Min-Area Retiming on Flexible Circuit Structures

Jason Baumgartner
IBM Enterprise Systems Group
Austin, TX 78758

Andreas Kuehlmann
Cadence Berkeley Labs
Berkeley, CA 94704

Abstract

In this paper we present two techniques for improving min-area retiming that combine the actual register minimization with combinational optimization. First, we discuss an on-the-fly retiming approach based on a sequential AND/INVERTER/REGISTER *graph. With this method the circuit structure is sequentially compacted using a combination of register "dragging" and* AND *vertex hashing. Second, we present an extension of the classical retiming formulation that allows an optimal sharing of fanin registers of* AND *clusters, similar to traditional fanout register sharing. The combination of both techniques is capable of minimizing the circuit size beyond that possible with a standard Leiserson and Saxe retiming approach on a static netlist structure. Our work is primarily aimed at optimizing the performance of reachability-based verification methods. However, the presented techniques are equally applicable to sequential redundancy removal in technology-independent logic synthesis. A large set of experiments using benchmark and industrial circuits demonstrate the effectiveness of the described techniques.*

1 Introduction

Retiming is a structural optimization technique that relocates the registers in a logic circuit with the objective of minimizing their total count, maximizing the circuit performance, or achieving both goals simultaneously [1, 2]. Traditionally, retiming is applied on a fixed circuit graph and repositions the registers without altering the actual logic structure. When interleaved with combinational optimization steps, a repeated application of retiming can optimize the overall circuit structure significantly.

In this paper we present two specific techniques that extend the classical formulation and application of retiming by allowing it to operate on a more flexible structure. First, we describe an on-the-fly retiming approach that is based on a sequential AND/INVERTER/REGISTER graph. It merges registers and combinational circuit components by structural hashing, which is applied during graph construction. Similar to inverter removal in combinational circuit compaction [3], the proposed approach "drags" registers through the sequential circuit graph as far as possible. As a result, many registers and AND vertices can be merged, which leads to a significant reduction of the circuit size without significant computational overhead.

The second technique is based on the idea that, similar to the sharing of fanout registers, fanin registers of an AND cluster can be optimally shared by adjusting the AND decomposition. As a result, the number of necessary registers is reduced to its optimum, which is equal to the maximum number of registers along any of the incoming cluster edges. We describe a corresponding extension of the retiming formulation. It is based on the AND/INVERTER/REGISTER graph and models the sharing of fanin registers in a similar manner as Leiserson and Saxe modeled fanout sharing [2]. We further describe an algorithm that optimally reconstructs an AND tree decomposition based upon the retiming solution.

The presented technique takes a new view of the retiming formulation by departing from the traditional use of a fixed circuit structure. The extended formulation provides an exact retiming model that considers all possible implementations of the AND clusters of a circuit. To our knowledge, there are only two previous publications related to our work. In [4], a technique is presented that simultaneously considers multiple structures for possible logic implementations using a choice node. This method is mainly aimed at technology mapping and, despite its recursive capability, must explicitly generate candidate structures for an AND cluster decomposition including possible retiming configurations. In our approach, we defer the actual decomposition step until after the optimal retiming is computed. The applied modeling guarantees that there exist a decomposition of the AND clusters with the exact number of registers minimized during retiming. In [5], the concept of algebraic factorization is extended to sequential expressions, which implicitly intertwines retiming with structural rewriting. Based on the concept of synchronous division, a set of sequential transformations is outlined, which can be applied in a general synthesis scenario. In contrast to our work, this technique is based on individual, local restructuring steps and does not model the decomposition flexibility of the expressions for global retiming.

The main focus of our work is to apply the presented retiming techniques for improving reachability-based functional verification. Although there is no clear dependency between circuit size and the complexity of BDD-based [6, 7] or SAT-based [8] reachability analysis, a smaller number of circuit elements generally correlates to lower computational resources in both techniques. In particular, with BDD-based state traversal, fewer registers results in fewer BDD variables, which typically decreases the size of the BDDs representing the set of states and transitions among them. SAT-based state exploration can be improved by reducing the total number of circuit elements as they establish potential points for case splits. The techniques presented in this paper are aimed at removing sequential redundancy, which effectively reduces register count and the number of combinational gates.

In this context we do not need to preserve the circuit's input/output behavior as long as the retiming transformation is sound and complete for proving properties. As shown in [9], retiming can be generalized for verification by: (1) omitting the need for equivalent reset states, (2) supporting negative registers, and (3) eliminating peripheral registers [10]. The presented techniques focus on these generalizations and their application to verification. However, we believe that they are equally applicable to logic optimization for removing sequential redundancy during technology-independent synthesis. For example, the applied peripheral retiming can be viewed as a temporary phase to "hide" or "borrow" registers from the environment. After optimization, these registers are moved back to or from the circuit to restore its original input/output behavior [11]. Similar precaution is needed for preserving initial state equivalence [12, 13].

2 Illustrative Example

We restrict our presentation to bit-level circuits based on edge-triggered or master/slave flip-flops (registers) with designated initial states. Extensions to level sensitive flip-flops or multi-bit registers are largely straight-forward and hence are not discussed in this paper.

Figure 1 provides an example circuit to demonstrate the two presented techniques. The original circuit shown in part (a) contains seven registers. On-the-fly retiming is derived from the concept of on-the-fly compaction of combinational circuits [14, 3]. In addition to AND vertex hashing, forward retiming is applied during graph construction by "dragging" as many registers through the vertices as possible. The graph is built starting from the primary inputs and, for cyclic circuits, from any cuts of the register loops. Figure 1b shows the result of on-the-fly retiming when the circuit of (a) is processed from the primary inputs and a cut at register r_1. As shown, registers $\{r_1, r_2, r_3, r_4\}$ and the two input inverters of gate g_2 have been dragged through that gate. This allows g_2 to merge with g_1, as well as the sharing of registers r_5 and r_6 with r_1'. No further forward retiming is possible because input x_2 does not provide a register that could be shared at the output of gate g_3. Note that this result is identical to an optimal retiming computed by the standard Leiserson and Saxe min-area retiming algorithm [2].

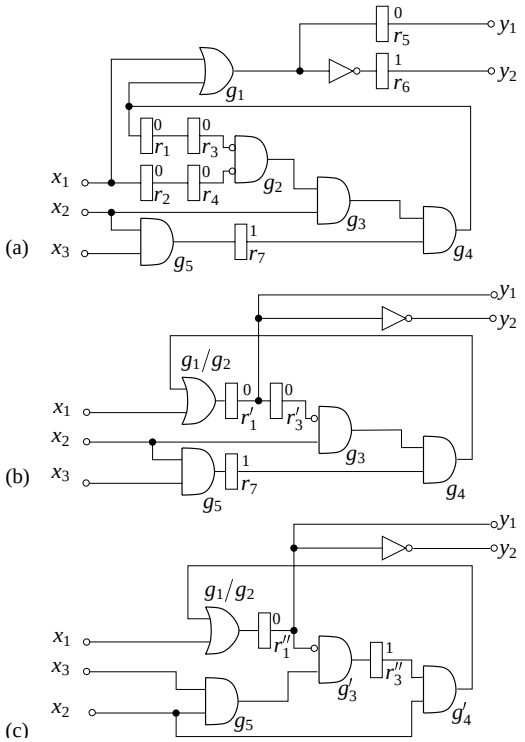

Figure 1: Example for the application of retiming: (a) original circuit, (b) circuit after on-the-fly retiming, (c) circuit after retiming with AND cluster input sharing.

Figure 1c shows a functionally equivalent version of the circuit that uses only two registers. Here the two registers r_3' and r_7 in front of the AND cluster g_3/g_4 have been merged into register r_3''. This structure can be obtained from circuit (b) by applying combinational synthesis – i.e., rearranging gates g_3 and g_4, followed by another retiming move. Clearly, in such a two-step approach, it is not obvious that this combinational optimization will perform the needed gate rearrangement because it cannot foresee its benefit in the following retiming step. On the other hand, if the retiming formulation could take into account all possible decompositions of the three-input AND cluster g_3/g_4, the optimal structure depicted in (c) could be generated in one step.

In Sections 4 and 6 we discuss the details of the on-the-fly retiming approach and the new retiming formulation that precisely models optimal register sharing for AND clusters, respectively.

3 AND/INVERTER/REGISTER Graph

Let $C = (G, E)$ denote a circuit where G represents a set of AND vertices, primary inputs, and primary outputs, and $E \subseteq G \times G$ is a set of edges connecting the vertices. Each edge $(u, v) \in E$ is associated with a non-negative weight $w(u, v) \in \mathbb{N}$ representing the number of registers along this edge, a set of corresponding initial values $I_1(u, v), \ldots I_w(u, v)$, and an inverter attribute $i(u, v) \in \{0, 1\}$ where $i = 1$ or $i = 0$ indicates whether the edge function is to be complemented or not, respectively.

The functions represented by two edges are sequentially equivalent if (though not necessarily "only if") they have: (1) the same source vertex, (2) identical inverter attributes, and (3) the same number of registers with matching initial values. We use a compact 64-bit word to uniquely represent an edge of the AND/INVERTER/REGISTER graph. The word is composed of four bit fields: an index into the array of graph vertices, the number of edge registers, an index to a canonical representation of their initial states, and a single bit to indicate edge complementation. Using this data structure, a simple comparison of two words can decide whether two edges are functionally equivalent.

The canonical representation of initial values is based on a tree structure where the paths correspond to sequences of initial values of the edges of C. The tree root is a dummy node, representing a NULL register. The first level of children represent all possible initial values of the first registers of the edges. The tree branching structure corresponds to the different combinations of initial val-

ues of all edges. By ensuring uniqueness of the individual paths and subpaths during tree construction and manipulation, a pointer to any of the tree nodes provides a representation that is canonical for that particular set of initial values.

Figure 2a illustrates the AND/INVERTER/REGISTER graph for the circuit example of Figure 1a including the corresponding initial value tree. The graph was built starting from the primary inputs and a cut at register r_1. Interior vertices represent AND gates.

4 On-the-fly Retiming

On-the-fly retiming is applied to remove sequential redundancy during the construction of the AND/INVERTER/ REGISTER graph. Similar to the use of an AND/INVERTER graph for combinational circuits [3], this approach can result in a significant compaction of the circuit representation without notable time or memory overhead. We canonize edges on-the-fly by dragging inversions past registers (which inverts their initial values). The on-the-fly retiming step is integrated into an algorithm for constructing an AND gate, which is given in Figure 3. The graph construction starts at the primary inputs and an arbitrary set of register cuts of the cyclic circuitry. For each register that is cut, first a dummy AND vertex is created and used as a place holder. Once the structure for the next-state function of a register is built, the placeholder is merged onto that structure. A repeated forward hashing can then be applied to possibly further compact the graph structure.

As shown, first the algorithm performs constant folding similar to methods applied in combinational circuit compaction [3]. Next,

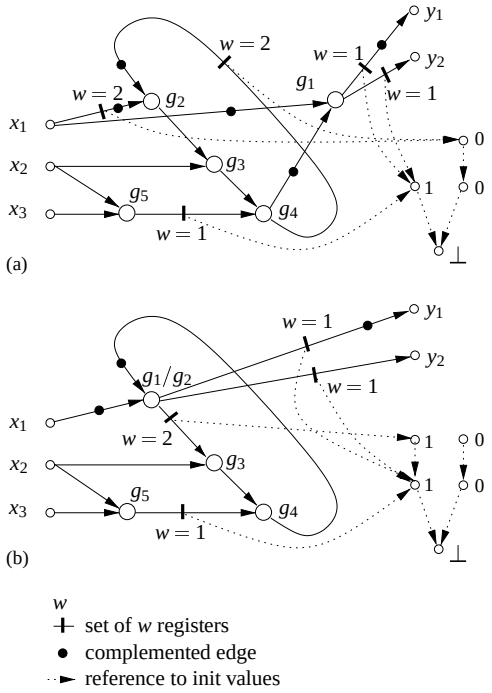

Figure 2: AND/INVERTER/REGISTER graph example: (a) graph for the original circuit of Figure 1a, (b) graph after on-the-fly retiming, which results in the circuit of Figure 1b.

```
/* Create_And takes two operand edges e1 and
   e2, and returns a vertex representing the
   AND of e1 and e2                              */
Algorithm Create_And(e1,e2) {
   if (e1 == const_0)     return const_0;
   if (e2 == const_0)     return const_0;
   if (e1 == const_1)     return e2;
   if (e2 == const_1)     return e1;
   if (e1 == e2)          return e1;
   if (e1 == ¬e2)         return const_0;

   /* Truncate as many registers as possible
      from both edges and store them in Ii   */
   wmin = Min(w(e1),w(e2));
   e'1,I1 = Truncate_Registers(e1,wmin);
   e'2,I2 = Truncate_Registers(e2,wmin);
   /* Merge the initial states by AND       */
   I  = And_Initial_States(I1,I2);

   /* Apply ranking to catch commutativity */
   if (Rank(e'1) > Rank(e'2)) Swap(e'1,e'2);
   /* Hash lookup for vertex with e'1 and e'2 */
   e = Hash_Lookup(e'1,e'2);
   if (e == NULL) {
   /* Allocate new vertex if lookup failed
      and add to hash table                  */
      e = Create_And_Vertex(e'1,e'2);
   }
   /* Add back AND of stripped registers   */
   return e+I;
}
```

Figure 3: Pseudo-code for constructing an AND vertex for the AND/INVERTER/REGISTER graph.

the registers of both edges are truncated by "dragging" as many registers as possible through the AND vertex. The initial states of the retimed registers are computed by a pairwise AND of the initial states of the original edge registers. Note that for complemented input edges, the initial values must be inverted before they can be combined. After truncation, the edges are hashed. If the hash lookup finds a pre-existing isomorphic vertex, it is reused; otherwise a new vertex is constructed. The "dragged" set of registers is added back before the edge is returned. The resulting AND/INVERTER/REGISTER graph for the example of Figure 1b is shown in Figure 2b. The graph was constructed from the original circuit shown in Figure 1 starting from the inputs and a cut at register r_1.

The application of the on-the-fly retiming step can be combined with structural rewriting techniques, methods to detect functionally identical vertices such as BDD sweeping, and circuit-based SAT [15]. The integrated retiming functionality would extend the equivalence checking capability of these algorithms beyond combinational verification and cover a significant class of practical problems to verify retimed circuits [16].

5 Min-Area Retiming

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

$$w_r(u,v) = w(u,v) + r(v) - r(u). \quad (1)$$

For min-area retiming, we are interested in minimizing the total number of registers of C_r, which comprises an integer linear program (ILP):

$$\sum_{\forall (u,v) \in E} |w_r(u,v)| \rightarrow \min . \qquad (2)$$

Note that in this formulation we explicitly omit the host vertex [2] resulting in a retiming which effectively removes all peripheral registers from the primary inputs and outputs. For synthesis applications, these registers are considered temporarily "hidden" or "borrowed" and must be added back after optimization [11]. Peripheral retiming can be disabled by simply adding a host vertex to the graph, which is connected to all input and output vertices [2]. Further, the optimal solution of formula (2) may include negative registers if there are no non-negativity constraints upon w_r. In reachability analysis, negative registers may be simply handled as an inverse constraint between time frames [9]. For synthesis, these registers are considered temporary, and again must be eliminated after optimization. In the following we limit our presentation and experiments to peripheral retiming using only non-negative registers.

6 Optimal Sharing of Fanin Registers

The retiming formulation given in formulas (1) and (2) does not consider the fact that the registers of edges fanning out from the same vertex can be shared. For example, if three fanout edges $e_1 = (u,v), e_2 = (u,v')$ and $e_3 = (u,v'')$ are assigned positive register weights $w_r(e_1), w_r(e_2)$ and $w_r(e_3)$, formula (2) accounts for their sum $w_r(e_1) + w_r(e_2) + w_r(e_3)$ during minimization. However, in a circuit implementation, only $w^{max} = \max(w_r(e_1), w_r(e_2), w_r(e_3))$ registers are needed for the edge with the maximum count. The other two edges can share their registers with this edge.

In [2], a solution is proposed that is based on a modified retiming graph structure. Figure 4a shows the general scheme to alter the retiming graph for fanout register sharing. The idea is to add a dummy vertex v for each regular vertex u with a fanout degree

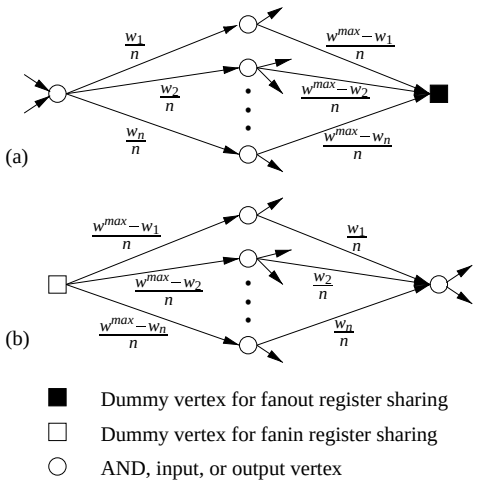

Figure 4: Sharing of fanout and fanin registers: (a) original idea of fanout register sharing [2], (b) extension to fanin register sharing.

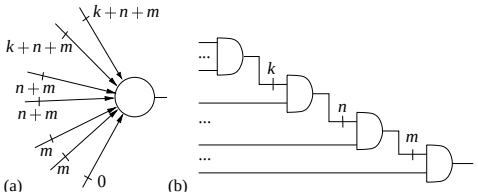

Figure 5: Optimal decomposition of an AND vertex: (a) vertex with incoming edges sorted by weight, (b) resulting AND tree.

greater than two, and to connect edges from the fanout vertices of u to v. Edge weights are modified as shown: each original weight w_i is divided by the number of fanout edges n, and the new edges to the dummy vertex are assigned a weight equal to the difference between $\frac{w^{max}}{n}$ and the modified weight of the corresponding fanout edge. This division is realized by associating a "cost per unit weight" $\beta(u,v) = 1/n$ with each edge, and minimizing a total weighted cost in (2). Note that the sum of all edge weights in each "sharing subcircuit" is equal to w^{max} and that the retiming formulation accounts for w_r^{max} in the overall minimization problem. This exactly models the described sharing of fanout registers.

A similar idea is applicable to fanin register sharing. If the vertices represent identical Boolean functions that are totally symmetric, all possible tree configurations establish a valid decomposition of their function. Examples of such symmetric functions are AND, OR, and XOR vertices. In the following, we use multi-input AND vertices as a base system for fanin sharing. However, the presented concepts are equally applicable to other totally symmetric functions.

Figure 4b shows how the concept of fanout register sharing is adapted to fanin register sharing. Similar to the previous case, a dummy vertex for fanin register sharing is created and edge weights are modified. With this configuration, the retiming optimization problem will minimize the maximum number of registers at any of the fanin edges. Once a min-area retiming is computed, the AND vertex can always be decomposed in a tree structure such that a maximum number of registers are shared.

The scheme for flexible tree decomposition that requires only w_r^{max} registers is illustrated in Figure 5. The algorithm first sorts the incoming edges of the AND vertex by weight. Next, an AND tree is built using the structure of Figure 5b. For each set of inputs with identical register counts, a balanced AND subtree is constructed. The individual subtrees are then connected by registers in a linear sequence. The number of registers assigned to the edges between the subtrees is equal to the difference of their weights.

For maximum fanin sharing, the AND/INVERTER/REGISTER graph produced by the on-the-fly retiming algorithm is first restructured to form maximum AND vertices. Next a retiming graph with the dummy vertices for fanin and fanout sharing is passed to the ILP solver. Note that for simultaneous modeling of fanin and fanout sharing, a splitting vertex must be introduced between two adjacent AND vertices. After computing the optimal retiming, an optimal two-input AND graph is rebuilt using the procedure described above.

Figure 6 shows the retiming graph for the example of Figure 1. Part (a) shows the edge weights for the original problem derived

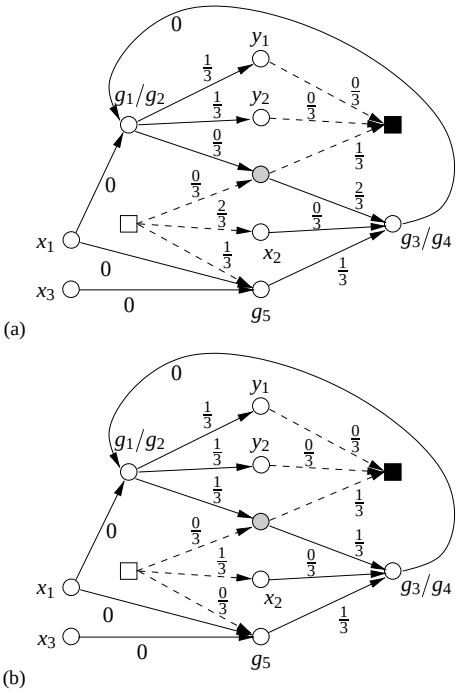

Figure 6: Retiming graph for the circuit of Figure 1b: (a) graph with original weight of 3 registers, (b) optimal solution with 2 registers for the circuit of Figure 1c.

from the on-the-fly retimed circuit of Figure 2b. The top portion of the graph shows the dummy vertex modeling the possible sharing of fanout registers of vertex g_1/g_2. The bottom portion models the possible sharing of fanin registers of vertex g_3/g_4. Note that we arbitrarily assigned the two registers between gates g_1/g_2 and g_3/g_4 to the input portion of gate g_3/g_4. An assignment to the output portion of gate g_1/g_2 would yield identical results. Part (b) shows the resulting weights from the ILP solver, which correspond to the optimal circuit of Figure 1c.

7 Experiments

In this section we provide a set of experimental results for retiming using the presented techniques. All experiments were run on an IBM ThinkPad Model T21, with an 800MHz PIII and 256-Megabyte main memory, running RedHat Linux 6.2. We implemented a retiming engine, utilizing the data structure and algorithms described earlier in this paper. As ILP solver we applied the primal network simplex algorithm from IBM's Optimization Solutions Library (OSL) [17].

In all experiments we used peripheral retiming [10]. We focus here on reduction of the size of the core circuit structure, hence we do not present any results on generating retimed initial values. The structure to produce the initial values of the retimed circuit is referred to as the *retiming stump* [9]. The retiming stump is generally small and does not constitute a bottleneck in the overall verification scheme. This is mainly due to the fact that even for multi-frame initialization structures, large portions may be eliminated by constant propagation.

Table 1 provides results for various retiming options for the ISCAS89 benchmarks. The results are based on the described AND/INVERTER/REGISTER graph representation of the circuit and report the number of 2-input AND vertices and registers. Columns 1 and 2 list the name of the circuits and their initial, unretimed sizes, respectively. Column 3 provides the circuit sizes for retiming without the application of on-the-fly retiming or fanin register sharing. This option is identical to classical peripheral retiming as per [2]. In column 4 we report the result for fanin-register sharing without on-the-fly retiming, whereas for the following column we enabled both.

Columns 6 through 8 provide the results for an iterated application of retiming interleaved with combinational restructuring. For the latter we utilized a combinational simplification engine as described in [15]. We iterated between both engines until no further improvement was gained and reported the best results. Column 6 shows these results using plain retiming (as in column 3), whereas column 7 reports the results of the best option of the techniques used in column 4 or 5. Column 8 indicates the required computing resources for the best run between columns 6 and 7, preferring minimum registers to minimum AND vertices – these two objectives are not always complementary (see below). In column 9 we provide previously published results. As shown, our technique almost always yields lower register counts. Despite detailed analysis, we could not reproduce the results reported in [10] for circuits S344 and S349.

Table 2 provides the data of a set of identical experiments for various IBM Gigahertz Processor (GP) circuits. There are several noteworthy trends in both tables. First, plain retiming decreases register count by an average of 16.8% on the ISCAS circuits, and by 50.1% on the GP circuits. Fanin register sharing allows an additional reduction of the register count by an average of 0.9% and 4.7% for the ISCAS and GP circuits. In addition, the AND count is significantly decreased, by 9.8% for ISCAS and 20.7% for GP.

The additional application of on-the-fly retiming has a varying effect upon size. Our experiments show that on average it hurts both register count and AND count. However, in individual cases, it can provide a substantial benefit. For example, for seven of the 42 ISCAS circuits and eleven of the 28 GP circuits, on-the-fly retiming further reduced the overall AND count. In addition, for three GP circuits the number of registers is decreased. Also, as illustrated in Figure 1, on-the-fly retiming alone may result in register reduction without even solving the retiming problem. For example, the GP circuit L_FLUSH is a reconvergent feed-forward pipeline. Before using the ILP solver to calculate an optimal retiming, the options used in columns 4 and 5 reduce the register count to 78 and 38, respectively. Nevertheless, often on-the-fly retiming temporarily hurts register count, which then gets rectified during the global retiming phase.

We briefly discuss how combinational simplification and on-the-fly retiming can in some cases hurt register count. As an example, assume u and v are two functionally identical yet distinct vertices. Suppose that all outgoing edges from u have a weight of one, and those of v have a weight of zero. It may be that by backwards-

Design	Original circuit	Plain retiming [2]	Retiming with fanin sharing	On-the-Fly retiming with fanin sharing	Iteration of interleaved retiming and combinational restructuring (iterated until no further improvements)			Previous results [10]/[18] (Number of Registers)
					Plain retiming	Best result of columns 4 or 5	CPU time (sec) Memory (MB)	
PROLOG	853 / 136	853 / 45	676 / 45	672 / 46	709 / 45	644 / 45	1.0 / 14.9	- / -
S1196	480 / 18	480 / 16	475 / 16	475 / 16	463 / 16	456 / 16	0.4 / 4.4	16 / -
S1238	533 / 18	533 / 16	532 / 16	532 / 16	518 / 16	513 / 16	0.5 / 6.5	17 / -
S1269	478 / 37	478 / 36	462 / 36	463 / 36	459 / 36	450 / 36	0.3 / 4.4	- / -
S13207_1	3205 / 638	3205 / 389	2604 / 390	2593 / 407	1295 / 266	1221 / 267	3.6 / 31.3	- / -
S1423	507 / 74	507 / 72	458 / 72	458 / 72	461 / 72	455 / 72	0.4 / 5.5	72 / 74
S1488	734 / 6	734 / 6	618 / 6	632 / 6	659 / 6	610 / 6	0.7 / 12.7	- / -
S1494	746 / 6	746 / 6	629 / 6	644 / 6	668 / 6	622 / 6	0.4 / 6.5	- / -
S1512	484 / 57	484 / 57	455 / 57	455 / 57	470 / 57	455 / 57	0.3 / 2.4	- / 57
S15850_1	3852 / 534	3852 / 495	3457 / 498	3465 / 498	3283 / 490	3112 / 475	9.3 / 34.5	- / -
S208_1	77 / 8	77 / 8	70 / 8	71 / 8	70 / 8	70 / 8	0.2 / 2.2	- / -
S27	8 / 3	8 / 3	8 / 3	8 / 3	8 / 3	8 / 3	0.1 / 2.3	- / -
S298	125 / 14	125 / 14	97 / 14	97 / 14	100 / 14	91 / 14	0.2 / 6.3	- / -
S3271	1125 / 116	1125 / 110	1091 / 110	1093 / 110	1082 / 110	1067 / 110	1.0 / 8.7	- / 116
S3330	820 / 132	820 / 45	657 / 45	654 / 46	692 / 45	624 / 45	0.7 / 9.7	- / -
S3384	1070 / 183	1070 / 72	1070 / 72	1070 / 72	1064 / 72	1062 / 72	0.9 / 6.7	- / 147
S344	109 / 15	109 / 15	102 / 15	102 / 15	101 / 15	98 / 15	0.2 / 2.3	7 / -
S349	112 / 15	112 / 15	104 / 15	104 / 15	101 / 15	98 / 15	0.2 / 2.3	7 / -
S35932	12204 / 1728	12204 / 1728	11948 / 1728	11948 / 1728	11660 / 1728	11660 / 1728	14.3 / 38.5	- / -
S382	148 / 21	148 / 15	134 / 15	136 / 15	140 / 15	134 / 15	0.2 / 2.3	15 / -
S38584_1	13479 / 1426	13479 / 1416	11769 / 1375	11811 / 1415	11794 / 1374	11464 / 1373	86.6 / 239.9	- / -
S386	188 / 6	188 / 6	126 / 6	133 / 6	166 / 6	125 / 6	0.2 / 4.3	- / -
S400	158 / 21	158 / 15	141 / 15	143 / 15	148 / 15	141 / 15	0.2 / 2.3	15 / -
S420_1	165 / 16	165 / 16	156 / 16	159 / 16	156 / 16	156 / 16	0.2 / 2.3	- / -
S444	169 / 21	169 / 15	150 / 15	153 / 15	155 / 15	149 / 15	0.2 / 2.3	15 / -
S4863	1750 / 104	1750 / 72	1537 / 37	1537 / 37	1376 / 37	1326 / 37	2.4 / 17.3	- / 96
S499	187 / 22	187 / 22	199 / 22	199 / 22	187 / 22	190 / 20	0.3 / 4.4	- / -
S510	213 / 6	213 / 6	213 / 6	213 / 6	211 / 6	206 / 6	0.3 / 6.4	- / -
S526N	251 / 21	251 / 21	191 / 21	191 / 21	202 / 21	183 / 21	0.3 / 6.4	- / -
S5378	1422 / 179	1422 / 115	1346 / 114	1321 / 124	1260 / 112	1242 / 113	1.4 / 15.0	- / 144
S635	190 / 32	190 / 32	190 / 32	190 / 32	161 / 32	161 / 32	0.2 / 2.3	- / -
S641	160 / 19	160 / 15	132 / 15	132 / 15	146 / 15	131 / 15	0.2 / 3.3	18 / -
S6669	2263 / 239	2263 / 92	2199 / 92	2199 / 92	2238 / 77	2174 / 76	1.1 / 5.8	- / -
S713	174 / 19	174 / 15	137 / 15	137 / 15	149 / 15	130 / 15	0.2 / 5.4	- / -
S820	468 / 5	468 / 5	325 / 5	335 / 5	345 / 5	317 / 5	0.5 / 12.6	- / -
S832	482 / 5	482 / 5	335 / 5	344 / 5	355 / 5	324 / 5	0.4 / 8.5	- / -
S838_1	341 / 32	341 / 32	328 / 32	335 / 32	328 / 32	328 / 32	0.2 / 2.3	- / -
S9234_1	2346 / 211	2346 / 172	1896 / 172	1891 / 174	1437 / 145	1377 / 146	1.8 / 14.3	- / -
S938	341 / 32	341 / 32	328 / 32	335 / 32	328 / 32	328 / 32	0.2 / 2.3	- / -
S953	348 / 29	348 / 6	356 / 6	343 / 6	340 / 6	332 / 6	0.3 / 4.4	- / -
S967	369 / 29	369 / 6	386 / 6	370 / 6	357 / 6	355 / 6	0.3 / 4.4	- / -
S991	299 / 19	299 / 19	297 / 19	297 / 19	297 / 19	297 / 19	0.2 / 2.3	- / -
% Reduction	**0.0 / 0.0**	**0.0 / 16.8**	**9.8 / 17.7**	**9.5 / 17.4**	**10.8 / 18.7**	**14.3 / 18.9**		

Table 1: Retiming results for the ISCAS89 benchmarks (number of two-input AND vertices/number of registers).

retiming u, we can share the retimed registers with registers in the fanin cone of u, decreasing total weight by one. However, if we merge u and v together, we can no longer backwards-retime the new vertex due to the zero-weight outgoing edges, thereby hurting the retiming results.

Iteration of combinational simplification and retiming can provide dramatic reductions. Compared to the single application, an additional average reduction of 4.5% and 1.2% on the ISCAS benchmarks, and 18.6% and 6.3% on the GP circuits, was achieved for the number of AND vertices and registers, respectively. Up to six iterations were applied during these runs, with an average number of 2.6 for ISCAS and 4.6 for GP. The reported results in column 7 utilized on-the-fly retiming on eight of the 42 ISCAS circuits and on six of the 28 GP circuits. One particularly interesting result is that an iterated application of the presented techniques with combinational restructuring significantly outperforms an interleaved classical retiming approach. This demonstrates the overall potential of the presented approaches for functional verification and technology-independent logic synthesis.

In [9], the impact of the presented techniques upon symbolic reachability analysis is discussed. In brief summary: for all circuits for which completion of reachability was at all possible, CPU time is decreased by our techniques 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 the GP circuits.

8 Conclusions and Future Work

In this paper we have presented two enhancements to min-area retiming that are capable of significantly reducing register count and size of the combinational circuitry by departing from the traditional application of retiming on static circuit graphs. We discussed two techniques that work on an AND/INVERTER/REGISTER graph. On-the-fly retiming provides an algorithm that applies forward retiming during graph construction and can identify sequentially redundant subcircuits without significant computing overhead. The concept of fanin register sharing defers the decomposition of clusters of symmetric functions until after retiming. A modified formulation of the retiming problem considers all possible decompositions of such clusters and ensures that an overall minimum number of registers is achieved.

Design	Original circuit	Plain retiming [2]	Retiming with fanin sharing	On-the-Fly retiming with fanin sharing	Iteration of interleaved retiming and combinational restructuring (iterated until no further improvements)		
					Plain retiming	Best result of columns 4 or 5	CPU time (sec) Memory (MB)
CHIP_RAS	2686 / 660	2686 / 585	2103 / 492	2159 / 492	2148 / 489	2039 / 489	4.9 / 32.4
CORE_RAS	2297 / 431	2297 / 379	2200 / 378	2209 / 387	1735 / 341	1873 / 348	2.0 / 14.5
D_DASA	1223 / 115	1223 / 100	967 / 100	968 / 100	844 / 100	815 / 100	0.8 / 8.9
D_DCLA	10916 / 1137	10916 / 771	10483 / 771	10506 / 771	7853 / 750	7443 / 750	23.9 / 94.1
D_DUDD	1295 / 129	1295 / 100	1143 / 100	1146 / 100	1119 / 100	1084 / 100	1.1 / 12.9
I_IBBC	389 / 195	389 / 43	228 / 41	217 / 41	207 / 43	196 / 37	0.5 / 9.7
I_IFAR	1202 / 413	1202 / 147	1031 / 142	1033 / 143	997 / 139	929 / 137	1.7 / 18.5
I_IFEC	334 / 182	334 / 46	302 / 45	309 / 45	308 / 46	287 / 45	0.7 / 15.0
I_IFPF	5896 / 1546	5896 / 705	5273 / 679	4715 / 612	2812 / 350	2768 / 355	43.9 / 78.0
L_EMQ	981 / 220	981 / 88	737 / 87	745 / 88	920 / 86	632 / 74	1.2 / 16.3
L_EXEC	1618 / 535	1618 / 168	1191 / 163	1193 / 197	1178 / 144	974 / 138	2.2 / 19.0
L_FLUSH	893 / 159	893 / 5	495 / 1	409 / 1	358 / 1	338 / 1	0.6 / 8.7
L_LMQ	14074 / 1876	14074 / 1196	12921 / 1190	12983 / 1190	5793 / 432	5363 / 428	41.5 / 91.9
L_LRU	581 / 237	581 / 94	524 / 94	518 / 94	469 / 94	439 / 94	1.0 / 13.1
L_PNTR	1453 / 541	1453 / 245	1351 / 245	1349 / 245	1387 / 245	1325 / 245	1.2 / 8.2
L_TBWK	1160 / 307	1160 / 125	829 / 124	829 / 124	279 / 40	267 / 40	0.8 / 11.0
M_CIU	4550 / 777	4550 / 459	3262 / 415	3244 / 415	2929 / 381	2757 / 379	4.8 / 35.8
S_SCU1	1520 / 373	1520 / 212	1296 / 204	1346 / 207	1308 / 201	1160 / 192	2.8 / 20.2
S_SCU2	8560 / 1368	8560 / 640	6632 / 566	5990 / 564	3928 / 432	4119 / 425	34.6 / 58.9
V_CACH	753 / 173	753 / 103	652 / 105	649 / 110	424 / 95	393 / 97	0.8 / 14.9
V_DIR	554 / 178	554 / 87	491 / 87	285 / 50	160 / 45	152 / 43	0.5 / 10.7
V_L2FB	120 / 75	120 / 26	103 / 26	103 / 26	107 / 26	95 / 26	0.3 / 4.4
V_SCR1	826 / 150	826 / 95	418 / 52	618 / 94	341 / 49	325 / 48	0.6 / 10.6
V_SCR2	2563 / 551	2563 / 458	1157 / 86	2343 / 460	524 / 82	510 / 82	1.4 / 14.3
V_SNPC	78 / 93	78 / 21	68 / 21	68 / 21	67 / 21	62 / 21	0.3 / 5.4
V_SNPM	2421 / 1421	2421 / 241	1843 / 237	1814 / 241	1800 / 232	1221 / 180	33.8 / 116.8
W_GAR	2107 / 242	2107 / 93	1775 / 91	1769 / 91	1896 / 91	1590 / 75	3.3 / 16.8
W_SFA	471 / 64	471 / 42	329 / 42	329 / 42	324 / 41	300 / 41	0.6 / 12.7
% Reduction	**0.0 / 0.0**	**0.0 / 50.1**	**20.7 / 54.8**	**20.3 / 51.8**	**33.6 / 60.3**	**39.3 / 61.1**	

Table 2: Retiming results for selected IBM Gigahertz Processor (GP) circuits.

The main focus of this research is to enhance reachability-based verification [9], for which min-area retiming is the single objective. However, these techniques are equally applicable to logic synthesis to remove sequential redundancy in the technology-independent phase. Our results indicate that the presented retiming can achieve significant reductions of the circuit size in terms of register count and number of combinational gates. In particular, we demonstrated that comparable results are not achievable by classical retiming on static circuit structures applied in an interleaved manner with combinational optimization.

Future work in this area includes improvements of the combined modeling of retiming and combinational optimization. Further, we wish to investigate the application of the AND/INVERTER/REGISTER graph for sequential equivalence checking.

References

[1] C. Leiserson and J. Saxe, "Optimizing synchronous systems," *Journal of VLSI and Computer Systems*, vol. 1, pp. 41–67, January 1983.

[2] C. Leiserson and J. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, pp. 5–35, 1991.

[3] M. K. Ganai and A. Kuehlmann, "On-the-fly compression of logical circuits," in *International Workshop on Logic Synthesis*, May 2000.

[4] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," *IEEE Transactions on Computer-Aided Design*, vol. 16, pp. 313–334, August 1997.

[5] G. D. Micheli, "Synchronous logic synthesis: Algorithms for cycle-time minimization," *IEEE Transactions on Computer-Aided Design*, vol. 10, pp. 63–73, January 1991.

[6] 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*, (Grenoble, France), Springer-Verlag, June 1989.

[7] 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*, (Philadelphia), pp. 428–439, IEEE, June 1990.

[8] T. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in *Proceedings of The European Conference on Design Automation*, pp. 214–218, IEEE, February 1991.

[9] A. Kuehlmann and J. Baumgartner, "Transformation-based verification using generalized retiming," in *Computer Aided Verification (CAV'01)*, (Paris, France), pp. 104–117, July 2001.

[10] 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)*, (Bad Herrenalb, Germany), pp. 350–353, Springer-Verlag, September 1999.

[11] 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.

[12] 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.

[13] 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.

[14] H. Hulgaard, P. Williams, and H. Andersen, "Equivalence checking of combinational circuits using Boolean expression diagrams," *IEEE Transactions on Computer-Aided Design*, vol. 18, July 1999.

[15] A. Kuehlmann, M. K. Ganai, and V. Paruthi, "Circuit-based Boolean reasoning," in *Proceedings of the 38th ACM/IEEE Design Automation Conference*, (Las Vegas, Nevada), pp. 232–237, ACM/IEEE, June 2001.

[16] G. P. Bischoff, K. S. Brace, S. Jain, and R. Razdan, "Formal implementation verification of the bus interface unit for the Alpha 21264 microprocessor," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 16–24, IEEE, October 1997.

[17] M. S. Hung, W. O. Rom, and A. D. Waren, *Optimization with IBM OSL*. Scientific Press, 1993.

[18] G. Cabodi, S. Quer, and F. Somenzi, "Optimizing sequential verification by retiming transformations," in *Proceedings of the 37th ACM/IEEE Design Automation Conference*, (Los Angeles), pp. 601–606, ACM/IEEE, June 2000.