Journal of Circuits, Systems, and Computers
Vol. 13, No. 5 (2004) 1111–1132
© World Scientific Publishing Company
World Scientific
www.worldscientific.com

MDG-BASED STATE ENUMERATION BY RETIMING AND CIRCUIT TRANSFORMATION

OTMANE AÏT MOHAMED[*,§], XIAOYU SONG[†], EDUARD CERNY[‡],
SOFIENE TAHAR[*,¶] and ZIJIAN ZHOU[‖]

[*] *Department of ECE, Concordia University,*
1455 de Maisonneuve Blvd. W., Montreal,
Quebec, H3G 1M8, Canada
[†] *Department of ECE, Portland State University, USA*
[‡] *D'IRO, Université de Montréal, Canada*
[§] *ait@ece.concordia.ca*
[†] *song@ece.pdx.edu*
[‡] *cerny@iro.umontreal.ca*
[¶] *tahar@ece.concordia.ca*
[‖] *zzhou@ti.com*

Received 12 August 2002
Revised 19 September 2003

Multiway Decision Graphs (MDGs) have recently been proposed as an efficient representation for RTL designs. In this paper, we illustrate the MDG-based formal verification technique on the example of the Island Tunnel Controller. We investigate several techniques on how to deal with the nontermination problem of abstract state exploration, including a novel method based on retiming and circuit transformation. We provide comparative experimental results for the verification of a number of properties for the example using two well-known ROBDD-based verification tools, namely, SMV (Symbolic Model Verifier) and VIS (Verification Interacting with Synthesis), and we show the strength of the MDG approach to handling arbitrary data widths.

Keywords: Multiway decision graphs; binary decision diagrams; state machine verification; retiming; termination; reachability analysis.

1. Introduction

ROBDDs[1] have proven to be a powerful tool for automated hardware verification.[2,3] However, they require a Boolean representation of the circuit and the size of a ROBDD grows, sometimes exponentially, with the number of Boolean variables. Therefore, ROBDD-based verification cannot be directly applied to mixed control-data-path circuits with large data words. There are two ways to alleviate the problem. The first one is to reduce the complexity of hardware designs by means of data

[‡]Currently with Synopsys, Inc, Marlborough, Massachusetts, USA.

abstraction.[4,5] The second one is to represent hardware designs at different levels of abstraction, and verify the designs hierarchically.

Recently, a number of ROBDD extensions, such as BMDs[6] and HDDs,[7] have been developed to represent arithmetic functions more compactly than ROBDDs. There also emerged a number of other methods[8–10] that verify the overall functionality of Register-Transfer Level designs at an abstract level. For instance, they use abstract variables to denote data signals and uninterpreted function symbols to denote data operations. A new class of decision graphs, called *Multiway Decision Graphs* (MDGs),[8] has been proposed that comprises, but is much broader than the class of ROBDDs. Underlying MDGs is a subset of a many-sorted first-order logic with a distinction between concrete and abstract sorts. A concrete sort has a finite enumeration while an abstract sort does not. Hence, a data signal can be represented by a single variable of abstract sort, rather than by a vector of Boolean variables, and a data operation can be viewed as a black box represented by an uninterpreted function symbol. MDGs are thus more compact than ROBDDs for mixed control-data-path designs, especially when there are data transformations and feedbacks from data-path to control.

A basic set of MDG operators and a reachability analysis algorithm based on abstract implicit enumeration are described in Ref. 8. The reachability algorithm verifies whether an invariant holds in all reachable states of an *Abstract State Machine* (ASM).[11] One application of the algorithm is the verification of observational equivalence of synchronous circuits. In Refs. 9 and 12, the authors proposed a validity checking algorithm for processor verification, which is also based on the use of abstract sorts and uninterpreted function symbols. A logic expression representing the correctness statement is generated using symbolic simulation. The algorithm is then used to check its validity. With carefully chosen heuristics for avoiding exponential case splitting, they verified a subset of the DLX RISC pipeline processor[9] and a protocol processor (PP).[12]

Cyrluk and Narendran[10] defined a first-order temporal logic-Ground Temporal Logic (GTL) which also uses uninterpreted function symbols. Using a decidable fragment of GTL, they can automate the verification in the PVS theorem prover. These methods, however, are not applicable to verification problems that require state-space exploration. They cannot represent sets of states and compute fixpoints on sets of states. MDGs, on the other hand, provide a tool for both validity checking and verification based on state-space exploration. Unfortunately, the reachability analysis algorithm of MDGs suffers in many cases from an important problem, namely nontermination when computing the set of reachable states. This could be a severe limitation on the use of MDGs as a verification tool. To illustrate this problem, consider an abstract description of a conventional (nonpipelined) microprocessor where a state variable *pc* of abstract sort represents the program counter, a generic constant zero of the same abstract sort denotes the initial value of *pc*, and an abstract function symbol *inc* describes how the program counter is

incremented by a nonbranch instruction. The MDG representing the set of reachable states of the microprocessor would contain states of the form:

$$(pc, \underbrace{inc(\cdots inc}_{k}(\mathsf{zero})\cdots))$$

for every $k \geq 0$.

Consequently, there is no finite MDG representation of the set of reachable states and the reachability algorithm will not terminate. This typical form of nontermination is due to the fact that the terms can be arbitrarily large and arbitrarily many. This problem could be avoided for certain class of circuit as it is suggested in Ref. 8 where the authors present a method based on the generalization of the state variable that causes divergence, like the variable pc in the example. Rather than starting the reachability analysis with a generic constant zero as the value of pc, a fresh[a] variable is assigned to pc at the beginning of the analysis. As a consequence, the initial set of states represented by pc thus represents any state, hence any incrementation of pc leads the ASM to a state where the new value of pc is an instance of its arbitrary value of the initial state. Another interesting solution to this problem is investigated in Ref. 20. Aït Mohamed *et al.* uses a special kind of terms instead of a fresh variable to start reachability analysis. These terms schematize the infinite MDG generated during reachability analysis so that it is possible to deal explicitly with infinite objects, by finite means.

In this paper, we investigate another kind of circuits for which the generalization approach cannot be directly applied. We analyze the reasons why failures occur in these circuits. In fact, the detailed analysis of the problem leads us to an original solution based on circuit transformations. The idea is to retime the original design and then carry out an appropriate structural transformation. The reachability analysis terminates on the transformed designs. We use the Island Tunnel Controller (ITC)[13] as a case study to illustrate how the retiming technique is applied. This example was originally used to illustrate the notation of a heterogeneous logic system supporting diagrams as logic entities.[13] However, no verification experiments were performed. Although the ITC example is small and does not represent the scale of designs we can verify, it is ideal for illustration of the techniques we propose for the nontermination problem, and it allows to illustrate most current MDG-based verification techniques. The rest of the paper is organized as follows: In Sec. 2, we briefly review Multiway Decision Graphs. The Island Tunnel Controller example is introduced in Sec. 3. We describe the model of the ITC at an abstract level in Sec. 4. In Sec. 5, we study the termination problem of abstract state enumeration, and discuss state generalization. Some heuristics are proposed for state generalization. In Sec. 6, we examine retiming and additional circuit transformations that help to avoid nontermination. In Sec. 7, we present the verification of a number of

[a] A fresh variable is disjoint from all the other variables.

invariants on the ITC example. We also report on experimental results, including a comparison with the results obtained using SMV and VIS. Finally, in Sec. 8 we conclude the paper.

2. MDGs and MDG-Based Verification Approaches

The formal logic underlying MDGs is a many-sorted first-order logic, augmented with the distinction between *abstract sorts* and *concrete sorts*. This is motivated by the natural division of datapath and control circuitry in RTL designs. Concrete sorts have *enumerations* which are sets of *individual constants*, while abstract sorts do not. Variables of concrete sorts are used for representing control signals, and variables of abstract sorts are used for representing datapath signals. Data operations are represented by uninterpreted function symbols. An n-ary function symbol has a type $\alpha_1 \times \cdots \times \alpha_n \rightarrow \alpha_{n+1}$, where $\alpha_1 \cdots \alpha_{n+1}$ are sorts. The distinction between abstract and concrete sorts leads to a distinction between three kinds of function symbols. Let f be a function symbol of type $\alpha_1 \times \cdots \times \alpha_n \rightarrow \alpha_{n+1}$. If α_{n+1} is an abstract sort, then f is an *abstract function symbol*. If all the $\alpha_1 \cdots \alpha_{n+1}$ are concrete, then f is a *concrete* function symbol. If α_{n+1} is concrete while at least one of $\alpha_1 \cdots \alpha_n$ is abstract, then we refer to f as a *cross-operator*; cross-operators are useful for modeling feedback signals from the datapath to the control circuitry.

A *Multiway Decision Graph* (MDG) is a finite, Directed Acyclic Graph (DAG). An internal node of a MDG can be a variable of a concrete sort with its edge labels being the *individual constants* in the enumeration of the sort; or it can be a variable of abstract sort and its edges are labeled by abstract terms of the same sort; or it can be a *cross-term* (whose function symbol is a cross-operator). A MDG may only have one leaf node denoted as *True*, which means all paths in the MDG are true formulas. Thus, MDGs essentially represent relations rather than functions. Just as Bryant's ROBDDs[1] must be *reduced* and *ordered*, MDGs must also be reduced and ordered, and obey a set of other well-formedness conditions given in Ref. 8. We developed a set of MDG algorithms for computing *disjunction*, *relational product* (*conjunction* followed by *existential quantification*[2]) and *pruning-by-subsumption* (PbyS). A detailed description of the algorithms can be found in Ref. 8.

A state machine is described using finite sets of input, state and output variables, which are pairwise disjoint. The behavior of a state machine is defined by its transition/output relations, together with a set of initial states. An *abstract description* of the state machine, called *abstract state machine*,[11] is obtained by letting some data input, state or output variables be of an abstract sort, and the datapath operations be uninterpreted function symbols. Just as ROBDDs are used to represent sets of states, and transition/output relations for finite state machines, MDGs are used to compactly encode sets of (abstract) states and transition/output relations for abstract state machines. We thus lift the implicit enumeration[2,3] technique from the Boolean level to the abstract level, and refer to it as *implicit abstract enumeration*.[8] Starting from the initial set of states, the set of states reached in

one transition is computed by the relational product operation. The frontier-set of states is obtained by *pruning* (removing) the already visited states from the set of newly reached states using pruning-by-subsumption. If the frontier-set of states is empty, then a least fixed point is reached and the reachability analysis procedure terminates. Otherwise, the newly reached states are merged (using disjunction) with the already visited states and the procedure continues the next iteration with the states in the frontier-set as the initial set of states.

One of the variations of the reachability analysis is invariant checking and the verification of observational equivalence of two ASMs verified as an invariant of the product ASM. When an invariant is violated at some stage of the reachability analysis, a counterexample facility gives a sequence of input-state pairs leading from the initial state to the faulty behavior. We also have an equivalence checking procedure for combinational circuits that takes advantage of the canonicity of MDGs.[8] The MDG operators and verification procedures are packaged as MDG tools.[14]

3. The Island Tunnel Controller

The Island Tunnel Controller (ITC) example was originally introduced by Fisler and Johnson.[13] There is a lane tunnel connecting the mainland to a small island, as shown in Fig. 1. At each end of the tunnel, there is a traffic light. There are four sensors for detecting the presence of vehicles: one at tunnel entrance (*ie*), one at tunnel exit on the island side (*ix*), one at tunnel entrance (*me*) and one at tunnel exit on the mainland side (*mx*). It is assumed that all cars are finite in length, that no car gets stuck in the tunnel, that cars do not exit the tunnel before entering the tunnel, that cars do not leave the tunnel entrance without traveling through the tunnel, and that there is sufficient distance between two cars such that the sensors can distinguish the cars.

In Ref. 13, an additional constraint is imposed: "at most sixteen cars may be on the island at any time". The number "sixteen" can be taken as a parameter and it can be any natural number. Thus the constraint may be read: "at most n ($n \geq 0$) cars may be on the island at any time". With ROBDD-based verification, the performance depends on the particular instance of n, as we shall see in Sec. 7.

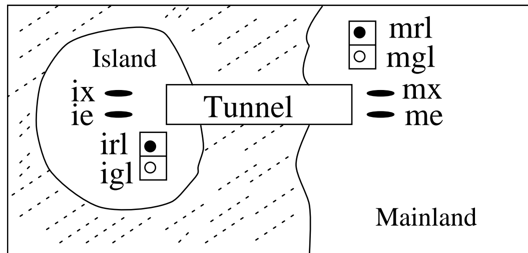

Fig. 1. The Island Tunnel Controller.

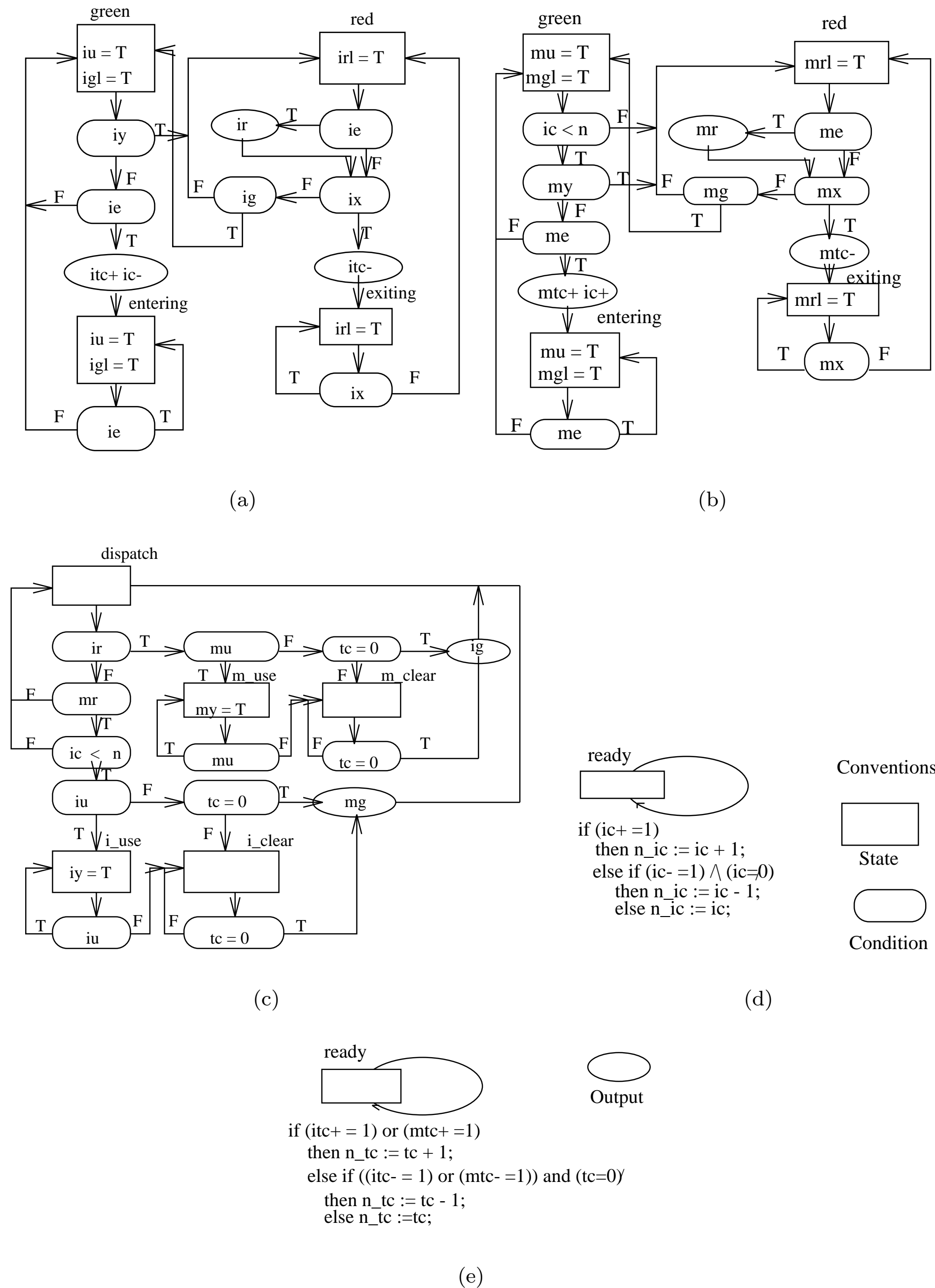

Fig. 2. The state transition diagrams of the Island Tunnel Controller: (a) Island light controller, (b) mainland light controller, (c) tunnel controller, (d) island counter and (e) tunnel counter.

Fisler and Johnson[13] proposed a specification of ITC using three communicating controllers. Their state transition diagrams are shown in Fig. 2 (Figs. 2(a)–2(c) are taken from Ref. 13 except that 16 is replaced by n). The Island Light Controller (ILC) (Fig. 2(a)) has four states: *green*, *entering*, *red* and *exiting*. The outputs *igl* and *irl* control the green and red lights on the island side, respectively; *iu* indicates that the cars from the island side are currently occupying the tunnel, and *ir* indicates that ILC is requesting the tunnel. The input *iy* requests ILC to release control of the tunnel, and *ig* grants control of the tunnel. A similar set of signals is defined for the Mainland Light Controller (MLC). The Tunnel Controller (TC) processes the requests for access issued by the ILC and MLC. The island and the tunnel counters keep track of the numbers of cars currently on the island and in the tunnel, respectively. For the tunnel counter, at each clock cycle, the count *tc* is increased by 1 depending on signals *itc*+ and *mtc*+ or decremented by 1 depending on *itc*− and *mtc*− unless it is already 0. The island counter operates in a similar way, except that the increment and decrement signals are *ic*+ and *ic*−, respectively.

4. MDG-Model of the Island Tunnel Controller

We take as *specification* the ITC state transition diagrams in Fig. 2. In this section, we show how they are modeled as abstract state machines.

Both the island and the tunnel counters have each only one control state, *ready*, hence no control state variable is needed. An abstract state variable *ic* (*tc*)

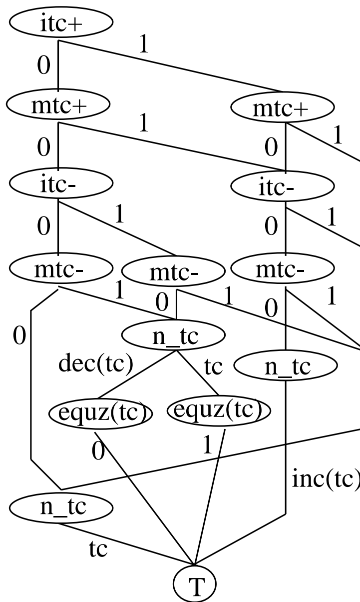

Fig. 3. Transition relation MDG of the tunnel counter.

represents the current count number. At each clock cycle, the count is updated according to the control signals. In this abstract description of a counter, the count ic (tc) is of abstract sort, say *wordn* for n-bit words. The control signals ($ic+$, $ic-$, etc.) are of *bool* sort with the enumeration $\{0, 1\}$. The uninterpreted function *inc* of type [*wordn* → *wordn*] denotes the operation of increment by 1, and *dec* of the same type denotes decrement by 1. The cross-function $equz(tc)$ of type [*wordn* → *bool*] represents the condition "$tc = 0$" and models the feedback from counter to the control circuitry. Figure 3 shows the MDG of the transition relation of the tunnel counter for a specific variable order.

Each of the controllers can have a single control state variable which takes all the possible states as its values. Thus the enumeration of those states constitutes the (concrete) sort of the variable. Let is, ms and ts be the control state variables of the three controllers ILC, MLC and TC, respectively. We assign the variables is and ms (and also their next state variables n_is and n_ms) the sort mi_sort having the enumeration {*green*, *red*, *entering*, *exiting*}. Similarly, we let variables ts and n_ts be of concrete sort ts_sort which has the enumeration {$dispatch$, i_use, m_use, i_clear, m_clear}. All other control signals (ie, ix, me, mx, etc.) are of sort *bool*. The condition "$ic < n$" is represented by the cross-function $lessN(ic)$ of type [*wordn* → *bool*].

An *implementation* is a netlist of components connected by signals. It is described using predefined component definitions in MDG–HDL which is our description language. The specification is also modeled as an ASM represented by MDGs, compiled from its MDG-HDL description (which is our description language).

5. Termination of Abstract State Enumeration

5.1. *Review*

An abstract state machine may have an infinite number of states due to the abstract variables and the uninterpreted nature of function symbols. Thus the least fixed point may not be reached in state enumeration. In Ref. 8, the nontermination problem is explained and a solution is given to a class of problems known as *processor-like* circuits by generalizing the initial state (i.e., by replacing abstract constants with abstract variables as initial values of some registers). Informally, a processor-like circuit usually starts from a *ready* state, performs data operations in one or more cycles and then returns to the *ready* state. We first briefly review the initial state generalization method on the tunnel counter (Fig. 2(e)) which is a processor-like circuit.

Let the initial state of the tunnel counter be a generic constant zero of the abstract sort *wordn*. Figure 4(a) shows the MDGs N_0, S_1 and N_1 representing the set of initial states, the set of states reached in one transition, and the frontier set of states, respectively. There are three paths in S_1, i.e., there are three abstract states. The state $tc = \mathsf{zero}$ is subsumed by N_0 and is thus pruned. The state $(tc = dec(\mathsf{zero})) \wedge (equz(\mathsf{zero}) = 0)$ is, in fact, an unreachable state. If we rewrite

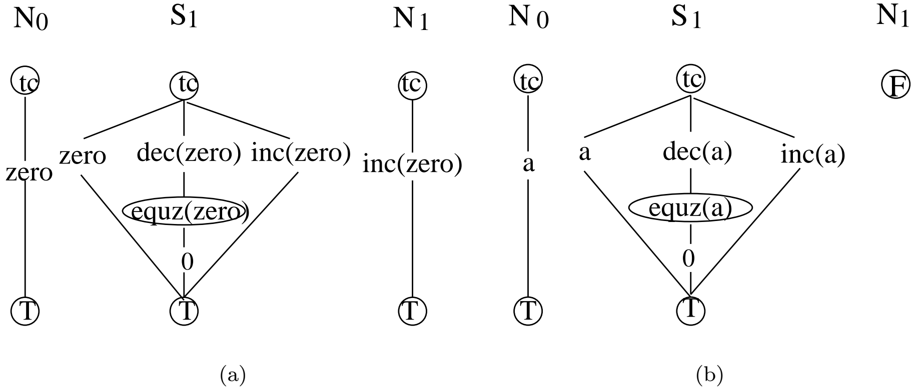

Fig. 4. Nontermination problem and initial state generalization method for the tunnel counter.

$equz(\mathsf{zero})$ to an individual constant 1 using the rewrite rule $equz(\mathsf{zero}) \rightarrow 1$, it yields a contradiction $(1 = 0)$ and the path can be eliminated. The frontier-set of states N_1 thus contains the only state $tc = inc(\mathsf{zero})$. If we continue the reachability analysis with N_1, at the kth iteration we will have the state $tc = inc^k(\mathsf{zero})$ (where $inc^k(\mathsf{zero})$ is a short hand for $inc(\cdots(inc(\mathsf{zero}))\cdots)$). This illustrates the nontermination problem due to the fact that the terms that label the edges can be arbitrarily large and hence arbitrarily many.

The nontermination problem can be avoided by *generalizing* the initial value zero of the state variable tc to a *fresh*[b] *abstract* variable a. Then, as shown in Fig. 4(b), the states $tc = inc(a)$ and $(tc = dec(a)) \wedge (equz(a) = 0)$ in S_1 become instances of $tc = a$, under the substitutions $inc(a)/a$ or $dec(a)/a$, respectively. They thus can be pruned and the state exploration procedure terminates right away (the MDG $N_1 = \mathit{False}$ represents an empty set). Note that the method may have false negatives since the reachable state space is enlarged by the state generalization. For instance, the state $(tc = dec(a)) \wedge (equz(a) = 0)$ becomes a reachable state and it covers the previously unreachable state $(tc = dec(\mathsf{zero})) \wedge (equz(\mathsf{zero}) = 0)$.

5.2. *Delayed state generalization*

As shown in Fig. 2, the complete ITC specification is composed of five communicating state machines. Among them, the tunnel and the island counters are typical processor-like circuits. However, the composed ASM is no more a processor-like circuit, and the initial state generalization technique is not applicable directly. In the following, we present a new state generalization technique which solves the nontermination problem for a larger class of circuits. To simplify the presentation, we consider only a specification consisting of the ASMs in Figs. 2(a)–2(c) and 2(e).

[b]A *fresh* variable is disjoint from all other variables.

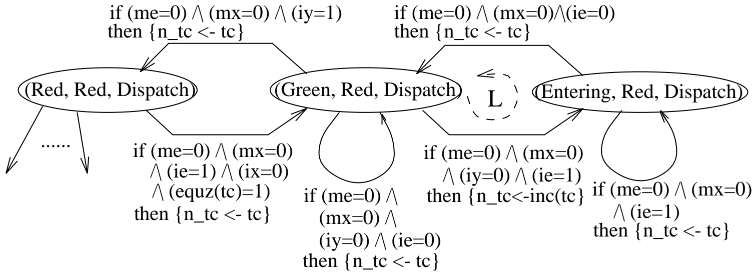

Fig. 5. A fraction of the state transition diagram for the composed machine.

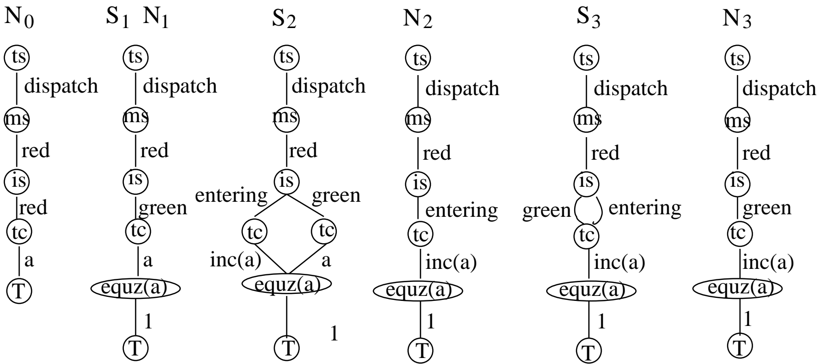

Fig. 6. State enumeration using initial state generalization technique.

Initially, we assume there are no cars in the tunnel. Therefore, the tunnel counter is reset to zero. To be safe, the lights on both the island and the mainland sides are reset to *red*, and the tunnel controller is in the state *dispatch*, ready to take requests. The set of initial states thus includes only one state $(is = red) \wedge (ms = red) \wedge (ts = dispatch) \wedge (tc = \mathsf{zero})$. In Fig. 5, we show a small fraction of the state transition diagram of the composed machine related to the transitions of the ILC between the states *red* and *green*, and the cycles between *green* and *entering*. MLC and TC keep their initial state values.

To illustrate the failure of the initial state generalization method on this example, we again try the above method by setting the initial state to $(is = red) \wedge (ms = red) \wedge (ts = dispatch) \wedge (tc = a)$ where a is an abstract variable of sort *wordn*. Figure 6 shows the MDGs representing the set of newly reached states $(S_i, i = 1, 2, \ldots)$ and the frontier sets $(N_i, i = 0, 1, 2, \ldots)$ in three iterations following the transitions depicted in Fig. 5.

Starting from N_0, after three transitions, the set S_3 of newly reached states contains two states: $(is = green) \wedge (ms = red) \wedge (ts = dispatch) \wedge (tc = inc(a)) \wedge (equz(a) = 1)$ and $(is = entering) \wedge (ms = red) \wedge (ts = dispatch) \wedge (tc = inc(a)) \wedge (equz(a) = 1)$, where the latter is subsumed by N_2. However, the former cannot be

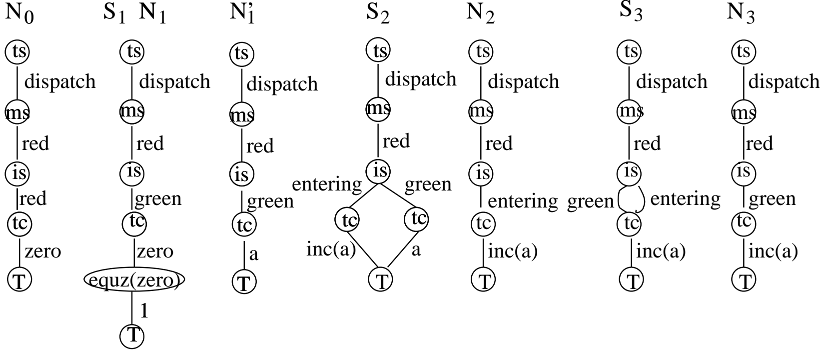

Fig. 7. State enumeration using the extended state generalization technique.

subsumed by N_1 since $inc(a)/a$ is not a valid substitution in this case. If we continue the reachability analysis with N_3, the value of tc will become $inc(\cdots inc(a)\cdots)$ with an unbounded number of *inc*.

A closer examination of this nontermination problem shows that the iterations from N_1 to N_3 represents the loop L (Fig. 5) with a data operation *inc* on the abstract state variable *tc*. This loop, in fact, resembles the ASM of a processor-like circuit. The state $(is = green) \wedge (ms = red) \wedge (ts = dispatch) \wedge (tc = a) \wedge (equz(a) = 1)$ in N_1 can thus be considered as the initial (entry) state of loop L. However, N_1 is not a generalized state because of the assumption $equz(a) = 1$, which is the reason for nontermination.

The above analysis leads to a characterization of a *processor-like loop*. Such a processor-like loop starts from a control state and returns to this control state after one or more transitions, with data registers updated according to the data operations. Note that a processor-like circuit is represented by an ASM having one and only one processor-like loop. This suggests that it is the entry state of a processor-like loop that should be generalized rather than the initial state of the state machine. In the above example, we have to generalize the state $(is = green) \wedge (ms = red) \wedge (ts = dispatch) \wedge (tc = a) \wedge (equz(a) = 1)$ in N_1 instead of the states in N_0. Figure 7 shows the state enumeration procedure using the extended method. Starting from N_0, S_1 is reached in one transition: $\{(is = green) \wedge (ms = red) \wedge (ts = dispatch) \wedge (tc = \mathsf{zero}) \wedge (equz(\mathsf{zero}) = 1)\}$. This can be simplified to $\{(is = green) \wedge (ms = red) \wedge (ts = dispatch) \wedge (tc = \mathsf{zero})\}$ using the rewrite rule $equz(\mathsf{zero}) \rightarrow 1$. As this state is the entry state of the processor-like loop L, we generalize the constant value of *tc* to a *fresh abstract* variable a and remove the constraint $equz(a) = 1$. Thus the frontier set of states becomes $N_1' = \{(is = green) \wedge (ms = red) \wedge (ts = dispatch) \wedge (tc = a)\}$. After two transitions, the frontier-set N_3 becomes $\{(is = green) \wedge (ms = red) \wedge (ts = dispatch) \wedge (tc = inc(a))\}$, where the only state in this set is subsumed by N_1' under substitution $inc(a)/a$, terminating the reachability analysis.

It remains to determine now, when and on which state variables the generalization is to be performed, i.e., how to identify processor-like loops and how to perform the generalization operation. For some circuits, e.g., simple microprocessors, it is possible to identify all processor-like loops by inspection and to perform state generalization manually on the entry states of the loops. However, in general, to find the entry states of all processor-like loops could be very difficult. In the next subsection, we propose a simple heuristic method, while in Sec. 6 we provide a general method based on circuit transformations.

5.3. *Heuristic state generalization*

We developed a heuristic method for state generalization based on the following observation: reachability analysis terminates if we generalize any state within a processor-like loop. Once we generalize a state in a loop, it covers all the abstract states having the same control state values, thus guaranteeing termination.

One method is to perform generalization on every abstract state variable at each clock cycle, i.e., to replace every term that labels an edge issuing from an abstract node with a fresh abstract variable. However, the reachable state space is unnecessarily enlarged since states that are not involved processor-like loops, are also generalized.

As a trade-off, we propose a heuristic solution to this problem: After a certain number of state transitions (specified by the user), if the MDG size of the frontier-set keeps increasing, the value of each state variable in the MDG is generalized. With this heuristics, the state to be generalized is more likely to be within a processor-like loop.

Termination of the abstract state enumeration can be obtained at the cost of false negatives introduced by the state generalization. If the reachability analysis succeeds, we know that the invariant holds even for the enlarged set of reachable states, but if it does not, then we have to examine, e.g., by simulation, whether the counterexample thus produced corresponds to an actual design error. Usually, finding new heuristics is not trivial and requires a lot of expertise. This observation leads us to think of a systematic technique that could help in avoiding nontermination. In the next section, we propose a method based on structural transformation and retiming of the nonterminating ASM. We discuss the method on a simple example before applying it to the Island Tunnel Controller example.

6. Retiming and Transformation for Nontermination Problem

Originally, retiming algorithms address the problem of minimizing the cycle-time or the area of synchronous circuits by changing the position of the registers.[15] In the first case, retiming aims at placing the registers in appropriate positions, so that the *critical paths*[c] they embrace are as short as possible. In the second

[c]i.e., the longest path between a pair of registers.

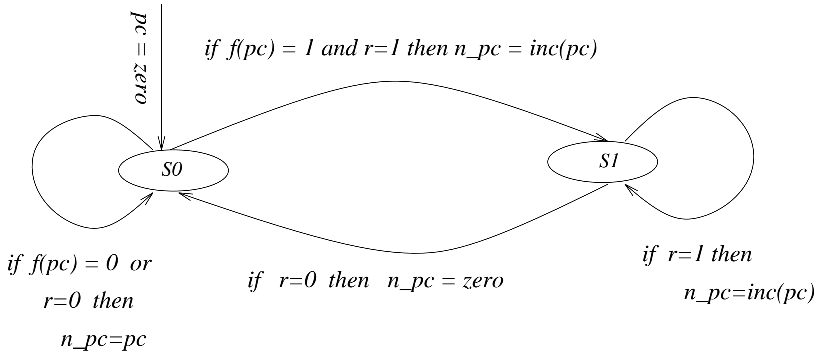

Fig. 8. A simple processor-like loop ASM M.

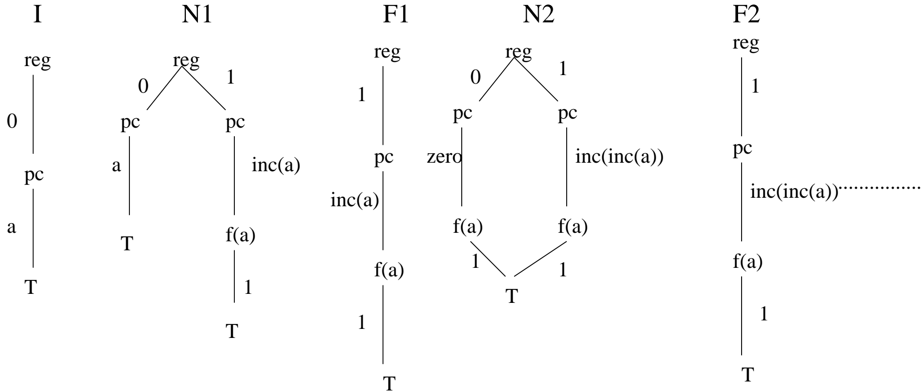

Fig. 9. First MDGs generated for the ASM.

case, retiming corresponds to minimizing the overall number of registers. A new application for retiming is investigated here. More precisely, we use rules of forward retiming to place the registers in appropriate positions such that the reachability analysis terminates after interpretation of the cross-operator on the initial (or reset) state.

We use a simple processor-like loop circuit to explain our method. This example is inspired by the ITC model and is shown in Fig. 8. It represents a state machine with two states, s_0 and s_1. A single register, *reg*, is used to encode the two states.

The reachability analysis for this machine does not terminate even if we generalize the state variable *pc*. Analysis of the first MDG generated in this example indicates that the cross-operator f is the cause of the nontermination. The MDGs in Fig. 9 are generated after two transitions of the ASM M. It shows the MDGs I, N_k and $F_k (k = 1, 2)$ representing the set of initial states, the set of states reached in two transitions, and the frontier set of states, respectively.

The initial state represented by the MDG I consists of $reg = 0$ and $pc = a$, where a is a fresh variable. From this initial state, the ASM can loop on the first state, where reg keeps the value 0 and pc the value a, under the condition $f(a) = 0$; or it can reach the state s_1, where reg takes the value 1 and pc the value $inc(a)$, under the condition $f(a) = 1$. This is represented in Fig. 9 by the MDG $N1$ which contains two paths. The frontier set (MDG $F1$) is computed by removing the path on the left-hand side since it is subsumed by the path from the initial state (MDG I). The MDG $N2$ represents the reachable states from the frontier set (MDG $F1$) in one step. If $r = 0$, then the ASM goes back to the initial state with the value zero loaded in the state variable pc. If $r = 1$, then the ASM stays in S_1 with $reg = 1$ and pc contaning $inc(inc(a))$. The path on the left-hand side of $N2$ is subsumed by the single path of the MDG I because this latter is more general than the former which is thus removed. For the path on the right-hand side of $N2$, $pc = inc(inc(a))$ is an instance of the state $pc = inc(a)$ in $N2$, but the presence of the same guard $f(a)$ in the paths causes failure of the termination, since there is no appropriate substitution to match the state $reg = 1 \wedge pc = inc(inc(a)) \wedge f(a) = 1$ (path on the right-hand side of $N2$) with the state $reg = 1 \wedge pc = inc(a) \wedge f(a) = 1$ (path on the right-hand side of $N1$). Let us point out that this guard is generated from the initial state, where pc would have the initial value zero. Hence the argument of f would have been zero.

Suppose we know that under a specific interpretation in the use context of the circuit, the value of the $f(\mathsf{zero})$ is 1. It would be possible to use this information to eliminate the guard, $f(\mathsf{zero}) = 1$, by using the rule $f(\mathsf{zero}) \rightarrow 1$. Unfortunately, this rule does not apply when we generalize to a as shown in the example. The basic idea to solve this kind of nontermination is to use the partial interpretation of the cross-operator and to delay the generalization until after the interpretation. To recover this information lost by generalization, we could save it in a new state variable. This state variable (register) must be found in the ASM structure by redistributing the existing registers so as not to change the original behavior. For this purpose, we use the rules of forward retiming. A new register will thus appear at the output of the cross-operator f. Forward retiming always guarantees to find the initial values for all registers. In general, we need additional circuit transformation, to maintain the interpreted value of the cross-term as long as it remains valid. Since retiming is usually applied to a structural description of the circuit, it is necessary to extract the circuit from the ASM description. This extraction may lead to a complex circuit where retiming may be difficult. To limit the retiming to just the necessary portion of the ASM, we decompose the original machine, say M, into two inter-dependent sub-machines M_1 and M_2. M_1 represents the control-part and contains only concrete state variables while M_2 represents the data-part that depends on the state of M_1 and contains abstract state variables. M_1 is thus a copy of M without abstract state variables and M_2 is reduced to a single control state. The result of this decomposition as applied to the ASM of Fig. 8 is shown in Figs. 10 and 11. M_1 communicates its state information to M_2 through the output

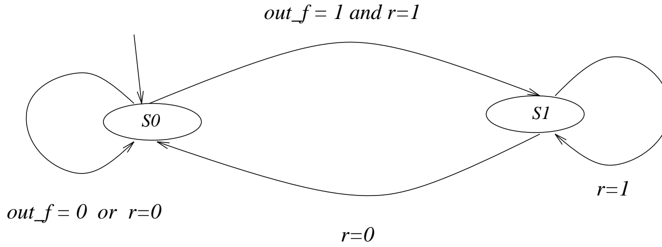

Fig. 10. The control-part (M_1) of the ASM M.

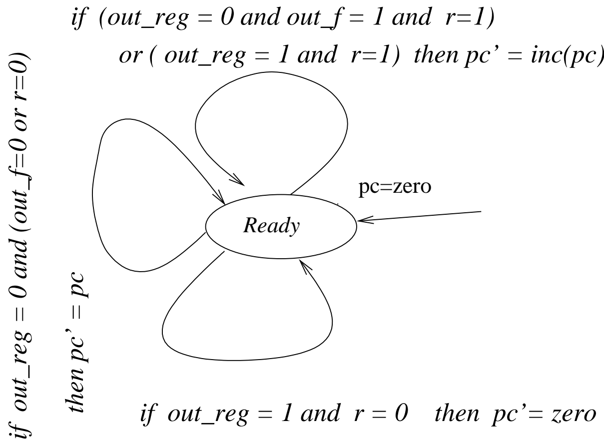

Fig. 11. The data-part (M_2) of the ASM M.

signal *out_reg*, while M_2 communicates the condition on the *pc* value through the output of the cross-term f, *out_f*. Note that the two machines share the primary inputs. By this decomposition, we have isolated the nontermination problem in M_2 that can be retimed as needed, i.e., to obtain a register at the output of the cross-term f.

In Fig. 12, we show the structural description of M_2 and its connection to M_1. The inputs of M_1 are r and $f(pc)$, and its ouput is *reg* which gives the information about the current state of M_1 to M_2.

The structural description of M_2 includes a data register *pc*, an 8 to 1 multiplexer, and two functional blocks represented by the uninterpreted function symbol *inc* and f. *inc* takes *pc* as its input and produces the abstract value *inc*(*pc*). f is a cross-term that takes as its abstract input *pc* and produces a concrete output *out_f* of sort bool. The transition relation of M_2 is defined as follows[d]:

[d]For simplicity, the conditional equation: if a then b else c is written: $a \rightarrow b|c$.

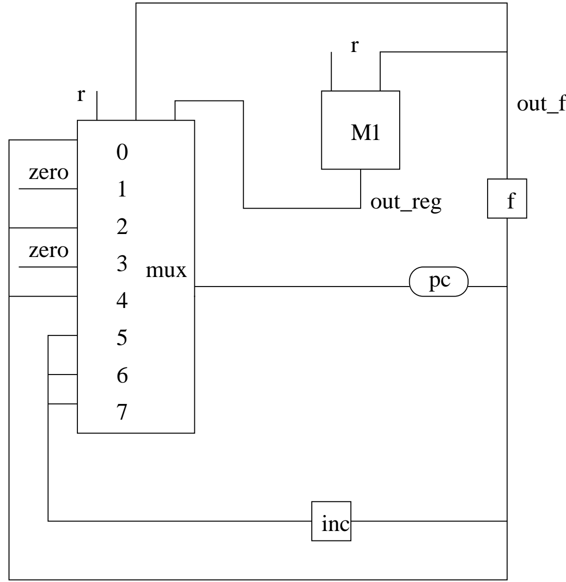

Fig. 12. Structural description of M_2 and its connections to M_1.

$$\begin{aligned}&(out_reg = 0 \wedge out_f = 1 \wedge r = 1)\\&\qquad\qquad \vee(out_reg = 1 \wedge r = 1) \rightarrow pc' = inc(pc)\\&|out_reg = 1 \wedge r = 0 \rightarrow pc' = zero\\&|pc' = pc\end{aligned}$$

In order to obtain a register at the output of f, we retime M_2 by moving the register pc forward to the input of inc and the output of f. The result of the retiming is shown in Fig. 13.

At the output of f, the register pc is replaced by a register pc_f of sort bool, and at the input of inc it is replaced by a register pc_inc. Since the initial value of the register pc was the generic constant zero, the equivalent initial state for the retimed circuit is obtained by letting the appropriate initial values for the two registers pc_inc and f_pc. These values are obtained by propagating the initial state to the new register positions. It follows that the initial value of inc_pc is zero and the initial value of pc_f is $f(\mathsf{zero})$, which is equal to 1.[e]

This partial interpretation of f must be retained until the control machine M_1 uses the condition $f(\mathsf{zero}) = 1$. Furthermore, when M_1 and M_2 return back to their initial states, the initial value (i.e., 1) of the register pc_f must be reloaded. The logic which controls the register pc_f can be implemented by a multiplexer, having

[e]Recall that to avoid nontermination of the reachability analysis, we must use the partial interpretation, $f(\mathsf{zero}) = 1$.

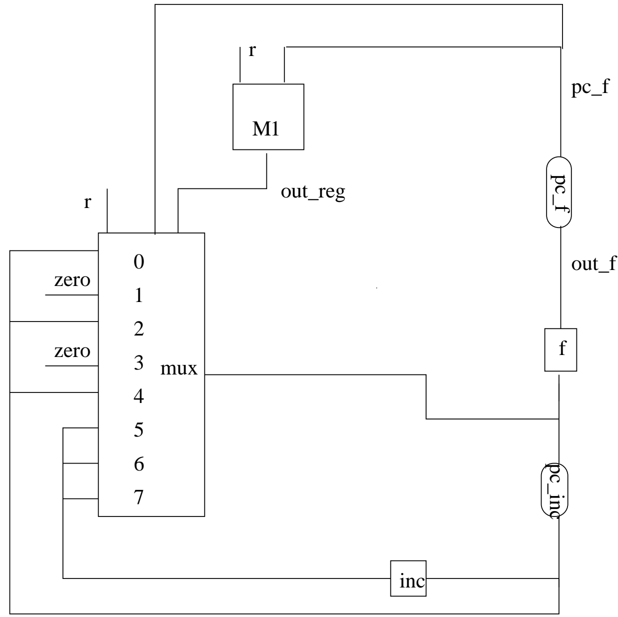

Fig. 13. Structural description of M_2 after retiming and its connections to M_1.

as control signals the primary input r, the register reg which provides information about the current state of the M_1 and the pc_f itself. In order to implement this control for pc_f, we use the equation related to pc_f from Fig. 13, where $pc_f = f(pc_inc)$. The next state for the register pc_f is given by $pc_f' = f(pc_inc')$.

Replacing pc_inc' by its value, which is the same as pc', we get:

$$\begin{aligned}
&(out_reg = 0 \wedge pc_f = 1 \wedge r = 1) \\
&\vee(out_reg = 1 \wedge r = 1) \rightarrow pc'_f = f(inc(pc_inc)) \\
&|out_reg = 1 \wedge r = 0 \rightarrow pc'_f = f(\mathsf{zero}) \\
&|pc'_f = f(pc_inc)
\end{aligned}$$

Note that in the third case, the register pc_inc keeps its previous value and thus pc_f does too. In the second case pc_inc loads the initial value zero and pc_f loads $f(\mathsf{zero})$, and in the first case pc_f contains a new value depending on the result of $f(inc(pc_inc))$. This case analysis on pc_inc' can be implemented by an 8 to 1 multiplexer as shown in Fig. 14.

Reachability analysis applied for the modified circuit terminates, as shown by the sequences of MDGs presented in Fig. 15. The initial state is $reg = 0$, $pc_f = 1$, and the value of pc_inc is generalized to a variable a. The initial value 1 of pc_f represents the partial interpretation of $f(\mathsf{zero})$ (Fig. 15, MDG I). From this initial

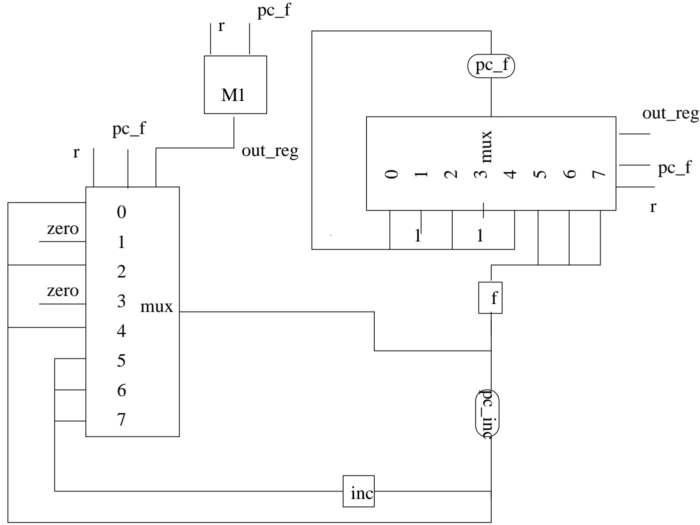

Fig. 14. Structural description of M_2 after retiming and circuit transformation and its connections to M_1.

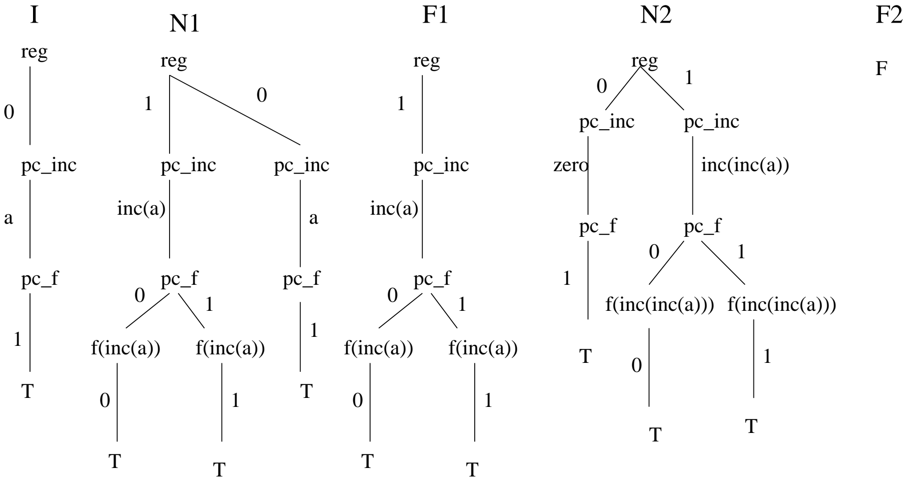

Fig. 15. MDGs generated for the retimed ASM.

state, the ASM can stay there if $r = 0$ or it can reach the state where reg takes the value 1 and pc_inc takes the value $inc(a)$. pc_f takes the value $f(inc(a))$ which is represented by the MDG $N1$ with two paths on which pc_f is either 0 or 1 depending on the value of $f(inc(a))$. The path on the right-hand side in $N1$ is subsumed by I, but the path on the left-hand side represents a new state. $F1$

represents the frontier set obtained by removing the path on right-hand side. The reachable states from the frontier set are represented by $N2$. If $r = 1$, the machine stays in this state and increments the counter such that $reg = 1$, $pc = inc(inc(a))$, and $R = f(inc(inc(a)))$. If $r = 0$, the transition leads back to the initial state by loading the value zero to pc_inc and the value 1 to pc_f. The path on left-hand side of $N2$ is subsumed by the single path of I, by letting a to zero, and the two paths on the right-hand side of $N2$ are subsumed by the paths of $N1$ by substituting a in $N1$ by $inc(a)$. Thus all the paths of N_2 are removed and the frontier set $F2$ is empty, thus terminating the reachability analysis.

We applied this technique on the ITC example. The abstract state enumeration successfully terminates during the reachability analysis. Experimental results for property checking on the ITC are discussed in the next section.

7. Checking Invariants on the Island Tunnel Controller

In this section we discuss the various verification experiments that we performed on the ITC example. All experiments (including those using SMV and VIS) were carried out on a Sun SPARCstation 10. In the subsequent tables, column *Time* is the CPU time in seconds used for compiling the circuit descriptions and for the invariant checking, including the counterexample generation if any. Column *Mem* is the memory allocated in MB. Column *#Nodes* is the total number of MDG (or ROBDD) nodes generated. Property checking is useful for verifying that a specification satisfies certain requirements. We list below three simple properties (invariants) that we verified.[f] We also provide the corresponding CTL formulas used for invariant checking by the tools SMV (V2.4.4)[16] and VIS.[17]

P1: Cars never travel both in directions in the tunnel at the same time.
AG $(!((igl = 1)\ \&\ (mgl = 1)))$.

P2: The tunnel counter is never signaled to increment simultaneously by ILC and MLC.
AG $(!((itc{+} = 1)\ \&\ (mtc{+} = 1)))$.

P3: The island counter is never signaled to increment and decrement simultaneously.
AG $(!((ic{-} = 1)\ \&\ (ic{+} = 1)))$.

For the purpose of comparison, we first show the experimental results for the verification of the above example invariants (P1, P2 and P3) using FSM-based methods. For MDG tools, the counts tc and ic are now assigned a concrete sort according to the counter width which is determined by the instantiation of the constraint, i.e., the maximum number of cars that are allowed on the island.

[f] Fisler and Johnson[13] proposed a set of properties that the ITC design should satisfy. Currently, we consider only the variation of invariants.

Table 1. Invariant checking of ITC specifications ("—" means that the verification did not terminate in certain amount of time, and "*" means that the verification was not possible).

Counter width	SMV Time (s)	SMV Mem (MB)	SMV # Nodes	VIS Time (s)	VIS Mem (MB)	VIS # Nodes	MDG Time (s)	MDG Mem (MB)	MDG # Nodes
4 bits	1.2	1.2	10043	15.4	0.5	6492	430	8	19670
5 bits	4.1	1.2	10463	18.9	0.5	3887	810	10	27668
6 bits	16.7	1.2	11240	44.5	0.6	8902	1719	15	41751
7 bits	79.7	1.2	15047	429.9	1.2	33447	5486	26	69911
8 bits	360	1.6	29474	1686	2.4	43428	—	—	—
9 bits	1564	2.1	59292	7584	5.1	128426	—	—	—
10 bits	6263	3.2	117890	31255	9.9	327090	—	—	—
11 bits	—	—	—	—	—	—	—	—	—
n bits	*	*	*	*	*	*	55	2.7	4329

Table 1 shows the results for checking the conjunction of P1, P2 and P3 for various values of n. The MDG tools can also verify the parameterized implementation having n bits, which is not the case for SMV and VIS. For the SMV columns, the *Time* is the user time, while for VIS and MDG columns, it is the elapsed time including loading the Verilog or MDG-HDL description file, compilation and invariant checking. For SMV and VIS, we used the node ordering generated by the systems, and used manual ordering in MDG since no heuristic ordering algorithm is available yet. Many different factors affect the experimental results shown in the table. The three tools use different integer encoding, different variable ordering, and different partitioning of the transition relation. Notwithstanding these differences, the table clearly shows the following: (i) Time increases exponentially with the counter width for concrete representations of the problem, and (ii) the MDG figures are substantially greater than the others for concrete representations. This is because the MDG data structure and its algorithms are far more complicated than those of ROBDDs. The last row in Table 1 gives the results when we model the design as an ASM instead of an FSM. To avoid the nontermination problem, we use the heuristic state generalization technique and the method based on retiming and the circuit transformation on the complete ITC specification composed of the five ASMs of Fig. 2. In these cases, the verification is performed efficiently using MDGs in time independent of the data-path width.

It may be argued that the data abstraction method[4,5] is sufficient to imply the correctness of this ITC example, i.e., we reduce n to a small number encoded by a few bits, e.g., 2 bits (4), 4 bits (16), etc. Yet in general, the equivalence of the reduced circuit against the original one is not verified mechanically. Also, it is not always obvious how to construct an appropriate data abstract function, or such data abstraction may not even be possible. One such example is the 4×4 Fairisle ATM

Table 2. Verification of P1, P2 and P3 for the transformed model.

Properties	Time	Mem	# Node
P1, P2, P3	11.59	7.9	7287

switch fabric recently verified using MDGs,[18,19] where the datapath contains mixed data and control information. In general, if the control information needs n bits, then it is impossible to reduce the word width to less than n. Hence, in this case the ROBDD-based data-path reduction technique is not applicable. On the other hand, using the MDG-based approach, we naturally allow the abstract representation of data-path while the control information is extracted using cross-functions.

In Table 2, we show the result of the verification of the properties P1, P2 and P3 for the retimed model. In this case, the number of MDGs nodes is higher in the retimed specification, because additional state variables were added by retiming. This affects memory usage, however, the CPU time is five times shorter than with the generalization heuristic as the reachability analysis terminates much faster without re-exploring the same transition.

8. Conclusions

In this paper, we demonstrated the feasibility of the MDG-based hardware verification at the RT level on a non trivial example — the Island Tunnel Controller. We investigated in details the nontermination problem of abstract state enumeration and presented a novel method based on circuit retiming and transformation to overcome this problem. We performed various verification experiments on the example including combinational verification, behavioral equivalence checking, and invariant checking. Furthermore, we gave a comparative evaluation of the results from invariant checking with the ROBDD-based tools SMV and VIS, and showed the strength of MDG approach by handling arbitrary data widths.

Acknowledgments

The work was supported by NSERC Canada Grant No. STRO167079 and workstations on loan from the Canadian Microelectronics Corporation.

References

1. R. E. Bryant, Graph-based algorithms for Boolean function manipulation, *IEEE Trans. Comput.* **35** (1986) 677–691.
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. Computer-Aided Design* **13** (1994) 401–424.
3. O. Coudert, C. Berthet and J. C. Madre, Verification of synchronous sequential machines based on symbolic execution, *Automatic Verification Methods for Finite State*

Systems, ed. J. Sifakis, Lecture Notes in Computer Science, Vol. 407 (Springer-Verlag, 1989).
4. E. M. Clarke, O. Grumberg and D. E. Long, Model checking and abstraction, *Proc. 19th ACM Symp. Principles of Programming Languages*, January 1992.
5. D. E. Long, Model checking, abstraction and compositional verification, PhD thesis, Carnegie Mellon University (1993).
6. R. E. Bryant and Y. Chen, Verification of arithmetic circuits with binary moment diagrams, *32nd ACM/IEEE Design Automation Conf.* (*DAC '95*), San Francisco, California, June 1995.
7. E. Clarke, M. Fujita and X. Zhao, Hybrid decision diagrams, *Proc. IEEE Int. Conf. Computer-Aided Design* (*ICCAD '95*), San Jose, California, USA, November 1995.
8. F. Corella, Z. Zhou, X. Song, M. Langevin and E. Cerny, Multiway decision graphs for automated hardware verification, *Formal Meth. Syst. Design* **10** (1997) 7–46.
9. J. R. Burch and D. L. Dill, Automatic verification of pipelined microprocessor control, *Proc. Work. on Computer-Aided Verification*, ed. D. L. Dill, Lecture Notes in Computer Science, Vol. 818 (Springer-Verlag, 1994).
10. D. Cyrluk and P. Narendran, Ground temporal logic: A logic for hardware verification, *Computer Aided Verification*, ed. D. L. Dill, Lecture Notes in Computer Science, Vol. 818 (Springer-Verlag, 1994).
11. F. Corella, M. Langevin, E. Cerny, Z. Zhou and X. Song, State enumeration with abstract descriptions of state machines, *Proc. IFIP WG 10.5 Advanced Research Working Conf. Correct Hardware Design and Verification Methods* (*CHARME '95*), Frankfurt, Germany, October 1995.
12. R. B. Jones and D. L. Dill, Efficient validity checking for processor verification, *Proc. IEEE Int. Conf. Computer-Aided Design* (*ICCAD '95*), San Jose, California, USA, November 1995.
13. K. Fisler and K. Johnson, Integrating design and verification environment through a logic supporting hardware diagrams, *Proc. IFIP Conf. Hardware Description Languages and Their Applications* (*CHDL '95*), Chiba, Japan, August 1995.
14. K. D. Anon, N. Boulerice, E. Cerny, F. Corella, M. Langevin, X. Song, S. Tahar, Y. Xu and Z. Zhou, MDG tools for the verification of RTL designs, *Proc. 8th Int. Conf. Computer-Aided Verification* (*CAV '96*), New Brunswick, New Jersey, USA, July 1996.
15. C. Leiserson and J. Saxe, Retiming synchronous circuitry, *Algorithmica* **6** (1991) 5–35.
16. K. L. McMillan, *Symbolic Model Checking* (Kluwer Academic Publishers, Boston, Massachusetts, 1993).
17. R. K. Brayton *et al.*, VIS: A system for verification and synthesis, Technical Report UCB/ERL M95, Electronics Research Laboratory, University of California, Berkeley, December 1995.
18. M. Langevin, S. Tahar, Z. Zhou, X. Song and E. Cerny, Behavioral verification of an ATM switch fabric using implicit abstract state enumeration, *Proc. Int. Conf. Computer Design* (*ICCD '96*), Austin, Texas, USA, October 1996.
19. S. Tahar, Z. Zhou, X. Song, E. Cerny and M. Langevin, Formal verification of an ATM switch fabric using multiway decision graphs, *Proc. Great Lakes Symp. VLSI* (*GLS-VLSI '96*), IEEE Computer Society Press, Iowa, USA, March 1996.
20. O. Aït Mohamed, X. Song and E. Cerny, On the nontermination of MDG-based abstract state enumeration, *Proc. IFIP W 10.5 Advanced Research Working Conf. Correct Hardware Design and Verification Methods* (*CHARME '97*), IFIP, Chapmann & Hall, Montréal, October 1997.

Copyright of Journal of Circuits, Systems & Computers is the property of World Scientific Publishing Company and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.