Model Checking: Its Basics and Reality

Masahiro Fujita
Fujitsu Laboratories of America
Santa Clara, CA95054, USA
fujita@fla.fujitsu.com

Abstract— Model checking is one of the most practical techniques by which we can automatically check if given specifications (properties) are satisfied by given designs. In this paper we review various verification efforts for real designs with model checking as well as a brief introduction to the algorithms relating to model checking. The goal of the paper is to give general ideas on how model checking can be applied to real designs in which way and what kind of human interaction is necessary for practical verification.

I. Introduction

As systems to be designed become more and more complicated, it is not sufficient at all to check the correctness of designs only by simulation. Subtle design errors can easily survive even under intensive and massive simulation. Also, detecting design errors in the late design stages is extremely costly and must be avoided as much as possible. This is why now there are lots of attentions paid to formal verification.

Formal verification is to mathematically prove that the behavior allowed by given specification(properties) contains the behavior performed by designs as shown in Figure 1. It is essentially an exhaustive check on every possible behavior of designs that is related to the given specification (property to be satisfied by the designs).

Model checking is an automatic method to prove such correctness and is now becoming to be fairly widely used in real design environments. In this paper, we review basic ideas on model checking without mentioning its mathematical aspects, show advantage and disadvantage of the use of model checking for verification, and review actual verification efforts of real designs with model checking.

II. What is model checking ?

Model checking is basically an exhaustive search in all possible states in the designs by checking whether the given specification is satisfied in all of them (Figure 1). That is mostly an implicit exhaustive search on state space of designs in the sense that state space of designs are represented symbolically instead of individually. This is why state-of-the-art model checking programs can verify designs having up to 10^{100} or more states [2].

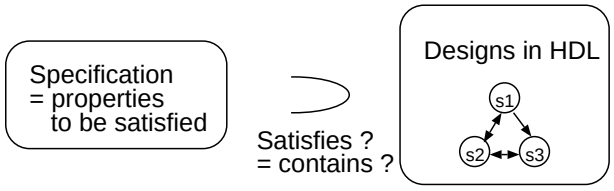

Fig. 1. Framework of model checking

Specification for model checking is a set of properties that the designs must satisfy. Property can be described either in temporal logic or automaton. Temporal logic is an extension to traditional logic with temporal operators by which we can describe relationships among variables in different time frames. Natural language description that corresponds an example of temporal logic formula is "it is always the case that if a request signal comes, there will eventually be an acknowledgment". Here, "it is always the case" and "there will eventually be" are example temporal operators and specify timings when the propositions must be held. Another example property is "if a service is started, on-duty signal will be on until that service finishes".

Another way to specify properties for model checking is to use automata. For example, if we like to specify the property: "Signal a will be not 1 in any two consecutive states", then figure 2 shows an automata that represents this property. We can use it to watch the behavior of the designs by simultaneously execute it in parallel to the designs. If we reach "bad state", then the property is not satisfied. This can be checked by model checking.

Figure 3 is an example design for the specification shown in figure 2. In model checking, each state in the designs is checked if it satisfies the property. In this example, if we check the property shown in figure 2 with the design shown in 3, each state in the design is marked as "good" or "bad" as shown in figure 3, corresponding whether a state satisfies the property in figure 2 (bad) or does not (good) . This can be done by tracing the state transitions in the design from each state along with the

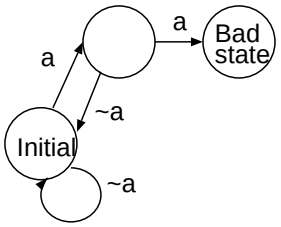

Fig. 2. An example specification based on automaton

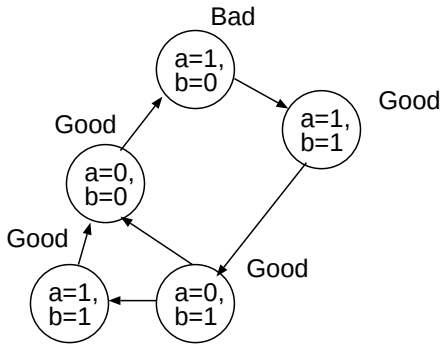

Fig. 3. An example design and its verification

specification shown in figure 2.

This process is essentially an exhaustive search on state space in designs. Therefore, if we check each state one by one, it will take exponential time with the number of flipflops or state variables in the designs. This is called an explicit state traversal. If the numbers of state to be traversed are not many (less than 1 million), this is a realistic way to verify designs. However, if the numbers of states in designs are large, explicit state traversal does not work at all. In such cases, implicit state traversal is applied. That is, the state space is represented symbolically instead of each individual state by state, and its state traversal is computed on sets of states instead of each state. Sets of states are represented as Boolean formulae and implicit state traversal can be treated as manipulation of Boolean formulae. By this implicit state traversal, large designs having more than 10^{100} can be verified within practical time ??.

III. Model checking tools

There are a number of model checking tools reported in the literature. They can be classified as: temporal logic model checkers and behavior conformance checkers.

Temporal logic model checkers are the tools that can check the properties described in temporal logic. Examples are EMC [4], CÆSAR [5], SMV [3], Spin [6], Murφ [7], UV [8], Concurrency Workbench [9], SVE [10], FORMAT [11], CV [12], HyTech [13], and Kronos [14].

Behavior conformance checkers are the tools that can check if designs represented in automaton conform the specification represented in automaton. Examples are Cospan/FormalCheck [15] and FDR [16].

Besides the aboves, there have been and there will be a number of model checking tools from various EDA vendors.

These tools offers essentially the same powerfulness/performance in the sense that all are based on the same basic algorithms. However, each tool may be tuned with particular application in mind. In that sense, performance/powerfulness of the tools for real design verification can be quite different depending on the types of designs. I.e., some are good at processor type designs whereas some others are good at cache coherence type designs.

IV. What is the realistic way to use model checking ?

Although there have been significant progress in terms of performance of model checkers since it was first proposed in early 80's, real industrial designs are often far larger than what can be verified with model checkers as they are. For example, even if we can say state-of-the-art model checkers can verify designs having more than 10^{100} states, they have just 200-300 flipflops or state variables. Typical industrial designs can easily have more than 1000 flipflops.

Moreover, the numbers of states for model checkers are not only just the numbers of states in designs to be verified, but also include the numbers of states in the peripheral systems that the designs to be verified are working with. For example, suppose we like to verify a bus arbiter circuit. It may not have many flipflops. However, it is working with several components that are connected to the bus. They can be large designs and the numbers of states including them can simply be huge.

So, real designs are just too large for model checking to verify them as they are. We need to abstract, reduce, or simplify given designs in practice before applying model checking. The problem of the current model checking tools is that this abstraction/reduction/simplification process is essentially a manual one. Although there have been active research on automatic techniques for them, they are not yet well matured to be easily used.

Examples of such abstraction/reduction/simplification process are the followings. Suppose we like to verify a bus arbiter shown in the left of Figure 4. Although we like to verify the bus arbiter only, its bus is connected to a processor, a co-processor, a memory system, and an I/O interface. In order to check properties relating to the bus arbiter, since it is working with others connected to the bus, we have to traverse state space for the entire system, not just the bus arbiter. This is simply too large, and so, we have to abstract, reduce, or simplify the processor, co-processor, memory system and I/O interface as shown

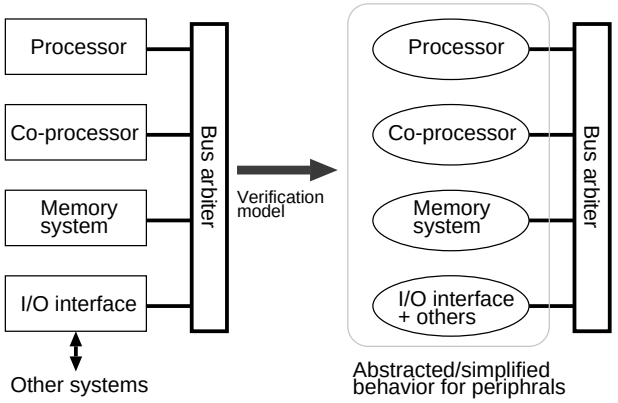

Fig. 4. An example design including bus arbiter

in the right of Figure 4. We may be able to verify the bus arbiter as it is, but we have to come up with simpler models for the components connected to the bus. This process must be a manual one right now and may take lots of human efforts.

Figure 5 is another typical situation on how we can use model checking in actual designs with abstraction/reduction/simplification process. Here we like to verify the cache coherence protocol used in the network interface chip in the figure. In this case, not only abstract, reduce, simplify peripherals, we have to extract the core of the cache coherence protocol from the network interface chip design. Moreover, we may have to come up with a simplified model for all of the other pairs of processing units and network interface chips, in order to further reduce the state space of entire system to be verified with model checking. All of these processes are currently manual ones. In other words, significant efforts are typically needed in order to use model checkers for practical designs. Of course, there can be expected lots of rewords; we may be able to find bugs in the early design stages.

V. Real verification examples

There are plenty of reports on actual application of model checking to real designs both in software and hardware [1]. We list some of them in the following with brief explanation, in order to give ideas on what can be realistically verified in which way.

The first two have some details on the goals of the verificaiton while the rest just gives brief descriptions. Especially the second verification is an example case where not only model checking but also other techniques such as theorem proving and normal simulation were applied to ensure the correctness of the entire parametric designs for ATM switch, which is one of the current direction of research on formal verificaiton.

- Cache coherence protocol on HAL parallel server [18]

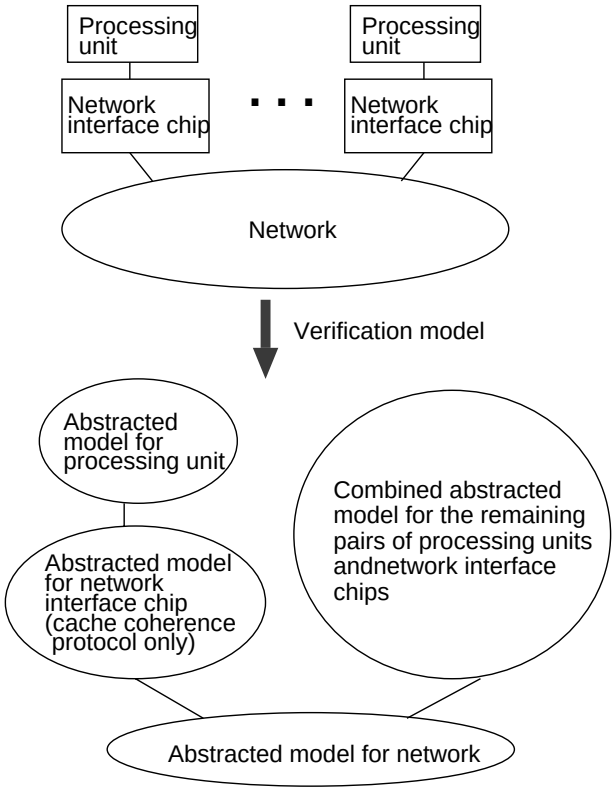

Fig. 5. An example distributed system

The cache coherence protocol of the HAL S1 System (Figure 6, a shared-memory and/or message-passing multiprocessor consisting of standard Intel Pentium Pro symmetric multiprocessing (SMP) servers connected by HAL's proprietary Mercury Interconnect to create a cache-coherent, non-uniform memory access (CC-NUMA) machine was tried to be verified with Murφ [7]. Cache coherent multiprocessors are an increasingly common architecture for high-performance servers.

Multiprocessor cache coherence protocols are hard to design and debug because they are intrinsically highly concurrent. Simulation is likely to miss important corner cases, and replicating and understanding any bugs that arise is extremely difficult. The goal was to quantify the potential usefulness of formal verification by carefully tracking the effort and results of applying formal verification, rather than simply demonstrating that verification was possible. Based on the records and experience, the conclusions that can be drawn on the costs, benefits, and appropriate methodologies of applying formal verification are: (1) if properly applied, protocol-level formal verification of cache coherence protocols has become sufficiently well-understood to be routinely undertaken by people who know model checking, (2) protocol-level formal verification, however, doesn't eliminate

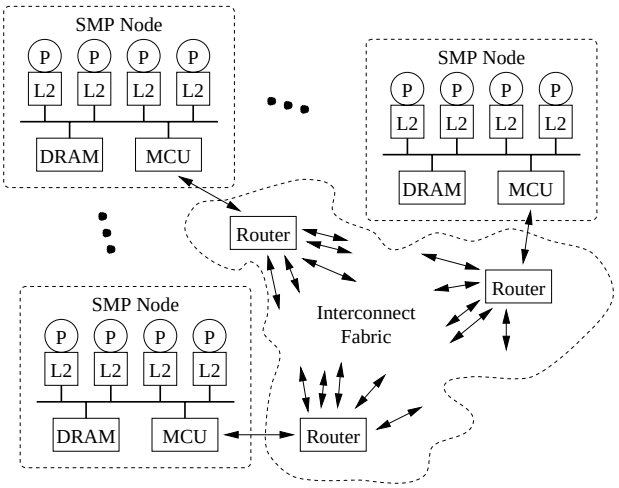

Fig. 6. Structure of the target parallel system

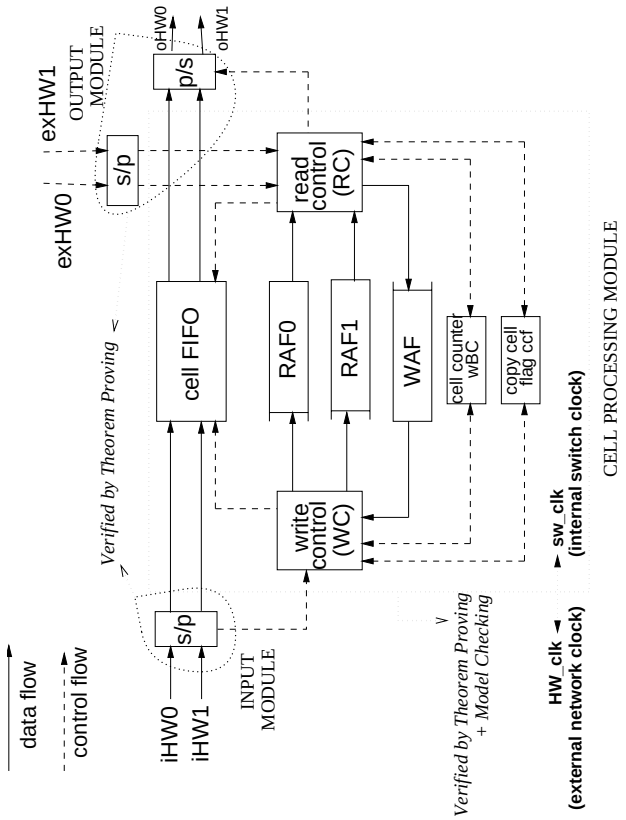

Fig. 7. Structure of the target ATM switch chip

the need for traditional implementation verification, which remains a bottleneck in the design process, and (3) problem-specific verification tools and better integration of formal verification tools into the design flow would greatly reduce the effort required.

- ATM switch chip [19] [20]

  Asynchronous Transfer Mode (ATM) technology has emerged as a backbone for high-speed broadband communications networks. An ATM network backbone typically consists of a number of small ATM switches interconnected in a matrix topology. An ATM switch takes data from input ports and forwards the input data to the proper output ports in the same order as the input data. An ATM switch is typically designed as a RAM-embedded Application Specific Integrated Circuit (ASIC) as shown in Figure 7. High-level modeling and hardware synthesis, especially if it is a parameterized modeling, delivers good results in ATM switch design. The difficulty in validation comes about because of complex control module design for a parametric number of concurrent processes, corresponding to an arbitrary number of input/output ports in the high-level model, communicating through shared signals. While simulation is efficient for validating portions of design that do not have many interacting concurrent processes, it is only by general purpose theorem proving that we can prove properties of designs with generic parameters, and model checking can efficiently handle the control part of the design with a small state space. A pragmatic combination of theorem proving and model checking in conjunction with small-scale conventional simulation was applied for efficient validation of the entire designs. With this approach all instances of the parameterized design can be verified.

- Cache coherence protocol on IEEE Futurebus+ standard [17]

  In 1992 Clarke and his students at CMU used SMV [3] to verify the cache coherence protocol in the IEEE Futurebus+ standard. They constructed a precise model of the protocol and showed that it satisfied a formal specification of cache coherence. In that process, they found a number of previously undetected errors in the design of the protocol. Although the development of the protocol began in 1988, all previous attempts to validate it were based entirely on informal techniques, and that is why there remained several errors in the protocol.

- Cache coherence protocol on IEEE Scalable Coherent Interface (SCI) [7]

  In 1992 Dill and his students at Stanford University used Murφ to verify the cache coherence protocol of the IEEE scalable coherent interface. They modeled a typical configuration using the C code in the definition of the SCI standard. Since the number of states of the model was very large, they verified only small instances of the system. Nevertheless, they found several errors, ranging from uninitialized variables to subtle logical errors. The errors also existed in the complete protocol, although it had been extensively discussed, simulated, and even implemented.

- Arbiter for multiprocessor architecture: PowerScale [21]

  In 1995 researchers from Bull and Verimag used LOTOS to describe the processors, memory controller, and bus arbiter of the PowerScale multiprocessor architecture. PowerScale is based on IBM's PowerPC and is used in Bull's Escala series of servers and workstations. They identified four correctness requirement for proper functioning of the arbiter. Correctness was established automatically in a few minutes using the CÆSAR tool.

- IBM/Motorola 60x bus protocol [22]

  IBM/Motorola 60x bus protocol was incrementally modeled at an abstract level in Verilog and verified using a model checker. The primary purpose of the modeling activity was to acquaint verification personnel with details of the 60x bus protocol and to document specific properties of the 60x bus that are necessary to guarantee compliance with hand-written protocol documents. This approach gave more precise specification than somewhat ambiguous English-language documentation.

- High-level Data Link Controller (HDLC) [23]

  A high-level data link controller (HDLC) was being designed at AT&T in Madrid. In 1996, researchers at Bell Labs offered to check some properties of the design. The design was almost finished, so no errors were expected. Within five hours, six properties were specified and five were verified, using the FormalCheck verifier. The sixth property failed, uncovering a bug that would have reduced throughput or caused lost transmissions. The error was corrected in a few minutes, and formally verified using FormalCheck.

- Token ring controller TC [24]

  A minimum configuration with Token ring controller and its peripheral, timing generator, was verified with model checking. First all reachable states from the initial state were carefully computed and then various properties were checked. The verification took some time, but all properties were verified that gave significant confidence for the correctness of the design.

- Analog control protocol for stereo components [25]

  in 1996, Bengtsoon et al. model checked a control protocol used in Philips stereo components. The protocol was originally verified manually. But here, the entire protocol was verified automatically with model checking approach.

- CCITT ISDN user part protocol [26]

  Formal modeling and automated verification were applied to the development of the CCITT ISDN user part protocol at AT&T in 1989-92. A team of five "verification engineers" formalized and analyzed 145 requirements using a special-purpose temporal logic model checker. A total of 7,500 lines of SDL source code (excluding comments) was verified. 112 errors were found; about 55% of the original design requirements were logically inconsistent.

- Control system to make buildings more resistant to earthquakes [27]

  In 1995 the Concurrency Workbench was used to analyze an active structural control system to make building more resistant to earthquakes. The control system sampled the forces being applied to the structure and used hydraulic actuators to exert countervailing forces. The first model had more than 10^{19} states and was not directly analyzable. By using semantic minimization it was possible to derive a much smaller model. A timing error was discovered that could have caused the controller that could have caused the controller to worsen, rather than dampen, the vibration experienced during earthquakes.

- Automotive chip to control the safety feature in a car [28]

  A complex automotive chip, FIRE, was verified with model checking. FIRE was used to implement safety features in a car. Smallest model necessary for the verification was automatically extracted from the RTL description for FIRE. A couple of bugs were found that demonstrated the usefulness of model checking.

VI. Concluding remarks

We have reviewed the current status of model checking tools with their verification examples. Right now the problem is that users have to spend significant time to come up with an appropriate model for the verification. Also, users must provide appropriate specification (sets of properties to be satisfied by the designs) in either temporal logic or automaton. Because of these, the model checking tools are not easily used by designers. On the other hand, once they become to be able to use the tools, they will be extremely important tools for early detection of design errors.

Currently significant researcher are conducting research on making model checking more easy to use. In the near future, model checking tools can hopefully be essential part of design process even for regular designers, not just for verification engineers.

Acknowledgments

The author would like to thank the following people for providing the information on actual verification efforts:

Edmund Clarke, Bob Kurshan, Carl Pixley, Thomas Filkorn, Kurt Ketzer, and Tom Shiple.

References

[1] E. M. Clarke, J. M. Wing, and E. Al, "Formal methods: state of the art and future directions,," *ACM Computing Survey*, Vol. 28, No. 4, pp. 626-643, 1996.

[2] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill, "Symbolic model checking for sequential circuit verification,," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, Vol. 13, No. 4, pp. 401-424, 1994.

[3] K. L. McMillan, "Symbolic model checking: an approach to the state explosion problem,," Kluwer, 1993.

[4] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications,," *ACM Trans. Program Lang. Syst.*, Vol. 8, No. 2, pp. 244-263, 1986.

[5] J. Queille and J. Sifakis, "Specification and verification of concurrent systems in CÆSAR," Proceedings of Fifth ISP, 1982.

[6] R. Gerth, D. Peled, M. Vardi, "Simple on-the-fly automatic verification of linear temporal logic," Proceedings of IFIP/WG6.1 Symposium on Protocol Specification, Testing, and Verification, June, 1995.

[7] D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang, "Protocol verification as a hardware design aid," Int. Conference on Computer Design, pp. 522-525, 1992.

[8] M. Kaltenbach, "Model checking for UNITY," Tehn. Rep. TR94-31, University of Texas at Austin, Dec., 1994.

[9] R. Cleaveland, J. Parrow, and B. Steffen, "The Concurrency Workbench: a semantics-based tool for the verification of concurrent systems," *ACM Trans. Program Lang. Syst.*, Vol. 15, No. 1, pp. 36-72, 1993.

[10] T. Filkorn, H. Schneider, A. Scholz, A. Strasser, and P. Warkentin, "SVE user's guide," Tech. Rep. ZFE BT SE 1-SVE-1. Siemens AG, Corporate Research and Development, Munich, Germany.

[11] W. Damm, B. Josko, and R. Schlor, "Specification and validation methods for programming language and sytems," Chap. Specification and verification of VHDL-based system-level hardware designs, Oxford University Press, New York, pp. 331-410, 1995.

[12] D. Deharbe and D. Borrione, "Semantics of a verification-oriented subset of VHDL," Proceedings of CHARME'95, 1995.

[13] R. Alur, T. Henzinger, and P. H. Ho, "Automatic symbolic verification of embedded systems," *IEEE Trans. Softw. Eng.*, Vol. 22, No. 3, pp. 181-201, 1996.

[14] C. Daws and S. Yovine, "Two examples of verification of multirate timed automata with KRONOS," Proceedings of 1995 IEEE Realtime systems symposium, RTSS'95, 1995.

[15] Z. Har'el and R. P. Kurshan, "Software for analytical development of communications protocols," *AT&T Bell Lab. Tech. J.* Vol. 69, No. 1, pp. 45-59, 1990.

[16] A. Roscoe, "Model checking CSP," In A classical mind: essays in honor of C.A.R. Hoare, A. Rosoe, Ed., Prentice-Hall, 1994.

[17] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness, "Verification of Futurebus+ cache coherence protocol,," Proceedings of CHDL, 1993.

[18] A. J. Hu, K. L. McMillan, and L. A. Ness, "Formal verification of the HAL S1 system cache coherence protocol," Proceedings of Int. Conference on Computer Design, 1997.

[19] S. P. Rajan, M. Fujita, K. Yuan, "High-level design and validation of ATM switch," Proceedings of IEEE International High Level Design Validation and Test Workshop, HLDVT'97, 1997.

[20] B. Chen, M. Yamazaki, and M. Fujita, "Bug identification of a real chip design by symbolic model checking," Proceedings of the European Design Automation Conference, March, 1994.

[21] G. Chehaibar, H. Garavel, L. Mounier, N. Wawbi, and F. Zulian, "Specification and verification of the powerscale bus arbitration protocol: an industrial experiment with LOTOS," Proceedings of PORTE/PSTV'96, 1996

[22] M. Kaufmann, and C. Pixley, "Intertwined development and formal verification of a 60x bus model" Proceedings of Int. Conference on Computer Design, pp. 25-30, October, 1997.

[23] J. Calero C. Roman, and G. D. Palma, "A practical design case using formal verification," Proceedings of Design-SuperCon'97, 1997.

[24] J. Bormann, J. Lohse, M. Payer, and G. Venzel, "Model checking in industrial hardware design," Proceedings of the Design Automation Conference, June, 1995.

[25] J. Bengtsson, W. Griffioen, K. Kristoffersen, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Verification of an audio protocol with bus collision using Uppaal," Proceedings of Computer Aided Verification '96, 1996. 1992

[26] G. Holzmann, "Practical methods for the formal validation of SDL specification," *Computet. Commun.* Special issue on Practical Uses of FDT's, 1992

[27] W. Elseaidy, R. Cleaveland, and J. Baugh, "Modeling and verifying active structural control systems," *Sci. Comput. Program.* 1996.

[28] J. Jang, S. Qadeer, M. Kaufmann, and C. Pixley, "Formal verification of FIRE: a case study," Proceedings of the Design Automation Conference, pp. 173-177, June, 1997.