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 enhancing min-area retiming. First, we discuss an on-the-fly retiming approach based on a sequential AND/INVERTER/REGISTER *graph. This technique sequentially compacts the circuit structure 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. Both techniques are aimed at optimizing the performance of reachability-based verification methods. 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, minimizing the longest combinational delay, 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 [3, 4, 5].

In this paper we present two 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 (AIR) 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 [6], 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.

The second technique is based on the idea that, similar to fanout register sharing, fanin registers of an AND cluster can be optimally shared by adjusting the AND decomposition. This adjustment can reduce the register count to the maximum number of registers on any incoming cluster edge. We describe a corresponding extension of the retiming formulation based upon the AIR graph which models fanin register sharing similar to the Leiserson and Saxe model of 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, providing 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 [7] a technique is presented that simultaneously considers multiple structures for possible logic implementations using a choice node. This method is aimed at technology mapping and, despite its recursive capability, must explicitly generate candidate structures for an AND cluster decomposition including possible retiming configurations. Our approach defers the actual decomposition step until after the optimal retiming is computed. In [8] the concept of algebraic factorization is extended to sequential expressions, which implicitly intertwines retiming with structural rewriting. 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.

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 straightforward and hence are not discussed in this paper.

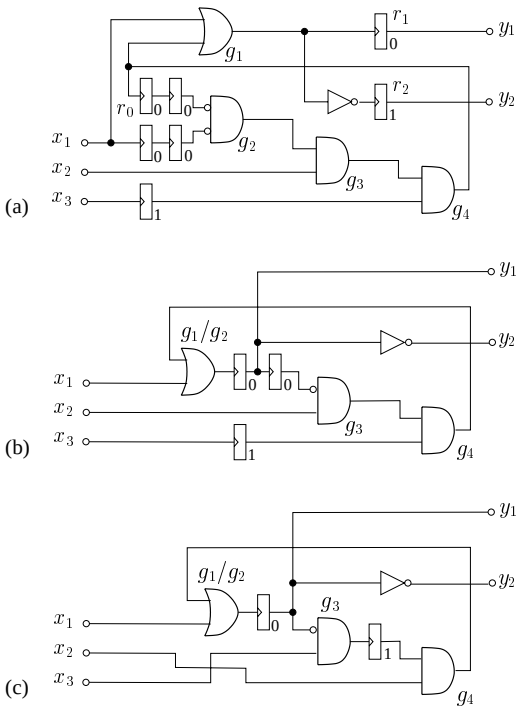

Figure 1: Example for the application of retiming: (a) original circuit, (b) circuit after on-the-fly retiming, (c) circuit after retiming with AND cluster fanin sharing.

Figure 1 introduces an example circuit to demonstrate the two presented techniques. As shown in part (a), the original circuit contains seven registers. The idea of on-the-fly retiming is derived from the concept of on-the-fly compaction of combinational circuits [6]. 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 r_0. As shown, the four registers in front of gate g_2 and two input inverters have been dragged through the gate. This allows gate g_2 to merge with g_1, and further allows the sharing of registers r_1, r_2, and r_0. No further forward retiming is possible since edge (x_2, g_3) has no registers to drag past gate g_3. Note that this particular result is identical to an optimal retiming computed by the standard Leiserson and Saxe min-area retiming algorithm.

Figure 1c shows a functionally equivalent version of the circuit that contains only two registers instead of three. Here the two registers in front of the AND cluster g_3/g_4 have been merged. 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 for 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.

We discuss the on-the-fly retiming approach and the new retiming formulation that models optimal register sharing for AND clusters in Sections 3 and 4, respectively.

3 On-the-fly Retiming

We apply on-the-fly retiming during construction of an AIR graph, where the vertices represent AND functions which have one or more incoming edges. Each edge holds three attributes – the number of registers along that edge, the initial values of those registers, and a flag to indicate whether the referred vertex function is complemented. The function represented by two edges are sequentially equivalent if they have: (1) the same source vertex, (2) identical complementation attributes, and (3) the same number of registers with matching initial values. For quick equivalence checking, the edge representation uses a 64-bit canonical word to represent these three items. 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 values, respectively. Lastly, a single bit is used to indicate edge complementation. Using this data structure, a simple comparison of two words can decide whether two edges are functionally equivalent.

The algorithm for constructing an AND gate for the AIR graph is given in Figure 2. The graph construction starts at the primary inputs and an arbitrary set of register cuts of the cyclic circuitry. For each register at a 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 [6]. 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. 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 AIR graph for the circuit of Figure 1b

```
/* Create_And takes two operand edges p1 and
   p2, and returns an edge representing the
   AND of p1 and p2                              */
Algorithm Create_And(p1, p2) {
   if (p1 == const_0)    return const_0;
   if (p2 == const_0)    return const_0;
   if (p1 == const_1)    return p2;
   if (p2 == const_1)    return p1;
   if (p1 == p2)         return p1;
   if (p1 == !p2)        return const_0;

   /* Drag as many registers as possible
      from both edges and store them in r  */
   n  = Min(Num_Regs(p1), Num_Regs(p2));
   q1,r1 = Truncate_Registers(p1, n);
   q2,r2 = Truncate_Registers(p2, n);
   /* AND the initial states               */
   r  = And_Initial_States(r1, r2);

   /* Apply ranking to catch commutativity */
   if (Rank(q1) > Rank(q2)) Swap(q1, q2);
   /* Hash lookup for vertex with q1 and q2 */
   q = Hash_Lookup(q1, q2);
   if (q == NULL) {
   /* Allocate new vertex if lookup failed
      and add to hash table                */
      q = Create_And_Vertex(q1, q2);
   }
   /* Add back stripped registers          */
   return q + r;
}
```

Figure 2: Pseudo-code for constructing an AND vertex for the AIR graph.

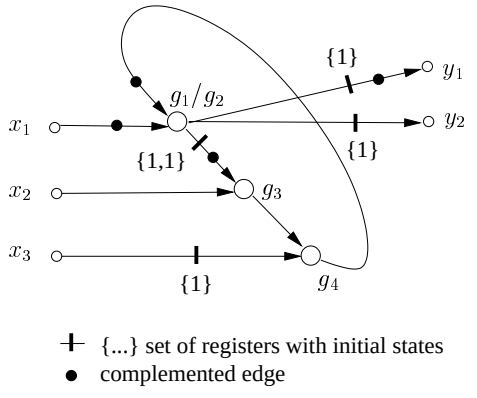

Figure 3: AIR graph for the circuit of Figure 1b.

is shown in Figure 3. The graph was constructed starting from the inputs and a cut at register r_0.

4 Optimal Sharing of Fanin Registers

Figure 4a shows a retiming graph that was annotated for fanout register sharing as described in [2]. The idea is to add

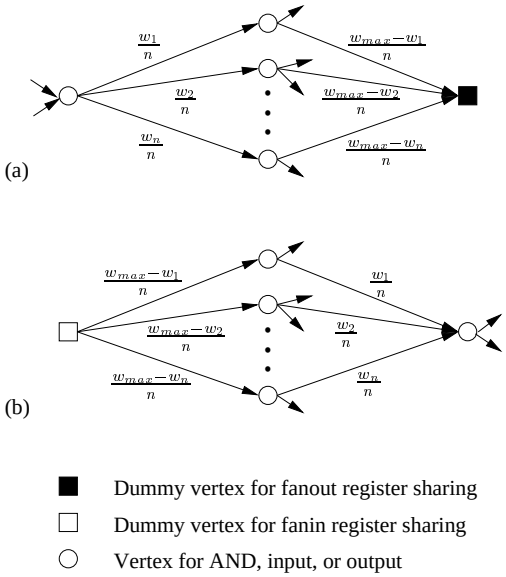

Figure 4: Sharing of fanout and fanin registers: (a) original idea of fanout register sharing [2], (b) extension to fanin register sharing.

a dummy sink vertex for each regular vertex with a fanout degree greater than one. This dummy vertex is then connected to all fanout vertices, and edge weights are assigned as shown (the division is handled by scaling-down a "cost-per-unit-weight" edge attribute to $1/n$). Such modeling results in a formulation that minimizes the maximum number of registers at any of the fanout edges. This reflects the fact that all fanout edges from a given vertex can share their registers with those along the maximum-weight edge.

Figure 4b shows how this idea can be adapted to fanin register sharing. Similar to the previous case, a dummy vertex for fanin register sharing is created. 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 cluster can be straight-forwardly decomposed to require only this "max-inedge-weight" number of registers. Note that fanin sharing and fanout sharing are in general orthogonal optimizations, and concurrent use of both can yield optimizations beyond that possible with either approach alone.

This decomposition is illustrated in Figure 5. For a given set of inputs of an AND cluster, the algorithm first sorts them by their retimed weight. Next, an AND tree is built using the structure of Figure 5b. For each set of inputs with an identical number of registers, a balanced AND subtree is constructed. The individual subtrees are then connected by registers in a linear sequence. The total number of resulting registers on the edges between the subtrees is equal to the difference of their register weights.

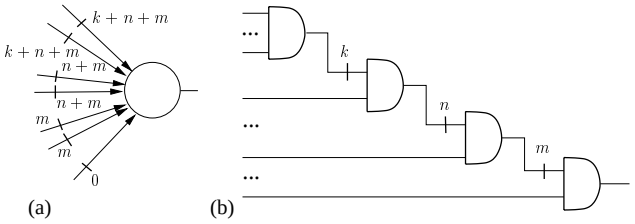

Figure 5: Decomposition of AND clusters for maximal register sharing: (a) vertex with inedges sorted by weight, (b) corresponding AND tree.

For maximum fanin sharing, the AIR graph produced by the on-the-fly retiming algorithm is first restructured to maximize the individual AND clusters. Next a retiming graph with the dummy vertices for fanin and fanout sharing is built. A splitting vertex is introduced onto edges involved in simultaneous fanin and fanout sharing. Once an optimal retiming is calculated 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 1b. Part (b) shows the resulting weights from the ILP solver which corresponds to the optimal circuit of Figure 1c.

5 Experiments

In this section we provide experimental results for retiming with the various techniques. All experiments were run on an IBM ThinkPad Model T21, with an 800Mhz PIII and 256 MBytes main memory, running the Linux operating system. Our implementation is a C-based retiming engine, utilizing the data structure and algorithms described in this paper. As ILP solver we used the primal network simplex algorithm from IBM's Optimization Solutions Library [9].

Since we apply the presented approach for enhanced verification, all experiments use peripheral retiming that removes as many registers as possible from the inputs and outputs. More details of this technique can be found in [5].

Table 1 provides the results for various retiming experiments for the ISCAS89 benchmarks. Table 2 gives the results for the same set of experiments for selected IBM Gigahertz Processor (GP) circuits. For each option, we report the number of 2-input AND vertices and registers. Column 1 and 2 give the name of the circuits and their initial, unretimed sizes, respectively. Column 3 provides the circuit sizes for plain retiming as in [2] without the application of on-the-fly retiming or fanin-register sharing. In column 4 we give the results for fanin-register sharing without on-the-fly retiming, whereas for the results of column 5 we enabled both fanin-register sharing as well as on-the-fly retiming.

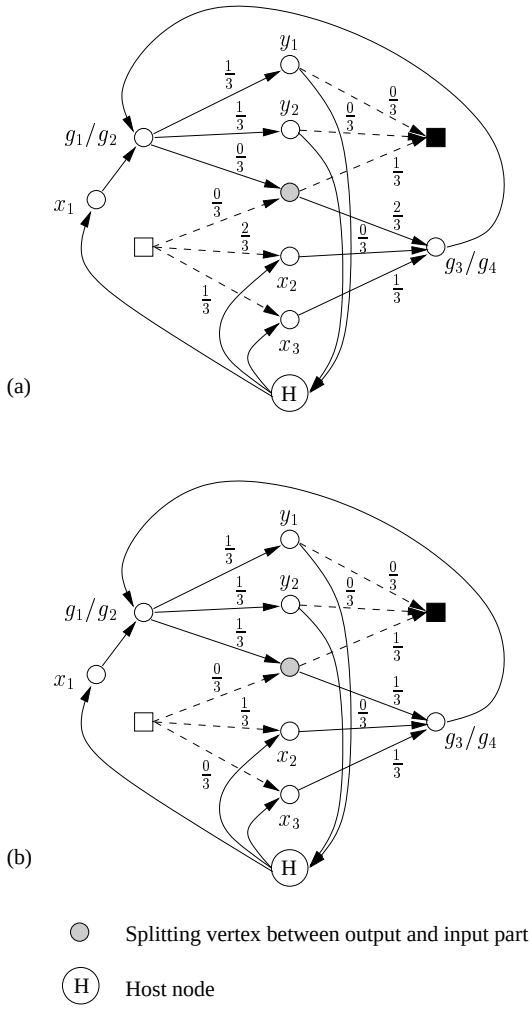

Figure 6: Retiming graph for the circuit of Figure 1: (a) graph for Figure 1b containing 3 registers, (b) optimal weight soultion of Figure 1c containing 2 registers.

In columns 6 and 7 we report results for an iterated application of retiming interleaved with combinational restructuring. For this we utilized a combinational simplification engine as described in [10]. We iterated between both engines until no further improvement was gained and reported the best results. Column 6 gives these results using plain retiming identical to the option used in column 3. Column 7 reports the best results of the iterations using the options of columns 4 or 5. For selecting the best run, we used the number of registers instead of the number of AND vertices. In column 8 we provide the required computing resources for the best result of the prior columns.

There are several noteworthy trends in the above tables. First, as expected, retiming alone can significantly decrease the register count. We obtained an average reduction of

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)
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
S1238	533 / 18	533 / 16	532 / 16	532 / 16	518 / 16	513 / 16	0.5 / 6.5
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
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
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
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
S344	109 / 15	109 / 15	102 / 15	102 / 15	101 / 15	98 / 15	0.2 / 2.3
S349	112 / 15	112 / 15	104 / 15	104 / 15	101 / 15	98 / 15	0.2 / 2.3
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
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
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
S4863	1750 / 104	1750 / 72	1537 / 37	1537 / 37	1376 / 37	1326 / 37	2.4 / 17.3
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
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
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).

16.8% on the ISCAS benchmarks, and 50.1% on the GP circuits. Fanin-register sharing combined with vertex hashing further reduces that count by 0.9% for ISCAS and 4.7% for GP. 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 11 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.

As illustrated in Figure 1, on-the-fly retiming alone may result in a register reduction even without solving the retiming problem. For example, the GP circuit L_FLUSH is a reconvergent feed-forward pipelined circuit. 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, in the majority of the cases, on-the-fly retiming temporarily hurts register count, which then gets rectified during the global retiming phase.

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

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.

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 potential of the presented approaches for applications in functional verification, as well as technology-independent logic synthesis.

6 Conclusions and Future Work

In this paper we present two new enhancements for min-area retiming that are capable of significantly reducing register and gate count by departing from the traditional approach of applying retiming on a static circuit graph. The main focus of this research is to enhance reachability-based verification [5] for which min-area retiming is the single objective. However, these techniques are clearly applicable to any sequential optimization problem. It would be interesting to investigate how these techniques can be adapted for general logic synthesis.

Our future work includes continued efforts to combine the modeling of retiming and synthesis. One research direction is to apply fanin-register sharing to a larger fragment of logic cones, such as XOR clusters. We further investigate adding more sophisticated combinational optimization techniques to our combinational restructuring engine.

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] 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.

[4] S. Hassoun and C. Ebeling, "Experiments in the iterative application of resynthesis and retiming," in *International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, ACM/IEEE, December 1997.

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

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

[7] 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.

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

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

[10] 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.