Verification of Abstracted Instruction Cache of Titac2: A Case Study

Tomohiro Yoneda

Department of Computer Science

Tokyo Institute of Technology

yoneda@cs.titech.ac.jp

Abstract In this paper, we demonstrate the formal verification of a practical timed asynchronous circuit. The target circuit is obtained by abstracting the instruction cache subsystem of a real asynchronous processor, TITAC 2. We also show several techniques to improve our verification method. The improved verifier could verify the target circuit in approximately 15 minutes, using less than 20 MBytes of memory.

Keywords: Formal verification, timed asynchronous circuits, partial order reduction, time Petri nets, instruction cache.

1. INTRODUCTION

In order to avoid the various difficulties which arise in designing large synchronous circuits, such as clock skews, high power consumption, and so on, designing asynchronous circuits without any clock systems has been attracting notice. In fact, several research groups demonstrated that an entire microprocessor could be designed and fabricated in this manner[Martin et al., 1989; Furber et al., 1994; Furber et al., 1996; Nanya et al., 1994; Takamura et al., 1997]. One significant problem that asynchronous circuit designers face is a lack of CAD systems. From a verification point of view, the cost of verifying asynchronous circuits is considerably higher than that for synchronous circuits because each wire of asynchronous circuits has states, and as a result, the state spaces of asynchronous circuits are often very large. Furthermore, in re-

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35498-9_57](https://doi.org/10.1007/978-0-387-35498-9_57)

L. M. Silveira et al. (eds.), *VLSI: Systems on a Chip*

© IFIP International Federation for Information Processing 2000

cent asynchronous circuit design, designers have preferred to use timed circuits for implementing fast and compact circuits. This also makes verification difficult, and prevents us from applying the latest verification techniques for untimed systems, such as symbolic model checking[Burch et al., 1992] or partial order reduction[Valmari, 1990; Katz and Peled, 1990; Godefroid, 1990]. Thus, our recent interest has been in developing efficient verification tools for timed asynchronous circuits. VINAS-P (Verifier based on time petri Nets for timed Asynchronous Systems using Partial order reduction) is our newest formal verification tool for timed asynchronous circuits using techniques proposed in [Yoneda and Ryu, 1999]. The main idea in these techniques is partial order reduction based on the timed version[Yoneda and Schlingloff, 1997] of the Stubborn set method[Valmari, 1990]. The most closely related work is probably the ATACS which was developed at the University of Utah[Belluomini and Myers, 1998], however, these two tools employ different treatments for the trade-off between efficiency and expressibility in specifications.

The purpose of this paper is to demonstrate the verification of a practical-sized timed asynchronous circuit using VINAS-P. The target circuit is obtained from TITAC 2[Takamura et al., 1997]. TITAC 2 is a real 32-bit fully asynchronous processor which was developed at the University of Tokyo and the Tokyo Institute of Technology in 1997. It accepts almost all MIPS R2000 instructions, and contains half a million CMOS transistors. It was designed under the scalable-delay-insensitive (SDI) model, where its verification problem can be reduced to that of bounded delay circuits. Thus, it contains numerous subcircuits which are suitable as benchmark circuits for timed asynchronous circuit verification. We focus on the instruction cache subsystem of TITAC 2 because it is one of the most complicated subsystems in TITAC 2, and verifying it formally is a challenging task. However, for the formal verification, 32-bit data/address buses are too large to be handled. Therefore, we need to obtain an abstracted and simplified version of the subsystem in which many interesting properties can still be verified. This abstracted circuit contains approximately 200 gates, and its time Petri net model includes 1527 places and 1697 transitions. The original VINAS-P could handle this circuit, but it was very slow. This was mainly due to the on-line analysis needed for the partial order reduction. This paper proposes some techniques to improve the performance of VINAS-P. The improved VINAS-P could finally verify the target circuit in about 15 minutes, using less than 20 MBytes of memory.

The rest of this paper is organized as follows. Section 2 shows the overview of the TITAC 2 instruction cache subsystem, the abstracted circuit, and its specification. In Section 3, after briefly explaining the

verification method of VINAS-P, we propose several techniques for the improvement of VINAS-P, and show some experimental results. Finally, we summarize the discussion.

2. TITAC 2 INSTRUCTION CACHE

2.1 OVERVIEW

Figure 1 shows the block diagram of the TITAC 2 instruction cache subsystem.¹ The cache memory contains 256 line (or block) frames, and each line frame contains eight words. Thus, the size of the cache memory is 8KBytes ($256 \text{ line frames} \times 8 \text{ words} \times 32 \text{ bits}$). The lines are direct-mapped and fetched in the early restart manner with the critical word first [Hennessy and Patterson, 1996]. Thus, when a word with address adr is read and it is not in the cache memory, the line containing the word is fetched in the order $[adr], [adr+1], \dots, [adr+n], [adr-m], \dots, [adr-1]$, where $m = adr \bmod 8$ and $n = 7 - m$. Furthermore, access for other words within the same line is responded to as soon as the words are fetched, while all access for words not in the line is suspended until the line fetch is completed, even if the access is on a hit (see Figure 2).

A brief explanation of the operation of the instruction cache subsystem is as follows:

Figure 1 Block diagram of the TITAC 2 instruction cache subsystem.

¹All information regarding the original circuit of the TITAC 2 instruction cache subsystem and several figures are from the Bachelor's thesis of Makoto Ishikawa [Ishikawa, 1997].

Figure 2 Early restart with critical word first (line size = 4).

1. Instruction Address *adr* is given, and Tag Check signal is activated.
2. If a line which does not contain *adr* is being fetched, access is suspended until the line fetch is completed.
3. I-Cache Controller starts both the tag check operation with *adr*[31:13] and the cache memory read operation with *adr*[12:5]. The Cache Memory Module consists of eight banks. Thus, all words in the specified line frame will be read simultaneously.
4. On a hit, when the cache memory read operation is completed, the corresponding word is selected by MUX according to *adr*[4:2].
5. On a miss, Line Fetch Controller starts the line fetch. When the first word is read from Main Memory, and is written into Cache Memory Module, the word is selected by MUX. Line Fetch Controller also sets the corresponding bit in Exist Register, which indicates the available word in the line currently being fetched. When other words within this line are read, they will be selected or the read operation is suspended according to the corresponding bits in the Exist Register.

The address bus and data bus are two-rail coded, that is, each bit is represented by two wires such that (01) corresponds to 0 and (10) to 1.

The completion of the instruction cache read operation is indicated by both COMP and Instruction Dataout. The operation is completed only when both, COMP is 1 and Instruction Dataout is a code word of two-rail code (i.e., every pair of wires in Instruction Dataout is (10) or (01)). Note that it is not known which of these two events occurs first.

After the completion of every read operation, a resetting phase is necessary. This is started by setting 0 to each bit of Instruction Address and Tag Check, and is completed when COMP and each bit of Instruction Dataout become 0.

2.2 ABSTRACTED CIRCUIT

We aim to verify that the TITAC 2 instruction cache subsystem works correctly. However, it is difficult to obtain an abstracted model for such a general property. Therefore, we focus on the LSB of the instruction and the main memory location 0 and 1. That is, in this case study, we verify the following property of the instruction cache subsystem :

The LSB of the instruction read from the cache subsystem is the one stored in the location of the main memory with the given address 0 or 1 independent of the result (hit or miss) of the tag check operation.

Although this property is rather restricted, we believe that it verifies the control circuit of the TITAC 2 instruction cache subsystem almost completely, and that its data path circuit is also fairly well verified. For this property, we can easily obtain an abstracted instruction cache subsystem, denoted by AIC, which has a 1-bit address bus and a 1-bit data bus.

Furthermore, in order to check all possible sequences of misses and hits, we separate the tag check module from AIC, and give new external lines HIT and MISS to AIC, where those inputs are controlled by the specification. This means that the actual tag check module will not be verified due to this simplification.

The gate level circuit of AIC is shown in our technical report [Yoneda, 1999]. In AIC, every gate delay is assumed as [4,5], and every delay element has [100,100] delay. The text files which contain Verilog-like descriptions for this circuit as well as the above document can be obtained from <http://yoneda-www.cs.titech.ac.jp/~yoneda/pub.html>.

2.3 SPECIFICATION

Since VINAS-P is based on the trace theoretic verification[Dill, 1988], the specification needs to express the expected input and output relation. The following is an outline of our specification for the above property.

RESET is kept at 1 for a sufficient time period, and is then set to 0. Either (01) or (10) is given to Instruction Address. Tag Check is then activated. Either hit or miss operation is selected. On a hit, HIT is activated. On a miss, the main memory mode is selected. Table 1 shows the relation between the main memory mode and “address”/“data value”. Then, MISS is activated. If the main memory address changes, the corresponding data is set according to the main memory mode. At this state, either COMP or Instruction Dataout can change. When COMP changes to 1, Instruction Dataout must be either

(00) or the code word (i.e., (01) or (10)) which is correct with respect to Instruction Address and the main memory mode. In the former case, the latter must follow eventually. If Instruction Dataout becomes an incorrect code word, it will be detected as a failure state as discussed in the next section. Once Instruction Dataout becomes a code word, it must not change until Tag Check becomes 0. Instruction Dataout can change in any way (except (11)) as long as COMP = 0. After COMP becomes 1 and Instruction Dataout becomes a code word, Tag Check is set to 0 and Instruction Address is set to (00). Then, either MISS or HIT is set to 0. If COMP becomes 0, then go back to the point where Instruction Address is set. Note that the choices for the instruction address, hit/miss operation, and the main memory mode are nondeterministic.

The formal expression of this specification is also shown in [Yoneda, 1999].

Table 1 Main memory mode.

main memory mode = "normal"		main memory mode = "opposite"	
address	data value	address	data value
0	0	0	1
1	1	1	0

3. VERIFICATION

3.1 METHOD

In VINAS-P, the gates in a circuit are translated by using a gate library into time Petri net modules which model the behavior of the gates, and a specification is expressed by a time Petri net module. A time Petri net[Merlin and Faber, 1976] consists of transitions (thick bars), places (circles), and arcs between transitions and places. A token (large dot) can occupy a place, and when every input place of a transition is occupied, the transition becomes enabled. Each transition has two times, the earliest firing time and the latest firing time. An enabled transition becomes ready to fire when it has been continuously enabled for its earliest firing time, and cannot be continuously enabled for more than the latest firing time, i.e., it must fire unless it is disabled. The firing of a transition consumes tokens in its input places and produces tokens in its output places. If transitions have one or more common input places, then we say that those transitions are in conflict. Usually, the firing of such one transition disables the remaining conflicting transitions.

Verification is performed by traversing the state spaces of the set of time Petri net modules with simultaneously firing every transition

associated with the same wire. The circuit is considered to be correct with respect to the specification, if no failure state is reached in the state-space enumeration process. A failure state is a state where a module wants to change an output wire but the corresponding input wire is not ready to change in some module. More precisely, if a transition associated with an output wire can fire without an advance of time, but there exists a module in which no transitions associated with the corresponding input wire are enabled, then it is a failure state. Typically, a failure state is reached when a circuit produces a bad output which the specification does not expect. In this case, the corresponding input transition is not enabled in the specification.

The current version of VINAS-P only supports the checking of this kind of safety properties and simple deadlock checking. Although checking other properties, such as liveness, may occasionally be needed, what VINAS-P can verify covers many interesting and important properties. Other restrictions of the current version are that only $[0, \infty]$ bounds are allowed for transitions associated with input wires, and that multiple transitions associated with the same output wire cannot exist in a module. These restrictions are important for reducing the computational cost of the algorithm.

The idea of the partial order reduction is to prune some successor states in the state-space traversal as long as the correctness of the verification results is not affected. For example, when two transitions are ready to fire, as shown in Figure 3(a), generating (untimed) states such as $\{p_1, p_4\}$, $\{p_2, p_3\}$, and $\{p_3, p_4\}$ are often too much for checking the reachability of failure states. In such cases, one firing sequence generating $\{p_1, p_4\}$ and $\{p_3, p_4\}$ is sufficient. Even if a failure state is generated

Figure 3 Concurrent and conflicting transitions.

by the firing of t_1 , it is reached anyway by the above firing sequence. On the other hand, in the time Petri net shown in Figure 3(b), the firing sequence starting from t_1 may miss a failure state caused by the firing of t_3 , because the firing of t_1 eliminates the possibility of the firing of t_3 . Actually, the firing sequence starting from t_2 can make t_3 ready to fire if the firing of t_1 is postponed. Therefore, we cannot prune the successor state by t_2 in this case. In order to handle general cases, if we want to fire a transition t at a state s , we compute $\text{dependent}(s, t)$ which is a set of enabled output transitions such that the interleavings of the firings of those transitions should be generated for the correct results (for example, $\text{dependent}(s, t_1) = \{t_1\}$ for Figure 1.3(a), and $\text{dependent}(s, t_1) = \{t_1, t_2\}$ for Figure 3(b)). In VINAS-P, the computation of $\text{dependent}(s, t)$ is implemented such that the transitions which enable the transitions in conflict with t (e.g., t_3 in Figure 3(b)) in the future are searched backwards until some enabled output transition (e.g., t_2) is found.² If the given time Petri nets are large and contain a lots of conflicts, the cost to compute the dependent sets becomes very high. The verification algorithm of VINAS-P is formally described in [Yoneda and Ryu, 1999].

3.2 IMPROVEMENT

For experimental reasons, we prepare three variants of the specification of AIC, denoted by *spec1*, *spec2*, *spec3*, where

- *spec1*: Instruction Address = (01), MISS = 1 and the main memory mode “normal” are always chosen.
- *spec2*: MISS = 1 and the main memory mode “normal” are always chosen.
- *spec3*: the main memory mode “normal” is always chosen.

For example, in *spec2*, the access to the instruction address 0 and 1 are verified, but only the miss case with the main memory mode “normal” is examined. Let *spec4* denote the complete specification mentioned in Section 2.3. The costs of verification for the four different specifications vary significantly, even if the same circuit is used. Let AIC1, AIC2, AIC3, and AIC4 denote the AICs with *spec1*, *spec2*, *spec3*, and *spec4*, respectively. In this subsection, we propose several techniques to improve VINAS-P, and evaluate them by using these different verification examples. All measurement in this paper was carried out on a UNIX workstation (Pentium II 450MHz, 512MB main memory).

² t_4 is not included because it is not enabled.

Figure 4 Ineffective conflicts.

In Figure 4(a), transitions t_1 and t_2 are in conflict. In fact, if t_1 fires, then t_2 can no longer fire. However, even if t_2 fires, t_1 can still fire. Actually, if the earliest and latest firing times of t_1 are $[0, 0]$ or $[0, \infty]$, the firing of t_2 does not influence the firing of t_1 .³ Thus, the first technique for the improvement is to ignore such ineffective conflicting relations during the dependent set computation. There is, however, one exceptional case. In the case shown in Figure 4(b), suppose that t_{in} is associated with an input wire, and that t_{out} is associated with the corresponding output wire. Then, the firing of t_1 leads to a failure state, because in the state obtained by firing t_1 , an output can change but the corresponding input is not ready to change (i.e., t_{out} can fire, but t_{in} is not enabled). On the other hand, the firing of t_{out} eliminates the chance to reach the above failure state, because t_{out} is no longer enabled after firing. Hence, $\text{dependent}(s, t_{out})$ must include t_1 in order to correctly detect failure states. The current algorithm of VINAS-P achieves this by using the conflicting relation between t_{in} and t_1 (see [Yoneda and Ryu, 1999] for details). For this reason, conflicting relations involving transitions which are associated with the input wires must not be ignored, even if the firings of those transitions never affect the firings of other transitions. For example, in Figure 4(c), if t_1 is associated with an output wire, $\text{dependent}(s, t_1)$ searches the transitions backwards from t_3 but not from t_2 , while the backward searches from both are necessary in the case where t_1 is associated with an input wire.

Similarly, transitions, such as t_4 in the above example, play no role in enabling t_3 in the future, because t_4 cannot generate a new token into p_1 . Thus, we also ignore these transitions in the backward search process.

Secondly, because the same transitions are reached many times via different paths during the backward search process, we introduced a

³If t_1 has other firing times, then the firing of t_2 resets the clock attached to t_1 and the firing time of t_1 is affected by the firing of t_2 .

Table 2 Comparison of performance of improved VINAS-P.

		AIC1	AIC2
original	(sec)	1190	56494
technique 1	(sec)	145	3715
technique 2	(sec)	99	1626
both techniques	(sec)	20	214

caching mechanism for the dependent set computation. We will cache the results of backward searches at a fixed state, i.e., the cache is flushed when a new state is reached, and we do not include states s in the keys for the cache. This is because numerous backward searches are activated in each state, and it is not reasonable to prepare such a large cache area that the cached results are used when the same states (or markings) are revisited. Thus, in each cache entry, we keep a transition and the result of the backward search from the transition. It is expected that many re-backward searches from the same transition will be omitted using this caching mechanism.

Table 2 summarizes how the CPU times are reduced using the above techniques. The row for “technique n ” shows the performance values when only “technique n ” is applied where $n = 1, 2$. Since these techniques are independent, they can be applied simultaneously. The row for “both techniques” shows the performance values when both techniques are applied. Because combining these techniques reduces the cost of the dependent set computation significantly, AIC3 and AIC4 can also be

Figure 5 Performance of improved VINAS-P.

verified rather easily. Figure 5 shows the CPU times for the verification of AIC1 through AIC4 using the final version of VINAS-P. It also shows the memory usage required for each verification. The number of generated states for AIC4 was 25976.

Since TITAC 2 was designed under the SDI model, it may not work correctly in cases where some components have delays which are too large. In order to demonstrate the timing aspect of the verification, we modified the delay of one C-element in AIC4 and set it to [500,500]. Then, VINAS-P found a failure state after generating 6381 states in 196 seconds.

4. CONCLUSION

This paper demonstrated the formal verification of a practical timed asynchronous circuit. The target circuit was obtained by abstracting and simplifying the instruction cache subsystem of a real asynchronous processor, TITAC 2. We could verify an interesting property of the abstracted model. We believe that this is one of the largest benchmark examples for the formal verification of timed asynchronous circuits.

Furthermore, in order to improve our verification tool VINAS-P, we implemented several techniques, and the improved VINAS-P could efficiently verify the target circuit.

On the other hand, we faced some difficulties in dealing with large specification time Petri nets. We need to develop some formal specification language which allows us to easily create and modify large specifications.

5. ACKNOWLEDGMENTS

We would like to thank Professor Takashi Nanya, Dr. Masashi Kuwako, Rafael Morizawa, and Hiroshi Ryu for valuable discussions.

6. REFERENCES

- Belluomini, W. and Myers, C. (1998). Verification of timed systems using POSETs. *LNCS 1427 Computer Aided Verification*, pages 403–415.
- Burch, J. R., Clarke, E. M., Dill, D. L., Hwang, L. J., and McMillan, K. L. (1992). Symbolic model checking: 10^{20} states and beyond. *Academic Press*, 98(2):142–170.
- Dill, D. L. (1988). *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. MIT press.
- Furber, S. B., Day, P., Garside, J. D., Paver, N. C., and Temple, S. (1996). AMULET2e. In Muller-Schloer, C., Geerinckx, F., Stanford-

- Smith, B., and van Riet, R., editors, *Embedded Microprocessor Systems*. Proceedings of EMSYS'96 - OMI Sixth Annual Conference.
- Furber, S. B., Day, P., Garside, J. D., Paver, N. C., Temple, S., and Woods, J. V. (1994). The design and evaluation of an asynchronous microprocessor. In *Proc. International Conf. Computer Design (ICCD)*. IEEE Computer Society Press.
- Godefroid, P. (1990). Using partial orders to improve automatic verification methods. *Proc. of Computer Aided Verification Workshop*.
- Hennessy, J. and Patterson, D. (1996). *Computer Architecture: A Quantitative Approach, Second Edition*. Morgan Kaufmann Publishers.
- Ishikawa, M. (1997). Study on instruction cache system for asynchronous processors. *Bachelor's thesis (in Japanese), Tokyo Institute of Technology*.
- Katz, S. and Peled, D. (1990). Defining conditional independence using collapses. *Semantics for concurrency, BCS-FACS Workshop, M. Kwiatkowska (ed.), Springer*.
- Martin, A. J., Burns, S. M., Lee, T. K., Borkovic, D., and Hazewindus, P. J. (1989). The first asynchronous microprocessor: the test results. *Computer Architecture News*, 17(4):95–110.
- Merlin, P. and Faber, D. J. (1976). Recoverability of communication protocols. *IEEE Trans. on Communication*, COM-24(9).
- Nanya, T., Ueno, Y., Kagotani, H., Kuwako, M., and Takamura, A. (1994). TITAC : Design of a quasi-delay-insensitive microprocessor. *IEEE Design & Test of Computers*, 11(2):50–63.
- Takamura, A., Kuwako, M., Imai, M., Fujii, T., Ozawa, M., Fukasaku, I., Ueno, Y., and Nanya, T. (1997). TITAC-2: An asynchronous 32-bit microprocessor based on scalable-delay-insensitive model. In *Proc. International Conf. Computer Design (ICCD)*, pages 288–294.
- Valmari, A. (1990). A stubborn attack on state explosion. *Proc. of Workshop on Computer Aided Verification*.
- Yoneda, T. (1999). Abstracted instruction cache of TITAC2 — as a benchmark circuit for timed asynchronous circuit verification. *TIT Technical Report*, 99TR-0002.
- Yoneda, T. and Ryu, H. (1999). Timed trace theoretic verification using partial order reduction. *Proc. of Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 108–121.
- Yoneda, T. and Schlingloff, H. (1997). Efficient verification of parallel real-time systems. *Formal Method in System Design*, pages 187–215.