International Journal of Computer and Electrical Engineering, Vol. 1, No. 5 December, 2009
1793-8163

SHE Methodology: Combining Performance and Functional Analysis of TLM Model

K.L. Man, H.L. Leung, M. Mercaldi, and J. Huang

***Abstract*—Software/Hardware Engineering (SHE) is a model- driven system-level design methodology for hardware/software sys- tems and embedded systems. It assists a designer in both the development of models for analysis purposes and the actual realisation of the system; and the toolset of SHE is the result of software engineering research with a very strong foundation in formal theories/methods. In the recent years, the Transaction Level Modelling (TLM) paradigm has been widely propagated for complex System-on-a- Chip (SoC) and embedded system designs; and it is being used in industry to solve a variety of practical problems during the design, development and deployment of such designs. This paper presents the application of the SHE methodology to combine Performance and Functional Analysis of TLM models. We illustrate the practical interest of our approach with an example: A CPU and Memory System.**

***Index Terms*— Software/Hardware Engineering (SHE), performance and functional analysis, Transaction Level Modelling (TLM), formal methods.**

I. INTRODUCTION

In the recent years, the Transaction Level Modelling (TLM) paradigm [1] has been widely propagated for complex System-on-a-Chip (SoC) and embedded system designs; and it is being used in industry to solve a variety of practical problems during the design, development and deployment of such designs. The practical problems include:

- providing an early platform for software development;
- aiding software/hardware integration;
- enabling both software performance analysis and system level design analysis;
- allowing early estimation of power consumption (by real software running on TLM models before Register Transfer Level (RTL) models are available to further assist the architect).

Principally, in a TLM model, components are modelled as modules in a parallel context. Such modules are communicated in the form of transactions through abstract channels. Furthermore, TLM models allow "very fast simulation", around 1000 times faster than pure RTL models.

K.L. Man is with Xi'an Jiaotong-Liverpool University, China, email: ka.man@xjtlu.edu.cn.
H.L. Leung is with Solari, Hong Kong, email: sales@solari-hk.com.
M. Mercaldi is with M.O.S.T., Turin, Italy, email: michele.mercaldi@ most.it.
J. Huang is with Philips & LiteOn Digital Solutions Netherlands, Advanced Research Center, The Netherlands, email: jinfeng.huang@gmail.com

In the literature, there are different definitions/views for the terminology "interoperability". For us, interoperability is the ability of components, systems or processes to work together to accomplish task/goal (based on the definitions given in [5]).

Performance analysis of TLM models has so far been mainly addressed in a simulation context (i.e. traditional and popular technique for performance analysis). Over the years, simulation is shown to be well-established and successful technique for the analysis of the dynamical behaviour of TLM models.

For *functional analysis*, the most popular technique used in industry for verifying functional properties of TLM models is *model checking* (i.e. a formal verification technique). This technique has also proven to be a successful technique to algorithmically check whether a specification satisfies a desired property.

It is worth mentioning that simulation and model checking are two complementary techniques for verifying the correctness of TLM models, i.e. one cannot replace another.

Over the years, serious efforts have been made to the performance analysis of TLM models (e.g. [2]) and functional analysis of TLM models (e.g. [3], [4]). However, to the best of our knowledge, little attention is paid so far to combine performance and functional analysis of TLM models.

Hence, in this paper, we propose an approach to interoperate performance and functional analysis of TLM models. Together to obtain full-blown performance as well as functional analysis through the *Software/Hardware Engineering* (SHE) [6].

Our approach is to formally describe TLM models using the formal language (i.e. with a well-defined semantics) POOSL [6] which is the modelling language of the SHE methodology that can be reasonable easily translated (even in a formal way) into models written in various input languages of existing tools for performance and functional analysis. While doing these, we can also compare the input languages, the techniques used for such tools as well as the analysis results. The interests of this paper are as follows:

- to study the principles of the TLM from a theoretical point of view;
- to increase understanding of industrial TLM models, to increase clarity of such models and to help solve problems and remove errors as early as possible (i.e. to reduce production cost and time);
- to focus on examining other techniques to performance analysis and functional analysis and trying to combine them for TLM models (i.e. to obtain full-blown perfor- mance as well as functional analysis);
- to compare the specification languages for TLM models, the expressiveness of the functional temporal logics and the algorithmic techniques for model checking that are used in the tools (for verification of TLM models);
- to inspire the development of tools to combine performance and functional analysis for industrial TLM

models; such research results and gained experience could be implemented into commercial products (i.e. toolset).

A. Organisation

The structure of this paper is as follows. Section II summarises the approaches and techniques that are being used for specification and analysis of TLM models in industry. Section III briefly presents the SHE methodology including the input language and toolset. A TLM model: a CPU and memory system is described in Section IV.

Such a TLM model is analysed by means of simulation and formal verification through the SHE methodology; and the results obtained from simulation and formal verification are also shown to be match in the same section. Finally, concluding remarks are made and the direction of future work is pointed out in Section VI.

II. SPECIFICATION AND ANALYSIS OF TLM MODELS IN INDUSTRY

SystemC [7], [8] is a modelling language consisting of C++ class library and a simulation kernel for Hardware Description Language (HDL) designs, encompassing system- level behavioural descriptions down to RTL representations. Nowadays, SystemC is becoming the de-facto-standard for system-level modelling and design as well as TLM in industry. SystemVerilog [9] extends the scope of Verilog [10] to object orientation, test-benches and with enhanced verification features. SystemVerilog is entirely appropriate for TLM because it contains all the constructs necessary to describe and execute TLM models efficiently. Perhaps, designers/verification engineers with a strong hardware background prefer to use SystemVerilog rather than SystemC to write TLM models.

Nevertheless, SystemC and SystemVerilog are two well-accepted design languages for TLM in industry. Currently, analysis of TLM models described in SystemC and SystemVerilog is mainly addressed in a simulation context using commercial simulators.

As we have mentioned already in Section I, simulation is not suitable for functional analysis. Also, there are no formal semantics defined for SystemC and SystemVerilog. For functional analysis of TLM models (described in SystemC or SystemVerilog), it has been generally done by implicitly proposing some forms of translation of the TLM models into formal models (e.g. [11], [12], [13], [14], [15], [16], [17]), the translations just being presented as an unavoidable intermediate step to reach such an functional analysis goal.

It is not hard to see that there might be a gap between the simulation and verification results because of using two specification languages/formalisms for modelling TLM models.

III. THE SHE METHODOLOGY

Software/Hardware Engineering (SHE) is a model-driven system-level design methodology that allows analysing both correctness and performance properties of design alternatives based on models. The actual evaluation is based on the application of several techniques for performance analysis and formal verification of correctness properties (i.e. functional analysis).

The designer is assisted in constructing models and applying the analysis techniques with various guidelines and modelling patterns. The main key feature of the SHE methodology is its foundation on formal methods, which ensures that the obtained analysis results are unambiguous. Also, SHE is accompanied with a set of user-friendly computer tools (most of them are also based on formal methods).

TABLE I POOSL STATEMENTS

S::=	E	expression
	$m(E_1, ..., E_i)(v_1, ..., v_j)$	methods call
	par S_1 and ... and S_n rap	parallel composition
	S_1; S_2	sequential composition
	$ch!m(E_1, ..., E_i)\{E\}$	message send
	$ch?m(v_1, ..., v_i \mid E_c)\{E\}$	(conditional) message receive
	sel S_1 or ... or S_n les	non-deterministic selection
	$[E_c]S$	guarded execution
	interrupt S_1 with S_2	interrupt
	abort S_1 with S_2	abort
	delay E	time synchronisation
	if E_c then S_1 else S_2 fi	choice
	while E_c do S od	loop
	ε	empty behaviour

A. POOSL

The core of the SHE methodology is a modelling language called *Parallel Object-Oriented Specification Language* (POOSL) [6]. POOSL is a language for system-level de- sign, which intends to deal with the complexity of hard- ware/software systems by bridging the gap between industrial practice and formal methods.

The POOSL language integrates a process part and a data part [18]. The process part of the POOSL is based on a timed and probabilistic extension of CCS, which is defined in a similar mathematical way as in common process algebras [19], [20], [21]. The data part inherits the concepts of traditional object-oriented languages such as Java and SmallTalk.

A POOSL model consists of a set of parallel processes, which perform their activities asynchronously and communicate with each other synchronously by message passing. Each process can call and execute its methods which are formed by the statements in Table I. Different from procedure or functions used in imperative programming languages, the process methods in POOSL allow tail-recursion to specify infinite behaviours in a succinct way.

Here we give a brief explanation of the language. More detailed information about the language can be found in [18] [22].

A POOSL model consists of a set of parallel processes connected by static channels. Each process has its own data space. It can only share its information with other processes through synchronous communication. Communications between processes are accomplished through ports connected by static channels. For instance, statement “out! request” indicates the willingness to send a request message through port “out” and “in? request” indicates the willingness to receive a request message from port “in”. When the “in” and “out” ports are connected by a channel, both parts are synchronised and the communication is performed.

In addition to the parallelism between processes, a finer grain of parallelism (parallel activities) can be also specified inside a process using the par statement ("par S_1 and...and S_n rap"). Each activity can share a data space with other activities, and exchange its information with others through shared data.

TABLE II THE POOSL MODEL OF THE CONSUMER AND PRODUCER EXAMPLE

```
Main()()
  par
    Consumer()()
  and
    Producer()()
  rap.

Consumer()()
  BOOL fready;
  ToPro ! request;
  FromPro ? ack (fready);
  if (fready) then
    delay 2
  else
  delay 0.1 fi;
  Consumer()().

Producer()()
  FromCon?reque;
  sel
    ToCon!ack(TRUE)
  or
    ToCon!ack(FALSE)
  les;
  Producer()().
```

The POOSL language provides the "delay" primitive to specify timing information in the model. Similar to many other formal languages, the timing semantics of POOSL relies on the two-phase execution model in [23]. The state of the model can change either by asynchronously executing actions (Phase 1) or by synchronously consuming time (Phase 2). Time advances only when no action can be performed. This timing semantics assumes that actions are instantaneous in the model, which largely simplifies the analysis of the timing behaviour.

In addition to the traditional "if" statement, nondeterministic selection ("sel S_1 or...or S_n les") does not specify conditions for its branches. This facilitates designers to abstract system behaviour. This is due to the fact that there are not always enough details available to determine the conditions when making an abstraction of a system behaviour or one component has no knowledge of its peers' behaviour.

For instance consider a system consisting of a consumer and a producer. The consumer requests for the product from the producer. If the producer is not empty, the consumer takes one product and consumes it within 2 time units. Otherwise, the consumer waits for 0.1 time units and checks availability of the producer. The above mentioned behaviour is endlessly repeated. Such a behaviour can be specified by three process methods in POOSL given in Table II.

The Main()() method consists of a parallel structure where Consumer()() and Producer()() are specified as two parallel activities. The Consumer()() and Producer()() methods specify the behaviour of the Consumer and the Producer respectively at an abstract level. In the specification, ToPro (FromPro) and FromCon (ToCon) are a pair of connected ports. The Producer's internal behaviour is abstracted by the non-deterministic structure (sel), which facilitates a fast evaluation of design ideas. Note that the infinite behaviour of the Consumer and the Producer are specified by the tail recursion.

In addition to the primitives mentioned above, the POOSL language offers a set of powerful primitives such as Guard, Interrupt and Abort primitives to facilitate designers to express their thoughts in a succinct way.

The formal semantics of POOSL can be found in [24] [25] [22]. The formal semantics of POOSL given in [24] addresses untimed behaviours covering parallelism, communi- cation, non-determinism and data. The semantics of the timed language (where data is abstracted from) is given in [25]. This work also introduces an execution mechanism for the language.

B. Tools for SHE

A number of tools have been developed for building and analysing a POOSL model. The most commonly used tools are *SHESim* and Rotalumis [6].

SHESim is a graphical tool, which allows designers to incrementally specify and modify classes of data, processes and clusters and easily express hierarchical and topological structure of the complex systems. Furthermore, it can create log files for performance evaluating purposes. An interaction diagram tool helps designers to inspect the history of all messages exchanged between different processes by generating interaction diagrams automatically during simulation, These diagrams can be used for validation purposes of a model.

Rotalumis is a textual tool which is initially for high-speed simulation after completing the validation of a POOSL model. Later, it is enhanced for the correctness-preserving code generation from a POOSL model. It has been demonstrated in [30] that Rotalumis can automatically generate code for timing- critical systems and preserve the timing properties proven in the model.

Next to these two tools, several formal verification tools (e.g. SPIN [31], [32] and UPPAAL [33]) can be used as back-end verification tools by translating POOSL models to the corresponding input models/specifications for such verification tools. Currently, SHE includes a few guidelines for construct- ing various input models/specifications of verification tools from a POOSL model [34].

IV. SPECIFICATION AND ANALYSIS OF TLM MODELS IN THE SHE METHODOLOGY

In order to show the applicability of our approach/the SHE methodology to combine performance and functional analysis of TLM models, we specify and analyse a TLM model in the SHE methodology.

A. CPU and Memory System

A simulation is performed on a system modelling one memory and one CPU[2] in which the memory is a pure slave and the CPU is a pure master. A memory interface dictates what services complying memories must feature; and the CPU implementation demonstrates all the supported

operations with the memory.

More precisely, the CPU first gets to know if it is possible to write to a random memory address. Eventually, the CPU issues a *write* and *read* request operation with a data unit and repeats the same with more operations provided by the memory.

[2] Such a system is taken from (http://mij.oltrelinux.com/devel/systemc/) and its implementation in SystemC is also available at same link.

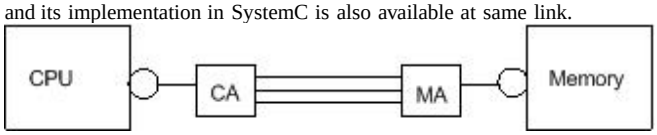

Fig. 1. A CPU and memory system

Two module interfaces are introduced in the system for describing the actual communication between the CPU and memory. They are namely CPU adapter and memory adapter for the CPU module and memory interface respectively. The CPU and memory system is shown in Fig. 1.

The hardware implementation of such an communication is specified through the usual signals paradigm. More specifi- cally, CPU adaptor (shortly, CA) and memory adaptor (MA) communicate through a set of signals wrapped to realize a full-featured bus.

A protocol must be specified for the communication to be accomplished across the bus. This is entirely enclosed into adaptor implementations and can be changed in the future without impact against the rest of the system relying on them (this is the key idea of TLM).

For illustration purpose, a part (describing the top-level) of the CPU and memory system implemented in SystemC is given below:

```
int  sc_main(int argc, char *argv[]) {
    // provide a memory module and a CPU
    // for  the simulation system
    Memory  mem("MemoryModuleA");
    CPU cpu0("CPU0");
    //construct two adapters for "refining"
    // cpu/mem communication
    MemAdaptor_Signals memadapt("MemAdaptA");
    CPUAdaptor_Signals cpuadapt("CPUAdapt0");
    // a set  of signals (CPU_Mem_Bus)  is used to
    // connect the adaptors' ports
    // CPU_Mem_Bus  cpumembus;
    cpumembus.reset();

    // binding bus ports to the cpu adaptor
    cpuadapt.addr(cpumembus.addr);
    cpuadapt.data(cpumembus.data);
    cpuadapt.memsize(cpumembus.memsize);
    cpuadapt.r_w(cpumembus.r_w);
    cpuadapt.req(cpumembus.req);
    cpuadapt.ack(cpumembus.ack);
    cpuadapt.err(cpumembus.err);

    // binding bus ports to the memory adaptor
    memadapt.addr(cpumembus.addr);
    memadapt.data(cpumembus.data);
    memadapt.memsize(cpumembus.memsize);

    memadapt.r_w(cpumembus.r_w);
    memadapt.req(cpumembus.req);
    memadapt.ack(cpumembus.ack);
    memadapt.err(cpumembus.err);

    // binding the memory adaptor and
    // the actual memory module
    memadapt.memory(mem);

    // binding the cpu adaptor to  the cpu's
    // memory port
    cpu0.memory(cpuadapt);

    // also try with two CPUs (output  messages jams
    // may happen)
    //CPU cpu1("CPU1");
    //cpu1.memory(mem);

    // starting the simulation
    sc_start();
}
```

B. Simulation

The (CPU and memory) system was first described in POOSL and then simulated using SHESim. A simulation run is a list of service-issue messages from the CPU, interleaved with messages from the memory adaptor responding to the bus protocol.

As an example of a simulation run:

```
MemAdaptor:  WRITE request  income,
        processing...  done.
MemAdaptor:  waiting for requests...
CPU:   getting the data...
MemAdaptor:  waiting for requests...
MemAdaptor:  READ request  income,
        processing...  done.
MemAdaptor:  waiting for requests...
```

C. Formal Verification

According to the guidelines given in [34], the (CPU and memory system) POOSL model was easily constructed into the equivalent PROMELA model [32] which is the input model for SPIN.

The SPIN model checker SPIN is a software package that allows the simulation of a specification written in the PROMELA. It accepts correctness claims specified in the syntax of standard *Linear Temporal Logic* [32] (LTL).

Applying the SPIN model checker to the (CPU and memory system) PROMELA model, the following properties (expressed in LTL) were verified successfully in few seconds using a modern PC:

1) *Deadlock free:* There is deadlock free in the state space generated for the (CPU and memory system) POOSL model system.
2) *Liveness property:* CPU eventually gets the data after each write request operation is executed.
3) *Safety property:* No read request operation is executed before a write request operation is called.

D. Interoperability

It is not hard to see that the simulation and formal verification results of the (CPU and memory system) POOSL model are match. More specifically, the properties (deadlock free, liveness and safety) are also shown to be held in the simulation run (by inspection of the simulation traces) as presented in the previous subsection.

This combine approach of the simulation and formal verification (i.e. interoperability approach) aids the designer to perform simulation-based analysis as well as functional anal- ysis of the POOSL model in the SHE Methodology.

V. CONCLUSIONS AND FUTURE WORK

We were not surprised to find that it was possible to specify and execute both performance and functional analysis of the CPU and memory TLM model in the SHE methodology.

In order to illustrate our work clearly, only a simple TLM was given in this paper. Nevertheless, the use of TLM is generally applicable to all sizes and levels of TLM in the SHE methodology.

As future work, we plan to use the SHE methodology to model and analyse complex TLM models.

VI. CONCLUSIONS AND FUTURE WORK

We were not surprised to find that it was possible to specify and execute both performance and functional analysis of the CPU and memory TLM model in the SHE methodology.

In order to illustrate our work clearly, only a simple TLM was given in this paper. Nevertheless, the use of TLM is generally applicable to all sizes and levels of TLM in the SHE methodology.

As future work, we plan to use the SHE methodology to model and analyse complex TLM models.

ACKNOWLEDGEMENT

Many thanks go to the sponsor Solari - Hong Kong (http://www.solari-hk.com/) of the research work as presented in this paper.

REFERENCES

[1] F. Ghenassia, Ed., *Transaction-Level Modeling*. Springer, 2005.
[2] E. Viaud, F. Pecheux, and A. Greiner, "An efficient TLM/T modeling and simulation environment based on conservative parallel discrete event principles," in *the Proceedings of Design, Automation, and Test in* Europe, 2006.
[3] C. Traulsen, J. Cornet, M. Moy, and F. Maraninchi, "A SystemC/TLM semantics in Promela and its possible applications," in *the 14th Workshop on Model Checking Software SPIN*, 2007.
[4] R. K. Shyamasundar, F. Doucet, R. Gupta, and I. H. Kruger, "Compositional reactive semantics of SystemC and verification in rulebase," in *the Proceedings of the Workshop on Next Generation Design and Verification Methodologies for Distributed Embedded Control Systems*, 2007.
[5] T. Krilavičius, "Study of tools interoperability," University of Twente, The Netherlands, Tech. Rep. TR-CTIT-07-01, 2007.
[6] SHE, "SHE: Hardware/software systems," http://www.es.ele.tue.nl /poosl.
[7] SystemC, SystemC Users Guide and SystemC Language ReferenceManual (Version 2.2), http://www.systemc.org.
[8] IEEE, IEEE Standard for SystemC Language Reference Manual (IEEE STD 1666TM-2005), IEEE, 2005.
[9] ——, IEEE Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language (IEEE STD 1800TM-2005), IEEE, 2005.
[10] ——, IEEE Standard for Verilog Hardware Description Language(Revision of IEEE STD 1364-2001), IEEE, 2006.
[11] K. L. Man, "SystemC FL : Formalization of SystemC," in the 12th Mediterranean Electrotechnical Conference MELECON. Dubrovnik, Croatia: IEEE, 2004.
[12] K. L. Man, M. Mercaldi, F. Garberoglio, A. Trischitta, H. Lai, and C. Ho, "SystemC FL : Motivation and development," in the Proceedings of the International MultiConference of Engineers and Computer Scientists 2008, Hong Kong, 2008.
[13] K. L. Man, A. Fedeli, M. Mercaldi, and M. P. Schellekens, "SystemC FL : An infrastructure for a TLM formal verification proposal (with an overview on a tool set for practical formal verification of SystemC descriptions)," in the 4th East-West Design & Test Workshop EWDTS. Sochi, Russia: IEEE, 2006.
[14] K. L. Man, A. Fedeli, M. Mercaldi, M. Boubekeur, and M. P.Schellekens, "SC2SCFL: Automated SystemC to SystemC FL trans- lation," in the 7th International Symposium on Systems, Architectures, Modeling and Simulation, ser. Lecture Notes in Computer Science 4599. Springer-Verlag, 2007, pp. 34–45.
[15] M. Mercaldi, A. Fedeli, and K. Man, "SC2SCFL: An overview," in the4th IEEE International SoC Conference, Seoul, South Korea, 2007.
[16] K. L. Man, M. Boubekeur, and M. P. Schellekens, "Process algebraic approach to SystemVerilog," in the 20th IEEE Canadian Conference on Electrical and Computer Engineering. Columbia, Canada: IEEE, 2007.
[17] K. L. Man, "PAFSV: A process algebraic framework for systemverilog," in International Multiconference on Computer Science and Information Technology. Wisla, Poland: IEEE, 2008.
[18] M. Geilen, J. Voeten, P. van der Putten, L. Bokhoven, and M. Stevens, "Object-oriented modelling and specification using SHE," Journal of Computer Languages, vol. 27, pp. 19–38, 2001.
[19] J. C. M. Baeten and W. P. Weijland, Process Algebra, ser. Cambridge Tracts in Theoretical Computer Science. Cambridge, United Kingdom: Cambridge University Press, 1990, vol. 18.
[20] X. Nicollin and J. Sifakis, "The algebra of timed processes, ATP: Theory and application," Information and Computation, vol. 114, pp. 131–178, 1994.
[21] [21] J. C. M. Baeten and C. A. Middelburg, Process Algebra with Timing, ser. EACTS Monographs in Theoretical Computer Science. Springer- Verlag, 2002.
[22] L. van Bokhoven, "Constructive tool design for formal languages from semantics to executing models," Ph.D. dissertation, Eindhoven University of Technology, 2002.
[23] X. Nicollin and J. Sifakis, "An overview and synthesis on timed process algebras," in Proceedings of the 3rd Workshop on Computer-Aided Verification, ser. Lecture Notes in Computer Science 575, A. K. G. Larsen, Ed. Springer-Verlag, 1991, pp. 376–398.
[24] P. van der Putten and J. Voeten, "Specification of reactive hardware/software systems," Ph.D. dissertation, Eindhoven University of Technology, 1997.
[25] M. Geilen, "Formal techniques for verification of complex real-time systems," Ph.D. dissertation, Eindhoven University of Technology, 2001.
[26] B. Theelen, J.Voeten, and R. Kramer, "Performance modelling of a network processor using POOSL," Journal of Computer Networks, Special Issue on Network Processors, vol. 41, no. 5, pp. 667–684, 2003.
[27] L. Noonan and C. Flanagan, "Modeling a network processor using object oriented techniques," in Proceedings of the Digital System Design, EUROMICRO Systems on DSD'04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 484–490.
[28] F. van Wijk, J. Voeten, and A. ten Berg, System Specification and Design Languages. Kluwer Academic Publishers, 2003, ch. An Abstract Modeling Approach Towards System-Level Design-Space Exploration, pp. 267–282.
[29] B. Theelen, J. Voeten, L. van Bokhoven P. vander Putten, G. de Jong, and A. Niemegeers, "Performance modeling in the large: A case study," in Proceedings of the European Simulation Symposium, 2001.
[30] J. Huang, J. Voeten, and H. Corporaal, "Predictable real-time software synthesis," International Journal on real-time systems, vol. 36, pp. 159–198, 2007.
[31] SPIN,www.spinroot.com/.
[32] G. J. Holzmann, The SPIN Model Checker: Primer and ReferenceManual. Boston: Addison Wesley Professional, 2003.
[33] K. G. Larsen, P. Pettersson, and W. Yi, "UP PA A L in a Nutshell," Int. Journal on Software Tools for Technology Transfer, vol. 1, no. 1–2, pp.134–152, 1997.
[34] M. Geilen, "Formal verification of complex real-time systems," Ph.D. dissertation, Eindhoven University of Technology, 2002.

Ka Lok Man He holds a Dr. Eng. degree in Electronic Engineering from Politecnico di Torino, Italy, and a PhD degree in Computer Science from Technische Universiteit Eindhoven, The Netherlands.

Currently, he is a lecturer at the Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, China.

His research interests include logic synthesis, formal verification and low power design methodologies for integrated circuits and systems, formalisation
of SystemC and SystemVerilog designs including TLM, formal methods, process algebras, formal analysis of real-time and hybrid systems, communication and wireless sensor networks, reversible computing, and software development.

On the above-mentioned topics, he has authored or co-authored about 80 refereed publications including books, edited books, journal articles, book chapters and conference proceedings.

He has been a committee member, reviewer, session chair and special session/workshop organiser of different IEEE, IASTED and IAENG confer- ences. He is also the editor-in-chief of the International Journal of Design, Analysis and Tools for Integrated Circuits and Systems (IJDATICS), the editor of the Journal of Computers (JCP), the associate editor of the Journal of Engineering, Computing and Architecture (JECA), the associate editor of the Journal of Computer Science, Informatics and Electrical Engineering (JCSIEE), the editor of the Journal of Computer Science and Information Technology (JCSIT), the editor of the International Journal of Advancements in Computing Technology (IJACT), the editor of the Magazine of Wireless and Cellular Networks (MWCN), the editor of the International Journal of Digital Content Technology and its Applications (JDCTA) and the editor of the Journal of Convergence Information Technology (JCIT).

Hoi Lam Leung He is the general manager of Solari - Hong Kong. Solari is the official sales agent of Sanyo camera modules and Showa Denko LED (Japan).

Michele Mercaldi He received his Dr. Eng, Degree in Software Engineering from the Politecnico di Torino (Italy) in 1998. From 1997 to to 1999 he worked at the Politecnico di Torino as software and database developer. From 2000 to 2007 he worked in MOST (Turin, Italy) which is a highly skilled firm for documental archiviation using Oracle and Postgresq running on Linux. Currently he is employed by the Critical Path (Turin, Italy) which is one of the world leading providers of internet messaging and directory software.

Jinfeng Huang He received his B.Eng. in computer science from China University of Mining and Tech- nology, China, in 1997, and his M.Sc. in computer science from Xian Jiaotong University, China, in 2000 respectively. In 2005, he received his Ph.D de- gree from Eindhoven University of Technology, The Netherlands, for his work on predictable real-time software design. From April 2005 till Aug. 2007, he worked as a postdoc researcher at Eindhoven University of Technology. His research interests included formal methods on concurrent, real-time and distributed systems and software synthesis. Since Sep. 2007, he works on designing of embedded systems at Philips.