Scalable Min-Register Retiming Under Timing and Initializability Constraints

Aaron P. Hurst, Alan Mishchenko, and Robert K. Brayton
University of California, Berkeley
Berkeley, CA

ABSTRACT

We demonstrate that a maximum-flow-based approach to register-minimization is a useful platform for incorporating varied design constraints. In this work, we extend the flow-based formulation to include timing constraints and to guarantee the existence of an equivalent initial state. Reducing the register count is motivated by positive consequences for physical design, verification, and power consumption, but it is critically necessary for synthesis that these timing and functionality requirements are also met. Our solution is optimum in the number of registers under either or both constraints and also possesses several other distinct advantages: the runtime is significantly faster than comparable techniques, the algorithm is capable of early termination with a timing-feasible solution, and both maximum and minimum path constraints can be specified.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids

General Terms

Algorithms, Design.

Keywords

Retiming, Sequential Optimization, Initial State, Min-Area.

I. Introduction

Retiming [8] moves registers over combinational nodes in a logic network, preserving output functionality and logic structure. This can be applied to many ends, including period minimization, register reduction, sequential verification, and improved physical design.

Minimizing the number of registers is a particularly useful application. The area and power consumption of the logic network are reduced, but the real benefits are realized in domains that are outside the reach of other tradeoffs for area and/or power. Because each register is an additional point to which the clock must be distributed, the difficultly of the clock network design is eased, and the dynamic power required to distribute the clock is reduced. The number of state elements is also a primary determinant of the complexity of sequential verification problem, and register reduction has been used with positive results in this domain.

However, some potential consequences of register minimization must be addressed for synthesis flows: 1) timing of the design is affected, and bounds must be placed on the lengths of the longest and shortest combinational paths to ensure that register setup and hold times are met, and 2) initialization of the sequential machine is affected, and if registers are retimed backward, the existence of an equivalent initial state is jeopardized without additional initialization logic.

We introduce a retiming algorithm that performs register minimization under timing and/or initializability constraints. The formulation is based upon the flow-based register minimization introduced in [6] and uses a maximum-flow solver as the computational core of the system. These additional design constraints can be imposed with simple modifications to the corresponding flow problem. Even with the additional constraints, the result is still minimal in the number of registers.

Timing-constrained minimum-register retiming was first described in the original work on retiming by [8]. The formulation does not scale well to larger designs, as all pair-wise delay constraints must be enumerated and incorporated into the problem. The scalability was improved with the introduction of the Minaret algorithm [9]; Minaret uses retiming-skew equivalence to eliminate the need to consider infeasible timing constraints, thus reducing the problem size. Our algorithm further restricts the pair-wise constraints by area criticality, providing another reduction in the problem size and improving the ability to scale to larger problems. Furthermore, it has the important property that every intermediate solution is timing-feasible: the runtime can be bounded with only a loss in the minimization potential. Finally, we also describe a means to consider constraints on both long and short paths.

Guaranteeing initializability has also been addressed, though the solutions remain unsatisfactory. If no equivalent initial state is found to exist, either the retiming must be restricted to the forward direction to ensure feasibility [8] or

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DAC 2008, June 8–13, 2008, Anaheim, California, USA
Copyright 2008 ACM 978-1-60558-115-6/08/0006…5.00

Algorithm 1: Unconstrained Min-Register Retiming

```
1:  /* forward retiming phase */
2:  loop {
3:      compute fwd min cut $R$
4:      move registers to $R$
5:  } until no decrease in number of registers
6:  /* backward retiming phase */
7:  loop {
8:      compute bwd min cut $R$
9:      move registers to $R$
10: } until no decrease in number of registers
```

additional combinational logic synthesized [17]. The solution proposed by [10] does not scale beyond small designs. Minimally transforming a useful retiming without an equivalent initial state into one that has an equivalent initial state is a difficult problem. We demonstrate how to do this with a minimal increase in the number of additional registers.

In summary, the contribution of this work is a unified problem structure for efficiently minimizing the number of registers under two critical types of design constraints. We demonstrate that our solution is more scalable than prior approaches, either when optimality is necessary or may be relaxed.

The paper is organized as follows. Section II describes the background information and Section III the foundational flow-based register minimization algorithm. Section IV demonstrates how to incorporate timing constraints, and Section V how to guarantee initial state feasibility. While presented separately, both constraints can be mixed within the same retiming flow. Experimental results are provided in Section VI.

II. Background

A circuit is modeled as a directed graph $\mathcal{G}$=<V,E> whose vertices V correspond to logic cells and directed edges E correspond to wires connecting the gates, decomposed into pair-wise connections from gate outputs to inputs. The vertices may be either combinational or sequential elements. The circuit's external connections are represented by additional vertices called primary inputs (PIs) and primary outputs (POs).

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 combinational nodes of the network reachable through the fan-out edges from v, captured by the function TFO(v): $V \rightarrow 2^V$. The sequential fan-out $\mathrm{TFO}_k(v)$ is the subset of combinational nodes that are reachable from v through exactly k registers.

All combinational elements n are assumed to have non-negative delays d_n. Let $d(u \rightarrow v)$ be the longest combinational delay along any path between u and v, and $d_k(u \rightarrow v)$ be the longest combinational delay that passes through exactly k registers.

A *combinational frame* of the circuit is comprised of the acyclic combinational network between the register outputs / PIs and register inputs / POs.

A. Retiming

Retiming [8] relocates the registers to optimize some circuit characteristic while preserving output functionality and optionally meeting some additional constraints. Any valid retiming is captured by a *retiming lag function* $r(v){:}V \longrightarrow \mathbb{Z}$ that describes the number of registers moved backward over each combinational node. At a minimum, a retiming lag function must satisfy Equation 1 to be physically implementable. $w_i(e)$ is the initial number of registers present on edge e.

$$r(u) - r(v) \leq w_i(e) \qquad \forall e = (u,v) \tag{1}$$

The body of known retiming algorithms has greatly proliferated and improved over the years since the problem was first described in [8]. A complete overview of these developments is beyond the scope of this paper; however, the available alternatives to solve the problems discussed in this paper are described in more detail in Sections III-V.

III. Min-Reg Retiming Algorithm

The flow-based register minimization method of Algorithm 1 (introduced in [6] and hereafter referred to as the *unconstrained minimum-register* retiming) forms the basis of both our timing- and initial-state-constrained retiming algorithms. This technique involves iterating a maximum-flow problem on a modified version of the circuit until a fix-point is reached. This iteration is done in two distinct phases: forward and backward, and it is proved in [6] that this results in the optimal minimum-register retiming. Using the determination of maximum-flow as the inner core of a register-minimizing retiming algorithm appears to be more computationally efficient than other approaches.

We highlight one important property utilized in the unconstrained register minimization algorithm, described by Lemma 1. This is used to enforce the sequential latency along all paths but will also be useful for additional constraints.

Lemma 1. *If there exists an unconstrained flow edge e=$u \rightarrow v$, a finite minimum cut will never lie between u and v.*

Proof. Consider the counterexample. The width of the minimum cut is exactly the sum of the capacity of the edges that cross it. If u lies before the cut but not v, e must cross the cut. Because the capacity of e is infinite, the cut can not be finite, thus violating the assumptions. ■

IV. Timing Constraints

In most synthesis applications (as opposed to the verification applications), it is necessary to introduce constraints on the minimum and maximum combinational path delays. This problem is known as *timing-constrained minimum-register* retiming. Its computational difficulty exceeds that of both the minimum-register and minimum-delay problems.

All known timing-constrained min-register algorithms utilize a version of the linear program first described in [8]. The fundamental bottleneck of this approach lies in the enumeration and incorporation of all pair-wise delay constraints. In the original algorithm, all connected pairs were examined, resulting in an O(v^3) procedure. The Minaret algorithm [9] provided a leap forward by using a retiming-skew equivalence to bind the constraints to those that are potentially *timing-critical*.

Our algorithm provides a leap forward over Minaret by further restricting the enumeration of the constraints to the subset of the circuit that is also *area-critical.* The number of paths that must be examined is much smaller. The power in this approach lies in the observation that the regions that are area- and timing-critical are very different; the intersection can be quite small in even a large circuit.

An additional advantage—critical for industrial scalability—is that the optimum solution is approached via a set of intermediate solutions, which are monotonically improving and always timing-feasible. Thus, the algorithm can be terminated at any point with an improved timing feasible solution. Finally, short-path timing constraints are handled also.

Consider the presence of a register on the output of some node v. Let $T_v^{\max}$ be the maximum allowable arrival time and $T_v^{\min}$ be the minimum allowable arrival time. Typically, these values would include local variations such as the estimated local clock skew (δ_v), the timing parameters of the register cell appropriate to drive the capacitive load (e.g. setup S_v and hold H_v), and the maximum period of the local clock domain (T_{clk}). Any physical information about the location of v will improve the precision of these values. Equations 2 and 3 suggest definitions in terms of these parameters.

$$T_v^{\max} = T_{clk} - S_v - \delta_v \qquad (2)$$

$$T_v^{\min} = H_v - \delta_{clk} \qquad (3)$$

We require that the initial positions of the registers meet these maximum and minimum arrival constraints. If it is desired that a higher frequency be achieved through retiming, the design would need to be retimed first by one of the many delay-minimizing retiming algorithms [8][13][11][15], among which efficient exact and heuristic solutions are available.

A. Conservative vs. Exact Timing Constraints

Consider retiming one or more registers in either direction within one combinational frame of the circuit to some vertex v. Let R_v be the potential new register on v's output. There are four timing constraints that are affected by this move: the latest and earliest arrival times on the timing paths that start and end at the retimed register. At the start or end of the path, two constraints are made potentially critical; the other two can be ignored. Observe that the degree of criticality of these constraints is strictly increasing with the distance that the registers move.

We introduce two versions of each of these constraints: *conservative* and *exact.*

In the conservative version, it is assumed that the end of the timing path opposite the moving register remains fixed. This is an over-constraint: the other register may have moved also in the same direction (and *only* the same direction within each forward or backward phase), thereby relaxing the timing criticality. The set of conservative constraints C_{cons} defines the vertices past which a register can not be retimed without potentially violating the timing.

The set of conservative constraints can be computed in O(E) time with a static timing analysis of the original circuit. The short-path constraints can be identified in one pass. The long-path constraints require two passes (to capture the components of the path on either side of the original register); register output arrivals are seeded with their input arrivals from the first pass and then those values are propagated forward.

A conservative timing constraint can be enforced in the flow graph by simply removing the constrained node from the graph (or, equivalently, redirecting its fan-ins/outs to the flow sink). After removal, these timing-constrained nodes will not participate in the resulting minimum cut.

For exact constraints, the other end of a timing path is not assumed to have remained stationary. Each exact constraint therefore encodes the position to which the other end of a timing path would have to move for register R_v to remain timing feasible. The set of exact constraints C_{exact} defines the node pairs (source and sink of a path) that describe these dependencies. These can be computed easily: the exact constraints at v are the roots U of the transitive fan-in/out cones whose depth is $T_v^{\min} / T_v^{\max}$ and cross exactly one register. There may be many several such pairs for each v.

Enforcement of the exact timing constraints is accomplished by introducing additional unconstrained flow edges into the graph. An edge $v \rightarrow u$ is added for every exact constraint, where v is the potential new register position and u is the point to which the register boundary must also move. By Lemma 1, these unconstrained flow edges will prohibit the resulting cut from violating this particular constraint; the cut will be the optimally minimum one that meets the exact constraints.

Because the depth of the cone is at least the current period, each timing arc will terminate at a node that is never

ALGORITHM 2: TIMING-CONSTRAINED MIN-REGISTER RETIMING

```
1:  /* forward retiming phase */
2:  loop {
3:      let $C_{cons}$ be $\{n : \exists m \text{ s.t. } d_1(m \to n) \geq T_n^{\max}\}$
4:      let $C_{exact}$ be an empty list of $V \times V$
5:      loop {
6:          compute fwd min cut $R_{under}$ under constraints $C_{exact}$
7:          compute fwd min cut $R_{over}$ under constr. $C_{exact}$, $C_{cons}$
8:          for all $n$ in $\text{TFO}(R_{over}) \cap \text{TFI}(R_{under}) \cap C_{cons}$
9:              $P_{long} \leftarrow \{m : T_n^{\max} - d_m < d_1(m \to n) \leq T_n^{\max}\}$
10:             $P_{short} \leftarrow \{m : T_m^{\min} - d_m \leq d_1(n \to m) < T_m^{\min}\}$
11:             $C_{cons} \leftarrow C_{cons} - n$
12:             $C_{exact} \leftarrow C_{exact} \cup (n \times P_{long}) \cup (n \times P_{short})$
13:     } until $R_{under} = R_{over}$
14:     move registers to $R_{under} = R_{over}$
15: } until no decrease in number of registers
16  /* backward retiming phase (omitted) */
```

topologically deeper than its source. The arcs will therefore always be in the direction from sink to source, and the flow from source to sink will remain finite. This motivates the requirement that the circuit initially meets the timing constraints.

Paths through these arcs indicate timing dependencies that stretch across multiple cycles. Note that there may even be cycles within the set of constraints; this occurs whenever a critical sequential cycle is present in the netlist. A correct result is such that moves of registers within a critical cycle are synchronized.

B. Iterative Refinement

The overall algorithm (Algorithm 2) consists of an iterative refinement on the conservatism of the timing constraints until the optimal solution has been reached for a combinational frame. The refinement need only be performed for regions whose timing conservative is preventing further area improvement, i.e. *area-critical*.

During an iteration, each node will be in one of three states: (i) *none*, if retiming a register past this node will not introduce a timing violation, (ii) *conservative*, or (iii) *exact*.

Refinement is accomplished as follows. We compute two minimum cuts: R_{under}, the minimum cut under the (current) exact constraints, and R_{over}, the minimum cut under both the exact and conservative constraints. Note that R_{under} is under-constrained and R_{over} is over-constrained. Therefore, R_{under} will be at least as deep as R_{over}, whose timing constraints prevent deeper register moves. The vertices whose timing are to be refined are those that are conservatively constrained and lie (topologically) between the two cuts. The exact constraints are computed for each of the tightened vertices, inserted into C_{exact}, and the vertex is removed from C_{cons}. The refinement terminates when $R_{under} = R_{over}$.

V. INITIAL IZABILITY

To be functionally correct, a retiming must have a set of equivalent initial states that reproduce the initial behavior of the original circuit. There are several ways to address this requirement: restricting the retiming to the forward direction only [8], introducing additional combinational initialization logic [17], or computing an equivalent initial state after backward retiming—if one exists [16]. The latter is preferable, allowing the full optimization power of retiming to be exploited with minimal perturbation to the combinational logic.

However, a problem arises when a retiming has no equivalent initial state. In this case, the problem must be transformed into one that has an equivalent initial state, even if some optimality is lost. This is the problem of ensuring *initializability*. For minimum-delay retiming, [12] describes a method that finds a feasible solution by restricting the target period; there is no equivalent target register count in the min-register problem. In [16] a heuristic transformation is introduced for altering any minimum-delay retiming. For the minimum-register problem, [10] describes a solution, but it requires the formulation of retiming as a mixed-integer linear program (MILP), making it only useful for small circuits.

We propose a technique that is both optimal in its result and empirically scalable in its runtime. It should be noted that the notion of optimality is relative to the point of observation where the initialization behavior of the two circuits is constrained to be identical. The most general location is at the outputs, where the maximal number observability don't cares (ODCs) will be introduced; however, this may lead to a very deep sequential equivalence checking problem during the initial state computation. We require identical behaviors as observed at the locations/states of the registers after the forward min-register retiming phase has been completed.

Once a potential backward retiming cut R has been generated for the circuit, we attempt to find an equivalent initial state. This problem can be constructed and solved as an instance of SAT. As the registers are pushed backward through the circuit, a variable is introduced for every nodes over which a register is retimed. Note that there may be multiple variables for each physical node; they can be differentiated by their lags at the point of retiming. The SAT

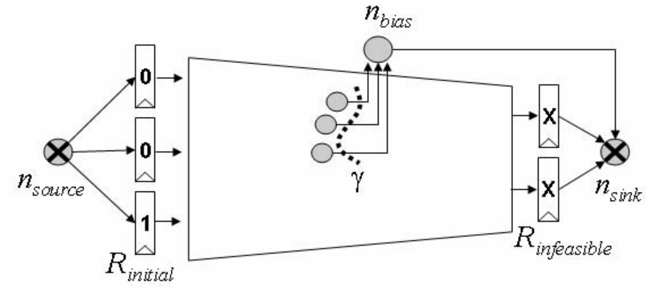

Fig. 1. The bias structure for the feasibility constraint γ. On the left of the unrolled circuit are the initial positions of the registers; the uninitializable positions are on the right. The width of all cuts deeper than γ are penalized by one with the addition of node n_{bias}.

problem is also incrementally constructed by adding constraints imposed on these variables by net connections, gate functionality, and the original initial states. If the solver is able to find an equivalent initial state, no further effort is necessary.

If there exists no equivalent initial state, we attempt to isolate the source of the conflict. Let a *feasibility constraint* γ be a subset of the problem variables (and a partial cut in the unrolled netlist) that has the following property: it is *sufficient* for a retiming to be as deep as γ to be infeasible. Correspondingly, a retiming must be at least partially shallower than γ to be feasible. Lemma 2 implies that infeasibility is monotonically increasing with topological order, and that such a partial cut γ must exist.

Lemma 2. *If a particular retiming is initial state infeasible, all strictly deeper retimings are also infeasible.*

Proof. Consider a feasible assignment at some strictly deeper cut. The forward propagation of these initial states implies a set of initial values at the location of shallower cut. This set of values comprises a feasible initial state for a retiming at the shallower cut and violates the assumption. ■

Algorithm 3: Initializable Retiming

```
/* forward retiming phase (omitted) */
/* backward retiming phase */
define init state problem variables as $V \times \mathbf{Z}$ (node × lag)
let $C_{feas}$ be an empty list of $2^V$
loop {
    save current register positions as $R'$
    loop {
        compute bwd min cut $R$ under constraints $C_{feas}$
        move registers to $R$
        update initial state problem
    } until  no decrease in number of registers
    if SAT($R$ ) terminate

    let topo be an topological ordering of $V$
    let  $\gamma$ be an empty $2^V$
    loop {
        binary search on $v$ until
            $\neg\text{SAT}(R)$ w/o variables $\text{TFO}(\gamma) \cup \{u : \text{topo}(u) \geq \text{topo}(v)\}$
            $\wedge\text{SAT}(R)$ w/o variables $\text{TFO}(\gamma) \cup \{u : \text{topo}(u) > \text{topo}(v)\}$
        push $v \rightarrow \gamma$
    } until SAT($R$ ) w/o variables TFO($\gamma$)
    $C_{feas} \leftarrow C_{feas} + \gamma$
    move registers to $R'$
} forever
```

We find a reasonably minimum partial cut γ as follows. The circuit is ordered topologically, and binary search is used to find the shallowest complete cut, which results in an UNSAT initial state. The last variable that was required to produce UNSAT is then added to γ, and the procedure is repeated until γ by itself is sufficient to imply UNSAT. The final γ is a feasibility constraint. Alternatively, the UNSAT core (if available) can be used to isolate the source of the conflict and greatly speed up the search.

Each new feasibility constraint is then added to the cumulative set C_{feas}. To implement each constraint γ, a *penalty structure* is added to the flow graph to bias it against any cuts that lie deeper than the corresponding partial cut. This is accomplished using the graph feature illustrated in Figure 1. Note that the net effect is that the minimum width of any cut that lies beyond γ is increased by one, thereby penalizing infeasible retiming solutions, i.e., each feasibility constraint introduces exactly one register.

As the register count is increased, one of two cases will occur: (i) the minimum cut is now shallower than γ and the result is initializable, (ii) the minimum cut is still as deep as γ and another penalty is necessary. In this manner, the register count is incremented until it first becomes possible to find an equivalent initial state. If multiple penalties with multiple overlapping elements are generated, search may be required to guarantee optimality; this was not the case in our examples, and we do not discuss it here.

Because the problem of computing an equivalent initial state after backward retiming—let alone transforming that retiming—is already NP-hard, is not possible to establish a polynomial upper bound on the runtime of this algorithm. However, this in no way precludes its speed and scalability on the class of circuits typically seen in the real world. Our experience has shown that, for the circuits that we examined, the check for an equivalent initial state via SAT is extremely fast. The total number of calls to the SAT solver is bounded by O(FR log R), where R is the original number of registers in the design and F is the number of additional registers that are required to ensure initial state feasibility. F is quite small in all of the examined circuits.

VI. Experimental results

We applied the proposed algorithm to a suite of gate-level circuits derived from public-domain hardware designs [1]. Altera tools were used to extract and optimize the logic networks, possibly including sequential optimization. These were then preprocessed by the ABC logic synthesis package [1] as follows: the designs were (a) flattened, (b) structurally hashed and (c) algebraically balanced. All experiments were conducted on a 3.0Ghz x64 machine.

Table 1 compares the performance of our timing-constrained algorithm against Minaret [9]. The testcases presented are the ones with over 1000 registers that were processed by Minaret without error. The maximum delay constraint for every net was set to the initial circuit delay, and minimum delay constraints were set to negative infinity (because these are not supported by Minaret). The runtimes of both Minaret and our flow-based method are listed. The average runtime of Minaret is 102x that of our tool.

Table 1 also lists the percentage of nodes in each circuit that were initially subject to a conservative delay constraint (in column *%Cons*). The next columns (*#Refined* and *#Exact*) list the number of nodes that were refined and the number of exact constraints that resulted. For each of these metrics, the worst-case values across all forward/backward

TABLE 1: TIMING-CONSTRAINED MIN-REG RETIMING RESULTS

	Original Circuit			Flow-Based					Minaret
Name	**Nodes**	**Regs**	**Delay**	**Regs**	**% Cons**	**# Refined**	**# Exact**	**Runtime**	**Runtime**
s38417	19.5k	1465	54.0	1288	15.0%	4	5	2.08s	17.8s
b17_opt	49.3k	1414	44.0	1413	71.5%	0	0	6.9s	227.6s
mux8_128bi	7.8k	1155	14.0	1149	27.1%	0	0	0.07s	0.5s
oc_cfft	19.5k	1051	111.0	874	23.3%	1654	6998	12.1s	769.s
oc_des_perf	41.3k	1976	31.0	1920	89.3%	27225	602773	10.2s	114.6s
oc_pci	19.6k	1354	88.0	1311	1.9%	4	8	0.10s	33.8s
oc_wb_dma	29.2k	1775	36.0	1754	28.6%	2	4	0.24s	24.6s
oc_vga	17.1k	1108	123.0	1079	25.6%	1	6	0.10s	30.6s
MEAN								**1x**	**102x**

retiming iterations are presented.

We implemented a unit timing model for comparison with Minaret, but the algorithm can be used with one that is much more descriptive. A second implementation used a standard load- and slew-dependent interpolating table lookup to compute path delays. Because computation of timing data dominates the runtime, this extra effort increased the runtime to 5x that of the unit delay version.

Table 2 describes the application of initial-state-feasible retiming on some of the benchmarks. Register minimization preserves the initial state in the overwhelming majority of cases; only one design in the entire suite ("s400") did not have an equivalent initial state. For the other benchmarks, all initial states in the design were set to random values to create conflicts. The number of registers in the infeasible retiming is listed in the column *Infeas. Regs*. Initial-state-feasible retiming was also applied, and the number of additional registers (or, equivalently, the number of iterations) is listed in the column *Feasible Regs*. Column *Avg.* $|\gamma|$ is the average number of nodes in each of the feasibility constraints.

The randomization of the initial states likely results in more difficult problems than would be generated in any actual design, and yet the optimal feasible retiming can be found in a median runtime of a little over a second. The circuit "radar12" is the outlier and presents a challenge due to its particular arithmetic structure.

ACKNOWLEDGEMENTS

This work was supported by SRC contracts 1361.001 and 1444.001, NSF grant CCF-0702668 "Sequentially Transparent Synthesis", and the California Micro Program with our industrial sponsors Actel, Altera, Calypto, Intel, Magma, Synopsys, Synplicity, and Xilinx.

TABLE 2: INITIALIZABLE MIN-REG RETIMING RESULTS

	Orig		Infeas	Feasible		
Name	**Gates**	**Regs**	**Regs**	**Reg**	**Avg. $\|\gamma\|$**	**Time**
s400	0.3k	21	18	+1	8.0	0.08s
oc_aes_core	16.6k	402	395	+3	2.0	2.55s
oc_vga_lcd	17.1k	1108	1087	+1	1.0	1.09s
nut_003	6.6k	484	450	+3	1.0	1.41s
radar12	71.1k	3875	3771	+27	2.3	108.3s
oc_wb_dma	29.2k	1775	1757	+2	3.5	5.70s
oc_minirisc	3.9k	289	271	+2	1.0	0.49s

REFERENCES

[1] Berkeley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and Verification*, Release 61104. http://www.eecs.berkeley.edu/~alanmi/abc/

[2] G. Cabodi, S. Quer and F. Somenzi, "Optimizing sequential verification by retiming transformations," *Proc. DAC'01*, pp. 601-606.

[3] 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.

[4] G. Even, I. Y. Spillinger, and L. Stok, "Retiming revisited and reversed", *IEEE Trans. CAD*, vol. 15(3), March 1996, pp. 348-357.

[5] A. Goldberg, *Network optimization library.* (Software tools) http://www.avglab.com/andrew/soft.html

[6] A. Hurst, A. Mishchenko, and R. Brayton, "Fast min-register retiming via binary max-flow", *Proc. of FMCAD*, 2007.

[7] M. Hutton and J. Pistorius, *Altera QUIP benchmarks.* http://www.altera.com/education/univ/research/unv-quip.html

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

[9] N. Maheshwari and S. Sapatnekar, "Efficient retiming of large circuits", *IEEE Trans VLSI*, 6(1), March 1998, pp. 74-83.

[10] N. Maheshwari and S. Sapatnekar, "Minimum area retiming with equivalent initial states", *Proc. ICCAD '97.*

[11] P. Pan, "Continuous retiming: Algorithms and applications". *Proc. ICCD '97*, pp. 116-121.

[12] P. Pan and G. Chen, "Optimal retiming for initial state computation", *Proc. Conf. on VLSI Design,* 1999.

[13] 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.

[14] N. Shenoy and R. Rudell, "Efficient implementation of retiming", *Proc. ICCAD '94*, pp. 226-233.

[15] D.R. Singh, V. Manohararajah, and S.D. Brown, "Incremental retiming for FPGA physical synthesis", *Proc. DAC '05*, pp. 433-438.

[16] L. Stok, I. Spillinger, and G. Even, "Improving initialization through reversed retiming", *Proc. of European Conf. on Design and Test*, 1995.

[17] 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.