Optimal FPGA Mapping and Retiming with Efficient Initial State Computation

Jason Cong and Chang Wu

***Abstract*—For sequential circuits with given initial states, new equivalent initial states must be computed for retiming, which unfortunately is NP-hard. In this paper, we present a novel *polynomial time algorithm for optimal field programmable gate array (FPGA) mapping with forward retiming* to minimize the clock period with efficient initial state computation. It considers forward retiming, initial state computation and mapping simultaneously. Our algorithm enables a new methodology of separating forward retiming from backward retiming. Since we guarantee to compute an optimal mapping with forward retiming solution, backward retiming can be performed as a preprocessing to try to push flip-flops (FF's) to primary inputs with consideration of only initial state computation. Thus, we can avoid the time-consuming iterations between retiming for clock period minimization and initial state computation. This algorithm compares very favorably to both conventional approaches of mapping followed by separate retiming and recent approaches of combined mapping with retiming, but without consideration of initial state computation [1], [2]. Our results show that our algorithm can reduce the clock period by 17.5% over conventional approaches of separate mapping with retiming. On the other hand, our algorithm can guarantee efficient initial state computation in the mapped and retimed circuits with only 2.8% increase in clock period over the optimal mapping with general retiming solutions, while the initial state computation for the latter solutions may need prohibitively long runtime for designs with over several hundred FF's.**

***Index Terms*—Field programmable gate array (FPGA), initial state, retiming, technology mapping.**

I. Introduction

RETIMING is a well-known optimization technique for sequential circuits. Many research results have been published in literature recently on combining optimal retiming presented in [3] with logic optimization, circuit partitioning and technology mapping [4]–[6]. For lookup-table (LUT)-based field programmable gate array (FPGA) designs, a significant advancement was made by Pan and Liu [1] on optimal simultaneous technology mapping with retiming for clock period minimization. Later on, Cong and Wu [2], [7] proposed a much more efficient optimal mapping with retiming algorithm which achieves 1000 times speedup over [1]. However, limited progress has been made on equivalent initial state computation for retiming.

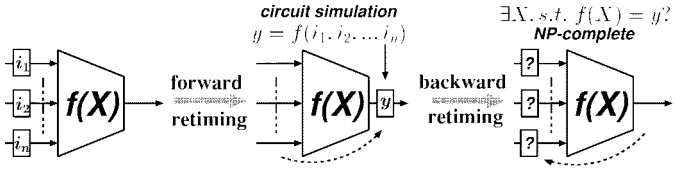


Fig. 1. Retiming and initial state computation. Each small rectangle represents a FF.

A general retiming procedure usually involves two kinds of flip-flop (FF) movements. *Forward retiming* (FRT) moves FF's from nodes' inputs to outputs, while *backward retiming* moving FF's in the opposite direction. As shown in Fig. 1, the equivalent initial state of forward retiming can be computed easily using circuit simulation. The equivalent initial state for backward retiming, however, requires solving a satisfiability problem which is NP-complete in general.

Touati and Brayton [8] proposed an initial state computation algorithm based on STG (state transition graph) traversal. Their algorithm can compute equivalent initial states under the condition that there exists a loop from the initial state to itself in the STG of the original circuit. However, unless provided externally, to determine the existence of such a loop needs to search the STG which may have an exponential number of nodes. Although representing the reset logic explicitly can guarantee the existence of such a loop and avoid the STG traversal, the extra cost of the reset logic separated from FF's after retiming can be high.[1] Furthermore, the reset logic may restrict possible retiming of the original circuit, thus, reduce the potential of retiming optimization. A recent paper by Singhal *et al.* [9] showed that by retiming the reset logic together with FF's will not affect the possibility of retiming and there is no extra cost on the area of the circuit. But their approach is applicable for only forward retiming, but not general retiming.

Several heuristics for initial state computation of a given retiming based on automatic test pattern generation (ATPG) techniques have been proposed recently [10]–[12]. For example, Stok *et al.* [11] proposed to reduce the initial state computation effort by minimizing the maximum retiming value which represents the maximum number of FF's to be moved backward across every gate. ATPG-based justification

Manuscript received February 13, 1999; revised July 26, 1999. This work was supported in part by the National Science Foundation (NSF) under Young Investigator Award MIP9357582 and in part grants from Xilinx, Lucent Technologies, and Quickturn Design Systems under the California MICRO program. This paper was recommended by Associate Editor M. Papaefthymiou.

The authors are with the Computer Science Department, University of California, Los Angeles, CA 90095 USA (e-mail: cong@cs.ucla.edu; changwu@cs.ucla.edu).

Publisher Item Identifier S 0278-0070(99)09468-3.

[1] Most FF's have built-in reset logic. But we can represent such FF's as generic FF's plus reset logic. After retiming, however, the reset logic may be separated from the generic FF's and cannot be repacked together. As a result, extra LUT's may need to implement those reset logic.

was used to compute equivalent initial states for backward retiming. Possible iterations of retiming and initial state computation may still be needed. However, ATPG itself is also NP-complete [13], [14]. Furthermore, we have observed that minimizing the maximum retiming value as formulated in [11] might not always lead to a simpler initial state computation problem. Our experience shows that unless a pure forward retiming solution can be found, the effort of initial state computation depends on the number of nodes involved in backward retiming, instead of the maximum retiming value.

Maheshwari and Sapatnekar [12] proposed to compute an upper bound on each node's retiming value based on ATPG techniques such that the equivalent initial state can be computed for a retiming within this bound. In general, the larger values of the upper bounds, the better the retiming solution. However, to increase one node's retiming upper bound value may need to reduce another node's retiming upper bound value and it is hard to tradeoff the values of different nodes to achieve the best possible solution. As a result, the authors proposed to perform retiming for many sets of upper bounds and choose the best possible results. Clearly, this can be time consuming in practice.

To better understand the difficulty of the initial state computation problem and the capability of existing initial state computation algorithms, we tested the retiming package in SIS [15] which computed initial states for retiming based on the algorithm in [8]. For circuit s9234.1 in the MCNC benchmark suite with 135 FF's and 1293 two-input gates, SIS could not finish the initial state computation for an optimal retiming after running 4 hours on a Sun ULTRA2 workstation with 256-MB memory. We see that the initial state computation problem being a major obstacle of preventing wide-spread use of the retiming technique in practice.

One common problem of existing retiming algorithms is lacking consideration of whether an equivalent initial state can be computed when computing a retiming. When failing to find an equivalent initial state for a retiming, most algorithms need to backtrack to compute another retiming with almost no assurance if an equivalent initial state can be computed for the new retiming. As a result, many iterations between retiming and initial state computation may be needed and the computation time can be very long.

In order to avoid high time complexity of initial state computation for *general* retiming, we propose to perform LUT-based FPGA mapping with forward retiming. Our main contribution is the development of the first polynomial time optimal algorithm for FPGA mapping with forward retiming, which has immediate benefit of guaranteed equivalent initial state computation in linear time. With this algorithm, we can try to push FF's back to primary inputs as much as possible to enlarge the solution space of mapping with forward retiming and achieve better results. One important advantage of our approach is that we only need to focus on initial state computation without considering the effect of retiming on clock period minimization during the preprocessing of backward retiming. As a result, we can still achieve effective retiming for circuit optimization without time-consuming iterations between retiming and initial state computation, slow STG traversal, or any extra reset logic in existing approaches [8], [10], [12], [16]. Our algorithm compares very favorably to both conventional approaches of separate mapping with retiming and recent approaches of combined mapping with retiming, but without initial state computation [1], [2]. It is also applicable to FPGA devices with prefixed initial state settings.[2]

The remainder of the paper is organized as follows. The problem formulation and definitions are presented in Section II. Our algorithm is presented in Section III. A post-processing for FPGA's with prefixed initial state settings is presented in Section IV. Our experimental results are presented in Section V followed by conclusions and future work in Section VI. A preliminary version of this work was presented at DAC'98 [17].

II. Problem Formulation and Definitions

Given a sequential circuit, the technology mapping problem for K-LUT-based FPGA's is to construct an *equivalent* circuit consisting of K-LUT's and flip-flops (FF's), such that the two circuits generate the same output sequence for any input sequence, starting from their individual initial states, respectively. The clock period of a circuit is the maximum combinational path delay. In this paper, we shall explore the following problem.

Problem 1: Given a sequential circuit with initial state s_0, find an equivalent LUT circuit with initial state s_0^r and minimum clock period under forward retiming.

As in [1] and [2], instead of solving the optimization Problem 1 directly, we shall solve its decision version and then binary search for the minimum clock period.

Problem 2: Given a sequential circuit with initial state s_0 and target clock period Φ, find an equivalent LUT circuit with initial state s_0^r and clock period of no more than Φ under forward retiming.

As proposed in [3], we use retiming graph to represent sequential circuits. The retiming graph $G(V, E, W)$ of a sequential circuit is a directed graph, where V is the set of nodes, E is the set of edges, and W is the set of edge weights. Each node in V represents a gate, a primary input (PI), or a primary output (PO) in the original circuit. Each edge $e(u, v)$ in E represents a directed connection from a node u to a node v. The *weight* $w(e)$ of an edge e is the number of FF's on the connection represented by the edge. The *path weight* $w(p)$ of a path p is the sum of all edge weights on the path. Under unit-delay mode, *node delay* $d(v)$ is one for internal nodes v, or zero for PI's or PO's. *Path delay* $d(p)$ of a path p is the sum of all node delays on the path. For simplicity, we consider the unit-delay model and synchronous sequential circuits with single-phase clock and edge-triggered FF's.[3] Depending on the context, we use G to represent both V and E.

[2] For example, the initial state of FF's on Xilinx XC5200 FPGA's can only be set to zero.

[3] For more complex delay models, such as the fanout-based nominal delay model, the FPGA mapping of combinational circuits for delay minimization has already been shown to be NP-hard [18]. We think, however, the techniques presented in this paper can be extended for more accurate delay models as a heuristic.

For a given retiming, the *retiming value* $\mathcal{R}(v)$ of node v is the number of FF's moved backward across v. A negative retiming value of a node means moving FF's forward across the node. A retiming is represented by a set of retiming values $\{\mathcal{R}(v)|\ \forall v \in G\}$, where $\mathcal{R}(v)$ is zero for every PI or PO which means that no FF's moved across either PI's or PO's.

As in [2], for a target clock period Φ, we define the *edge length* [denoted length(e)] of an edge $e(u, v)$ to be $d(v) - \Phi \cdot w(e)$. The *path length* [denoted length(p)] is $\sum_{e\in p}$ length(e). In an LUT network M, a node's l-value $l_M(v)$ is defined to be the maximum path length of all paths from PI's to v, which represents the node arrival time after retiming. (Here, we assume that every node is reachable from at least one PI. In cases that there exist nodes not reachable from any PI's, the l-values of those nodes are not well defined. Extension to the following theory and our algorithm will be presented in Section III-F.) It is not difficult to prove the following.

Theorem 1: Given an LUT network M and a target clock period Φ, the clock period of M under forward retiming is no more than Φ, if and only if the l-values $l_M(v)$ are no more than Φ for every LUT v in M.[4]

Proof: ($\Rightarrow$) Let $\{\mathcal{R}(v)|\ \forall v \in M, \mathcal{R}(v) \le 0\}$ be a forward retiming to achieve Φ. After retiming, every combinational path has delay of no more than Φ. Thus, for any path p: PI $\rightsquigarrow v$, the path length after retiming satisfies the inequality that length$^r(p) = d(p) - \Phi \cdot w^r(p) = d(p) - \Phi \cdot (w(p) + \mathcal{R}(v)) \le \Phi$. Since $\mathcal{R}(v) \le 0$, the path length in M before retiming length$(p) =$ length$^r(p) + \Phi \cdot \mathcal{R}(v)$ is also no more than Φ. As a result, $l_M(v) = \max\{\text{length}(p)|\ \forall p\text{: PI} \rightsquigarrow v\} \le \Phi$.

($\Leftarrow$) Since $l_M(v) \le \Phi$, we perform the forward retiming defined as

$$\mathcal{R}(v) = \begin{cases} 0, & \text{if } v \text{ is a PI or PO} \\ \left\lceil \dfrac{l_M(v)}{\Phi} \right\rceil - 1, & \text{otherwise.} \end{cases}$$

We prove that the retiming will reduce the clock period of M to Φ. First, we show that this is a legal retiming such that the edge weight of every edge $e(u, v)$ after retiming is nonnegative. By definition of the l-value, we have that $l_M(v) \ge l_M(u) + \text{length}(e) = l_M(u) - \Phi \cdot w(e) + 1$. Without loss of generality, we assume both u and v are LUT's. (Notice that they may also be PI's or PO's with delay of 0. The proof for this case is very similar and is omitted here.) The retimed edge weight satisfies the inequality that $w^r(e) = w(e) + \mathcal{R}(v) - \mathcal{R}(u) = w(e) + (\lceil l_M(v)/\Phi \rceil - 1) - (\lceil l_M(u)/\Phi \rceil - 1) \ge w(e) + \lceil (l_M(u) - \Phi \cdot w(e) + 1)/\Phi \rceil - \lceil l_M(u)/\Phi \rceil \ge 0$.

Second, we show that $w^r(p) \ge 1$ for any path p: $u \rightsquigarrow v$ with $d(p) > \Phi$. For such a path p, since $d(p)$ is an integer, we have that $d(p) \ge \Phi + 1$ and $l_M(u) + \text{length}(p) = l_M(u) + d(p) - 1 - \Phi \cdot w(p) \le l_M(v)$. Therefore

$$\begin{aligned} w^r(p) &= w(p) + \mathcal{R}(v) - \mathcal{R}(u) \\ &\ge w(p) + \left(\left\lceil \frac{l_M(u) + d(p) - 1 - \Phi \cdot w(p)}{\Phi} \right\rceil - 1 \right) \\ &\quad - \left(\left\lceil \frac{l_M(u)}{\Phi} \right\rceil - 1 \right) \\ &= \left\lceil \frac{l_M(u) + d(p) - 1}{\Phi} \right\rceil - \left\lceil \frac{l_M(u)}{\Phi} \right\rceil \\ &\ge \left\lceil \frac{l_M(u) + \Phi}{\Phi} \right\rceil - \left\lceil \frac{l_M(u)}{\Phi} \right\rceil \\ &= \left\lceil \frac{l_M(u)}{\Phi} \right\rceil + 1 - \left\lceil \frac{l_M(u)}{\Phi} \right\rceil \\ &= 1. \end{aligned}$$

■

Definition 1: Given a mapping with forward retiming solution M and a target clock period Φ, the number of FF's moved forward across LUT v from each input of v, denoted $r_M(v)$, is called the *forward retiming value* of v in M. The s-value is defined to be $s_M(v) = l_M(v) - \Phi \cdot r_M(v)$, where $l_M(v)$ is the l-value of v in M.

The s-value represents the node arrival time before forward retiming. The reason of considering the s-value is that it is easier to compute than the l-value because it does not depend on retiming, but the latter does.

According to Theorem 1, we have the following.

Corollary 1: A mapping solution M has a clock period of no more than a given Φ under forward retiming, if and only if $s_M(v) + \Phi \cdot r_M(v) \le \Phi$ for every LUT root v in M.

Definition 2: Given a circuit and a target clock period Φ, if there exists a FRT mapping solution with clock period of no more than Φ under forward retiming (called a *feasible FRT mapping solution*), the s-label $S(v)$ for node v is defined to be the minimum $s_M(v)$ among *all feasible FRT mapping solutions* and, the r-label $R(v)$ is defined to be the minimum $r_M(v)$ *among all feasible FRT mapping solutions* with $s_M(v) = S(v)$. The node label pair is defined to be a two-tuple $(S(v), R(v))$.

According to Corollary 1, we can solve Problem 2 by computing the node label pairs and checking if the inequality of $S(v) + \Phi \cdot R(v) \le \Phi$ holds for every node. The minimum clock period $\Phi_{\min}$ of the optimal mapping solution can be computed with binary search. After getting $\Phi_{\min}$ and the corresponding label pairs, we construct a mapping solution and perform a post-processing of forward retiming step to achieve $\Phi_{\min}$. Minimizing $S(v)$ is to minimize the longest path delay from PI's to v after mapping, thus, reduce the clock period after retiming. Minimizing $R(v)$ is to push forward as few FF's as possible for each node v to leave the maximum freedom to subsequent forward retiming step, because those FF's pushed forward can no longer be pushed back. On the other hand, minimizing $S(v)$ may increase $R(v)$, because we need to construct larger LUT's which may include more FF's. Our algorithm can simultaneously minimize both $S(v)$ and $R(v)$ to compute optimal FRT mapping solutions.

[4]Note that this result is similar to [1, Theorem 3]. The difference is that both forward and backward retimings were allowed in [1], while only forward retiming is allowed in our case. Consequently, [1, Theorem 3] checks only the l-values of PO's, while we need to check that of every node.

As in [1], [2], and [19], we assume that the initial circuit is K bounded. (When a circuit is not K bounded, we can use gate decomposition algorithms as presented in [20]–[22] to decompose gates with more than K fanins.)

Before proceeding to present our algorithm, we first introduce the definition of K-cuts. In a directed graph with one sink and one source, a *cut* $(X, \overline{X})$ is a partition of the graph such that the sink is in $\overline{X}$ and the source is in X. The *node cut-set* $V(X, \overline{X})$ is the set of nodes in X that are connected directly to nodes in $\overline{X}$. If $|V(X, \overline{X})| \leq K$, a cut $(X, \overline{X})$ is called a *K-feasible cut*, or a *K-cut* in short. A cut is a *min-cut*, if $|V(X, \overline{X})|$ is minimal. To determine the existence of a K-cut, one can compute a maximum flow from the source to the sink and check whether its value is larger than K. This process is called *K-cut computation*.

III. The TurboMap-frt Algorithm

Our algorithm, named TurboMap-frt, is to compute optimal mapping with forward retiming solutions for synchronous sequential circuits with given initial states to minimize the clock period. It performs binary search on the clock period from one to an upper bound computed by existing algorithms, for example, FlowMap [19]. For a given clock period, a procedure named FRTcheck is used to decide whether there exists a feasible solution through label computation. An overview of the label computation procedure can be described as follows. First, we assign an initial lower bound on the value of each node label pair. Then, we iteratively increase those lower bounds until they all converge to the values of node label pairs if there exists a feasible solution, or stop if we conclude there is no feasible solution.

A. Review of Expanded Circuit for LUT Representation with Backward Retiming

To compute an optimal solution, we need to search all possible LUT's for every node under node duplication and forward retiming. Pan and Liu [1] proposed to search all possible LUT's rooted at a node which can be formed under *backward* retiming on the expanded circuit of the node. An *expanded circuit* $\mathcal{E}_v^l$ of node v with *control number* l is a directed acyclic graph (DAG) rooted at v and constructed from the original retiming graph G with node replication, such that all paths from a node in $\mathcal{E}_v^l$ to the root have the same number of FF's. If a replication of a node u passes w FF's before reaching the root v in $\mathcal{E}_v^l$, it is denoted as u^w [1]. The *control number* l is the shortest distance (in terms of the number of edges) between the root and each leaf u^w if u is an internal node in G.[5] For example, Fig. 2(b)–(e) shows four expanded circuits $\mathcal{E}_c^0$, $\mathcal{E}_c^1$, $\mathcal{E}_c^2$, and $\mathcal{E}_c^3$ of node c. With *backward retiming* Pan and Liu [1] showed that any K-LUT of a node v can be derived from a K-cut of $\mathcal{E}_v^{Kn}$ and any K-cut of $\mathcal{E}_v^{Kn}$ can be used to derive a K-LUT of v, where n is the number of nodes in the original circuit.

[5]The control number does not apply to any leaf u^w if u is a PI in G.

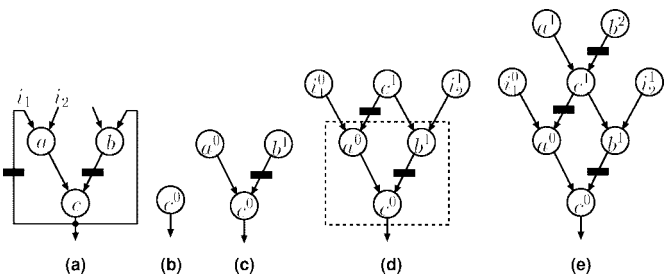


Fig. 2. Expanded circuits for LUT representation with backward retiming.

B. Expanded Circuit Construction for LUT Representation with Forward Retiming

With forward retiming only, the one-to-one correspondence between K-LUT's and K-cuts in $\mathcal{E}_v^{Kn}$ introduced in Section III-A no longer holds. As shown in Fig. 2(d), to pack all the nodes in the dotted box into an LUT, we have to move *backward* the FF on the fanout edge of b^1 to its fanin edges, because the retiming to push the FF forward to the fanout edge of c^0 is illegal as we have no FF to move forward on edge $e(a^0, c^0)$. Thus, the corresponding K-cut does not correspond to any K-LUT for FRT mapping.

In the following, we propose to construct a smaller expanded circuit for every node based on its *maximum forward retiming value* so that every K-LUT in an FRT mapping solution corresponds to a K-cut on the expanded circuit and, vise versa.

Definition 3: The *maximum forward retiming value* $\text{frt}(v)$ of a node v is the maximum number of FF's which can be moved forward from the (transitive) fanins of v to the output of v.

Lemma 1: For a node v in a retiming graph, $\text{frt}(v)$ is the minimum path weight from PI's to v, i.e., $\text{frt}(v) = \min\{w(p)|\ \forall p\text{: PI} \rightsquigarrow v\}$.

Proof: First, $\text{frt}(v) \leq \min\{w(p)|\ \forall p\text{: PI} \rightsquigarrow v\}$. If otherwise, assume there is a path p: PI $\rightsquigarrow v$ and $w(p) < \text{frt}(v)$. After moving $\text{frt}(v)$ FF's forward across v, the new path weight $w^r(p) = w(p) - \text{frt}(v) < 0$, which means the retiming is illegal.

Let $m_u = \min\{w(p)|\ \forall p\text{: PI} \rightsquigarrow u\}$ for a node u. Clearly, $m_{u_1} + w(e) \geq m_{u_2}$ for any edge $e(u_1, u_2)$. We shall show that we can always push m_v FF's forward across v for every node v. To prove this, we simply need to perform a forward retiming $\{\mathcal{R}(u) = -m_u|\ \forall u \in G\}$ and show it is legal, i.e., $w^r(e) \geq 0$ for every edge e. For any edge $e(u_1, u_2) \in G$, $w^r(e) = w(e) + \mathcal{R}(u_2) - \mathcal{R}(u_1) = w(e) - m_{u_2} + m_{u_1} \geq 0$. So it is a legal forward retiming. ■

Since the edge weight is nonnegative, the maximum forward retiming values of all the nodes in a retiming graph can be computed in $O(n^2)$ time with Dijkstra's shortest path algorithm [23].

Now we define a set of expanded circuits of a node v for FRT mapping. The expanded circuit $\mathcal{F}_v^i$ for a given upper bound i of forward retiming value of a node v is a sub-DAG of $\mathcal{E}_v^{Kn}$ with root v^0 such that, u^w is an internal node of $\mathcal{F}_v^i$, if and only if u^w is an internal node of $\mathcal{E}_v^{Kn}$, and $w \leq i$; u^w is a leaf of $\mathcal{F}_v^i$, if and only if u^w is either a leaf of $\mathcal{E}_v^{Kn}$ or a fanin of an internal node of $\mathcal{F}_v^i$ with $w > i$. For example,

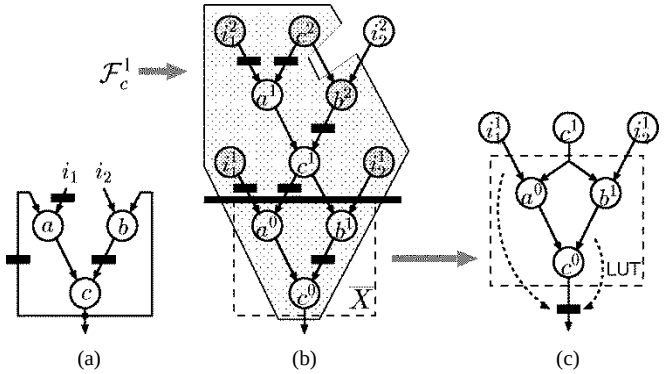


Fig. 3. LUT formation for FRT mapping based on the expanded circuits. Edge $e(c^2, b^2)$ shown in (b) does not belong to the expanded circuit $\mathcal{F}_c^1$, i.e., b^2 is a leaf node in $\mathcal{F}_c^1$.

the shaded area of Fig. 3(b) represents $\mathcal{F}_c^1$ for the circuit shown in Fig. 3(a), where the heavy shaded nodes are leaves of $\mathcal{F}_c^1$.

Definition 4: For a K-cut $(X, \overline{X})$ of an expanded circuit of node v, the *cut-weight* is defined to be $w(X, \overline{X}) = \max\{w | \forall u^w \in \overline{X}\}$, which represents the minimum forward retiming value needed to push all the FF's inside $\overline{X}$ to the fanout edge of v.

Obviously, the cut-weight of any K-cut of $\mathcal{F}_v^i$ is no more than i. Let us assume that the initial states of FF's on different fanout edges of a node are the same, we have the following theorem.[6]

Theorem 2: Every K-feasible cut $(X, \overline{X})$ of $\mathcal{F}_v^{\text{frt}(v)}$ corresponds to a K-LUT rooted at v under possible node duplication and forward retiming, where all the leaves are in X and the root is in $\overline{X}$. On the other hand, any K-LUT in an FRT mapping solution corresponds to a K-feasible cut of $\mathcal{F}_v^{\text{frt}(v)}$.

Proof: Obviously, every K-cut $(X, \overline{X})$ of $\mathcal{F}_v^{\text{frt}(v)}$ corresponds to a K-LUT in FRT mapping, because $w(X, \overline{X}) \leq \text{frt}(v)$ and we can always push all the FF's within $\overline{X}$ forward to the output of v. Now we prove that the K-cut $(X, \overline{X})$ corresponding to a K-LUT rooted at v in an FRT mapping solution must be included in $\mathcal{F}_v^{\text{frt}(v)}$, or in other words, it must be that $\overline{X} \subset \mathcal{F}_v^{\text{frt}(v)}$. Clearly $w(X, \overline{X}) \leq \text{frt}(v)$, because we have to push all the FF's within $\overline{X}$ forward to the fanout of v, which means $w \leq \text{frt}(v)$ for any $u^w \in \overline{X}$. By definition of $\mathcal{F}_v^{\text{frt}(v)}$, any $u^w \in \overline{X}$ is an internal node of $\mathcal{F}_v^{\text{frt}(v)}$. On the other hand, any leaf u^w of $\mathcal{F}_v^{\text{frt}(v)}$ cannot be included in $\overline{X}$, because either $w > \text{frt}(v)$ or u^w is also a leaf of $\mathcal{E}_v^{Kn}$ such that $u^w \notin \overline{X}$. (Notice that it is proved in [1] that for any K-cut $(X, \overline{X})$ corresponding to a K-LUT, any leaf of $\mathcal{E}_v^{Kn}$ is not contained in $\overline{X}$.) This concludes our proof. ■

Let us look at the circuit shown in Fig. 3(a). Notice that its only difference with the circuit in Fig. 2(a) is one extra FF on edge (i_1, a) which makes $\text{frt}(c) = 1$. Now the 3-cut $(X, \overline{X})$ shown in Fig. 3(b) can form a 3-LUT as shown in Fig. 3(c).

With the assumption that every edge has at most one FF, $\mathcal{E}_v^{Kn}$ has $O(Kn^2)$ nodes and $O(K^2n^2)$ edges [1]. Clearly, $\mathcal{F}_v^{\text{frt}(v)}$ has no more than $O(Kn^2)$ nodes and $O(K^2n^2)$ edges as it is a sub-DAG of $\mathcal{E}_v^{Kn}$. Practically, however, using the technique of efficient K-cut computation on *partial flow networks* proposed in [2], the expanded circuits needed to be constructed always have far less than n nodes and Kn edges for all the benchmarks we have tested.

[6] In case the initial states of FF's on different fanout edges of a node are not the same, the theorem still holds after a simple buffer insertion. The detail will be presented at the end of this section.

```
FRTcheck(G(V,E,W), Φ)
1   assign initial lower-bound (0,0) for each PI and
      (−∞,0) for the other nodes
2   converge ← FALSE, i ← 0
3   sort all nodes from PIs to POs with DFS
      (depth-first search)
4   while (not converge and i ≤ n²) do {
5     converge ← TRUE
6     for each node v from PIs to POs do {
7       (s_new(v), r_new(v)) ←LabelUpdate(v, Φ)
8       if (s_new(v) > s(v)) then {
9         s(v) ← s_new(v)
10        converge ← FALSE
11      }
12    }
13    i ← i + 1
14  }
15  if (converge) then return(TRUE)
16  else return(FALSE)
```

Fig. 4. Label computation for a target clock period Φ.

C. Iterative Label Computation

For a target clock period Φ, we compute all node label pairs of a circuit to decide the existence of a feasible FRT mapping solution. The algorithm for this operation is named FRTcheck. It assigns a pair of lower bounds, denoted $(s(v), r(v))$, on the value of the label pair for every node and iteratively updates the values of the lower bounds until they all converge to the values of the node label pairs, i.e., $s(v) = S(v)$ and $r(v) = R(v)$ for every node v. The initial values of the lower bounds are $(0, 0)$ for the PI's and $(-\infty, 0)$ for the other nodes. If the lower bounds cannot converge after n^2 *iterations* (one iteration is the process of updating the lower bound of every node's label pair once), FRTcheck stops the computation and concludes that no feasible FRT mapping solution exists for the given Φ. The pseudocode of the FRTcheck algorithm is shown in Fig. 4. In the remainder of this section, we will present the details of the LabelUpdate procedure (line 7 in Fig. 4), which is to update the lower bound of the label pair for a single node once based on the current lower bound values of all nodes.

Definition 5: Given a set of current lower bounds $(s(v), r(v))$, where $s(v) \leq S(v)$, and $r(v) \leq R(v)$ if $s(v) = S(v)$, the *cut-height* of a K-cut $(X, \overline{X})$ in the expanded circuit $\mathcal{F}_v^{\text{frt}(v)}$ of node v is

$$h(X, \overline{X}) = \max_{\forall K\text{-cut}(X, \overline{X}) \text{ on } \mathcal{F}_v^{\text{frt}(v)}} \{s(u) - \Phi \cdot w + 1 | \forall u^w \in V(X, \overline{X})\}.$$

To compute a tighter lower bound $s_{\text{new}}(v)$ for a node v, we first compute a value $\varsigma(v) = \max\{s(u) - \Phi \cdot w(e) | \forall e(u, v) \in G\}$, where G is the retiming graph of the original circuit. If

```
LabelUpdate(v, Φ)
1   compute ς(v)
2   if ς(v) < s(v) then
3       s_new(v) ← ς(v), r_new(v) ← 0
4   else if does not exist a K-cut with height of
        no more than ς(v) then
5       s_new(v) ← ς(v) + 1, r_new(v) ← 0
6   else {
7       compute a K-cut with the minimum weight
        w_min and h(X, X̄) ≤ ς(v)
8       if ς(v) + Φ · w_min ≤ Φ then
9           s_new(v) ← ς(v), r_new(v) ← w_min
10      else s_new(v) ← ς(v) + 1, r_new(v) = 0
11  }
```

Fig. 5. Label pair update for a single node v.

$\varsigma(v) < s(v)$, we keep $s_{\text{new}}(v)$ to be $s(v)$ and set $r_{\text{new}}(v)$ to be 0. If $\varsigma(v) \geq s(v)$, we decide whether there exists a K-cut of $\mathcal{F}_v^{\text{frt}(v)}$ with height of no more than $\varsigma(v)$, based on the max-flow computation on $\mathcal{F}_v^{\text{frt}(v)}$. (In fact, for efficiency consideration, we perform the max-flow K-cut computation on a partial flow network of $\mathcal{F}_v^{\text{frt}(v)}$ with much smaller size as in [2].) If there does not exist such a K-cut, we select a cut with $\overline{X} = \{v\}$ and set $s_{\text{new}}(v) = \varsigma(v)+1$. Obviously, $r_{\text{new}}(v) = 0$ in this case, because $\overline{X}(=\{v\})$ does not include any FF's.

If, however, there does exist such a K-cut with height of no more than $\varsigma(v)$, we compute a K-cut with minimum cut-weight (see Definition 4) $w_{\min}$ and height of no more than $\varsigma(v)$ by binary searching the cut-weight from zero to frt(v) as follows.

For a given $w \in [0, \text{frt}(v)]$, to decide whether there is a K-cut with height of no more than $\varsigma(v)$ and cut-weight of no more than w, we first construct an expanded circuit $\mathcal{F}_v^w$ and then, decide whether there exists a K-cut on $\mathcal{F}_v^w$ with height of no more than $\varsigma(v)$. If there exists such a K-cut, clearly, it is a K-cut with height of no more than $\varsigma(v)$ and cut-weight of no more than w.

Let $(X, \overline{X})$ be such a K-cut with height of $\varsigma(v)$ and cut-weight of $w_{\min}$ computed above. If $\varsigma(v) + \Phi \cdot w_{\min} \leq \Phi$, we update the new lower bound to be $(\varsigma(v), w_{\min})$. Otherwise, we update the new lower bound to be $(\varsigma(v) + 1, 0)$. Recall that if the minimum height of all K-cuts is larger than $\varsigma(v)$, we also update the new lower bound to be $(\varsigma(v) + 1, 0)$. This concludes the LabelUpdate procedure. The pseudocode is shown in Fig. 5.

Theorem 3: For a sequential circuit which has a feasible FRT mapping solution for a target clock period Φ, starting from the initial lower bounds $(0, 0)$ for PI's and $(-\infty, 0)$ for the other nodes, the inequalities, 1) $s(v) \leq S(v)$ and 2) $r(v) \leq R(v)$ when $s(v) = S(v)$, hold all the time after any number of iterations of label update.

The proof of this theorem is presented in the Appendix.

This theorem shows that $s(v)$ and $r(v)$ are truly lower bounds on the values of $S(v)$ and $R(v)$. Furthermore, we shall prove in the Appendix that $s(v)$ will monotonically increase and both $s(v)$ and $r(v)$ will converge to $S(v)$ and $R(v)$, respectively, in n^2 iterations, if there exists a feasible solution for the target clock period. If, on the other hand, both $s(v)$ and $r(v)$ converge to finite values for all the nodes, we shall show that we can form an FRT mapping solution with clock period of no more than the target value in Section III-D. Consequently, we conclude that TurboMap-frt can compute an optimal FRT mapping solution with the minimum clock period in polynomial time.

D. Mapping Generation with Forward Retiming and Initial State Computation

After computing the minimum clock period $\Phi_{\min}$ with binary search and obtaining the label pairs $(S(v), R(v))$ of all nodes v, the last step of our algorithm is to generate the mapping solution based on the K-cuts computed during the label computation and perform forward retiming with initial state computation.

First, we get all the LUT roots in the mapping solution based on the K-cuts computed during the label computation. Obviously, all the PO's of the original circuit are LUT roots. If v is an LUT root, all the nodes in the node-cut set of the K-cut of v are LUT roots. The complete set of LUT roots can be computed as follows: starting with a first-in-first-out (FIFO) queue including all the PO's, we repeatedly extract nodes from the head of the queue until the queue is empty. For each node extracted from the queue, we mark it as an LUT root and put all the nodes in its node-cut set to the end of the queue.

Second, we create a new equivalent network by connecting the K-feasible cones $\overline{X}_v$ of the K-cuts $(X_v, \overline{X}_v)$ of those LUT roots v. Then, we compute a forward retiming for all nodes v on the new network as follows:

$$\mathcal{R}(v) = \begin{cases} 0, & \text{if } v \text{ is a PI or PO} \\ \left\lceil \dfrac{S(v)}{\Phi_{\min}} \right\rceil - 1, & \text{if } v \text{ is an LUT root} \\ \mathcal{R}(u) + w, & \text{if } v^w \in \text{LUT}_u, \\ & \text{where } u \text{ is an LUT root.} \end{cases}$$

After retiming all the FF's within each $\overline{X}_v$ will be pushed forward outside the $\overline{X}_v$, because $w^r(u^w \rightsquigarrow v^0) = w + \mathcal{R}(v) - \mathcal{R}(u) = 0$ for any $u^w \in \overline{X}_v$. Then, we collapse each $\overline{X}_v$ and put it into a K-LUT to form the final mapping solution which has a clock period of no more than $\Phi_{\min}$.

Lemma 2: The retiming defined above is a legal forward retiming to achieve clock period $\Phi_{\min}$.

Proof: First, we prove it is a forward retiming. Let v be a LUT root. Since $S(v) \leq \Phi_{\min}$, $\mathcal{R}(v) = \lceil S(v)/\Phi_{\min} \rceil - 1 \leq 0$. If v^w is an internal node of LUT$_u$, we prove that $\mathcal{R}(u) \leq -R(u)$ and then $\mathcal{R}(v) \leq 0$. Since $S(u) + \Phi_{\min} \cdot R(u) \leq \Phi_{\min}$ based on the definition of the node label pair and Corollary 1, we have that $\lceil (S(u) + \Phi_{\min} \cdot R(u))/\Phi_{\min} \rceil \leq 1 \Rightarrow \mathcal{R}(u) = \lceil S(u)/\Phi_{\min} \rceil - 1 \leq -R(u)$. Furthermore, for any $v^w \in \text{LUT}_u$ we have that $w \leq w(\overline{\text{LUT}}_u, \text{LUT}_u) \leq R(u)$. As a result, $\mathcal{R}(v) = \mathcal{R}(u) + w \leq -R(u) + R(u) = 0$.

After retiming, every K-feasible cone $\overline{X}$ will become a pure combinational block and can be collapsed into a K-LUT. We shall prove that the clock period of the LUT network is no more than $\Phi_{\min}$ in two steps.

1) *The Retiming is Legal*: Suppose LUT_u rooted at u is a fanin of LUT_v rooted at v and the path from u to v has w FF's in the original circuit, $S(v) \geq S(u) - \Phi_{\min} \cdot w + 1$ and $\lceil S(v)/\Phi_{\min} \rceil \geq \lceil S(u)/\Phi_{\min} \rceil - w$. As a result, the new edge weight between the two LUT's after retiming is $w^r(\text{LUT}_u, \text{LUT}_v) = w - \mathcal{R}(u) + \mathcal{R}(v) = w - (\lceil S(u)/\Phi_{\min} \rceil - 1) + (\lceil S(v)/\Phi_{\min} \rceil - 1) \geq 0$.
2) *The Retimed Circuit has a Clock Period of no More than* $\Phi_{\min}$: For any path p: $\text{LUT}_u \rightsquigarrow \text{LUT}_v$ with delay $d(p) > \Phi_{\min}$, we prove that $w^r(p) \geq 1$. Since both $d(p)$ and $\Phi_{\min}$ are integral, it must be that $d(p) \geq \Phi_{\min} + 1$. Since $S(v) \geq S(u) + \text{length}(p) = S(u) - \Phi_{\min} \cdot w(p) + d(p) - d(\text{LUT}_u)$ and $d(\text{LUT}_u) = 1$, we have that $\mathcal{R}(v) \geq \mathcal{R}(u) - w(p) + \lceil (d(p) - 1)/\Phi_{\min} \rceil \geq \mathcal{R}(u) - w(p) + 1$. It means that $w^r(p) = w(p) + \mathcal{R}(v) - \mathcal{R}(u) \geq 1$.

■

In the constructed network, since the retiming is a forward retiming, the equivalent initial state can be computed with circuit simulation. Since each K-input node can be simulated in $O(K)$ time,[7] for a K bounded circuit with n gates and $O(Kn)$ edges, the first step of getting LUT roots can be done in $O(Kn)$ time. There are $O(n)$ LUT's in the constructed LUT network. Each K-LUT includes $O(Kn)$ nodes as proved in [2, Theorem 4]. The total number of nodes need to perform forward retiming is $O(Kn^2)$. Furthermore, $|\mathcal{R}(v)| = O(n)$. The runtime of forward retiming is $O(Kn^2 \cdot n \cdot K) = O(K^2n^3)$. However, the number of node in each K-LUT is almost bounded by a constant $C = O(K)$ in practice. So the runtime of forward retiming is $O(K^3n^2)$ in practice.

Theorem 4: For a sequential circuit with n gates, the optimal FRT mapping solution with the minimum clock period can be computed in $O(K^3n^5 \log^2 n)$ time with $O(K^2n^2 + S)$ space, where $S = O(2^Kn)$ is the space needed to represent the original circuit.

The proof of this theorem is presented in the Appendix.

Notice that Theorem 4 is based on the worst case result that the expanded circuits have $O(K^2n^2)$ edges and we need to go through n^2 iterations of label update. In practice, however, the expanded circuits have no more than Kn edges with the efficient K-cut computation on partial flow networks [2] and the number of iterations for each target clock period is always much less than n for all the examples we have tested with the DFS ordering of nodes. Practically, our algorithm runs in $O(K^2n^3 \log^2 n)$ time with $O(Kn + S)$ space requirement, where $S = O(2^Kn)$ is the space to represent the original circuit.

E. Buffer Insertion for Incompatible Initial States of Fanout FF's

In Theorem 2 we assumed that the FF's on different fanout edges of a node had *compatible* initial states. Let us number the FF's on an edge $e(u, v)$ from u to v. The FF's on fanout edges of a node u have *compatible initial states* if for any i, the initial state of the ith FF on edge $e(u, v_1)$ is the same as that of the ith FF on edge $e(u, v_2)$, where v_1 and v_2 are two different fanouts of u. If, however, there are FF's with incompatible initial states as shown in Fig. 6(a), the outputs of the two FF's may constitute two inputs to LUT_v as shown in Fig. 6(b). On the other hand, if we push the two FF's to the LUT's fanout as shown in Fig. 6(c), the two inputs to the LUT can be merged into one. As a result, a K-LUT may not correspond to a K-cut. Consequently, our label computation may not be optimal. To solve this problem, we simply insert a buffer on one fanout edge of u as shown in Fig. 6(d), before the label computation. Obviously, it will change neither any path's delay after mapping, nor the minimum clock period of FRT mapping solutions. In the worse-case, we only need to add $O(Kn)$ buffers, because each edge needs at most one. After mapping, all those buffers can be collapsed into LUT's without increasing either delay or area.

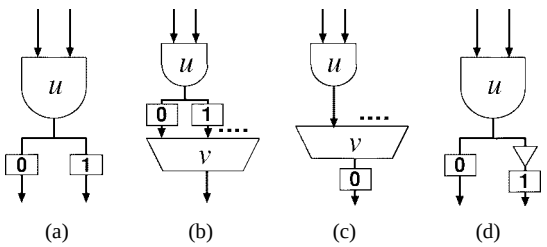


Fig. 6. Incompatible initial states of fanout FF's. Each small rectangle represents a FF with a digit in it representing its initial state.

Theorem 5: For a given sequential circuit, the minimum clock periods of FRT mapping before and after buffer insertion are the same.

Proof: Let the minimum clock period of the optimal mapping solution of the original circuit (denoted G) be Φ_1, and the minimum clock period for the circuit after buffer insertion (denoted G') be Φ_2. Obviously, $\Phi_2 \geq \Phi_1$ because any mapping solution of G' is also a mapping solution of G.

Now we prove that $\Phi_2 \leq \Phi_1$. Let M1 be any mapping solution of G with clock period of ϕ. We shall show that we can construct a mapping solution M2 with the same clock period ϕ for G'. It means that the minimum clock period of an optimal solution of G' is no more than the minimum clock period of an optimal solution of G, i.e., $\Phi_2 \leq \Phi_1$. Let $e(u, v)$ be one edge in G. It is a *visible edge* in M1 if u is an LUT root in M1, or an *invisible edge*, otherwise. To construct M2 for G' based on M1, we add a buffer on $e(u, v)$ if we added a buffer on the corresponding edge in G'. If $e(u, v)$ is an invisible edge in M1, the adding of the buffer will not change the functionality of the LUT in which $e(u, v)$ is included. If $e(u, v)$ is a visible edge in M1, we can pack the buffer in the LUT rooted at u with possible node duplication as shown in Fig. 7(d). Clearly, what we get is an FRT mapping solution of G' and the clock period is ϕ, the same as that of M1. This concludes our proof. ■

Buffer insertion shall not increase the area of any mapping solution. First, all the buffers can be deleted without changing the circuit behavior. Second, the node duplication caused by the buffer insertion shown in Fig. 7(d) can be eliminated easily with a postprocessing of merging LUT's rooted at a node u or u with a buffer as shown in Fig. 7(e).

[7] To be precise, a node can be simulated in $O(l)$ time, where l represents the size of the node, for example, the number of literals in the cube-cover representation or the number of nodes in the binary decision diagram representation.

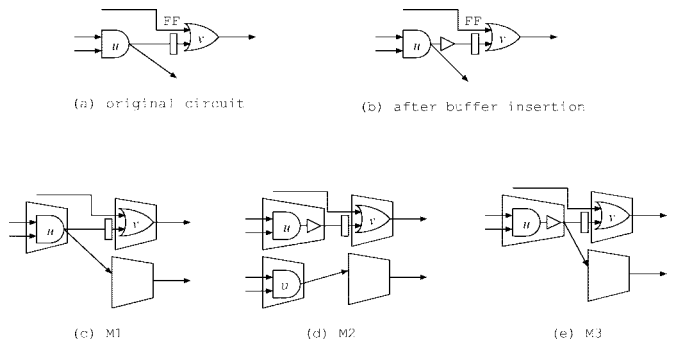


Fig. 7. Buffer insertion will not change the clock period. M1 is a mapping solution of (a). M2 is a mapping solution of (b). M3 is the result of M2 after duplication removal.

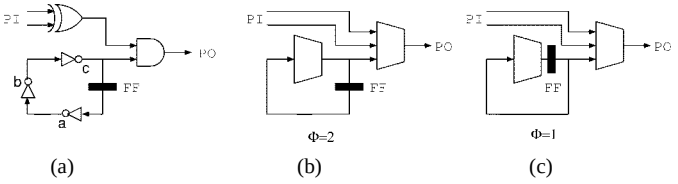


Fig. 8. Mapping for circuits with loops isolated from PI's. Forward retiming is performed for the LUT in the loop of the solution shown in (c). (a) Original circuit, (b) mapping without retiming, and (c) mapping witrh forward retiming.

F. Label Computation for Circuits with Nodes Not Reachable from PI's

In Theorem 1, we assumed that every node is connected to at least one PI. However, there exist circuits with some nodes not connected with any PI's. The l-values of those nodes are not well-defined, and as a result, Theorem 1 no longer holds. Notice that [1, Theorem 3] also suffers the same problem and the algorithms proposed in this paper and [1], [2] cannot be directly applied for this kind of circuits. Nevertheless, the mapping and retiming on those nodes can still affect the clock period of the entire circuit as shown in Fig. 8(b) and (c). We need to consider those nodes in the mapping algorithm.

First, we assume that those nodes which are not connected with PI's must form at least a loop. (Otherwise, they must have at least one predecessor without any input and with constant value. We can either collapse those nodes with constant value into their fanouts or mark them as PI's.) By definition of legal retiming in [3], we know that paths from those loops to PO's will never be critical, because we can insert as many number of FF's on those paths as we want by moving FF's in those loops around. Let us look at the examples shown in Fig. 8(a). Whenever we move the FF around the loop of three inverters once, we will insert one FF on the edge from inverter c to the AND gate. The only possibility that those loops will increase the clock period is when one of them has positive path length. To detect this case with Theorem 1, we can create a pseudo PI and connect it to every node that has no connection with any real PI's with a new edge. We then put a very large number of FF's on those edges. As a result, the l-values of those nodes can still be defined and must be very values, unless they form at least one loop with positive path length.

We can modify our algorithm as follows. We first detect nodes that are not connected with any PI's and then assign their initial lower bounds to be a very small value, but not $-\infty$. (For example, we can assign the initial lower bound of node labels to be $-\phi \cdot F$ for internal nodes and PO's and zero for PI's, respectively, where F is the total number of FF's on the circuit.) The initial lower bound assignment of the rest of node is the same as presented before, i.e., zero for PI's and $-\infty$ for internal nodes and PO's. We then update the iterative label until all the lower bounds converge to values no more than the target clock period ϕ or any of the lower bounds exceed ϕ. In the first case, we conclude that ϕ is a feasible clock period. In the latter case, we conclude that ϕ is infeasible. Clearly, this modification works for mapping with general retiming algorithms proposed in [1] and [2] as well, except that in that case we only need to check the labels of the PO's.

IV. Post-Processing for FPGA's with Fixed Initial State Setting

In Section III, we assume that the target FPGA device provides both set and reset signals, thus, we can set the initial states of FF's arbitrarily to be either one or zero. However, there are some FPGA devices providing only one such control signal. To map a circuit onto such kinds of FPGA devices, the initial states of all the FF's need to be the same (either one or zero, depending on which signal, set or reset, is provided on the devices).[8] In this case, we propose to perform a postprocessing of state switching with inverter insertion on the mapping solution generated by the TurboMap-frt algorithm presented in the previous section. We guarantee that the clock period will remain the same. Furthermore, in most cases, the number of LUT's will also remain the same.

Without lose of generality, we suppose that the target FPGA device provides only reset signal for each FF, i.e., the initial state of every FF needs to be zero. Let F_1 be an FF with initial state of one in an FRT mapping solution. We insert a pair of inverters on the fanin and fanout edges of F_1 and change the initial state of F_1 to be zero as shown in Fig. 9(a)–(c). The inverters can then be packed in LUT's (or I/O pads if they have built-in inverters) connected to the FF as shown in Fig. 9(d)–(f). In case that the inverters connected only to FF's or input-output (I/O) pads which do not have built-in inverters, those inverters need to be implemented with new LUT's. Clearly, we can change the initial state of every FF with this operation and the clock period will remain the same, except a few more LUT's might be needed to implement some inverters if they cannot be packed into existing LUT's.

Theorem 6: The clock period will remain the same after inverter insertion and initial state switching.

Proof: Let p be a combinational path ended with either PI/PO's or FF's and the clock period be $\Phi(\geq 1)$ before inverter insertion. We consider the following two cases.

1) If the number of LUT's on p is larger than zero, all the added inverters on p can be either cancelled out or packed into LUT's on p. As a result, neither the number

[8] For example, Xilinx XC5200 FPGA's provide only reset signal to each FF.

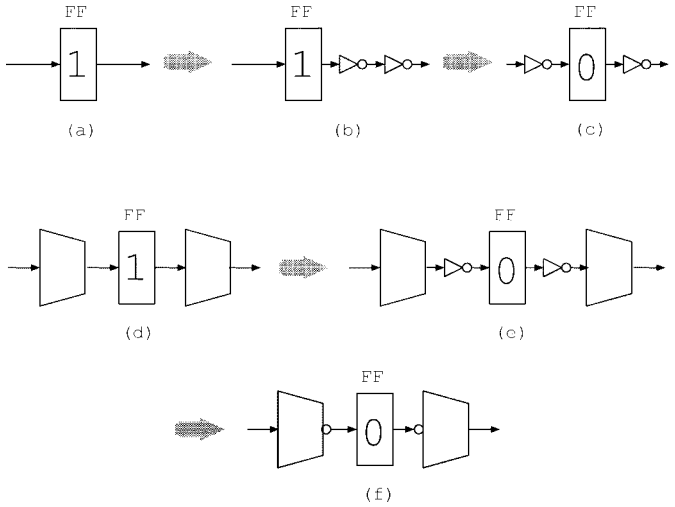


Fig. 9. Changing the initial state of a FF with inverter insertion.

TABLE I
INITIAL CIRCUITS OF MCNC AND ISCAS BENCHMARKS

circuit	PI	PO	GATE	FF
bbara	4	2	28	10
bbtas	2	2	15	5
dk16	2	3	162	5
dk17	2	3	42	5
kirkman	12	6	106	5
ex1	8	19	140	5
ex2	2	2	16	7
s1	8	6	107	5
sse	7	7	74	4
keyb	7	2	134	5
styr	9	10	281	5
sand	11	9	327	17
planet1	7	19	348	6
scf	27	54	516	7
s9234.1	36	39	1293	135
s5378	35	49	1503	164
s15850.1	76	148	3789	515
s38417	28	106	9763	1464

of LUT's on p, nor the clock period of the circuit will change.

2) If the number of LUT's on p is zero, after inverter insertion and cancellation (of inverter pairs), there is at most one LUT on p. Since the clock period in the original mapping solution must be at least one, the delay increase on p from zero to one will not change the clock period as well.

■

V. EXPERIMENTAL RESULTS

The TurboMap-frt algorithm has been implemented in C language on Sun workstations and incorporated into the SIS package [15] and the UCLA RASP FPGA synthesis system [24]. Our first test set consists of 14 MCNC FSM benchmarks and four ISCAS'89 benchmarks. Our second test set consists of eight larger ISCAS benchmarks with more FF's. SIS sequential synthesis commands and the DMIG decomposition method [21] are used to generate the initial circuits as shown in Table I. Columns GATE and FF list the numbers of gates and FF's in each circuit, respectively.

Table II shows the comparison of TurboMap-frt with FlowMap-frt and TurboMap [2] on the first test set. Our experiments were performed on a Sun ULTRA2 workstation with 256-MB memory. K was set to be five. All the mapping results of TurboMap-frt were computed and verified by *verify_fsm* of SIS [15] which performs formal verification for sequential circuits, except the four largest ones which were verified by circuit simulation with input sequences of 3008 random vectors. FlowMap-frt represents the conventional approach. It first partitions a sequential circuit into a set of combinational subcircuits by cutting at inputs and outputs of all FF's, then, maps every subcircuits independently using the depth-optimal FlowMap algorithm [19]. After merging the mapped LUT subcircuits with the original FF's, a post-processing of forward retiming for clock period minimization was performed. TurboMap, on the other hand, computes optimal mapping with *general* retiming solutions, but without consideration of initial states [2]. We used the initial state computation utility in SIS [15] which is based on the algorithm in [8] to try to compute an equivalent initial state for every TurboMap solution. In Table II, Columns LUT and FF list the numbers of LUT's and FF's, respectively, in each mapping solution. Columns Φ list the clock periods of the mapping results. Columns CPU list the CPU time in seconds for each algorithm. Those marked with $\star$ are examples for which SIS failed to compute equivalent initial states for the TurboMap solutions in 2 hours. Column Best lists the best valid solutions (with computed equivalent initial states) by TurboMap and FlowMap-frt.

To show the effectiveness of our algorithm on larger examples with more FF's, we selected eight ISCAS examples with more than 70 FF's. We first performed pipeline insertion to further increase the number of FF's. The pipeline insertion was performed to the extent that the clock period of the original circuit (before mapping) could not be further reduced, i.e., it equals to the maximum loops' delay-to-register ratio. The sizes of the eight examples are shown in Table III. The test results are shown in Table IV. Because all the circuits have more than 100 FF's, we did not try to compute new initial states for TurboMap solutions with SIS, because it would take too long CPU time. We noticed that with more FF's in the circuit, TurboMap-frt needs more time to compute optimal solutions. This is because with more FF's, frt(v) is generally larger and the binary search of $w_{\min}$ from zero to frt(v) (shown in Fig. 5) needs longer computation time.

The results show that TurboMap-frt can reduce the clock period by 17% as compared with FlowMap-frt for both test sets. Comparing with the results by TurboMap [2], which represent the minimum possible clock period of mapping with general retiming, the clock period by TurboMap-frt is 3.6% or 2.8% longer, respectively, for the two test sets. However, there are ten out of 18 TurboMap solutions in the first test set for which SIS concludes no equivalent initial states or cannot find one due to either large memory requirement (more than 300 MB) or long runtime (longer than 2 hours). If we compare the best valid solutions by TurboMap and FlowMap-frt, TurboMap-frt can still reduce the clock period by 8%.

TABLE II
COMPARISON OF TURBOMAP-FRT WITH FLOWMAP-FRT AND TURBOMAP FOR 5-LUT. "GMEAN" LISTS THE GEOMETRIC MEANS OF THE RESULTS BY EACH APPROACH, RESPECTIVELY. THE RUNTIME WAS RECORDED ON A SUN ULTRA2 WITH 256-MB MEMORY. THOSE MARKED WITH ⋆ ARE CIRCUITS THAT SIS FAILED TO COMPUTE INITIAL STATES FOR TURBOMAP SOLUTIONS

	FlowMap-frt				TurboMap				Best	TurboMap-frt			
circuit	Φ	LUT	FF	CPU	Φ	LUT	FF	CPU	Φ	Φ	LUT	FF	CPU
bbara	4	13	10	0.2	3	12	7	0.4	3	3	12	12	0.2
bbtas	2	7	5	0.1	1	6	4	0.2	1	1	6	4	0.1
dk16	14	101	5	0.9	14	103	14	3.8	14	14	103	9	1.7
dk17	2	10	5	0.2	1	6	3	0.4	1	1	6	3	0.2
ex1	8	83	5	0.7	8	92	21	1.9	8	8	92	20	1.3
ex2	2	9	7	0.2	⋆1	4	3	0.2	2	1	4	3	0.1
keyb	10	75	5	0.6	10	79	5	1.6	10	10	81	5	1.0
kirkman	6	48	5	0.7	⋆5	57	24	1.2	6	5	57	14	0.8
planet1	19	213	6	2.0	⋆19	201	18	12.5	19	19	199	37	5.0
s1	7	58	5	0.5	7	63	11	1.2	7	7	56	6	0.7
sand	16	176	17	1.8	⋆15	178	30	10.6	16	15	176	12	4.3
scf	14	325	7	2.8	⋆13	304	20	19.8	14	13	301	27	8.8
sse	7	42	4	0.4	6	45	10	0.9	6	6	44	8	0.5
styr	17	163	5	1.6	⋆16	168	8	5.2	17	17	168	12	3.2
s5378	4	421	204	7.9	⋆4	444	301	51.5	4	4	427	261	40.3
s9234.1	6	462	161	8.5	⋆4	498	217	>7200	6	5	441	203	58.8
s15850.1	10	1240	504	30.3	⋆8	1161	732	>7200	10	10	1166	621	205.6
s38417	8	3526	1464	561.5	⋆6	3420	2264	1201.8	8	6	3301	2573	1210.6
GMEAN	7.0	100	15	1.4	5.6	94	24	7.4	6.3	5.8	92	23	2.8
%	+20.2	+8.4	-33.8	-47.6	-2.8	+1.9	+6.0	+167.0	+8.6	1	1	1	1

TABLE III
EIGHT PIPELINED TEST CIRCUITS FROM ISCAS BENCHMARK SUITE

circuit	PI	PO	GATE	FF
bigkey	262	197	3093	464
s1423	17	5	634	105
s5378	35	49	1284	403
s9234.1	36	39	931	135
s13207.1	62	152	3162	643
s15850.1	77	150	2982	614
s38417	28	106	9763	1464
s38584.1	38	304	9681	1356

Notice that we did not perform the preprocessing of backward retiming for TurboMap-frt in our experiments. We believe that the clock period by TurboMap-frt would have been even closer to that by TurboMap if such preprocessing had been performed. However, we do not see a compelling reason in doing this as the best possible improvement could be very marginal (only 2.8%–3.6% as shown in our experiments). Our results also show that all the three algorithms compute results with similar numbers of LUT's. But FlowMap-frt uses 30%–40% fewer FF's. In general, we think simultaneous mapping with retiming leads to smaller clock period but tends to use more FF's.

VI. CONCLUSION AND FUTURE WORK

For sequential circuits with initial states, we present a new algorithm TurboMap-frt for FPGA mapping with forward retiming and initial state computation to minimize the clock period. Unlike previous retiming algorithms which compute retiming to minimize the clock period at first and then try to compute an equivalent initial state, we compute optimal mapping with forward retiming with guaranteed equivalent initial state in one step. Our algorithm enables a new methodology of separating forward retiming from backward retiming. Since we guarantee to compute an optimal mapping with forward retiming solution, backward retiming can be performed as a preprocessing step to try to push FF's to primary inputs as much as possible with consideration of only initial state computation. Thus, we can avoid the time-consuming iterations between retiming for clock period minimization and initial state computation.

The experimental results show that TurboMap-frt is very efficient and effective comparing with conventional approaches of separated mapping with retiming. Furthermore, the results by TurboMap-frt are very close to that by the optimal mapping with general retiming algorithm TurboMap [2] with regard to both area and clock period, while many solutions by TurboMap cannot compute initial states.

In the future we plan to extend our work for library-based technology mapping with (forward) retiming for high-performance gate array and standard cell designs. A general framework on retiming with multiple clock designs was proposed recently by Legl *et al.* [25]. (The retiming proposed in [3] and used in this and many other papers consider sequential circuits with single clock.) We plan to accommodate our approach into their framework as well.

APPENDIX

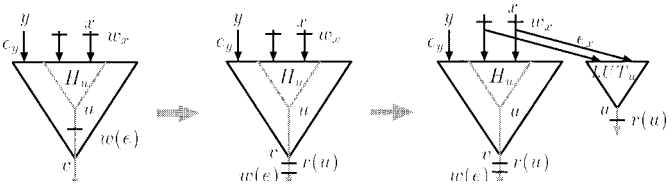
To prove Theorems 3 and 4, we first show the important monotone property of the node s-labels. Given a retiming graph G of a sequential circuit which has a feasible FRT mapping solution for a target clock period Φ, a set of $S(v)$ is *monotone* if $S(u) - \Phi \cdot w(e) \leq S(v)$ for any edge $e(u, v) \in G$.

Theorem 7—Monotone Property: Given a retiming graph G of a sequential circuit which has a feasible FRT mapping solution for a target clock period Φ, the set $\{S(v) | v \in G\}$ is monotone.

Proof: Given a node v of an edge $e(u, v)$, there must exists a FRT mapping solution M such that $s_M(v) = S(v)$. The l-values $l_M(v) = s_M(v) + \Phi \cdot r_M(v)$ and $l_M(u) =$

TABLE IV
COMPARISON OF TURBOMAP-FRT WITH FLOWMAP-FRT AND TURBOMAP FOR 5-LUT. "GMEAN" LISTS THE GEOMETRIC MEANS OF THE RESULTS BY EACH APPROACH, RESPECTIVELY. THE RUNTIME WAS RECORDED ON A SUN ULTRA2 WITH 512-MB MEMORY. NO INITIAL STATE COMPUTATION WAS PERFORMED FOR TURBOMAP SOLUTIONS

	FlowMap-frt				TurboMap				TurboMap-frt			
circuit	Φ	LUT	FF	CPU	Φ	LUT	FF	CPU	Φ	LUT	FF	CPU
bigkey	2	923	466	5.0	2	1136	985	26.0	2	1138	577	171.4
s1423	10	228	105	1.0	9	212	136	9.9	9	208	119	27.7
s5378	4	519	394	1.1	3	481	504	21.5	3	559	603	15.7
s9234	6	313	135	1.1	4	349	212	38.9	5	349	194	19.1
s13207	7	1024	643	3.3	5	1200	869	675.6	5	1193	1018	644.9
s15850	7	1202	614	4.0	6	1136	906	190.8	6	1129	751	309.2
s38417	8	3526	1464	22.3	6	3379	2405	866.1	6	3785	2979	3753.5
s38584	6	3879	1356	10.9	5	3514	2398	503.8	5	3439	1952	2333.8
geo-mean	5.7	933.3	461.0	3.4	4.6	948.4	710.6	102.4	4.7	973.9	659.3	199.3
%	+21.2	-4.2	-30.1	-98.3	-2.8	-2.6	+7.8	-48.6%	1	1	1	1



(a) (b) (c)

Fig. 10. Proof of monotone property of node labels for Case 2.

$s_M(u) + \Phi \cdot r_M(u)$. Let LUT_v be the K-LUT rooted at v in M. The edge weight $w^r(e)$ after forward retiming is $w^r(e) = w(e) + r_M(u) - r_M(v)$. (Notice that $r_M(v)$ is the number of FF's moved forward across v in M.) We consider the following two cases:

Case 1: u is a fanin to LUT_v. Since $l_M(u) - \Phi \cdot w^r(e) + 1 \leq l_M(v)$, we have that $s_M(u) + \Phi \cdot r_M(u) - \Phi(w(e) + r_M(u) - r_M(v)) + 1 \leq l_M(v) = s_M(v) + \Phi \cdot r_M(v)$. It means that $S(u) - \Phi \cdot w(e) \leq s_M(u) - \Phi \cdot w(e) < s_M(v) = S(v)$.

Case 2: u is covered inside LUT_v. Let H_u be the largest sub-DAG rooted at u inside LUT_v as shown in Fig. 10(a). The number of FF's moved across u in LUT_v is $r(u) = r_M(v) - w(e)$. Clearly, H_u can be covered by a K-LUT. By replicating H_u explicitly outside LUT_v to form LUT_u and push $r(u)$ FF's forward to u's output,[9] we can form another FRT mapping solution M' with the same clock period of M. So

$$\begin{aligned} s_{M'}(u) &= l_{M'}(u) - \Phi \cdot r(u) \\ &\leq l_{M'}(v) - \Phi \cdot r(u) \\ &= s_M(v) + \Phi \cdot r_M(v) - \Phi \cdot r(u) \\ &= s_M(v) + \Phi \cdot (w(e) + r(u)) - \Phi \cdot r(u) \\ &= s_M(v) + \Phi \cdot w(e). \end{aligned}$$

As a result, $S(u) - \Phi \cdot w(e) \leq s_{M'}(u) - \Phi \cdot w(e) \leq s_M(v) = S(v)$. ■

Based on the monotone property we can prove the following.

Lemma 3: Given a sequential circuit which has a feasible FRT mapping solution for a target clock period Φ, the inequality $s(v) \leq s_{\text{new}}(v) \leq S(v)$ holds for every node v all the time during labeling iterations.

[9]Notice that only forward retiming is allowed.

Proof: It is obvious that $s(v) \leq s_{\text{new}}(v)$ based on our FRTcheck algorithm shown in Fig. 4. We now prove by mathematical induction that $s_{\text{new}}(v) \leq S(v)$ holds after every label update.

Initially, $s(u) = -\infty < S(u)$. We prove that if $s(u) \leq S(u)$ for every node u, $s_{\text{new}}(v) \leq S(v)$ after each label update for a node v. In the following, we assume $\varsigma(v) \geq s(v)$, because it is obvious that $s_{\text{new}}(v) = s(v) \leq S(v)$ if $\varsigma(v) < s(v)$.

Let M be a feasible FRT mapping solution such that $s_M(v) = S(v)$ and $\overline{X}$ is the LUT rooted at v in M. In M the cut-height $h_M(X, \overline{X}) = s_M(v) = S(v)$ and the cut-weight $w_M(X, \overline{X}) = R(v) \leq frt(v)$. Clearly, $(X, \overline{X})$ is a K-cut on $\mathcal{F}_v^{\text{frt}(v)}$. Based on Corollary 1, we have that $h_M(X, \overline{X}) + \Phi \cdot w_M(X, \overline{X}) \leq \Phi$. Since $s(u) \leq S(u) \leq s_M(u)$ hold for every node u, the height of the cut based on current $(s(u), r(u))$ satisfies the inequality of $h(X, \overline{X}) \leq h_M(X, \overline{X}) = S(v)$. Furthermore, $w_M(X, \overline{X}) = w(X, \overline{X})$ is an invariant during labeling iterations for a particular $\overline{X}$. As a result, $h(X, \overline{X}) + \Phi \cdot w(X, \overline{X}) \leq h_M(X, \overline{X}) + \Phi \cdot w_M(X, \overline{X}) \leq \Phi$. Based on the monotone property

$$\begin{aligned} \varsigma(v) &= \max\{s(u) - \Phi \cdot w(e) |\ \forall e(u, v) \in G\} \\ &\leq \max\{S(u) - \Phi \cdot w(e) |\ \forall e(u, v) \in G\} \\ &\leq S(v) = h_M(X, \overline{X}). \end{aligned}$$

Suppose first, that $h(X, \overline{X}) > \varsigma(v)$. Obviously, $s_{\text{new}}(v) \leq \varsigma(v) + 1 \leq h(X, \overline{X}) \leq h_M(X, \overline{X}) = S(v)$.

Second, if $h(X, \overline{X}) \leq \varsigma(v)$, the cut $(X, \overline{X})$ is a K-cut and

$$\begin{aligned} h(X, \overline{X}) + \Phi \cdot w(X, \overline{X}) &\leq \varsigma(v) + \Phi \cdot w(X, \overline{X}) \\ &\leq h_M(X, \overline{X}) + \Phi \cdot w_M(X, \overline{X}) \\ &\leq \Phi. \end{aligned}$$

As a result, $s_{\text{new}}(v) = \varsigma(v) \leq h_M(X, \overline{X}) = S(v)$. ■

Lemma 4: Given a sequential circuit G which has a feasible FRT mapping solution for a target clock period Φ, $r_{\text{new}}(v) \leq R(v)$ when $s_{\text{new}}(v) = S(v)$ for any v in G.

Proof: Based on the proof of the previous lemma, there must exist a K-cut $(X, \overline{X})$ such that $h(X, \overline{X}) \leq S(v)$, $w(X, \overline{X}) = R(v)$, and $h(X, \overline{X}) + \Phi \cdot w(X, \overline{X}) \leq \Phi$ on $\mathcal{F}_v^{\text{frt}(v)}$. If $s_{\text{new}}(v) = h(X, \overline{X}) = S(v)$, it must be that

$r_{\text{new}}(v) \leq w(X, \overline{X}) = R(v)$, because we always compute a K-cut with minimum weight and height of no more than $S(v)$. ■

As a result, we have the following theorem:

Theorem 3: For a sequential circuit which has a feasible FRT mapping solution for a target clock period Φ, starting from the initial lower bounds $(0, 0)$ for PI's and $(-\infty, 0)$ for the other nodes, the inequalities 1) $s(v) \leq S(v)$ and 2) $r(v) \leq R(v)$ when $s(v) = S(v)$ hold all the time after any number of iterations of label update.

The optimality of the FRTcheck algorithm can be proved as follows.

Lemma 5: If there is a feasible FRT mapping solution for a target clock period Φ, the two-tuples $(s(v), r(v))$ computed by FRTcheck will converge to the node label pairs $(S(v), R(v))$ for all nodes after no more than n^2 iterations.

Proof: According to Lemma 3, $s(v)$ will monotonically increase and converge to $S(v)$ finally. According to Lemma 4, $r(v)$ will converge to $R(v)$ as well when $s_{\text{new}}(v) = S(v)$. As a result, we only need to prove that $s(v)$ will converge to $S(v)$ within n^2 iterations for all the nodes.

According to [2], the lower bound $l(v)$ of every node v computed by TurboMap, which equals to the height of a min-height K-cut on the expanded circuit of v, will converge to the node label $L(v)$ within n^2 iterations. In our label computation, we have the same initial lower bounds and a tighter upper bounds on the node labels because we require $S(v) \leq \Phi$ for *every* node v, while TurboMap requires only $L(v) \leq \Phi$ for those PO v. Clearly, we only need to prove that $s_{\text{new}}(v) \geq l_{\text{new}}(v)$ for every node v, where $s_{\text{new}}(v)$ and $l_{\text{new}}(v)$ are the new lower bounds computed by TurboMap-frt and TurboMap [2], respectively, after each label update.

Since $s_{\text{new}}(v)$ in TurboMap-frt equals to the cut-height of a min-height-min-weight K-cut, while $l_{\text{new}}(v)$ in TurboMap equals to the cut-height of a min-height K-cut, it is obvious $s_{\text{new}}(v) \geq l_{\text{new}}(v)$. ■

Lemma 6: For a target clock period Φ, if FRTcheck returns TRUE, there must exist a feasible FRT mapping solution.

Proof: To prove there exists a feasible FRT mapping solution, we only need to construct one and prove it has a clock period of no more than the target Φ. This has been shown in Lemma 2. ■

As in [1] and [2], we assume that each edge in the original circuit has at most one FF. Based on Lemmas 5 and 6, we have that:

Theorem 4: For a sequential circuit with n gates, the optimal FRT mapping solution with the minimum clock period can be computed in the worst case time complexity of $O(K^3n^5 \log^2 n)$ and worst case space requirement of $O(K^2n^2 + S)$, where $S = O(2^K n)$ is the space needed to represent the original circuit.

Proof: There are $O(Kn)$ edges and $O(Kn)$ FF's in a K bounded circuit. To update one node's lower bound, we need $O(\log(Kn)) = O(\log n)$ K-cut computations. Since the expanded circuits have $O(Kn^2)$ nodes and $O(K^2n^2)$ edges [1], each K-cut computation can be finished in $O(K^3n^2)$ time [2]. Thus, computing $(s_{\text{new}}(v), r_{\text{new}}(v))$ for one node needs $O(K^3n^2 \log n)$ time. There are at most n^2 labeling iterations for one target clock period and we need to binary search n possible clock periods. Each labeling iteration will update the lower bounds of n nodes once. Thus, the total label computation time is $O(K^3n^5 \log^2 n)$. All the frt(v) can be computed in $O(n^2)$ time. The mapping generation and forward retiming need $O(K^3n^3)$ time with $O(K^2n^2)$ space in the worst case. As a result, the total time complexity of the TurboMap-frt algorithm is $O(K^3n^5 \log^2 n)$ with space requirement of $O(K^2n^2+S)$. It is obvious that $S = O(2^K n)$, because the functionality of each K-input node can be represented in $O(2^K)$ space and the number of edges is $O(Kn)$.■

REFERENCES

[1] P. Pan and C. L. Liu, "Optimal clock period FPGA technology mapping for sequential circuits," *ACM Trans. Design Automat. Electron. Syst.*, vol. 3, no. 3, 1998

[2] J. Cong and C. Wu, "An efficient algorithm for performance-optimal FPGA technology mapping with retiming," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 738–748, Sept. 1998.

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

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

[5] H. Touati, N. Shenoy, and A. Sangiovanni-Vincentelli, "Retiming for table-lookup field programmable gate arrays," in *Proc. FPGA'92*, 1992, pp. 89–94.

[6] S. Malik, K. J. Singh, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance optimization of pipelined circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1990, pp. 410–413.

[7] J. Cong and C. Wu, "An improved algorithm for technology mapping with retiming for lookup-table-based FPGA's," Univ. California, Los Angeles, Tech. Rep. UCLA-CSD 960012, Apr. 1996.

[8] H. Touati and R. K. Brayton, "Computing the initial states of retimed circuits," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 157–162, Jan. 1993.

[9] V. Singhal, S. Malik, and R. K. Brayton, "The case for retiming with explicit reset circuitry," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1996, pp. 618–625.

[10] H. Yotsuyanagi, S. Kajihara, and K. Kinoshita, "Retiming for sequential circuits with a specified initial state and its application to testability enhancement," *IEICE Trans. Inform. Syst.*, vol. E78-D, no, 7, pp. 861–867, July 1995.

[11] G. Even, I. Y. Spillinger, and L. Stok, "Retiming revisited and reversed," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 348–357, Mar. 1996.

[12] N. Maheshwari and S. S. Sapatnekar, "Minimum area retiming with equivalent initial states," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1997, vol. 13, pp. 216–219.

[13] H. Fujiwara and S. Toida, "The complexity of fault detection: An approach to design for testability," *FTCS-12*, pp. 101–108, 1982.

[14] S. Kundu, L. M. Huisman, I. Nair, and V. Iyengar, "A small test generator for large designs," in *Proc. Int. Test Conf.*, 1992, pp. 30–40.

[15] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Electron. Res. Lab., Memo. UCB/ERL M92/41, 1992.

[16] L. Stok, I. Spillinger, and G. Even, "Improving initialization through reversed retiming," in *Proc. Euro. Design Test Conf.*, 1995, pp. 150–154.

[17] J. Cong and C. Wu, "Optimal FPGA mapping and retiming with efficient initial state computation," in *Proc. ACM/IEEE Design Automation Conf.*, 1998, pp. 330–335.

[18] J. Cong and Y. Ding, "On nominal delay minimization in LUT-based FPGA technology mapping," *Integration—The VLSI J.*, vol. 18, pp. 73–94, 1994.

[19] ——, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table-based FPGA designs," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1–12, Jan. 1994.

[20] R. K. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 1062–1081, June 1987.

[21] K. C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, "DAG-Map: Graph-based FPGA technology mapping for delay optimization," *IEEE Design Test Comput.*, pp. 7–20, 1992.

[22] J. Cong and Y.-Y. Hwang, "Structural gate decomposition for depth-optimal technology mapping in LUT-based FPGA design," in *Proc. 33rd ACM/IEEE Design Automation Conf.*, 1996, pp. 726–729.
[23] T. H. Cormen, C. H. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* Cambridge, MA: MIT Press, 1990.
[24] J. Cong, J. Peck, and Y. Ding, "RASP: A general logic synthesis system for SRAM-based FPGA's," in *Proc. ACM 4th Int. Symp. FPGA*, 1996, pp. 137–143.
[25] K. Eckl, J. Madre, P. Zapter, and C. Legl, "A practical approach to multiple-class retiming," in *Proc. ACM/IEEE Design Automation Conf.*, 1999, pp. 237–242

Jason Cong, for a photograph and biography, see p. 420 of the April 1999 issue of this TRANSACTIONS.

Chang Wu received the B.S. degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 1986, the M.S. degree from the Institute of Pattern Recognition and Artificial Intelligence at the same university in 1989, and the Ph.D. degree in computer science from University of California at Los Angeles in 1999.

From 1989 to 1995, he worked at Beijing Integrated Circuit Design Center as a Senior Software Engineer developing the Panda VLSI CAD System. He was a Visiting Scholar at the University of California at Los Angeles from 1995 to 1996. He is now a Senior Software Engineer and Researcher at Aplus Design Technologies, Inc., Los Angeles, CA. His major interests are layout driven logic synthesis and technology mapping with retiming for high-performance VLSI designs.