Performance-directed Retiming for FPGAs using Post-placement Delay Information

Ulrich Seidl
Technical University of Munich
80290 Munich, Germany

Klaus Eckl
Synopsys GmbH
85609 Aschheim, Germany

Frank Johannes
Technical University of Munich
80290 Munich, Germany

Abstract

In today's deep-submicron designs, the interconnect delays contribute an increasing part to the overall performance of an implementation. Particularly when targeting field programmable gate arrays (FPGAs), interconnect delays are crucial, since they can easily vary by orders of magnitude. Many existing performance-directed retiming methods use simple delay models which either neglect routing delays or use inaccurate delay estimations. In this paper, we propose a retiming approach which overcomes the problem of inaccurate delay models. Our retiming technique uses delay information extracted from a fully placed and routed design and takes account of register timing requirements. By applying physical constraints, we ensure that the delay information remains valid during retiming. In our experiments, we achieved up to 27% performance improvement.

1. Introduction

Field programmable gate arrays (FPGAs) have become a widely accepted technology especially for implementing low volume designs. Their ability to rapidly implement different designs without a time and cost intensive manufacturing process has greatly contributed to their success. Today, even very big designs like processors can be packed on one single FPGA. Common FPGA architectures are, e.g., the Xilinx Virtex and the Altera Apex series.

In this paper, we focus on timing optimization by retiming for the Xilinx Virtex FPGA series. Those FPGAs consist of an array of configurable logic blocks (CLBs) together with programmable interconnects. Each CLB consists of two similar slices whereas each slice contains 2 logic cells (LCs). Each logic cell (see Fig. 1) contains a 4-input look-up table (LUT) that is capable of implementing any combinational function with up to 4 inputs. The output of the LUT feeds one output of the LC directly and another through a flip-flop. The implementation of a circuit on a FPGA is achieved by configuring the CLBs and programming the various routing resources.

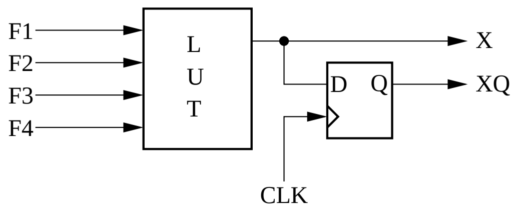

Figure 1. Simplified logic cell (LC)

A major disadvantage of FPGAs compared to ASICs is their inferior performance. The overall performance of a design depends on logic and interconnect delay. Nowadays, the interconnect delay already contributes about 50% to the overall delay. This percentage will even increase when feature sizes continue to shrink. Therefore, the consideration of interconnect delay is mandatory when applying performance oriented optimizations to a design. Due to the numerous different routing resources like switch matrix, long lines, hex lines, double lines, and direct connections, the prediction of these delays is quite a delicate issue. In contrast to the logic delays, which are well known for FPGAs, the interconnect delays can easily vary from a tenth of the delay of a LC to ten times the delay of a LC. Due to the limited routing resources, the distance of two connected LCs is not sufficient for delay estimation. The knowledge of the absolute placement of the source and sink LCs is necessary to precisely predict interconnect delays.

Retiming is a circuit transformation that improves circuit performance by moving registers across combinational elements. For successfully guiding performance-directed retiming the consideration of register timing requirements and the application of precise interconnect delay information is crucial. We propose to extract interconnect delay information after completely placing and routing a design to guide a retiming transformation. To ensure that the delay information remains valid during the optimization step we model physical constraints in an extended retiming graph which still fits well into common retiming algorithms. We demonstrate, how to achieve the optimum solution by applying the exact FEAS retiming algorithm.

The remainder of this paper is organized as follows: Section 2 gives a brief overview of related work. Background material regarding retiming and the FEAS algorithm is given in Section 3. Section 4 describes our approach. We show how to add physical constraints to the retiming graph and how to model register timing requirements. Experimental results are provided in Section 5.

2. Related Work

Retiming is an optimization technique for sequential circuits originally proposed by Leiserson and Saxe [8]. It is based on relocating registers for improving circuit performance or decreasing circuit area. Many improvements have been reported since then. In the following we want to briefly overview the published approaches for performance optimization that are related to our work.

In [13], Shenoy and Rudell show how to efficiently apply retiming to large circuits. They provide acceleration techniques for the original FEAS algorithm. Later on, in [12] another acceleration technique is proposed which exploits the equivalence of clock skew optimization and retiming.

Even et al. [4] alleviate initial state computation by reversing the original FEAS algorithm of Leiserson and Saxe. Through the use of required times instead of arrival times their approach prefers forward register moves while still achieving a solution with equivalent performance. Further on, [20, 15] take the initial state problem into account.

In [16, 17, 6, 7], the authors show that retiming can principally handle more sophisticated timing models. These approaches are preferably used when the so-called "path monotonicity constraint" does not hold. This means that an increasing length of a path does not always lead to an increasing overall delay along that path. Furthermore, the author of [11] shows how to handle register setup and hold-time requirements that introduce short paths.

In [1], retiming is combined with physical planning in order to refine delay information. Neumann and Kunz [9] reveal the attractive properties of the FEAS algorithm and introduce a delay model that considers load dependent delays of fanout trees.

The retiming approaches of [22, 10, 2, 3] target the FPGA architecture. Unfortunately they use the inaccurate unit delay model that neglects the routing delays.

The approach of Touati et al. [19] applies retiming after placement and routing. Thereby they can extract precise routing delay information. They use the very common FPGA architecture Xilinx XC 3000/4000 for their experiments. By applying physical constraints they ensure that the former placement stays applicable. Therefore, the routing delay information remains valid during retiming. However, by applying physical constraints to retiming they restrict the mobility of registers too much, such that the optimization does not yield any improvement. When using the retiming transformation to pipeline a design, improvements are reported.

Singh and Brown [14] apply the *vpr* tool to FPGA designs to determine post placement routing delay information. Unlike Touati et al., they accept a certain degree of logic replication which leads to placement changes. This partially invalidates the routing delay information used during retiming. They try to minimize the discrepancy of their routing delay information by keeping the number of placement changes small.

Although retiming is a very effective circuit optimization technique, it has not yet found wide acceptance in industrial practice. This is mainly for two reasons: First, the verification of a retimed circuit is not straightforward. However, the advances in recent years [18, 21, 5] hopefully will evolve to approaches that can solve the problems that are currently widely encountered. Second, the use of inaccurate delay models that base on theoretic assumptions often lead to unusable results in practice.

This paper addresses the second problem affecting the acceptance of retiming. Through the use of precise interconnect delay information and the consideration of register timing requirements we can achieve convincing results that perfectly reflect the real performance of the final implementation. Like in [19], we completely prevent placement changes, but in contrast to them our approach preserves enough optimization potential to yield significant performance improvements when applied to FPGAs. In order to accomplish experiments we target the modern Xilinx Virtex architecture. However, since our approach is quite generic it is also applicable to other common FPGAs, e.g., the Altera Apex architecture.

3. Basics of Retiming

In the following we introduce basic notations for the retiming transformation.

3.1. Retiming Graph

The retiming graph is an edge-weighted directed graph $G = (V, E, d, w)$. The vertices V represent the combinational logic blocks of the circuit, the edges E correspond to interconnects between the combinational blocks. Each vertex v in V has a nonnegative real-valued label $d(v)$ that denotes the combinational delay of v. Each edge e in E is a directed connection between two vertices u and v and is written as $u \xrightarrow{e} v$. Each edge $u \xrightarrow{e} v$ has a nonnegative integer-valued label $w(e)$ that corresponds to the number of registers along the interconnect between u and v. An example of a circuit with corresponding retiming graph is shown in Fig. 2.

The minimum feasible clock period $\Phi(G)$ of a circuit depends on the delay of the longest path. To improve circuit performance, retiming tries to shorten the longest path.

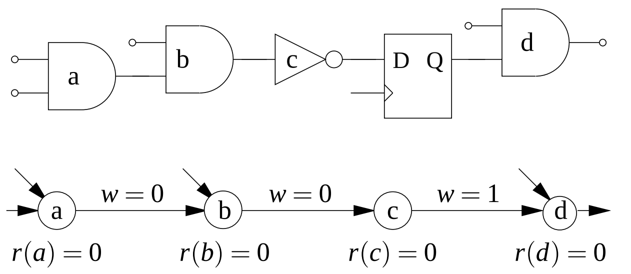

Figure 2. Circuit with retiming graph

Therefore, an integer-valued vertex label $r(v)$ called the retiming value is assigned to each vertex v in V. The retiming value $r(v)$ denotes the number of registers that have to move backwards across the vertex v during retiming.

To improve the performance of a circuit we have to change the retiming values r of the vertices according to rules established by the specific retiming algorithm that is used. The retimed graph $G_r = (V, E, d, w_r)$ can easily be derived from the original graph G by computing the retimed edge weight $w_r(e)$ for each edge $u \xrightarrow{e} v$ in E:

$$w_r(e) = w(e) + r(v) - r(u) \tag{1}$$

To achieve a *legal* retiming the number of registers on each edge $u \xrightarrow{e} v$ has to be nonnegative:

$$w_r(e) = w(e) + r(v) - r(u) \geq 0 \tag{2}$$

Example: We assume that all gates have unit delay. Let the longest path of the circuit in Fig. 2 consist of the vertices a, b, and c. Therefore, the minimum feasible clock period is $\Phi(G) = 3$. To improve the circuit performance we increase the retiming value $r(c)$ by one. According to equation (1) this leads to $w_r(b \xrightarrow{e} c) = 1$ and $w_r(c \xrightarrow{e} d) = 0$. Since (2) is fulfilled, the achieved solution is legal. This retiming transformation moves the register at the output of vertex c back to the input of vertex c. The resulting minimum feasible clock period is $\Phi(G_r) = 2$.

Even though our actual implementation uses the more advanced reverse retiming algorithm [4], for reasons of clarity, we prefer to explain the fundamental ideas of our approach on the basis of the well-known FEAS algorithm. However, the basic ideas of our approach are applicable to both algorithms.

3.2. The FEAS Algorithm

The FEAS algorithm [8] is a renowned approach for performance directed retiming. It checks if there exists a legal retiming for a given clock period c with $O(|V||E|)$ time complexity. If the given clock period is feasible it returns the retiming value r for each vertex. Due to the various improvements like the acceleration technique mentioned in [13] it has gained wide acceptance.

To determine the retiming values r, the FEAS algorithm relies on arrival times. The arrival time $at(v)$ of a vertex v is determined by the largest delay seen along any combinational path that terminates at the output of vertex v:

$$at(v) = d(v) + \max_{u \in FI(v)} \{at(u) \mid w(u \xrightarrow{e} v) = 0\} \tag{3}$$

$FI(v)$ denotes the set of fanin vertices of v. The minimum feasible clock period is given by

$$\Phi(G) = \max_{v \in V} \{at(v)\}. \tag{4}$$

To check if a given clock period c is feasible, the FEAS algorithm executes the following steps on the retiming graph G:

1. For each vertex $v \in V$, set $r(v) = 0$.
2. Repeat the following steps $|V| - 1$ times:
   (a) Compute graph G_r with the current values for r.
   (b) Compute arrival time $at(v)\ \ \forall v \in V$.
   (c) Increase $r(v)$ by 1 for each v where $at(v) > c$.
3. If $\Phi(G_r) > c$, then no feasible retiming exists. Otherwise the proposed clock period c can be achieved by applying the current retiming values r.

The FEAS algorithm only checks a given clock period c for feasibility and in case it is feasible it returns retiming values. In order to find the minimum feasible clock period, the FEAS algorithm usually runs within a binary search loop over a range of clock periods.

4. Performance-directed Retiming for FPGAs

When trying to optimize the performance of FPGA designs with the FEAS algorithm, we first have to enhance it in order to consider interconnect delays. Furthermore, we show how to model register timing requirements. Additionally, we have to ensure that the interconnect delay information remains valid during the optimization process.

4.1. Considering Interconnect Delays

For the Xilinx Virtex FPGA architecture we can assume that the LUTs within the slices are responsible for the logic delay d_{logic} (see Fig. 3). The delay of the internal connections within each slice is negligible. The interconnect delay $d(u \xrightarrow{e} v)$ comprises all delays between a slice output and the input of the successor slice.

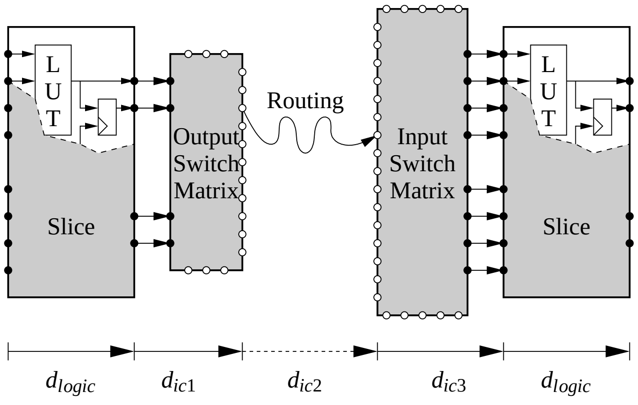

Figure 3. Two Slices with interconnects

Accurate interconnect delays can be determined after placement and routing. To consider them within the FEAS algorithm we have to enhance the arrival time calculation by the interconnect delay. Especially the edges $u \xrightarrow{e} v$ with $w(u \xrightarrow{e} v) > 0$ need special attention. We have to find out how the register interacts with the arrival time of v. The arrival time of registers is defined to be zero. Since the register is usually located in the same slice as the driving LUT u the interconnect delay $d(u \xrightarrow{e} v)$ must be added to the register output. The arrival time $at(v)$ can be determined as

$$at(v) = d(v) + \max \begin{cases} at(u) + & d(e) \;\; ; \;\; w(e) = 0 \\ & d(e) \;\; ; \;\; w(e) \geq 1 \end{cases}$$
$$\text{where} \quad u \in FI(v) \quad \text{and} \quad e : u \xrightarrow{e} v. \tag{5}$$

4.2. Register Timing Requirements

Whenever we have to consider register timing requirements it is not appropriate to determine the minimum feasible clock period $\Phi(G)$ according to equation (4). To ensure proper circuit function we additionally have to take register setup-times t_{set} and sequential propagation delays d_{seq} (sometimes denoted as "Clock-to-Q" delays) into account. The input signal of the register has to be stable t_{set} before the clock-trigger and the output signal becomes available d_{seq} after the clock-trigger. The register timing requirements can be understood as delays which are attributed to a register. Each path that starts at a register is extended by the sequential propagation delay d_{seq} and each path that ends at a register is extended by the setup-time t_{set}. Therefore, by considering register timing requirements we introduce additional delays that move through the retiming graph G whenever a retiming value of a vertex is changed. In the following, we define a new retiming graph $G^* = (V, E, d^*, w)$ that can handle register timing requirements without introducing delays that move through the design during retiming.

Each design is composed of combinational paths that either start at a register or an input vertex pi and either end at a register or an output vertex po. The classification of combinational paths and the resulting delay is given as follows:

1. Path-type: Register $\longrightarrow$ Register:
   $d(p) = d_{seq} + \ldots + t_{set}$
2. Path-type: Register $\longrightarrow$ Output-pad:
   $d(p) = d_{seq} + \ldots + d(po)$
3. Path-type: Input-pad $\longrightarrow$ Register:
   $d(p) = d(pi) + \ldots + t_{set}$
4. Path-type: Input-pad $\longrightarrow$ Output-pad:
   $d(p) = d(pi) + \ldots + d(po)$

Since each FPGA register has the same sequential propagation delay d_{seq} and setup-time t_{set} we can easily introduce a new retiming graph G^* without delays which are bound to registers. Each combinational path of the original retiming graph G is virtually shortened by the sequential propagation delay d_{seq} at the source and the setup-time t_{set} at the sink. Further on, we can neglect the register timing requirements within G^*. The delays d^* of the input vertices $PI(V)$ and output vertices $PO(V)$ of the retiming Graph G^* are determined by

$$\begin{aligned} d^*(v) &= d(v) - d_{seq} \qquad \forall v \in PI(V), \\ d^*(v) &= d(v) - t_{set} \qquad \forall v \in PO(V). \end{aligned} \tag{6}$$

Applying retiming to G^* will result in a clock period $\Phi(G_r^*)$. To obtain the real clock period $\Phi(G_r)$ of the retimed circuit we have to consider the path shortening due to d_{seq} and t_{set}:

$$\Phi(G_r) = \Phi(G_r^*) + d_{seq} + t_{set} \tag{7}$$

4.3. Introducing Physical FPGA Constraints

So far we have shown how to use interconnect delay information to guide a performance-directed retiming of a circuit. Furthermore, we have to take precautions that the delay information gathered after placement and routing remains valid during retiming. Because retiming does not affect the combinational blocks, their delay remains unchanged. Hence, we only have to ensure that the interconnect delays do not change.

The interconnect delays can be divided into 3 groups according to Fig. 3. While the delays of the output and input switch matrix d_{ic1} and d_{ic3} are almost homogeneous throughout the whole design, the delay d_{ic2} heavily varies depending on the absolute placement of the two connected slices. In order to avoid invalidation of interconnect delays during retiming, we have to ensure that the original placement of each slice stays applicable.

In the given FPGA architecture, a path from one LUT to its successor LUT can carry none or one register. No change of interconnect delay will be observable, whether we use the register or not since the modification is restricted to the output switch matrix. Using more than one register on a given connection between two LUTs would imply introducing additional slices which in turn would require undesirable changes in the placement. Therefore, we restrict the weight of edges in the retiming graph G^*:

$$w_r(u \xrightarrow{e} v) \leq 1 \quad \forall e \in E \tag{8}$$

This constraint can be realized in the retiming graph G^* by taking advantage of the fact, that the number of registers on a cycle can not change. By inserting a constraint cycle for each edge, we restrict the edge weights to be at most 1. In doing so, we have to take special precautions not to introduce combinational paths. The introduction of a constraint cycle is illustrated in Fig. 4:

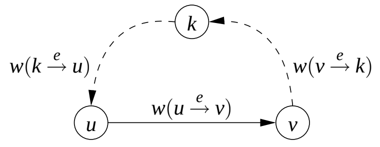

Figure 4. FPGA Constraint cycle

Let $u \xrightarrow{e} v$ be an edge of the retiming graph G^* with weight $w(u \xrightarrow{e} v)$. The dashed edges and the vertex k represent the constraint cycle. The delays of the introduced edges are determined as $d(v \xrightarrow{e} k) = d(k \xrightarrow{e} u) = 0$. The delay $d(k)$ of vertex k is set to the currently targeted clock period c^*. We restrict the influence of vertex k on the arrival times of the vertices u and v by imposing a weight of at least one to its incoming and outgoing edge:

$$\begin{aligned} w(v \xrightarrow{e} k) &\geq 1 \\ w(k \xrightarrow{e} u) &\geq 1 \end{aligned} \tag{9}$$

Thus, we have isolated vertex k from the remaining circuit by at least two registers which intersect the cycle and we have not introduced a combinational path. Furthermore, the initial weights $w(v \xrightarrow{e} k)$ and $w(k \xrightarrow{e} u)$ have to be determined before retiming such that

$$w(u \xrightarrow{e} v) + w(v \xrightarrow{e} k) + w(k \xrightarrow{e} u) = 3. \tag{10}$$

As the delay $d(k)$ of k is c^*, a legal retiming that meets the targeted clock period c^* will yield

$$\begin{aligned} w_r(v \xrightarrow{e} k) &\geq 1 \quad \text{and} \\ w_r(k \xrightarrow{e} u) &\geq 1. \end{aligned} \tag{11}$$

With a total number of three registers on the constraint cycle (10) and with equation (11) we can conclude, that one of the three registers is freely movable. Thus, we have ensured that at most one register is available for the edge $u \xrightarrow{e} v$ and equation (8) holds.

4.4. Limiting the Number of Introduced Constraints

Since the computational effort of the FEAS algorithm is related to the total number of edges and vertices of the retiming graph G^*, we want to minimize the number of constraint cycles. Therefore, we initially start retiming without any physical constraints. Whenever retiming reports feasibility we inspect each edge whether equation (8) is fulfilled. For edges that violate the condition we introduce a constraint cycle. Whenever we detect violating edges we reset the returned retiming values and run the FEAS algorithm for the same clock period again. We can even further reduce the number of introduced constraint vertices by sharing one constraint vertex with all constraint cycles that belong to input edges and sharing another vertex with all constraint cycles that belong to output edges. This is permissible since the retiming value has to be the same for each input vertex as well as for each output vertex.

In our experiments, we observed that we usually can apply all necessary constraints with one additional FEAS run during the binary search loop while introducing less than $3\% * |E|$ constraint vertices on average.

5. Experimental Results

To demonstrate the performance improvements achieved by our approach, we used the sequential circuits from the LGSynth93 benchmark. They were mapped to the Xilinx Virtex architecture with Synopsys design tools. For each design, we chose an individual FPGA size such that it provides just sufficient slices and input-/output resources for the implementation. The designs were placed and routed for optimal performance with the Xilinx *par* tool. By iteratively tightening the performance constraints for *par* we optimized the placement and routing of each design for the minimum achievable clock period $\Phi(G)$.

The best implementation was used to extract interconnect delay information which we took to guide our retiming approach. After applying retiming we took the original placement and reran the router. The best achievable clock period $\Phi(G_r)$ of the retimed circuit is reported in Table 1 and compared to the formerly achieved clock period $\Phi(G)$.

Even though we have achieved improvements for smaller designs, we only report results for designs larger than 100 slices. In our opinion, only large enough designs can reasonably demonstrate the effectiveness of our approach.

Our approach achieves an improvement of up to 27% in terms of the clock period. The average improvement for the benchmark circuits is 10%. Retiming the largest design (s38584.1) with our actual implementation takes 17 seconds cpu-time on a 900 MHz Pentium III.

Table 1. $\Phi(G)$ before and $\Phi(G_r)$ after retiming.

Design	#Slices	$\Phi(G)$	$\Phi(G_r)$	Imprvmnt.
daio_receiver[1]	100	16.86	16.42	2%
diffeq[2]	376	20.04	16.93	16%
dsip	459	14.87	12.87	13%
ecc	121	15.95	12.24	23%
elliptic[2]	950	34.78	25.55	27%
frisc[2]	1161	42.70	34.89	18%
mm30a	149	70.26	70.26	-
parker1986	210	28.49	28.49	-
s1196	101	18.87	18.87	-
s1238	105	19.27	19.27	-
s5378[1]	260	17.97	15.34	15%
s38584.1[1]	1845	20.08	20.05	0%
tseng[2]	314	19.61	17.62	10%
Average				10%

In contrast to Touati et al. [19] we were able to show that retiming can significantly improve the performance of designs implemented on a common FPGA architecture. Since the authors of [14] seem to use an artificial FPGA architecture, our results are hardly comparable to theirs.

6. Conclusion

In this paper we have shown how to predictably increase circuit performance by applying retiming to a circuit implemented on a FPGA. We have exactly modelled register timing requirements within the FEAS algorithm and used delay information extracted after placement and routing to guide the retiming transformation. Furthermore, by applying physical constraints we have ensured that the original placement can be kept and that the delay information used during optimization remains valid. By effectively minimizing the number of introduced constraints the computational effort remains low. In our experiments, we achieved up to 27% performance improvement.

References

[1] J. Cong and S. K. Lim. Physical planning with retiming. In *ICCAD Conference*, pages 2 – 7, 2000.

[2] J. Cong and C. Wu. An improved algorithm for performance optimal technology mapping with retiming in LUT-based FPGA design. In *ICCD Conference*, pages 572–578, 1996.

[3] J. Cong and C. Wu. FPGA synthesis with retiming and pipelining for clock period minimization of sequential circuits. In *DAC Conference*, pages 644–649, 1997.

[1]Design initially contains edges with $w(e) > 1$. This occurs when using a not explained feature of a slice that enables bypassing the LUT and directly feeding the register. Constraints according (8) have been omitted for such edges.

[2]Design originally contains bi-directional input-output buffers which were split into uni-directional buffers to enable retiming.

[4] G. Even, I. Y. Spillinger, and L. Stok. Retiming revisited and reversed. *Trans. on CAD of Integrated Circuits and Systems*, pages 348–357, March 1996.

[5] S.-Y. Huang, K.-T. Cheng, and K.-C. Chen. AQUILA: An equivalence verifier for large sequential circuits. In *ASPDAC Conference*, pages 455–460, 1997.

[6] K. N. Lalgudi and M. C. Papaefthymiou. DELAY: An efficient tool for retiming with realistic delay modeling. In *DAC Conference*, pages 304–309, 1995.

[7] K. N. Lalgudi and M. C. Papaefthymiou. Retiming edge-triggered circuits under general delay models. *Trans. on CAD of Integrated Circuits and Systems*, pages 1393–1408, dec 1997.

[8] C. E. Leiserson and J. B. Saxe. Optimizing synchronous systems. *Journal of VLSI and Computer Systems*, pages 41–67, Spring 1983.

[9] I. Neumann and W. Kunz. Placement driven retiming with a coupled edge timing model. In *ICCAD Conference*, pages 95–102, 2001.

[10] P. Pan and C. L. Liu. Optimal clock period FPGA technology mapping for sequential circuits. In *DAC Conference*, pages 720–725, 1996.

[11] M. C. Papaefthymiou. Asymptotically efficient retiming under setup and hold constraints. In *ICCAD Conference*, pages 396–401, 1998.

[12] S. S. Sapatnekar and R. B. Deokar. Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits. *Trans. on CAD of Integrated Circuits and Systems*, pages 1237–1248, oct 1996.

[13] N. Shenoy and R. Rudell. Efficient implementation of retiming. In *ICCAD Conference*, pages 226–233, 1994.

[14] D. P. Singh and S. D. Brown. Integrated retiming and placement for field programmable gate arrays. In *Int. Symposium on Field-Programmable Gate Arrays*, pages 67 – 76, 2002.

[15] V. Singhal, S. Malik, and R. K. Brayton. The case for retiming with explicit reset circuitry. In *ICCAD Conference*, pages 618–625, 1996.

[16] T. Soyata and E. G. Friedman. Retiming with non-zero clock skew, variable register, and interconnect delay. In *ICCAD Conference*, pages 234–241, 1994.

[17] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr. Incorporating interconnect, register, and clock distribution delays into the retiming process. *Trans. on CAD of Integrated Circuits and Systems*, pages 105–120, jan 1997.

[18] D. Stoffel and W. Kunz. Record & play: A structural fixed point iteration for sequential circuit verification. In *ICCAD Conference*, pages 394–399, 1997.

[19] H. Touati, N. Shenoy, and A. Sangiovanni-Vincentelli. Retiming for table-lookup field-programmable gate arrays. In *Int. Symposium on Field-Programmable Gate Arrays*, pages 89–94, 1992.

[20] H. J. Touati and R. K. Brayton. Computing the initial states of retimed circuits. *Trans. on CAD of Integrated Circuits and Systems*, pages 157–162, jan 1993.

[21] C. A. J. van Eijk. Sequential equivalence checking without state space traversal. In *Design Automation and Test in Europe (DATE)*, pages 618–623, 1998.

[22] U. Weinmann and W. Rosenstiel. Technology mapping for sequential circuits based on retiming techniques. In *Euro-DAC Conference*, pages 318–323, 1993.