ELSEVIER

INTEGRATION, the VLSI journal 22 (1997) 1–21

INTEGRATION the VLSI journal

Retiming: Theory and practice

Narendra Shenoy

Synopsys Inc., 700 E. Middlefield Road, Mountain View, CA 94043, USA

Received 31 October 1996

Abstract

Retiming is a technique for optimizing sequential circuits. It repositions the registers in a circuit leaving the combinational portion of circuitry untouched. The central objective of retiming is to find a circuit with the minimum number of registers for a specified clock period. There are two common variants of this theme; minimizing the clock period without regard to the number of registers in the final circuit or minimizing the number of registers in the final circuit with no constraints on the clock period.

Over a decade has elapsed since Leiserson and Saxe first presented a theoretical formulation to solve this problem for single-clock edge-triggered sequential circuits. The proposed algorithms have polynomial complexity. Since then research efforts have focussed on incorporating retiming in a synthesis framework, addressing issues that arise due to retiming, and extending the domain of circuits for which retiming can be applied. This paper presents a survey of retiming as it evolved from a theoretical formulation to a powerful optimization technique available in commercial tools. The issues that arise for a practical implementation of the basic retiming formulation are discussed. Some of the relevant research issues and key contributions are presented.

1. Introduction

Retiming is a sequential logic optimization technique. Leiserson and Saxe provided the first formulation and theoretical solution to this problem in 1983 [25], although their later paper [26] has the most complete overview of this work. Retiming uses the flexibility provided by repositioning memory elements to optimize a circuit for one of several objective functions:

(1) *min-period*: minimize the clock period of the circuit,
(2) *min-area*: minimize the number of registers in the circuit,
(3) *constrained min-area*: minimize the number of registers in the circuit subject to a maximum constraint on the clock period, or indicate failure if the target period cannot be achieved.

As a means of motivating and introducing the concept of retiming, consider a simple example in Fig. 1. Assume that each gate has the delay shown inside it. The solid rectangles represent edge-triggered registers. A single clock signal is used to drive the clock pins of registers. The best clock period for such a circuit (neglecting clock skew and set-up time of registers) is given by the maximum delay of a path consisting of gates. The clock period for the circuit shown in Fig. 1 is 6 units. An equivalent circuit with 3 registers and a clock period of 4 units can be obtained by repositioning registers as shown in Fig. 2. This circuit has the minimum number of registers. On the

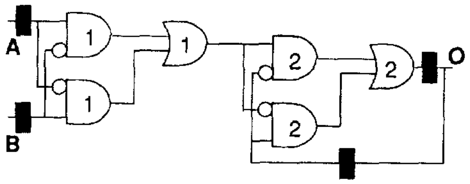

Fig. 1. A simple circuit.

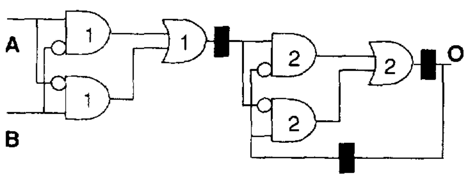

Fig. 2. Retiming for minimum registers.

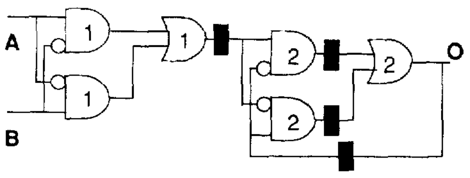

Fig. 3. Retiming for minimum period.

other hand, the minimum period achievable by moving registers is 2 units at a cost of 4 registers as shown in Fig. 3. Thus, a simple re-configuration of memory elements yields designs with differing area costs (number of registers) and performance (clock period). Retiming exploits this trade-off to provide solutions for varying clock periods.

The survey is organized as follows. Section 2 provides a theoretical foundation for retiming. The exposition for min-period retiming is provided first as it enables the development of the constrained min-area retiming formulation. A practical implementation of these algorithms is presented in Section 3. Issues related to the verification of a retiming operation are discussed in Section 4. The focus of Section 5 is the notion of equivalence between a circuit and its retimed version. The equivalence between clock skew and retiming is elucidated in Section 6. Efforts to extend the model for circuits are summarized in Section 7. Section 8 explores the use of retiming with other optimization techniques in logic synthesis. We draw some conclusions on the subject in Section 9.

2. Problem definition and assumptions

A sequential circuit is an interconnection of logic gates and memory elements which communicates with its environment through primary inputs and primary outputs. A sequential circuit can be represented by a directed graph $G(V,E)$, where each vertex v corresponds to a gate v. Each directed edge e_{uv} represents a flow of signal from the output of gate u at its source to the input of gate v

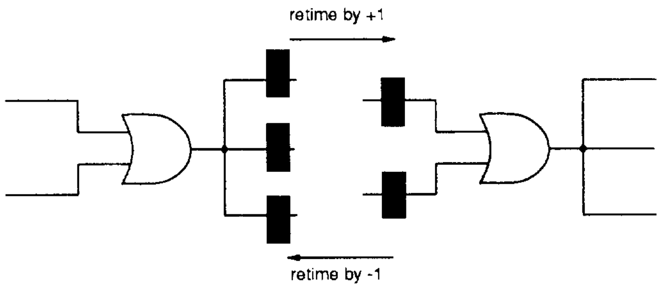

Fig. 4. Retiming a gate.

at its sink. Each edge has a weight $w(e_{uv})$ which indicates the number of registers that the signal at the output of gate u must propagate through before it is available at the input of gate v. Each vertex v has a constant delay $d(v)$. If there is an edge from vertex u to vertex v, u is called a fanin of v and v is called a fanout of u. The set of fanouts (fanins) of u is denoted by $\mathrm{FO}(u)$ $(\mathrm{FI}(u))$. A special vertex called the host vertex is introduced in the graph with edges directed from the host vertex to all vertices that represent primary inputs and edges directed from all vertices representing primary outputs to the host vertex. The host vertex and its associated edges capture the interaction of the circuit with its environment. The host vertex has a delay of 0 and all edges directed into and from the host vertex have edge weights of 0.

A retiming is a labeling of the vertices $r: V \rightarrow Z$; where Z is the set of integers. The weight of an edge e_{uv}, after retiming is denoted by $w_r(e_{uv})$ and is given by

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

A positive (negative) retiming label $r(v)$, for a vertex v, represents the number of registers moved from its outputs (inputs) towards its inputs (outputs). A retiming of zero implies no movement of registers. Fig. 4 shows an AND gate being retimed by $+1$ from left to right. A retiming of the gate on the right by -1 yields the original configuration of memory elements. The retiming of a circuit is an assignment of retimings to all the combinational gates in the circuit.

A path p is defined as a sequence of alternating vertices and edges, such that each successive vertex is a fanout of the previous vertex and the intermediate edge is directed from the former to the later. A path can start and end at vertices only. The existence of a path from vertex u to vertex v is represented as $u \rightsquigarrow v$. The weight of a path $w(p)$ is the sum of the edge weights for the path. The delay of a path $d(p)$ is the sum of the delays of the vertices on the path. A 0-weight path p, is a path with $w(p) = 0$. The clock period c is determined by the following equation:

$$c = \max_{p \mid w(p)=0} \{d(p)\}. \tag{2}$$

The goal is to either minimize c (min-period retiming), or minimize $\sum_{e \in E} w_r(e)$ (min-area retiming), or to minimize $\sum_{e \in E} w_r(e)$ subject to a given clock period c (constrained min-area retiming).

Retiming makes the following assumptions:

(1) The delay $d(u)$ of vertex u is non-negative for all $u \in V$.
(2) The edge-weight $w(e)$ of edge e is non-negative for all $e \in E$.
(3) Every directed cycle in G contains a register.

(4) All memory elements are edge-triggered devices, clocked by the same signal. All registers latch the input data on the same clock edge. In addition, the clock signal undergoes identical skew to all registers.
(5) Complex registers in the circuit can be modeled as simple D flip-flops and associated logic.
(6) A register is assumed to be ideal; it has 0 set-up time, 0 hold requirement and infinite drive.
(7) The delay of a gate is constant and does not change with the change in loading caused by a rearrangement of registers.
(8) The environment ignores the outputs of the machine until after the application of a reset sequence which establishes the internal state of the machine starting from a random power-up state.

Among the assumptions summarized above, the first six are easily handled. Extension of the delay model to capture pin-to-pin delays and separate rise and fall are trivial for the algorithms considered here. A practical issue that has been often neglected in retiming is the modeling of complex registers in a design. A design typically contains a mixture of simple D registers, registers with complex logic at the synchronous data input pin, registers with conditional latching (load enable pin), registers with synchronous set/reset pins, and registers with asynchronous set/reset pins. The simplest model for correct retiming is to disallow retiming for circuits with asynchronous set/reset registers. Synchronous set, synchronous reset and some complex registers can be adequately modeled by D registers and some adjoining logic on the data input pins. The newly introduced logic is mapped and retiming is carried out on a circuit with only D registers. A nominal set-up time is easily incorporated in the algorithm by providing a margin around the target clock period. Hold time considerations are not critical for single clock systems with edge-triggered devices and can be easily corrected after retiming. Early in the design process, worst-case slew information can be incorporated by providing a conservative margin around the target clock period. Recall that the central step of optimization involves re-positioning of memory elements. More precise skew considerations are difficult to handle this early in the design process, though efforts have been made in this direction.

An unchanging delay for each gate is a serious restriction. Since registers are repositioned, loading at the gate outputs and the outputs of registers cannot be predicted a priori. If gate delays are not assigned carefully, a circuit after retiming can have a clock period larger than the estimated value. Techniques have been developed that address these issues and we defer this discussion for later (Section 7).

The environment assumption is extremely restrictive and has been investigated by several researchers. Touati et al. [49] consider retiming under the constraint that every register has a synchronous set or reset pin. In this case, the known initial state is modified into an equivalent state after retiming. This model is often deemed to be restrictive. Correctly functioning circuits do not require the state of every register to be established in a single cycle; to do so is wasteful in terms of area and delay. Often it is safe to prefer a model where no registers have synchronous set or reset pins, (i.e. the synchronous reset logic is modeled as part of the surrounding combinational logic). Using this model, Singhal et al. [45] show that without any assumptions on the environment, retiming does not preserve a replace-ability property: namely, the retimed circuit may not be a valid replacement for the original circuit (see Section 5). However, there exist a large class of circuits (known as finite-memory circuits) for which lax constraints are imposed by their environment. Such circuits are commonly found in data-paths of processors and signal processing designs. Hence, retiming is an optimization step than can be safely applied to these circuits.

2.1. Problem formulation

The central retiming algorithms reside on the framework provided by Leiserson and Saxe [26]. An important concept is the definition of 2 matrices, the W matrix and the D matrix. They are defined as follows:

$$W(u,v) = \min_{p:u \rightsquigarrow v} \{w(p)\} \tag{3}$$

and,

$$D(u,v) = \max_{p:u \rightsquigarrow v \text{ and } w(p) = W(u,v)} \{d(p)\}. \tag{4}$$

The matrices are defined for all pairs of vertices (u,v) such that v is reachable from u by a sequence of edges and the path does not include the host vertex. Intuitively, an entry of the W matrix determines the minimum latency (in clock cycles) for data flowing from u to v. An entry in the D matrix gives the maximum delay from u to v for the minimum latency. W may be computed by solving the all-pairs shortest paths problem on the graph G. Since the edge weights are non-negative and the shortest path is required for each vertex pair, other algorithms like Dijkstra's can be used for efficiency. Computing D differs from computing W due to 2 facts:
(1) each entry $D(u,v)$ depends on $W(u,v)$, and
(2) the vertex weights are non-negative and the longest path is required for a given $W(u,v)$ value. Leiserson et al. [25, 26] suggest a single algorithm based on Floyd–Warshall to compute the W and D matrices simultaneously. Each edge $e: u \to v$ is assigned an ordered pair $(w(e), -d(u))$. The additions in the Floyd–Warshall algorithm are replaced by component-wise additions of the ordered pair. The comparison for the minimum value is replaced by comparison of the ordered pair: the second term of the pair being compared only when a tie is obtained on comparing the first term. If (x,y) be the value for (u,v) entry of the Floyd–Warshall matrix, then $W(u,v) = x$ and $D(u,v) = -y + d(v)$.

2.1.1. Minimum period retiming

The objective is to obtain a circuit with the minimum clock period without any consideration to the area penalty due to an increase in the number of registers. Leiserson et al. introduce 3 algorithms to solve the min-period retiming problem. The retiming constraints for a target clock cycle c translate into 2 sets of inequalities:

(1) Non-negativity of edge-weights after retiming requires $w_r(e_{uv}) \geqslant 0$, $e_{uv} \in E$. In other words,

$$r(v) - r(u) \geqslant -w(e_{uv}) \quad \forall e_{uv} \in E. \tag{5}$$

(2) Correct clocking at a clock period c requires that the delay of 0-weight paths after retiming be less than c. In fact, the correct invariant is that for all paths $u \rightsquigarrow v$ with $D(u,v) > c$, we require $W_r(u,v) \geqslant 1$; in other words,

$$r(v) - r(u) \geqslant -W(u,v) + 1 \quad \forall u \rightsquigarrow v \text{ such that } D(u,v) > c. \tag{6}$$

The two sets of constraints from Eqs. (5) and (6) are in the difference of 2 variables form. It is well known that such a set of constraints can be solved using the Bellman–Ford relaxation technique developed for the "shortest path on a graph" problem [24]. There are $O(|V|^2)$ inequalities in $O(|V|)$

variables, and can be solved in $O(|V|^3)$ time using the Bellman–Ford algorithm. This leads to the first (of three) algorithm for min-period retiming which requires $O(|V|^3 \log |V|)$ time.

```
Compute $D$ for $G(V,E)$
Sort distinct elements of $D$ into a list $L$
Binary search the list $L$ to check if the entry $l$ of $L$ is feasible
Feasibility is checked by constructing inequalities 5 (fixed) and inequalities 6
(dependent on $l$)
    and using Bellman–Ford to solve them
```

A more efficient technique known as the relaxation method is the second method. Let $\Delta(v)$ denote the largest delay seen along any combinational path that terminates at the output of v. It denotes the latest arrival time at v.

$$\Delta(v) = d(v) + \max_{u \in \mathrm{FI}(v) \text{ and } w(e_{uv}) = 0} \{\Delta(u)\}. \tag{7}$$

It can be shown that the clock period c is given by the expression

$$c = \max_{v \in V} \{\Delta(v)\}. \tag{8}$$

The relaxation algorithm has a $O(|V||E|)$ subroutine which determines if a retiming exists for a given clock period c.

```
For each $v \in V$ set $r(v) = 0$.
For $|V| - 1$ times {
    Compute retimed edge weights (Eq. (1))
    Compute $\Delta(v)$, for all $v \in V$ (Eq. (7))
    For all $v \in V$, such that $\Delta(v) > c$, set $r(v) = r(v) + 1$
}
Compute retimed edge weights (Eq. (1))
If $\max_{v \in V}\{\Delta(v)\} > c$, then there is no feasible retiming
    else the current values of $r$ yield a legal retiming
```

The reader is referred to [41] for a proof of correctness of this algorithm. The algorithm requires $O(|V|)$ iterations; in each iteration, computing for all vertices takes $O(|E|)$ steps. If an upper bound on the clock period is T (the clock period of the input circuit is an upper bound), this yields a complexity of $O(|V||E| \log T)$ for the min-period retiming problem. Note that the relaxation algorithm for min-period retiming does not require the W and D matrices to be computed.

The third technique is based on mathematical programming framework and is of theoretical interest only.

2.1.2. Constrained min-area retiming

In practice, it turns out that there are several solutions to the min-period retiming problem with a large variation in the number of registers amongst them. This is expected since the formulation does not impose any restrictions on the number of registers. The aim of constrained min-area retiming is to constrain the number of registers for a target clock period. Under the assumption that all registers have the same area, the min-area retiming problem reduces to seeking a solution with the minimum number of registers in the circuit. The number of registers in a circuit after a retiming r is given by

$$R = \sum_{e \in E} w_r(e)$$
$$= \sum_{e \in E} w(e) + \sum_{v \in V} r(v)(|\mathrm{FI}(v)| - |\mathrm{FO}(v)|).$$

The first term in the summation is a constant representing the number of registers in the original circuit. The second term is a linear sum of the retiming labels. Constraints for retiming to be valid are the same as in Eqs. (5) and (6). These are linear constraints and the objective function is also a linear function of the retiming variables, so linear programming techniques can be used to solve this problem. However Leiserson et al. [26] indicate that the dual of this problem is an instance of minimum cost circulation on a graph, for which $O(|V||E| \log |V| \log(|V|C))$ algorithms exist (where C is the largest cost of an edge in the circulation graph). They also indicate that an initial feasible solution can be obtained directly from the problem.

The primal to dual translation is described in detail since it is key to understanding the techniques which speed the implementation. Consider the coefficient of $r(v)$ in the objective function; let

$$\alpha_v = |\mathrm{FI}(v)| - |\mathrm{FO}(v)|. \tag{9}$$

It represents the per unit cost for $r(v)$; i.e. the cost of moving 1 register from all the outputs (thereby saving $|\mathrm{FO}(v)|$) to all its inputs (thereby incurring a cost of $|\mathrm{FI}(v)|$). The primal problem P is the retiming formulation

$$\min \left(\sum_{v \in V} \alpha_v r(v) \right)$$
$$r(v) - r(u) \geqslant -w(e_{uv}) \quad \forall e_{uv} \in E,$$
$$r(v) - r(u) \geqslant -W(u,v) + 1 \quad \forall D(u,v) > c.$$

The edges in the retiming graph that originate due to the translation from the circuit are termed "circuit" edges. Consider the retiming graph, augmented by edges (called "period" edges) from u to v for all pairs (u,v), for which the last inequality is required (i.e. the inequalities arising from the clock period constraints). Note that the number of period edges can be very large and may destroy the sparsity of the graph; this issue is dealt with later. Let the edge weights for the period edges be $W(u,v) - 1$. So each constraint in the primal is associated with an edge in the graph and each primal variable is associated with a vertex in the graph.

Let the dual variable associated with each constraint (and edge) be denoted by $s(e)$; it represents a flow on edge e. The dual problem can be formulated as

$$\max \left(\sum_{e_{uv} \in E} w(e_{uv}) s(e_{uv}) \right)$$
$$\sum_{w \in \mathrm{FO}(v)} s(e_{vw}) - \sum_{u \in \mathrm{FI}(v)} s(e_{uv}) = \alpha_v \quad \forall v \in V,$$
$$s(e_{uv}) \geqslant 0 \quad \forall e_{uv} \in E.$$

The dual cost is associated with each edge and is the edge weight ($w(e_{uv})$ for a circuit edge or $W(u,v) - 1$ for a period edge). Create a source vertex S, a sink vertex T and an edge from T directed to S. If $\alpha_v > 0$, put an edge from S to v, if $\alpha_v < 0$ put an edge from v to T. Such an edge

Table 1
Edge capacity and edge costs for minimum cost circulation

Edge type		Cost	Capacity
Circuit edge	e_{uv}	$w(e_{uv})$	$+\infty$
Period edge	e_{uv}	$W(u,v) - 1$	$+\infty$
Capacity edge	$S \rightarrow v$	0	α_v
Capacity edge	$v \rightarrow T$	0	$-\alpha_v$
Sink to source edge	$T \rightarrow S$	$-\sum_{e \in E} w(e) - 1$	$\sum_{\alpha_v > 0} \alpha_v$

is called a capacity edge and has 0 dual cost. The flow on this edge is constrained to be less than or equal to α_v. The edge from T to S has a capacity which equals

$$\sum_{v \in V:\, \alpha_v > 0} \alpha_v = -\sum_{v \in V:\, \alpha_v < 0} \alpha_v. \tag{10}$$

It is assigned a cost which is less than the negative of the largest cost of any cycle in the graph. The rationale for doing so is that it forces all the capacity edges to be saturated and hence respect the equality in the dual. Its magnitude is chosen to be one more than the number of registers in the original circuit. The capacity edges have 0 cost and hence do not contribute to the maximum cost along any cycle. A period edge always has a cost which is less than the cost of any path (consisting of circuit edges only) between its source and its sink. Thus, an upper bound for the cost of a cycle considering only the circuit edges is a valid bound. The goal is to compute a minimum cost circulation on this graph. Table 1 summarizes the cost and capacity for the different edge types. Note that setting $s(e_{uv})$ to 1, for every edge $e_{uv} \in E$, gives a feasible solution to the dual constraints. Registers on more than one fanout of a gate can be shared. The formulation so far does not capture this effect. Register sharing can be correctly modeled by augmenting the circuit graph with dummy vertices and edges (see [26, 41] for details).

3. From theory to practice

From a design perspective, the only interesting objective function is the constrained min-area problem. There can be no realistic applications of retiming to digital design where minimizing either the area or delay metric for the circuit while leaving the other parameter unconstrained is useful. Also, a solution to the constrained min-area problem can be used to solve the min-period problem (via binary search) and the min-area problem (by specifying a large clock period). For these reasons, the primary goal is an efficient solution to the constrained min-area problem.

However, the min-period retiming problem is important in its own right as a step in solving the constrained min-area problem. This is because the min-period problem is computationally less intensive than the constrained min-area problem and because it provides a lower-bound for the best delay achievable by the constrained min-area problem. It also provides an initial solution for the constrained min-area problem which allows for more efficient solutions to that problem. For this reason, emphasis is given to performing both min-period and constrained min-area optimization efficiently.

Although the algorithms described in Section 2 are polynomial in the circuit size, naive implementations suffer the worst-case ($O(n^3)$ time and $O(n^2)$ space) for all circuits. Consequently, early implementations of retiming by De Micheli [6], Bartlett et al. [1] and Sentovich et al. [42] are unable to provide optimal solutions for large circuits. Munzner and Hemme [37] propose a heuristic using minimal cuts of acyclic graphs, for converting combinational circuits into pipelined data paths. This section details the implementation issues which have been overcome to successfully solve min-period retiming and constrained min-area retiming for circuits with 10 000 combinational cells. For min-period retiming, the bottleneck is the requirement that we iterate $O(|V|)$ times to prove that there is no feasible retiming. For constrained min-area retiming, the bottleneck is that even when the graphs are sparse, the W and D matrices are dense. Further, the number of clock period edges which are implied by the retiming equations indicate that the retiming graph augmented with clock period edges will be dense.

3.1. Minimum period retiming

Let us focus on the relaxation algorithm to determine if c is a feasible clock period. It is empirically observed that if c is feasible then the retiming labels converge rapidly before completing $|V|-1$ iterations. On the other hand, one cannot determine that a clock period c is infeasible until all $|V|-1$ iterations have been exhausted and the retiming labels have failed to converge. Thus, any hope of speeding up min-period retiming must focus on detecting if a clock period is infeasible before completing the requisite $|V|-1$ iterations if possible.

A little investigation into the failure for convergence indicates that one or more cycles in the retiming graph are the culprits. One approach by Shenoy et al. [43] suggests identifying sub-portions of the graph using a predecessor heuristic. This is a well-known technique for determining convergence and has been used for other applications [48]. Each vertex maintains a pointer to a fanin vertex (or edge) known as the predecessor vertex (edge). During the update of retiming labels in the relaxation method (line 5 of the algorithm), this pointer is set to the fanin vertex (edge) that caused the update. Thus, portions of circuit that are updated repeatedly constitute the underlying sub-graph determined by predecessor vertices (and edges). Every few iterations (say 10), the predecessor graph is examined for cycles. If a cycle exists, it is tested for a retiming at the current clock period. Failure to converge yields a certificate that the current clock period is unachievable for the circuit. A successful convergence on the cycle provides no conclusive answer. An implementation of this approach, reported in [43], is able to speed-up the naive implementation of min-period retiming by a factor of 50–100. Using the cycle obtained as a certificate of infeasibility to update the lower bound on the clock period is useful, as the bias eliminates feasibility checks at clock periods less than the bound which are guaranteed to be infeasible.

An alternative approach is to determine a good lower bound for the clock period before applying retiming. Chakradhar [2] proposes using linear programming techniques to determine the lower bound that involves the cycles in a retiming graph. A binary search using the relaxation algorithm is then applied for clock periods greater than the bound.

3.2. Constrained minimum area retiming

Minimum area retiming poses 2 major hurdles;
(1) computing the W and D matrices, and
(2) implementing minimum cost circulation.

3.2.1. Computing W and D matrices

The method proposed by Leiserson et al. [26] to compute the W and D matrices has $O(|V|^3)$ time complexity and an $O(|V|^2)$ space requirement. The W and D matrices are required to add the clock period edges which in turn are required to solve the minimum cost circulation problem using cost scaling techniques. Further, all of the clock period edges must be added prior to solving the circulation problem.

The number of clock period edges greatly increases the density of the original graph. However, only a small subset of the period edges implied by Eq. (6) are required for the computation; the rest are redundant constraints. The original algorithm has two drawbacks: the $O(|V|^2)$ memory and the inability to prune the clock period edges. Shenoy et al. [43] propose an algorithm which has a worse complexity than the original formulation, but whose memory is $O(|V|)$ and is able to generate a smaller set of constraints for sparse circuit graphs.

Eq. (6) describes the conditions under which clock period edges need to be added to the retiming graph. In the original formulation of constrained min-area retiming, clock period edges are required between all vertices u and v such that

$$u \rightsquigarrow v \quad \text{and} \quad D(u,v) > c. \tag{11}$$

To see why a smaller set of clock period edges is sufficient for constrained min-area retiming, note that if

$$r(v) - r(u) \geqslant -W(u,v) + 1 \tag{12}$$

is true for a sub-path of a path, then it is also true for the entire path. Hence, a period edge need only be added to vertex v, reachable from w, such that

$$D(w,v) > c \quad \text{and} \quad D(w,u) \leqslant c \quad \forall u \text{ lying on } w \rightsquigarrow \mathrm{FI}(v). \tag{13}$$

Consider a vertex w. We are interested in the period edges that have their source as w. To do this, it is necessary to examine only a single row of the W and D matrices (i.e., the row $W(w,.)$ and the row $D(w,.)$). The set of vertices can be partitioned into disjoint sets depending on the value of $W(w,.)$ as shown in Fig. 5. The directed edges in Fig. 5 represent paths to other vertices from w. The dashed curve represents the set of vertices v that meet the condition of Eq. (13). We can ignore any period edges between w and the fanouts of such vertices. Thus, only some of the entries of $W(w,.)$ and $D(w,.)$ need to be computed. The elements of $W(w,.)$ can be computed using Dijkstra's algorithm since the edge weights $w(e_{uv}) \geqslant 0$, $\forall e_{uv} \in E$. The computation of $D(w,.)$ is complicated due to 2 facts; the dependence on $W(w,.)$ and the non-monotonicity of the gate delays (akin to finding the longest path in a graph with positive edge weights).

A technique to efficiently generate a row of the W and D matrices and to add a minimal set of constraint edges that satisfy Eq. (13) is described by Shenoy et al. [43]. For some of the large circuits described in their experiments, it is almost impossible to finish computing the W and D matrices in a reasonable amount of time using a Floyd–Warshall implementation. With the modified algorithm, the execution time is comparable to minimum cost circulation computation.

3.2.2. Minimum cost circulation using cost-scaling

A disadvantage of cost-scaling techniques is that the graph cannot change during the computations; consequently, all the period edges must be determined before starting the cost-scaling algorithm.

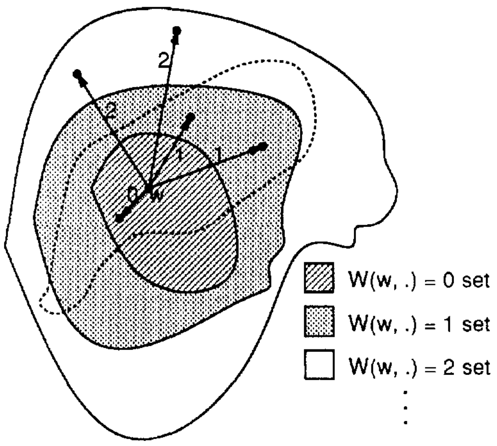

Fig. 5. Disjoint partitions of the vertices.

The implementation in [43] is based on the generalized cost-scaling framework of Goldberg and Tarjan [14] and has a complexity of $O(|V||E|\log|V|\log(|V|C))$ where C is the largest cost in the graph i.e. one more than the number of registers in the circuit. If $|V|$ is 10 000, and we restrict ourselves to 32-bit integer arithmetic, C must be less than 200 000; since C is the number of registers in the circuit, this is a reasonable assumption. The algorithm operates by maintaining an error from optimality and successively halving it. At the start this error is $|V|C$. However, if an initial flow is known, the value of C can be reduced to be the minimum value that has to be added to the cost of each edge so that the graph does not have a negative cost cycle. This fact reduces the value of C by an order of magnitude. An initial flow for the circulation graph can be constructed easily. For most circuits C turns out to be less than 10, effectively making the algorithm independent of C. As an aside, one has to be careful with the edge capacities. In general, these are real numbers and can cause convergence problems. For the sake of stability, the edge capacities are scaled to integers using a factor which is a function of the least common multiple of the number of fanouts of vertices in the graph.

It has been observed that the min-area solution for the minimum clock period can have fewer registers than a min-period solution by as much as a factor of 5 [43]. Solving the constrained min-area retiming problem is expensive but has been performed on large industrial designs with reasonable run times [43]. On a set of real industrial designs (3 control, 5 data-flow, 2 mixed), constrained minimum area retiming gave an average improvement of 17% over the existing clock period [31]. The area increased on an average by 4%. The design sizes range from 3000 gates to 30 000 gates.

4. Verification of retimed circuits

Any optimization technique must be adequately supported by a verification process to ensure that the optimization process did not introduce errors. For combinational synthesis, techniques using binary decision diagrams, automatic test pattern generation and simulation are useful for checking equivalence. Verification of sequential circuits can be done by implicit state computation or by extensive simulation. For large circuits these approaches are either infeasible or guarantee only a limited coverage. A practical approach is to reduce the problem to combinational verification treating the register inputs (outputs) as pseudo-inputs (pseudo-outputs). Retiming breaks this assumption since registers are relocated in the circuit. Hence, verifying a retimed circuit with the original specification is an extremely hard problem. However, if we restrict attention to verifying a circuit G with its retimed version G_r, Shenoy et al. [44] show the following properties of retiming can be exploited to simplify the problem:

(1) Retiming does not change the nature or the topology of the combinational gates in the circuit.
(2) Retiming is valid if and only if the number of registers in all cycles of G is preserved. Although the number of cycles in G can be exponential it suffices to check the property (of conservation of registers) on a smaller set of cycles called the fundamental cycle set (which can be efficiently enumerated).

A fundamental cycle set is a set of cycles that permit any cycle in the graph to be expressed as a linear combination of the fundamental set. There are many fundamental cycle sets associated with a graph; one such set can be easily found by computing a spanning tree of the graph, and using the non-tree edges to define the set (see [44] for details). Fundamental cycles are interesting when a linear property is defined on a cycle in the graph. It can be shown that if such a property is preserved on a fundamental set, it is also preserved on all cycles. We define this property is defined to be the sum of the edge-weights in a cycle. The validity of retiming will be demonstrated by identifying a retiming label s with each gate that converts G to G_r or providing a cycle in G in which the register count is not preserved. The procedure is as follows:

```
Find a spanning tree $T$ rooted at $v_h$ on the underlying undirected sub-graph of $G_r$.
Let $s(v_h)=0$
For $e_{uv} \in T$,
    $s(v)=s(u)+w_r(e_{uv})$ if $u$ is closer than $v$ to $v_h$ in $T$
    $s(u)=s(v)-w_r(e_{uv})$ if $u$ is farther than $v$ from $v_h$ in $T$
For $e_{uv} \notin T$, verify $s(v)-s(u)=w_r(e_{uv})$.
```

The verification process is extremely fast and takes $O(|E|)$ time. It is recommended that any implementation of retiming verify the retimed edge-weights. This is specially encouraged for the min-area retiming problem since the dual problem is solved and primal feasibility is only true when the dual has found an optimum solution.

5. A note on equivalence of retimed circuits

Retiming was first formulated by Leiserson et al. [25] in the context of systolic systems. They assumed that the environment of the circuit could be modified to wait for a fixed number of cycles (dependent on retiming) before applying the inputs. In the context of retiming a gate level net-list, the issue of equivalence needs to be re-visited.

The equivalence of a circuit and its retimed version depends on the environment of the circuit. Some registers in a typical design may be connected to a global set or reset line. This line is asserted at some instant after power-up by the environment to force the circuit into a known initial state. This is true for control sections of a design. A register with set/reset capabilities is typically larger.

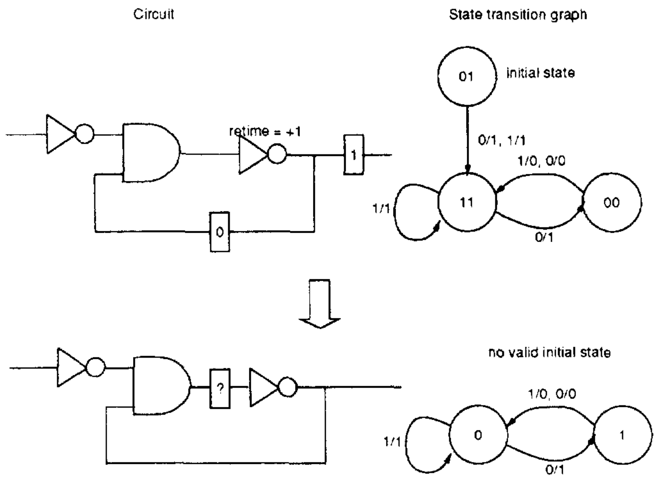

Fig. 6. Example of retiming to an inadmissible initial state.

Its use in data-path components such as multipliers is not favored as it is costly and unnecessary. Sometimes even control sections of a design may not contain such registers. The initialization of the circuit is then done by providing an initialization sequence that forces the circuit into a well-defined state irrespective of the state that the circuit powered up. Thus, there are also portions of a design that contain registers without set/reset signals. In either case, the environment must be unable to distinguish the retimed circuit from the original version from its input–output behavior. This is the classical notion of equivalent finite state machines. Section 5.1 deals with the case when the initial state is relevant for the circuit. Section 5.2 discusses the notion of *safe replace-ability* when registers do not have set/reset lines.

5.1. Retiming in the presence of well-defined initial states

It is interesting to note that retiming can be considered as a re-encoding of the original circuit. Given an original circuit and a retiming, the objective is to determine the initial values of the relocated registers so that the behavior of the circuit is preserved. Consider the example provided by Touati et al. [49] in Fig. 6. Observe that the register after retiming does not have a valid initial state.

Assume that the state transition graph for the circuit can be extracted. If the retiming labels for all gates in the circuit are negative, i.e. registers are only moved forward then a simulation (forward implication) of the network can be used to compute the new initial state. Any retiming can be converted to a legal retiming that satisfies this constraint by subtracting $\max_{v \in V} r(v)$ from each retiming label. This may cause the retiming of the host vertex $r(v_h)$ to be negative. This corresponds to moving the initial state of the original circuit to a state in the transition graph (which may not necessarily exist), from which a sequence of $|r(v_h)|$ transitions leads to the original initial state. When this condition is not satisfied, extra logic has to be inserted in the circuit to permit retiming of the initial state. This has been adequately discussed in works by Bartlett et al. [1] and Malik et al. [33]. It is not possible prior to retiming the circuit to determine the amount of modification (if any) that is required. Adding circuitry after retiming may invalidate the retiming results. In [49] the approach is to add circuitry before retiming to enable the initial state be reachable from itself. This ensures that for any finite $N > 0$, there is a sequence of transitions that lead to the original initial state. This is conservative, but the process of retiming includes the delays and circuit topology due to the possible modifications.

The biggest drawback of the previous approaches is that the state transition graph needs to be extracted. For large circuits this is undesirable. Even et al. [11] propose a technique that avoids the state extraction problem. They modify the relaxation technique for min-period retiming to define *reverse* retiming as follows:

(1) Instead of computing $\Delta(v)$, the required time $\delta(v)$ is computed at each vertex.

$$\delta(v) = -d(v) + \min_{u \in \mathrm{FO}(v)} \begin{cases} \delta(u) & \text{if } w(e_{uv}) = 0, \\ c & \text{if } w(e_{uv}) > 0. \end{cases} \tag{14}$$

(2) Each vertex v with $\delta(v) < 0$, has its retiming label decremented.

This algorithm has the same convergence properties of the relaxation technique and in addition minimizes the maximum normalized lag defined by $\max_{v \in V}(r(v) - r(v_h))$. Thus, this procedure guarantees finding a retiming with all negative retiming labels (forward movement) if one exists. The initial state calculation for such circuits is easy. For circuits with a positive normalized lag, a procedure based on backward justification and forward implication [21] is utilized to compute the initial state values. If an initial state cannot be found due to conflicts in the backward justification at a gate v, the *reverse retiming* problem is re-solved by bounding the retiming label with a constraint edge in the graph.

5.2. Retiming in the absence of well-defined initial states

Singhal et al. [45] demonstrate that when retiming is applied to a design without an explicit initial state, the post-retimed circuit will not necessarily be equivalent to the original circuit. They show that the retimed circuit satisfies a notion of *delayed* equivalence instead. If the environment is willing to stall observation of the retimed circuit outputs for a known number of clock cycles then the retimed circuit will provide an identical input–output behavior as the original circuit. They identify the cause of such behavior to retiming across multi-fanout gates whose ranges are strict subsets of the possible values. For such gates, it is not possible to justify a set of output values by means of an input vector; and are hence termed *non-justifiable* gates. An inverter-buffer gate available in most ASIC libraries is a simple example of such a gate. A junction in a circuit with a fanout of n gates can also be considered a non-justifiable gate since it maps $B \to \{(0,\ldots,0),(1,\ldots,1)\}$. In Fig. 7, for the circuit on the left the output z is 0 at all clock periods irrespective of the state of the register at the input of the multiple output gate. If the multiple output gate is retimed by -1, then the registers are placed at the outputs. It is conceivable that upon power-up both the registers have a value of 1, thereby making the output z take a value of 1 in the first clock period. Thus, the 2 circuits have differing behavior. It is possible to extend this behavior to demonstrate that an initializing sequence for the unretimed circuit may not be an initializing sequence for the retimed circuit. Retiming can also be shown to violate the preservation of test vectors that check for faults in the circuit [19].

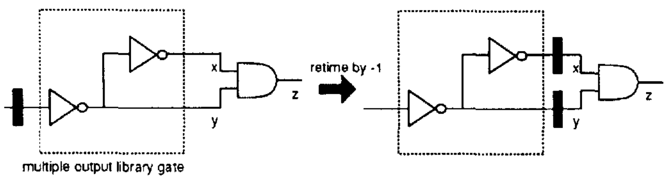

Fig. 7. Example of retiming causing incorrect behavior.

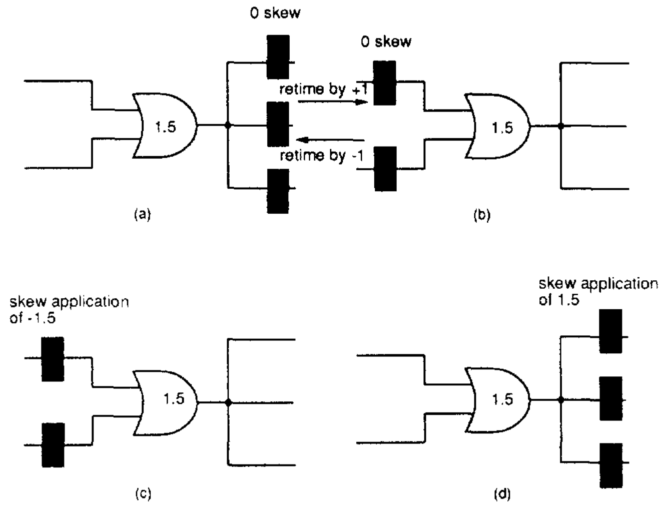

Fig. 8. Retiming and clock skew.

6. Clock skew and retiming

So far, we have assumed that the latching edge of the clock arrives at all registers at the same instant of time. In practice, this need not be true. The *skew* on a clock line is defined as the maximum interval of time between the arrivals of the same latching edge at a pair of registers. Skew arises due to routing of the clock signal over long distances and possibly asymmetric loads at the different end points. Designers can choose to either ensure a zero skew in the design (see [50, 27]) or utilize the skew as a resource to improve the design performance (see [13]). The addition of a positive (negative) clock skew to a register is similar to relocating the register to a cut-set of its fanin (fanout) cone of logic. The equivalence between retiming and skew has been observed and used in works by Martin [35], Lockyear et al. [29] and Chao et al. [4]. Deokar et al. [7] provide a proof for the equivalence between retiming and assigning clock skews which has been informally used by several researchers. Consider the circuit in Fig. 8(a). The effect of retiming the gate in Fig. 8(a) by $+1$ while maintaining 0 clock skew to the registers (Fig. 8(b)) is equivalent to delaying the arrival of the clock signal to the registers by the delay of the gate (Fig. 8(d)). Conversely retiming Fig. 8(b) by -1 is equivalent to advancing the arrival of the clock signal by the delay of the gate (Fig. 8(c)). They provide an alternate view to the min-period retiming problem. Instead of assigning a set of lags to combinational gates in the circuit, Deokar et al. [7] focus on determining an optimum set of skews to registers so as to minimize the clock period. A positive (negative) skew assignment is then translated to a forward (backward) retiming of the register.

The skew optimization problem seeks to assign a skew x_i to with each register i, so that the clock period is minimized. The problem is formulated on a graph $G_1(V_1, E_1)$, such that

(1) each register i has a vertex v_i,
(2) there is an edge $v_i \rightarrow v_j$, if there is a path of combinational gates from register i to register j, and
(3) each edge $v_i \rightarrow v_j$ has a weight d_{ij},

$$d_{ij} = \max_{p \in \text{combinational path from } i \text{ to } j} (d(p)). \tag{15}$$

For correct circuit operation, the skews associated with $v_i \rightarrow v_j$ must satisfy

$$x_i + d_{ij} \leqslant x_j + c \quad \text{long path constraint} \tag{16}$$

and

$$x_i + d_{ij} \geqslant x_j \quad \text{short path constraint.} \tag{17}$$

See [5,48] for a detailed explanation of these constraints. The objective is to minimize c. The approach by Deokar et al. is to neglect the short path constraints and solve the following linear programming instance:

$$\min c$$

$$x_j - x_i \geqslant d_{ij} - c.$$

Once the values for the skews have been found, the registers are relocated. If the skew is positive (negative) the registers are moved backward (forward). The authors [7] provide a detailed constructive technique for relocating registers so that the skew at each register is no more than half a gate delay. This in turn implies that the final clock period after retiming is no worse than the optimal clock period for the skew problem plus the maximum gate delay in the circuit. Through experimental results the authors demonstrate that this approach to minimum period retiming is linear in the circuit size (gate count + initial register count).

7. Timing models

A very simple delay model for the retiming problem has been assumed so far. A gate is assumed to have a fixed delay throughout the retiming process. It is not too difficult to extend this model to include input pin to output pin delays for a gate. All the algorithms discussed for retiming so far can be applied with little modification. Efforts are made by Lalgudi et al. [22,23] to extend gate delays to be load dependent. Memory elements which are assumed to be ideal are permitted to have realistic set-up/hold times and skews in efforts by Soyata et al. [47,46], Lalgudi et al. [22] and Lockyear et al. [30]. The inclusion of register delays is made by some of these efforts [47,46,22].

The extended delay model was first described by Soyata et al. in [47]. Each edge in the retiming graph is annotated with the following characteristics.

(1) The skew from the clock pad to a register (if one is located) on the edge.

(2) The internal delay from the clock pin to the next state output of the register.
(3) The set-up of the register.
(4) Any interconnect delay associated with the edges.
A fundamental property associated with retiming that fails in this scenario is that the delay of a path ceases to be a monotonic function of the path. Soyata et al. propose a branch and bound approach to solve the problem. Lalgudi et al. provide an elegant solution to the problem by reducing it to a monotonic integer programming formulation in [22].

Gate delays are rarely the crisp numbers that are assumed in most algorithms (including retiming). It is often easier from a design perspective, to think of gate delays as fuzzy sets. Karkowski et al. [20] extend the classical formulation of retiming to handle gate delays described as fuzzy sets. Their model relies on the use possibilistic programming. Using a special linear mapping function, the possibilistic programming formulation is reduced to a classical retiming problem with linear increase in the number of variables.

One of the most interesting extensions to the retiming model is to include circuits with multiple clocks. An excellent description of timing constraints for correct behavior of such circuits can be found in [40]. Designs with multiple clocks use level-sensitive latches as memory elements; which behave differently from edge-triggered registers in a fundamental manner. A latch permits data at the input pin to stream to the output pin during the active phase of the clock (typically when the clock is high). The use of latches also introduces a minimum path delay constraint in the formulation arising from the hold time requirement at each latch. Efforts to extend retiming for level-sensitive clocked circuits can be found in [18, 28]. There have also been extensions to include a sophisticated delay model [29, 30, 17] for multiple clock designs.

8. Retiming and other optimizations

Retiming assumes that the combinational structure (nature of gates and the topology) is fixed. The quality of the final solution could be further improved by changing the combinational logic and then applying retiming. Peripheral retiming introduced by Malik et al. [33, 32] is an extension of retiming where negative register count is temporarily permitted on a restricted set of edges of the circuit graph. The concept of peripheral retiming is best explained by a simple example (see Fig. 9) taken from [32]. Since the circuit is cyclic, a cut indicated by the dashed line in (Fig. 9(a)) is introduced to obtain an acyclic circuit (Fig. 9(b)). In Fig. 9(c) the registers are moved forward, temporarily borrowing a register from the environment from at input b. The NOR gate (whose output is marked by X) is untestable for a stuck at 0 fault in the purely combinational circuit. After this simplification (Fig. 9(d)), the registers are relocated to yield a valid retiming (Fig. 9(e)). Finally, the wires broken by the cut are rejoined to yield the new circuit (Fig. 9(f)).

Peripheral retiming permits an edge connected to the host vertex to have a negative register count. This deficit is incurred by borrowing registers from the environment. A circuit which has been successfully peripherally retimed has all registers on the peripheral edges (i.e. edges connected to and from the host vertex). The main motivation for peripheral retiming is to apply well-understood combinational optimization techniques to the circuit; since all registers have been relocated to the periphery. Finally, the registers are retimed to new locations in the optimized circuit. The final retiming step has to yield a legal retiming. Thus, negative edge weights (register counts) are not

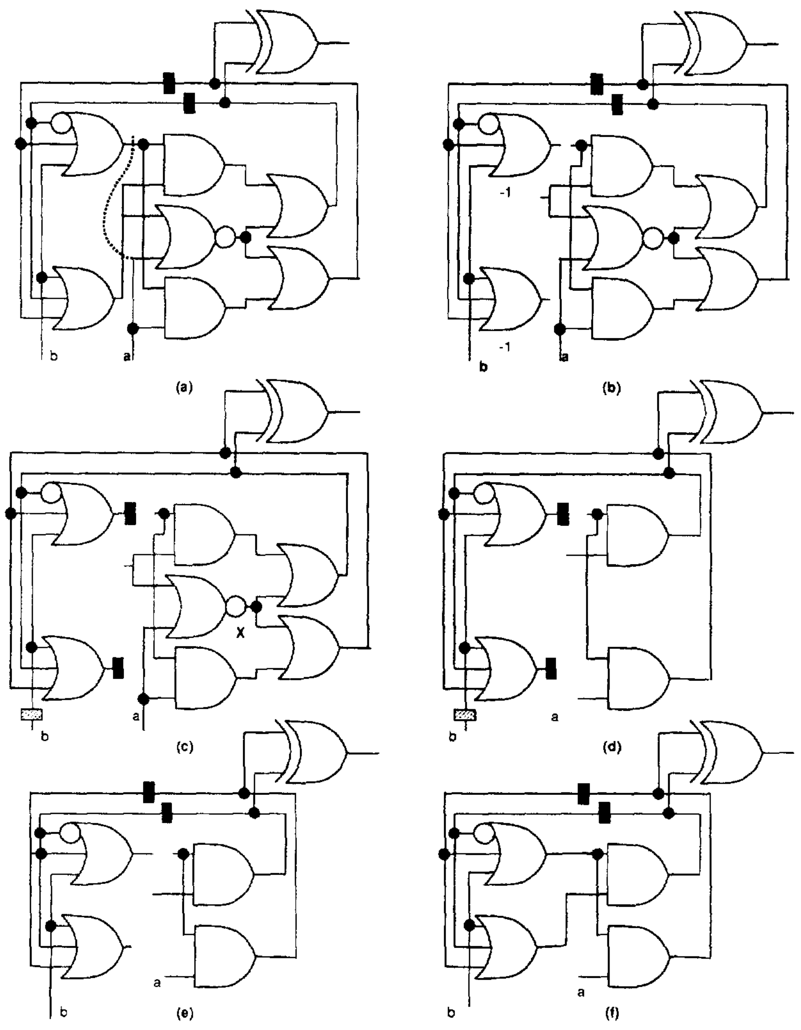

Fig. 9. Peripheral retiming.

permitted in the final retiming step. This technique, called as retiming and resynthesis, is proposed and demonstrated in [33, 34]. More recent advances in peripheral retiming are discussed by Fehlauer et al. [12]. Peripheral retiming can only be applied to acyclic circuits. To extend the concept of retiming and resynthesis to general sequential circuits, a partitioning strategy is invoked. The main contribution by Dey et al. [8] is to describe a method of breaking cycles in the circuit graph based on corolla partitioning. The notion of circuit partitioning using corollas is described in [9] and is based on the analysis of signal reconvergence. The corollas are then individually subjected to a process of retiming and resynthesis.

It has also been noted that certain circuit topologies can cause bottlenecks that prevent retiming. Dey et al. [10] propose a technique to identify portions of combinational logic that can be restructured to enable retiming to take place. The constraints for the combinational optimization are derived carefully so as to enable the ensuing retiming satisfy the clock period constraint. A more complete treatment of determining the constraints for the combinational sub-portions can be found in the work by Chakradhar et al. [3]. Grodstein et al. [15] incorporate retiming in the process of technology mapping of circuits using dynamic programming techniques. Pan et al. [38] exploit retiming during technology mapping for lookup-up table based FPGA (field programmable gate array) synthesis. The relationship between state encoding and retiming has also been explored by Malik [32].

Retiming has also been investigated as a potential technique for power minimization in a circuit. Monteiro et al. [36] suggest using the movement of registers to block signals with high activity. Since the output of a register is synchronized with the latching edge, the output of a register is effectively shielded from the signal transitions at the input. This lessens the transition activity in the fanout cone of logic of the register, thereby decreasing power consumption. For a design that is sensitive to a power budget, we recommend that the constrained min-area retiming technique be used. We have noticed that a simple reduction in the number of registers (with no sacrifice in performance) can reduce the power significantly. Decreasing the load capacitance on the clock net, which is often a net with high signal transition activity, can yield considerable power savings.

9. Conclusions

In this survey, we have focussed on retiming as a gate level optimization technique. The concept of retiming has been applied in other related fields. Most notable is its use as a potential transform in deciding architectural trade-offs [39, 16]. Over the last dozen years retiming has been an important topic in most research conferences. We have presented a brief review of its original formulation, described issues for a practical implementation and provided a survey of related topics. Retiming has come a long way from being a purely theoretical optimization technique to one of practical use.

There are still several issues that prove to be barriers in the full-fledged adoption of retiming by the design community. Inability to handle designs with a rich mix of registers (including those with asynchronous signals) is a problem. Although the extensions to multi-clock designs has been made, a practical implementation capable of handling large circuits has proven to be elusive. Verification of a circuit that has undergone retiming with its original description remains an intractable problem.

Acknowledgements

The author wishes to thank Don MacMillen and Reiner Genevievre for helping solve several issues during the application of retiming to real industrial designs.

References

[1] K.A. Bartlett, G. Boriello, S. Raju, Timing optimization of multi-phase sequential logic, *IEEE Trans. Comput. Aided Des.* 10 (1991) 51–62.

[2] S.T. Chakradhar, Optimum retiming of large sequential circuits, *Proc. 8th Internat. Conf. on VLSI Design*, 1995, pp. 135–140.

[3] S. Chakradhar, S. Dey, M. Potkonjak, S.G. Rothweiler, Sequential circuit delay optimization using global path delays, in: *Proc. Design Automation Conf.* 1993, pp. 483–489.

[4] L.-F. Chao, E.H.-M. Sha, Retiming and clock skew for synchronous systems, in: *IEEE Internat. Symp. Circuits and Systems*, 1994, pp. 1.283–1.286.

[5] M.R. Dagenais, N.C. Rumin, Automatic determination of optimal clocking parameters in MOS VLSI circuits, in: *Advanced Res. in VLSI: Proc. of the 5th MIT Conf.* 1988, pp. 19–33.

[6] G.de. Micheli, Synchronous logic synthesis: Algorithms for cycle-time minimization, *IEEE Trans. Comput. Aided Des.* (1991) 63–73.

[7] R.B. Deokar, S.S. Sapatnekar, A fresh look at retiming via clock skew optimization, in: *Proc. Design Automation Conf. ACM/IEEE*, 1995, pp. 310–315.

[8] S. Dey, F. Brglez, G. Kedem, Partitioning sequential circuits for logic optimization, in: *Proc. Internat. Workshop on Logic Synthesis*, 1991.

[9] S. Dey, F. Brglez, G. Kedem, Partitioning sequential circuits for logic synthesis, *IEEE J. of Solid-State Circuits* 26(3) (1991) 350–360.

[10] S. Dey, M. Potkonjak, S.G. Rothweiler, Performance optimization of sequential circuits by eliminating retiming bottlenecks, in: *Proc. Internat. Conf. Computer-Aided Design*, 1992, pp. 504–509.

[11] G. Even, I.Y. Spillinger, L. Stok, Retiming revisited and reversed, *IEEE Trans. Comput. Aided Design*, to be published.

[12] E. Fehlauer, S. Rulke, G. Franke, EPR – A synthesis tool for speed optimization, *Tech. Report, FhG IIS Erlangen-Department EAS Dresden*, 1995.

[13] J. Fishburn, Clock skew optimization, *IEEE Trans. Comput.* 39 (1990) 945–951.

[14] A. Goldberg, E. Tardos, R.E. Tarjan, Network flow algorithms, *Tech. Report No. STAN-CS-89-1252, Department of Computer Science*, 1989, Stanford University, CA.

[15] J. Grodstein, E. Lehman, H. Harkness, H. Touati, B. Grundmann, Optimal latch mapping and retiming within a tree, in: *Proc. Internat. Conf. on Computer-Aided Design, IEEE*, New York, 1994.

[16] S. Hassoun, C. Ebeling, Architectural retiming: Pipelining latency-constrained circuits, in: *Proc. Design Automation Conf.* 1996, pp. 708–713.

[17] A.T. Ishii, Retiming gated clocks and precharged circuit structures, in: *Proc. Internat. Conf. Computer-Aided Design*, 1993, pp. 300–307.

[18] A. Ishii, C.E. Leiserson, M.C. Papaefthymiou, Optimizing two-phase level-clocked circuitry, *Adv. Res. VLSI: Proc. 1992 Brown/MIT Conf.*, MIT Press (1992) pp. 245–264.

[19] M.A. Iyer, D. Long, M. Abramovic, Identifying sequential redundancies without search, *Proc. Design Automation Conf.* 1995, pp. 457–462.

[20] I. Karkowski, R.H.J.M. Otten, Retiming synchronous circuitry with imprecise delays, in: *Proc. Design Automation Conf.* 1995, pp. 322–326.

[21] S. Kundu, L. Huisman, I. Nair, V. Iyengar, L. Reddy, A small test generator for large designs, in: *Proc. Internat. Test Conf.* 1992, pp. 30–40.

[22] K.N. Lalgudi, M. Papaefthymiou, DelaY: An efficient tool for retiming with realistic delay modeling, in: *Proc. Design Automation Conf. ACM/IEEE*, 1995, pp. 304–309.

[23] K.N. Lalgudi, M. Papaefthymiou, Efficient retiming under a general delay model, in: *Advanced Research in VLSI: Proc. 1995 MIT/UNC-Chapel Hill Conf. IEEE Press*, New York, 1995.

[24] E.L. Lawler, Combinatorial Optimization: Networks and Matroids. Holt, Rinehart & Winston, New York, 1976.

[25] C.E. Leiserson, J.B. Saxe, Optimizing synchronous systems, *J. VLSI and Comput. Systems* (1983) 41–67.

[26] C.E. Leiserson, J.B. Saxe, Retiming synchronous circuitry, *Algorithmica* 6(1) (1991) 5–35.

[27] Y-M. Li, M.A. Jabri, A zero-skew clock routing scheme for VLSI circuits, in: *Proc. Internat. Conf. Computer-Aided Design*, 1992, pp. 458–463.

[28] B. Lockyear, C. Ebeling, Optimal retiming of multi-phase level-clocked circuits, *Adv. Res. VLSI* (1992).

[29] B. Lockyear, C. Ebeling, Minimizing the effect of clock skew via circuit retiming, Uw-cse-93-05-04, University of Washington, Seattle, 1993.

[30] B.E. Lockyear, C. Ebeling, The practical application of retiming to the design of high-performance systems, in: *Proc. Internat. Conf. Computer-Aided Design*, 1993, pp. 288–295.

[31] R. Maheshwary, N. Shenoy, Behavioral retiming boosts RTL design performance, Impact! – *Quarterly Newsletter for users of Synopsys Products and Services*, 3(1) (1996) 4–5.

[32] S. Malik, Combinational logic optimization techniques in sequential logic synthesis, Ph.D. Thesis, University of California, Berkeley CA-94720, November, 1990.

[33] S. Malik, E. Sentovich, R.K. Brayton, A. Sangiovanni-Vincentelli, Retiming and resynthesis: Optimization of sequential networks with combinational techniques, in: *Proc. Hawaii Internat. Conf. on System Sciences*, January 1990, pp. 397–406.

[34] S. Malik, K.J. Singh, R.K. Brayton, A. Sangiovanni-Vincentelli, Performance optimization of pipelined circuits, in: *Proc. Internat. Conf. on Computer-Aided Design, IEEE*, New York, 1990, pp. 410–413.

[35] H.-G. Martin, Retiming by combination of relocation and clock delay adjustment, in: *Proc. European Design Automation Conf.* 1993, pp. 384–389.

[36] J. Monteiro, S. Devadas, A. Ghosh, Retiming sequential circuits for low power, in: *Proc. Internat. Conf. on Computer-Aided Design*, 1993, pp. 398–402.

[37] A. Munzner, G. Hemme, Converting combinational circuits into pipelined data path, in: *Proc. Internat. Conf. on Computer-Aided Design*, 1991, pp. 368–371.

[38] P. Pan, C.L. Liu, Optimal clock period technology mapping for FPGA circuits, in: *Proc. Design Automation Conf.* 1995, pp. 720–725.

[39] M. Potkonjak, J. Rabaey, Maximally fast and arbitrarily fast implementation of linear computations, in: *Proc. Internat. Conf. on Computer-Aided Design*, 1992, pp. 304–308.

[40] K. Sakallah, T. Mudge, O.A. Olukotun, Analysis and design of latch-controlled synchronous circuits, in: *Proc. Design Automation Conf. IEEE/ACM*, 1990, pp. 111–117.

[41] J.B. Saxe, Decomposable searching problems and circuit optimizations by retiming: Two studies in general transformations of computational structures, Ph.D. Thesis, Carnegie-Mellon University, 1985.

[42] E. Sentovich et al., Sequential circuit design using synthesis and optimization, in: *Proc. Internat. Conf. on Computer Design*, 1992.

[43] N. Shenoy, R. Rudell, Efficient implementation of retiming, in: *Proc. Int. Conf. on Computer-Aided Design*, 1994, pp. 226–233.

[44] N.V. Shenoy, K.J. Singh, R.K. Brayton, A.L. Sangiovanni-Vincentelli, On the temporal equivalence of sequential circuits, in: *Proc. Design Automation Conf. ACM/IEEE*, 1992, pp. 405–409.

[45] V. Singhal, C. Pixley, R.L. Rudell, R.K. Brayton, The validity of retiming sequential circuits, in: *Proc. Design Automation Conf. ACM/IEEE*, 1995, pp. 316–321.

[46] T. Soyata, E. Friedman, Retiming with non-zero clock skew, variable register and interconnect delay, in: *Proc. Internat. Conf. on Computer-Aided Design, IEEE*, New York, 1994, pp. 234–241.

[47] T. Soyata, E. Friedman, J.H. Mulligan, Integration of clock skew and register delays into a retiming algorithm, in: *Proc. IEEE Internat. Symp. on Circuits and Systems, IEEE*, 1993, pp. 1483–1486.

[48] T.G. Szymanski, Computing optimal clock schedules, in: *Proc. Design Automation Conf.* 1992, pp. 399–404.

[49] H. Touati, R.K. Brayton, Computing the initial states of retimed circuits, *IEEE Trans. Comput. Aided Des.* (1993), pp. 157–162.

[50] R.S. Tsay, Exact zero skew, in: *Proc. Internat. Conf. on Computer-Aided Design*, 1991, pp. 336–339.