Transformation-Based Verification Using Generalized Retiming

Andreas Kuehlmann[1] and Jason Baumgartner[2]

[1] Cadence Berkeley Labs, Berkeley, CA 94704
[2] IBM Enterprise Systems Group, Austin, TX 78758

Abstract. In this paper we present the application of generalized retiming for temporal property checking. Retiming is a structural transformation that relocates registers in a circuit-based design representation without changing its actual input-output behavior. We discuss the application of retiming to minimize the number of registers with the goal of increasing the capacity of symbolic state traversal. In particular, we demonstrate that the classical definition of retiming can be generalized for verification by relaxing the notion of design equivalence and physical implementability. This includes (1) omitting the need for equivalent reset states by using an initialization stump, (2) supporting negative registers, handled by a general functional relation to future time frames, and (3) eliminating peripheral registers by converting them into simple temporal offsets. The presented results demonstrate that the application of retiming in verification can significantly increase the capacity of symbolic state traversal. Our experiments also demonstrate that the repeated use of retiming interleaved with other structural simplifications can yield reductions beyond those possible with single applications of the individual approaches. This result suggests that a tool architecture based on re-entrant transformation engines can potentially decompose and solve verification problems that otherwise would be infeasible.

1 Introduction

The main bottleneck of temporal property checking is the potentially exorbitant computational resources necessary for state traversal. In general, there is no clear dependency between the structure or size of the analyzed circuit and the resource requirements to perform reachability analysis. However, a smaller number of state bits, i.e., registers, generally correlates with a lower memory and runtime consumption for performing state traversal. In particular, for BDD-based techniques [1, 2] fewer registers result in fewer BDD variables which typically decreases the size of the BDDs representing the set of states and transitions among them. Similarly, in SAT-based state enumeration [3], the complexity of the state recording device directly depends on the number of registers. A second motivation for our work comes from the observation that a reduced number of registers often decreases the functional correlation between them. Intuitively, this produces a less scattered state encoding which results in a more compact BDD or cube structure for BDD or SAT-based reachability analysis, respectively.

In this paper we discuss the application of retiming to reduce the number of registers with the goal of improving symbolic reachability analysis. Retiming is commonly referred to as a structural transformation of a circuit-based design description

that changes the positions of the state holding elements without modifying the input-output behavior [4]. Traditionally, the use of retiming is focused on design synthesis with two constraints that fundamentally limit the solution space: the circuit must be physically implementable and it must preserve its original input-output behavior. In property verification these restrictions can be lifted, which results in significantly more freedom for register minimization. There are three extensions of classical retiming for a generalized application in verification. First, a temporally partitioned state traversal eliminates the restriction on the retimed circuit of having an equivalent reset state. Second, a generalized symbolic state traversal algorithm can handle "negative registers." This significantly increases the solution space for legal retimings by removing the non-negative register count constraints from the problem formulation. Third, state bits which are exclusively driven by primary inputs or drive only primary outputs represent a mere temporal shift of peripheral values, and can be suppressed for state space traversal.

In this paper we describe the application of retiming for verification using these three generalizations. This work provides a specific approach in a more general scheme for property checking which uses a set of targeted circuit transformations. In an engine-based architecture, a retiming engine is applied as one step in a series of transformations which gradually simplify the verification problem until it can be solved by a terminal engine (e.g., BDD- or SAT-based). Note that such a modular, transformation-based approach was key in making automatic logic synthesis practical [5].

2 Illustrating Example

Figure 1a shows a circuit example with six registers $R_1, \ldots, R_6$, two inputs a and b, and one output p. Using a notation introduced in Section 4, the initial states of the six registers are assumed to be $I = (I^1_{21}, I^1_{24}, I^2_{24}, I^1_{36}, I^1_{54}, I^1_{6p}) = (1, 0, 0, 1, 0, 1)$. The subscript and superscript denote the circuit arc and the register position along this arc, respectively. Further, let $p \equiv 1$ be a predicate to be checked for all reachable states.

Retiming moves registers forward and backward across gates with the goal of minimizing their count. The corresponding optimization problem can be formulated as an Integer Linear Program (ILP) using a directed graph model of the circuit [4]. The graph vertices and arcs represent gates and interconnection (i.e., wires), respectively. A special *host* vertex is introduced which is connected to all inputs and outputs. Figure 1b shows the retiming graph for the given example. The arc labels denote the number of registers at the corresponding nets. The ILP determines a *lag* for each vertex which represents the number of registers moved backward through it [4].

The original definition of retiming for synthesis requires preserving input-output behavior. With this restriction, the circuit of Figure 1a cannot be retimed since registers R_1 and R_2 have incompatible initial states and cannot be merged by a backward move. To show this, if both registers were shared with a joint initial state of 1, the sequence $(a, b) = ((0, 0), (1, 0), (0, 0))$ would produce $p = (1, 1, 1)$ and $p = (1, 1, 0)$ in the original and retimed circuit, respectively. Similarly, for a joint initial state of 0 the sequence $(a, b) = ((1, 0), (0, 0), (1, 0), (0, 0))$ would distinguish the behavior of the circuits.

In verification, we need not to preserve input-output equivalence of the retimed circuit as long as we can preserve the truth of the given properties. The requirement for

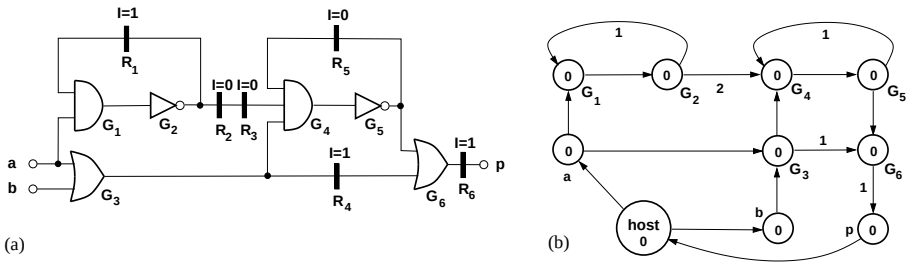

Fig. 1: Retiming example: (a) original circuit, (b) retiming graph.

equivalent reset states can be relaxed by unrolling the circuit for multiple cycles until the set of retimed initial states is uniquely determined. This corresponds to a temporal decomposition of the verification task into two parts: (1) checking a bounded acyclic initialization structure, further referred to as the *retiming stump*, and (2) checking the retimed circuit, further referred to as the *retimed recurrence structure*. The first part involves a SAT check to prove the correctness of the properties for the time frames that are included in the retiming stump. The second part involves model checking the retimed circuit, which effectively provides an inductive correctness proof for all remaining time frames. The initialization state of the retimed circuit can be computed by symbolically simulating the retiming stump up to the retimed recurrence structure.

Registers at the inputs and outputs are mere temporal signal offsets and do not impact the state reachability of the circuit core [6]. Thus, they can be ignored during reachability analysis. For failing properties, the offsets are restored by temporal shifts in the counter-example trace. Adopting the terminology from Malik et al. [7] we will refer to this method as *peripheral retiming*. For peripheral retiming the host vertex is removed from the retiming graph, causing the ILP to pull as many registers as possible out of the circuit. Figure 2a shows the graph for a maximal peripheral retiming of the example ignoring initial state equivalence. The arc labels represent the register counts of the original and retimed circuit. The vertex labels denote their lag, i.e., the number of registers that have been pushed backward through them. As shown, by merging R_1 and R_2 and removing R_6, the register count could be reduced from six to four.

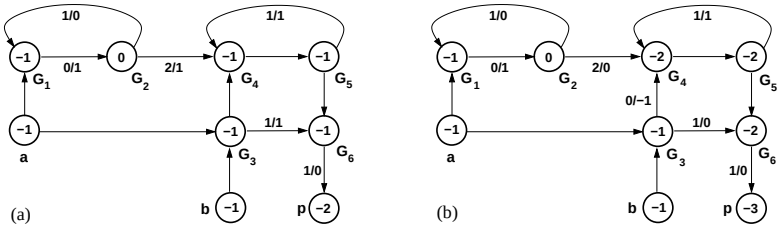

Fig. 2: Graphs for relaxed retimings for the example of Fig. 1 : (a) peripheral retiming ignoring reset state equivalence, (b) retiming with negative registers permitted.

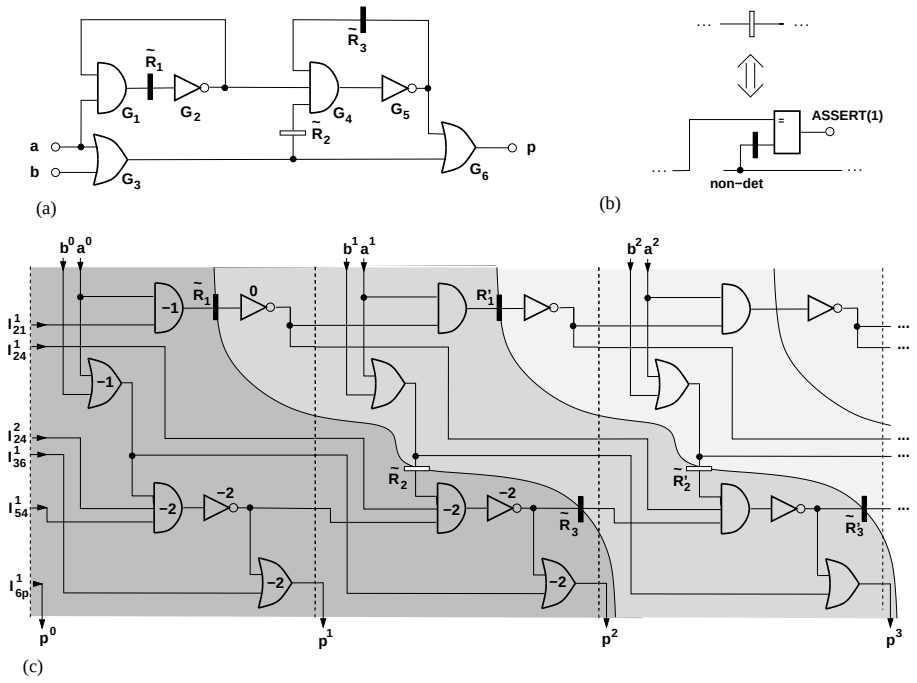

Fig. 3: Retiming result of Fig. 2b: (a) retimed circuit, (b) intuitive interpretation of negative registers, (c) interpretation of the unrolled circuit structure (dark: retiming stump, medium shaded: retiming recurrence structure, lightly shaded: retiming top).

A third relaxation of retiming is achieved by enabling negative register counts at the arcs. This approach is motivated by the fact that registers merely denote functional relations between different time frames. In logic synthesis, clocked or unclocked delay elements are used to physically implement these relations. Such delays can only realize backward constraints, each consisting of a combinational expression in the present and a variable in a future time frame. In symbolic verification, this limitation can be lifted and arbitrary relations can be handled. This includes forward constraints between variables in the current time frame and expressions in future time frames, represented by negative registers. In contrast to the common case of symbolic forward traversal, constraints imposed by negative registers delay the decision about the actual reachability of a state until all referred future time frames are processed. This results in a third component for the above described temporal verification decomposition, reflected by the *retiming top*.

To enable negative registers, the non-negativity constraints on the arc labels are removed from the ILP. Figure 2b shows the resulting retiming graph for the example. By using one negative register, the total register count is reduced to three. Figure 3a shows the resulting circuit. Note that these three registers reflect the actual temporal relations present in the loops and reconverging paths of the original circuit. Figure 3b gives an intuitive interpretation of negative registers in a circuit context. In symbolic reachability

analysis, negative registers can simply be handled by exchanging the current and next state variables in the transition relation. Figure 3c illustrates the retiming process using the unrolled circuit structure. The medium shaded area reflects the retimed recurrence structure which is passed to symbolic model checking. The dark area denotes the retiming stump which is used to compute the initial state for the retimed circuit and to verify p for the first two time frames. The lightly shaded area represents the retiming top.

The actual verification process consists of several steps. First, we need to prove that the property holds for the retiming stump using a SAT check. In the given example, it easy to show that $p^i \equiv 1$ for $i = 0, 1, 2$. Further, the set of initial states $\tilde{I}$ for the retimed recurrence structure is computed by symbolically executing the stump, resulting in $\tilde{I} = \{(\tilde{I}^1_{12}, \tilde{I}^1_{34}, \tilde{I}^1_{54}) \mid \exists a^0.\exists b^0.\exists v.(\tilde{I}^1_{12} \equiv a^0 \wedge \tilde{I}^1_{34} \equiv v \wedge \tilde{I}^1_{54} \equiv 1)\} = \{(0,0,1),(0,1,1),(1,0,1),(1,1,1)\}$. Next, starting from these initial states, symbolic traversal is performed on the retimed structure. This leads to a counter example for the initial state $(\tilde{I}^1_{12}, \tilde{I}^1_{34}, \tilde{I}^1_{54}) = (0,1,1)$ with the inputs $a^1 = 0$ and $b^1 = 0$. Further, the retiming top imposes a constraint on the negative register $\tilde{I}^1_{34} \equiv a_2 \vee b_2$ which can be satisfied for the given failing state. A complete counter-example trace is composed by a satisfying assignment of the retiming stump for generating the required reset state of the retimed structure, a counter-example trace generated by the retimed structure, and a satisfying assignment for the constraint imposed by the negative registers. For the given example, this results in $(a, b) = ((0,0),(0,0),(0,1))$.

3 Previous Work

The application of structural circuit transformations in sequential verification is a relatively new research area. Hasteer et al. [8] proposed the concepts of retiming and state space folding for sequential equivalence checking. Their state-folding technique works for circuits in which the number of latches contained in loops and reconverging paths is constant modulo n. In this case n succeeding state transitions can be concatenated for symbolic state traversal. Baumgartner et al. [9] extend the state-folding concept to handle arbitrary registers and general CTL property checking. The idea of state space folding is orthogonal to the retiming approach described in this paper, and the combination of both techniques is a promising subject of our future research.

For logic optimization, Leiserson and Saxe [10] describe the application of structural retiming and propose an ILP [4] formulation using a graph model. Malik et al. [7] were the first to introduce peripheral retiming with the objective of moving a maximum number of registers to the circuit boundaries. This makes the combinational circuit core as large as possible for providing maximum freedom for conventional combinational optimizations. They also introduced the concept of negative registers as a method of temporarily "borrowing" registers from inputs and outputs. After finishing the combinational optimization, these registers are "legalized" by retiming them back to positive registers. In contrast, our paper describes the direct application of negative registers for verification and gives formal algorithms to fully handle them.

The problem of generating valid initial states for the retimed circuit has been addressed in several publications. Touati and Brayton [11] proposed a method for adding reset circuitry which forces an equivalent initial state. Even et al. [12] described a mod-

ified retiming algorithm that favors forward retiming, allowing a simple computation of the initial states. All previous work on reset state computation assumes input-output equivalence. In this paper we propose a method of eliminating that limitation for verification and describe how a more generalized reset state can be obtained.

Gupta et al. [6] were first to propose the application of maximal peripheral retiming in the context of simulation-based verification. They showed that peripheral registers can be omitted during test generation without compromising the coverage of the resulting transition tour. Still, their approach is focused on test generation and does not consider full reachability. Further, the paper does not address the initialization problem and does not use the concept of negative registers. The work of Cabodi et al. [13], which uses retiming to enhance symbolic reachability analysis, is the closest to ours. However, they use an original synthesis retiming algorithm with the above-mentioned limitations regarding enforced reset state equivalence and non-negative registers. Further, the applied retiming grid is based on next-state functions which significantly reduces the optimization freedom. Consequently, the reported results show mostly modest improvements over existing techniques.

4 Generalized Retiming for Verification

Let $C = (G, E)$ denote a circuit where G represents a set of combinational gates, primary inputs, and primary outputs, and $E \subseteq G \times G$ is a set of arcs connecting the gates. Each arc $(u, v) \in E$ is associated with a non-negative weight $w(u, v)$ representing the number of registers at this arc. Clearly, for all hardware designs we can assume that the initial register count of all arcs is non-negative: i.e., $w(u, v) \geq 0$. Further, without loss of generality, we assume that the circuit does not contain combinational loops.

Let $I^i_{uv}, 1 \leq i \leq w(u, v)$ denote the initial value of register i along arc (u, v) and $g_u(f_{ju}, \ldots, f_{ku})$ be the function of gate u using the functions $f_{ju}, \ldots, f_{ku}$ of arcs $(j, u), \ldots, (k, u)$ at its inputs. If u represents a primary input, g_u denotes the sampled input value at a given time. The state of C at time $t \geq 0$ is computed recursively as:

$$f^t_{uv} = \begin{cases} I^{w(u,v)-t}_{uv} & \text{if } t < w(u, v), \\ g_u^{t-w(u,v)} & \text{otherwise,} \end{cases}$$
$$g^t_u = g_u(f^t_{ju}, \ldots, f^t_{ku}). \tag{1}$$

This definition of f can be used to express the function of any internal net of the design modeled by C. For example, the value at time t of the net connecting the output of register i with the input of register $i + 1$ of arc (u, v) is $f^{t+w(u,v)-i}_{uv}$.

A retiming of C is defined as a gate labeling $r : G \to \mathbb{Z}$, where $r(u)$ is the *lag* of gate u denoting the number of registers that are moved backward through it. The new arc weights $\tilde{w}$ of the retimed circuit $\tilde{C}$ are computed as follows:

$$\tilde{w}(u, v) = w(u, v) + r(v) - r(u). \tag{2}$$

In this context we are interested in minimizing the total number of registers of $\tilde{C}$:

$$\sum_{\forall (u,v) \in E} |\tilde{w}(u, v)| \to \min. \tag{3}$$

Note that due to the missing host vertex, the formulation aims at maximal peripheral retiming which removes registers from the primary inputs and outputs. The given modeling does not take into account that the registers of the outgoing arcs from a gate can be shared and must be counted only once in the objective function. A correct ILP modeling of "register sharing" can be achieved by a slightly modified problem formulation for which the details are presented in [4]. In contrast to retiming for synthesis, we do not impose a non-negative constraint on $\tilde{w}$. Therefore, the new circuit may have negative arc weights, representing negative registers.

Equation (2) imposes an equivalence relation on the set of retimings. Two retimings r_1 and r_2 result in identical circuits and are said to be equivalent if and only if $r_1 = r_2 + c$, where c denotes an integer constant. We define a normalized retiming r' as:

$$r' = r - \max_{\forall u} r(u). \tag{4}$$

In the following we will use the term retiming to denote normalized retimings. Similar to formula (1), for a given retiming r the state of $\tilde{C}$ at time t can be computed as:

$$\tilde{f}^t_{uv} = \begin{cases} \tilde{I}^{\tilde{w}(u,v)-t}_{uv} & \text{if } t < \tilde{w}(u,v), \\ \tilde{g}^{t-\tilde{w}(u,v)}_u & \text{otherwise,} \end{cases}$$
$$\tilde{g}^t_u = g_u(\tilde{f}^t_{ju}, \ldots, \tilde{f}^t_{ku}), \tag{5}$$

where the $\tilde{I}^i_{uv}$ represent the initial states of $\tilde{C}$. In contrast to formula (1), it is not obvious that this formula is well formed, because the $\tilde{w}(u,v)$ can assume negative values.

Theorem 1. *Let C be a circuit containing a finite number of gates, arcs, and non-negative registers without combinational loops, and r be a retiming resulting in circuit $\tilde{C}$. The evaluation of formula (5) for computing the state of $\tilde{C}$ at time t will terminate for any finite $t \geq 0$.*

Proof. First, it is obvious that t remains non-negative during the evaluation of (5). Second, since C and therefore $\tilde{C}$ contain a finite number of gates, any non-terminating evaluation of formula (5) must involve an infinite recursion on at least one gate. Let u be one of those gates and $p = u \overset{(u,u_1)}{\longrightarrow} u_1 \overset{(u_1,u_2)}{\longrightarrow} \ldots \overset{(u_n,u)}{\longrightarrow} u$ be the circular path in $\tilde{C}$ corresponding to the recursion. The difference between t and t' of two suceeding recursions is then $t - t' = \tilde{w}(u,u_1) + \tilde{w}(u_1,u_2) + \ldots + \tilde{w}(u_n,u)$. A substitution using (2) leads to $t - t' = w(u,u_1) + w(u_1,u_2) + \ldots + w(u_n,u)$. All registers are positive ($w(u_i,u_j) \geq 0$), and there are no combinational loops ($\exists (u_i,u_j) \in p \; with \; w(u_i,u_j) > 0$). Therefore t strictly decreases after each recursion which causes the evaluation to terminate once $t < \tilde{w}(u_i,v_j)$ for some arc $(u_i,u_j) \in p$. □

The retiming stump of a retiming r is a partial unrolling of C and is defined as:

$$S = \{s^t_{uv} \mid s^t_{uv} = f^t_{uv} \; \wedge \; (u,v) \in E \; \wedge \; 0 \leq t < \tilde{w}(u,v) - r(v)\}. \tag{6}$$

The new verification structure is composed of S and $\tilde{C}$, where S provides the arc functions for the first cycles and the initial states for the positive registers of $\tilde{C}$ as follows:

$$\tilde{I}^i_{uv} = s^{\tilde{w}(u,v)-i-r(v)}_{uv}, \;\; 0 < i \leq \tilde{w}(u,v). \tag{7}$$

Note that this formula is well formed for normalized retimings because $r(v) \leq 0$.

Theorem 2. *Let C be a circuit containing a finite number of gates, arcs, and non-negative registers without combinational loops and r be a retiming resulting in circuit $\tilde{C}$ and the retiming stump S. The following relations provide a bijective mapping between each arc function of $\{\tilde{C}, S\}$ to the corresponding arc function of C and vise versa:*

$$f_{uv}^t = \begin{cases} s_{uv}^t & \text{if } t < \tilde{w}(u,v) - r(v), \\ \tilde{f}_{uv}^{t+r(v)} & \text{otherwise}, \end{cases} \tag{8}$$

$$\begin{aligned} s_{uv}^t &= f_{uv}^t && \text{if } t < \tilde{w}(u,v) - r(v), \\ \tilde{f}_{uv}^t &= f_{uv}^{t-r(v)}. \end{aligned} \tag{9}$$

Proof. First we show that function (8) correcly maps $\{\tilde{C}, S\}$ to C: For $t < \tilde{w}(u,v) - r(v)$, (8) reflects the definition of s given in (6). For $t \geq \tilde{w}_{uv} - r(v)$, after substitution using (5), we must show that $f_{uv}^t = g_u(\tilde{f}_{iu}^{t+r(v)-\bar{w}(u,v)}, \ldots, \tilde{f}_{ju}^{t+r(v)-\bar{w}(u,v)})$ which is done by inductively proving for the arguments of g_u that $\tilde{f}_{iu}^{t+r(v)-\bar{w}(u,v)} = f_{iu}^{t-w(u,v)}$. *Base case* $(t + r(v) - \tilde{w}(u,v) < \tilde{w}(i,u))$: Using (5) and (7) we get $\tilde{f}_{iu}^{t+r(v)-\bar{w}(u,v)} = \tilde{I}_{iu}^{\bar{w}(i,u)-t-r(v)+\bar{w}(u,v)} = f_{iu}^{t+r(v)-\bar{w}(u,v)-r(u)}$ which, after applying (2), shows the required equality. *Inductive step* $(t + r(v) - \tilde{w}(u,v) \geq \tilde{w}(i,u))$: A substitution using (5) results in $\tilde{f}_{iu}^{t+r(v)-\bar{w}(u,v)} = g_i(\tilde{f}_{hi}^{t+r(v)-\bar{w}(u,v)-\bar{w}(i,u)}, \ldots, \tilde{f}_{ki}^{t+r(v)-\bar{w}(u,v)-\bar{w}(i,u)})$. If $\tilde{w}(i,u) > 0$ we can immediately reduce the arguments of g_i by induction which results in $g_i(f_{hi}^{t-w(u,v)-w(i,u)}, \ldots, \tilde{f}_{ki}^{t+w(u,v)-w(i,u)}) = f_{iu}^{t-w(u,v)}$ and show equivalence. If $\tilde{w}(i,u) \leq 0$, then the right hand side needs to be further expanded until an inductive reduction can be performed. A termination analysis similar to the proof of theorem 1 can be applied showing the superscript value of f will eventually decrease and therefore the expansion will terminate after a finite number of steps. Next, showing that (9) correctly maps $\{\tilde{C}, S\}$ to C is straight forward by using the definition for s for the first part and an inductive proof identical to the one used in the first theorem for the second part. □

Corollary 1. *Let $\tilde{C}$ be derived from C by retiming and c be a Boolean constant, then*

$$\forall t.(f_{uv}^t \equiv c) \Leftrightarrow \forall t.[(0 \leq t < \tilde{w}(u,v) - r(v)) \Rightarrow (s_{uv}^t \equiv c)] \wedge \forall t'.(\tilde{f}_{uv}^{t'} \equiv c). \tag{10}$$

In other words, generalized retiming provides a circuit transformation that is sound and complete for verifying properties of the form $AG(p)$, where the primary circuit inputs are non-deterministic and p is a predicate on any net of the circuit. Its application for more complex safety properties requires that the property formula be expressed as a circuit which is composed with the actual design before retiming can be applied. Similarly, in order to handle constrained circuit inputs, the verification environment must be composed with the circuit before retiming can be applied.

Corollary 2. *Let $\tilde{C}$ be a circuit derived from C by retiming and S be the corresponding retiming stump. Further, let $AG(p)$ be a property that fails for $\tilde{C}$ for an initial state $\tilde{I}$ resulting in a counter-example trace $\tilde{T}$. The counter example T for the original circuit C can be obtained by applying formula (8) on $\tilde{T}$ and S.*

In essence, formula (8) provides the mechanism for trace lifting that back-translates any counter example from the retimed circuit to the original circuit.

5 Transformation-based Verification

We implemented the retiming transformation as a re-entrant reduction engine with a "push" interface similar to a BDD package. The engine consumes a circuit from a higher-level engine, performs retiming, and then passes the resulting circuit down to a lower-level engine. For debugging of failing properties, the engine implements a back-translation mechanism that passes counter-example traces from the lower-level engine back to the higher-level. This setting allows an iterative usage of retiming and other reduction algorithms until the circuit can be passed to a "terminal" decision engine.

As an internal data structure we use a two-input AND/INVERTER graph similar to the one presented in [14] except that registers are modeled as edge attributes. This representation allows the application of several on-the-fly reduction algorithms, including inverter "dragging" and forward retiming of latches, both enabling a generalized identification of functionally identical structures by hashing. As an ILP solver we utilized the primal network simplex algorithm from IBM's Optimization Solutions Library (OSL) [15] to solve the register minimization problem.

As a second simplification engine, we implemented an algorithm for combinational redundancy removal which was adopted from an equivalence checking application [14]. This engine uses BDD sweeping and a SAT procedure to identify and eliminate functionally equivalent circuit structures, including the removal of redundant registers. As a terminal reachability engine we adapted VIS [16] version 1.4 (beta) for our experiments. In addition to the partitioned transition relation algorithm, VIS 1.4 incorporates a robust hybrid image computation approach.

6 Experimental Results

We performed a number of experiments to evaluate the impact of retiming on symbolic reachability analysis, using 31 sequential circuits from the ISCAS89 benchmarks and 27 circuits randomly selected from IBM's Gigahertz Processor (GP) design. All experiments were done on an IBM RS/6000 Model 260, with a 256 MBytes memory limit.

In the first set of experiments we assessed the potential of generalized retiming for reducing register count. In particular, we evaluated an iterative scheme where the retiming engine (RET) and the combinational reduction engine (COM) are called in an interleaved manner. The results for the ISCAS and GP circuits are given in Table 1. For the ISCAS benchmarks, we list only the circuits with more than 16 registers since smaller designs are of less interest for these experiments. Columns 2, 3, and 4 report the number of registers of the original circuit, after applying COM only, and RET only, respectively. The following columns give the register counts after performing various numbers of iterations of COM followed by RET. The number of negative registers, if non-zero, is given in parentheses. For brevity, we report only up to three iterations; more iterations provided only marginal improvements. The reported maximum lag in column 9 gives an indication of the size of the retiming stump.

Overall, the results indicate that generalized retiming has a significant potential for reducing the number of registers for verification. For the ISCAS benchmarks we obtained a maximum register reduction of 79% with an average of 27%. For the processor circuits we achieved an average reduction of 62%.

Design	Number of Registers (negative)						Relative	Max.	Time (s) /	Results of
	Original	COM Only	RET Only	COM-RET 1 Iteration	COM-RET 2 Iterations	COM-RET 3 Iterations	Reduction (Best)	Lag	Memory (MB) (Best)	[6]/ [13] (Registers)
PROLOG	136	81	45 (1)	45 (1)	45 (3)	44 (2)	67.6%	2	1.4 / 22.4	- / -
S1196	18	16	16	14	14	14	22.2%	1	0.6 / 10.7	16 / -
S1238	18	17	16	15	14	14	22.2%	1	0.9 / 21.1	17 / -
S1269	37	37	36	36	36	36	2.7%	1	0.4 / 6.2	- / -
S13207_1	638	513	390	343	292 (1)	289	54.7%	11	3.8 / 34.7	- / -
S1423	74	74	72	72	72	72	2.7%	1	0.5 / 6.2	72 / 74
S1512	57	57	57	57	57	57	0.0%	1	0.5 / 6.2	- / 57
S15850_1	534	518	498	488	485	485	9.2%	6	5.3 / 31.8	- / -
S3271	116	116	110	110	110	110	5.2%	5	0.7 / 7.0	- / 116
S3330	132	81	44 (2)	44 (3)	44 (2)	44 (2)	66.7%	3	0.7 / 7.0	- / -
S3384	183	183	72	72	72	72	60.7%	6	0.7 / 7.1	- / 147
S35932	1728	1728	1728	1728	1728	1728	0.0%	1	7.2 / 38.0	- / -
S382	21	21	15	15	15	15	28.6%	1	0.3 / 5.9	15 / -
S38584_1	1426	1415	1375	1375	1374	1374	3.6%	5	29.4 / 127.4	- / -
S400	21	21	15	15	15	15	28.6%	0	0.3 / 5.9	15 / -
S444	21	21	15	15	15	15	28.6%	1	0.3 / 5.9	15 / -
S4863	104	88	37	37	37	37	64.4%	4	0.9 / 7.3	- / 96
S499	22	22	22	22	20	20	9.1%	1	0.6 / 15.1	- / -
S526N	21	21	21	21	21	21	0.0%	2	0.4 / 5.9	- / -
S5378	179	164	112 (6)	112 (6)	111 (6)	111 (6)	38.0%	5	1.6 / 18.4	- / 144
S635	32	32	32	32	32	32	0.0%	1	0.4 / 5.9	- / -
S641	19	17	15	15	15	15	21.1%	2	0.4 / 5.9	18 / -
S6669	239	231	92	75	75	75	68.6%	5	1.6 / 14.1	- / -
S713	19	17	15	15	15	15	21.1%	2	0.4 / 5.9	- / -
S838_1	32	32	32	32	32	32	0.0%	0	0.5 / 6.1	- / -
S9234_1	211	193	172	172	165	131	37.9%	3	2.5 / 26.2	- / -
S938	32	32	32	32	32	32	0.0%	0	0.4 / 6.1	- / -
S953	29	29	6	6	6	6	79.3%	0	0.4 / 6.1	- / -
S967	29	29	6	6	6	6	79.3%	0	0.4 / 6.1	- / -
S991	19	19	19	19	19	19	0.0%	2	0.4 / 6.0	- / -
C_RAS	431	431	378	370	348	348	19.3%	3	6.0 / 22.6	- / -
D_DASA	115	115	100	100	100	100	13.0%	2	0.9 / 7.1	- / -
D_DCLA	1137	1137	771	750	750	750	34.0%	1	35.4 / 36.2	- / -
D_DUDD	129	129	100	100	100	100	22.5%	3	0.9 / 7.0	- / -
I_IBBC	195	195	40	40	38	36	81.5%	2	1.6 / 21.6	- / -
I_IFAR	413	413	142	139	136	136	67.1%	4	3.1 / 19.5	- / -
I_IFEC	182	182	45	45	45	45	75.3%	6	0.7 / 7.0	- / -
I_IFPF	1546	1356	673 (4)	661 (4)	449 (2)	442 (2)	71.4%	10	46.5 / 127.9	- / -
L_EMQ	220	220	87	88	74	74	66.4%	4	3.4 / 18.5	- / -
L_EXEC	535	535	163	137	135	134	75.0%	6	9.8 / 28.1	- / -
L_FLUSH	159	159	1	1	1	1	99.4%	3	0.8 / 7.0	- / -
L_LMQ	1876	1831	1190	1185	433 (3)	425 (3)	77.3%	3	50.7 / 139.1	- / -
L_LRU	237	237	94	94	94	94	60.3%	2	1.1 / 7.1	- / -
L_PNTR	541	541	245	245	245	245	54.7%	3	1.8 / 8.8	- / -
L_TBWK	307	307	124	124	40	40	87.0%	3	2.7 / 18.0	- / -
M_CIU	777	686	415	415	411	387 (1)	50.2%	15	26.3 / 76.6	- / -
S_SCU1	373	373	204	200	192	192	48.5%	3	9.0 / 20.6	- / -
S_SCU2	1368	1368	566	565	426	423	69.1%	5	102.2 / 67.4	- / -
V_CACH	173	155	104 (2)	96 (3)	96 (2)	95 (1)	45.1%	9	1.1 / 24.0	- / -
V_DIR	178	151	87	83	43	42 (1)	76.4%	5	0.9 / 22.3	- / -
V_L2FB	75	75	26	26	26	26	65.3%	2	0.5 / 5.9	- / -
V_SCR1	150	128	52	48 (1)	48 (1)	48	68.0%	4	0.7 / 10.9	- / -
V_SCR2	551	551	86	82	82	82	85.1%	4	4.4 / 15.0	- / -
V_SNPC	93	93	21	21	21	21	77.4%	4	0.5 / 6.8	- / -
V_SNPM	1421	1216	233 (7)	233 (7)	231 (11)	227 (8)	84.0%	15	14.7 / 65.2	- / -
W_GAR	242	232	91 (1)	90	90	79 (1)	67.4%	2	3.2 / 25.4	- / -
W_SFA	64	64	42	42	41	41	35.9%	1	1.0 / 16.0	- / -

Table 1: Retiming results for ISCAS circuits (upper part) and GP circuits (lower part).

Design	Original Circuit			Reduced Circuit				Relative
	Number of Registers	Reachability Steps, Algo	Time (sec) / Memory(MB)	Number of Registers	Reachability Steps, Algo	BDD_{init} Nodes	Time (sec) / Memory(MB)	Improvement Time / Memory
PROLOG	136	17 C I	2285 / 134.5	45	16 C H	611	81.6 / 27.5	96.4% / 79.6%
S1196	18	4 C I	1.1 / 6.5	14	2 C I	122	0.5 / 6.3	54.5% / 3.1%
S1238	18	4 C I	1.2 / 6.5	14	2 C I	159	0.1 / 6.3	91.7% / 3.1%
S1269	37	11 C H	13194 / 185.5	36	11 C H	901	13395 / 187.5	-1.5% / -1.1%
S3330	132	17 C H	668.0 / 35.3	45	16 C I	194	35.8 / 15.6	94.6% / 55.8%
S382	21	13 C I	< 0.1 / 6.2	15	11 C I	17	< 0.1 / 6.1	0.0% / 1.6%
S400	21	10 C I	< 0.1 / 6.2	15	10 C H	16	< 0.1 / 6.1	0.0% / 1.6%
S444	21	4 C I	< 0.1 / 6.1	15	3 C H	27	< 0.1 / 6.1	0.0% / 0.0%
S4863	104	3 I	14400 / 174.2	37	4 C I	199	14.8 / 16.6	99.9% / 90.5%
S499	22	1 C H	0.2 / 6.2	20	1 C H	21	< 0.1 / 6.2	100% / 0.0%
S641	19	6 C I	0.8 / 6.4	15	5 C I	15	1.0 / 6.4	-25.0% / 0.0%
S713	19	6 C I	0.9 / 6.3	15	5 C I	15	0.6 / 6.4	33.3% / -1.6%
S953	29	6 C I	0.8 / 6.4	6	5 C H	7	< 0.1 / 6.1	100% / 4.7%
S967	29	4 C I	1.1 / 6.3	6	3 C H	7	< 0.1 / 6.1	100% / 3.2%
C_RAS	431	1028 C I	724.3 / 57.2	370	1026 C I	415	424.0 / 51.8	41.5% / 9.4%
D_DASA	115	6 C I	19.7 / 7.8	100	5 C I	200	33.0 / 11.6	-67.5% / -48.7%
D_DUDD	129	13 C I	953.3 / 112.8	100	11 C H	2568	359.1 / 33.7	62.3% / 70.1%
I_IBBC	195	5 C H	145.3 / 11.4	40	3 C H	41	4.4 / 6.4	97.0% / 43.9%
I_IFAR	413	5 I	14400 / 87.0	139	22 C I	719	2302 / 102.0	84.0% / -17.2%
I_IFEC	182	6 C I	66.3 / 8.4	45	2 C H	151	28.0 / 6.9	57.8% / 17.9%
L_EMQ	220	8 C H	323.7 / 17.0	88	5 C H	5519	205.6 / 33.0	36.5% / -94.1%
L_EXEC	535	5 H	14400 / 63.2	137	9 C I	1856	593.6 / 103.2	95.9% / -63.3%
L_FLUSH	159	4 C I	37.4 / 7.7	1	2 C H	2	< 0.1 / 6.2	100% / 19.5%
L_PNTR	541	6 C I	6687 / 138.5	245	3 C I	242	2423 / 51.2	63.8% / 63.0%
L_TBWK	307	6 C H	184.1 / 9.1	124	4 C H	123	74.0 / 7.4	59.8% / 18.7%
S_SCU1	373	14 C H	8934 / 165.8	200	12 C H	755	1195 / 118.1	86.6% / 28.8%
V_CACH	173	11 C H	92.1 / 17.2	97	8 C I	910	20.0 / 8.9	78.3% / 48.3%
V_DIR	178	8 C H	57.9 / 8.3	83	2 C I	95	11.1 / 7.0	80.8% / 15.7%
V_L2FB	75	4 C I	2.9 / 6.3	26	2 C H	27	< 0.1 / 6.1	100% / 3.2%
V_SCR1	150	20 C H	250.0 / 17.7	48	17 C I	90	5.0 / 15.5	98.0% / 12.4%
V_SCR2	551	22 C I	1201 / 105.0	82	20 C I	220	260.0 / 36.7	78.4% / 65.0%
V_SNPC	93	4 C H	4.9 / 6.6	21	1 C H	17	< 0.1 / 6.2	100% / 6.1%
W_GAR	242	11 C I	109.8 / 25.0	90	9 C H	191	82.5 / 13.0	24.9% / 48.0%
W_SFA	64	7 C I	3.7 / 6.8	42	6 C I	14	3.6 / 6.9	2.7% / -1.5%

Table 2: Effect of retiming on reachability analysis (C = completed within the time limit of four hours, H = hybrid image computation, I = IWLS95 image computation).

The number of negative registers generated by retiming is surprisingly small. This can be explained by the two-input AND/INVERTER data structure used as circuit representation. One can show that within each strongly connected component (SCC) of such circuits, there exists an optimal retiming with only positive registers. Only paths between the SCCs may require negative registers for an optimal solution.

Table 2 gives the performance results for symbolic reachability analysis. We report results for all circuits of Table 1 for which retiming resulted in a register reduction and reachability analysis could be completed. We ran each experiment with two options for the VIS image computation: the IWLS95 partitioned transition relation method and the hybrid approach. The best of the two results on a per-example basis are then reported. Although after reduction we can complete traversal for only three additional circuits, the results clearly show that retiming significantly improves the overall performance. The CPU time is decreased by an average of 53.1% for ISCAS and 64.0% for GP circuits, respectively. The corresponding memory reductions are 17.2% and 12.3%, respectively. The cumulative run time speedup is 55.7% for the ISCAS benchmarks and 83.5% for

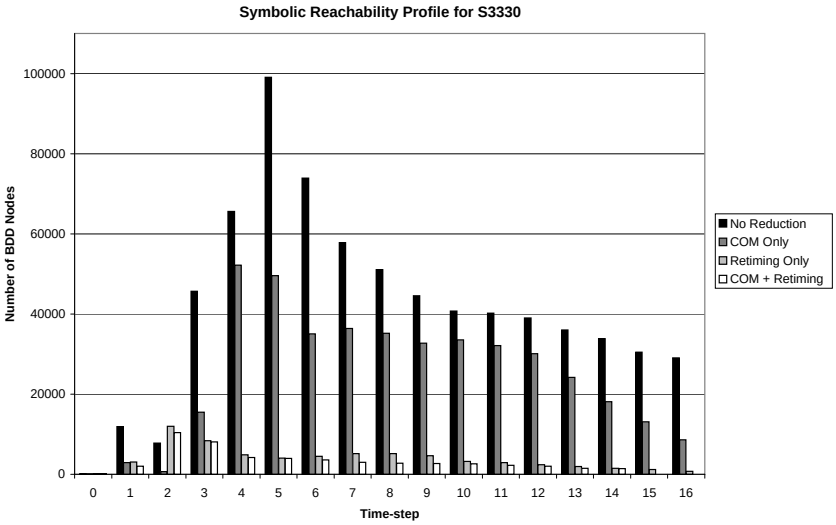

Fig. 4: BDD size profile for traversing S3330 with method IWLS95 after various transformations.

the GP circuits. To illustrate the complexity of the retiming stump, we report the BDD sizes for the initial states in column 7. As shown, these BDDs remain fairly small and do not impact the complexity of the reachability analysis.

Figure 4 shows the profile of the BDD size while traversing benchmark S3330 for the original circuit and after applying various reduction steps. This example demonstrates how retiming typically benefits the performance of the traversal. To further illustrate the effect of retiming on the correlation of the state encoding, we analyzed the traversal of circuit S4863. Reachability timed out during the third traversal step of the original circuit. Using retiming, the correlation between the remaining registers was completely removed resulting in full reachability of all 2^{37} states. While such a profound result is likely atypical, this is strong evidence of the power of both structural simplification and retiming to reduce register correlation.

7 Conclusions and Future Work

We presented the application of generalized retiming for enhancing symbolic reachability analysis. We discussed three extensions of the classical retiming approach which include: (1) eliminating the need for equivalent reset states by introducing the concept of an initialization stump, (2) supporting negative registers, handled as general functional relations to future time frames, and (3) removing peripheral registers by converting them into simple temporal offsets. We implemented the presented algorithm in a transformation-engine-based tool architecture that allows an efficient iteration between multiple reduction engines before the model is passed to a terminal reachability algorithm. Our experiments based on standard benchmarks and industrial circuits indicate that the presented approach significantly increases the capacity of standard reachability algorithms. In particular, we demonstrated that the repeated interleaved application of

retiming and other restructuring algorithms in a transformation-based setting can yield reduction results that cannot be achieved with a monolithic approach.

In this paper the application of retiming is focused on minimizing the total number of registers as an approximate method for enhancing reachability analysis. It does not take into account that the actual register placement can have a significant impact on other algorithms used for improving symbolic state traversal. An interesting problem for future research is to extend the formulation of structural transformations beyond simple retiming to obtain a more global approach for improving reachability analysis.

References

1. O. Coudert, C. Berthet, and J. C. Madre, "Verification of synchronous sequential machines based on symbolic execution," in *International Workshop on Automatic Verification Methods for Finite State Systems*, Springer-Verlag, June 1989.
2. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: 10^{20} states and beyond," in *IEEE Symposium on Logic in Computer Science*, pp. 428–439, IEEE, June 1990.
3. T. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in *The European Conference on Design Automation*, pp. 214–218, IEEE, February 1991.
4. C. Leiserson and J. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, pp. 5–35, 1991.
5. J. A. Darringer, D. Brand, J. V. Gerbi, W. H. Joyner, and L. H. Trevillyan, "Logic synthesis through local transformations," *IBM Journal on Research and Development*, vol. 25, pp. 272–280, July 1981.
6. A. Gupta, P. Ashar, and S. Malik, "Exploiting retiming in a guided simulation based validation methodology," in *Correct Hardware Design and Verification Methods (CHARME'99)*, pp. 350–353, September 1999.
7. S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and resynthesis: Optimizing sequential networks with combinational techniques," *IEEE Transactions on Computer-Aided Design*, vol. 10, pp. 74–84, January 1991.
8. G. Hasteer, A. Mathur, and P. Banerjee, "Efficient equivalance checking of multi-phase designs using retiming," in *IEEE International Conference on Computer-Aided Design*, pp. 557–561, November 1998.
9. J. Baumgartner, A. Tripp, A. Aziz, V. Singhal, and F. Andersen, "An abtraction algorithm for the verification of generalized C-slow designs," in *Conference on Computer Aided Verification (CAV'00)*, pp. 5–19, July 2000.
10. C. Leiserson and J. Saxe, "Optimizing synchronous systems," *Journal of VLSI and Computer Systems*, vol. 1, pp. 41–67, January 1983.
11. H. J. Touati and R. K. Brayton, "Computing the initial states of retimed circuits," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 157–162, January 1993.
12. G. Even, I. Y. Spillinger, and L. Stok, "Retiming revisited and reversed," *IEEE Transactions on Computer-Aided Design*, vol. 15, pp. 348–357, March 1996.
13. G. Cabodi, S. Quer, and F. Somenzi, "Optimizing sequential verification by retiming transformations," in *37th ACM/IEEE Design Automation Conference*, pp. 601–606, June 2000.
14. A. Kuehlmann, M. K. Ganai, and V. Paruthi, "Circuit-based Boolean reasoning," in *Proceedings of the 38th ACM/IEEE Design Automation Conference*, ACM/IEEE, June 2001.
15. M. S. Hung, W. O. Rom, and A. Waren, *Optimization with IBM OSL*. Scientific Press, 1993.
16. The VIS Group, "VIS: A system for verification and synthesis," in *Conference on Computer Aided Verification (CAV'96)*, pp. 428–432, Springer-Verlag, July 1996.