μCRL: A Computer Science based Approach for Specification and Verification of Hardware Circuits

K.L. Man
Centre for Efficiency-Oriented Languages (CEOL)
Department of Computer Science, University College Cork, Ireland
Email: pafesd@gmail.com
URL:http://digilander.libero.it/systemcfl/pafesd

***Abstract*— μCRL is a process algebraic language for the formal specification and analysis of the behaviour of distributed systems. The toolset of μCRL is the result of software engineering research with a very strong foundation in formal theories/methods, which supports the analysis and manipulation of μCRL specifications. This paper investigates a Computer Science based approach for specification and verification of hardware circuits using μCRL and its toolset. Two standard benchmark circuits are described in μCRL and analysed by the μCRL toolset together with the software tools CADP and SPIN, which are well-equipped with the μCRL toolset.**

I. Introduction

Formal methods provide a set of notations that can be used to build mathematical models of systems; and techniques for automatic verification of such models. Over the years, formal methods have been widely and successfully used in a wide range of problems and in practical applications in both academia and industry for the specification and analysis of many different systems. *Formal verification* is intended to prove some properties (e.g. expressed in temporal logic) hold in the system (i.e. a mathematical model) under analysis. Although formal verification has shown to be very useful for analysis of various systems (e.g. hardware circuits), its power is still limited by the complexity of the analysis that grows very large as the size of the systems increases (namely *state space explosion problem*).

On the other hand, *Formal languages* with a semantics formally (i.e. mathematically) defined in *Computer Science* increase understanding of systems, increase clarity of specifications and help solving problems and remove errors. Over the years, several flavours of formal languages have been gaining industrial acceptance. *Process algebras* [1] are formal languages that have formal syntax and semantics for specifying and reasoning about different systems. They are also useful tools for verification of various systems. Generally speaking, process algebras describe the behaviour of processes and provide operations that allow to compose systems in order to obtain more complex systems. Moreover, the analysis and verification of systems described using process algebras can be partially or completely carried out by mathematical proofs using equational theory.

In addition, the strength of the field of process algebras lies in the ability to use *Algebraic reasoning* (also known as equational reasoning) that allows rewriting processes using axioms (e.g. for commutativity and associativity) to a simpler form. By using axioms, we can also perform calculations with processes. These can be advantageous for many forms of analysis. Process algebras have also helped to achieve a deeper understanding of the nature of concepts like observable behaviour in the presence of non-determinism, system composition by interconnection of system components modelled as processes in a parallel context, and notions of behavioural equivalence (e.g. bisimulation [1]) of such systems.

On the other hand, in order to efficiently model systems of ever increasing complexity and size, and to effectively analyse them, powerful techniques or approaches are needed. In process algebras, *Linearisation* [2], [3] is a transformation of a recursive specification into a linear representation (without parallelism), i.e., a kind of normal form that is convenient for many forms of analysis. Note that these linear representations are expressed as recursive specifications as well, but they use only a small subset of the full process algebra. In general, such linear representations can also be considered very compact representations of a possibly infinite state space. The original recursive specification and its transformation are required to be bisimilar, which ensures that the relevant specification properties are preserved. Furthermore, complex systems can be constructed containing lots of parallelism and it is quite difficult to analyse such complex systems. Therefore, it is always useful to transform complex systems to linear representations for analysis.

To formally specify complex hardware circuits and effectively analyse them, in this paper, we propose a Computer Science based approach to use the μCRL language [4] as the specification formalism to formally describe the behaviour of hardware circuits and to apply μCRL toolset [4] (possibly together with other back-end verification tools for μCRL) to analyse them. For the use in this paper, the above choices (using the μCRL language and its toolset) are made, because of the following:

1) the μCRL language comprises mathematical specifications for hardware circuits;
2) the μCRL language allows for description and (syntax-based) analysis of hardware circuits in a compositional fashion;
3) the μCRL language offers the possibility to apply algebraic reasoning on specifications (e.g. to refine the

specifications);

4) the μCRL toolset has a lineariser which automatically converts a μCRL specification into a linear representation (to reduce complexity and to ease analysis);
5) the μCRL toolset offers the possibilities for both simulation and model checking on μCRL specifications;
6) the μCRL toolset is also well-equipped with several tools (e.g. CADP [5], [6] and SPIN [7], [8]) to analyse models of complex systems;
7) the μCRL language can be reasonable easily translated to other formalisms (e.g. petri-nets and theory of automaton) and this leads to ease of verification using existing formal verification tools;
8) the μCRL toolset is free in distribution, well maintained and well documented.

Related Work. Serious efforts have been made in the past to deal with systems (e.g. real-time systems [9], [10] and hybrid systems [11], [12], [13], [14]) in a process algebraic way. Over the years, also several process algebraic theories (e.g. [15], [16], [17], [18]) were applied in the context of the formal specification and analysis of hardware circuits. However, no linearisation algorithms have been developed for such process algebraic theories. As shown in [19], [2], linearisation algorithms are the key of success to analyse complex systems described in process algebra based formalisms.

In this paper, we show the practicability of our Computer Science based approach by means of two standard benchmark hardware circuits. To the best of our knowledge, this is the first article to present the application of μCRL and its toolset (as well as its back-end verification tools) to formally specify and to analyse hardware circuits.

Structure. The structure of the paper is as follows. section II presents μCRL including the syntax, its toolset, verification/proof techniques, etc. For the use in this paper, the tool CADP, the SPIN model checker, μ-calculus [20] and PROMELA [7] are briefly introduced in section III. The application of μCRL and its toolset (together also with the tool CADP and the SPIN model checker) to formally specify and to analyse two standard benchmark hardware circuits is presented in sections IV and V. Finally, concluding remarks are made in section VI and the direction of future work is pointed out in the same section.

II. μCRL

μCRL (*micro Common Representation Language*) is an algebraic specification language that can be used to formally specify and to analyse the behaviour of distributed systems. In principal, μCRL is based on the *Algebra of Communicating Processes* (ACP) [1] extended with equational abstract data types to interwine processes, actions and recursion variables that can be parameterised with data types. In addition, a conditional construct (*if-then-else* can be used to have data elements influence the course of a process, and *alternative quantification* (also known as *choice quantification*) is introduced to sum over possibly infinite data domains.

Data types and Actions. In a μCRL specification, any data type (e.g. natural numbers) can be defined. However, one has to define the boolean type in each μCRL specification. Distinct data types are characterised by their sets of data constructors. Moreover, operations can be defined over data by means of rewrite rules. In μCRL, actions/communication actions with or without parameters can be declared in each specification. When parameters are used the data types of such parameters needed to be given (see the below syntax of μCRL and section V for details and examples).

Syntax and Semantics As in many process algebras, the basic ways of combining processes in μCRL are alternative composition, sequential composition, parallel composition, etc. The μCRL language has a clear syntax and well-defined semantics. However, presenting the syntax and semantics of μCRL is far beyond the scope of this paper, we refer to [21] for a complete description (i.e. syntax and semantics) of the μCRL language.

μCRL Toolset. The μCRL toolset is the result of software engineering research with a very strong foundation in formal theories/methods, which supports the analysis and manipulation of μCRL specifications. Also, μCRL toolset is well-equipped with several tools (e.g. CADP and SPIN) to analyse models of complex systems. The μCRL toolset comprises a lineariser which can transform a μCRL specification into a corresponding *linear process equation* (LPE) which is a linear representation (as already explained in Section I). All other tools in the μCRL toolset use LPEs as their starting point. The μCRL simulator can simulate interactively the behaviour of a LPE. The μCRL state space generator can be used to generate a state space from a LPE. There are several tools that allow analysis and optimisations on the level of LPEs expressed as so-called ".aut" format. Furthermore, the generated state space of a LPE (in the aut format) can be read, visualised and analysed by CADP.

III. CADP, μ-CALCULUS, SPIN MODEL CHECKER AND PROMELA

For the use in this paper, in this section, we shortly introduce CADP, μ-calculus, SPIN Model Checker and PROMELA.

CADP and μ-calculus. CADP (*CAESAR/ALDEBARAN Development Package*) is a very popular toolbox for the design of communication protocols and distributed systems. CADP offers a wide set of functionalities, ranging from step-by-step simulation to massively parallel model-checking. In particular, using this tool, one can express properties in the regular alternation-free μ-calculus for model checking on the state space generated from the model.

SPIN Model Checker and PROMELA. SPIN is a software package that allows the simulation of a specification written in the language PROMELA. It accepts correctness claims specified in the syntax of standard *Linear Temporal Logic* (LTL) [7]. SPIN can be applied to the verification of several types of properties, such as model checking of LTL formulas, verification of state properties, unreachable code, etc.

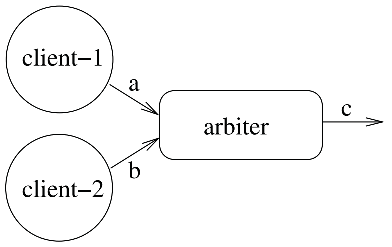

Fig. 1. An asynchronous arbiter.

PROMELA is a modelling language to describe finite-state systems. It resembles the programming *language C* with *Communicating Sequential Processes* (CSP) [22] features. For a complete description of the syntax and semantics of PROMELA, we refer the reader to [7].

IV. An Asynchronous Arbiter

Asynchronous arbiter circuits are standard hardware verification benchmark circuits. An arbiter circuit controls the exclusive access of one out of a number possibly competing processes to a shared resource. Figure 1 shows an (untimed) asynchronous arbiter (taken from [18]) such that two clients (client-1 and client-2) complete for a shared resource. Each client sends a request (a number 1 for client-1 and a number 2 for client-2) for the resource to the arbiter via an individual channel (a and b). The arbiter chooses non-deterministically between clients with pending requests, and then sends the number of the selected client-(1 or 2) via another channel (c) to the environment.

Due to reason of space, the specification of the asynchronous arbiter in μCRL is given below without a detailed description[1].

$$\begin{array}{ll}
\textbf{act} & s_a, s_b, s_c, r_a, r_b, c_a, c_b : \Delta \\
\textbf{comm} & s_a \mid r_a = c_a \\
 & s_b \mid r_b = c_b \\
\textbf{proc} & C_1 = s_a(1).C_1 \\
 & C_2 = s_b(2).C_2 \\
 & A = \sum_{d:\Delta}(r_a(d) + r_b(d)).s_c(d).A
\end{array}$$

Where C_1 denotes client-1, C_2 denotes client-2 and A represents the behaviour of the asynchronous arbiter and such an arbiter initially (as specified by the keyword **init**) consists of C_1, C_2 and A in a parallel context (defined by means of the operator $\|$), which is described as follows:

$$\textbf{init}\ (C_1 \parallel C_2 \parallel A)$$

A. Verification

Using the μCRL toolset, the asynchronous arbiter specification in μCRL was first linearised (this step serves to obtain a smaller size for representation) and then its state space was generated in ".aut" format (consisting of 3 states and 14 transitions only), which is one of the input formats of the CADP tool. The following properties were verified successfully in few seconds using the CADP tool and a modern PC:

- *Deadlock free*. The absence of deadlock (in the state space generated for the asynchronous arbiter specification in μCRL) was verified which is the built-in functionality of the CADP tool.
- *Liveness properties*. If the client-1 sends a number 1 via channel a, the number 1 will be eventually sent to the environment via channel c. Similarly, if the client-2 sends a number 2 via channel b, the number 2 will be eventually sent to the environment via channel c.

[1]However, a detailed account of the description of the asynchronous arbiter in μCRL, the hazardous circuit in μCRL presented in Section V and its translation to PROMELA as well as verification runs on such specifications in μCRL using various verification tools can be found at [23].

V. A Hazardous Circuit

By means of an example: a hazardous (combinational) circuit, this section shows that SPIN (another well-equipped tool with the μCRL toolset) can also be reasonable easily used as a verification engine for μCRL specifications by translating them to the corresponding models in PROMELA that are the input formats of SPIN.

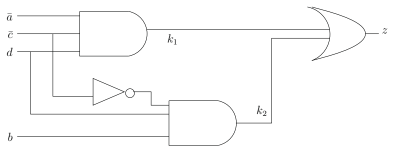

Fig. 2. A hazardous circuit.

Fig. 2 shows a (combinational) circuit that contains a hazard. The output of the circuit has the logical function $z = \bar{a}\bar{c}d + bcd$. Depending on the delays of the inverter and wires, during a transition on signal or wire/port c, a spike may occur. For instance, while $\bar{c}$ is changing from "$T(true)$" to "$F(false)$", the other input signals are still constant. Hence, this leads a hazard at the output z.

The μCRL specification of the hazardous circuit consists of two process definitions *Circuit* and *Stimulus* as follows:

$$\begin{array}{l}
\textbf{proc} \\
\mathit{Circuit}(\bar{a}, b, \bar{c}, d, k_1, k_2, z, new_{sp} : \mathit{Bool}) = \\
\sum_{\{\bar{a},b,\bar{c},d,k_1,k_2,z,new_{sp}:\mathit{Bool}\}}((\\
r_{\bar{a}}(\bar{a}).r_{\bar{c}}(\bar{c}).r_d(d).(s_{k_1}(T) \lhd \mathit{and}(\mathit{and}(\bar{a},\bar{c}),d) \rhd s_{k_1}(F)) \\
+ \\
r_{\bar{c}}(\bar{c}).s_c(\mathit{not}(\bar{c})).r_d(d).r_b(b). \\
(s_{k_2}(T) \lhd \mathit{and}(\mathit{and}(\mathit{not}(\bar{c}),d),b) \rhd s_{k_2}(F)) \\
+ \\
r_{k_1}(k_1).r_{k_2}(k_2).(s_z(T) \lhd \mathit{or}(k_1,k_2) \rhd s_z(F))). \\
s_{new_{sp}}(a_F(new_{sp})) \\
).\mathit{Circuit}(\bar{a}, b, \bar{c}, d, k_1, k_2, z, new_{sp})
\end{array}$$

$$\begin{array}{l}
\mathit{Stimulus}(\bar{a}, b, \bar{c}, d, new_{sp}, old_z : \mathit{Bool}) = \\
\sum_{\{\bar{a},b,\bar{c},d,k_1,k_2,z,new_{sp},old_z:\mathit{Bool}\}}(\\
s_{new_{sp}}(a_T(new_{sp})).s_{old_z}(z).(s_{\bar{a}}(\mathit{not}(\bar{a})) + s_{\bar{c}}(\mathit{not}(\bar{c}))+ \\
s_b(\mathit{not}(b)) + s_d(\mathit{not}(d))) \\
).\mathit{Stimulus}(\bar{a}, b, \bar{c}, d, new_{sp}, old_z)
\end{array}$$

Whole System. Since the process definitions *Circuit* and *Stimulus* execute concurrently, the parallel composition is used to model the complete system. The complete system with appropriated initial values for variables is given below:

$$\begin{array}{l}\textbf{init}\quad \tau_{\{c_{\bar{a}}(*),c_b(*),c_{\bar{c}}(*),c_d(*)\mid *\in Bool\}}(\\ \partial_{\{s_{\bar{a}}(*),s_b(*),s_{\bar{c}}(*),s_d(*),r_{\bar{a}}(*),r_b(*),r_{\bar{c}}(*),r_d(*)\mid *\in Bool\}}(\\ Circuit(T,F,\bar{c},F,k_1,k_2,z,T)\ \| \\ Stimulus(T,F,\bar{c},F,T,old_z)) \\ )\end{array}$$

Again, we refer to [23] for a detailed description of the hazardous circuit in μCRL and its translation to PROMELA.

A. Verification

A crucial details of the translation from a subset of μCRL to PROMELA was given in [24]. After having translated the hazardous circuit μCRL to the corresponding PROMELA model, such a model in PROMELA was tested using SPIN and a modern PC; and a hazard was found in few seconds (see also [23] for details).

VI. Concluding Remarks and Future Work

As we have seen in this paper, μCRL and its toolset (together with different well-equipped back-end verification tools) can efficiently and effectively be used to formally specify and to analyse asynchronous circuits as well as combinational circuits. Also, we believe that the use of μCRL and its toolset (together with various back-end verification tools) is generally applicable to many other types of hardware circuits (e.g. sequential and arithmetic circuits). However, it is not clear yet whether μCRL and its toolset are useful for the formal specification and analysis of larger hardware circuits than examples considered in this paper.

Recently, the formal specification language mCRL2 [25] has been defined, which is the successor of μCRL and extends the μCRL language with new features and improvements (e.g. multi-actions, local communication, higher-order function types, etc).

As future work, we plan to make use of the linearisation algorithms and verification/proof techniques of μCRL and mCRL2 to analysis large hardware circuits described in μCRL and mCRL2.

Acknowledgement

K.L. Man would like to thank Anton Wijs and Jens Calame for analysis of the properties of the asynchronous circuit (in resulting .aut format) using CADP. He wishes to thank Michel Reniers, Pieter Cuijpers, Muck van Weerdenburg and Yaroslav Usenko for many stimulating and helpful discussions on the linearisation algorithms for HyPA [12] and μCRL. He is grateful to have been able to attend the teaching course in μCRL from Jan Friso Groote.

Many thanks also go to the industrial collaborators of the research in this direction: International Software and Productivity Engineering Institute (USA), Intelligent Support Ltd. (United Kingdom), Solari (Hong Kong), Minteos (Italy) and Department of Microelectronic Engineering, University College Cork (Ireland)

References

[1] 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.

[2] P. van de Brand, M. A. Reniers, and P. J. L. Cuijpers, "Linearization of hybrid processes," Eindhoven University of Technology, Department of Computer Science, The Netherlands, Tech. Rep. CS-Report 04-29, 2004.

[3] Y. S. Usenko, "Linearization in μCRL," Ph.D. dissertation, Eindhoven University of Technology, 2002.

[4] μCRL, http://homepages.cwi.nl/~mcrl/.

[5] J. C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu, "CADP - a protocol validation and verification toolbox," in *Proceedings 8th Conference on Computer Aided Verification (CAV'96)*, ser. Lecture Notes in Computer Science, vol. 1102, 1996, pp. 437–440.

[6] CADP, http://www.inrialpes.fr/vasy/cadp/.

[7] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*. Boston: Addison Wesley Professional, 2003.

[8] SPIN, www.spinroot.com/.

[9] J. C. M. Baeten and C. A. Middelburg, *Process Algebra with Timing*, ser. EACTS Monographs in Theoretical Computer Science. Springer-Verlag, 2002.

[10] D. A. van Beek, K. L. Man, M. A. Reniers, J. E. Rooda, and R. R. H. Schiffelers, "Syntax and semantics of timed Chi," Eindhoven University of Technology, Department of Computer Science, The Netherlands, Tech. Rep. CS-Report 05-09, 2005.

[11] K. L. Man and R. R. H. Schiffelers, "Formal specification and analysis of hybrid systems," Ph.D. dissertation, Eindhoven University of Technology, 2006.

[12] P. J. L. Cuijpers, "Hybrid process algebra," Ph.D. dissertation, Eindhoven University of Technology, 2004.

[13] J. A. Bergstra and C. A. Middelburg, "Process algebra for hybrid systems," *Theoretical Computer Science*, vol. 335, no. 2/3, pp. 215–280, 2005.

[14] T. Krilavičius, "Hybrid techniques for hybrid systems," Ph.D. dissertation, University of Twente, 2006.

[15] K. L. Man and M. P. Schellekens, "Mathematical modelling of digital hardware systems in Timed Chi," in *the 26th IASTED International Conference on Modelling, Identification and Control*, Innsbruck, Austria, 2007.

[16] *SystemC*$^{\mathbb{FL}}$, http://digilander.libero.it/systemcfl/.

[17] 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.

[18] G. Salaun and W. Serwe, "Translating hardware process algebras into standard process algebras - illustration with CHP and LOTOS," in *5th International Conference on Integrated Formal Methods*, Eindhoven, The Netherlands, 2005.

[19] J. C. M. Baeten, D. A. van Beek, and J. E. Rooda, "Process algebra for dynamic system modeling," Eindhoven University of Technology, The Netherlands, Tech. Rep. CS-Report 06-03, 2006.

[20] R. Mateescu and M. Sighireanu, "Efficient on-the-fly model-checking for regular alternation-free mu-calculus," *Science of Computer Programming*, vol. 46, pp. 255–281, 2003.

[21] J. F. Groote and A. Ponse, "The syntax and semantics of μCRL," CWI, The Netherlands, Tech. Rep. SEN-R9076, 1990.

[22] C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, 1978.

[23] K. Man, Draft paper, 2008, Specification and verification of hardware circuits using μCRL.

[24] Y. S. Usenko, "A comparison of SPIN and the μCRL toolset on HAVi leader," CWI, The Netherlands, Tech. Rep. SEN-R9917, 1999.

[25] mCRL2, http://www.mcrl2.org.