Continuous Retiming: Algorithms and Applications

Peichen Pan
Department of Electrical and Computer Engineering
Clarkson University, Potsdam, NY 13699
panp@sun.soe.clarkson.edu

Abstract

This paper introduces a continuous version of retiming (called c-retiming*). As retiming, a c-retiming of a circuit is also an assignment of values to the nodes in the circuit. However, values in c-retiming can be real numbers as opposed to integers in retiming. Retiming and c-retiming are strongly related. In fact, a c-retiming can be converted to a retiming by a simple rounding, and the potential degradation in clock period is less than the largest gate delay in a circuit. C-retiming has two very attractive properties. It can be computed much more efficiently than retiming. Consequently, one can compute a retiming by computing a proper c-retiming. Our experimental results indicate this approach can drastically speed up the solution of retiming problems. More importantly, c-retiming can be combined with circuit modifications. Because of this property, c-retiming can be used as a tool to study synthesis and optimization problems in conjunction with retiming. We demonstrate this using the classical tree mapping problem, for which we derive an algorithm that produces a solution with a clock period provably close to optimal while considering retiming.*

1 Introduction

Retiming is a transformation that relocates the sequential elements in a circuit while preserving the functionality. It can be applied to circuits with either edge-triggered flip-flops (FFs) [1] or level-sensitive latches [2, 3]. In this paper, we focus on circuits with FFs but the ideas may also be applicable to circuits with latches. Retiming can be used to optimize the clock period, the number of FFs, or both without modifying the logic of a circuit. Many algorithms for doing so have been proposed [1]-[8].

In this paper, we introduce a continuous version of retiming (referred to as c-retiming). As (conventional) retiming, a c-retiming is also an assignment of values to the nodes in a circuit. The values can, however, be real numbers as opposed to integers in retiming. To put it in another way, we allow the introduction of *fractional FF* in c-retiming.

The main motivation of introducing c-retiming is to study synthesis and optimization problems that involve circuit modifications, in conjunction with retiming. None of the existing retiming approaches allow us to do so. As a result, retiming is mostly used as a *stand-alone* optimization technique and its potential is not fully utilized.

C-retiming has several other important properties. It is closely related to retiming. By a simple rounding, a c-retiming can be converted to a retiming with limited potential degradation in clock period. Moreover, the problem of determining a c-retiming for a target clock period can be reduced to a single-source longest path problem on the circuit graph.

We also present new retiming algorithms based on c-retiming. The algorithm is able to minimize the clock period of each of the sequential benchmark circuits in ISCAS89 suite in a few seconds on a SPARC 5.

The remainder of this paper is organized as follows: In Section 2, we introduce c-retiming. We also discuss how to minimize the clock period and the amount of fractional FF in c-retiming. In Section 3, we show the close relationship between retiming and c-retiming. We present a faster algorithm for c-retiming a circuit to a target clock period in Section 4. Section 5 presents a new retiming algorithm based on c-retiming and presents our experimental results on ISCAS89 circuits. In Section 6, we use the tree mapping problem as an example to show how c-retiming can be used to study synthesis and optimization problems in conjunction with retiming. Finally, Section 7 concludes this paper.

2 C-retiming

A circuit can be modeled as an edge-weighted, node-weighted, directed (multi-)graph (called *circuit graph*). The nodes are the primary inputs (PIs), the primary outputs (POs), and the logic gates. The weight $d(v)$ of a node v represents the propagation delay of v. The edges represent the interconnections. The weight $w(e)$ of an edge e from u to v (denoted by $u \xrightarrow{e} v$) is the number of FFs on the interconnection.

Retiming a node by a value i is an operation that removes i FFs from each fan-out edge and adds i FFs to each fan-in edge of the node. Figure 1 shows the case where $i = 1$ and -1. In general, the nodes in a circuit can be retimed collectively. It has been shown retiming preserves the functionality, when the retiming values for the PIs and POs are zero[1] [1, 9].

Let r be a retiming of a circuit where $r(v)$ is the retiming value for a node v. The weight of an edge $u \xrightarrow{e} v$ in the retimed circuit, $w_r(e)$, is $w(e) + r(v) - r(u)$. A retiming that results in nonnegative edge weights is called a *legal* retiming.

The *clock period* of a circuit is the maximum delay on the combinational paths (paths without FFs) in the circuit. A circuit can be retimed to a clock period if there is a legal

[1] Actually, as long as the PIs and POs have the same retiming value, the functionality is preserved.

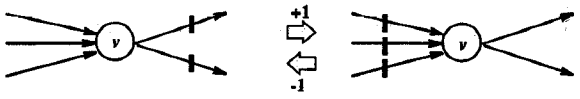

Figure 1: Retiming a node.

retiming such that the retimed circuit has the clock period. The following is an equivalent definition:

Definition 1 *A circuit is retimed to a clock period ϕ by a retiming r if the following two conditions are satisfied: (1) $w_r(e) \geq 0$ and (2) $w_r(p) \geq 1$ for each path p such that $d(p) > \phi$, where $w_r(p) = \sum_{e \in p} w_r(e)$ and $d(p) = \sum_{v \in p} d(v)$.*

As retiming, a c-retiming is also an assignment of values to the nodes. However, the values can be a real numbers. The values for the PIs are zero, but the values for the POs can be nonzero but must be less than or equal to 1. We will use s to denote a c-retiming. In a c-retiming we can move a fraction of a FF across a gate. Similar to retiming, the amount of fractional FF on an edge $u \xrightarrow{e} v$ in the retimed circuit is $w_s(e) = w(e) + s(v) - s(u)$. We define the clock period in a c-retimed circuit as follows:

Definition 2 *A circuit is c-retimed to a clock period of ϕ by a c-retiming s if $w_s(e) \geq \frac{d(v)}{\phi}$ for each edge $u \xrightarrow{e} v$.*

Let us discuss the intuition behind Definition 2. As in retiming, we also require nonnegative edge weight after c-retiming. The condition $w_s(e) \geq \frac{d(v)}{\phi}$ obviously enforces that, but it does even more. Let p be a path from u_1 to u_2 in the circuit, we then have $w_s(p) \geq \frac{d(p)-d(u_1)}{\phi}$, where $w_s(p) = \sum_{e \in p} w_s(e)$. If $d(p) - d(u_1) \geq \phi$, then $w_s(p) \geq 1$.

Therefore, the condition in Definition 2 plays the similar roles as both conditions (1) and (2) in Definition 1.

To compute a c-retiming for a target clock period ϕ, we introduce another set of weights on the edges in the circuit. The second weight $w_1(e)$ of an edge $u \xrightarrow{e} v$ is defined as $-w(e) + \frac{d(v)}{\phi}$. We have the following results:

Theorem 1 *A circuit can be c-retimed to a clock period of ϕ iff the weight of the longest paths from the PIs to each PO is less than or equal to 1, using the w_1 edge weight.*

Based on Theorem 1, we can determine whether or not a circuit can be c-retimed to a given clock period by solving a single-source longest path problem *on the circuit graph*. In fact, if the weight of the longest paths for each PO is no more than 1, the c-retiming value of a node can be set to the weight of the longest paths from the PIs to the node. Figure 2 shows the pseudo-code for computing a c-retiming for a target clock period ϕ. It returns 'success' if such a retiming exists; otherwise it returns 'failure'. The algorithm is essentially the Bellman-Ford algorithm [10]. The difference is that the relaxation is re-organized in such a way that all edges ending at the same node are relaxed as a group. This grouping is useful for integrating other design transformations into the algorithm.

```
for each node v
  if (v is a PI) s(v) ← 0;
  else s(v) ← −∞;
for i ← 1 to n { // n, the number of nodes
  done ← TRUE;
  for each node v {
    tmp ← max_{u→v} {s(u) − w(e) + d(v)/φ}
    if (v is a PO and tmp > 1) return failure;
    if (tmp > s(v)) {
      s(v) ← tmp ;
      done ← FALSE;
    }
  }
  if (done = TRUE) return success;    // s achieves φ.
}
return failure;
```

Figure 2: Algorithm for computing a c-retiming for a target clock period.

We now consider the problem of minimizing the amount of (fractional) FF subject to a given clock period ϕ. This problem is simply the following linear program:

$$\textbf{minimize} \quad \sum_e w_s(e)$$

$$\textbf{subject to:} \quad w_s(e) \geq \frac{d(v)}{\phi} \quad \text{for each } u \xrightarrow{e} v$$

As in retiming, the dual of this program is an uncapacitated min-cost flow problem. However, the flow network is simply the circuit graph. The net flow out of each node is the difference of the out-degree and the in-degree of the node. The cost of an edge e is simply $w_1(e)$. Fractional FF sharing can also be modeled in the min-cost flow formulation. We omit the details.

3 Retiming and c-retiming

The purpose of introducing c-retiming is to use it as a tool to study retiming. Hence, it is important to know the relationship between them. Ultimately, we retime (not c-retime) a circuit.

C-retiming is more flexible in moving FFs around since it can reposition a portion of a FF across a gate. As a result, one would expect if a clock period is achievable by retiming, it should also be achievable by c-retiming. This is true as we have the following result:

Theorem 2 *If a circuit can be retimed to a clock period of ϕ, then it can also be c-retimed to ϕ.*

On the other hand, retiming can be viewed as a coarse-grain version of c-retiming. Converting a c-retiming to a retiming in general may not preserve the clock period. However, it can be shown that the possible degradation is very limited. Let D be the largest gate delay in a circuit, we have the following result:

Theorem 3 *Let s be a c-retiming that achieves clock period ϕ. Let r be the retiming defined as follows:*

$$r(v) = \begin{cases} 0 & v \text{ is a PI or a PO} \\ \lceil s(v) \rceil - 1 & \text{otherwise.} \end{cases}$$

Then r can achieve a clock period less than $\phi + D$.

The following is a corollary of Theorems 2 and 3:

Corollary 1 *The minimum clock period that can be achieved by retiming is less than D plus the minimum clock period that can be achieved by c-retiming.*

When each gate has one unit of delay (the unit-delay model), $D = 1$ and the clock period of a retimed circuit is an integer. In this case, from Theorem 3, the clock period from the retiming r is less than $\phi + 1$. Thus, it is less than or equal to ϕ if ϕ is an integer. Combining the above analysis with Theorem 2, we have the following result for the unit-delay model.

Corollary 2 *Assuming the unit delay model, for any positive integer ϕ, a circuit can be c-retimed to a clock period of ϕ iff the circuit can be retimed to of clock period of ϕ.*

We are not able to establish any formal relationship between minimizing the total amount of fractional FF in c-retiming and minimizing the number of FFs in retiming. Our experimental results seem to indicate that minimizing total amount of fractional FF is a reasonably good heuristic for minimizing the number of FFs, but further research is needed in this direction.

4 A faster c-retiming algorithm

In this section, we present a faster algorithm for determining a c-retiming for a target clock period.

We have shown that determining a c-retiming for a target clock period is equivalent to solving a single-source longest path problem on the circuit graph using the w_1 edge weight. The algorithm in Figure 2 has a time cost $O(nm)$, where n and m are the respective numbers of nodes and edges in the circuit. In this section, we describe a technique to speed up the algorithm. With the speed-up the algorithm exhibits a time cost $O(n + m)$ in practice.

The technique is based on the observation that if the circuit contains no feedback loops, The c-retiming algorithm in Figure 2 needs only one iteration of relaxation if the nodes are relaxed in topological order starting with the PIs. A topological order respects all node dependencies by finding longest paths to predecessors before successors. For a circuit with feedback loops, there is no linear ordering that respects all dependencies. Intuitively, however, the more dependencies the algorithm respects during relaxation, the less the number of relaxation iterations there are. To formalize this idea, let U be a feedback vertex set (FVS) of the circuit. That is, the removal of the vertices in U breaks all feedback loops in the circuit. Let V be a topological ordering of the nodes in the circuit excluding PIs and POs, after the nodes in U are removed. Consider the following ordering of the nodes in the circuit: PIs, V, U followed by POs (referred to as a *pseudo-order*). If the nodes are relaxed according to a pseudo-order, we expect the algorithm needs fewer relaxation iterations than using an arbitrary order. This is indeed the case as we have the following result:

Theorem 4 *If the nodes are relaxed according to the pseudo-order, the algorithm stops in at most $|U| + 1$ relaxation iterations if there is no positive cycle.*

Figure 3 is the improved algorithm. Its time cost is $O(|U|m)$ in the worst-case. A more detailed analysis can show the time cost is $O(cm)$, where c is the maximum number of nodes shared by U and any simple path from a PI to a PO. c is expected to be much smaller than $|U|$.

```
CTCHECK(G, φ)
  // G is the circuit and φ is the target clock period
  Let v_1, v_2, ..., v_n be a pseudo-order of the nodes;
  for each node v
    if (v is a PI) s(v) ← 0;
    else s(v) ← −∞;
  for i ← 1 to |U| + 2 {
    done ← TRUE;
    for j ← 1 to n {
      tmp ← max_{u →e v_j} {s(u) − w(e) + d(v_j)/φ}
      if (v_j is a PO and tmp > 1) return failure;
      if (tmp > s(v_j)) {
        s(v_j) ← tmp ;
        done ← FALSE;
      }
    }
    if (done = TRUE) return success;
  }
  return failure;
```

Figure 3: Improved algorithm for computing a c-retiming for a target clock period.

One issue we have yet to address is finding a FVS. Although finding a minimum FVS is an NP-hard problem, finding a FVS is not difficult. An obvious FVS consists of all nodes that have at least one fan-out edge with non-zero weight. This is a FVS because if we remove all such nodes (and also the edges incident to them), the remainder of the circuit contains no FFs, so must be acyclic. Similarly, the nodes having a non-zero weight fan-in edge also form a FVS. Note that the number of nodes in either of these FVS's is less than or equal to the number of FFs in the circuit. If we use either of these FVS's, the algorithm in Figure 3 will have a time cost $O(f(n + m))$, where f is the number of FFs in the circuit. For practical circuits, obviously $f << n$. Thus, the modified algorithm is much faster even if we use these obvious FVS's. Moreover, efficient heuristic algorithms exist that can find a FVS with a size very close to minimum for practical circuits (see, for example, [11]). Our experiments show even using the obvious FVS's the algorithm stops in less than ten iterations in all cases. Thus, in practice, the algorithm exhibits a time cost of $O(n + m)$.

5 Retiming via c-retiming

We now present a retiming algorithm based on c-retiming. The algorithm first searches for the minimum clock period that can be achieved by c-retiming. This can be done by calling the procedure CTCHECK to carry out a binary search on the target clock period. After the minimum clock period is found, the algorithm determines a c-retiming by minimizing the total amount of fractional FF subject to the minimum clock period. Finally, the c-retiming is converted to a retiming according to Theorem 3. Figure 4 is the pseudo-code of the algorithm where ϵ is a controlling factor.

```
RECRE(G, ε)
  φ_h ← largest combinational delay in G;
  φ_l ← smallest combinational delay between FFs in G;
  while (φ_h − φ_l ≥ ε) {
    φ ← (φ_l + φ_h)/2;
    if (CTCHECK(G, φ) = success) φ_h ← φ;
    else φ_l ← φ;
  }
  Determine a c-retiming s that minimizes the amount
     of fractional FF subject to the clock period φ_h;
  Convert s to retiming r;
  Retime G according to r;
  Return the retimed circuit;
```

Figure 4: A retiming algorithm based on c-retiming.

Let ϕ_{opt} be the minimum clock period that can be achieved by retiming. Based on the discussion in Section 3, we have the following result:

Theorem 5 *The clock period of the circuit returned by* RECRE *is less than* $\phi_{opt} + D + \epsilon$. *When each gate in* G *has one unit of delay and* $\epsilon = 1$, *the clock period of the circuit is exactly* ϕ_{opt}.

RECRE has been implemented. In our implementation, we use a min-cost flow program based on the cost-scaling technique to minimize the amount of fractional FF [12]. Fractional FF sharing is also included in our implementation. We tested RECRE on the sequential benchmark circuits in ISCAS89 suite. The results are reported in Table 1. (We dropped small circuits and a few circuits without reduction in clock period.) The unit-delay model is used in our experiment (so the clock period of the retimed circuit is actually minimum). The experiments were done on a SPARC 5 with 32MB memory. We list in the table the CPU times for searching the minimum clock period (ϕ) and minimizing the amount of fractional FF (FF). As can be seen, the algorithm is very efficient.

Overall RECRE reduces the clock period by 30% with 13% increase in the number of FFs. It should be pointed out the number of FFs may not be optimal.

circuit		initial		RECRE		times (s)	
name	gates	ϕ	FFs	ϕ_{opt}	FFs	ϕ	FF
s838	446	17	32	16	52	0.01	0.38
s938	446	17	32	16	52	0.01	0.35
s953	395	16	29	13	35	0.03	0.41
s967	394	14	29	12	35	0.03	0.40
s991	519	59	19	54	22	0.01	0.38
s1269	569	35	37	19	90	0.05	0.56
s1423	657	59	74	53	76	0.05	0.63
s1488	653	17	6	16	7	0.01	0.68
s1494	647	17	6	16	8	0.01	0.66
s1512	780	30	57	23	75	0.06	0.66
prolog	1601	26	85	13	231	0.06	1.68
s3271	1572	28	116	15	198	0.21	1.78
s3330	1789	29	132	14	218	0.06	1.80
s3384	1685	60	183	27	207	0.11	2.50
s4863	2342	58	88	30	183	0.15	3.20
s5378	2779	25	164	21	189	0.15	2.85
s6669	3080	93	231	26	355	0.33	8.48
s9234	5597	58	211	38	276	0.43	6.81
s13207	7951	59	638	51	487	0.50	10.48
s15850	9772	82	534	63	559	0.83	12.11
s35932	16065	29	1728	27	1729	0.86	39.16
s38417	22179	47	1636	32	1958	17.70	40.92
s38584	19253	56	1426	48	1432	1.18	43.04
Total		931	7493	643	8474		
Ratio		1	1	0.69	1.13		

Table 1: Experimental results: retiming via c-retiming.

6 Applications

Many existing approaches to sequential synthesis and optimization operate on only the combinational logic between FFs. That is, the FFs in a sequential circuit are simply removed to obtain a combinational network. Then the combinational network is optimized. Finally, the FFs are connected back. Obviously, these approaches can only explore a small portion of the available design space since they do not consider different FF configurations that can be obtained by retiming. It also fails to consider the signal dependencies across FF boundaries since the network is segmented into independent pieces after the removal of FFs.

There have been a few methods that try to consider retiming during synthesis and optimization [13, 14]. However, most of them use retiming as either a pre-processing step (i.e., determining a "good" initial FF configuration) [15], or a post-processing step [16, 17]. In some cases, limited interaction of retiming and circuit transformations is explored. By and large, retiming and other design transformations are done separately. One major obstacle to combining retiming with other design transformations is that existing retiming algorithms cannot give useful guidance to these transformations, since these algorithms assume the circuit structures and gate delays are given and fixed.

We now use the *tree mapping problem* [18] as an example to show that c-retiming can be combined with other design transformations. It should be pointed out that tree covering is a relatively simple problem. This is, in fact, one of the reasons we use it as an illustrating example.

In the tree mapping problem we are given a cell library P, a matching procedure MATCH, and a sequential tree network N. The matching procedure takes a node v in the

network and the cell library P and produces all cells in P that match a subtree rooted at v. (We assume the matching algorithm ignores FFs in finding matches). Each cell p has a propagation delay $d(p)$. The problem is to cover the network such that the clock period of the resulting network comprised of library cells (called bound network) is minimized[2].

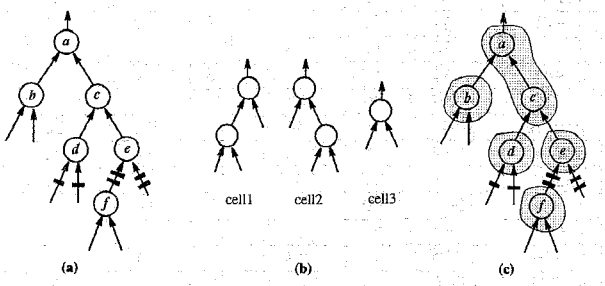

Figure 5: (a) A tree network, (b) a simple cell library, and (c) a bound network from the traditional approach.

When the network is combinational, the problem can be solved optimally using dynamic programming [20, 19, 21]. For sequential circuits, the traditional approach simply removes the FFs, covers the remaining combinational network optimally, and finally places the FF back. This approach, however, may not find the best solution. Consider the network in Figure 5(a) with a library having three cells as shown in Figure 5(b). We assume each cells has one unit delay. The solution produced by the traditional approach is shown in Figure 5(c). The resulting bound network has a clock period of two and retiming at this stage cannot reduce the clock period since there is a combinational path from a PI to the PO with two cells. However, if we move the two FFs on the output of gate f to its inputs and cover the network as shown in Figure 6(b) (NOT optimal for the combinational network), the resulting bound network can be retimed to a clock period of one as shown in Figure 6(c).

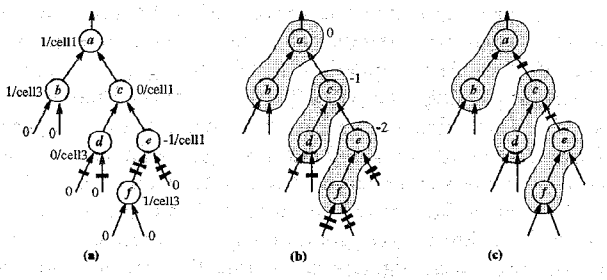

Figure 6: (a) c-retiming value and the best cell at each node, (b) the initial bound network, and (c) the retimed network.

[2]For simplicity reason, we ignore load in delay calculation. To consider load, the *load binning* technique [19, 18] can be easily incorporated.

Grodstein *et. al.* [22] proposed a tree mapping algorithm that takes into consideration of retiming. Using c-retiming, we derive another algorithm. Unlike the algorithm in [22], our algorithm does not enumerate all possible retiming values at each node.

Figure 7 summarizes our mapping algorithm for a target clock period ϕ. The algorithm examines the nodes in N in topological order starting with PIs. For each node, it calls BESTCELL to determine a cell for the node that results in the minimum c-retiming value at the node. The algorithm can also be viewed as a modification of CTCHECK with the procedure BESTCELL is used to update *tmp*. (Actually, variable *tmp* is no longer needed and the value assigned to *tmp* can be directly assigned to $s(v)$, since the network contains no loops.)

```
SEQUENTIALTREEMAP(N, φ)
 Let v_1, v_2, ..., v_n be a topological order of the nodes
 in N, starting with PIs;
 for each node v
     if (v is a PI) s(v) ← 0;
 for j ← 1 to n {
     s(v_j) ← BESTCELL(v_j, P);
     if (v_j is a PO and s(v_j) > 1) return failure;
 }
 return success;

BESTCELL(v, P)
 d_min ← ∞;
 for each p ∈ MATCH(v, P) {
     for each u in inputs(p)
         f_u ← the number of FFs on the path from u to v
     d ← max_{inputs(p)}{s(u) − f_u + d(p)/φ};
     if (d_min > d) {
         d_min ← d;
         cell ← p;
     }
 }
 return d_min;
```

Figure 7: Mapping algorithm for sequential tree network for the target clock period ϕ.

After all the minimum c-retiming values have been computed, the next step is to cover the output of the network with the best cell (as determined by BESTCELL) and move any FFs covered by a cell to the inputs of the cell, then continue to cover the inputs using the best cells until the PIs are reached. The final step is converting the c-retiming s to a retiming and apply the retiming to the initial bound network. For the network in Figure 5, assuming $\phi = 1$ the c-retiming values and the corresponding best cells are listed in Figure 6(a) and Figure 6(b) is the initial bound network. We then convert the c-retiming to a retiming (see Theorem 3). The retiming values are also shown in Figure 6(b). Applying the retiming to the bound network, we obtain exactly the network in Figure 6(c). We have the following result:

Theorem 6 *If* SEQUENTIALTREEMAP *returns failure, then N does not have a bound network with a clock period of ϕ; if returns success, the retiming converted from the c-retiming s achieves a clock period less than $\phi + D$, where D is the largest cell delay.*

Using binary search on ϕ as in RECRE, we can find a bound network with a clock period less than $D + \epsilon$ away from the optimal one.

7 Conclusion and future work

In this paper, we introduce a fine-grain version of retiming called c-retiming. C-retiming has several nice properties. The most important one is that it can be combined with other design transformations. It can also be computed much faster than retiming. A c-retiming can be converted to a retiming with limited degradation in clock period and number of FFs. We proposed a fast retiming algorithm based on c-retiming. We also use an example to illustrate that c-retiming can be used to study synthesis and optimization problems in conjunction with retiming.

Several directions are currently being pursued. One is using c-retiming to re-examine other accepted techniques for sequential synthesis and optimization with the objective of incorporating retiming. In fact, the c-retiming concept has been applied to technology mapping for LUT-based FPGAs and circuit clustering [23, 24]. Novel algorithms have been proposed that can produce designs with optimal clock periods. Another problem is to study how to use c-retiming to minimize the number of FFs. Yet another direction is to find direct retiming approaches (not via c-retiming) that can accommodate circuit modifications.

References

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

[2] B. Lockyear and C. Ebeling, "Optimal retiming of multi-phase level-clocked circuits," in *Advanced Research in VLSI*, pp. 265–280, 1992.

[3] A. T. Ishii, C. E. Leiserson, and M. C. Papaefthymiou, "Optimizing two-phase, level-clocked circuitry," in *Advanced Research in VLSI*, pp. 245–264, 1992.

[4] N. V. Shenoy and R. Rudell, "Efficient implementation of retiming," in *Intl. Conf. on Computer-Aided Design (ICCAD)*, pp. 226–233, 1994.

[5] R. B. Deokar and S. S. Sapatnekar, "A fresh look at retiming via clock skew optimization," in *ACM/IEEE Design Automation Conf. (DAC)*, pp. 310–315, 1995.

[6] N. Maheshwari and S. S. Sapatnekar, "An improved algorithm for minimum-area retiming," in *ACM/IEEE Design Automation Conf. (DAC)*, pp. 2–7, 1997.

[7] T. Soyata and E. Friedman, "Retiming with non-zero clock skew, variable register, and interconnect delay," in *Intl. Conf. on Computer-Aided Design (ICCAD)*, pp. 234–241, 1993.

[8] K. Lalgudi and M. Papaefthymiou, "DELAY: An efficient tool for retiming with realistic delay modeling," in *ACM/IEEE Design Automation Conf. (DAC)*, pp. 304–309, 1995.

[9] V. Singhal, C. Pixley, R. Rudell, and R. Brayton, "The validity of retiming sequential circuits," in *ACM/IEEE Design Automation Conf. (DAC)*, pp. 316–321, 1995.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. New York: McGraw-Hill Book Company, 1990.

[11] D. H. Lee and S. M. Reddy, "On determining scan flip-flops in partial-scan designs," in *Intl. Conf. on Computer-Aided Design (ICCAD)*, pp. 322–325, 1990.

[12] A. Goldberg, "An efficient implementation of a scaling minimum-cost flow algorithm," Tech. Rep. STAN-CS-92-1439, Stanford University, 1992.

[13] G. De Micheli, "Synchronous logic synthesis: algorithms for cycle-time minimization," *IEEE Trans. on Computer-Aided Design*, vol. 10, pp. 63–73, 1991.

[14] U. Weinmann and W. Rosenstiel, "Technology mapping for sequential circuits based on retiming techniques," in *Proc. European Design Automation Conf.*, pp. 318–323, 1993.

[15] 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*, vol. 12, pp. 568–578, 1993.

[16] 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)*, pp. 504–509, 1992.

[17] 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)*, pp. 483–489, 1993.

[18] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, Inc., 1994.

[19] R. Rudell, "Logic synthesis for VLSI design," Tech. Rep. Memorandum UCB/ERL M89/49, University of California at Berkeley, 1989. (Ph.D. dissertation).

[20] K. Keutzer, "Dagon: Technology binding and local optimization by DAG matching," in *ACM/IEEE Design Automation Conf. (DAC)*, pp. 341–347, 1987.

[21] H. Touati, "Performance-oriented technology mapping," Tech. Rep. UCB/ERL M90/109, University of California at Berkeley, 1990. (Ph.D. dissertation).

[22] J. Grodstein, E. Lehman, H. Harkness, H. Touati, and B. Grundmann, "Optimal latch mapping and retiming within a tree," in *Intl. Conf. on Computer-Aided Design (ICCAD)*, pp. 242–245, 1994.

[23] P. Pan and C. L. Liu, "Optimal clock period FPGA technology mapping for sequential circuits," in *ACM/IEEE Design Automation Conf. (DAC)*, pp. 720–725, 1996.

[24] A. Karandikar, P. Pan, and C. L. Liu, "Optimal clock period clustering for sequential circuits with retiming," in *Intl. Conf. on Computer Design (ICCD)*, 1997. (this proceedings).