Hazard–Freedom Checking in Speed–Independent Systems*

Husnu Yenigun[1], Vladimir Levin[1], Doron Peled[1], and Peter A. Beerel[2]

[1] Bell Laboratories, 600 Mountain Av., Murray Hill, NJ 07974, USA
{husnu,levin,doron}@research.bell-labs.com
[2] University of Southern California, Los Angeles, CA 90089, USA
pabeerel@eiger.usc.edu

Abstract. We describe two approaches to use the model checking tool COSPAN to check the hazard freedom in speed–independent circuits. First, we propose a straight forward approach to implement a speed–independent circuit in S/R. Second, we propose a reduction technique over the first approach by restricting the original system with certain constraints. This reduction is implemented on the top of COSPAN which also applies its own reductions, including symbolic representation (BDD).

1 Introduction

Speed–Independent systems are a special subclass of asynchronous systems in which gates are modeled as instantaneous functional elements followed by arbitrarily long delay components while assuming zero delay in wires. The advantage of this approach is that the design works regardless of the delay of the individual gates thus eliminating the need for any timing assumptions in the circuit. There are several techniques developed to help verify speed–independent systems (e.g., [1,2,3,4]).

The design of speed–independent systems is complicated since one has to make sure that, the unwanted signals, *hazards*, which cause the circuit malfunction, do not appear in the design. In this paper we propose two approaches to use the model checking engine COSPAN [5] to check the hazard freedom of speed–independent circuits. In the first approach, a speed–independent system is completely specified in S/R (the input language of COSPAN), and COSPAN is used to check the states of the system exhaustively to search for a hazard state. This approach, however, suffers from the state explosion problem. Therefore, we propose a reduction method, similar to the partial order reduction [6,7], to force COSPAN to search only a subset of the reachable states of the system. Our technique is based on the static partial order reduction [8].

2 Definitions

A speed–independent system SI is given as a tuple $SI = (G, I, F)$. G is the set of gates in the circuit. $I : G \to 2^G$ is a function giving the interconnection of

* This work was supported in part by SRC Contract no. 98-DJ-486.

L. Pierre and T. Kropf (Eds.): CHARME'99, LNCS 1703, pp. 317–321, 1999.

the circuit. For two gates g_1, g_2, if $g_1 \in I(g_2)$, then the output of the gate g_1 is an input to the gate g_2. We can also have $g \in I(g)$ for a gate g whose output is also an input to itself. And finally, F is a function mapping each gate $g \in G$ to a boolean function, so $F(g)$ is a boolean function of arity $|I(g)|$.

A state s of the system is a function $s : G \rightarrow \{0, 1\}$ where for a gate $g \in G$, $s(g)$ is the current output value of gate g at s. Let S be the set of all such functions and $s_0 \in S$ be the initial state of the system.

Given a gate g, and a state s, $I_s(g)$ is the bit vector $[v_1, v_2, \ldots, v_{|I(g)|}]$ representing the current input vector to g at s. $F(g)(I_s(g))$, or shortly $F_s(g)$, is *the desired value* which the current inputs of gate g tend to derive as the new output of g. A gate g is said to be *enabled at state* s if $s(g) \neq F_s(g)$, i.e., the current value of the gate is different from the desired value of the gate. The set of enabled gates at s is given by $enabled(s) = \{g \in G | s(g) \neq F_s(g)\}$.

The interleaving semantics of speed–independent systems requires that only one of the enabled gates may change its value at a time. The transitions of a speed–independent system are given by the relation $T \subseteq S \times S$, such that, $(s, (s/g)) \in T \Leftrightarrow g \in enabled(s)$, where s/g is the state obtained from s by inverting the current output value of g.

A *run from* s_1 is a finite sequence of states, $r = (s_1, s_2, \ldots, s_k)$ such that for $1 \leq i < k$, $(s_i, s_{i+1}) \in T$. The *full state space* (or reachable state space) $S_F \subseteq S$ of the system is the set of states such that $s \in S_F$ iff there exists a run $r = (s_1, s_2, \ldots, s_k)$ such that $s_1 = s_0$ and $s_k = s$.

A transition $(s, s/g)$ is called a *hazard–transition* if $\exists g' \neq g$ and $g' \in enabled(s)$ and $g' \notin enabled(s')$. A state s is called a *hazard–state* if there exists two gates g and g' such that $g, g' \in enabled(s)$ and $g' \notin enabled(s/g)$. A *hazard–run from* s_1 is a run from s_1, $r = (s_1, s_2, \ldots, s_k)$ such that for all $1 \leq i \leq k-1$, s_i is not a hazard–state, and s_k is a hazard–state. The system SI is said to be *hazardous* if there exists a hazard–run from s_0, the initial state of the system.

3 Reduced Hazard Check

A sequence of gates $c = g_1, g_2, \ldots, g_n$ is called a *gate–cycle* if $\forall i < n$, $j \leq n : g_i \in I(g_{i+1})$, $g_n \in I(g_1)$ and $i \neq j$ implies $g_i \neq g_j$. A gate–cycle is a simple cycle in the interconnection structure of the gates. Let $\overline{c}$ denote the *set* of gates in the sequence c. Let $C = \{c_1, c_2, \ldots, c_m\}$ be the set of all gate–cycles in the given circuit and $G_{sticky} \subseteq G$ be a set of gates such that, $\forall c \in C$, $\exists g \in \overline{c}$ such that $g \in G_{sticky}$.

Given two gates g and g' such that $g \in I(g')$, and a state s, g is called *disabling–input for* g' at s if $g' \notin enabled(s)$ and $\forall s'$, $(s(g) = s'(g)) \wedge (s(g') = s'(g'))$ implies $g' \notin enabled(s')$. Intuitively, if g is disabling–input for g' at s, then g' cannot be enabled as long as g stays at its current value. g is called *enabling–input for* g' at s if $g' \notin enabled(s)$ and $g' \in enabled(s/g)$. Let $fanout(g) = \{g' | g \in I(g')\}$ denote the fanout gates of gate g.

A gate g is called *ample* at s, if (1) $g \in enabled(s)$; (2) $s(g) = 1$ or $g \notin G_{sticky}$; (3) $\forall g' \in fanout(g)$, g is a disabling–input for g' at s; and (4) $\exists g' \in fanout(g)$, g is an enabling–input for g' at s. State s is an *ample state* if there exists an ample gate g at s.

The defining feature of our reduced algorithm over that of the original algorithm is that it explores only one transition associated with an ample gate out of every ample state whereas it explores all transitions from non-ample states. The reduced algorithm is still guaranteed to catch a hazard if the system is not hazard free.

4 Implementation

To implement our algorithms within COSPAN, we developed two different S/R process–type libraries for basic gates. Each gate is represented by a S/R process instantiated from the appropriate S/R process type.

In the first library, a process type has a state variable named *out* which keeps the current output value of the gate and a selection variable named *output*, assigned to the value of the state variable *out*, to inform the current output value to the fanout gates using appropriate instantiation connections. It also has another selection variable, *enabled*, which is set to true whenever the gate is enabled and a selection variable, named *hazard_flag*, which is set to true whenever the current state is a hazard state on this gate. If the process type implements an n input basic gate, then it has $2n+1$ formal parameters. Specifically, it imports the *output* and *enabled* selection variables from each fanin gate g and a last formal parameter from a process called *Asynchrony_Manager*, or shortly *AM*. Since an S/R system is a synchronous system, whereas a speed–independent system is asynchronous, we mimic the asynchrony by using AM. Every gate informs AM whether it is enabled or not (via its *enabled* selection variable). AM lets the enabled gates execute a transition in a mutually exclusive manner.

The second process–type library is used for the reduced case analysis. In addition to all the state and selection variables of the previous case, a process type in the reduced case has two more selection variables for each fanin component it has to inform the corresponding fanin gate if it is a disabler and enabler input for this gate or not. It also has another selection variable called *ample* which is set to true only if the gate is an ample gate at the current state. We also modify AM in the reduced case to import the *ample* selection variables of the gates. If at a state, there exists a gate whose *ample* is true, then AM of the reduced case only allows this gate to execute. If none of the *ample* selection variables are true, then it again lets all the enabled gates execute mutually exclusively.

In both cases, COSPAN is run on the system and checks the property that no gate's hazard_flag is ever set.

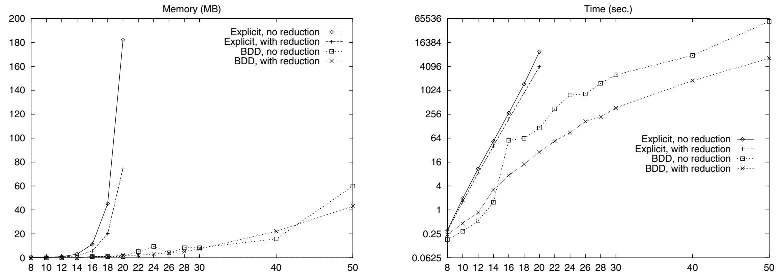

Fig. 1. Memory and time. The x–axis are the number of C–elements in the circuit.

5 Experiments and Discussion

Currently, there is no tool implemented to instantiate a system using the process type libraries. Therefore, we experimented with a hand-generated scalable family of FIFO queue circuits implemented using Muller C-elements and inverting and non–inverting buffers.

Neither the original system nor the reduced system is reported to be hazardous. As the figure illustrated, the reduction ratio of the number of states (i.e. the number of states in the original system divided by the number of states in the reduced system) is exponential though with a smaller degree than the increase in the number of states. As a consequence of this reduction, the time required to analyze the reduced system is always less than that required to analyze the original system and the reduced case always uses less memory, but it is still exponential.

In the symbolic (BDD) case, we do not see any stable correlation between reduced and original cases in terms of memory usage. The reduced analysis, however, is faster after the scale goes above 16 C–elements despite the fact that the partial reduced system being verified is more complicated.

Although somewhat promising, more experiments are needed to better judge the merits of the proposed reduction approach.

References

1. J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan and D. L. Dill, *Symbolic model checking for sequential circuit verification*, IEEE Transactions on CAD, Vol.13, No.4, April 1994, pp.401–424. 317
2. K. L. McMillan, *A technique of state space search based on unfolding*, Formal Methods in System Design, Vol.6, pp.45–65, 1995. 317
3. P. A. Beerel, J. R. Burch and T. H.-Y. Meng, *Checking Combinational Equivalence of Speed-Independent Circuits*, in Formal Methods on System Design, May 1998. 317
4. D. L. Dill, *Trace theory for automatic hierarchical verification of speed–independent circuits*, ACM Distinguished Dissertations, 1989. 317

5. R. P. Kurshan, *Computer–aided verification of coordinating processes*, Princeton University Press, Princeton, New Jersey, 1994. 317
6. D. Peled, *Combining partial order reductions with on the fly model checking*, 6^{th} CAV, June 1994. 317
7. A. Valmari, *A stubborn attack on state space explosion*, 2^{nd} CAV, 1990, pp.25–42. 317
8. R. P. Kurshan, V. Levin, M. Minea, D. Peled and H. Yenigun, *Static Partial Order Reduction*, TACAS, 1998. 317