VINAS-P: A Tool for Trace Theoretic Verification of Timed Asynchronous Circuits

Tomohiro Yoneda

Tokyo Institute of Technology
2-12-1 O-okayama Meguro-ku Tokyo 152-8552, Japan
yoneda@cs.titech.ac.jp

1 Introduction

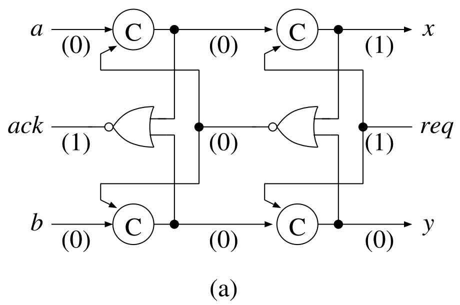
Asynchronous circuit design can probably avoid the occurrence of various problems which arise in designing large synchronous circuits, such as clock skews and high power consumption. On the other hand, the cost of the verification of asynchronous circuits is usually much higher than that of synchronous circuits. This is because every change of wires should be taken into account in order to capture the behavior of asynchronous circuits unlike in the case of synchronous circuits. Furthermore, asynchronous circuit designers have recently preferred to use timed circuits for implementing fast and compact circuits. This trend increases the cost of verification, and at the same time increases the demands for formal verification tools. VINAS-P is our newest formal verification tool for timed asynchronous circuits using the techniques proposed in [1]. The main idea in these techniques is the partial order reduction based on the timed version [2] of the Stubborn set method [3].

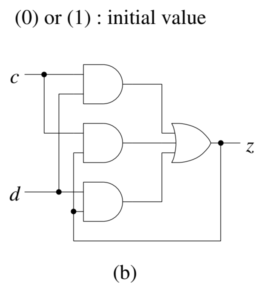
This short paper mainly introduces what VINAS-P can verify and how we use it. We are planning to release VINAS-P on our web site soon.

2 What Can VINAS-P Verify?

In order to formally verify timed circuits, VINAS-P uses the timed version [1] of the trace theoretic verification method [4], and as its internal model, time Petri nets are used. Thus, its implementation and specification are both modeled by time Petri nets, and it produces the result whether the implementation conforms to the specification or not. Since the time Petri nets for an implementation are automatically generated from a Verilog-like description and the VINAS-P gate library, users must usually handle a time Petri net only for the specification. The properties to be verified by the current version of VINAS-P are safety properties. That is, it checks whether a failure state where the circuit produces an output that the specification does not expect (i.e., the specification is not ready for accepting the output) is reachable or not. Any causality relation between input and output wires can be expressed in the specifications. Checking liveness properties (e.g., that some output is actually produced) will be supported in the future.

Here, we will demonstrate verification using VINAS-P with a simple example. Consider the circuit shown in Figure 1(a), which is a two-stage asynchronous FIFO. The gate with a "C" symbol is actually implemented as shown in (b). To describe this circuit, we prepare a Verilog-like description as shown in (c). We do not have to describe primitive gates such as ANDs or ORs, because they





```
module FIFO;
  fifo_cell cell1(a,b,w1,w2,w3,ack);
  fifo_cell cell2(w2,w3,req,x,y,w1);
  initial
  begin
    #0 x=1; ack=1; req=1; w2@C1@cell2=1;
  end
endmodule

module fifo_cell(in1,in2,req,out1,out2,ack);
  celm       C1(out1,req,in1);
  celm       C2(out2,req,in2);
  nor2 [1,2] NOR(ack,out1,out2);
endmodule
```

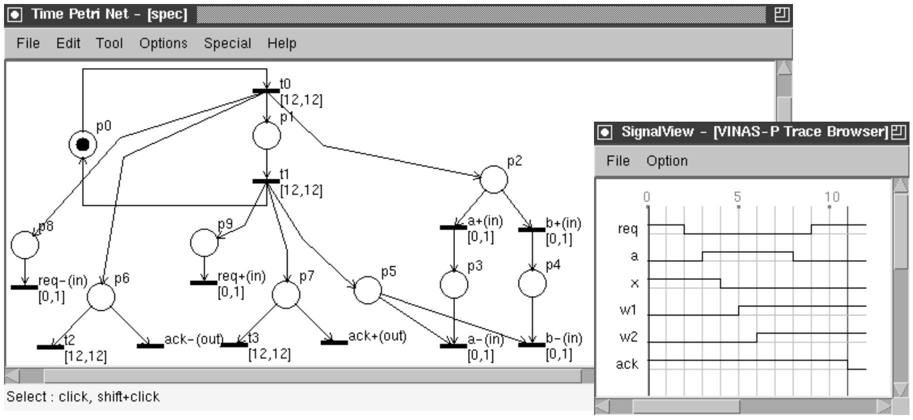
```
module celm(z,c,d);
  and2 [1,2] AND1(w1,c,d);
  and2 [1,2] AND2(w2,c,z);
  and2 [1,2] AND3(w3,d,z);
  or3  [1,2] OR(z,w1,w2,w3);
endmodul
```

(c)

Fig. 1. A circuit to be verified.

are included in the gate library. A module `celm` is for the "C" gate. This gate produces the value v only when v is given to both inputs, otherwise, it holds the previous value. The delay values (e.g., [1,2]) are specified for each primitive gate. VINAS-P assumes the bounded delay model, and these delay values represent the minimal and maximal delays.

Suppose that this circuit is used in the following environment. Either (0,1) or (1,0) and then (0,0) is given to (a, b) periodically, say, every 12 time units. However, due to clock skew, up to one time unit inaccuracy can occur. For *req*, 0 is given when (0,1) or (1,0) is given to (a, b), and 1 for (0,0). What we want to verify is that this FIFO never causes overflow, that is, that *ack* is always activated (i.e., 0 for (0,1) or (1,0), and 1 for (0,0)) before the next data is given to (a, b). In order to describe both the environment and this property, we prepare a time Petri net as shown in Figure 2(a). Some transitions have timing information like $[p, q]$. It means that the transition must have been enabled for p time units or more before its firing, and that it must fire before q time units have passed unless it is disabled. Thus, the transitions labeled $t0$ and $t1$ fire exactly every 12 time units. [0,1] in some transitions models the effect of the clock skew. The transitions which are related to the input wires of the circuit are labeled "(in)", and the transitions related to the output wires are labeled "(out)". The names of these transitions end with "+" or "−". The firings of "+" ("−") transitions correspond to $0 \to 1$ $(1 \to 0)$ signal transitions. The firings of these transitions are synchronized with the changes of the corresponding wires of the circuit. Since the output wires are controlled by the circuit, the specification cannot specify any timing information to those transitions. It is assumed that they have $[0, \infty]$. The transitions without "(in)" or "(out)" are not related to



(a) (b)

Fig. 2. A specification and a failure trace.

any wires, and are called internal transitions. The transitions $t2$ and $t3$ which conflict with the transitions for $ack-$ or $ack+$ are for checking the properties to be verified. These transitions become enabled when the change of a or b is triggered. If the circuit changes the *ack* before the firings of these transitions, then it is the correct behavior of the circuit (no overflow occurs). On the other hand, if the *ack* is not changed for 12 time units, it means that the overflow can be caused by the next data. In this case, $t2$ (or $t3$) fires and the transitions for the *ack* become disabled. Thus, when the circuit eventually produces *ack*, the situation in which the circuit produces an output that the specification does not expect occurs. VINAS-P detects this as a failure. It is straightforward to see that this time Petri net expresses both the behavior of the environment of the circuit and the expected behavior of the circuit itself. All we should do for this verification is to push the "verify" button and see the results. If the circuit does not conform to the specification (actually it does not), the failure trace shown in Figure 2(b) is displayed.

The major advantage of VINAS-P is that verification can be performed without traversing the entire state space of the given system. This is possible by the partial order reduction technique, and it often reduces the verification cost dramatically. For example, an abstracted TITAC2 instruction cache subsystem which contains around 200 gates (modeled by over 1500 places and 1500 transitions) was verified in about 15 minutes, using less than 20 MBytes of memory [5]. No conventional methods can complete this verification.

The most closely related work is probably ATACS developed at the University of Utah. It also uses some kind of partial order reduction [6]. However, since ATACS only uses restricted information relating to the concurrency between events, VINAS-P is expected to be faster in many cases. On the other hand, the expressibility of the ATACS modeling language is superior to that of VINAS-P.

3 User Interface

The verifier core of VINAS-P written in C is almost stable, and the graphical user interface written in Java is being improved. It can select circuit files and specification Petri net files, and launch editors for them. The Petri net editor (see view in Figure 2(a)) recognizes Petri net objects such as transitions, places, and arcs, and is designed so that the creation or modification of Petri nets can be easily done. Information of the Petri net objects created by this editor is also used when browsing failure traces.

When VINAS-P detects a failure state, it shows a trace which leads the system from the initial state to the failure state as shown in Figure 2 (This trace is obtained simply from the recursion stack, and thus there is no timing information in it). The user can select a set of wires to be shown. When this failure trace is displayed, the specification net is also shown, and by moving the vertical cursor on the failure trace, the corresponding marking of the specification net is displayed by an animation. This function of VINAS-P helps users tremendously in debugging circuits.

For early stages of debugging circuits, VINAS-P also provides a function which we call “guided simulation”. It allows users to easily specify input sequences for simulation and to browse the output traces.

4 Future Works

¿From some experimental results [7], we feel that the approach based on compressing and thinning out the visited state information is more effective in reducing the memory usage than symbolic approaches based on decision diagrams especially for the partial order reduction technique of timed systems. We are currently implementing this idea in VINAS-P.

At present, we are also designing a specification description language for VINAS-P in order to easily handle large specifications. Although CCS, CSP, and their extensions can be used for this purpose, we do not think that it is easy for nonexperts to use them. Instead, we have designed a Java-like language, and are now checking its expressibility in comparison with Petri nets.

Acknowledgment

We would like to thank Yoshinori Tomita for helping with the development of the Petri net editor and Trace browser of VINAS-P.

References

[1] T. Yoneda and H. Ryu. Timed trace theoretic verification using partial order reduction. *Proc. of Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 108–121, 1999.

[2] T. Yoneda and H. Schlingloff. Efficient verification of parallel real–time systems. *Formal Method in System Design*, pages 187–215, 1997.

[3] A. Valmari. Stubborn sets for reduced state space generation. *LNCS 483 Advances in Petri Nets*, pages 491–515, 1990.

[4] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. MIT press, 1988.

[5] T. Yoneda. Verification of abstracted instruction cache of titac2. *VLSI: Systems on a chip (Kluwer Academic Publishers)*, pages 373–384, 1999.

[6] W. Belluomini and C. Myers. Verification of timed systems using POSETs. *LNCS 1427 Computer Aided Verification*, pages 403–415, 1998.

[7] K. Oikawa, K. Kitamura, and T. Yoneda. Saving memory for verification of asynchronous circuits. *IEICE Technical Report (in Japanese)*, FTS99(11):37–44, 1999.