MDG-based Verification by Retiming and Combinational Transformations

O. Aït Mohamed, E. Cerny and X. Song
D'IRO, Université de Montréal
C.P. 6128, Succ. Centre-Ville, Montréal, H3C 3J7, Canada
e-mail: {ait, cerny, song}@iro.umontreal.ca

Abstract

Multiway Decision Graphs (MDGs) have been recently proposed as an efficient verification tool for RTL designs based on an efficient representation mechanisms. In MDG, a data value is represented by a single variable of abstract sort, and a data operation is represented by an uninterpreted function symbol. In this work we investigate the non-termination problem of MDG-based verification. We present a novel approach to dealing with the problem based on retiming and circuit transformations that preserve the behavior of the circuit. We demonstrate the effectiveness of our method on the example of the Island Tunnel Controller (ITC).

Keywords: Formal Verification, Multiway Decision Graphs, Retiming, Circuit Transformations, Non-termination.

1 Introduction

Traditional ROBDD-based methods of automated verification suffer from the drawback that they require a binary representation of the circuit. The major problem with these methods is that the size of the state space may grow very rapidly with the size of the model, which is known as the *state explosion problem.* To alleviate this problem, a new verification approach has been proposed. The approach is based on *abstract descriptions of state machines* (ASM) which are encoded by a new class of decision graphs, called *Multiway Decision Graphs* (MDGs) [3], of which ROBDDs are a special case. With MDGs, a data value is represented by a single variable of abstract type, rather than by Boolean variables, and a data operation is represented by an uninterpreted function symbol. Many MDG-based verification works on abstract state exploration have been performed successfully for combinational/sequential circuits, microprocessors and ATM switch fabric [8, 6, 3].

However, the MDG-based method suffers in many cases from an important problem, namely non-termination when computing the set of reachable states. This can be a severe limitation on the use of MDGs as a verification tool.

In this paper, we propose a novel approach to dealing with this problem. The method is based on retiming [7] and circuit transformations. The transformations lead to an equivalent design on which the verification becomes possible. It has been shown that retiming preserves the functionality of the circuit [7], and the transformations that we perform are in fact one possible implementation of a partial interpretation of the uninterpreted function symbols.

The organization of the paper is as follows: In Section 2 we present MDGs and review some aspects of the non-termination problem. In Section 3, we present, through an example, our verification method for non-terminating specification. Section 4 is devoted to the application of our method to the verification of the ITC specification. Finally, we conclude the paper in Section 5.

2 MDG-based Verification and Non-termination

The logic underlying MDGs is a many-sorted first-order logic, augmented with the distinction between *abstract* and *concrete sorts.* This is motivated by the natural division of data-path and control circuitry in RTL designs. Concrete sorts have finite *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 data-path signals. Data operations are represented by uninterpreted function symbols. An n-ary function symbol has a type $\alpha_1 \times \ldots \times \alpha_n \rightarrow \alpha_{n+1}$, where $\alpha_1 \ldots \alpha_{n+1}$ 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 $\alpha_1 \times \ldots \times \alpha_n \rightarrow \alpha_{n+1}$. If α_{n+1} is an abstract sort then f is an *abstract function symbol.* If $\alpha_1 \ldots \alpha_{n+1}$

0-8186-8409-7/98 $10.00

are concrete, f is a *concrete* function symbol. If α_{n+1} is concrete while at least one of $\alpha_1 \ldots \alpha_n$ 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 an MDG can be a variable of a concrete sort with its edge labels be 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). An MDG may only have one leaf node denoted as T, 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 [3]. A set of MDG algorithms has been developed including algorithms for computing *disjunction*, *relational product* (*Conjunction* followed by *existential quantification* [2]) and *pruning-by-subsumption*. A detailed description of the algorithms can be found in [3]. Pruning is implemented in MDGs by pruning-by-subsumption (PbyS) operation. It takes as input two MDGs G and H and produces the result G' by removing zero or more paths from G thus reducing the graph if needed. PbyS is used for multiple purposes. It provides a form of frontier-set simplfication [4, 2]. It is also used for detecting termination and checking that an invariant is satisfied during reachability analysis [3].

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* (ASM) [3], 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, 4] technique from the Boolean level to the abstract level, and refer to it as *implicit abstract enumeration* [3]. This algorithm is based on abstract state enumeration [3] where sets of states, as well as transition and output relations, are represented using MDGs. 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.

Because of abstract variables and the uninterpreted nature of function symbols, the reachability analysis algorithm may not terminate [3].

For instance, consider an abstract sort $Wordn$ defined only by a generic constant $Zero$ and an abstract function inc. $Zero$ represents the initial value of the sort $Wordn$ and inc is an abstract function which generates elements of sort $Wordn$, having the form: $inc(zero), inc(inc(zero)), \ldots$. A design which uses this sort would have an infinite set of states. For example, consider an abstract description of a conventional (non-pipelined) microprocessor where a state variable pc of sort $Wordn$ represents the program counter, $zero$ denotes the initial value of pc, and inc describes how the program counter is incremented by a non-branch instruction. The state variable pc would contain each value of the abstract sort $Wordn$ starting from any fixed value. If at step k, pc contains v then at step $k+1$ it will contain $inc(v)$. More precisely, at each step a new state is created and reachability algorithm will not terminate. To ensure termination of reachability analysis, we can use a fresh variable, say x, as the value of the variable pc instead of any fixed value of sort $Wordn$ such that x matches any modification of pc. This process is called *generalization* in [3] and it can be applied to all processor-like circuits which possess a cyclic behavior.

However, this kind of generalization may not work for circuits with uninterpreted function symbols used as cross-operators [3] as we will see in the next Section.

3 Retiming and Transformation for Non-Termination

Originally, retiming algorithms address the problem of minimizing the cycle-time or the area of synchronous circuits by changing the position of registers [7]. In the first case retiming aims at placing registers in appropriate positions, so that the *critical paths*[1] they embrace are as short as possible. In the second

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

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.

In this section we review, by an example, the main reasons of non-termination of the abstract state enumeration in MDGs and present a solution to overcome it. Consider an Abstract State Machine, say M, shown in Figure 1. It consists of two states, s_0 and s_1. A single register, *reg*, is used to encode the two states. The

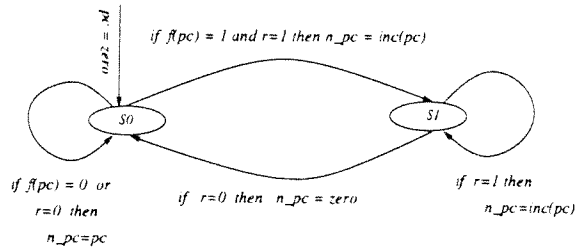

Figure 1: A simple processor-like loop ASM (M)

reachability analysis for this machine does not terminate even if we generalize the state variable *pc*. An analysis of the first MDG generated for this example shows that the cross-operator, f, is the reason for this non-termination. The MDGs in Figure 2 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

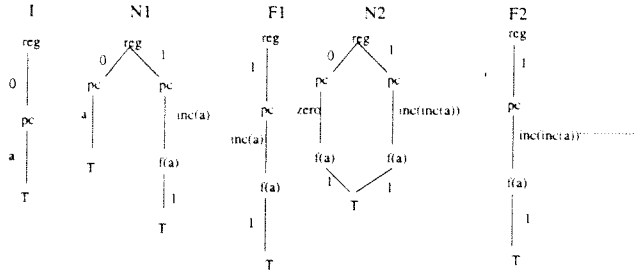

Figure 2: First MDGs generated for the ASM in Figure 1

of $reg = 0$ and $pc = a$, where a is a fresh variable. In the initial state, the ASM can stay there or it can reach the state s_1 under the condition $f(a) = 1$ with $reg = 1$ and $pc = inc(a)$. This is represented in Figure 2 by the MDG $N1$ which contains two paths. The frontier set $F1$ is computed by removing the paths on the left-hand side of $N1$, since it is subsumed by I. $N2$ represents the reachable states from $F1$ in one step. If $r = 0$ then the ASM goes back to the initial state with *zero* loaded in the state variable *pc*. If $r = 1$, then the ASM stays in s_1 with $reg = 1$ and *pc* containing $inc(inc(a))$. The path on the left-hand side in $N2$ is subsumed by the single path of I, because the 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 $pc = inc(a)$ in $N2$, but the presence of the same guard $f(a)$ in both paths results in non-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 in $N2$) with the state $reg = 1 \wedge pc = inc(a) \wedge f(a) = 1$ (path on the right-hand side in $N1$). Let us point out that this guard is generated from the initial state, where *pc* would have had the initial value *zero*. Hence the argument of f would have been *zero*. Suppose, that we know that, under a specific interpretation in the use context of the circuit, the value of $f(zero)$ is 1. It would be possible to use this information to eliminate the guard, $f(zero) = 1$, by using the rule $f(zero) \rightarrow 1$. Unfortunately, this rule does not apply when we generalize to a as shown in the example. The basic idea to solve this kind of non-termination is to use the partial interpretation of the cross-operator and delay the generalization until after the interpretation. To restore 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. The 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 for which 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 M of Figure 1 is shown in

Figure 3 and Figure 4.

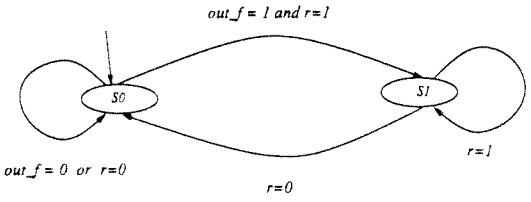

Figure 3: The control-Part (M_1) of the ASM M

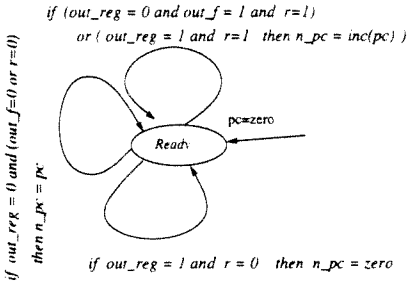

Figure 4: The data-Part (M_2) of the ASM M

M_1 communicates its state information to M_2 through the output 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 non-termination problem in the machine M_2, that can be retimed as needed, i.e, to obtain a register at the output of the cross-term f. In Figure 5

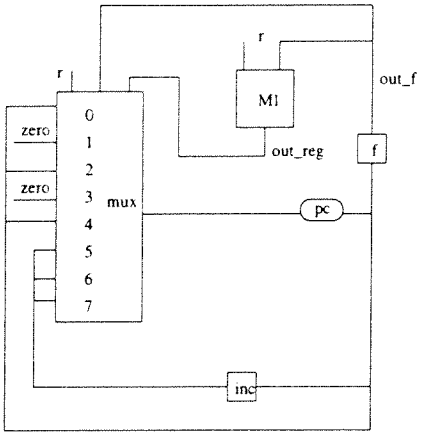

Figure 5: Structural description of M_2 and its connections to M_1

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 as follows[2]:

$$(out\_reg = 0 \wedge out\_f = 1 \wedge r = 1) \vee (out\_reg = 1 \wedge r = 1) \rightarrow n\_pc = inc(pc)$$
$$\mid out\_reg = 1 \wedge r = 0 \rightarrow n\_pc = zero$$
$$\mid n\_pc = pc$$

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 this retiming is shown in Figure 6. At the output of f, the register *pc* is replaced by a register *pc_f* of sort bool, and at the input of *inc* is replaced by a regsiter *pc_inc*.

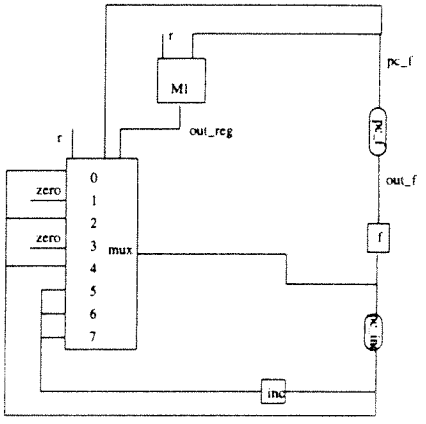

Figure 6: Structural description of M_2 after retiming and its connections to M_1

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 *pc_inc* is *zero* and the initial value of *pc_f* is $f(zero)$, which is equal to 1[3]. This partial interpretation of f must be retained in the register *pc_f* until the control machine M_1 uses the condition $f(zero) = 1$. 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.

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

[3]Recall that to avoid non-termination of the reachability analysis, we must use the partial interpretation, $f(zero) = 1$

The logic which controls the register pc_f can be implemented by a multiplexer, having as control signals the primary input r, the register reg which provides information about the current state of M_1 and pc_f itself. In order to implement this control for pc_f, we use the equation related to pc_f from Figure 6, where $pc\_f = f(pc\_inc)$. The next state for the register pc_f is given by $n\_pc\_f = f(pc\_inc')$. By replacing pc_inc' by its value, which is the same as n_pc, we get:

$(out\_reg = 0 \wedge pc\_f = 1 \wedge r = 1)$
$\vee (out\_reg = 1 \wedge r = 1) \rightarrow n\_pc\_f = f(inc(pc\_inc))$
$\mid out\_reg = 1 \wedge r = 0 \rightarrow n\_pc\_f = f(zero)$
$\mid n\_pc\_f = f(pc\_inc)$

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 $f(zero)$, 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 Figure 7.

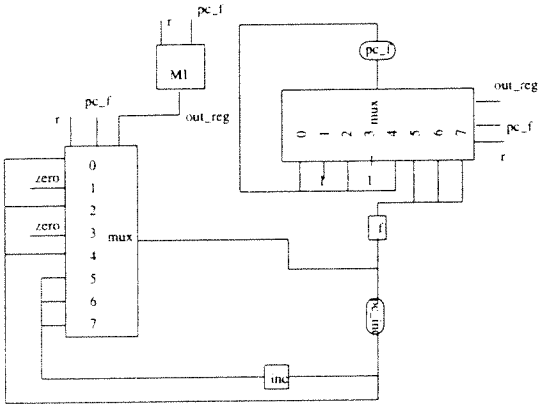

Figure 7: Structural description of M_2 after retiming and circuit transformation and its connections to M_1

Reachability analysis applied for the modified circuit terminates, as shown by the sequences of MDGs presented in Figure 8. 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 $f(zero)$ (Figure 8, MDG I). From this initial state, the ASM can stay there if $r = 0$ or it can reache 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.

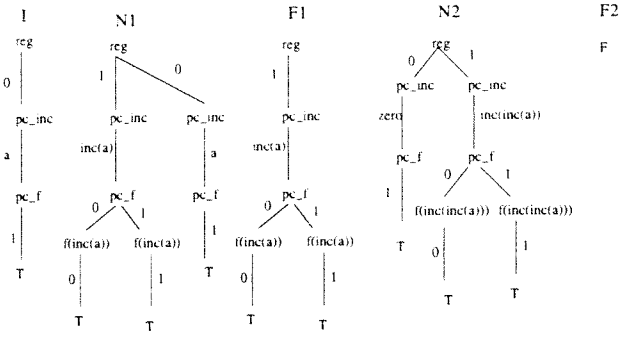

Figure 8: MDGs generated for the retimed ASM.

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

4 Experimental results

We have applied our methodology to the Island Tunnel Controller (ITC). The Island Tunnel Controller was first verified in [9]. The complete ITC specification is composed of 5 communicating state machines: the Island Light Controller (ILC), and the Mainland Light Controller, and the Tunnel Controller, and the Island and Tunnel Counters (see [9, 5] for a detailed description). The tunnel and island counters are typical processor-like loop. However, the composed ASM is no more a processor-like circuit, and the initial state generalization fails. In [9], the authors suggest a heuristic method for state generalisation based on the generalization of every abstract state variable at each clock cycle. This method enlarges unnecessarily the reachable state which implies manual analysis if an invariant is violated. The authors recognize that for complex designs, manual analysis poses difficulties since we have to simulate manually the behaviour of the composite ASM. The application of our method is

systematic, we check just syntactic conditions on the MDG-desciption of the composite ASM and apply the necessary transformations. This operation could be easily automated by using any shell language like Perl. At this momemt this is done manually. Our experimental results are shown in the table 4. They were done on an Ultra SPARC with 124 MB of memory. The CPU time is in seconds and the memory usage is given in megabytes. We verify the same properties for the original specification and the retimed one. In the first case, we use the heuristic state generalization while in the second case the initial state generalization on the retimed specification. The properties that we have verified are listed below:

P1: Cars never travel both 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 (! ($(ic- = 1)$ & $(ic+ = 1)$)).

	Orig spec			Ret Spec		
Prop.	T	M	#N	T	M	#N
P1,P2,P3	55	2.7	4329	11.59	7.9	7287

Table 4: Experimental results.

The number of MDGs nodes is increased in the retimed specification because additional registers are added due to retiming. This slightly affects the memory usage and CPU time. It is the cost to pay for the termination of the abstract state enumeration.

5 Conclusions

In this paper we have shown a novel method to deal with the MDGs non-termination problem. This problem may occur as abstract sort and uninterpreted function symbols are used to model circuits and finite state machines. The key point of our method is based on the idea of retiming in repositioning registers so that state exploration terminates. We have demonstrated the effectiveness of our method on a non trivial example, the Island Tunnel Controller on which state exploration did not terminate.

Acknowledgments

The work was partially supported by NSERC Canada-Nortel Cooperative Research Grant *CRD* 191958.

References

[1] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.

[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 Transactions on Computer-Aided Design*, 13(4):401–424, April 1994.

[3] F. Corella, Z. Zhou, X. Song, M. Langevin, and E. Cerny. Multiway decision graphs for automated hardware verification. *Formal Methods in System Design*, 10(1):7–46, February 1997.

[4] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchrounous sequential machines based on symbolic execution. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[5] K. Fisler and K. Johnson. Integrating Design and Verification Environments Through A Logic Supporting Hardware Daigrams. In *Proc. IFIP Conference on Hardware Description Languages and their Applications (CHDL'95)*, Chiba,Japan, August 1995.

[6] M. Langevin, S. Tahar, Z. Zhou, X. Song, and E. Cerny. Behavioral verification of an ATM switch fabric using implicit abstract state enumeration. In *Proc. of the International Conference on Computer Design (ICCD'96)*, Austin, Texas, USA, October 1996.

[7] C. Leiserson and J. Saxe. Retiming Synchronous Circuitry. *Algorithmica*, 6:5–35, 1991.

[8] S. Tahar, Z. Zhou, X. Song, .E Cerny, and M. Langevin. Formal verification of an ATM switch fabric using multiway decision graphs. In *Proc. of the Great Lakes Symposium on VLSI (GLS-VLSI'96)*, Arres, Iowa, USA, March 1996. IEEE Computer Society Press.

[9] Z. Zhou, X. Song, S. Tahar, F. Corella, E. Cerny, and M. Langevin. Formal verification of the island tunnel controller using multiway decision graphs. In *Proc. of International Conference on Formal Methods in Computer Aided Design (FMCAD'96)*, Palo Alto, Calfornia, USA, November 1996.