Min-Area Retiming on Dynamic 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 for 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 dynamic 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. This can be done in such a manner that the number of necessary registers is reduced to its optimum, which is equal to the maximum number of registers at any of the incoming cluster edges. We describe a corresponding extension of the retiming formulation. It is based on the mentioned 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 functional verification based on reachability analysis. Although there is no clear dependency between the circuit size and the complexity of BDD-based [6, 7] or SAT-based [8] reachability analysis, a smaller number of circuit elements generally correlates with a lower effort in both techniques. In particular, in BDD-based state traversal, fewer registers result in fewer problem 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 the register count and the number of combinational circuit 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 for verification. However, we believe that they are equally applicable for logic optimization to remove 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 into the circuit to restore its original input/output behavior [11]. Similar precaution is needed for preserving initial state equivalence (e.g., [12, 13]).

2 Illustrating 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 vectored registers are largely straight-forward and hence are not discussed in this paper.

Figure 1 gives an example to demonstrate the idea of the two presented techniques. The original circuit is shown in part (a) and contains seven registers. The idea of on-the-fly retiming is derived from the concept of on-the-fly compaction of combinational circuits [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 gives the result of the on-the-fly retiming when the circuit of (a) is processed from the primary inputs and a cut at register r_1. As shown, the four registers

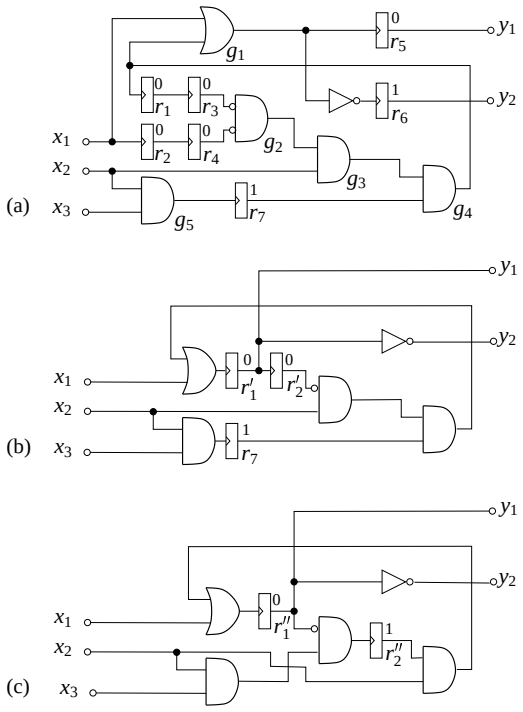

Figure 1: Example for the application of retiming: (a) original circuit, (b) circuit after on-the-fly retiming, (c) circuit after retiming with AND cluster input sharing.

$\{r_1, r_2, r_3, r_4\}$ and the two input inverters of gate g_2 have been dragged through that gate. This allows the retimed gate g_2 to merge with gate g_1 and further to share 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 particular result is identical to an optimal retiming computed by the standard Leiserson Saxe min-area retiming algorithm [2].

Figure 1c shows a functionally equivalent version of the circuit that uses only two registers. Here the two registers r'_2 and r_7 in front of the AND cluster g_3/g_4 have been merged into register r''_2. 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 a combinational optimization move 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 as 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 $w(u,v) \in \mathbf{N}$ representing the number of registers at 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 they have: (1) the same source vertex, (2) identical inverter attributes, and (3) the same number of registers with matching initial values. Similar to the application in BDD packages, 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. The first three fields represent an index into the array of graph vertices, the number of edge registers, and an index to a canonical representation of their initial states, respectively. The last field includes 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 (possibly partial) 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 values 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 gives 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.

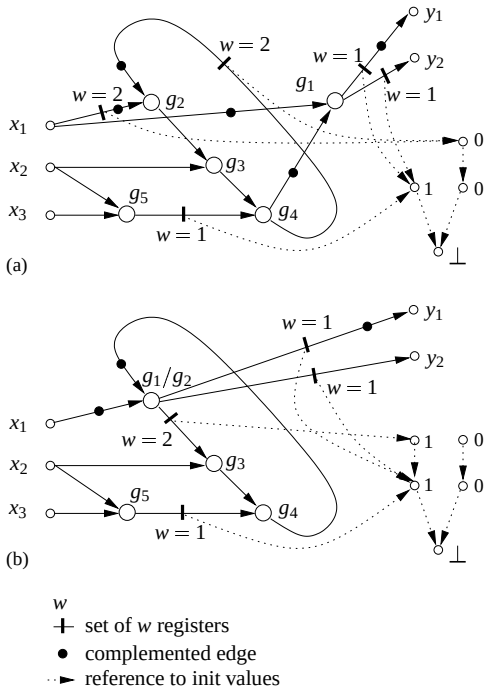

Figure 2: AND/INVERTER/REGISTER graph example: (a) graph for the original circuit of Figure 1a, (b) graph after on-the-fly retiming, which results in the circuit of Figure 1b.

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

```
/* 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 (e1 == 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));
   e1', I1 = Truncate_Registers(e1, wmin);
   e2', I2 = Truncate_Registers(e2, wmin);
   /* Merge the initial states by AND        */
   I  = And_Initial_States(I1, I2);

   /* Apply ranking to catch commutativity */
   if (Rank(e1') > Rank(e2')) Swap(e1', e2');
   /* Hash lookup for vertex with e1' and e2' */
   e = Hash_Lookup(e1', e2');
   if (e == NULL) {
   /* Allocate new vertex if lookup failed
      and add to hash table                  */
      e = Create_And_Vertex(e1', e2');
   }
   /* Add back AND of stripped registers     */
   return e+I;
}
```

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

without noticable time or memory overhead. The on-the-fly retiming step is integrated into the algorithm for constructing an AND gate and 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, 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 need to 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. Note that the "dragged" set of registers is added back before the edge is returned. The resulting AND/INVERTER/REGISTER graph for the given example of Figure 1b is shown in Figure 2b. The graph was constructed from the original circuit shown in part (a) starting from the inputs and a cut at register r_1.

Note that the application of the on-the-fly retiming step can be combined with structural rewriting techniques, methods to detect functional identical vertices such as BDD sweeping, and circuit-based SAT [14]. 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 [15].

5 Min-Area Retiming

A retiming of C is defined as a gate labeling $r : G \rightarrow \mathbf{Z}$, 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:

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

In this context we are interested in minimizing the total number of registers of C_r:

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

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 need to 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 the non-negativity constraints of (1) are relaxed. In reachability analysis, negative registers can be simply handled as an inverse constraint between time frames [9]. For synthesis, these registers are considered temporary, and again need to be eliminated after optimization. In the following we limit our presentation and experiments on peripheral retiming using only non-negative registers.

6 Optimal Sharing of Fanin Registers

The retiming formulation given in formulae (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 terminal vertex to

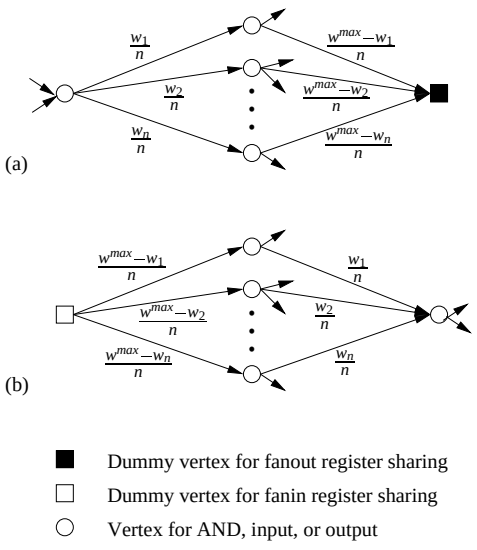

Figure 4: Sharing of fanout and fanin registers: (a) original idea of fanout register sharing [2], (b) extension to fanin register sharing.

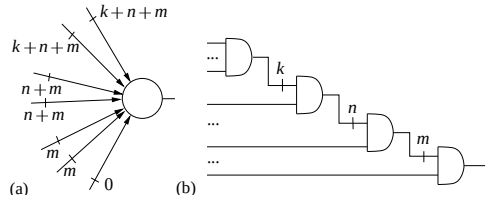

Figure 5: Decomposition of an AND vertex that requires only $w^{max} = k+n+m$ registers: (a) vertex with edges sorted by their weights, (b) corresponding AND tree.

each regular vertex with a fanout greater than two. This dummy vertex is then connected to all fanout vertices and the edge weights are modified in the shown manner. 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 fractional weight is realized by associating a "cost per unit weight" β with each edge, and minimizing a total weighted cost in (2) ($\beta(u,v) = 1/n$). Note that the sum of all edge weights 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 applied for fanin register sharing. If the vertices represent Boolean functions that are totally symmetric, all possible tree configurations establish a valid decomposition of their function. Example for such symmetric functions are AND, OR, or 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 for 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 the 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 as many registers as given by the value of w_r^{max} is illustrated in Figure 5. For a given set of retimed register weights at the inputs of an AND vertex, the algorithm first sorts them according to these weights. Next, an AND tree is built using the structure of Figure 5b. For each set of inputs with identical register numbers, 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 register 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 built. 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 by the ILP solver, the 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) gives the edge weights for the original problem derived 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.

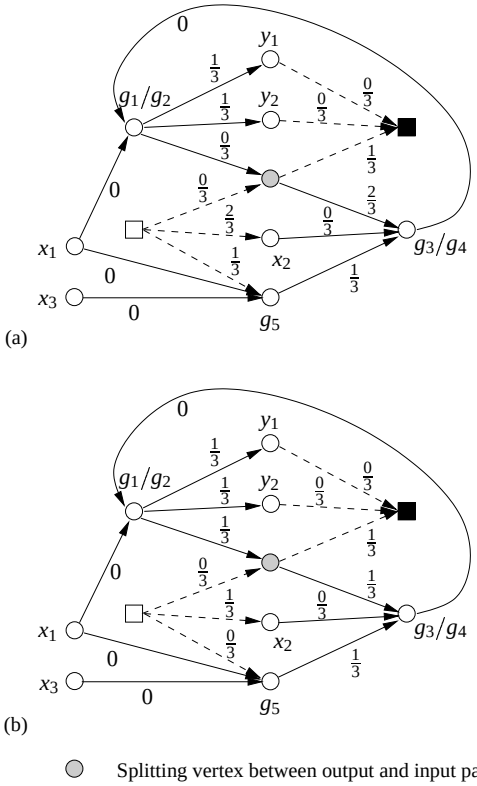

Figure 6: Retiming graph for the circuit of Figure 1b: (a) graph with original weights for 3 registers, (b) optimal solution resulting in 2 registers for the circuit of Figure 1c.

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 Megabytes main memory, running RedHat Linux 6.2. Our implementation is a C-based 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) [16].

In all experiments we used peripheral retiming, which effectively removes as many registers from the inputs and outputs as possible. We focused on the evaluation of combinational simplification and retiming to reduce 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*. Details of this technique can be found in [9]. The resulting retiming stump for the evaluated circuits are generally small and do not constitute a bottleneck in the overall verification scheme. This is mainly due to the fact that even for multi-frame initialization structures, large parts get 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. Column 1 and 2 give the name of the circuits and the size for their initial, unretimed netlists, 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 according to [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.

In Columns 6 through 8 we report results for an iterated application of retiming interleaved with combinational restructuring. For the latter we utilized a combinational simplification engine as described in [14]. We iterated between both engines until no further improvement was gained and reported the best results. Column 6 gives 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 (just fanin register sharing) or 5 (fanin register sharing with on-the-fly retiming). We preferred to apply the register count for comparison, since minimizing AND count does not always minimize the number of register, as will be discussed later in this section. Column 8 gives the required computing resources for the best run between Columns 6 and 7. In Column 9 we provide previously published results. As shown, for all but two circuits for which other results were available, our technique gives lower register counts. Despite detailed analysis, we could not reproduce the results reported in [10] for circuits S344 and S349.

Table 2 gives the results of a set of identical experiments for various IBM Gigahertz Processor (GP) circuits. There are several noteworthy trends in both tables. First, as expected, plain retiming decreases register count by an average of 16.8% on the ISCAS circuits, and by 50.1% on the GP circuits. The presented fanin register sharing gives an additional reduction of the register count by and average of 0.6% and 5.3% for the ISCAS and GP circuits. More importantly, this technique provides a significant reduction of the AND count by 17.0% and 31.6% in the two cases.

In comparison to the application of plain fanin sharing, the additional use of on-the-fly retiming gives mixed results. Our experiments show that on average it hurts both register count and AND count. However, in many individual cases on-the-fly retiming can provide a substantial benefit. For example, it further reduces the AND count on 7 out of the 42 ISCAS circuits, and 9 out of the 28 GP circuits. It also further reduces register count on 4 GP circuits. These reductions may be quite dramatic. For example, circuit I_IFPF has its AND and register count reduced by an additional 23.6% and 15.0%, respectively. Also, as illustrated in Figure 1, this technique alone may provide retiming reductions even without solving the retiming linear program. The GP circuit L_FLUSH is a reconvergent feed-forward pipelined circuit. Before calling the ILP solver to calculate an optimal retiming, the options for Columns 3 and 4 reduce the register count to 78 and 38, respectively.

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 outedges from u have a weight of one, and those of v have a weight of zero. It may be that by backwards-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 outedges, thereby hurting retiming results.

As expected, iteration of combinational simplification and retiming can provide dramatic further reductions; an additional 2.2% and 8.4% on ISCAS, and 6.5% and 17.7% on GP, for AND vertex and register decrease, respectively. These reductions were achieves within up to four iterations on the ISCAS circuits (with an average number of iterations for the optimal solution of 2.7), and up to five iterations on the GP circuits (with an average of 3.0). The reported results in Column 7 applied on-the-fly retiming on 15 of the 42 ISCAS circuits and on 13 of the 28 GP circuits. A 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 power and poten-

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]/[17] (Number of Registers)
					Plain retiming	Best result of columns 4 or 5	CPU time (sec) Memory (MB)	
PROLOG	853 / 136	853 / 45	618 / 45	612 / 47	696 / 45	573 / 45	0.6 / 5.9	- / -
S1196	480 / 18	480 / 16	425 / 16	425 / 16	423 / 14	364 / 14	0.6 / 8.9	16 / -
S1238	533 / 18	533 / 16	474 / 16	474 / 16	469 / 14	418 / 14	0.6 / 11.1	17 / -
S1269	478 / 37	478 / 36	427 / 36	427 / 36	445 / 36	399 / 36	0.6 / 12.1	- / -
S13207_1	3205 / 638	3205 / 389	2378 / 390	2510 / 415	1277 / 266	1164 / 267	4.1 / 28.3	- / -
S1423	507 / 74	507 / 72	399 / 72	407 / 73	461 / 72	395 / 72	0.6 / 7.9	72 / 74
S1488	734 / 6	734 / 6	589 / 6	608 / 6	582 / 6	515 / 6	0.4 / 5.8	- / -
S1494	746 / 6	746 / 6	601 / 6	620 / 6	626 / 6	555 / 6	0.5 / 6.8	- / -
S1512	484 / 57	484 / 57	448 / 57	449 / 57	470 / 57	449 / 57	0.4 / 6.8	- / 57
S15850_1	3852 / 534	3852 / 495	3288 / 497	3361 / 514	2962 / 446	2779 / 460	4.5 / 25.1	- / -
S208_1	77 / 8	77 / 8	61 / 8	63 / 8	70 / 8	61 / 8	0.2 / 6.6	- / -
S27	8 / 3	8 / 3	7 / 3	7 / 3	8 / 3	7 / 3	0.2 / 4.6	- / -
S298	125 / 14	125 / 14	86 / 14	90 / 15	100 / 14	79 / 14	0.3 / 8.8	- / -
S3271	1125 / 116	1125 / 110	1042 / 110	1090 / 111	1051 / 109	995 / 109	1.3 / 11.1	- / 116
S3330	820 / 132	820 / 45	595 / 45	603 / 47	678 / 45	567 / 45	0.5 / 5.8	- / -
S3384	1070 / 183	1070 / 72	1056 / 72	1057 / 72	1060 / 72	1038 / 72	1.1 / 6.8	- / 147
S344	109 / 15	109 / 15	100 / 15	101 / 15	98 / 15	93 / 15	0.3 / 8.8	7 / -
S349	112 / 15	112 / 15	100 / 15	103 / 15	98 / 15	93 / 15	0.3 / 8.8	7 / -
S35932	12204 / 1728	12204 / 1728	11948 / 1728	11980 / 1728	9932 / 1728	9932 / 1728	12.0 / 119.3	- / -
S382	148 / 21	148 / 15	119 / 15	133 / 18	140 / 15	119 / 15	0.5 / 5.8	15 / -
S38584_1	13479 / 1426	13479 / 1416	11065 / 1375	11066 / 1415	11501 / 1372	10584 / 1374	28.3 / 137.9	- / -
S386	188 / 6	188 / 6	114 / 6	120 / 8	166 / 6	113 / 6	0.3 / 6.7	- / -
S400	158 / 21	158 / 15	123 / 15	140 / 18	148 / 15	123 / 15	0.2 / 4.5	15 / -
S420_1	165 / 16	165 / 16	141 / 16	143 / 16	156 / 16	137 / 16	0.3 / 6.7	- / -
S444	169 / 21	169 / 15	142 / 15	143 / 16	155 / 15	141 / 15	0.2 / 4.5	15 / -
S4863	1750 / 104	1750 / 72	1537 / 37	1528 / 37	1333 / 37	1249 / 37	2.0 / 14.8	- / 96
S499	187 / 22	187 / 22	168 / 22	146 / 22	187 / 22	134 / 20	0.4 / 8.9	- / -
S510	213 / 6	213 / 6	195 / 6	200 / 6	205 / 6	177 / 6	0.3 / 6.7	- / -
S526N	251 / 21	251 / 21	170 / 21	177 / 22	200 / 21	164 / 21	0.2 / 4.6	- / -
S5378	1422 / 179	1422 / 115	1242 / 114	1231 / 129	990 / 112	861 / 113	1.5 / 15.7	- / 144
S635	190 / 32	190 / 32	187 / 32	188 / 32	103 / 32	69 / 32	0.2 / 4.5	- / -
S641	160 / 19	160 / 15	113 / 16	113 / 16	146 / 15	115 / 15	0.4 / 6.7	18 / -
S6669	2263 / 239	2263 / 92	2211 / 92	2208 / 92	2196 / 75	2145 / 75	2.4 / 8.5	- / -
S713	174 / 19	174 / 15	121 / 16	121 / 16	149 / 15	115 / 15	0.4 / 8.9	- / -
S820	468 / 5	468 / 5	275 / 5	291 / 5	328 / 5	251 / 5	0.4 / 8.9	- / -
S832	482 / 5	482 / 5	284 / 5	300 / 5	338 / 5	259 / 5	0.4 / 8.9	- / -
S838_1	341 / 32	341 / 32	301 / 32	303 / 32	328 / 32	285 / 32	1.4 / 10.4	- / -
S9234_1	2346 / 211	2346 / 172	1715 / 173	1777 / 179	1497 / 154	1196 / 145	2.4 / 17.1	- / -
S938	341 / 32	341 / 32	301 / 32	303 / 32	328 / 32	285 / 32	0.4 / 6.8	- / -
S953	348 / 29	348 / 6	313 / 6	306 / 7	334 / 6	294 / 6	0.5 / 8.9	- / -
S967	369 / 29	369 / 6	340 / 6	331 / 7	336 / 6	303 / 6	0.4 / 6.7	- / -
S991	299 / 19	299 / 19	283 / 19	283 / 19	297 / 19	283 / 19	0.3 / 4.6	- / -
% Reduction	**0.0 / 0.0**	**0.0 / 16.8**	**17.0 / 17.4**	**15.8 / 14.8**	**14.2 / 19.3**	**25.4 / 19.6**		

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

tial of the presented approach for applications in functional verification and technology independent logic synthesis.

8 Conclusions and Future Work

In this paper we presented two enhancements for min-area retiming that are capable of significantly reducing the 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 circuit parts without a significant computing overhead. The concept of fanin register sharing defers the decomposition of large 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.

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 in 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 focuses on improvements of the combined modeling of retiming and combinational opti-

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	1961 / 491	1955 / 503	2050 / 489	1809 / 489	4.0 / 24.0
CORE_RAS	2297 / 431	2297 / 379	1891 / 370	2024 / 406	1723 / 340	1718 / 348	1.6 / 11.9
D_DASA	1223 / 115	1223 / 100	867 / 100	870 / 101	827 / 100	702 / 100	1.1 / 11.5
D_DCLA	10916 / 1137	10916 / 771	8177 / 750	10144 / 771	7683 / 750	7582 / 750	26.0 / 57.1
D_DUDD	1295 / 129	1295 / 100	1064 / 100	1070 / 100	1042 / 100	949 / 100	0.7 / 7.0
I_IBBC	388 / 195	388 / 42	190 / 40	200 / 39	203 / 42	166 / 36	0.5 / 8.0
I_IFAR	1202 / 413	1202 / 147	888 / 140	923 / 148	911 / 135	791 / 137	1.6 / 15.9
I_IFEC	334 / 182	334 / 46	244 / 45	259 / 46	283 / 46	216 / 45	0.6 / 9.1
I_IFPF	5896 / 1546	5896 / 705	4884 / 690	3491 / 458	2779 / 352	2571 / 353	451.9 / 66.6
L_EMQ	981 / 220	981 / 88	677 / 86	748 / 88	818 / 88	570 / 69	1.0 / 9.3
L_EXEC	1618 / 535	1618 / 168	963 / 163	1074 / 197	914 / 136	693 / 118	2.5 / 15.3
L_FLUSH	893 / 159	893 / 5	433 / 1	360 / 1	332 / 1	282 / 1	0.6 / 8.0
L_LMQ	14074 / 1876	14074 / 1196	11224 / 1193	11552 / 1205	5321 / 429	4067 / 241	102.4 / 95.0
L_LRU	581 / 237	581 / 94	436 / 94	432 / 94	465 / 94	394 / 94	1.2 / 11.5
L_PNTR	1453 / 541	1453 / 245	1217 / 245	1240 / 245	1349 / 245	1169 / 245	2.6 / 19.6
L_TBWK	1160 / 307	1160 / 125	754 / 125	637 / 119	274 / 40	146 / 40	1.0 / 9.4
M_CIU	4550 / 777	4550 / 459	3016 / 415	3078 / 436	2650 / 381	2305 / 349	5.4 / 46.7
S_SCU1	1520 / 373	1520 / 212	1076 / 200	1089 / 202	1169 / 190	938 / 184	3.6 / 13.1
S_SCU2	8560 / 1368	8560 / 640	5640 / 563	5115 / 575	3862 / 432	3178 / 411	105.3 / 61.1
V_CACH	753 / 173	753 / 103	435 / 97	558 / 113	381 / 94	347 / 97	0.8 / 14.5
V_DIR	554 / 178	554 / 87	345 / 84	243 / 49	145 / 45	124 / 43	0.6 / 11.2
V_L2FB	120 / 75	120 / 26	96 / 26	96 / 26	85 / 26	63 / 26	0.3 / 6.7
V_SCR1	826 / 150	826 / 95	392 / 52	393 / 52	329 / 49	298 / 48	0.6 / 9.0
V_SCR2	2563 / 551	2563 / 458	1084 / 86	1061 / 89	511 / 82	479 / 82	1.4 / 13.9
V_SNPC	78 / 93	78 / 21	64 / 21	64 / 21	54 / 21	42 / 21	0.3 / 8.9
V_SNPM	2421 / 1421	2421 / 241	1672 / 237	1671 / 237	1692 / 220	1080 / 179	18.6 / 42.1
W_GAR	2107 / 242	2107 / 93	1388 / 81	1549 / 121	1731 / 84	1255 / 75	1.7 / 8.3
W_SFA	471 / 64	471 / 42	291 / 42	291 / 42	311 / 41	260 / 41	0.5 / 9.0
% Reduction	**0.0 / 0.0**	**0.0 / 50.1**	**31.6 / 55.4**	**31.3 / 54.8**	**38.7 / 60.6**	**49.3 / 61.9**	

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

mization. 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), 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] 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), ACM/IEEE, June 2001.

[15] 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.

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

[17] 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.