Optimal FPGA Mapping and Retiming with Efficient Initial State Computation

Jason Cong and Chang Wu
Department of Computer Science
University of California, Los Angeles, CA 90095

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 propose a novel *polynomial time algorithm for optimal FPGA mapping with forward retiming* to minimize the clock period with guaranteed initial state computation. It enables a new methodology of separating forward retiming from backward retiming to avoid time-consuming iterations between retiming and initial state computation. Our algorithm compares very favorably with both of the conventional approaches of separate mapping followed by retiming [1, 8] and the recent approaches of combined mapping with retiming [12, 2]. It is also applicable to circuits with partial initial state assignment.

1 Introduction

Retiming is a well known technique to reduce the clock period by repositioning flipflops (FFs) in sequential circuits originally proposed by Leiserson and Saxe [8]. Many studies have been done on combining retiming with logic optimization, circuit partitioning and technology mapping, e.g., [11, 16, 10]. For lookup-table (LUT) based FPGA designs, a significant advancement was made by Pan and Liu [12] on simultaneous optimal technology mapping with retiming for clock period minimization. For designs with initial states, however, equivalent initial states need to be computed for retiming, which unfortunately is NP-hard [14].

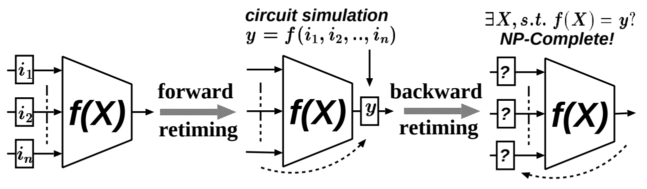

Figure 1: Retiming and initial state computation. Each small rectangle represents a FF.

A general retiming procedure usually involves two kinds of FF movements. *Forward retiming* moves FFs from the inputs of some gates to their outputs, while *backward retiming* moves FFs in the opposite direction. As shown in Figure 1, the equivalent initial state of forward retiming can be computed easily using circuit simulation. The initial state for backward retiming, however, needs to solve a satisfiability problem which is in general NP-complete.

Several heuristics for initial state computation based on ATPG (automatic test pattern generation) techniques [17, 4, 9] or STG (state transition graph) traversal [15] have been proposed in recent years. However, none of them guarantees to find an equivalent initial state for a given retiming. For example, L. Stok, et. al. [4] proposed to try to exclude backward retiming by minimizing the maximum retiming value. If backward retiming is still needed for a target clock period, ATPG based justification was proposed to compute equivalent initial states with possible iterations of retiming and initial state computation. However, ATPG itself is also NP-complete [5, 6]. We also observe that minimizing the maximum retiming value as formulated in [4] does not always lead to a simpler initial state computation problem. Our experience shows that unless a complete forward retiming solution can be found, the effort of initial state computation depends on the number of nodes with backward retiming, instead of the maximum retiming value. Alternatively, Touati and Brayton [15] proposed an initial state computation algorithm based on STG traversal. Their algorithm can compute equivalent initial states under the condition that there exists an input sequence for the original initial state going back to itself in the STG of the original circuit [15]. However, unless provided externally, to determine the existence and compute such an input sequence needs to search the STG which may have an exponential number of nodes. Although representing the reset logic explicitly can avoid the STG traversal [15], the extra cost of the reset logic separated from FFs after retiming can be high. Furthermore, such kind of reset logic may restrict possible retiming of the original circuit, thus reduce the potential of retiming optimization.

In order to avoid the high 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 FFs back to PIs as much as possible to enlarge the solution space of mapping with forward retiming, thus achieve better results. One important advantage of our approach is that during the preprocessing of backward retiming we only need to focus on the initial state computation without considering the effect on the clock period. As a result we can still achieve effective retiming for circuit optimization without time-consuming iterations between retiming and initial state computation, avoiding the slow STG traversal procedure or any extra reset logic in existing approaches [9, 17, 14, 15]. Our algorithm compares very favorably with both of the conventional approaches of separate mapping with retiming [1, 8] and the recent approaches of combined mapping with retiming [12, 2]. It is also applicable to circuits with partial initial state assignment.

The rest of the paper is organized as follows. Section 2 is the problem formulation and definitions and an overview of our algorithm. Our algorithm is presented in Section 3. The experimental results are presented in Section 4 and the conclusions and our future work in Section 5. Due to the page limit, the details of the algorithm and the proofs of the theorems are left out. They are available in [3].

2 Problem Formulation and Definitions

Given a *sequential* circuit, the technology mapping problem for K-LUT based FPGAs is to construct an *equivalent* circuit consisting of K-LUTs and flipflops (FFs), such that both of the circuits generate the same output sequence for any input sequence, starting from their individual initial states, respectively. We propose to study the following problem in this paper.

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

As in [12, 2], instead of solving the optimization Problem 1 directly, we shall solve its decision version and use binary search to get 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 the clock period of no more than Φ under forward retiming.*

In this paper, sequential circuits are represented as retiming graphs. 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 [8]. 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 node u to node v. The *weight* $w(e)$ of an edge e is the number of FFs 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 the unit-delay mode[1], *delay* $d(v) = 1$ for every internal node v, or 0 for every PI or PO. *Path delay* $d(p)$ of a path p is the sum of all node delays on the path.

For a target clock period Φ, we define the edge length $length(e)$ to be $d(v) - \Phi \cdot w(e)$ for an edge $e(u, v)$. The path length $length(p)$ is $\sum_{e \in p} length(e)$. In an LUT network M, a node's l-**value** is defined to be $l_M(v) = max\{length(p)\}$ for all paths p from PIs to v. It can be proved that,

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 $l_M(v) \leq \Phi$, for every LUT $v \in M$.*

Let M be one solution of mapping with forward retiming, thereafter called *FRT mapping solution*. In M, the output of an LUT rooted at v is generally $r(v) (\geq 0)$ cycles ahead

[1]We use the unit-delay mode and assume synchronous sequential circuits with single-phase clock and edge-triggered flipflops in this work.

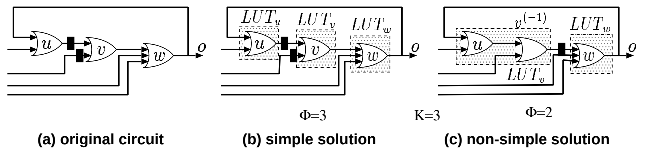

Figure 2: Simple vs. non-simple mapping solutions. Each dotted block represents a 3-LUT.

the output of v in the original circuit due to forward retiming. A mapping solution with $r(v) \leq 0$ for every LUT root v is called a *simple mapping solution* [12]. One important property of simple mapping solution is that the path weight of any path from PIs to a node is the same as that in the original circuit. Thus, we only need to focus on reducing the path delay for path length and clock period minimization. It was shown in [12] that, there always exists a *simple* mapping solution whose clock period will be the minimum after a second step of *general* retiming. However, there may not exist any simple FRT mapping solutions with the minimum clock period after optimal *forward* retiming. For the example shown in Figure 2(a) with $K = 3$, it is not difficult to verify that there does not exist a simple FRT mapping solution with $\Phi = 2$ under forward retiming, while there does exist a non-simple FRT mapping solution with $\Phi = 2$ as shown in Figure 2(c). As a result, the minimum l-values among all possible FRT mapping solutions are much more difficult to compute, because the path weight from PIs to a node will also be changed which makes the monotone property of l-values shown in [2] no longer holds. To overcome these difficulties we separate the l-value into two values such that the first one has the monotone property defined in [2] and the second one can be computed according to the first one.

Definition 1 *Given a mapping solution M and a target clock period Φ, the number of FFs moved forward across LUT v, denoted $r_M(v)$, is called the* **forward retiming value** *of v in M.*[2] *The* s-**value** $l_M^s(v) = l_M(v) - \Phi \cdot r_M(v)$ *represents the l-value in the corresponding simple mapping solution of M after pushing back the $r_M(v)$ FFs to the inputs of v, where $l_M(v)$ is the l-value of v in M.*

According to Theorem 1 we have that,

Corollary 1 *A mapping solution M has a clock period of no more than a given Φ under forward retiming, if and only if $l_M^s(v) + \Phi \cdot r_M(v) \leq \Phi$ for all the LUT roots 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 solution*), the* s-**label** $L^s(v)$ *is defined to be the minimum $l_M^s(v)$ 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 $l_M^s(v) = L^s(v)$. The* **node label pair** *is defined to be a 2-tuple $(L^s(v), R(v))$.*

We shall show that $L^s(v)$ has the important monotone property in § 3.2. According to Corollary 1, we can solve the decision Problem 2 by computing the node label pairs and checking if $L^s(v) + \Phi \cdot R(v) \leq \Phi$ holds for every node.

[2]In [8], the retiming value $\Re(v) \leq 0$ corresponds to forward retiming at node v. We define $r_M(v) \geq 0$ for FRT mapping.

After getting the minimum clock period Φ_{min} and the corresponding label pairs, we construct a mapping solution and perform a separate forward retiming step to achieve Φ_{min}.[3] Minimizing $R(v)$ means that we want to push forward as few FFs as possible for each node v to leave the maximum freedom to the subsequent forward retiming step, because those FFs pushed forward can no longer be pushed back. On the other side, $R(v)$ cannot be *too* small because we also want to construct larger LUTs to minimize $L^s(v)$ by pushing forward some FFs. Our algorithm can simultaneously minimize both $L^s(v)$ and $R(v)$ to compute optimal solutions.

3 TurboMap-frt Algorithm

Our algorithm, named TurboMap-frt, computes 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 1 to $|V|$.[4] 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 update those lower-bounds until all of them converge to the values of node label pairs if there exists a feasible solution, or stop if we conclude there is no feasible solution. Expanded circuits are used to represent all possible LUTs under node duplication and forward retiming for label pair update.

Before presenting the details of our algorithm, we review the definition of K-cuts here. In a directed graph with one sink and one source, a *cut* $(X, \overline{X})$ is a partition of the nodes in 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 $|V(X, \overline{X})| \leq K$, $(X, \overline{X})$ is called a *K-feasible cut*, or *K-cut* in short, and $\overline{X}$ is called a K-feasible cone. A cut is a *min-cut*, if $|V(X, \overline{X})|$ is minimum among all the cuts in the graph.

3.1 LUT Formation

To compute an optimal solution we need to be able to search all possible LUTs for every node under node duplication and forward retiming. Pan and Liu [12] proposed to search all possible LUTs rooted at a node on expanded circuits of the node. An *expanded circuit* $\mathcal{E}_v$ for a node v is a directed acyclic graph (DAG) rooted at v^0 and constructed from the original circuit with node duplication as shown in Figures 3b – 3e. One important property of $\mathcal{E}_v$ is that for any node, denoted u^w, every path from u^w to the root v^0 passes exactly w FFs. Pan and Liu [12] showed that *to examine all K-LUTs for a node v, it sufficed to examine all the K-LUTs that can be derived from the K-cuts in a large enough expanded circuit $\mathcal{E}_v^{Kn}$ of v* due to the one-to-one correspondence between K-cuts and K-LUTs.[5]

However, the one-to-one correspondence between K-cuts on $\mathcal{E}_v^{Kn}$ and K-LUTs rooted at v under forward retiming no

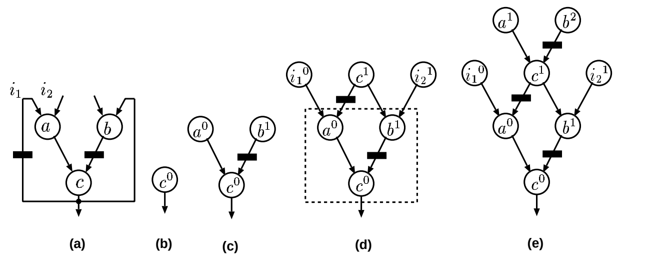

Figure 3: Expanded circuits for LUT representation.

longer holds. As shown in Figure 3(d), to cluster the nodes in the dotted box to form an LUT we have to move the FF backward from the fanout of b^1 to the fanins of b^1, because the path $i_1^0 \rightarrow a^0 \rightarrow c^0$ does not have any FF to push forward. Thus, such a cluster is not a valid LUT formation for FRT mapping. For FRT mapping, we construct a smaller expanded circuit for every node based on its *maximum forward retiming value* so that there is a one-to-one correspondence between K-LUTs to K-cuts on the expanded circuits.

Definition 3 *The* **maximum forward retiming value** *$frt(v)$ of a node v in a retiming graph is the maximum number of FFs which can be moved forward from the (transitive) fanins of v to the output of v.*

Lemma 1 *$frt(v) = min\{w(p) \mid \forall\ p : PI \rightsquigarrow v\}$.*

Since the edge weight $w(e) \geq 0$, the set $\{frt(v)\}$ of all the nodes in a circuit can be computed in $O(|V|^2)$ time by Dijkstra's shortest path algorithm.

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 node v is a sub-DAG of $\mathcal{E}_v^{Kn}$ (defined in [12]) 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 $w \leq i$, and 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 $\mathcal{F}_v^i$ and $w > i$. Notice that if two leaves ${u_1}^{w_1}$ and ${u_2}^{w_2}$ have an edge in $\mathcal{E}_v^{Kn}$, the edge will not appear in $\mathcal{F}_v^i$ because a leaf cannot have any inputs. For example, the shaded area of Figure 4(b) represents $\mathcal{F}_c^1$ for the circuit in Figure 4(a), where the heavy shaded nodes are leaves of $\mathcal{F}_c^1$.

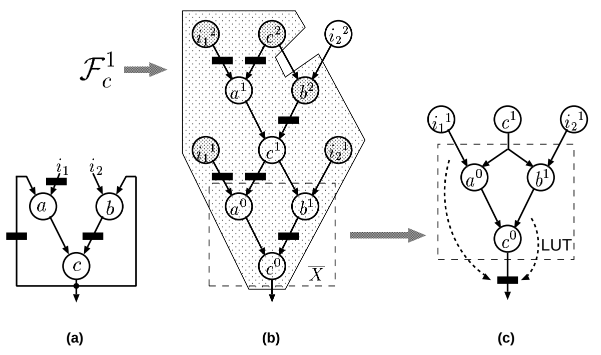

Figure 4: LUT formation for FRT mapping based on the expanded circuits.

When i is set to $frt(v)$, we have that:

[3] Notice that an optimal forward retiming after mapping can be computed easily based on [8] by adding linear constraints $\Re(v) \leq 0$ for every node v. In fact, we can compute a forward retiming directly from the node label pairs as shown in § 3.3.

[4] Much smaller upper-bound can be computed by performing depth-optimal mapping (FlowMap [1]) on combinational subcircuits following by forward retiming.

[5] It was shown that the expanded circuit had no more than $O(Kn^2)$ nodes and $O(K^2n^2)$ edges for a circuit with n gates [12].

Theorem 2 *Every K-feasible cut $(X,\overline{X})$ on $\mathcal{F}_v^{frt(v)}$ corresponds to a K-LUT rooted at v under 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 a FRT mapping solution corresponds to a K-feasible cut on $\mathcal{F}_v^{frt(v)}$.*

To further demonstrate our approach, let us look at the circuit shown in Figure 4(a). Notice that its only difference with the circuit in Figure 3(a) is one extra FF on edge (i_1,a) which makes $frt(c)=1$. Now the 3-cut shown in Figure 4(b) can form a 3-LUT as shown in Figure 4(c).

3.2 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. First, we assign an initial lower-bound $(0,0)$ on the value of label pair for every PI and $(-\infty,0)$ for the other nodes. Then we iteratively update the lower-bounds until they converge to the values of node label pairs. If, however, the lower-bounds cannot converge to finite values after $|V|^2$ iterations, we conclude that there is no feasible FRT mapping solution for the given Φ. The pseudo-code of the FRTcheck algorithm is shown in Figure 5.

```
FRTcheck($G(V,E,W)$, $\Phi$)
1   assign initial lower-bound $(0,0)$ for each PI and
      $(-\infty,0)$ for the other nodes
2   converge ← FALSE, $i \leftarrow 0$
3   while (not converge and $i \le |V|^2$)
4       converge ← TRUE
5       for each node $v \in V$
6           $(l^s_{new}(v), r_{new}(v)) \leftarrow$LabelUpdate$(v,\Phi)$
7           if $(l^s_{new}(v) > l^s(v))$
8               $l^s(v) \leftarrow l^s_{new}(v)$
9               converge ← FALSE
10      $i \leftarrow i+1$
11  return (converge? TRUE : FALSE)
```

Figure 5: Label computation for a target clock period Φ.

The most important part of our algorithm is the update of the lower-bound pair $(l^s(v), r(v))$ for every node v (procedure LabelUpdate, line 6 in Figure 5). For each node v, we update $l^s(v)$ at first and then $r(v)$ through the computation of min-height-min-weight K-cuts on $\mathcal{F}_v^{frt(v)}$.

Definition 4 *For a K-cut $(X,\overline{X})$ in an expanded circuit of node v, the* **cut-weight** *is $w(X,\overline{X}) = max\{w \mid \forall u^w \in \overline{X}\}$.*

Definition 5 *Given a set of lower-bounds $(l^s(v), r(v))$, where $l^s(v) \le L^s(v)$ and, $r(v) \le R(v)$ when $l^s(v) = L^s(v)$, the* **cut-height** *of a K-cut $(X,\overline{X})$ in the expanded circuit $\mathcal{F}_v^{frt(v)}$ of node v is defined to be*

$$h(X,\overline{X}) = \max\{l^s(u) - \Phi\cdot w + 1 \mid \forall u^w \in V(X,\overline{X})\}.$$

To compute a tighter lower-bound $l^s_{new}(v)$ for node v, we compute $\mathcal{L}^s(v) = max\{l^s(u) - \Phi\cdot w(e) \mid \forall e(u,v) \in G\}$ on the original circuit G. Then we decide whether there exists a K-cut on $\mathcal{F}_v^{frt(v)}$ with height of no more than $\mathcal{L}^s(v)$ based on the max-flow K-cut computation on a partial flow network presented in [2].[6] If there does not exist such a K-cut, we set $l^s_{new}(v) = \mathcal{L}^s(v)+1$ and $r_{new}(v) = 0$, because it can then be proved that $L^s(v) > \mathcal{L}^s(v)$.

[6]We refer readers to [2] for the detail.

If, however, there does exist such a K-cut with height of no more than $\mathcal{L}^s(v)$, we then compute a K-cut with the minimum cut-weight among those with height of no more than $\mathcal{L}^s(v)$ by binary search on the cut-weight from 0 to $frt(v)$ as follows. (Recall that $R(v)$ is the minimum $r_M(v)$ among all the feasible FRT mapping solutions M with $l^s_M(v) = L^s(v)$. To minimize $w(X,\overline{X})$ is to guarantee that $r_{new}(v) = w(X,\overline{X}) \le R(v)$ when $l^s_{new}(v) = L^s(v)$.)

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

Let $(X,\overline{X})$ be such a K-cut with height of $\mathcal{L}^s(v)$ and weight of w_{min} computed above. If $\mathcal{L}^s(v) + \Phi\cdot w_{min} \le \Phi$, we update the new lower-bound to be $(\mathcal{L}^s(v), w_{min})$. Otherwise, we update the new lower-bound to be $(\mathcal{L}^s(v)+1, 0)$.

It can be proved that $l^s_{new}(v) \le L^s(v)$ and,
$r_{new}(v) \le R(v)$ when $l^s_{new}(v) = L^s(v)$. In other words, the updated pair is still a lower-bound on the value of the label pair of v. This result is based on the monotone property of $L^s(v)$. Given a sequential circuit G which has a feasible FRT mapping solution for a target clock period Φ, a set of $L^s(v)$ is *monotone* if, $L^s(u) - \Phi\cdot w(e) \le L^s(v)$ for any edge $e(u,v) \in G$.

Theorem 3 (Monotone Property) *Given a sequential circuit G which has a feasible FRT mapping solution for target clock period Φ, the set $\{L^s(v) \mid v \in G\}$ is monotone.*

Let one *iteration* denote the computation process where the lower-bound $(l^s(v), r(v))$ is updated once for every node v (lines 5 $\sim$ 9 in Figure 5). Based on the monotone property we can prove that:

Lemma 2 *Given a sequential circuit which has a feasible FRT mapping solution for a target clock period Φ, the inequality $l^s(v) \le l^s_{new}(v) \le L^s(v)$ holds all the time after any number of iterations.*

Lemma 3 *Given a sequential circuit G which has a feasible FRT mapping solution for a target clock period Φ, $r_{new}(v) \le R(v)$ when $l^s_{new}(v) = L^s(v)$.*

As a result, we have the following theorem:

Theorem 4 *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 PIs and $(-\infty,0)$ for the other nodes, the inequalities 1. $l^s(v) \le L^s(v)$ and, 2. $r(v) \le R(v)$ when $l^s(v) = L^s(v)$ hold all the time after any number of iterations of label update.*

Based on the above label pair update procedure, we can prove the optimality of FRTcheck algorithm as follows.

Lemma 4 *If there is a feasible FRT mapping solution for a target clock period Φ, the 2-tuples $(l^s(v), r(v))$ computed by FRTcheck will converge to the node label pairs $(L^s(v), R(v))$ for all nodes after no more than $|V|^2$ iterations.*

Lemma 5 *For a target clock period Φ, if FRTcheck returns TRUE, there must exist a feasible FRT mapping solution.*

Theorem 5 *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 $O(K^3 n^5 \log^2 n)$ and worst-case space requirement of $O(K^2 n^2)$.*

Notice that the above result is based on the worst-case scenario that the expanded circuits have $O(K^2 n^2)$ edges [12] and we need go through n^2 iterations (lines 5~9 in Figure 5). In practice, the expanded circuits have much fewer than Kn edges using the efficient max-flow computation on partial flow networks [2] and the number of iterations for each Φ is around $5 \sim 15$ based on a computation order proposed in [2]. Practically, the runtime of our algorithm is in the order of $K^2 n^2 \log^2 n$, the space requirement is in the order of Kn. Our experiments show that the optimal solution for s38417 with 9800 gates and 1500 FFs can be computed in 20 minutes on a Sun Ultra2 with 256MB memory.

3.3 Mapping Generation

After computing Φ_{min} based on binary search and obtaining the label pairs $(L^s(v), R(v))$ of all nodes v under Φ_{min}, the last step 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 final mapping solution based on the K-cuts computed during the label computation. In the final mapping solution, all the POs of the original circuit are LUT roots and, if v is an LUT root, all the nodes in the node-cut set of the K-cut of v are LUT roots. Those LUT roots can be computed as follows: starting with a first-in-first-out (FIFO) queue with all the POs, 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 the LUT roots v. Suppose each LUT has $O(C)$ nodes for a constant $C \ll n$, where n is the number of nodes in the original network, the constructed network has $O(Cn)$ nodes. We compute a forward retiming $\{\Re(v)\}$ on the new network, where $\Re(v) = \lceil \frac{L^s(v)}{\Phi_{min}} \rceil - 1 \le 0$ for LUT roots v and $\Re(u) = \Re(v) + w$ for $u^w \in \overline{X}_v$ or $\Re(u) = 0$ for $u^w \in V(X_v, \overline{X}_v)$ in each LUT rooted at v. After retiming, all the FFs within each $\overline{X}_v$ will be moved forward outside the cone, because $w^r(u^w \rightsquigarrow v^0) = w + \Re(v) - \Re(u) = 0$ for all $u^w \in \overline{X}$. So we can collapse each $\overline{X}_v$ into a K-LUT and the final circuit has a clock period of no more than Φ_{min} as follows.

Theorem 6 $\{\Re(v)\}$ *is a legal forward retiming to achieve clock period* Φ_{min}.

Since the retiming is a forward retiming, an equivalent initial state can be computed in linear time with circuit simulation based on the approach in [15]. 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. The forward retiming on the constructed circuit can be done in $O(CK^2 n^2)$ time since $|\Re(v)| \le O(Kn)$ for each v and each K-input node can be simulated in $O(K)$ time [15].[7] So the mapping generation with forward retiming and initial state computation can be finished in $O(CK^2 n^2)$ time.

[7] To be precise, a node can be simulated in $O(l)$ time, where l is the number of variables to represent the function of the node, for example, the number of literals on cover table representation or the number of variables in the BDD representation.

4 Experimental Results

The TurboMap-frt algorithm has been implemented in C language and incorporated into the SIS package [13]. Our test set consists of 14 MCNC FSMs and 4 ISCAS'89 benchmarks. The initial circuits are shown in Column Original of Table 1, in which N means the number of nodes and F means the number of FFs, respectively, in the circuits.

Our experiments were performed on a Sun Ultra2 with 256MB memory. K was set to be 5. All the mapping results of TurboMap-frt were computed and verified by *verify_fsm* of SIS [13], except the 4 largest ones which were verified by simulations with input sequences of 3008 random vectors due to huge amount of memory (>1GB) needed by *verify_fsm* for those cases. We compared TurboMap-frt with FlowMap-frt and TurboMap [2]. FlowMap-frt is based on the FlowMap algorithm [1]. It first map each combinational subcircuit bounded by FFs independently with the FlowMap algorithm. Then it merges the mapped LUT subcircuits with the original FFs and performs a postprocessing of forward retiming for clock period minimization. TurboMap [2] computes *optimal* mapping with *general* retiming solutions for synchronous sequential circuits. The initial states of TurboMap solutions were computed with SIS [13] based on the algorithm in [15]. The experimental results are shown in Table 1, where Columns LUT and FF list the numbers of LUTs and FFs, respectively, in the final mapping solutions by each approach. Columns Φ list the clock periods of the results. Columns CPU list the CPU time. Those marked with $\star$ are examples for which SIS failed to compute equivalent initial states for TurboMap solutions. Column Best lists the best valid solutions (with computed equivalent initial states) by TurboMap and FlowMap-frt.

The results show that comparing with TurboMap-frt, FlowMap-frt computed results with 20.2% larger clock period. Though TurboMap [2] can compute results with 2.8% smaller clock period, there are 10 out of 18 solutions by TurboMap SIS concludes no equivalent initial states exist or cannot find them due to huge memory requirement (>300MB) and long runtime (>2hours). Even if the best valid solutions by TurboMap or FlowMap-frt are used for comparison, TurboMap-frt still outperforms the two algorithms by 8.6% on the clock periods. On area part, TurboMap-frt used 8.4% and 1.9% less LUTs as compared with FlowMap-frt or TurboMap, respectively. TurboMap-frt also reduced 6% FFs vs. TurboMap. But TurboMap-frt used 33.8% more FFs as compared with FlowMap-frt. In general we think simultaneous mapping with retiming leads to smaller clock period but tends to use more FFs.

5 Conclusions and Future Work

For sequential circuits with initial states, we presented 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 at first and then try to compute equivalent initial states, we compute optimal mapping with forward retiming solutions with equivalent initial states in one step. Our algorithm enables a new methodology of performing backward retiming separately before forward retiming. Since we can compute the optimal mapping with forward retiming solutions, backward retiming can be used to push back all the FFs to PIs as much as possible as long as the equivalent initial states can be computed without taking into consideration of the impact on the final clock period. This approach is also applicable to sequential circuits with partial initial state assignments by allowing FFs with unassigned initial state to be moved backward or forward freely with other FFs.

The experimental results show that TurboMap-frt is very efficient and effective comparing with conventional design flow of mapping each combinational subcircuits separately.

circuit	Original	FlowMap-frt				TurboMap				Best	TurboMap-frt			
	N/F	Φ	LUT	FF	CPU	Φ	LUT	FF	CPU	Φ	Φ	LUT	FF	CPU
bbara	28/10	4	13	10	0.2	3	12	7	0.4	3	3	12	12	0.2
bbtas	15/5	2	7	5	0.1	1	6	4	0.2	1	1	6	4	0.1
dk16	162/5	14	101	5	0.9	14	103	14	3.8	14	14	103	9	1.7
dk17	42/5	2	10	5	0.2	1	6	3	0.4	1	1	6	3	0.2
ex1	140/5	8	83	5	0.7	8	92	21	1.9	8	8	92	20	1.3
ex2	16/7	2	9	7	0.2	⋆1	4	3	0.2	2	1	4	3	0.1
keyb	134/5	10	75	5	0.6	10	79	5	1.6	10	10	81	5	1.0
kirkman	106/5	6	48	5	0.7	⋆5	57	24	1.2	6	5	57	14	0.8
planet1	348/6	19	213	6	2.0	⋆19	201	18	12.5	19	19	199	37	5.0
s1	107/5	7	58	5	0.5	7	63	11	1.2	7	7	56	6	0.7
sand	327/17	16	176	17	1.8	⋆15	178	30	10.6	16	15	176	12	4.3
scf	516/7	14	325	7	2.8	⋆13	304	20	19.8	14	13	301	27	8.8
sse	74/4	7	42	4	0.4	6	45	10	0.9	6	6	44	8	0.5
styr	281/5	17	163	5	1.6	⋆16	168	8	5.2	17	17	168	12	3.2
s5378	1503/164	4	421	204	7.9	⋆4	444	301	51.5	4	4	427	261	40.3
s9234.1	1299/135	6	462	161	8.5	⋆4	498	217	>7200	6	5	441	203	58.8
s15850.1	3801/515	10	1240	504	30.3	⋆8	1161	732	>7200	10	10	1166	621	205.6
s38417	9817/1464	8	3526	1464	561.5	⋆6	3420	2264	1201.8	8	6	3301	2573	1210.6
geomean		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 1: Comparison of TurboMap-frt with FlowMap-frt and TurboMap for 5-LUT. "geomean" lists the geometric means of the results by each approach. The runtime was recorded on a Sun Ultra2 with 256MB memory. Those marked with ⋆ are circuits that SIS failed to compute initial states for TurboMap solutions. Column Original lists the original circuit size, where N and F represent the numbers of nodes and FFs, respectively, in the original circuits.

It can also produce results very close to the optimal mapping with general retiming solutions in terms of both area and clock period.

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. [7]. We plan to accommodate our approach into this framework as well.

6 Acknowledgements

This work is partially supported by National Science Foundation Young Investigator Award MIP9357582 and grants from Xilinx, Lucent Technologies and Quickturn Design Systems under the California MICRO program.

References

[1] J. Cong and Y. Ding. FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs. *IEEE Trans. on Computer-Aided Design of Integrated Circuits And Systems*, 13(1):1–12, 1994.

[2] J. Cong and C. Wu. An Improved Algorithm for Performance Optimal Technology Mapping with Retiming in LUT-Based FPGA Design. In *IEEE International Conference on Computer Design*, pages 572–578, 1996.

[3] J. Cong and C. Wu. *Optimal FPGA Mapping and Retiming with Efficient Initial State Computation.* UCLA-CSD 980016, Technique Report, March 1998.

[4] G. Even, I. Y. Spillinger, and L. Stok. Retiming Revisited and Reversed. *IEEE Trans. on Computer-Aided Design of Integrated Circuits And Systems*, 15(3):348–357, 1996.

[5] H. Fujiwara and S. Toida. The Complexity of Fault Detection: An Approach to Design for Testability. In *FTCS-12*, pages 101–108, 1982.

[6] S. Kundu, L. M. Huisman, I. Nair, and V. Iyengar. A Small Test Generator for Large Designs. In *International Test Conference*, pages 30–40, 1992.

[7] C. Legl, P. Vanbekbergen, and A. Wang. Retiming of Edge-Triggered Circuits with Multiple Clocks and Load Enables. In *International Workshop on Logic Synthesis*, 1997.

[8] C. E. Leiserson and J. B. Saxe. Retiming Synchronous Circuitry. *Algorithmica*, 6:5–35, 1991.

[9] N. Maheshwari and S. S. Sapatnekar. Minimum Area Retiming with Equivalent Initial States. In *IEEE International Conference on CAD*, pages 216–219, 1997.

[10] S. Malik, K. J. Singh, R. K. Brayton, and A. Sangiovanni-Vincentelli. Performance Optimization of Pipelined Logic Circuits Using Peripheral Retiming and Resynthesis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits And Systems*, 12(5):568–578, 1993.

[11] G. D. Micheli. Synchronous Logic Synthesis: Algorithms for Cycle-Time Minimization. *IEEE Trans. on Computer-Aided Design of Integrated Circuits And Systems*, 10(1):63–73, 1991.

[12] P. Pan and C. L. Liu. Optimal Clock Period FPGA Technology Mapping for Sequential Circuits. *ACM Transactions on Design Automation of Electronic Systems*, 4(1), 1999. http://www.acm.org/todaes/V4N1/L166/paper.ps.gz.

[13] 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.* Electronics Research Laboratory, Memorandum No. UCB/ERL M92/41, 1992.

[14] L. Stok, I. Spillinger, and G. Even. Improving Initialization through Reversed Retiming. In *Proc. Euro. Design & Test Conference*, pages 150–154, 1995.

[15] H. Touati and R. K. Brayton. Computing the Initial States of Retimed Circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits And Systems*, 12(1):157–162, 1993.

[16] H. Touati, N. Shenoy, and A. Sangiovanni-Vincentelli. Retiming for Table-Lookup Field-Programmable Gate Arrays. In *FPGA'92*, pages 89–94, 1992.

[17] H. Yotsuyanagi, S. Kajihara, and K. Kinoshita. Retiming for Sequential Circuits with a Specified Initial State and Its Application to Testability Enhancement. *IEICE Trans. INF. & SYST.*, E78-D(7):861–867, July 1995.