

Fast Minimum-Register Retiming via Binary Maximum-Flow

Aaron P. Hurst, Alan Mishchenko, and Robert K. Brayton

University of California, Berkeley

Abstract— We present a formulation of retiming to minimize the number of registers in a design by iterating a maximum network flow problem. The retiming returned will be the optimum one, which involves the minimum amount of register movement. Existing methods solve this problem as an instance of minimum-cost network flow, an algorithmically and practically more difficult problem than maximum flow. Furthermore, because all flows are unitary, the problem can be simplified to binary marking. Our algorithm has a worst-case bound of $O(R^2E)$, where R is the number of registers and E the number of edges. We demonstrate on a set of circuits that our formulation is 5x faster than minimum-cost-based methods.

Index Terms—Retiming, Sequential Verification, State Minimization, Maximum Flow.

I. INTRODUCTION

RETIMING [13] moves registers over combinational nodes in a logic network, preserving output functionality and logic structure. Retiming can target a number of objectives: (i) minimize the delay of the circuit (*min-delay*), (ii) minimize the number of registers under a delay constraint (*constrained min-register*), and (iii) minimize the number of registers (*unconstrained min-register*). Numerous approaches have been proposed to achieve these goals [13]-[18], with most of the emphasis on the first two objectives.

In this paper, we focus on unconstrained min-register retiming, which has several applications in logic synthesis and verification. In synthesis, minimizing the number of registers can save area and power or be used to ameliorate local congestion. Even if delay constraints are ignored, often any timing violations can be corrected with logic sizing, combinational resynthesis, or intentional clock skewing [8]. In verification, min-register retiming minimizes the number of state variables [12], which reduces the size of the sequential verification problem and has been demonstrated to reduce the difficulty of the verification problem [3]. This may be critical for proof completion.

Although retiming problems are traditionally expressed as general linear programs, they can be solved efficiently as minimum-cost network circulation problems using suitable algorithms. Instead, we propose a retiming method that is based on iterated binary maximum network flow, which is an asymptotically and practically easier problem. The number of iterations required appears to be quite small. Because the result of each iteration is strictly better than the previous one,

the computation can be bounded and still result in an improvement. It was found experimentally that the first iteration of max-flow accounts on average for 90% of the total reduction in the number of registers. This can be used to trade quality for runtime when a problem is large or when fast computation is critical.

To support these claims, we provide experimental results on moderately-sized industrial benchmarks and a few larger artificial ones. They demonstrate the efficiency of the new algorithm: the optimum result can be generated for circuits with more than a million gates in less than a minute and much faster than using existing methods. On pre-optimized benchmark circuits, the average reduction in register count was 11%, ranging between 0% and 63%.

An important feature of our algorithm is that it always returns the minimum-register retiming that is closest to the current position of the registers. If a register in the input circuit cannot be retimed to minimize the total register count, it is not touched. This simplifies the computation of the initial states and minimizes the total perturbation.

The paper is organized as follows. Section II describes the background information and existing approaches to minimum-area retiming. Section III describes the new algorithm. Section IV reports experimental results.

II. BACKGROUND

A circuit is modeled as a directed graph $\mathcal{G}=\langle V,E\rangle$ whose vertices V correspond to logic gates and directed edges E correspond to wires connecting the gates, decomposed into pair-wise connections from gate outputs to inputs. The circuit's external connections are represented by additional primary input (PI) and primary output (PO) vertices. The terms *network*, *graph*, and *circuit* are used interchangeably in this paper.

A node has zero or more fan-ins, i.e. nodes that are driving this node, and zero or more fan-outs, i.e. nodes driven by this node. The transitive fan-out of a vertex v is a subset of all nodes of the network reachable through the fan-out edges from v , captured by the function $\text{TFO}(v): V \rightarrow 2^V$.

A *combinational frame* of the circuit is comprised of the acyclic network between the register outputs / PIs and register inputs / POs. An example of this is illustrated in Figure 1; the inputs (the register outputs) lie on the left, and the outputs (the register inputs) on the right. The registers are duplicated for the ease of illustration.

A. Retiming

The problem of retiming is to relocate the registers in a circuit to optimize some circuit characteristic while preserving output functionality and optionally meeting some additional constraints. The repositioning is captured by a *retiming lag function* $r(v):V \rightarrow \mathbb{Z}$ that describes the number of registers moved backward over each combinational node. There are several formulations of the retiming problem, but for the purposes of this paper, we utilize the linear program (LP) in Equations 1-2 from [13]. $w_i(e)$ is the initial number of registers present on edge e .

$$\min \sum_{\forall e=(u,v)} r(v) - r(u) \quad s.t. \quad (1)$$

$$r(u) - r(v) \leq w_i(e) \quad \forall e=(u,v) \quad (2)$$

The dual of this LP is a minimum-cost network circulation problem and can be solved efficiently using specialized algorithms. Using the method described by Goldberg [9], the minimum-cost flow can be computed in $O\left(VE \log(V^2/E) \log(VC)\right)$ worst-case time, where C is the maximum cost on any edge. The number of vertices and edges in the corresponding network problem is proportional to the size of the combinational circuit.

After a retiming has been determined, an equivalent set of initial values must be computed for the registers. If no such initial state exists, the particular retiming must be rejected or the circuit otherwise altered. We discuss consequences on this computation but refer to [19] for its details.

III. MIN-REG RETIMING ALGORITHM

We introduce an algorithm for optimum unconstrained minimum-register retiming that is based on an iterative maximum network flow problem. This is motivated by the observation that computing the maximum flow through a network is an algorithmically and practically easier problem than determining the minimum-cost circulation. Our algorithm requires repeated iteration, but for practical circuits, the number of iterations is typically quite small.

A. Single Iteration

The core of the algorithm consists of minimizing the number of registers within a single combinational frame. Let us consider only the paths through the combinational logic that lie between two registers (thus temporarily ignoring the primary inputs and outputs). The current position of the registers clearly forms a complete cut through the network, immediately at its inputs. The width of the cut is the initial number of registers.

Consider retiming the registers in only the forward direction through the circuit. As the registers are moved over any of the combinational nodes, the corresponding cut moves forward through the network and may grow or shrink in width as registers are replicated and/or shared as dictated by the graph structure. In the initial circuit, it is evident that any path in the combined graph passes through exactly one register, and any

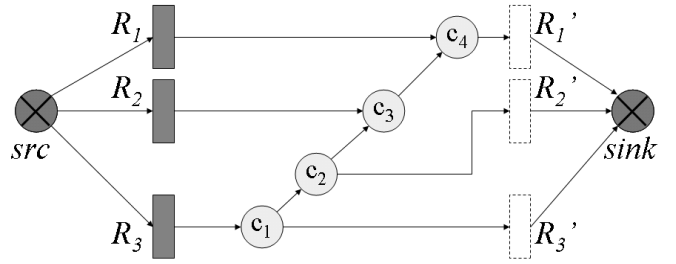


Fig. 1. An example of the flow problem for a small circuit. The combinational elements are light grey; the initial positions of the registers lie to the left, and their inputs are replicated on the right. This is also an example of a network where the minimum-cut in the directed graph is not a valid retiming: the graph can be completely cut with exactly two registers (at the outputs of c_1 and c_4), but this results in a path ($R_3 \rightarrow R_1'$) with altered sequential latency.

valid retiming must preserve this property. If this were not the case, the latency of that path would be altered and the sequential behavior of the circuit changed.

The problem of minimizing the number of registers by retiming them to new positions within the scope of the combinational frame is equivalent to finding a minimum width cut. This is the dual of the maximum network flow problem, for which efficient solutions exist. To construct the flow problem, the flow source is connected to all register outputs, and all of the edges to the register inputs are redirected to the flow sink. For now, the capacity of all edges is assumed to be one.

Solving the general maximum flow problem using the preflow-based method proposed by Cherkassky and Goldberg [4] is $O\left(VE \log(V^2/E)\right)$ in the worst case. Alternatively, the

classical augmenting path-based methods (e.g. [6]) are bounded by the maximum length of an augmenting path times the maximum flow, U : $O(UE)$. Because the flow constraints in our problem are of unit capacity, the width of the input to the graph establishes a worst-case bound of $O(RE)$, where R is the initial number of registers. The worst-case bound on the maximum flow problem using either the preflow- or augmenting path-based algorithm is therefore asymptotically faster than the best known bound on the minimum-cost flow problem on an identical graph structure.

Once the maximum flow has been established, we have available the residual graph that describes the remaining edge capacities with respect to this flow. The residual graph is used to generate a corresponding minimum cut. Via duality, the width of this minimum cut is exactly the volume of the maximum flow and the flow through the saturated edges crossing the cut. To determine its location, the vertices in the network are partitioned into two sets: S , those that are reachable in the residual graph with additional flow from the source, and R , those that are not. Generating this partition is $O(E)$ in the worst case. The partition must be complete cut, because there is no additional flow path from the source to the sink if the maximum has already been assigned. If there exists an edge $u \rightarrow v$ such that $(u \in S \wedge v \in R)$, a register must be

placed on the output of the gate associated with u . The registers are removed from their current locations and placed on the graphs edges that cross the minimum cut.

There may exist multiple cuts of minimum width, but this method always generates the one that is unambiguously closest to the source node. This results in the minimal movement of the registers, simplifying the initial state computation and minimizing the design perturbation.

However, as stated, this procedure may generate an illegal retiming. A cut in a directed graph only guarantees that all paths in the graph are cut *at least* once. This is a necessary but not sufficient condition for the cut to be a valid retiming. We seek the minimum cut in the graph such that all paths are crossed *exactly* once. Figure 1 illustrates an example of this problem. The network flow problem must be altered to eliminate the possibility that a path is crossed more than once.

Reverse edges with unbounded capacity are added in the direction opposite to the constrained edges in the original network. These additional paths may increase the maximum flow (and therefore the size of the minimum cut) but guarantee that the resulting minimum cut will correspond to a legal retiming. For a path in the original graph to cross the finite-width cut more than once from $S \rightarrow R$, there must be at least one edge that crosses from $R \rightarrow S$. If the unbounded reverse edges are also present, this would imply an infinite-capacity edge from $S \rightarrow R$, thus violating the finite cut-width assumption and ensuring that such a cut will not be generated.

It is also needed to account for sharing registers along hypergraph fan-out edges. This requires another simple modification to the network flow problem. Each circuit node is decomposed into two vertices: a *receiver* of all of the former fan-in arcs and an *emitter* of all of the former fan-out arcs. The flow constraints are removed from these edges; instead, a single edge with a unit flow constraint is inserted from the receiver to the emitter. Note that the reverse edges connect pairs of receivers. As above, the unconstrained fan-out edges can not participate in the minimum cut; only the internal edge is available to make a unit contribution to the cut-width. Each node will therefore require at most one register regardless of its fan-out degree. This idea can also be extended to model fan-in sharing as in [1].

The final network for computing the maximum flow computation is depicted in Figure 2.

The unitary flow constraints can also be used to simplify the implementation of the maximum flow solver such that the flow network of Figure 2 need not be explicitly built. Both the preflow- and augmenting path-based techniques can be performed on the original circuit structure with only binary flow marking. Because the flows on the non-unit arcs are unconstrained, they need only be implicitly maintained with pointers to the flow predecessors.

In summary, a single iteration of our retiming algorithm involves computing the maximum flow through the combinational frame, identifying the unique topologically closest minimum cut, and moving the register boundary to the

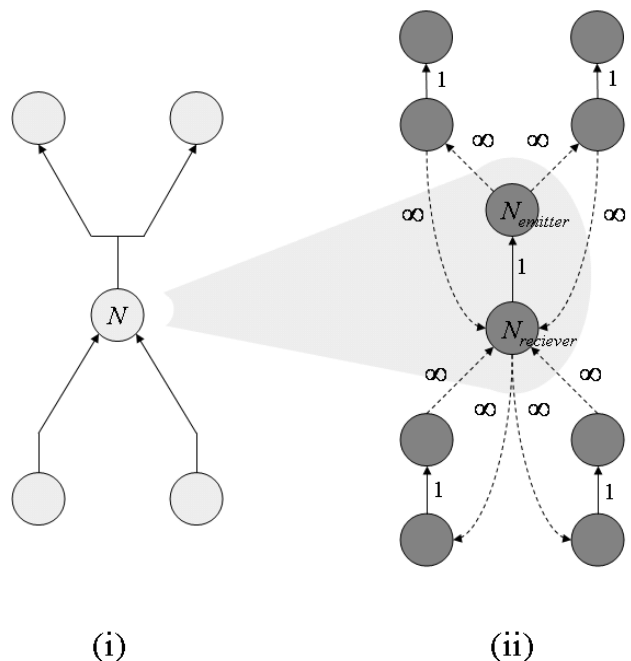


Fig. 2. The original hypergraph circuit around the central node N in (i) is expanded to form a network flow problem (ii) to compute a valid forward retiming with maximal fan-out sharing. Each of the light nodes in (i) is replaced with a pair of dark grey nodes in (ii), and the flow paths as illustrated. The capacities of each path is labeled, with solid edges having unit capacity and dashed edges having infinite capacity.

new location.

B. Primary Inputs and Outputs

The primary inputs and outputs (PIOs) can be treated in different ways, depending on the application. The allowed locations of the minimum cut and the subsequent insertion/removal of registers can be adjusted to either fix or selectively alter the sequential behavior of the circuit.

In synthesis, the relative latencies at all of the PIOs is assumed to be invariant. One way that this can be accomplished is with the addition of a host node. In verification applications, it is not necessary to preserve the synchronization of the inputs and outputs. It may be desirable to borrow or loan registers to the environment individually for each PIO if the result is a net decrease in the total register count.

To allow register borrowing, the external connections should be left dangling. Registers will be donated to the environment if the minimum cut extends past the dangling primary outputs (POs); conversely, registers will be borrowed if the minimum cut appears in the transitive fan-out region of the dangling primary inputs (PIs). The inclusion of this region introduces additional flow paths and introduces additional possibilities for minimizing the total register count.

To disallow desynchronization with the environment, the POs are connected to the sink and the transitive fan-out of the PIs is blocked from participating in the minimum cut. All paths through the combinational network that originate from a PI have a sequential latency that must remain at *zero*. Inserting

a register anywhere in the $\text{TFO}(\{\text{PIs}\})$ will alter this. To exclude this region, one of two methods can be used: (i) temporarily redirecting to the sink all edges $e=(u,v)$ where $v \in \text{TFO}(\{\text{PIs}\})$, or (ii) replacing the constrained flow arc with an unconstrained one, thus preventing that node from participating in the minimum cut. Both methods exclude the invalid portion from participating in the retiming solution.

Because register borrowing requires the initial values of the new registers to be constrained to those reachable in the original circuit, it is necessary to construct additional combinational logic for computing the initial state. If the size of this logic grows undesirably large, register borrowing can be turned off at any point.

C. Multiple Iterations

This section shows how to compute the globally optimum min-register retiming by iteratively applying the maximum-flow algorithm of Section III.B.

Thus far, we have only considered forward retiming of registers in the circuit. It is sufficient to consider only one direction if the circuit is strongly connected (i.e. through the use of a host node) and normalization is applied. However, in general, the optimum minimum-register retiming requires both forward and backward moves. The procedure for a single iteration of backward retiming is essentially identical, except that it computes the maximum flow from the register inputs (sources) to the primary inputs and register outputs (sinks).

The overall algorithm consists of two iterative phases: forward and backward. The procedure is outlined in Algorithm 1. In each phase, the single-frame optimization is repeated until the number of registers reaches a fix-point. In terms of the retiming lag function, each node's lag is either unchanged or changed by one in each iteration.

At no point during retiming is it necessary to unroll the circuit or alter the combinational logic; only the register boundary is moved by extracting registers from their initial position and inserting them in their final position. However, this does change the definition of the combinational frame and the connections to the flow source and sink.

The ordering of the two phases (forward and backward)

ALGORITHM 1: UNCONSTRAINED MIN-REGISTER RETIMING

```

1 while(improvement) { // forward
2   set up forward retiming flow network
3   mark restricted locations
4   compute maximum flow
5   derive nearest minimum cut
6   move registers to cut
7 }
8 while(improvement) { // backward
9   set up backward retiming flow network
10  mark restricted locations
11  compute maximum flow
12  derive nearest minimum cut
13  move registers to cut
14 }
15 compute initial states

```

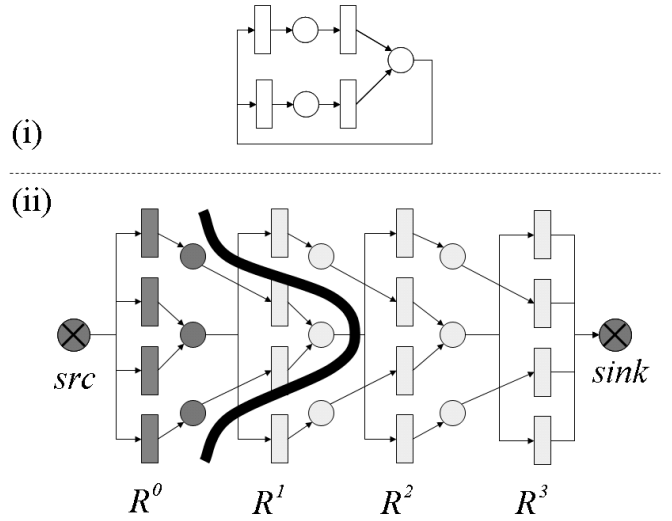


Fig. 3. The circuit in (i) is unrolled three times into (ii). Each copy comprises a combinational frame, the first of which is dark gray, and the register boundaries are labeled with the corresponding frame number. The globally minimum width retiming cut of the registers in the reference frame R^0 is illustrated by the bold line. The minimum width retiming cut stretches across multiple frames and would therefore require multiple iterations of our algorithm.

doesn't affect the number of registers in the result, but we chose to perform forward retiming first because in general min-register retiming is not unique. This approach reduces the amount of logic that has to be retimed backward, thereby reducing the difficulty of computing a new initial state [19].

D. Proof of Correctness

Given a retiming lag function $r(v): V \rightarrow \mathbb{Z}$, consider unrolling the sequential circuit by n cycles, where $n > \max_{\forall v} r(v) - \min_{\forall v} r(v)$. This corresponds to stacking multiple combination frames, as illustrated in Figure 3.

The positions of the registers of the reference cycle after any retiming $r(v)$ can be expressed as a cut C in the edges of this unrolled circuit. The elements of C are the register positions. The unretimed cut, C_{init} , (such that $r(v)=0$) lies at the base of the unrolled circuit. The size of this cut, $|C|$, is the number of registers post-retiming, or equivalently, the number of combinational nodes whose fan-outs hyper-edges cross the cut.

A cut C is a *valid retiming* if every path through the combinational network passes through it exactly once. This implies that for any two registers $R_1, R_2 \in C$, $R_1 \cap \text{TFO}(R_2) = \emptyset$ and vice versa. If this were not the case, additional latency would be introduced and functionality of the circuit would be altered.

Consider an optimal minimum register retiming and its corresponding cut C_{min} . While there exist many such cuts, assume C_{min} to be the one that lies strictly forward of the initial register positions is topologically closest to C_{init} . It can be shown with Lemma 1 that there is one unambiguously closest cut.

Theorem 1: Upon termination of our algorithm, the resulting cut is exactly C_{min} .

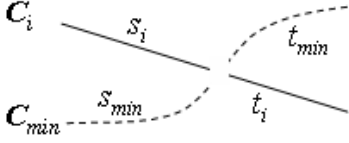


Fig. 4. Definitions of cut partitions for Section III.D.

Proof. Our algorithm iteratively computes the nearest cut of minimum width reachable within one combinational frame and terminates when there is no change in the result. Let the resulting cut after iteration i be C_i . The cut C_i at termination will be identical to C_{min} if the following two conditions are met.

Condition 1. No register in C_i lies topologically forward of any register in C_{min} .

Condition 2. After each iteration, $|C_{i+1}| < |C_i|$ unless $C_i = C_{min}$.

Lemma 1. Let C_i and C_j be two valid retiming cuts, and $\{s_i, t_i\}$ be $\{s_j, t_j\}$ be a partitioning of each such that $s_i \subseteq TFO(s_j)$ and $t_j \subseteq TFO(t_i)$. The cuts $\{s_i, t_j\}$ and $\{s_j, t_i\}$ are also valid retimings.

First, we should point out that this partitioning is valid and that every element must fall into either s or t as defined. Because C_i and C_j are valid retiming cuts, every path must intersect them both. Given the points of intersection $R_i \in C_i$ and $R_j \in C_j$, their membership in either s or t is implied by their topological order.

Now, consider some path p from the source to the sink of the network. Because C_i is a complete cut and a valid retiming, the path must pass through exactly one of either s_i or t_i . Similarly for C_j . If p passes through s_i , it can not pass through t_j , because t_j lies in strictly within $TFO(t_i)$, and p would have had to intersect t_i . Also, if p does not pass through s_i , it can not intersect s_j because s_j lies strictly within $TFO(s_i)$ and so must pass through t_j . Therefore, it must pass through exactly one of s_i and t_j , and $\{s_i, t_j\}$ is a valid retiming cut. Similarly for $\{s_j, t_i\}$.

Proof of Condition 1. Consider a cut C_i that violates Condition 1. Let $\{s_i, t_i\}$ be a partition of C_i and $\{s_{min}, t_{min}\}$ be a partition of C_{min} such that s_i is the subset of registers in C_i that lie topologically forward of the subset s_{min} of the registers in C_{min} . This is illustrated in Figure 4. By Lemma 1, we know that both $\{s_i, t_{min}\}$ and $\{s_{min}, t_i\}$ are valid cuts.

Because a single iteration returns the closest cut of minimum width within a frame, this $C_i = \{s_i, t_i\}$ must be strictly smaller than the closer $\{s_{min}, t_i\}$. This implies that $|s_i| < |s_{min}|$ and that $|\{s_i, t_{min}\}| < |\{s_{min}, t_{min}\}| = C_{min}$. This is impossible by definition. Therefore, Condition 1 must be true.

Observation 1. Retiming by an entire combinational frame does not change any of the register positions in the resulting circuit and also represents a valid retiming cut. Because a register is moved over every combinational node, the retiming lag function is universally incremented. The number of registers on a particular edge is a relative quantity, and the result is structurally identical to the original.

Proof of Condition 2. We can use the minimum cut to generate a cut that is strictly smaller than C_i and reachable within a combinational frame. Consider the cut C_{min} that is generated from C_{min} via Observation 1 such that its deepest point is reachable within the combinational frame of C_i . Some of the retiming lags may be temporarily negative. Let $\{s_i, t_i\}$ be a partition of C_i and $\{s_{min}, t_{min}\}$ be a partition of C_{min} such that s_{min} are the deepest registers in C_{min} that lie topologically forward of the subset s_i of the registers in C_i . $s_{min} \neq \emptyset$ if $C_i \neq C_{min}$. Using the reasoning from condition 1, both $\{s_i, t_{min}\}$ and $\{s_{min}, t_i\}$ are valid cuts.

We know that $|s_{min}| < |s_i|$, otherwise there would be implied the existence of a topologically closer cut $|\{s_i, t_{min}\}| \leq C_{min}$. Therefore, the cut $\{s_{min}, t_i\}$ is strictly smaller than C_i and is reachable within one combinational frame and would be returned by a single iteration of the algorithm. Note that this doesn't imply that there aren't other smaller cuts, only that there must exist at least one that is strictly smaller. Therefore, Condition 2 must also be true. ■

E. Complexity Analysis

As described in Section III.A, the complexity of computing the minimum cut in each iteration of our algorithm is $O(RE)$. The maximum number of iterations can also be bounded by R via Condition 2 in the above proof. The total worst-case runtime is therefore $O(R^2E)$. While this is neither strictly less nor greater than the best known bound for the equivalent minimum-cost network flow problem [9], the results in Section IV indicate that the average runtimes are smaller for the considered circuits.

For the set of benchmarks that we examined, the number of iterations required was small: the average was 2.7 with a maximum of 15. Figure 5 illustrates the fraction of the total register reduction that was contributed by each iteration. Almost all of the reduction in the number of registers occurs after the first iteration in either direction. The number of iterations can be bounded to a small constant to fix the total worst-case runtime to $O(RE)$.

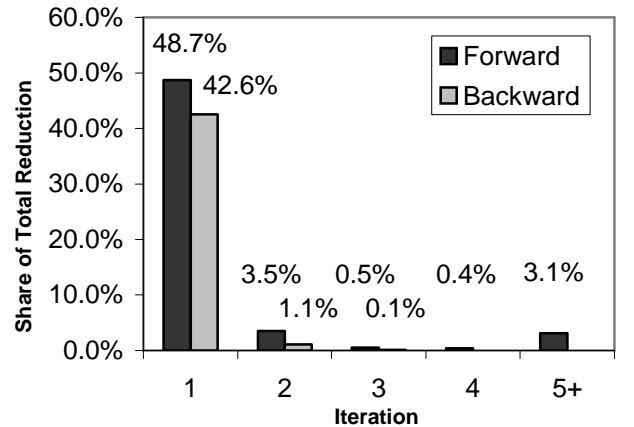


Fig. 5. The contribution of each iteration to the total reduction in the number of registers.

IV. EXPERIMENTAL RESULTS

We applied the proposed algorithm to a suite of gate-level circuits derived from public-domain hardware designs [11]. Altera tools were used to extract and optimize the logic networks. This optimization may have already included reductions in the number of registers. These were then minimally preprocessed by the ABC logic synthesis package [2] as follows: the original hierarchical designs were (a) flattened, (b) structurally hashed and (c) algebraically balanced. From the set of 63 benchmarks, we removed one combinational circuit and 19 circuits whose initial register

count was already minimum, leaving 43 circuits shown in Table 1.

Our algorithm was implemented in C++. The maximum network flow problem was internally solved using the HIPR package available at [10] and described in [4].

Table 1 is divided into three groups of columns, each describing the characteristics of a particular retiming. The first section of Table 1 shows the statistics about the circuit with the registers in their initial positions. The second section describes the results of an incremental heuristic min-delay retiming algorithm [18] implemented in ABC to provide perspective on the area/delay tradeoffs. The third set of

TABLE 1: MINIMUM REGISTER RETIMING RESULTS ON REAL BENCHMARKS

Name	Original Circuit			Min-Delay Retiming			Min-Register Retiming				
	AIG	A	D	A	D	T	F-iter	B-iter	A	D	T
barrel16a	397	37	11	124	4	0.02	1	0	32	11	<0.01
barrel16	357	37	10	85	4	0.01	1	0	32	11	<0.01
barrel32	902	70	12	166	5	0.03	1	0	64	13	<0.01
barrel64	2333	135	14	422	5	0.06	1	0	128	14	<0.01
mux32_16bit	1851	533	9	873	4	0.05	1	1	505	11	0.01
mux64_16bit	3743	1046	13	1460	5	0.12	1	0	991	13	0.01
mux8_128bit	3717	1155	7	2297	3	0.18	1	1	1029	8	<0.01
mux8_64bit	1861	579	7	1145	3	0.07	1	1	517	8	<0.01
nut_000	1262	326	58	393	27	0.05	1	2	312	60	<0.01
nut_001	3179	484	93	558	57	0.08	2	2	435	109	0.03
nut_002	873	212	24	232	10	0.02	2	2	158	25	<0.01
nut_003	1861	265	37	304	24	0.04	3	1	228	46	0.01
nut_004	713	185	13	213	6	0.02	2	2	164	15	<0.01
oc_aes_core_inv	11177	669	25	669	25	0.25	1	1	658	25	0.04
oc_aes_core	8732	402	24	402	24	0.14	1	1	394	24	<0.01
oc_aquarius	23109	1477	207	1575	200	0.81	1	0	1473	206	0.08
oc_ata_ocidec1	1601	269	14	275	11	0.02	1	0	268	14	<0.01
oc_ata_ocidec2	1813	303	14	310	11	0.02	1	1	293	14	<0.01
oc_ata_ocidec3	3957	594	14	599	13	0.06	1	1	562	19	<0.01
oc_ata_vhd_3	3933	594	14	599	13	0.06	1	1	568	14	<0.01
oc_ata_v	838	157	14	169	10	0.02	1	0	156	14	<0.01
oc_cfft_1024x12	9498	1051	61	1672	26	0.91	12	1	704	346	0.70
oc_cordic_p2r	8430	719	55	975	45	0.26	1	0	718	55	0.01
oc_dct_slow	879	178	32	207	14	0.03	0	1	176	32	<0.01
oc_des_perf_opt	21281	1976	15	4656	14	1.27	15	0	1015	233	1.18
oc_fpu	16115	659	2661	1578	543	30.65	2	0	247	2712	0.12
oc_hdlc	2221	426	14	426	13	0.03	1	3	375	17	<0.01
oc_minirisc	1918	289	36	290	33	0.03	2	1	253	39	0.01
oc_oc8051	10315	754	92	757	87	0.19	1	1	743	92	0.01
oc_pci	10426	1354	46	1405	26	0.39	1	1	1308	46	0.02
oc_rtc	1093	114	41	114	29	0.02	1	0	86	41	<0.01
oc_sdrum	860	112	13	109	12	0.02	1	0	109	12	<0.01
oc_simple_fm_rec	2300	226	66	276	40	0.05	0	1	223	75	<0.01
oc_vga_lcd	9086	1108	35	1126	25	0.24	2	1	1078	35	0.02
oc_video_dct	36465	3549	60	8525	16	12.84	1	1	2305	73	0.30
oc_video_huff_dec	1591	61	21	65	18	0.02	0	1	60	22	<0.01
oc_video_huff_enc	1720	59	19	90	13	0.02	1	0	47	32	<0.01
oc_wb_dma	15026	1775	19	1794	17	0.45	1	1	1751	34	0.08
os_blowfish	9806	891	79	906	61	0.30	1	0	827	78	<0.01
os_sdrum16	1156	147	23	162	17	0.02	1	0	144	23	<0.01
radar12	38058	3875	110	3991	56	3.71	2	3	3754	110	0.21
radar20	75149	6001	110	6363	56	6.92	2	1	5364	110	1.34
uoft_raytracer	145960	13079	237	16974	208	23.70	3	2	11610	537	3.76
AVERAGE		1.0	1.0	1.41	0.66				0.89	1.56	

TABLE 2: MIN-COST VERSUS ITERATIVE MAX-FLOW MINIMUM REGISTER RETIMING ON LARGE ARTIFICIAL BENCHMARKS

Name	Original circuit		Minimum-Register Retiming					
	AIG	A	A	Min-Cost Flow T	Iterative Max-Flow		Speedup	
				F-iter	B-Iter	R		
large1	1 006 k	72.9 k	66.9 k	147.9s	3	3	33.0s	4.48
large2	1 005 k	82.7 k	76.9 k	131.3s	3	3	24.5s	5.36
deep3	1 010 k	74.7 k	67.6 k	182.0s	3	21	34.2s	5.32
deep4	1 074 k	86.4 k	82.0 k	130.3s	3	3	17.9s	7.27
larger5	2 003 k	151.1 k	139.5 k	410.6s	3	3	67.2s	6.11
largest6	4 008 k	300.1 k	279.0 k	818.3s	3	3	139.9s	5.85

columns shows the results produced by the proposed minimum-register retiming algorithm.

The following notation is used in the table. Columns labeled “A” refer to the number of registers in the network (area). Columns labeled “D” refer to the number of nodes on the longest combinational path. Columns labeled “T” refer to the cumulative runtime of the flow computations in seconds measured on a 64-bit 2.0Mhz Pentium Xeon. For the minimum register retiming algorithms, the number of forward and backward iterations that are required before the fix-point is reached are also listed (“F-iter” and “B-iter”, respectively).

Because these benchmarks are only of moderate size, a set of larger artificial circuits was created by combining the benchmarks in Table 1. These are described in Table 2. As the number of retiming iterations required appears to be independent of the circuit size—probably because the maximum latency around any loop or from input to output is also size independent—the circuits “large1” and “large2” were constructed via parallel composition to preserve this property. The 2 and 4 million gate circuits, “larger5” and “larger6”, were generated similarly. In contrast, the two circuits “deep3” and “deep4” were built by splitting and serial composition.

In Table 2, the results of our iterative maximum flow-based algorithm are compared against a single minimum-cost flow-based implementation as described by [4]. The latest CS2 package from [10] was used as the solver. In every case, the iterative maximum flow-based implementation required less time to complete.

V. CONCLUSIONS

This paper presented an application of a simplified maximum flow computation to the problem of minimizing the number of registers after retiming. The presented method is very simple, straight-forward to implement, fast, memory efficient, and scalable for large industrial circuits. Potential applications of the method include sequential synthesis and verification.

ACKNOWLEDGEMENTS

This work was supported by SRC contracts 1361.001 and 1444.001, and the California Micro Program with our industrial sponsors Altera, Intel, Magma, and Synplicity.

REFERENCES

- [1] J. Baumgartner and A. Kuehlmann, “Min-area retiming on flexible circuit structures”, *Proc. ICCAD’01*, pp. 176-182.
- [2] Berkeley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and Verification*, Release 61104. <http://www.eecs.berkeley.edu/~alanmi/abc/>
- [3] G. Cabodi, S. Quer and F. Somenzi, “Optimizing sequential verification by retiming transformations,” *Proc. DAC’01*, pp. 601-606.
- [4] B. V. Cherkassky and A. Goldberg, “On Implementing Push-Relabel Method for the Maximum Flow Problem,” *Algorithmica* 19, 1997, pp. 390-410.
- [5] J. Cong and C. Wu, “Optimal FPGA mapping and retiming with efficient initial state computation”, *IEEE Trans. CAD*, vol. 18(11), Nov. 1999, pp. 1595-1607.
- [6] J. Edmonds and R. Karp, “Theoretical improvements in algorithmic efficiency for network flow problems”, *Journal of the ACM*, vol. 19 (2), 1972, pp. 248-264.
- [7] G. Even, I. Y. Spillinger, and L. Stok, “Retiming revisited and reversed”, *IEEE Trans. CAD*, vol. 15(3), March 1996, pp. 348-357.
- [8] J. P. Fishburn, “Clock skew optimization”, *IEEE Trans. Comp.*, vol. 39(7), July 1990, pp. 945-951.
- [9] A. Goldberg, “An efficient implementation of a scaling minimum-cost flow algorithm”, *J. Algorithms* 22, 1997, pp. 1-29.
- [10] A. Goldberg, *Network optimization library*. (Software tools) <http://www.avglab.com/andrew/soft.html>
- [11] M. Hutton and J. Pistorius, *Altera QUIP benchmarks*. <http://www.altera.com/education/univ/research/unv-quip.html>
- [12] A. Kuehlmann and J. Baumgartner, “Transformation-based verification using generalized retiming”, *Proc. CAV’01*.
- [13] C. E. Leiserson and J. B. Saxe. “Retiming synchronous circuitry”, *Algorithmica*, 1991, vol. 6, pp. 5-35.
- [14] N. Maheshwari and S. Sapatnekar, “Efficient retiming of large circuits”, *IEEE Trans VLSI*, 6(1), March 1998, pp. 74-83.
- [15] P. Pan, “Continuous retiming: Algorithms and applications”. *Proc. ICCD ’97*, pp. 116-121.
- [16] S. S. Sapatnekar and R. B. Deokar, “Utilizing the retiming-skew equivalence in a practical algorithms for retiming large circuits”, *IEEE Trans. CAD*, vol. 15(10), Oct.1996, pp. 1237-1248.
- [17] N. Shenoy and R. Rudell, “Efficient implementation of retiming”, *Proc. ICCAD ’94*, pp. 226-233.
- [18] D.R. Singh, V. Manohararajah, and S.D. Brown, “Incremental retiming for FPGA physical synthesis”, *Proc. DAC ’05*, pp. 433-438.
- [19] H. J. Touati and R. K. Brayton, “Computing the initial states of retimed circuits”, *IEEE Trans. CAD*, vol. 12(1), Jan 1993, pp. 157-162.