pesch, St. Shebs, et al. *Debugging with GDB*. Free Software Foundation, Inc., [Online], http://www.gnu.org/software/gdb, accessed July 2008.

[GDLA03] D. Große, R. Drechsler, L. Linhard, and G. Angst. Efficient Automatic Visualization of SystemC Designs. In *Forum on specification and Design Languages*, pp. 646–657, 2003.

[GDPG01] A. Gerstlauer, R. Dömer, J. Peng, and D.D. Gajski. *System Design – A Practical Guide with SpecC*. Springer, 2001.

[GED07] D. Große, R. Ebendt, and R. Drechsler. Improvements for Constraint Solving in the SystemC Verification Library. In *Great Lakes Symposium on VLSI*, pp. 493–496, 2007.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GPKD08] D. Große, H. Peraza, W. Klingauf, and R. Drechsler. Measuring the Quality of a SystemC Testbench by Using Code Coverage Techniques. In *Embedded Systems Specification and Design Languages: Selected Contributions from FDL'07*, pp. 73–86, Springer, 2008.

[GV+94] D.D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design of Embedded Systems*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[GWSD08] D. Große, R. Wille, R. Siegmund, and R. Drechsler. Contradiction Analysis for Constraint-Based Random Simulation. In *Forum on Specification & Design Languages*, pp. 130–135, 2008.

[Hum04] W. S. Humphrey. The quality attitude. In *news@sei 2004 | Number 3*, [Online], http://www.sei.cmu.edu/publications/news-at-sei/columns/watts_new/2004/3/watts-new-2004-3.htm, accessed May 2008.

[HCNC05] S. Hangal, N. Chandra, S. Narayanan, and S. Chakravorty. IODINE: A Tool to Automatically Infer Dynamic Invariants for Hardware Designs. In *Design Automation Conference*, pp. 775–778, 2005.

[HCXE02] S. Hallem, B. Chelf, Y. Xie, and D. Engler. A System and Language for Building System-Specific, Static Analyses. In *Conference on Programming Language Design and Implementation*, pp. 69–82, 2002.

[HFG08] P. Herber, J. Fellmuth, and S. Glesner. Model Checking SystemC Designs Using Timed Automata. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 131–136 , 2008.

[HJM04] T.A. Henzinger, R. Jhala, and R. Majumdar. Race Checking by Context Inference. In *International Conference on Programming Language Design and Implementation*, pp. 1–13, 2004.

[HL02] S. Hangal and M.S. Lam. Tracking down Software Bugs using Automatic Anomaly Detection. In *International Conference on Software Engineering*, pp. 291–301, 2002.

[HR06] M. Holzer and M. Rupp. Static Code Analysis of Functional Descriptions in SystemC. In *IEEE International Workshop on Electronic Design, Test and Applications*, pp. 243–248, 2006.

[HR08] C. Haufe and F. Rogin. Ad-Hoc Translations to Close Verilog Semantics Gap. In *IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pp. 195–200, 2008.

[HT04] A. Habibi and S. Tahar. On the Extension of SystemC by SystemVerilog Assertions. In *Canadian Conference on Electrical and Computer Engineering*, Volume: 4, pp. 1869–1872, 2004.

[HT05] A. Habibi and S. Tahar. On the Transformation of SystemC to AsmL Using Abstract Interpretation. *Electronic Notes in Theoretical Computer Science*, Vol. 131:39–49, 2005.

[HT06] A. Habibi and S. Tahar. Design and Verification of SystemC Transaction-Level Models. *IEEE Transactions on VLSI Systems*, 14(1):57–68, 2006.

[IB06] B. Isaksen and V. Bertacco. Verification through the Principle of Least Astonishment. In *IEEE/ACM International Conference on Computer-Aided Design*, pp. 860–867, 2006.

[IE+05a] IEEE Computer Society. IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language. 2005.

[IE+05b] IEEE Computer Society. IEEE Standard for Property Specification Language. 2005.

[IE+06] IEEE Computer Society. IEEE Standard SystemC Language Reference Manual. 2006.

[IIS] Fraunhofer IIS. “Modeling SystemC” training class. [Online], http://www.iis.fraunhofer.de/bf/train/kurse/sysc, accessed October 2008.

[ITRS07] International Technology Roadmap for Semiconductors – Design, ITRS Edition 2007. [Online], http://www.itrs.net, accessed January 2009.

[Joh97] R.E. Johnson. Frameworks = (components + patterns). In *Communications of the ACM*, 40(10):39–42, 1997.

[Kog06] T. Kogel. Peripheral Modeling for Platform Driven ESL Design. In *Platform Based Design at the Electronic System Level*, pp. 71–85, Springer, 2006.

[Kro99] T. Kropf. *Introduction to Formal Hardware Verification*. Springer, 1999.

[KL88] B. Korel and J. Laski. Dynamic program slicing. *Information Processing Letters*, 29(3):155–163, 1988.

[KP99] B.W. Kernighan and R. Pike. *The Practice of Programming*. Addison-Wesley, 1999.

[KPKG02] A. Kuehlmann, V. Paruthi, F. Krohm, and M.K. Ganai. Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. *IEEE Transactions in Computer-Aided Design*, 21(12):1377–1394, 2002.

[KS05] D. Kröning and N. Sharygina. Formal Verification of SystemC by Automatic Hardware/Software Partitioning. In *Conference on Formal Methods and Programming Models for Codesign*, pp. 101–110, 2005.

[KSF99] D. Kranzlmüller, N. Stankovic, and J. Volkert. Debugging Parallel Programs with Visual Patterns. In *IEEE Symposium on Visual Languages,* pp. 180–181, 1999.

[KT07] A. Kasuya and T. Tesfaye. Verification Methodologies in a TLM-to-RTL Design Flow. In *Design Automation Conference*, pp. 199–204, 2007.

[LMB92] J. Levine, T. Mason, and D. Brown. *Lex and Yacc: UNIX Programming Tools (A Nutshell Handbook)*. O'Reilly Media, 1992.

[LW+05] M.S. Lam, J. Whaley, V.B. Livshits, M.C. Martin, D. Avots, M. Carbin, and C. Unkel. Context-Sensitive Program Analysis as Database Queries. In *Symposium on Principles of Database Systems*, pp. 1–12, 2005.

[Mar98] F. Martin. PAG – An Efficient Program Analyzer Generator. *International Journal on Software Tools for Technology Transfer*, 2(1):46–67, 1998.

[Mor98] R. Morgan. *Building an Optimizing Compiler*. Digital Press, 1998.

[Muc97] S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann, 1997.

[MMM05a] 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*, pp. 26–35, 2005.

[MMM05b] 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*, pp. 317–324, 2005.

[MR+01] W. Mueller, J. Ruf, D. Hoffmann, J. Gerlach, T. Kropf, and W. Rosenstiehl. The Simulation Semantics of SystemC. In *Design, Automation and Test in Europe*, pp. 64–70, 2001.

[MS06] G. Misherghi and Z. Su. HDD: Hierarchical Delta Debugging. In *International Conference on Software Engineering*, pp. 142–151, 2006.

[NB05] N. Nagappan and T. Ball. Static Analysis Tools as Early Indicators of Pre-Release Defect Density. In *International Conference on Software Engineering*, pp. 580–586, 2005.

[NH06] B. Niemann and C. Haubelt. Assertion-Based Verification of Transaction Level Models. In *Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen"*, pp. 232–236, 2006.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.

[NS+02] J. Niere, W. Schäfer, J.P. Wadsack, L. Wendehals, and J. Welsh. Towards Pattern-Based Design Recovery. In *International Conference on Software Engineering*, pp. 338–348, 2002

[NW+04] N. Nagappan, L. Williams, J. Hudepohl, W. Snipes, and M. Vouk. Preliminary Results On Using Static Analysis Tools For Software Inspection. In *International Symposium on Software Reliability Engineering*, pp. 429–439, 2004.

[OHT04] K. Oumalou, A. Habibi, and S. Tahar. Design for verification of a PCI bus in SystemC. In *International Symposium on System-on-Chip*, pp. 201–204,2004.

[OSCI] OSCI. SystemC home page. [Online], http://www.systemc.org, accessed July 2008.

[OVL] Accellera Organization. Accellera Standard OVL V2. [Online], http://www.accellera.org, accessed April 2008.

[Pal03] S. Palnitkar. *Verilog HDL: A Guide to Digital Design and Synthesis*. Prentice Hall, 2nd edition, 2003.

[Parr07] T. Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Programmers, 2007.

[PC95] C. Potter and T. Cory. *CAST Tools: An Evaluation and Comparison*. Bloor Research Group, 1995.

[PC05] S. Park and S. Chae. A C/C++-Based Functional Verification Framework Using the SystemC Verification Library. In *IEEE International Workshop on Rapid System Prototyping*, pp. 237–239, 2005.

[PE04] J.H. Perkins and M.D. Ernst. Efficient Incremental Algorithms for Dynamic Detection of likely Invariants. In *ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 23–32, 2004.

[PS03] E. Pozniansky and A. Schuster. Efficient On-the-Fly Data Race Detection in Multithreaded C++ Programs. In *Principles and Practice of Parallel Programming*, pp. 179–190, 2003.

[Rapp08] C.W. Rapp. SMC Programmer's Manual. [Online], http://smc.sourceforge.net/SmcManual.htm, accessed June 2008.

[RDR09] F. Rogin, R. Drechsler, and Steffen Rülke. Automatic Debugging of System-on-a-Chip Designs. In *IEEE International SOC Conference*, pp. 333–336, 2009.

[RF04] F. Rogin and E. Fehlauer. FSM-Based Rule Specification Aiming at a Generic Code Analysis Library. In *Work-in-Progess Session at EUROMICRO*, 2004.

[Rog02] F. Rogin. *Konzeption und Prototyp-Implementierung eines Parser-basierten Analysesystems für Spezifikationen in e-Code*. Diplomarbeit, BTU Cottbus, 2002.

[RF+07] F. Rogin, E. Fehlauer, S. Rülke, S. Ohnewald, and T. Berndt. Non-Intrusive High-level SystemC Debugging. In *Advances in Design and Specification Languages for Embedded Systems*, pp. 131–144, Springer, July 2007.

[RFHO07] F. Rogin, E. Fehlauer, C. Haufe, and S. Ohnewald. Debug Patterns for Efficient High-level SystemC Debugging. In *IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pp. 403–408, 2007.

[RFSH05] F. Rogin, E. Fehlauer, A. Schneider, and J. Haase. Automatische Generierung von Dokumentationen für VHDL-AMS-Modellbibliotheken. In *ASIM Workshop*, 2005, Slides [Online], http://swt.cs.tu-berlin.de/asim-sts-05/folien/rogin.pdf.

[RGDR08] F. Rogin, C. Genz, R. Drechsler, and S. Rülke. An Integrated SystemC Debugging Environment. In *Embedded Systems Specification and Design Languages: Selected papers from FDL 2007*, pp. 59–71, Springer, 2008.

[RH06] V.P. Ranganath, and J. Hatcliff. An Overview of the Indus Framework for Analysis and Slicing of Concurrent Java Software. In *International Workshop on Source Code Analysis and Manipulation*, pp. 3–7, 2006.

[RHKR01] J. Ruf, D.W. Hoffmann, T. Kropf, and W. Rosenstiel. Simulation-Guided Property Checking on Multi-Valued AR-Automata. In *Design, Automation and Test in Europe*, pp. 742–748, 2001.

[RK+08] F. Rogin, T. Klotz, G. Fey, R. Drechsler, and S. Rülke. Automatic Generation of Complex Properties for Hardware Designs. In *Design, Automation, and Test in Europe*, pp. 545–548, 2008.

[RK+09] F. Rogin, T. Klotz, G. Fey, R. Drechsler, and S. Rülke. Advanced Verification by Automatic Property Generation. *IET Computers & Digital Techniques*, 3(4):338–353, 2009.

[Sal03] A. Salem. Formal Semantics of Synchronous SystemC. In *Design, Automation and Test in Europe*, pp. 376–381, 2003.

[Shu02] F. Shull et al. What We Have Learned About Fighting Defects. In *IEEE International Symposium on Software Metrics*, pp. 249–258, 2002.

[Sto00] S.D. Stoller. Model-Checking Multi-threaded Distributed Java Programs. In *International SPIN Workshop on SPIN Model Checking and Software Verification*, pp. 224–244, 2000.

[Summit] Summit Design, Inc. Vista Design Environment for SystemC Homepage. [Online] http://www.sd.com, accessed August 2008.

[Syn] Synopsys Inc. Simple 8 bit micro-controller. VCS simulator example package.

[SB+97] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson. Eraser: A Dynamic Data Race Detector for Multithreaded Programs. *ACM Transactions on Computer Systems*, 15(4):391–411, 1997.

[SBR02] A. Siebenborn, O. Bringmann, and W. Rosenstiel. Worst-case Performance Analysis of Parallel, Communicating Software Processes. In *International Workshop on Hardware/Software Codesign*, pp. 37–42, 2002.

[SC+96] S. Shende, J. Cuny, L. Hansen, J. Kundu, S. McLaughry, and O. Wolf. Event and State-Based Debugging in TAU: A Prototype. In *Symposium on Parallel and Distributed Tools*, pp. 21–30, 1996.

[SCV] SystemC Verification Group. SystemC Verification Standard Specification - Version 1.0e. [Online], http://www.systemc.org, accessed November 2008.

[SLU05] O. Spinczyk, D. Lohmann, and M. Urban, AspectC++: an AOP Extension for C++. *Software Developer's Journal*, pp. 68–76, 2005.

[SMA04] K. R.G. da Silva, E. U.K. Melcher, and G. Araujo. An Automatic Testbench Generation tool for a SystemC Functional Verification Methodology. In *Symposium on Integrated Circuits and System Design*, pp. 66–70, 2004.

[SO06] N. Shi and R.A. Olsson. Reverse Engineering of Design Patterns from Java Source Code. In *International Conference on Automated Software Engineering*, pp. 123–134, 2006.

[Tas02] G. Tassey. The Economic Impacts of Inadequate Infrastructure for Software Testing. *National Institute for Standards and Technology*, 2002.

[TH07] J. Teich and C. Haubelt. *Digitale Hardware/Software-Systeme*. 2.erweiterte Auflage, Springer-Verlag Berlin Heidelberg, 2007.

[TLM2] TLM Working Group. OSCI TLM2 User Manual. In *TLM2 documentation package*, [Online], http://www.systemc.org, accessed July 2008.

[Ull89] J.D. Ullmann. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, Rockville, MD, 2nd Edition, 1989.

[Vera] Synopsys Inc. OpenVera. [Online], http://www.open-vera.com, accessed May 2008

[Vok06] M. Vokac. An efficient Tool for Recovering Design Patterns from C++ Code. *Journal of Object Technology*, 5(1):139–157, 2006.

[Wei84] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, 1984.

[Wen03] L. Wendehals. Improving Design Pattern Instance Recognition by Dynamic Analysis. In *Workshop Dynamic Analysis*, pp. 29–32, 2003.

[Wil01] E.D. Willink. *Meta-Compilation for C++*. PhD Thesis, Computer Science Research Group, University of Surrey, 2001.

[WD+05] A. Wieferink, M. Doerper, T. Kogel, G. Braun, A. Nohl, R. Leupers, G. Ascheid, and H. Meyr. A System Level Processor/Communication Co-Exploration Methodology for Multi-Processor System-on-Chip Platforms. In *IEE Proceedings: Computers & Digital Techniques*, 152(1):3–11, 2005.

[WM96] R. Wilhelm and D. Maurer. *Übersetzerbau. Theorie, Konstruktion, Generierung*. Springer, 2. überarbeitete und erweiterte Auflage, 1996.

[WN05] W. Weimer and G. Necula. Mining Temporal Specifications for Error Detection. In *International Conference on Tools and Algorithm for the Construction and Analysis of Systems*, pp. 461–476, 2005.

[WOLB92] L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications*. McGraw-Hill, 2nd edition, 1992.

[XR08] G. Xu and A. Rountev. AJANA: A General Framework for Source-Code-Level Interprocedural Dataflow Analysis of AspectJ Software. In *Conference on Aspect-Oriented Software Development*, pp. 36–47, 2008.

[YE04] J. Yang and D. Evans. Automatically Inferring Temporal Properties for Program Evolution. In *International Symposium on Software Reliability Engineering*, pp. 340–351, 2004

[YE+06] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and D. Das. Perracotta: Mining Temporal API Rules from Imperfect Traces. In *International Conference on Software Engineering*, pp. 282–291, 2006.

[YPA06] J. Yuan, C. Pixley, and A. Aziz. *Constraint-Based Verification*. Springer, 2006.

[YRC05] Y. Yu, T. Rodeheffer, and W. Chen. RaceTrack: Efficient Detection of Data Race Conditions via Adaptive Tracking. *ACM SIGOPS Operating Systems Review*, 39(5):221–234, 2005.

[Zel05] A. Zeller. *WHY PROGRAMS FAIL – A Guide to Systematic Debugging*. Morgan Kaufmann, 2005.

[Zel08] A. Zeller. Using Delta Debugging - A short tutorial. [Online], http://www.st.cs.uni-sb.de/dd/ddusage.php3, accessed October 2008.

[ZH02] A. Zeller and R. Hildebrandt. Simplifying and Isolating Failure-inducing Input. *IEEE Transactions on Software Engineering*, 28(2):183–200, 2002.

[ZW+06] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J.P. Hudepohl, and M.A. Vouk, On the Value of Static Analysis for Fault Detection in Software. *IEEE Transactions on Software Engineering*, 32(4):240–253, 2006.

List of Acronyms

The following list defines the acronyms used throughout the book:

ABV	Assertion-Based Verification
API	Application Programming Interface
ASM	Abstract State Machine
BDD	Binary Decision Diagrams
BFM	Bus Functional Model
BMC	Bounded Model Checking
CFG	Control Flow Graph
DB_A	REGATTA analysis database
DB_R	REGATTA report database
DFA	Data Flow Analysis
EBNF	Extended Backus Naur Form
ESL	Electronic System Level
EST	Extended Syntax Tree
FDC	FSM-Described Configurations
FSM	Finite-State Machine
HDL	Hardware Description Language
IP	Intellectual Property
OVL	Open Verification Library
PSL	Property Specification Language
RTL	Register Transfer Level
SCV	SystemC Verification Library

SDL	Specification and Description Language
SIMD	Single Instruction Multiple Data
SoC	System-on-a-Chip
SVA	SystemVerilog Assertions
TLM	Transaction Level Modeling
VCD	Value Change Dump

Index of Symbols

(t_i, δ_n)	activation time point at simulation time t_i and delta cycle δ_n of D
$\perp_i$	ith parse point annotated to a grammar rule in Γ
A	analysis alphabet of Γ
$a_{p_i}[t]$	activity of property p_i at time t
$a_{s_i}[t]$	activity of signal s_i at time t
B	basic block
C	set of property checkers
C_{basic}	set of basic property checkers
$C_{complex}$	set of complex property checkers
c_F	failing test case
c_P	passing test case
Δ	delta (difference) between two process schedules
δ_n	nth delta cycle while simulating D
D	SystemC design
f_A	function annotating data flow information to G
G	control flow graph
Γ	context-free grammar
I	set of method and thread processes instantiated in D
Π	set of all parse points $\perp_i$ for Γ
P	set of properties valid on T
P_{cand}	set of property candidates
p_i	property in P valid on T
rx	regular expression string
S	vector of signals in T
S_{EX}	set of excluded signals from S
s_i	signal from S
$s_i[t]$	value (signal state) of signal s_i at time t
S_M	set of signals from S matching rx
$S_d^X(w)$	dynamic slice for variable v at execution position q and program input x with $w = (v, q, x)$
$S_s^X(w)$	static slice for variable v at execution position q with $w = (v, q)$
T	simulation trace
t	time reference
t_{cyc}	length of simulation trace T
$v[t]$	design state at time t
Ψ	process schedule of a

	concrete simulation run of D
X	set of activation time points of D
x^n	n^{th} activation of an instantiated process $x \in I$ while simulating D
Z	FDC specification

Index