Performance-driven Integration of Retiming and Resynthesis*

Peichen Pan
Strategic CAD Labs
Intel Corporation, Hillsboro, OR 97124

Abstract – We present a novel approach to performance optimization by integrating retiming and resynthesis. The approach is oblivious of register boundaries during resynthesis. In addition, it guides resynthesis by a criterion that is directly tied to the performance target. The proposed approach obtains provable results. Experimental results further demonstrate the effectiveness of our approach.

1 Introduction

Over the years, combinational timing optimization has been intensively studied and a significant level of maturity has been attained [3, 11, 12]. In comparison, sequential timing optimization has been lagging behind, due to the additional complexity of handling registers.

Traditionally, sequential circuits are viewed as a special case of combinational circuits and only the logic between registers is optimized. This approach not only results in stringent resynthesis constraints, but also does not permit interaction between logic separated by registers. Retiming [7] is another technique for sequential optimization. Although it is very useful, the effectiveness of retiming is limited since it does not change the logic.

Efforts have been made to improve upon the direct application of combinational techniques to sequential circuits. Techniques were proposed that take into consideration the existence of post-resynthesis retiming during resynthesis by generating a set of relaxed resynthesis constraints [5, 2]. In another direction, techniques were proposed to exploit signal dependencies across register boundaries. The approach proposed in [9] first retimes the circuit with the objective to expose signal dependencies across register boundaries. Then, it carries out resynthesis on the logic between registers in the retimed circuit. These techniques can be improved by repeating the retiming/resynthesis loop [6]. Attempts have also been made to generalize combinational logic synthesis techniques to sequential circuits [1, 4, 8].

Although these approaches have achieved some degree of success, it is evident that a true integration of retiming and resynthesis is still lacking since retiming and resynthesis are carried out separately. Even when retiming and resynthesis are tightly integrated, no effective criteria have been provided to guide the application of these techniques. As a result, local logic transformations are not strongly tied to the performance target.

In this paper, we propose a new approach to integrate retiming and resynthesis for performance optimization. The approach is based on two recent concepts: *expanded circuits* and *l-values* which were originally proposed in the context of FPGA technology mapping [10]. Our approach produces provably good results under a very general assumption.

The rest of this paper is organized as follows. An overview of the preliminary concepts underlying our approach is presented in Section 2. In Section 3, we introduce our approach. Section 4 deals with the experimental results obtained. Section 5 concludes the paper.

2 Preliminaries

A sequential circuit is represented as a directed graph. Each node denotes either a primary input (PI), a primary output (PO) or a gate, and each edge $u \xrightarrow{e} v$ represents a connection from node u to node v. Each edge e is weighted by its number of registers, $w(e)$. Each node has an area and a delay associated with it. The *cycle time* of a sequential circuit is the maximum delay on the combinational paths.

Retiming [7] is a transformation that repositions the registers in a circuit without altering its functionality. Retiming a node by a value i is the operation of removing i registers from each fan-out edge and adding i registers to each fan-in edge of the node. In general, all nodes can be retimed collectively to arrive at a retiming of the circuit.

To exploit the flexibility of dynamic register positions due to retiming, we make use of *expanded circuits* [10]. The expanded circuit at a node is formed by *unrolling* the circuit over all time frames, starting from the node. It is essentially the combinational logic for the node in the circuit. For example, for the circuit in Fig. 1(1), Fig. 1(2) is a sketch of the expanded circuit at g. The index of each node is the number of backward time frames (registers) relative to the output node g^0. Expanded circuits has the following property [10].

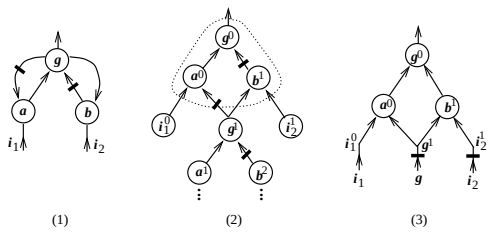

Figure 1: Expanded circuit and resynthesis cone.

*This work was done while the author was with Clarkson University, Potsdam, NY.

DAC 99, New Orleans, Louisiana

Theorem 1 *Let u^t be a node in the expanded circuit at node v, any path from u^t to the output v^0 has exactly t registers.*

If we take an output cone of the expanded circuit at a node, the registers within the cone can be retimed out of the cone to form a combinational subcircuit for the node. Thus, the concept of expanded circuits is a systematic way to extract logic across register boundaries. For example, from the cone indicated in the expanded circuit in Fig. 1(2) we have the combinational subcircuit for g in Fig. 1(3). We will refer the combinational subcircuits derived from expanded circuits as *resynthesis cones* as the proposed approach carries out resynthesis on them.

We will employ the concept of *l-values* introduced in [10] to guide resynthesis. The l-values are defined for a given target cycle time ϕ. The l-value of a node v in a circuit is defined as the *maximum* weight of the paths from the PIs to v according to a set of new edge weights defined as follows. For each edge $u \stackrel{e}{\rightarrow} v$, the new weight of e, $w_1(e) = -\phi \cdot w(e) + d(v)$, where $d(v)$ is the delay of node v. The l-values can be used to predict whether the target cycle time ϕ can be attained by retiming, without actually relocating registers [10].

Theorem 2 *If there is a PO whose l-value is greater than ϕ, the circuit cannot be retimed to a cycle time of ϕ. On the other hand, if the l-values of all POs are less than or equal to ϕ, the circuit can be retimed to a cycle time less than $\phi + D$, where D is the largest gate delay.*

3 The proposed approach

The basic idea in our approach is to extract resynthesis cones and resynthesize them using a combinational timing optimizer. Because of Theorem 2, instead of considering cycle time directly, we want to resynthesize a circuit so that the l-values of all POs are less than or equal to ϕ. The following is a precise definition of the problem:

Problem 1 *Given a combinational timing optimizer $\mathcal{T}$ and a set of cones $\mathcal{C}(v)$ for each node v, resynthesize one cone in $\mathcal{C}(v)$ for each v such that the l-values of all POs in the equivalent circuit formed by the resynthesized cones are less than or equal to ϕ.*

We now present an algorithm for this problem. The algorithm has two phases: a *labeling phase* and an *assembly phase*. In the labeling phase, we compute a label and an associated resynthesized cone for each node. Based on the label we know whether there is a solution to Problem 1. If the answer is affirmative, we connect the resynthesized cones together in the assembly phase to form a solution to the problem.

In the labeling phase, we want to find the minimum l-value that can be obtained for each node with resynthesis. We determine the minimum l-values by iterative improvement. For each node v, we maintain a label $l(v)$, which is a lower bound on the minimum l-value at v, and then successively approximate the minimum l-value by updating it. We begin by initializing the labels of the non-PI nodes to $-\infty$. The labels of the PIs are set to zero assuming all input signals arrive at the same clock edge. As the process of resynthesis continues, the labels are gradually increased. If the label of any PO ever exceeds ϕ, the labeling procedure simply stops and returns FAILURE as there is no solution to Problem 1, based on the selected resynthesis cones and the combinational timing optimizer. Fig. 2 shows the outline of the labeling procedure where `update` is the procedure that updates the label at each node.

```
ReRe(G, φ)        // G is the circuit
    for each node v in G do
        if v is a PI then l(v) ← 0;
        else l(v) ← −∞;
    while (labels changed) do
        for each non-PI node v in G do
            l_tmp ← update(v)
            if l_tmp > l(v) then
                l(v) ← l_tmp;
            if v is a PO and l(v) > φ
                then return FAILURE
    return SUCCESS;
```

Figure 2: The labeling procedure.

We now discuss how to update the label at each node. To maintain $l(v)$ as a lower bound on the minimum l-value at v, we resynthesize the cones in $\mathcal{C}(v)$ in such a way that the updated label is minimized. Suppose we resynthesize $c \in \mathcal{C}(v)$ and let c' be the resulting cone. If we use c' as the logic for generating (the output signal of) v, then by the definition of l-values, the l-value at v is at least as follows:

$$\max\{l(u) - \phi \cdot t + d_{u,t} \mid u^t \text{ is an input to } c'\},$$

where $d_{u,t}$ is the maximum path delay from u^t to the output v in c'. Now suppose that we set the arrival time at u^t to $l(u) - \phi \cdot t$ as shown in Fig. 3. It is easy to see that the above quantity is exactly the arrival time at the output v in c'.

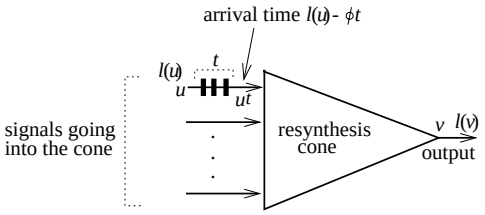

Figure 3: Setting arrival times in a cone.

Consequently, we constrain the resynthesis of each cone in $\mathcal{C}(v)$ by assigning appropriate arrival times at the inputs of the cone as indicated in Fig. 3. Then we resynthesize the cone to minimize the arrival time at the output of the cone. Therefore, we translate the problem of determining the new lower bound to that of resynthesizing each cone to minimize the arrival time at the output. Among all cones in $\mathcal{C}(v)$, we pick the one that has minimum arrival time after resynthesis in order to minimize the new label at v. Let $\mathcal{T}(c)$ denote the arrival time at the output v in c after resynthesis (with appropriate arrival times at the inputs) using $\mathcal{T}$. Then, $\texttt{update}(v) = \min_{c \in \mathcal{C}(v)} \mathcal{T}(c)$.

We use an example to illustrate the labeling procedure. Consider the circuit in Fig. 4(1) which has a cycle time of

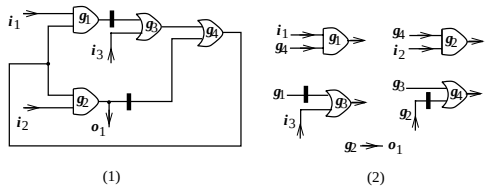

Figure 4: A circuit and selected resynthesis cones.

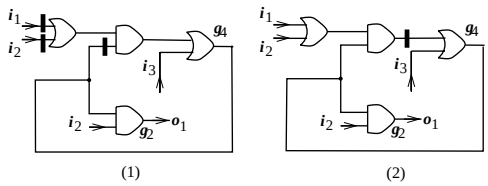

Figure 6: Resynthesized circuits.

three units, assuming that each gate has one unit of delay. We want to resynthesize the circuit with a target cycle time $\phi = 2$. Suppose that each node has the *trivial cone* formed just by itself as shown in Fig. 4(2). (Note that the presence of trivial represents that we may choose not to to resynthesize at the node.) Suppose that node g_4 has the two additional resynthesis cones shown in Fig. 5(1) and (2). To simplify our discussion, we assume that the combinational timing optimizer can only produce the resynthesized cones shown in Fig. 5(3) and (4) for the cones in Fig. 5(1) and (2), respectively, regardless of the arrival times at their inputs.

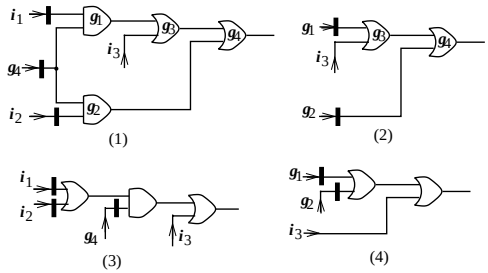

Figure 5: More resynthesis cones.

At the beginning, $l(i_1) = l(i_2) = l(i_3) = 0$ and $l(v) = -\infty$ for all other nodes. Suppose we visit the nodes g_1, g_2, g_3, g_4 in this order in the labeling procedure. In the first iteration, since g_1 only has the trivial cone, we have $l(g_1) = \max\{l(i_1)+1, l(g_4)+1\} = \max\{0+1, -\infty+1\} = 1$. Similarly, $l(g_2) = 1$ and $l(g_3) = 1$. For node g_4, the arrival time from the trivial cone is $\max\{l(g_3)+1, l(g_2)-\phi+1\} = 2$; from the cone in Fig. 5(3) the arrival time is $\max\{l(i_1)-\phi+3, l(i_2)-\phi+3, l(g_4)-\phi+2, l(i_3)+1\} = 1$; from the cone in Fig. 5(4) the arrival time is $\max\{l(g_1)-\phi+2, l(g_2)-\phi+2, l(i_3)+1\} = 1$. Thus, $l(g_4) = 1$. For the output node $l(o_1) = l(g_2) = 1$ from the trivial cone at o_1.

In the second iteration, since $l(g_4) = 1$, we have $l(g_1) = 2$, $l(g_2) = 2$, and $l(g_3) = 1$. For node g_4, the cone in Fig. 5(3) gives the smallest arrival time 1, so $l(g_4) = 1$. For the output node $l(o_1) = l(g_2) = 2$. Since labels have changed, the procedure goes to the third iteration. However, no more change in labels will occur and the procedure stops by returning SUCCESS.

After the procedure outlined in Fig. 2 terminates with SUCCESS, we proceed to generate the resynthesized circuit. Recall that for each node in the initial circuit, we not only have its label, but also an associated resynthesized cone that realizes the label. In the assembly phase, we simply connect together the resynthesized cones that realize the labels. This is followed by a cleanup step to remove nodes that do not drive, directly or indirectly, the POs of the circuit. For the example in Fig. 4, the resynthesized cone in Fig. 5(3) realizes the label at g_4. For each of the other nodes, its label is realized by its trivial cone. The resulting resynthesized circuit is shown in Fig. 6(1).

Given a single-output combinational circuit, a set of arrival times at its inputs is *pairwise smaller* than another set if the arrival time at each input in the set is less than or equal to the corresponding arrival time in the other set. A combinational timing optimizer is *order-respecting* if a pairwise smaller set of arrival times at the inputs results in a resynthesized circuit with a smaller or the same arrival time at the output.

Theorem 3 *Assuming that $\mathcal{T}$ is order-respecting, there is a solution to Problem 1 iff the procedure returns SUCCESS.*

To obtain a circuit with the target cycle time ϕ, we can simply retime the circuit generated in the assembly phase to minimize its cycle time. The resulting circuit is guaranteed to have a cycle time less than ϕ plus a largest gate delay. For the resynthesized circuit in Fig. 6(1), after retiming, we obtain the circuit in Fig. 6(2) which has the desired cycle time of two units.

We introduce a factor called the *depth* to limit the size of the resynthesis cones selected from the expanded circuit for each node. This factor determines the size of the logic of the node that will be passed on to the combinational logic optimizer. A large value results in a large cone, which in turn, presents better resynthesis potential. On the other hand, area overhead and computation time could be large too. An optimal choice balances the overhead and the potential for resynthesis.

Ideally, one would like to consider many cones for each node. In practice, however, resynthesizing several cones at each node may greatly increase the computation time. Our strategy to overcome this problem is to *dynamically* select *one* cone to resynthesize at each node in each iteration during the labeling phase.

In practice, only a small set of nodes constrain the performance of a circuit. Moreover resynthesizing nodes that do not contribute to achieving the target cycle time may result in unnecessary area overhead and computation time. This suggests the need for an effective technique for selecting a few "strategic" nodes for resynthesis. Resynthesis is directed towards this set of nodes first. We designed an effective technique for node selection. The details of the technique is omitted here due to space limitation.

4 Experimental results

This section describes our experimental results on the sequential benchmark circuits in ISCAS89 suite with Addendum93. Our program, referred to as `SeqRe`, is integrated with the logic synthesis tool SIS. Combinational resynthesis is performed by `speed_up` in SIS. A min-cost based node selection technique was also implemented.

`SeqRe` performs repeated calls to `ReRe` outlined in Fig. 2, to target a cycle time that is less than the current one until the cycle time cannot be reduced. Our program introduces a parameter *step*, to control the difference between the current and targeted cycle times. We set *step* to one initially. If we come across a failure to meet the target cycle time, we then increase it to two and attempt once more. The other parameter *depth* that controls the size of the resynthesis cones is set to 4 in our experiments. Throughout the experiment, the unit-delay model is used.

circuit	initial			retiming		SeqRe		
name	gates	FFs	ϕ	FFs	ϕ	gates	FFs	ϕ
s208	81	8	10	8	10	79	15	7
s298	107	14	8	24	5	116	42	4
s344	121	15	14	26	11	132	22	7
s349	124	15	14	27	11	150	35	7
s382	140	21	9	30	6	139	38	5
s386	167	6	10	6	10	189	14	7
s400	147	21	9	30	6	152	37	5
s420	175	16	12	16	12	172	25	9
s444	159	21	10	46	6	175	56	5
s499	209	22	12	22	12	403	109	8
s510	225	6	11	6	11	260	25	8
s526	227	21	8	40	6	231	59	5
s526n	226	21	8	35	6	226	61	5
s635	220	32	35	36	20	267	56	7
s641	169	19	20	19	20	245	27	11
s713	180	19	20	19	20	249	24	11
s820	400	5	11	6	10	406	6	9
s832	410	5	11	6	10	414	6	9
s838	363	32	16	32	16	342	45	11
s938	363	32	16	32	16	342	45	11
s953	371	29	13	31	11	438	87	8
s967	390	29	12	29	11	453	61	8
s991	317	18	40	19	38	462	57	13
s1196	499	18	19	18	19	582	18	13
s1238	552	18	21	18	21	642	18	15
s1269	515	37	28	82	17	624	112	12
s1423	504	74	55	78	49	1004	134	14
s1488	689	6	12	6	12	701	6	11
s1494	698	6	12	6	12	712	6	11
s1512	523	57	19	60	16	626	91	8
s3271	1246	116	21	176	12	1298	235	9
s3330	999	132	20	137	9	1040	136	8
s3384	1221	183	55	190	26	1094	254	9
s4863	1822	104	52	129	28	2006	189	21
s5378	1530	179	16	235	12	1447	262	10
s6669	2455	239	81	306	22	2599	366	16
s9234	2246	211	31	248	20	1390	250	10
prolog	1032	136	20	159	9	1068	126	8
s13207	3210	638	32	492	29	3015	490	22
s15850	4049	534	44	621	33	3817	743	23
s35932	11980	1728	19	1728	19	12905	1909	11
s38417	9783	1636	29	1671	24	9951	1707	20
s38584	14200	1426	30	1669	25	13207	1682	20
total	65044	7905	945	8574	698	65770	9686	451
ratio	0.99	0.82	2.10	0.89	1.55	1	1	1

Table 1: Experimental results.

We report, in Table 1, the results of `SeqRe` on all the benchmark circuits. For each benchmark we list the number of gates, number of registers and the cycle time, of the `initial`, `retimed` and `SeqRe` optimized circuits. It is evident from the table, significant improvement in cycle time is achieved by `SeqRe` over the initial circuits — over 50% reduction overall. From Table 1, we can also see that `SeqRe` achieves much better results than retiming alone. On the whole `SeqRe` reduces the cycle time by an additional 35% (with a 13% increase in the number of registers and a 1% increase in the number of gates) over the optimally retimed circuits. We point out that our current program does not try to minimize the number of registers. We expect the number of registers can be reduced considerably once a register minimization step is added. Comparison with existing retiming and resynthesis methods has been done although not reported here due to space limitation.

5 Conclusions

We have developed a novel timing optimization approach that integrates retiming with resynthesis to form a powerful combined technique. This approach has several important features. Firstly, it extracts combinational logic out of a circuit for resynthesis instead of carrying out resynthesis directly on the circuit (or its retimed one) to expose signal dependencies. By doing so, it becomes truly oblivious of register boundaries. Secondly, it tightly constrains logic resynthesis so that the resynthesized circuit is guaranteed to meet the performance target. Thirdly, it is independent of the combinational timing optimizer. Experimental results show that the proposed approach can improve the performance of a sequential circuit significantly.

References

[1] S. Bommu, M. Ciesielski, N. O'Neill, and P. Kalla. Retiming-based factorization for multi-level logic optimization. In *Intl. Workshop on Logic Synthesis*, 1997.

[2] S. T. Chakradhar, S. Dey, M. Potkonjak, and S. G. Rothweiler. Sequential circuit delay optimization using global path delays. In *ACM/IEEE Design Automation Conf. (DAC)*, pages 483–489, 1993.

[3] K. C. Chen and S. Muroga. Timing optimization for multi-level combinational circuits. In *ACM/IEEE Design Automation Conf. (DAC)*, pages 339–344, 1990.

[4] G. DeMicheli. Synchronous logic synthesis: algorithms for cycle-time minimization. *IEEE Trans. on Computer-Aided Design*, 10:63–73, 1991.

[5] S. Dey, M. Potkonjak, and S. G. Rothweiler. Performance optimization of sequential circuits by eliminating retiming bottlenecks. In *Intl. Conf. on Computer-Aided Design (ICCAD)*, pages 504–509, 1992.

[6] S. Hassoun and C. Ebeling. Experiments in the iterative application of resynthesis and retiming. In *Intl. Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, 1997.

[7] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991.

[8] B. Lin. Restructuring of synchronous logic circuits. In *European Conf. on Design Automation*, pages 205–209, 1993.

[9] S. Malik, K. J. Singh, R. Brayton, and A. L. Sangiovanni-Vincentelli. Performance optimization of pipelined logic circuits using peripheral retiming and resynthesis. *IEEE Trans. on Computer-Aided Design*, 12:568–578, 1993.

[10] P. Pan and C.L. Liu. Optimal clock period FPGA technology mapping for sequential circuits with retiming. *ACM Trans. on Design Automation of Electronic Systems*, 3(3), 1998.

[11] K. J. Singh, A. R. Wang, R. Brayton, and A. L. Sangiovanni-Vincentelli. Timing optimization of combinational logic. In *Intl. Conf. on Computer-Aided Design (ICCAD)*, pages 282–285, 1988.

[12] H. J. Touati, H. Savoj, and R. K. Brayton. Delay optimization of combinational logic circuits by clustering and partial collapsing. In *Intl. Conf. on Computer-Aided Design (ICCAD)*, pages 188–191, 1991.