Mapping and Retiming Revisited

Alan Mishchenko **Robert Brayton**
Department of EECS, UC Berkeley
{alanmi, brayton}@berkeley.edu

Masahiro Fujita
University of Tokyo
fujita@ee.t.u-tokyo.ac.jp

Abstract

Twenty five years ago, combinational mapping was extended to sequential circuits by proposing a new way to compute the arrival time at a node in cyclic networks containing registers. Despite offering significant (up to 25%) delay improvements, this method did not gain wide-spread use because of the implementation complexity and an expectation of slow runtime. This paper offers a fresh look at the method and introduces simplifying assumptions to decouple the mapping phase and the retiming phase of the computation while still providing a substantially reduced delay. Moreover, it is shown that the delay optimization can be done with a negligible runtime overhead, compared to state-of-the-art combinational mapping.

1 Introduction

In 1996, Peichen Pan and Chung-Laung Liu introduced a new way of reasoning about timing in sequential circuits [26]. Instead of cutting the circuit at the register boundary and doing timing analysis and logic optimization on the combinational circuit, they extended the notion of combinational delay to sequential circuits. The novel idea was, when the arrival time computation passes over an edge between two nodes containing a register, to subtract the target clock period given by the user.

This seemingly minor variation has profound implications on the analysis and optimization of sequential circuits. It allows for computing sequential arrival times at a node by iterating the delay computation over the circuit until a fixed point is reached. In a way, this delay computation is similar to inductive computation of sequentially equivalent nodes, which extends combinational proof methods to work for sequential circuits by iterating the proof attempts until a fixed point is reached [21].

One method that emerged from the new way to compute the sequential arrival times, is the possibility of integrating *mapping*, which finds a structural cover of the circuit, and *retiming*, which moves registers over the circuit nodes.

Retiming is traditionally performed before and/or after mapping. However, in this case, the cover derived by mapping may prevent finding an efficient retiming. A similar *structural bias* [4] exists when a circuit is optimized for a given LUT size but mapped into a different LUT size. For example, optimizing for 4-input LUTs requires that multi-input gates are broken down into smaller gates favoring 4-input grouping. Thus, a 32-input and-gate given as a well-balanced tree of two-input gates fits 4-input LUTs well but does not fit 6-input LUTs well.

When mapping and retiming are integrated, the sequential arrival times are exploited by the mapper to adjust the cuts used to build the cover of the subject graph in such a way that a delay-efficient retiming is possible. Thus, combining the two transforms helps overcome structural bias that is present when mapping is performed independently from retiming.

The proposed integration is in line with the recent trend to integrate different aspects of the synthesis process, motivated by the shrinking of DSM technologies. As a result, more synthesis aspects are seen as interrelated and computed simultaneously. Examples of this kind of integration are as follows:

1. Tech-independent synthesis and mapping [18][4][17]
2. Mapping and retiming [20][27][8][9]
3. Retiming and placement [2][6]
4. Re-synthesis and retiming [2][28]
5. Tech-independent synthesis and placement [3][16][13]
6. Re-wiring and placement [5]
7. Clock skewing and placement [14]

Integrated methods have greater potential because they explore several solution spaces simultaneously, rather than sequentially when a solution found in one space is fixed before running optimization in the next space, and so on.

The contributions of this paper are:

(1) Reviewing the methodology of the original integration of mapping and retiming [26][27][28].

(2) Introducing a number of simplifying assumptions, which make it easier to implement the integrated flow.

(3) Showing how the mapping phase and the retiming phase can be cleanly separated without compromising the delay gains.

(4) Demonstrating experimentally that the integrated flow has a negligible runtime overhead, compared to the runtime of the combinational mapper used in the mapping phase.

The presentation in this paper, and our current implementation of the integrated flow is limited to designs with a single clock domain and edge-triggered D-flip-flops with given initial states. However, the framework can be extended to handle designs with multiple clock domains and explicit set/reset logic.

The paper is organized as follows. Section 2 describes the background. Section 3 presents the integration procedures. Section 4 shows simplifications of these procedures that make implementation easier. Section 5 shows experimental results. Section 6 concludes the paper and outlines future work.

2 Background

A *Boolean network* is a directed acyclic graph (DAG) with nodes corresponding to logic gates and directed edges corresponding to the wires. AIG is a Boolean network composed of two-input ANDs and inverters. The terms Boolean network, design, and circuit are used interchangeably.

Each node has a unique integer number called the *node ID.* A node has zero or more *fanins,* i.e. nodes that are driving this node, and zero or more *fanouts*, i.e. nodes driven by this node. The *primary inputs* (PIs) of the network are nodes without fanihs in the current network. The *primary outputs* (POs) are a subset of nodes of the network. If the network is sequential, the memory elements are assumed to be D-flip-flops with initial states. Terms memory elements, flop-flops, and registers are used interchangeably in this paper.

A transitive *fanin* (*fanout*) *cone* of node *n* is a subset of all nodes of the network reachable through the fanin (fanout) edges from the given node. The *level* of a node is the length of the longest path from any PI to the node. The node itself is counted towards the path length but the PIs are not.

The area and delay of an FPGA mapping is measured by the number of LUTs and the number of LUT levels respectively. The delay of a standard cell mapping is computed using pin-to-pin delays of gates assigned to implement a cut. The load-independent timing model is assumed throughout the paper.

An *And-Inverter Graph* (AIG) is a Boolean network whose nodes are two-input ANDs. Inverters are marked by a special attribute on the edges of the network.

A cut *C* of node *n* is a set of nodes, called leaves, such that

1. Each path from any PI to *n* passes through a leaf.
2. For each leaf, there is a path from a PI to *n* passing through the leaf and not through any other leaf.

Node *n* is called the root of *C*. A *trivial cut* of node *n* is the cut $\{n\}$ composed of the node itself. A non-trivial cut is said to *cover* all the nodes found on the paths from the leaves to the root, including the root but excluding the leaves. A trivial cut does not cover any nodes. A cut is *K-feasible* if the number of its leaves does not exceed *K*. A cut C_1 is said to be *dominated* by C_2 if there is another cut $C_2 \subset C_1$.

A *cover* of an AIG is a subset *R* of its nodes such that for every $n \in R$, there exists exactly one non-trivial cut $C(n)$ associated with it such that:

1. If *n* is a PO, then $n \in R$.
2. If $n \in R$, then for all $p \in C(n)$ either $p \in R$ or *p* is a PI.
3. If *n* is not a PO, then $n \in R$ implies there exists $p \in R$ such that $n \in C(p)$.

The last requirement ensures that all nodes in *R* are "used".

We use an AIG accompanied with a cover to represent a logic network. This is motivated by the previous work on AIG-based technology mapping [24]. The advantage is that different covers of the AIG (and thus different network structures) can be easily explored using fast cut enumeration. The *logic function* of each node $n \in R$ of a cover is simply the Boolean function of *n* computed in terms of the cut leaves, $C(n)$. During the cut computation, this function can be derived as a truth table using the underlying AIG between the root AIG node and its cut.

3 Integrating mapping with retiming

This section summarizes the previous work [26][27][28] on the integration of mapping and retiming.

The key insight here is that delay-aware mapping for standard cells [18] and FPGAs [24] can be extended to sequential circuits by considering registers as labels on the edges connecting logic nodes; the DAG becomes a cyclic circuit with labels. The overall mapping procedure for cyclic circuits is similar to the traditional combinational mapping with a few modifications: (1) the concept of arrival times is extended to account for register labels on the edges; (2) computation of the arrival times is done by iterating over the circuit, and; (3) the resulting mapping has a retiming associated with it, which when performed on the mapped circuit, leads to the minimum clock period over all possible mappings and retimings. Below, we describe these modifications in detail.

Computing all *K*-cuts and their matches is done for each test-case once as a preprocessing step. However, the computation of sequential arrival times may be repeated for different clock periods, ϕ, as well as during area recovery.

3.1 Sequential arrival times

Sequential arrival times are computed assuming a fixed clock period, ϕ. The delay of a (possibly cyclic) path *p* is defined as:

$$l(p) = \sum_{n \in p} d(n) - \phi \sum_{e \in p} t(e),$$

where *d(n)* is the delay of node *n* and *t(e)* is the number of registers on edge *e*. Thus, the sequential delay is the difference between the sum of delays of nodes on the path and ϕ multiplied by the total number of registers on the path. The rational is that each register delays the signal at the end of the path by one clock cycle. The sequential arrival time [27] at node *n* is the maximum of the arrival times of all (possibly cyclic) paths originating at a PI and ending at *n*: $l(n) = \max_{p \in PATH(PI \to n)} l(p)$. As in the combinational case, the clock period ϕ is feasible if and only if the arrival time at a PO does not exceed ϕ at any time during the iterative computation. Since cycles are included, the computation involves iteration.

3.2 Iterative computation of the arrival times

The computation is shown in Figure 1 [27]. The arrival times of the PI nodes are initialized to 0 and those of internal nodes and the POs to $-\infty$. In each iteration, nodes are visited in a topological order and new arrival times are computed as:

$$l_{new}(n) = \min \max \{l(u) - t_c^u \phi + d_c^u\}$$

where minimum is over all cuts *c* of node *n*, maximum is over all leaves *u* of cut *c*, $l(u)$ is the arrival time of leaf *u*, t_c^u is the number of registers along the path from *u* to *n,* and d_c^u is the pin-to-pin delay of cut *c*. The delay is one in the case of unit-delay model. Thus, for each node, *n*, we consider all possible cuts and record the one that yields the smallest new arrival time. Since the sequential cuts have already been pre-computed and stored, this computation is fast.

```
**SequentialArrivalTimes** ( network $G$, clock period $\phi$ ) {
  **for** each node $n$ in $G$ **do**
      **if** $n$ is a PI **then** $l(n) = 0$ **else** $l(n) = -\infty$
  **do** {
      **for** each non-PI node $n$ in $G$ **do**
          $l_{new}(n) = \min_{M \in \text{matches}(n)} \max_{u \in \text{fanin}(M)} \{l(u) - t_{u \to n}\phi + d_{u \to n}\}$
          $l(n) = \max\{l(n), l_{new}(n)\}$
      **if** $n$ is a PO and $l(n) > \phi$
          **return** INFEASIBLE
  } **while** (the arrival times of some nodes have changed)
  **return** FEASIBLE
}
```

Figure 1. Iterative computation of sequential arrival times.

The arrival time of the node is updated if the new value is larger than the old value. Thus the arrival time at a node increases monotonically during the computation. If the arrival time at any PO exceeds ϕ, the iteration is stopped and the clock period is declared infeasible. Otherwise, the arrival times are guaranteed to converge and the clock period is feasible in the sense that there exists a retiming of the circuit to achieve this (modulo the maximum delay of any gate in the library).

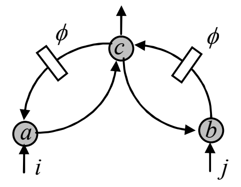
To find an optimum clock period, a binary search is performed. In each step of the binary search, the iterative procedure in Figure 1 is repeated with a new clock period. The computation is fast since all cuts and all matches for all choices have been pre-computed. Thus the best match is found which minimizes the clock period. Note that the best match at a node changes dynamically as the sequential arrival times continue to increase before converging.

3.3 Retiming associated with the final mapping

When the optimum clock period, ϕ^{opt}, is known, the mapping (standard cell or FPGA) for the circuit is selected, as described in Section 3.1. For each node *n,* which is the output of a gate of the mapping, its retiming lag is computed as follows [27]:

$$r(n) = \begin{cases} 0, \text{ if } n \text{ is a PI or PO} \\ \left\lceil \dfrac{l^{opt}(n)}{\varphi^{opt}} \right\rceil - 1, \text{ otherwise} \end{cases}$$

If the mapped circuit is retimed using this formula, the resulting clock period can be slower than the optimum one, ϕ^{opt}, by at most the delay of a gate [29]. When the unit delay model is used in the fixed-LUT-size FPGA mapping, this retiming gives the optimum clock period.



$\phi = 1$

Iter	*a*	*b*	*c*
0	-∞	-∞	-∞
1	1	1	2
2	-	-	-
3	-	-	-

$\phi = 2$

Iter	*a*	*b*	*c*
0	-∞	-∞	-∞
1	1	1	2
2	1	3	2
3	1	3	2

Figure 2. Example of sequential arrival time computation.

Example. The network in Figure 2 illustrates the computation of the sequential arrival times for the clock periods of 1 and 2 shown in the tables. The combinational delay of an internal node is 1. The longest combinational path (*a, c, b*) has delay 3. Initially, the arrival times of the PIs, *i* and *j*, are set to 0, and arrival times of the internal nodes, *a*, *b*, and *c*, are set to -∞. The clock period of 1 is infeasible because the arrival times of PO node *c* exceeds the clock period after the first iteration. The clock period of 2 is feasible because the arrival times converge after two iterations. The final retiming is $r(a) = r(c) = 0$, $r(b) = 1$. Indeed, if the register on the edge (*b, c*) is retimed backward over node *b*, the longest combinational path has delay 2.

3.4 Overview of the integration

Figure 5 outlines the overall flow, emphasizing where the various computations enter the picture. Computation begins by performing logic synthesis and accumulating functionally equivalent networks, which are processed by the FRAIG manager, resulting in the choice AIG. Next, the cuts are computed for each node, and matches are found for each cut. All such cuts and matches are stored for use in the sequential arrival time computations. A binary search is performed to find the best achievable clock period. If the result is *not okay* in the sense that the optimum clock period is not acceptable, additional synthesis can be applied, resulting in more choices, which may improve the achievable clock period.

Once the target clock period is found, the associated retiming is performed, and the final network, after area recovery, is produced. Note that steps denoted *cuts and matches* and *seq arrival times* (in double boxes in Figure 3) involve iteration over the sequential (cyclic) AIG until convergence, as discussed in the previous sections. Thus, computing cuts and matches is done once, while computing sequential arrival times is repeated several times during the binary search.

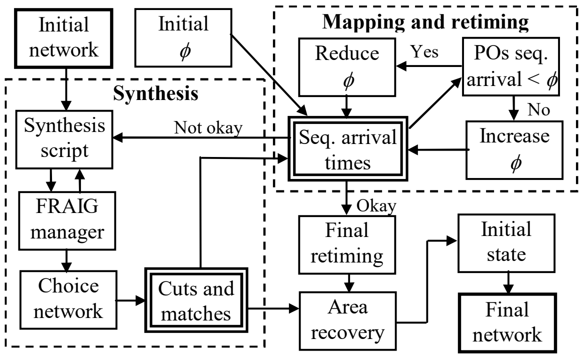


Figure 3. High-level view of the integration flow.

Besides the final retiming, Figure 3 shows two post-processing steps, area recovery, and initial state computation. Area recovery involves two aspects, remapping the off-critical paths in order to save area, and moving the registers to minimize their number. The first can be done using the area recovery methods for combinational mapping [4][24] provided the registers are fixed. However, we will want to move the registers to decrease their number, especially for standard cell designs. Also, the retiming lags computed as described in Section 3.3 are generally bad for area, since they move the registers to their most forward position. One idea for area recovery is to use the incremental methods [30][31].

The computation of the initial states for the new register positions is based on breaking down retiming into a sequence of forward and backward retimings. Computation of the initial state for forward retiming is easy. Computation of the initial state for backward retiming is reduced to a SAT problem, which "records" the sequence of backward register movements during retiming. Although this is not always satisfiable, our experience with the benchmarks so far is that such non-satisfiability rarely happens. Retiming operations of networks with arbitrary gates or logic nodes can be reduced to retiming of a sequential AIG that reflects the structure of the network.

4 Proposed simplifications

In this section, we describe one of the contributions of the paper: several ways to simplify the above original computation without compromising the quality of results reported in [23].

The proposed simplifications make the integrated mapping and retiming easier to understand and implement.

4.1 Using one combinational cut per node

It has been shown [20] that only one cut per node, computed by a dedicated procedure, is sufficient for finding a delay-optimal combinational mapping for 95% of benchmarks. For the remaining 5%, the resulting delay is typically one level worse than the delay computed using a typical number of cuts, which in most technology mappers ranges from 8 to 16.

The fact that only one cut needs to be computed and updated allows for efficient memory allocation (when cuts for all nodes are stored in one pre-allocated array) and fast computation (when one pass over the subject graph runs 10x faster than a delay-oriented pass in a combinational mapper). This explains why the proposed integrated approach, which finds the optimal retiming, takes only about 10% of the runtime of state-of-the-art combinational mappers, which can be used to perform the final mapping, as discussed in Section 4.3 below.

Moreover, the cuts used to evaluate the sequential arrival times do not have to be sequential, as in the original formulation [26][27][28]. It is ok to limit the computation to only combinational cuts (that is, the cuts that do not cross the register boundary). This leads to a 3% degradation in the delay, which is small, compared to the overall 25% gain in delay achieved by the proposed method. However, the complexity of the implementation is greatly reduced because using sequential cuts makes both mapping and retiming more tedious to implement.

4.2 Limiting retiming to one timeframe

One downside of the retiming computed using the sequential arrival times, as shown in Section 3.3, is that it does not attempt to minimize the number of register moves, nor does it impose a limit on how far forward or backward the registers can move.

It was found experimentally that, for most of the circuits considered, the flop moves can be limited to only one timeframe, without losing much of the expected delay gains.

The restriction of the retiming to be only within one timeframe means that the retiming lags of the nodes belong to the set {-1, 0, 1}. This implies that the nodes are divided into three disjoint classes: those retimed forward once (-1), those not retimed (0), and those retimed backward once (1).

This observation allows for a number of desirable short-cuts in the implementation of the procedure that derives the final circuit after retiming and the new equivalent initial state after retiming. The latter is needed for proving sequential equivalence after applying integrated mapping and retiming using a sequential equivalence checker, such as [21].

4.3 Using any mapper to derive the final mapping

The original formulation of the integrated mapping and retiming [26][27][28] implies that the same sequential cuts that are used to evaluate sequential arrival times, are later re-used during mapping. The same assumption was used in the early implementation [23] that independently validated the original formulation. However, in the follow-up work on technology mapping, it was found [24] the cuts used to evaluate sequential arrival times do not have to be re-used by the mapper.

The present work is based on a similar approach. We first compute sequential arrival times using one combinational cut per node. Then, we find the retiming, which guarantees that the reduced clock period can be achieved, and move the flops to the new positions indicated by this retiming. Finally, we perform combinational mapping with the same cut size to drive the resulting mapping. In our experiments, this mapping, as expected, results in the same minimized clock period that is predicted by the integration of mapping and retiming.

5 Experimental results

The integrated flow is implemented in ABC 0 as command *&sif*. The resulting circuits are checked for sequential equivalence using equivalence checker *dsec* [21].

The benchmarks used are sequential AIGs derived from 10 designs included in the IWLS `05 benchmark set [15].

The total runtime of command *&sif –K* 6 for the designs in Table 1, was close to 3 seconds on a mainstream CPU running a single thread. Combinational mapping using command *&if –K* 6 took about 1 minute on the same computer.

The results of mapping into 6-input LUTs are presented in Table 1. The experiment shows that the logic level is reduced, on average, by 18.3%, which is in agreement with the previous evaluations of the integration [23]. However, the LUT count and the flop count have increased substantially.

These results are preliminary in that the expected delay optimization is performed and the resulting circuits having the expected delay were derived and verified, but reducing register count after retiming is not yet implemented. (The register count minimization is expected in the final version of the paper.)

Here is a summary of other experiments with integrated mapping and retiming reported in the previous work [23] for the same set of benchmarks:

1. The whole is more than the sum of its parts; the effectiveness of both mapping and retiming is enhanced by integration beyond what it can do alone.
2. Thus, retiming by itself was relatively ineffective (~3%) but when integrated with mapping it leads to substantial gains (18% without choices and 26% with choices).
3. The use of the integrated flow is equally effective for both FPGAs and SCs.
4. The runtimes confirm the scalability of the proposed integrated flow.

6 Conclusion and future work

This paper focuses on a synergistic integration of technology mapping and retiming. The resulting delay is provably the smallest one that exists in the combined solution space of all possible structural mappings and retimings of the given subject graph. The implementation scales well and results in a delay reduction of up to 25% for both standard cells and FPGAs, while the delay reduction achieved by retiming before and after mapping does not exceed 10%, as shown in [23].

The general conclusion is that separating retiming prevents substantial delay gains, which can only be obtained when retiming is integrated with other steps in the synthesis flow.

This paper proposes several simplifications that allow for the expected delay gains due to the integration to be achieved while (a) computing only one combinational cut per node, (b) limiting retiming to move the registers within one timeframe, and (c) using any combinational mapper to derive the final mapping.

Future work will focus on the following improvements:

- *Register minimization during retiming.* Our current proof-of-concept implementation does not produce the final retiming with the minimum number of registers. One way to minimize the number of registers is to perform minimum-perturbation retiming, as described in [30][31].
- *Sequential critical path detection and restructuring.* It can be observed that the proposed computation of sequential arrival times allows for the notion of the combinational critical path to be extended to the sequential domain. Based on this, logic restructuring methods (for example, [25]) can be applied, resulting in additional delay reductions.
- *Leveraging recent progress in technology mapping*. Several new ideas have been recently proposed to improve state-of-the-art in combinational mapping [11]. These can be applied to sequential mapping described in this work, resulting in additional area reductions.

Acknowledgements

This research was supported in part by SRC Contract 3173.001 "Standardizing Boolean transforms to improve quality and runtime of CAD tools", the NSA grant "Novel methods for synthesis and verification in cryptanalytic applications" and donations from AMD, Siemens, and Synopsys.

References

[1] Berkeley Logic Synthesis and Verification Group. *ABC: A System for Sequential Synthesis and Verification*. December 2005 Release. http://www-cad.eecs.berkeley.edu/~alanmi/abc

[2] S. Bommu, N. O'Neill, and M. Ciesielski. "Retiming-based factorization for sequential logic optimization", *ACM TODAES*, Vol. 5(3), July 2000, pp. 373-398.

[3] S. Chatterjee and R. Brayton, "A new incremental placement algorithm and its application to congestion-aware divisor extraction", *Proc. ICCAD '04*, pp. 541-548.

[4] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping", *Proc. ICCAD '05*.

[5] P. Chong, Y. Jiang, S. Khatri, F. Mo, S. Sinha, and R. Brayton, "Don't care wires in logical/physical design", *Proc.IWLS'00*, pp.1-9.

[6] P. Chong and R. Brayton, "Characterization of feasible retimings", *Proc. IWLS '01*, pp. 1-6.

[7] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs", *IEEE Trans. CAD*, vol. 13(1), January 1994, pp. 1-12.

[8] J. Cong and C. Wu, "An efficient algorithm for performance-optimal FPGA technology mapping with retiming", *IEEE Trans. CAD*, vol. 17(9), Sep. 1998, pp. 738-748.

[9] J. Cong and C. Wu, "Optimal FPGA mapping and retiming with efficient initial state computation", *IEEE Trans. CAD*, vol. 18(11), Nov. 1999, pp. 1595-1607.

[10] J. Cong, C. Wu and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," *Proc. FPGA `99*, 29-35.

[11] L. Fan and C. Wu, "FPGA technology mapping with adaptive gate decomposition", *Proc. FPGA'23*, pp. 135-140.

[12] M. K. Ganai, A. Kuehlmann, "On-the-fly compression of logical circuits", *Proc. IWLS '00*.

[13] W. Gosti, S. Khatri and A. Sangiovanni-Vincentelli. "Addressing the timing closure problem by integrating logic optimization and placement", *Proc. ICCAD'01*, pp. 224-231.

[14] A. P. Hurst, P. Chong, A. Kuehlmann, "Physical placement driven by sequential timing analysis". *Proc. ICCAD '04*, pp. 379-386.

[15] IWLS 2005 Benchmarks. http://iwls.org/iwls2005/benchmarks.html

[16] Y. Jiang and S. Sapatnekar. "An integrated algorithm for combined placement and libraryless technology mapping," *Proc. ICCAD '99*.

[17] V. N. Kravets. *Constructive multi-level synthesis by way of functional properties*. Ph.D. Thesis, University of Michigan, 2001.

[18] Y. Kukimoto, R. Brayton, P. Sawkar, "Delay-optimal technology mapping by DAG covering", *Proc. DAC '98*, pp. 348-351.

[19] S. Malik, K.J. Singh, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance optimization of pipelined logic circuits using peripheral retiming and resynthesis", *IEEE YCAD*, Vol. 12(5), 1993, pp. 568-578.

[20] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Cutless FPGA mapping", *ERL Technical Report*, EECS Dept., UC Berkeley, 2007.

[21] A. Mishchenko, M. L. Case, R. K. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis", *Proc. ICCAD '08*, pp. 234-241.

[22] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to technology mapping for LUT-based FPGAs", *Proc. FPGA '06.*

[23] A. Mishchenko, S. Chatterjee, R. Brayton, and P. Pan, "Integrating logic synthesis, technology mapping, and retiming", *ERL Technical Report*, EECS Dept., UC Berkeley, April 2006.

[24] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts", *Proc. ICCAD '07*, 354-361.

[25] A. Mishchenko, R. Brayton, A. T. Calvino, and G. De Micheli, "Boolean decomposition revisited", *Submitted to IWLS'23*.

[26] P. Pan and C. L. Liu, "Optimum clock period FPGA technology mapping for sequential circuits", *Proc. DAC '96*, pp. 720-725.

[27] P. Pan and C.-C. Lin, "A new retiming-based technology mapping algorithm for LUT-based FPGAs", *Proc. FPGA '98*, pp. 35-42.

[28] P. Pan, "Performance-driven integration of retiming and resynthesis", *Proc. DAC '99*, pp. 243-246.

[29] M. Papaefthymiou, "Understanding retiming through maximum average-delay cycles", *Math. Syst. Theory*, No. 27, 1994, pp. 65-84.

[30] S. Ray, A. Mishchenko, R. K. Brayton, S. Jang, and T. Daniel, "Minimum-perturbation retiming for delay optimization". *Proc. IWLS'10*.

[31] D. P. Singh, V. Manohararajah, and S. D. Brown, "Incremental retiming for FPGA physical synthesis". *Proc. DAC '05*, pp. 433-438.

Table 1. Preliminary results of the integration of mapping and retiming for 6-input LUTs.

Design	FF	LUT	Level	FF	LUT	Level
des_perf	8808	7570	4	8808	7570	4
ethernet	10544	17591	9	11729	19319	6
mem_ctrl	1083	4191	9	2084	5008	8
pci_bridge32	3359	5378	7	5672	6496	6
systemcaes	670	2408	9	2090	3942	7
tv80	359	2429	12	1389	3417	9
usb_funct	1746	3693	6	2486	4441	5
vga_lcd	17079	28917	7	19499	35039	5
wb_conmax	770	13771	7	5816	17382	6
wb_dma	563	1105	8	747	1193	7
Geomean	1.000	1.000	1.000	1.928	1.219	0.817