Compositional Verification of an ATM Bit Error Rate Monitor Circuit

Hong Peng and Sofiène Tahar

Technical Report

Dept. of Electrical & Computer Engineering, Concordia University
1455 de Maisonneuve W., Montreal, Quebec, H3G 1M8 Canada
E-mail: {pengh,tahar}@ece.concordia.ca

March 1999

Abstract

In this paper, we propose a practical compositional verification approach based on symbolic model checking. The verification here is automatically and mechanically checked using the VIS (Verification Interacting with Synthesis) tool. Based on the proposed compositional verification approach, we verified an Asynchronous Transfer Mode (ATM) Bit Error Rate MONitor (BERMON) circuit, which entire state space is beyond the capability of VIS and the memory available in the machine we used. Through out this paper, we use the BERMON circuit to illustrate the proposed abstraction and compositional reasoning techniques.

Contents

1 Introduction

Many finite state systems consist of multiple processes running in parallel. It is often the case that we cannot verify the whole system due to state space explosion. However, the specifications of such systems can often be decomposed into local properties that describe the behavior of small parts of the system. If we can deduce that the system satisfies each local property, and if we know that the conjunction of the local properties implies the overall specification, then we can conclude that the complete system satisfies this specification as well. This leaves us the obligation of proving that the component specifications in turn imply the specification of the entire system. This is called compositional verification [11].

There are a number of difficulties involved in the above reasoning. First, we must be able to check that a specific component satisfies a given local property. Usually, the given property can only be satisfied under certain environment constrains. We hence need to construct such an environment during the verification process. Second, we must also verify the environment itself to guarantee its soundness. In addition, we must provide a reasoning rule for checking that the given property is valid in the real implementation in case some abstraction modules are used in the verification process. These difficulties are called environment problems [11] in the compositional verification.

In this paper, we propose a practical compositional verification approach based on symbolic model checking. The proof here is automatically and mechanically checked using a symbolic model checking tool, VIS (Verification Interacting with Synthesis) [3]. Applying this approach, we succeeded the verification of an Asynchronous Transfer Mode (ATM) Bit Error Rate MONitor (BERMON) circuit which entire state space is beyond the capability of VIS and the memory available in the machine we used (Sun Ultra2 with 768MB memory).

Related work on compositional verification can be classified into three major categories, namely *compositional minimization*, *assume-guarantee reasoning* and *circular reasoning* [11]. The advantage of compositional minimization is that the reasoning rule is simple and completely automatic [4]. Its disadvantage, however, is that no abstraction is used hence not allowing much model reduction. The assume-guarantee style reasoning [5] supports hierarchical verification, in which a "specification" at one level becomes an "implementation" at the next level. The disadvantage of the assume-guarantee reasoning is that assumptions must be discharged relative to other assumptions, and hence may lead to infinitum [5]. In the circular reasoning approach, the proof of a property holds under the assumption that a complement property holds assuming the former one [1] [2] [10]. This kind of circular reasoning rule leaves us the obligation of breaking the circularity.

In this paper, we adopt a circular reasoning which is similar in principle to the work in [2] [7] [1] [10]. The circularity, however, is broken by modeling the environment as an abstract module, hence yielding a conceptually simpler reasoning rule. The abstract modules are constructed using nondeterministic machines, which only describe the behavior of the parts of modules that are visible to their environments.

These abstract modules are kept at the same RT level as the actual implementation and the desired property is directly mapped to a specific module without any intermediate abstraction layer between them. In contrast to most related work found in the literature, we illustrate the proposed approach in the verification of an RTL (Verilog) implementation of a large communication circuit that can be synthesized automatically into an FPGA or ASIC design.

The rest of the paper is structured as follows. We begin with an overview of the proposed compositional verification approach (Section 2). Then, we introduce the implementation of the BERMON circuit (Section 3) and an attempt to verify a set of basic properties directly in VIS (Section 4). Next, using the compositional verification approach, we show how the verification of the BERMON circuit can be completed (Section 5). Finally, we conclude the paper with some comments on the proposed compositional verification approach (Section 6).

2 The Compositional Approach

In this section, we describe our compositional reasoning rule and introduce the required environment abstraction techniques.

2.1 The Reasoning Rule

In the simplest case, assume we have a closed system composed of two FSM modules M_1 and M_2, which communicate with each other via some interface channels (Figure 1).

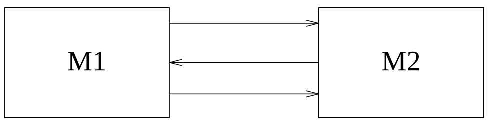

Figure 1: Two reactive machines

We denote the parallel composition of M_1 and M_2 as $M_1||M_2$ and propose to verify $M_1||M_2 \preceq P$, where $M \preceq P$ means a model M satisfies the desired abstract specification (property) P. P can either be a formula [5] or an FSM module [7]. In general, $\preceq$ is regarded as a preorder relation [5] [2].

Let M_2' be an abstract version of M_2 that characterizes only the interface behavior of M_2 that is visible to M_1 (Figure 2). M_2' hides the implementation details of M_2 that are not visible to M_1 and accordingly has a relatively smaller state space. We

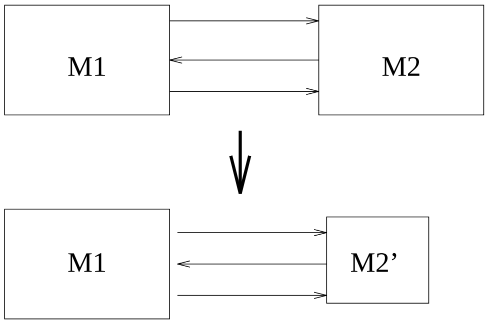

Figure 2: Abstraction of a Module

can now reason according to following inference rule:

$$\frac{\begin{array}{rcl} M_2 & \preceq & M_2' \\ M_1 \parallel M_2' & \preceq & P \end{array}}{M_1 \parallel M_2 \preceq P}$$

meaning that if M_2 can satisfy M_2', and $M_1 \parallel M_2'$ satisfies P, then $M_1 \parallel M_2$ satisfies P. The state space of $M_1 \| M_2'$ is much smaller than that of $M_1 \| M_2$ and hopefully can be handled by a model checking tool.

Based on the above general rule, we propose the following composition theorem.

[**Composition Theorem**] *Let Ω be a closed system composed of n implementation modules M_i $(i = 1 \ldots n)$, i.e., $\Omega = \|_{i=1}^{n} M_i$. P_k is a desired abstract specification (property) of module M_k. We define E_k^I as the abstraction of all input behaviors from the environment of M_k to the module M_k and E_k^O as the abstraction of all output behaviors from M_k to its environment. M_k should satisfy P_k assuming all other modules $M_j, j \neq k$ as its environments.*
If
(1) $M_k \| E_k^I \preceq P_k \| E_k^O$;
(2) $\forall j \in [1:n], j \neq k; M_j \| E_j^I \preceq E_j^O$;
then, $\Omega \preceq P_k$

Proof [1] We can simply prove this theorem as follows given any preorder relation.
Because Ω is closed, there are no external input/output signals to $\|_{i=1}^{n} M_i$, i.e.,
$\|_{i=1}^{n} E_i^I = \|_{i=1}^{n} E_i^O$; (a)
Since $\preceq$ is a preorder relation [2] [1], we derive from (1) and (2)
$\|_{i=1}^{n} M_i \|_{i=1}^{n} E_i^I \preceq P_k \|_{i=1}^{n} E_i^O$; (b)
Now from (a) and (b), we can deduce with $\Omega = \|_{i=1}^{n} M_i$:
$\Omega \preceq P_k$; ▮

[1] Another proof of this compositional theorem can be found in appendix A

From this composition theorem, two corollaries can be derived as follows.

[**Corollary 1**] *M_1 and M_2 are implementation modules building a closed system. P is the desired abstraction specification (property) of M_1, for which the environment constraint is M_2. There exists a module X and a module Y, where the state space of X and Y is smaller than that of M_2 and M_1 respectively. If*
(1) $M_1 \| X \preceq P \| Y$;
(2) $Y \| M_2 \preceq X$;
then, $M_1 \| M_2 \preceq P$

Proof We can let X be the module abstracting the behaviors of output signals from M_2 to M_1 and Y be the module abstracting the behaviors of output signals from M_1 to M_2 at their interfaces (Figure 3). X is the part of M_2 which is visible to M_1 while Y is the part of M_1 which is visible to M_2, i.e., $(Y = E_1^O = E_2^I)$ and $(X = E_2^O = E_1^I)$.

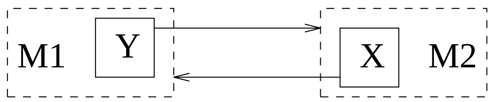

Figure 3: Two modules with X and Y

Since X is an output abstraction of M_2, the state space of X is supposed to be smaller than that of M_2. Because of the same reason, the state space of Y is smaller than that of M_1. From the composition theorem described above, we can deduce $M1 \| M2 \preceq P$. ∎

[**Corollary 2**] *Let M_1 and M_2 be two implementation modules building a closed system. P is the desired abstraction specification (property) of M_1, which environment constraint is M_2. M_2 is a closed module, namely M_2 has no inputs. There exists a module X such that the state space of X is smaller than that of M_2. If*
(1) $M_1 \| X \preceq P$;
(2) $M_2 \preceq X$;
then, $M_1 \| M_2 \preceq P$

Obviously, this corollary is a special case of $Corollary$ 1, where M_2 is a closed system (Figure 4). The proof of $Corollary$ 2 can easily be deduced from the composition theorem, where $X = E_1^I = E_2^O$.

Figure 4: Two modules with X

2.2 Environment Abstraction

According to the composition theorem and in order to avoid state space explosion and break the circularity, we have to construct an abstract specification from the implementation module that characterizes just the behavior of this module which is visible to its environment via the interfaces connecting them. The abstract specification module hides internal details which are not visible to its environment. Because the output of the abstract specification module has the same output ports as the implementation module, it is possible to construct $M \| X$, where M is an implementation module and X is the abstract specification module of M's environment.

We propose to use *nondeterministic* machines as abstract specification modules. A nondeterministic machine has no internal implementation but only output ports. It is only of interest to know the output patterns, but not how these patterns are created. These patterns have nondeterministic values. The drawback of using a nondeterministic machine, however, is the fact that such a dummy machine is not synthesizable. The advantage, on the other hand, is its ability to reduce the entire state space during the verification, which is a key issue in the model checking.

In the following sections, we will show the practicability of this compositional verification approach using an ATM Bit Error Rate MONitor (BERMON) circuit as a case study.

3 The BERMON Circuit

Protocol Data Units (PDUs) transported through an ATM network must be segmented into short, fixed-length packets, called cells, at the source end and reassembled at the destination end. Several such adaptation layer protocols are defined by the ATM standards [8], which specify how the process is carried out. ATM Adaptation Layer 5 (AAL5) is one of the ATM standards which is used in the frame-relay services or the signaling information transmission. It has strong bit error detection capabilities because of its 32 bit cyclic redundancy checking (CRC). It also makes better use of the bandwidth, because, unlike other adaptation layer protocols, AAL3/AAL4, it does not require overhead in any of the cells, except for some bytes in the information field of the last one. For these reasons, AAL5 tends to be the preferred method for breaking PDUs into cells. Because data integrity must be ensured end to end, a standard CRC technique is utilized on the entire AAL5 PDU. At the remote end, while reassembly is performed, the PDU is checked for data integrity. The polynomial chosen by the relevant standards bodies to implement the AAL5 CRC is the 32-degree polynomial:
$p(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$
All the information in a PDU received at the remote end will be divided by the polynomial. The remainder equals to Hex C704DD7B meaning that the PDU is an exact copy of that one in the source end and no error occurs. The CRC checking and generation are generally carried out on a per-byte basis [8], in an attempt to cope

with the dramatic increase of the data throughput of higher-speed lines.

The Bit Error Rate MONitor (BERMON) module is a part of the AAL5 error detection system, which is shown in Figure 5. It acts as a monitor of incoming AAL5 PDUs on an ATM connection. There is a counter in the BERMON to record the number of PDUs with CRC errors. Whenever the incoming PDU contains an CRC errors according to the CRC checking, the BERMON will increase the counter by one. The FIFO is used to buffer incoming AAL5 PDUs. The CTRL is a controller which can access the counter. It also can perform some maintenance functions of the BERMON, like for example a forced reset.

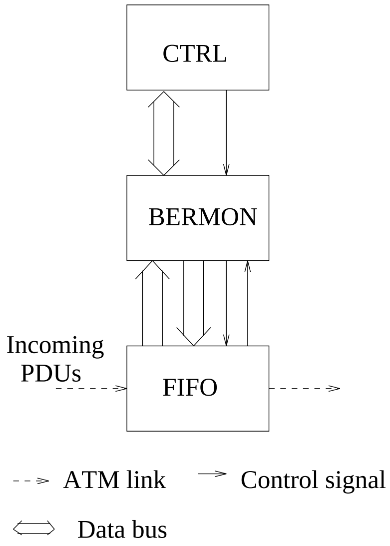

Figure 5: AAL5 Bit Error Detection

For monitoring the bit errors of an AAL5 PDU, the PDU is first read byte by byte from the input buffer in the FIFO module. At the same time, the BERMON calculates the CRC value of the PDU on a per-byte basis. The calculation of the CRC value of a PDU is done according to the ITU-I.363 protocol standard [8]. Since the BERMON cannot change the content of a PDU as a monitor, the PDU should be sent back to the output buffer in the FIFO. The number of PDU errors is stored in the counter, which is visible to the controller, CTRL. The CTRL can read this counter and reset it to zero.

The BERMON includes four modules, which are shown in Figure 6. The *Sysres* module is responsible for producing a synchronous system start signal. The *Crceng* module is the 32 bit CRC checking engine in the BERMON. It checks whether the CRC of an incoming AAL5 PDU complies with the ITU-I.363 standard [8]. It works on a per-byte basis. Since an AAL5 PDU may contain many bytes, normally 48*N bytes, the byte based CRC checking results will be stored in a shift register array in the *Movavg* module. The *Main* module decides whether the CRC of the whole PDU is positive depending on the values in the shift register array. The *Main* module can

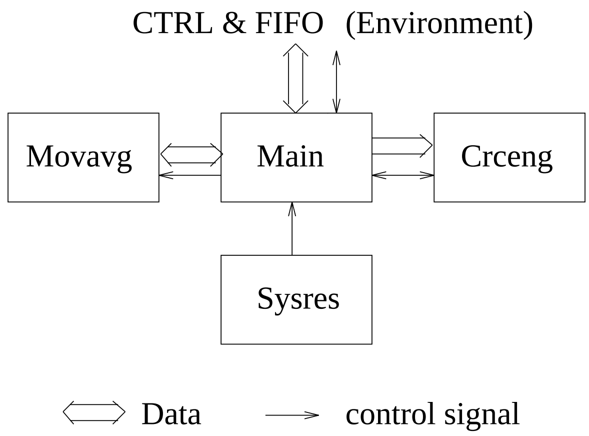

Figure 6: Structure of the BERMON

also control the read/write operations from/to the FIFO and coordinate *Crceng* and *Movavg* according to the CTRL.

4 Modeling and Verification of the BERMON in VIS

The original RTL design of the Bit Error Rate MONitor (BERMON) circuit was available in VHDL [6]. In our work, we translated the VHDL model to Verilog, which is the front-end of VIS. Besides the translation, we also reorganized the modules into equivalent medium size blocks in Verilog, while keeping the circuits behaviors unchanged. This should ease future compositional verification as the original VHDL design was composed of one very large module surrounded by several small modules.

Once the model established, we checked its state space in VIS and realized through extensive experiments (See appendix D) that the maximum reachable states of the BERMON cannot be reached in VIS due to an "out of memory" error. The experiment was performed on a 296 MHz Sun Ultra-2 workstation with 768MB of memory (All subsequent experimental results in this paper are done on this same machine). Generally, the capability of VIS is about 100-200 latches, while the BERMON model has more than 375 latches (See appendix D). This means that the verification of some properties may not be completed in VIS.

In following, we expose a set of properties describing relevant features of the BERMON circuit. For illustration purposes, we present here three example CTL properties. Interested readers can refer to appendix F for a larger set.

Property 1: Once the CTRL sends a forced-reset command *55* to the BERMON, the PDU error counter is reset to 0.

$AG((d[7:0] = 01010101) \rightarrow errcount = 0);$

d [7 : 0] is the 8-bit data bus between the BERMON and the CTRL. The CTRL puts a 01010101 data on the bus meaning that a forced reset of the BERMON is requested. In that case, the PDU error counter, *errcount*, should be reseted to 0.

Property 2: Since the incoming PDUs are nondeterministic, the per byte basis CRC result eventually will be all zero, meaning that the incoming PDU contains errors.

$$AG(AF(movavg[127:0] = 0));$$

The *movavg* [127 : 0] is the shift register array used for storing the per byte basis CRC result. When the CRC is positive, one bit of the *movavg* will be set to 1. Otherwise, it will be set to 0.

Property 3: Once there is no CRC error in an incoming PDU, the BERMON will send 0 to the CTRL.

$$AG(EF(cdat[7:0] = 0));$$

The *cdat* contains the value of the error counter, which is ready to be put on the data bus between the CTRL and the BERMON. $cdat[7:0] = 0$ means no CRC errors found in the PDU.

Property	CPU Time(s)	Memory(MB)	BDD Nodes
P_1	305.1	97	46608
P_2	264.3	110	3448687
P_3	-	-	-

Table 1: Model checking of the original BERMON model

The experimental results for the verification of above properties in VIS are reported in Table 1, including the verification CPU time, the memory usage and the number of BDD nodes generated. Accordingly, while we were able to complete the verification of properties P_1 and P_2, the system ran out of memory by property P_3. We noticed that, to verify P_1 and P_2 VIS only needs to traverse the *Main* module and the *Movavg* module, respectively. To verify property P_3, however, VIS needs to traverse the entire state space of the BERMON. In next section, we show how we succeeded verifying this property using the compositional reasoning approach described in Section 2.

5 Compositional Verification of the BERMON

In this section, we demonstrate the practical application of the compositional verification approach described in Section 2 and also introduce the way how to construct a nondeterministic abstract specification module during the verification. These techniques will be illustrated through the verification of property P_3 of Section 4 which we did not succeed in verifying directly in VIS.

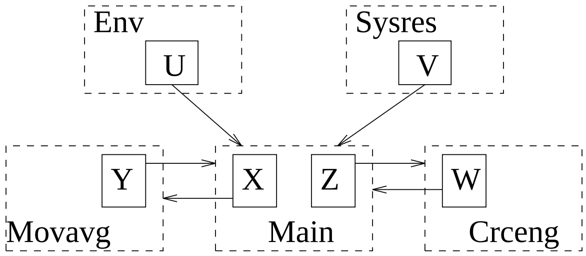

Figure 7: Abstract modules of the BERMON

Figure 7 represents the abstraction modules we used in the BERMON model. The dashed line boxes are the actual implementation modules. The solid line boxes are the environment modules, which abstract the outputs behaviors. For example, the abstraction module X abstracts the outputs behaviors from the module *Main* to the module *Movavg* at the interface between them, i.e., $X = E^I_{Movavg} = E^O_{Main}$.

For the compositional verification of the BERMON, we define the nondeterministic abstract specification modules as illustrated in Figure 7. Our goal is to verify $Main\|Env\|Movavg\|Crceng\|Sysres \preceq P_3$
Using these abstraction modules and according to the composition theorem of Section 2, this equation can be verified by applying following inference rule.

$$\begin{array}{rcll}
Sysres & \preceq & V & (1)\\
Env & \preceq & U & (2)\\
Crceng\|Z & \preceq & W & (3)\\
Movavg\|X & \preceq & Y & (4)\\
Main\|V\|Y\|W\|U & \preceq & X\|Z\|P_3 & (5)\\
\hline
Main\|Sysres\|Movavg\|Crceng\|Env & \preceq & P_3 &
\end{array}$$

If we can verify the soundness of above equations (1) to (5), we can conclude that the BERMON satisfies property P_3.

5.1 Construction of Abstraction Modules

Since we know all the possible behaviors of the I/O signals between the BERMON modules, we can use nondeterministic machines to construct abstract specifications for U, X, Y, Z and W, respectively. In following, W is used as an illustration example.

W is the partial behavior of the module *Crceng*, which is visible to the module *Main*. There is only one output signal from *Crceng* to *Main*, namely *oksig* [6]. If the CRC of the incoming PDU is positive, *oksig* is *high*. Otherwise, *oksig* is *low*. The I/O signals of the *Crceng* module are shown in Figure 8. The initial value of *oksig* is *low*.

Figure 8: I/O structure of the Crceng module

Because the data input port is one byte wide and the input PDU is at least 48-byte long, it will take at least 6 clock cycles to read in a whole PDU. If the CRC of the 48-byte input PDU is positive, *oksig* will have the pattern shown in Figure 9 (a), namely *oksig* will be *high* after at least 6 clocks. If the CRC of the 48-byte input PDU is negative, the *oksig* will be *low* after at least 6 clocks (Figure 9 (b)). It means that *oksig* can be either 1 or 0 after every 6 clocks depending on the CRC checking result.

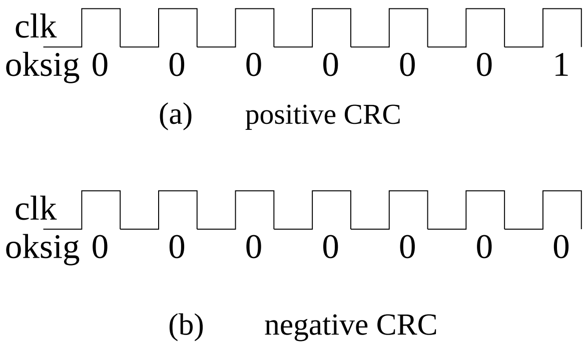

Figure 9: I/O behavior of the Crceng module

The states space of W is much smaller than that of the actual *Crceng* module (See appendix B and C). Using this same abstraction approach, we constructed the abstract specification modules of X, Y, Z and U, respectively.

5.2 Compositional Verification of the BERMON in VIS

Once the abstraction modules established, we use our compositional theorem to verify property P_3 against the BERMON model. Accordingly, we need first check the equations (1) to (5) in VIS. The experimental results of the verification obtained are in Table 2, including the CPU time, memory usage and the number of BDD nodes generated.

Equation	CPU time (s)	Memory usage (MB)	BDDs
$Sysres \preceq V$	0.1	0.24	21
$Env \preceq U$	0.3	3.25	6332
$Crceng \| Z \preceq W$	252282.2	143	374575001
$Movavg \| X \preceq Y$	233.9	124	2209944
$Main \| Sysres \| Y \| W \| U \preceq X \| Z \| P_3$	114431.8	58	1572712983

Table 2: Compositional verification of property 3

Because the state space of above decomposed modules are manageable (See appendix E), all verification succeed, hence proving equations (1) to (5) are sound. According to the composition theorem in Section 2, we can conclude that

$Main \parallel Movavg \parallel Crceng \parallel Sysres \parallel Env \preceq P_3$

Namely, the BERMON circuit seen as the parallel composition
$Main \parallel Movavg \parallel Crceng \parallel Sysres$ satisfies property P_3 under the environment *Env*.

6 Conclusions

Model checking in VIS is not directly scalable to large designs due to state space explosion. It is important to find techniques that can be used in conjunction with such verification tools in order to extend the size of the systems that can be verified. In this paper, we presented a general compositional verification approach which we applied to verify a BERMON circuit implementation. The original model of BERMON was too large to be directly verified in VIS. During the verification, the environment of an FSM module is regarded as an abstract specification module so that it can be modeled by a nondeterministic machine. Then, the original circuit implementation model of the BERMON is reconstructed by using its nondeterministic abstract environments and the implementation module to be verified. The state space of the reconstructed module is much smaller than that of the original one and can be successfully traversed by VIS. Finally, the verification of the abstract environment is done according to the composition theorem in Section 2.

In this work, we conduct our compositional verification approach on an RTL synthesizable Verilog model. The abstract specification modules are also kept at the same RT level as the actual implementation. The verification is hence performed without the need of intermediate abstraction layers between specification and implementation as in [9]. Given the simplicity of our abstraction approach, it is possible to systematically extract these abstract modules from the original design documents.

The framework we described in this paper also has some limitations. For instance, the verification using nondeterministic machines is CPU time and memory consuming in a model checking tool as VIS, which is optimized for deterministic systems. All nondeterministic behaviors in the model actually are simulated by the tool in terms of deterministic behaviors which is a costly procedure. We intend in our future work to investigate whether the proposed compositional approach can permit the verification of much more complicated systems.

Acknowledgments

We would like to thank J. M. Smith and I. Hadzic at University of Pennsylvania, for providing the original BERMON VHDL model. This work is partially funded by a Concordia University graduate student scholarship and an NSERC research grant no. OGP0194302.

References

[1] M. Abadi and L. Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, May 1995.

[2] R. Alur and T. Henzinger. Reactive modules. In *Proc. 11th IEEE Symposium on Logic in Computer Science*, pages 207–218. IEEE Computer Society Press, 1996.

[3] R. K. Brayton et al. VIS: A system for verification and synthesis. In T. Henzinger and R. Alur, editors, *Eigth Conference on Computer Aided Verification (CAV'96)*, pages 428–432. Springer-Verlag, Rutgers University, 1996. LNCS 1102.

[4] E. M. Clarke, D. E. Long, and K. L. McMillan. Compositional model checking. In *Proc. 4th Annual Symposium on Logic in Computer Science*, pages 353–62, Pacific Grove, CA, June 1989.

[5] O. Grumberg and D. E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, May 1994.

[6] I. Hadzic and J. M. Smith. P4: A platform for FPGA implementation of protocol boosters. In *Proc. 11th International Workshop on Field Programmable Logic and Applications (FPL'97)*, Sept. 1997.

[7] T. Henzinger, S. Qadeer, and S. K. Rajamani. You assume, we guarantee: Methodology and case studies. In *Proc. Computer Aided Verification (CAV'98)*, Vancouver, BC, Canada, June/July 1998.

[8] ITU-T. *Recommendation I.363 on ATM Adaptation Layers,.* International Telecommunication Union (ITU) Telecommunication Standardization Sector, Geneva, Switzerland, July 1992.

[9] K. L. McMillan. Compositional rule for hardware design refinement. In *Proc. Computer Aided Verification (CAV'97)*, pages 24–35, Haifa Israel, June 1997.

[10] K. L. McMillan. Verification of an implementation of Tomasulo's algorithm by compositional model checking. In *Proc. Computer Aided Verification (CAV'98)*, Vancouver, BC, Canada, June/July 1998.

[11] H. Peng and S. Tahar. Survey on compositional verification. Technical report, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, November 1998.

Appendix

A Another Proof of the Compositional Theorem

Here, we propose another proof of the compositional theorem of section 2.

[Definition] *A closed system Ω is composed of n implementation modules M_i ($i = 1 \ldots n$). An abstraction environment E_Ω is an $N \times N$ array defined as follows,*

$$E_\Omega = \begin{pmatrix} E_{11} & E_{12} & \ldots & E_{1,(n-1)} & E_{1n} \\ E_{21} & E_{22} & \ldots & E_{2,(n-1)} & E_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ E_{(n-1),1} & E_{(n-1),2} & \ldots & E_{(n-1),(n-1)} & E_{(n-1),n} \\ E_{n1} & E_{n2} & \ldots & E_{n,(n-1)} & E_{nn} \end{pmatrix}$$

where E_{ij} stands for an abstraction of all interface behaviors from module M_i to M_j. If there are no signals from module M_i to module M_j, E_{ij} is denoted with 0. Obviously, we also have $E_{ij} \| 0 = E_{ij}$.

E_Ω can be seen as a row vector or a column vector as follows,

$$\begin{pmatrix} E_0^I & E_1^I & \ldots & E_n^I \end{pmatrix}$$

where $E_i^I = E_{1i} \| E_{2i} \| \ldots \| E_{ni}$, or

$$\begin{pmatrix} E_0^O \\ E_1^O \\ \vdots \\ E_n^O \end{pmatrix}$$

where $E_i^O = E_{i1} \| E_{i2} \| \ldots \| E_{in}$.

From above definitions, E_i^I is obviously the abstraction of all input behaviors from the environment of M_i to the module M_i, and E_i^O is the abstraction of all output behaviors from M_i to its environment.

[Composition Theorem] *Let Ω be a closed system composed of n implementation modules M_i ($i = 1 \ldots n$), i.e., $\Omega = \|_{i=1}^n M_i$. E_Ω is the abstraction environment array of Ω. P_k is a desired abstract specification (property) of module M_k, $k \in [1:n]$. We define E_k^I as the abstraction of all input behaviors from the environment of M_k to the module M_k and E_k^O as the abstraction of all output behaviors from M_k to its environment. M_k should satisfy P_k assuming all other modules $M_j, j \in [1:n], j \neq k$ as its environments.*

If
(1) $M_k \| E_k^I \preceq P_k \| E_k^O$;
(2) $\forall j \in [1:n], j \neq k; M_j \| E_j^I \preceq E_j^O$;
then, $\Omega \preceq P_k$

Proof We can simply prove this theorem as follows given any preorder relations [2] [1].
From (1) and (2), we can deduce using the compositional inference rule:
$(\|_{i=1}^{n} M_i)\|(\|_{i=1}^{n} E_i^I) \preceq P_k\|(\|_{i=1}^{n} E_i^O)$ (a).
Because
$E_i^I = E_{1i}\|E_{2i}\| \ldots \|E_{ni}$ and $E_i^O = E_{i1}\|E_{i2}\| \ldots \|E_{in}$, $i \in [1:n]$,
From (a), we can deduce:
$(\|_{i=1}^{n} M_i)\|(\|_{i=1}^{n}\|_{j=1}^{n} E_{ji}) \preceq P_k\|(\|_{i=1}^{n}\|_{j=1}^{n} E_{ij})$ (b).
Since $(\|_{i=1}^{n}\|_{j=1}^{n} E_{ji})$ and $(\|_{i=1}^{n}\|_{j=1}^{n} E_{ij})$ both equal to the parallel ($\|$) sum of the entries of the environment array, and $E_{ij}\|\mathit{0} = E_{ij}$, we obtain
$(\|_{i=1}^{n}\|_{j=1}^{n} E_{ji}) = (\|_{i=1}^{n}\|_{j=1}^{n} E_{ij})$ (c).
From (b) and (c), we can deduce with $\Omega = \|_{i=1}^{n} M_i$ that
$\Omega \preceq P_k$ ∎

B An Example of Nondeterministic module

Let A and B be two systems (Figure 10) performing the same functions and having the same structure. There is only one output signal from the module M_1/M_2 to the module M. The only difference is that M_1 is implemented using latches, but M_2 is implemented using nondeterministic machines, which is shown in the following.

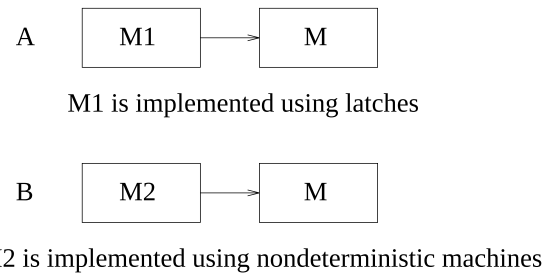

Figure 10: Abstract modules

$Program\ I:\ Verilog\ code\ of\ M_1$

```
module M1(out1);
   output out1;
   reg    out1;
   initial out1 = 0;
   always begin
      if (out1 == 0) out1 = 1;
   end
   always begin
      if (out1 == 1) out1 = 0;
   end
endmodule // M1
```

$Program\ II:\ Verilog\ code\ of\ M_2$

```
module M2(out1);
   output out1;
   wire           out1;
   assign out1 = ND(0, 1);
endmodule // M2
```

M_1/M and M_2/M are put into VIS to check their state space. The VIS experiment results are shown in Table 3, which includes size of the state space, the number of the combinational gates, the number of latches and the number of nondeterministic variables.

System	States	Comb.	Latches	ND Variables
M_1/M	15	45	4	0
M_2/M	8	33	3	1

Table 3: The size of M_1/M and M_2/M

From this table, we can see that the state space of system M_1/M is almost twice as much as that of system M_2/M.

C Construction of Abstraction Module X

X is the partial behavior of the module *Main*, which is visible to the module *Movavg*. fr is the output of *Main*. When CTRL sends a forced reset request to *Main*, the module *Main* will distribute this forced reset request to *Movavg*. Because CTRL can send a forced reset request to *Main* at every clock cycle, fr can be 0 or 1 at each clock cycle. $ldfilter$ is another output of *Main* meaning that a whole PDU has been received. It will be *high* if a whole PDU has been received, otherwise it will be *low*. Because it takes at least 6 clock cycles to receive a whole PDU, $ldfilter$ becomes *high* after at least every 6 clock cycles. The only constraint between fr and $ldfilter$ is that they cannot be *low* at the same time. The nondeterministic behavior of these two signals is described as follows.

```
module X{
output fr, ldfilter;
initial fr = 0, ldfilter = 0;
integer counter;
always(clock){
    counter = counter + 1;
    if(counter == 7){
        counter = 0;
        ldfilter = random(0, 1);}
    if(ldfilter == 1)
        fr = ND(0, 1);
    else
        fr = 1;
}
}
```

	Combinational	Latches	Reachable States
Main	1004	86	79716400
X	34	5	29

Table 4: The size of *Main* and *X*

The sizes of *Main* and *X* are shown in Table 4, including the number of combinational gates, latches and reachable states. Obviously, after the abstraction, *X* has much smaller state space than that of module *Main*.

D Reachability Analysis on the BERMON

We implemented the BERMON at the Verilog RT level. After the model was established, its state space is checked in VIS. The experiment results are shown in Table 5, including the number of combinational gates, latches, nondeterministic variables and reachable states. The experiment was performed on the same machine as before.

Combinational	Latches	ND Variables	States
35689	376	304	-

Table 5: The size of original BERMON

From Table 5, we can see that the maximum reachable states cannot be reached in VIS due to an out of memory error. Generally, the capability of VIS is about 100-200 latches, while the BERMON model has more than 375 latches. It means that the verification of some properties may not be completed due to state space explosion.

Table 6 gives the size of the decomposed BERMON modules, which includes the number of combinational gates, latches, the nondeterministic variables and the reachable states of some modules.

Models	Combinational	Latches	ND Variables	Reachable States
Sysres	19	2	0	3
Env	263	23	19	98305
$Crceng \| Z$	9827	99	3	35184400000000
$Movavg \| X$	25091	138	9	369622000
$Main \parallel Sysres \parallel Y \parallel W \parallel U$	1047	94	22	-

Table 6: The size of decomposed BERMON

Note that even after decomposition the maximum state space of $Main \parallel Sysres \parallel Y \parallel W \parallel U$ that cannot be reached in VIS meaning that there may be some properties cannot be verified due to state space explosion.

E Additional Properties of BERMON

Here, we exposed some more properties on the BERMON.

Property 4: Once the input data is correct, the PDU error counter is 0 (The input data is set in the code).

$$AGAF(errcount = 0);$$

Property 5: Once there is a PDU in the input buffer, this PDU will be fetched by the controller. $fifo_1.ff$ is the flag of input buffer full; $latchflag$ is the flag of a end of PDU reception.

$$AG((fifo_1.ff = 1) \rightarrow AF(latchflag = 1));$$

Property 7: BERMON cannot change the content of the input data. d is the array to keep the input data. q is the array to keep the output data. $dbuf$ is a temporary array in the process.

$$AG(((d[7:0] == dbuf[7:0]) \rightarrow AX(dbuf[7:0] == q[7:0])));$$

Property 8: When the state of *Main* is in *lastpdu*, a whole PDU was received. This case eventually happens.

$$AG(EF(state = lastpdu));$$

Property 9: if ef =1 and ff =1, then start the CRC engine. ef and ff are two flags in the program to start the CRC engine.

$$AGEF((ef = 1) * (ff = 1) \rightarrow (runCRC = 1));$$

The experimental results for the verification of above properties in VIS are reported in Table 7, including the verification CPU time, the memory usage and the number of BDD nodes generated.

During the above verification, no compositional approach is used because the state space is manageable. The results are conducted on the same machine as before.

Property	CPU Time(s)	Memory(MB)	BDD Nodes
P_4	252.7	48	344365
P_5	201.3	11	120156
P_6	239	34	20141
P_7	137	56	61542
P_8	240.5	59	57030
P_9	154.1	30	31432

Table 7: Model checking of correctness properties