UDC 621.3.049.771.14:681.325.65

Techniques for Effectively Applying Model Checking to Design Projects

●Tsuneo Nakata ●Satoshi Kowatari ●Hiroaki Iwashita ●Koichiro Takayama

(Manuscript received February 25, 2000)

This paper describes some techniques for applying model checking to actual design projects. Because of the rapid growth of digital systems, logic verification will be a main problem in the design flow. Although simulation-based verification has been adopted, it is widely accepted that the evolutionary progress of simulation techniques will not provide a solution to the verification crisis. We have been conducting research on formal verification, especially on model checking. We have developed some advanced techniques that should work effectively on actual designs and have applied our tool "BINGO" to some design projects. This paper compares model checking with simulation-based verification. It also describes how to use model checking techniques complementally with simulation.

1. Introduction

After continuous research over 10 years, symbolic model checking is now considered to be an effective method for verifying complex designs. It is, however, only able to handle small designs with a few hundred registers. The research on model checking has been mainly focused on verification of abstract design models rather than verification of actual designs. Most designers regard symbolic model checkers as effective tools for designs of limited areas, although some research has been conducted to overcome this situation.

This paper describes a set of fringe technologies for the application of symbolic model checking to actual designs. To fill the gap between theory and practice, we have to integrate technologies that are mainly focused upon HDL manipulations. In Chapter 2, symbolic model checking and its recent extensions are presented. Chapter 3 describes some techniques for converting and reducing design models into more effective ones, and Chapter 4 describes some techniques for extracting properties for verification. In Chapter 5, an actual application of these techniques is described as a case study.

2. Symbolic model checking

Symbolic model checking[1),2)] is a method of proving mathematically that a design satisfies a given property such that:

- The design is expressed as a finite state machine defined by a logic function.
- The property is expressed as a set of temporal logic expressions.
- The proof is done by an implicit state traversal algorithm using binary decision diagrams.[3)]

Symbolic model checking is suitable for verification of temporal relationships between signals. Some typical examples of properties are:

- Deadlocks never occur.
- An Acknowledge signal should be asserted within 12 cycles after a Request signal is asserted.
- The value of a certain register should not be updated until it is to be used.

Logic simulation has been widely used as a verification tool for hardware. However, while logic simulation requires the input vectors of the design and the quality of verification depends heavily on the quality of the vectors, symbolic model checking automatically investigates all possible behaviors and proves that the property is satisfied. Since serious design errors can be hidden among the behaviors that designers cannot easily check, symbolic model checking should, therefore, be a powerful tool for hardware verification.

The cost of computation in symbolic model checking is very large. Theoretically, it is considered to grow exponentially with the number of registers in the design. Currently available symbolic model checking tools can handle designs having up to a few hundred registers, but there may be some cases when even a few hundred is too large a number for these tools.

To overcome the exponential growth of computation, research on the following has been conducted:

- Model checking with bounded time frames.[4)]
- Model checking based on automatic test pattern generation algorithms.[5)]
- Model checking based on satisfiability check algorithms.[6)]

These three methods can handle larger designs in some cases, but they also introduce other constraints such as limits on the circuit or properties.

3. RTL manipulation techniques

Most designers now use hardware description languages to design hardware, especially at the register transfer level (RTL). **Figure 1** shows a typical verification flow for applying symbolic model checking to RTL designs. This chapter focuses on the devices used to generate an effective verification model from RTL descriptions.

3.1 Generation of finite state machines from RTL descriptions

Verification models are expressed as finite state machines (FSMs) in symbolic model checking. Since the RTL descriptions describe data transfers between registers, we can find a direct mapping from an RTL description to an FSM. In reality, since actual descriptions reflect the structures of hardware, the conversion from an RTL description to a finite state machine is not straightforward. There are three issues concerning conversion:

- Asynchronous signals
- Multiple clocks
- Bi-directional signals

In RTL descriptions, we can assume that all signals except reset should be synchronous. Since a reset mainly sets registers to initial values, we can eliminate the reset signal and assign a set of initial states to the FSM. When the design uses asynchronous signals, the transitions of the FSM should also be asynchronous, which will raise the computational complexity of verification significantly.

If a design has only one clock, we can express its behavior by a finite state machine that has transitions synchronized with the clock. When a design has two or more clocks, we have to define a virtual clock as shown in **Figure 2**, with which all of the actual clocks are synchronized. The virtual clock drives the different sets of registers, and

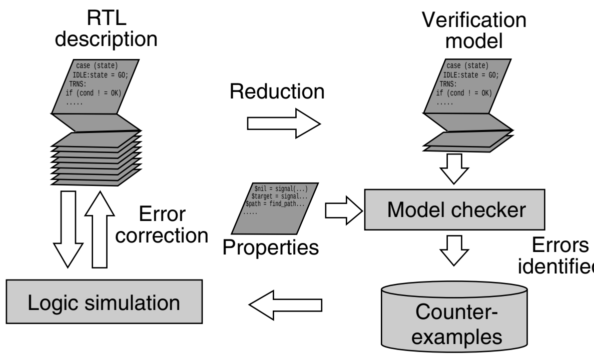

Figure 1
Typical verification flow.

the enable logic guarantees that each set is triggered correctly.

The FSM is composed of a set of registers and combinational logic. The combinational logic specifies a function that describes the transitions from the current states to the next states, and all signals in it should be unidirectional. If a design contains a bi-directional signal, it should be converted into a pair of unidirectional signals and only one of these signals should be active at a time.

3.2 Reduction of RTL descriptions

As mentioned earlier, symbolic model checking cannot be applied to large designs. Most properties for verification, however, are highly localized, so we can restrict the symbolic model checking of any particular property to the portion of the circuit where the checking will work well.

The purpose of RTL reduction is to eliminate the portions of the circuit that need not be considered for verification. Generally, symbolic model checking focuses on the verification of temporal dependencies between control signals. We can, therefore, eliminate a large portion of the circuit by ruling out most data path signals. Because RTL reduction may yield a model that has a different behavior from the original one, the reduction process should be done very carefully.

There are four basic types of reduction procedures. These have been developed based on experience with applying symbolic model checkers to actual designs. The four types are:

- Assignment and propagation of constant values
- Bit width reduction
- Full control of internal signals
- Reduction of items in enumeration data types

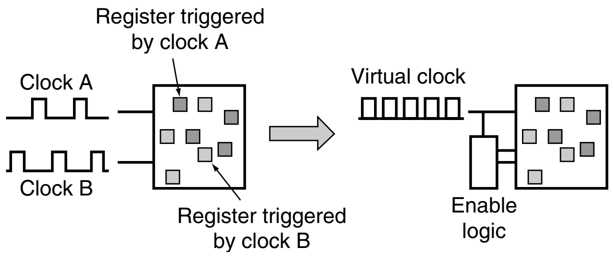

Figure 2
Virtual clock.

The first type forces the value of a signal to a certain constant and then propagates its effect. When a model has a set of exclusive behaviors, we can fix the model's behavior by setting some control signals to constant values to get a smaller model. This may cause under-approximations in verification, which means that:

- If the verifier finds an error, it is an actual error in the original model, and
- even if the verifier does not find an error, the original model is not necessarily OK.

The bit width reduction is mainly applied to data path signals. When the values on the multi-bit data path do not affect the verification process, the bit width can be reduced to 0 or 1 and the related registers can also be eliminated. This may cause over-approximations in verification, which means that:

- If the verifier says OK, it means the original model is also OK, and
- even if the verifier finds an error, we cannot say the error will occur in the original model.

If we find that the value of a register can be fully controlled from the primary inputs, we can treat the output of the register as a pseudo primary input. This may result in full control of internal signals and does not cause any approximation in verification.

Finally, the reduction of items in enumeration data types means the amalgamation of items that are equivalent under verification. In the case of verifying pipelined control for microprocessors, we do not distinguish between the ADD, SUBTRACT, AND, and OR instructions. We can therefore merge these instructions to get a smaller set, and the design size will also be significantly smaller. However, this may also cause over-approximation.

The conversion procedures are implemented as transformations of RTL descriptions. The compiler translates the RTL descriptions into an internal data structure called the "control-data flow graph (CDFG)," which contains control flows, data flows, and their relationships. **Figure 3** shows an example CDFG and its reduction. In Figure 3 (a), a statement in an RTL description is converted into a CDFG structure by the compiler. The control flow graph contains a branch node (BR), two assignment nodes (S), and a merge node (MG), which express the structure of the statement.

If we assume that signal X should not affect the properties, we can apply reduction procedures to this statement. First, the statements on the assignment to X in the data flow graph and the related nodes in the control flow graph are eliminated by bit width reduction (3-(b)) because signal X does not affect verification. This will leave a meaningless control flow graph that contains only a branch node and a merge node. If we erase the graph, then the related condition statement is also unnecessary (3-(c)). The entire CDFG, therefore, is eliminated.

The reduction procedures are implemented on our HDL compiler, and the user proceeds with the reduction step-by-step using the compiler's GUI. Although automatic reduction is strongly expected, current technology does not allow it. Designers are responsible for the overall reduction process, and the computer applies the designated conversion correctly.

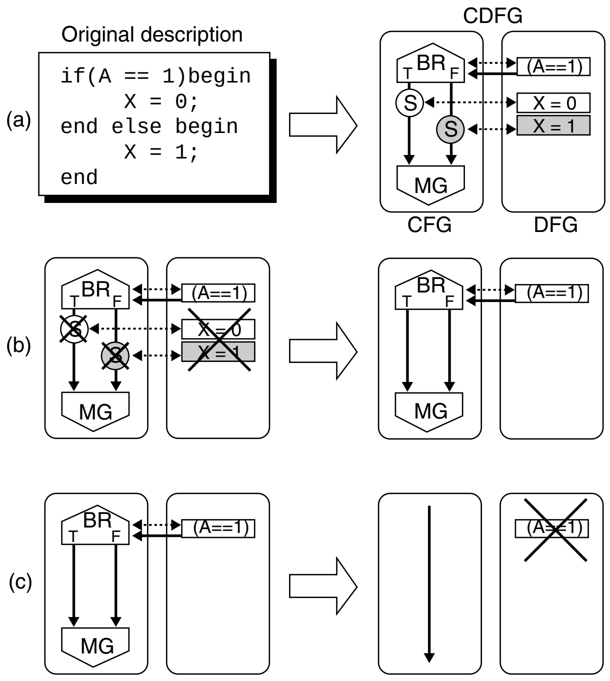

Figure 3
Example CDFG and its reduction.

4. Extraction of properties

One of the keys to successful verification is to find a good set of properties. In symbolic model checking, a property is a formula of the Computation Tree Logic (CTL).[2] The CTL is a temporal extension of propositional logic. Although it is theoretically sound, most designers think it is non-intuitive and hard to learn. Designers usually use time diagrams to express temporal dependency among signals and derive properties from them.

To describe properties, we introduced Path Set Expressions (PSEs),[7),8)] which are regular expressions for characterizing state transition sequences.[note 1)] A PSE matches an infinite sequence only when it includes the "**" operator. A list of available forms of PSEs are shown below arranged in order from the highest to lowest precedence:

- "^" matches one of the initial states.
- "." matches arbitrary single-step sequences.
- "[*f*]" matches the single-step sequence at which propositional logic formula *f* is true.
- "*P***" matches the infinite sequences that are matched by *P* an infinite number of times, where *P* must not match any infinite sequence.
- "*P*{*n*, *m*}" matches the sequences that are matched by PSE *P* at least *n* times but not more than *m* times.
- "*P*{*n*,}" matches the sequences that are matched by *P* at least *n* times.
- "*P*+" is equivalent to "*P*{1,}", "*P**" is equivalent to "*P*{0,}".

note 1) Theoretically, model checking for PSEs is classified as a variety of language containment checking.[9),10)]

Figure 4 shows the intuitiveness of the PSE properties. This time diagram expresses a property such that if signal p is asserted, then signal q should be asserted accordingly. To compose a PSE property, the total interval is divided into four subintervals according to the values of p and q. The first subinterval means "signal p is deasserted, and signal q can have an arbitrary value," which is written as "[~p]" in a PSE. A similar process is applied to the rest of the subintervals, and we can get the final PSE as shown in the figure. While this expression directly corresponds to the time diagram, the PSE can include more information than the time diagrams can do. If the interval between the assertions of p and q is 3 to 5 cycles, the PSE expression should be "[~p][p][~q]{3,5}[q]." In the time diagram, we have to add an informal comment to express this information, while the PSE can express this formally. If we use CTL for properties, we have to write the formula shown in Figure 4. Errors are likely to be made when we derive such a complex formula from this time diagram because it is quite a difficult task.

The PSEs, therefore, provide a quick and easy way to form properties systematically from specifications expressed as time diagrams and/or natural languages.

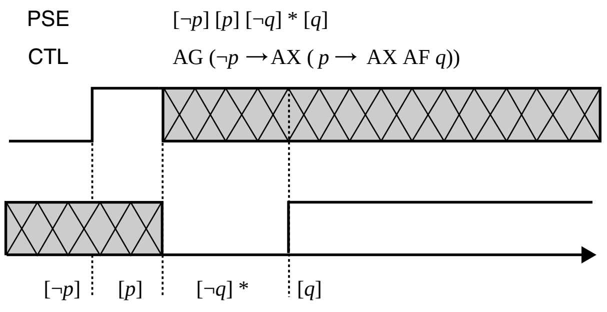

Figure 4
Expression of properties.

5. Case study: multimedia processor

We have developed a symbolic model checker called "BINGO" and an HDL reduction system and have applied them to the design projects shown in **Table 1**. This chapter describes the application of BINGO to a multimedia processor.[11] The block diagram of the processor is shown in **Figure 5**. Since the processor contained more than 10 000 registers and could not be handled by symbolic model checkers, we focused on a portion of the circuit and verified with the following steps.

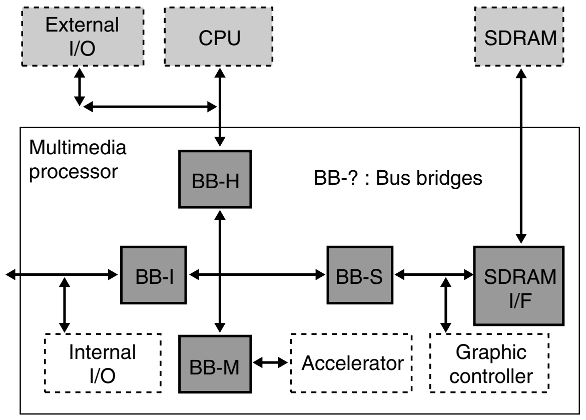

Figure 5
Multimedia processor.

Table 1
Applications of BINGO.

Design	Design level	Target of verification	Verification results
Multiprocessor system	Behavior	Cache coherence protocol	Guaranteed correctness of protocol
Network switch for servers	Behavior	Bus protocol	Guaranteed no livelocks
Multimedia processor	RTL	Bus arbitration	Revealed incomplete correction against bus arbitration errors
Switchboard subsystem	RTL	Clock synchronization	Revealed malfunction in initialization
Embedded processor	RTL	Memory/bus control	Revealed possibility of register leakage
VLIW processor	RTL	Cache control	Revealed malfunction in cancellation of instructions

- **Define target**
  The advantage of symbolic model checking is that it can be used to check complex controls exhaustively. In this case, the target was the bus arbitration algorithm, which is very complex and highly parallel. The processor has four types of busses, which are connected via bus bridges.
- **Extract a portion of the design**
  To verify the algorithm for arbitration between the bridges, the verification model had to include the four bridges and the SDRAM interface. The other units could simply be eliminated by the "full control to internal signals" reduction procedure. The original description had 60 000 lines of Verilog-HDL code and more than 10 000 registers. The extracted description had 15 000 lines and 4000 registers.
- **Reduce the verification model**
  Since we focused on the arbitration, the data path part was completely independent of the verification. Then, we could eliminate the signals and registers related to the data path by the "bit width reduction" reduction procedure. The resultant description had 9000 lines and 500 registers.
- **Extract properties**
  The eight properties were extracted from the specification related to bus accesses. For example, the data must be returned when the processor sends a request for data to SDRAM. This task is also crucial in simulation-based verification. While we have to specify a complete behavior for simulation-based verification, we can concentrate on essential information in symbolic model checking. The detailed behavior is explored automatically.
- **Apply BINGO**
  We applied the properties to the original design, which contained several design errors, and to the corrected design. In BINGO, the properties, the script for verification, and the formatting of the results are written in Perl. We obtain the verification results by running the Perl script as a form of Yes or No and obtain counterexamples in addition when the results are No.
- **Analyze the results**
  BINGO revealed the known design errors in the original design and proved that they no longer survived in the corrected design with one exception that still caused a malfunction in bus arbitration. This information was reported to the designers, and the error was corrected before fabrication. The number of states traversed in this verification process exceeded 10^{13}; a number which logic simulation is unlikely to be able to cover.

The time required for this verification process was as follows:

- Two person-months for understanding the design
- Two person-months for reduction
- Two person-months for verification (including diagnosis)

We could not prepare automatic reduction tools in time for the reduction. However, based on the experience we gained, we estimate we will be able to halve the amount of work in the future. As shown in this case study, the application of symbolic model checking to a large design takes a long time. The most work, however, was devoted to understanding the design. If the verification team knows the design well, they will get the first verification results within a month. Logic simulation can do almost nothing with the verification of complex behaviors when units work interactively. Symbolic model checking will be a sole solution in these cases and will provide results that justify the required person-months.

6. Conclusions

We have described the effectiveness of symbolic model checking for the verification of large designs. To accomplish successful results, it is mandatory to introduce RTL manipulation tools and the methodology of property extraction.

Since we cannot expect a drastic advance in the research on symbolic model checking based on BDD techniques any time soon, we have to integrate some complementary techniques with the current symbolic model checking to achieve robustness in the verification of large circuits. Possible candidates are symbolic model checking based on automatic test pattern generation and/or satisfiability check technologies with more advanced HDL manipulation techniques. Design methodologies are also important in verification. Design for verifiability will be a key issue in research and development in the near future.

References

1) K. L. McMillan: Symbolic Model Checking, Kluwer Academic Publishers, 1993.
2) E. Clarke et al.: Model Checking, MIT Press, 1999.
3) R. E. Bryant: Graph Based Algorithm for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8), pp. 677-691 (1986).
4) Biere et al.: Symbolic Model Checking without BDDs. TACAS '99, 1999.
5) V. Boppana et al.: Model Checking Based on Sequential ATPG. CAV'97 Computer-Aided Verification, pp.418-430, Springer, 1999.
6) A. Biere et al.: Symbolic Model Checking using SAT Procedures Instead of BDDs. Proc. 36th Design Automation Conference, 1999, pp.317-326.
7) H. Iwashita et al.: CTL Model Checking Based on Forward State Traversal. Proc. Int'l Conf. Computer-Aided Design, 1996, pp.82-87.
8) H. Iwashita et al.: Forward Model Checking Techniques Oriented to Buggy Designs. Proc. Int'l Conf. Computer-Aided Design, 1997, pp.400-404.
9) H. J. Touati et al.: Testing Language Containment for ω-automata Using BDD's. Proc. 1991 International Workshop on Formal Methods in VLSI Design, 1991.
10) The VIS Group: VIS: A System for Verification and Synthesis. Proc. 8th Conference on Computer Aided Verification, 1996, pp.428-432.
11) K. Takayama et al.: An Approach to Verify a Large Scale System-on-a-Chip Using Symbolic Model Checking. Proc. ICCD-98, 1998.

Tsuneo Nakata received the B.S. degree in Electronic Engineering and the M.S. and Ph.D. degrees in Information Engineering from the University of Tokyo, Tokyo, Japan in 1981, 1983, and 1986, respectively. He joined Fujitsu Laboratories Ltd., Kawasaki, Japan in 1986 and has been engaged in research and development of computer-aided design. He spent one year as a Visiting Scholar at the University of California at Berkeley from 1993 to 1994. He is a member of the IEEE and the Information Processing Society of Japan (IPSJ). His research interests include design methodology and verification of VLSIs.

Hiroaki Iwashita received the B.S. and M.S. degrees in Electronic Engineering from Osaka University, Osaka, Japan in 1989 and 1991, respectively. He joined Fujitsu Laboratories Ltd., Kawasaki, Japan in 1991 and has been engaged in research and development of computer-aided design. He spent one year as a Visiting Scholar at Carnegie Melon University from 1996 to 1997. He is a member of the Information Processing Society of Japan (IPSJ). His research interests include verification of VLSIs, especially model checking techniques.

Satoshi Kowatari received the B.S. and M.S. degrees in Electrical and Computer Engineering from Nagoya Institute of Technology, Nagoya, Japan in 1991 and 1993, respectively. He joined Fujitsu Laboratories Ltd., Kawasaki, Japan in 1993 and then moved to Fujitsu Ltd., Kawasaki, Japan in 1995. He has been engaged in research and development of computer-aided design systems. His research interests include design methodology and verification of system level design.

Koichiro Takayama received the B.E. and M.E. degrees in Electronic Engineering from Osaka University, Osaka, Japan in 1985 and 1987, respectively. He joined Fujitsu Laboratories Ltd., Kawasaki, Japan in 1987. His research interests include VLSI CAD systems and design methodologies. He is currently working with Fujitsu Laboratories of America.