Optimization of Critical Paths in Circuits with Level-Sensitive Latches

Timothy M. Burks[1] and Karem A. Sakallah[2]

[1]Systems Technology and Architecture Division, IBM Corporation, Austin, TX
[2]Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI

Abstract

A simple extension of the critical path method is presented which allows more accurate optimization of circuits with level-sensitive latches. The extended formulation provides a sufficient set of constraints to ensure that, when all slacks are non-negative, the corresponding circuit will be free of late signal timing problems. Cycle stealing is directly permitted by the formulation. However, moderate restrictions may be necessary to ensure that the timing constraint graph is acyclic. Forcing the constraint graph to be acyclic allows a broad range of existing optimization algorithms to be easily extended to better optimize circuits with level-sensitive latches. We describe the extension of two such algorithms, both of which attempt to solve the problem of selecting parts from a library to minimize area subject to a cycle time constraint.

1 The critical path method and timing-driven design

When a circuit must be designed to satisfy stringent timing constraints, we say that the design is *timing-driven*. Researchers have described a wide variety of timing-driven design problems: logic synthesis, retiming, transistor sizing, part selection, input ordering, and placement and routing. Despite the range and variety of these problems, each approach is derived from a common framework for representing and enforcing timing constraints: the Critical Path Method (CPM) [1].

The application of CPM and a related technique called PERT to digital circuits was first described by Kirkpatrick and Clark [2] and later by Hitchcock, Smith, and Cheng [3]. In this paper, circuits are represented by a graph in which directed arcs represent delays and nodes represent electrically equipotential regions. Two special nodes, the *source* and *sink*, group circuit inputs and outputs, respectively. The circuit computation time is obtained by making a single pass through the graph. Beginning with the source node and proceeding in topological order, an *event time* for each node is calculated:

$$e(v) = \max_{u \in P(v)} [e(u) + t_{u,v}] \quad (1)$$

$e(u)$ and $e(v)$ are the event times for nodes u and v, $t_{u,v}$ is the delay of arc $u \rightarrow v$, and $P(v)$ is the set of node predecessors of node v. When the event time for the source node is zero, the event time of the sink node gives the delay of the circuit.

A *critical path* is a sequence of arcs that connects the source and sink nodes and whose delays determine the circuit completion time. Critical paths can be identified by computing *required times* in a single pass which visits nodes in reverse topological order.

$$r(v) = \min_{u \in S(v)} [r(u) - t_{u,v}] \quad (2)$$

$r(u)$ and $r(v)$ are the required times for nodes u and v and $S(v)$ is the set of successors of node v. The required time for the sink node is set to the time that the circuit calculation must complete. The *slack* of a node is defined as:

$$s(v) = r(v) - e(v) \quad (3)$$

A similar quantity, *float*, is defined for each arc:

$$f(u \rightarrow v) = r(v) - e(u) - t_{u,v} \quad (4)$$

A critical path can be identified as a sequence of arcs having the most negative float in the graph.

Originally, critical path methods were applied only to combinational circuits. Synchronous sequential circuits were analyzed by first partitioning them into combinational sections whose inputs were driven from edge-triggered flip-flops or primary inputs, and with outputs connected to primary outputs or edge-triggered flip-flops. When level-sensitive latches were used, it was necessary to assume fixed signal departure times, effectively treating latches as edge-triggered devices. The purpose of this paper is to relax these assumptions as much as possible.

2 CPM-based algorithms for timing-driven part selection

The timing-driven part selection problem can be stated as follows: given a netlist of parts and a library containing a discrete set of implementations for each part, where each implementation has different drive capability, input load, and cost (*e.g.* area or power), select an implementation for each part to minimize the total cost subject to a fixed constraint on circuit timing, typically a minimum cycle time.

A commonly-used delay model gives the delay Δ_i through a part P_i as

$$\Delta_i = \tau_i + R_i C_{Li} \quad (5)$$

τ_i is the intrinsic delay of P_i, R_i is an effective output resistance, and C_{Li} is the capacitance seen by the part output. Typically, τ_i and R_i decrease as the size of P_i is increased and C_{Li} increases as the sizes of the fanouts of P_i increase. This nonlinearity complicates the optimization problem, since we cannot guarantee that the fastest circuit will be the one composed of the largest parts.

We examine two CPM-based part selection algorithms, each of which is easily extended to the optimization of circuits with level-sensitive latches. The first is an adaptation of the TILOS algorithm [5] for transistor sizing. An iterative procedure, TILOS first identifies the critical path or paths in a circuit and then selects one transistor from the path(s) to be resized. The transistor chosen is the one with the largest *sensitivity* value, which is defined as the amount of delay reduction per incremental increase in area. Although later work showed that the TILOS algorithm may not always produce optimal sizings [6], it is generally seen to produce good results with moderate running times. TILOS was originally developed to size individual transistors and allowed for a nearly-continuous range of sizes. However, a similar approach can be used for the part selection problem, and Lin et. al. developed a comparable procedure which also used sensitivity information to guide sizing [7]. In the version of the algorithm that we use, each pass computes actual and required times for each node. The arcs sharing the smallest float are examined, and from these, the gate with the largest sensitivity is resized. Each iteration runs in time linearly related to the number of parts. The number of iterations varies with the number of parts that must be resized to satisfy the timing constraints.

The second algorithm was based on an algorithm for optimally sizing a chain of parts developed by Hinsberger and Kolla, who also developed an algorithm for fanout-free trees [8] and showed that the general problem of timing driven part selection is NP-complete [8, 9]. To optimize arbitrary circuits, Hinsberger and Kolla proposed using their sizing algorithms to iteratively resize the most critical path or tree in a circuit. Iteration stops when the target cycle time is obtained or when it is impossible to reduce the delay of the critical path. Experiments in [9] suggested that iterations of the path-based approach provided solutions of comparable quality in significantly less time than iteratively resizing trees. As a result, we use the simpler algorithm which iteratively resizes chains of parts.

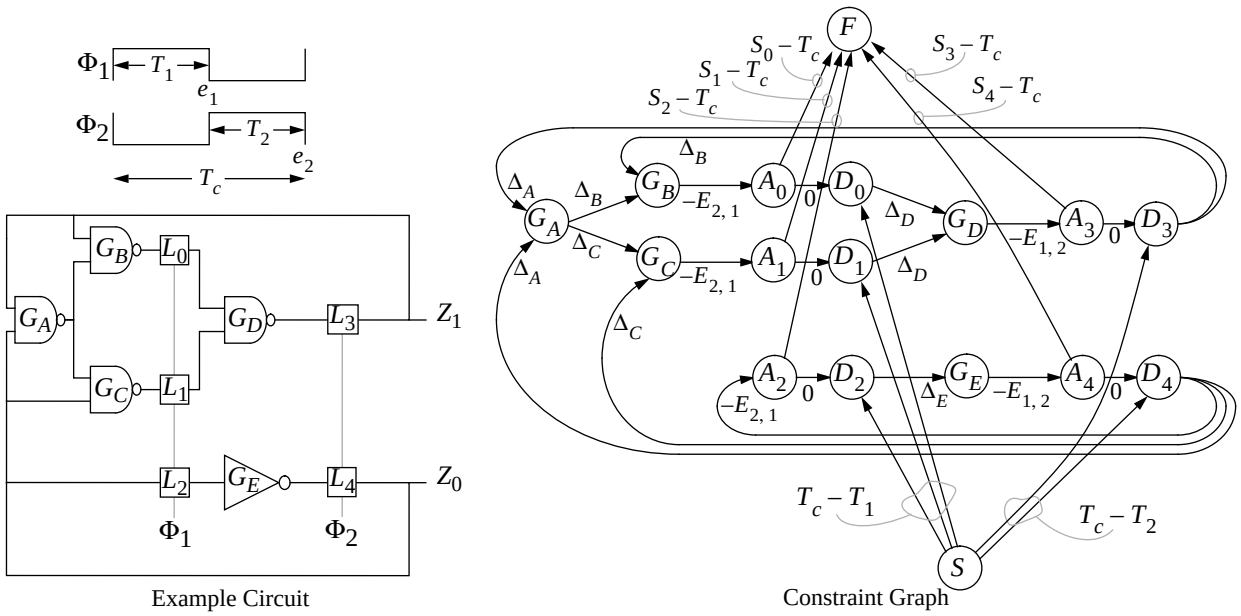
3 Extending CPM for circuits with level-sensitive latches

The approaches of the previous section were originally designed to work on combinational logic only[1], but they can be easily extended to optimize across level-sensitive latches, allowing latch arrival and departure times to move freely during the optimization. We distinguish this extended optimization as *cross-latch optimization* and consider it a superset of the *inter-latch* optimization techniques originally developed using CPM. Inter-latch optimization optimizes logic *between* latches, cross-latch optimization is able to optimize *across* latch boundaries.

To describe the necessary extensions, we use the latch timing model developed by Sakallah, Mudge, and Olukotun [10]. Model equations and constraints are listed in Table 1, where we include only those relevant to the latest arriving signals. Variables describing clock signals include the cycle time T_c, phase widths T_p, and ending times e_p of each phase Φ_p specified in a common frame of reference. Circuit model parameters include latch setup times S_i, and the maximum delay between each connected pair of latches Δ_{ji}[2]. The data input of each latch is modeled by the latest possible time at which a new signal can arrive A_i. Latch outputs are modeled by the latest times at which new signals depart from the latch D_i. Arrival and departure times are defined in a frame-of-reference local to the corresponding latch. p_i denotes the clock controlling latch *i*. A phase-shift operator E_{p_j, p_i} is used to convert signal times from the frame-of-reference of latch *j* to that of latch *i*.

In [11], it was observed that the constraints in Table 1 can be represented by a graph, which was used to find the optimal clock schedule for a circuit. A similar graph for-

1. TILOS allowed optimization across a parameterizable number of latches but recommended that this number be kept small, probably to avoid difficulties due to feedback loops in circuits being optimized.
2. For simplicity, latch delays are omitted. They can be included by the addition of terms to equation (7) or (8).



Example Circuit

Constraint Graph

Figure 1: Late Signal Constraint Graph

$$A_i \le T_c - S_i \quad (6)$$

$$D_i = \max(A_i, T_c - T_{p_i}) \quad (7)$$

$$A_i = \max_{j \in FI(i)}(D_j + \Delta_{ji} - E_{p_j p_i}) \quad (8)$$

$$E_{p_j p_i} = T_c - ((e_{p_j} - e_{p_i}) \bmod T_c) \quad (9)$$

Table 1: Timing Model Summary

mulation is shown in Figure 1. In these *late signal constraint graphs*, each latch is represented by a pair of nodes labeled A_i and D_i, which correspond to the arrival and departure times for the latch. A zero-weight arc connects the arrival and departure time nodes and reflects the arrival time terms in equation (7). Arcs labeled Δ_x model individual gate delays. Arcs labeled $-E_{p_j, p_i}$ connect gates to latch inputs and complete the representation of equation (8). All paths through these $-E_{p_j, p_i}$ arcs must come from a latch controlled by phase p_j, however, fanins of multiple phases can be accommodated by duplicating sections of the constraint graph. The clock system and its associated constraints are incorporated into the constraint graph with two additional vertices and sets of associated arcs. Clock distribution is modeled by connecting a source vertex *S* to each latch departure time vertex with arcs weighted $T_c - T_i$. These arcs model the occurrence of the rising edge of the clock controlling latch *i*. Arrival time constraints are enforced by connecting arrival time vertices to a sink vertex *F* with arcs weighted $S_i - T_c$. If necessary, clock skew parameters can be included in the weights of these arcs connected to the source and sink nodes. Setting $e(S) = r(F) = 0$, $e(F) > 0$ if and only if a setup violation exists in the circuit, and when positive, $e(F)$ is the amount of the largest setup violation in the circuit.

If the circuit contains cycles of latches, we cannot directly apply the CPM-based techniques of Section 2, since in cyclic circuits of level-sensitive latches, we must be careful not to violate constraints imposed by loops of transparent latches [11, 12]. Each such loop adds a constraint of the form $\Delta_{\text{TOTAL}} \le nT_c$, where Δ_{TOTAL} is the total delay around the loop and nT_c is the time available for signals to propagate around the loop. Although there is one constraint for each loop, they can be combined into a single lower bound on T_c.

Timing-driven design requires slack values to indicate which specific constraints are violated and how changes to individual delays affect circuit timing. The late signal constraint graphs were formulated so that slack and float values correspond to the amounts by which times or delays could be increased without violating a setup constraint, but there is no similar quantity available to ensure that the loop constraints are all satisfied. It is not difficult to construct circuits with large setup time slacks but with a critical loop constraint. For these circuits, increases in delays based on setup slacks can result in timing errors.

There are a few alternatives for ensuring that loop constraints are satisfied. If the optimization is formulated as a linear or nonlinear programming problem [13], the constraints will be enforced implicitly. If we require slack values, one solution would enumerate all possible cycles in the graph and calculate loop slacks based on total loop delays and loop timing budgets. Clearly, however, this is impractical for general circuits, as the number of loops can grow exponentially with the number of latches.

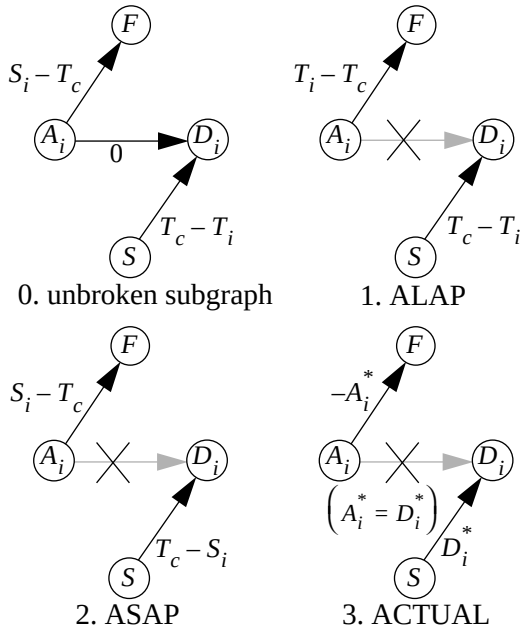


Figure 2: Cycle-Breaking Strategies

Our approach breaks cycles in the constraint graph, modifying the graph to guarantee that all loop constraints will be satisfied. We artificially break the cycles in the constraint graph by fixing the departure times of selected latches to some maximum value and requiring that their arrival times be no greater than these fixed departure times. This ensures that the departure times will be no greater than the specified values, allowing us to safely ignore the dependency between arrival and departure times for this subset of latches. After breaking loops in this manner, we have an acyclic graph to which we can apply a wide variety of CPM-based analysis techniques, including those of Section 2. The approach is conservative, since it only allows the optimization to cross a subset of latches, but is easy to implement, and the remaining graph accurately models all the unmodified latches in the circuit and allows the arrival and departure times at these latches to vary during the optimization. We may fix the departure times at breakpoint latches in several ways, including:.

1. ALAP: Signals depart as late as possible. Departure times are set to their latest possible values.

2. ASAP: Signals depart as early as possible. Arrival time constraints are tightened to allow this departure.

3. ACTUAL: Signals are assumed to arrive and depart at times A_i^* and D_i^* determined by a preliminary timing analysis.

Each of the three cycle-breaking methods is illustrated in Figure 2. ALAP and ASAP arbitrarily fix a departure time at its latest or earliest possible value, possibly making a feasible cycle time appear infeasible under the modified constraints. The third method, ACTUAL, uses times determined by a preliminary analysis. If the analysis is performed at a feasible cycle time, the added constraints will not cause this cycle time to appear infeasible. However, this requires that all loop constraints be satisfied at the target cycle time from the outset

When selecting arcs to remove from the constraint graph, a reasonable goal would be to minimize the effects of breaking loops on the timing of the circuit being optimized. Since each broken arc adds extra timing constraints, it would be natural to seek to break as few arcs as possible to make the circuit acyclic. This goal reduces to the problem FEEDBACK ARC SET, a well-known NP-hard problem [14]. However, a depth-first traversal will quickly find a sufficient set of arcs to remove that can make the graph acyclic. We recursively traverse the graph, marking nodes as they are visited. When a mark is found, a cycle has been located and can be observed in a stack of nodes currently being expanded. We can then simply look back into this stack to find the first $A_i \rightarrow D_i$ arc, which is removed to break the cycle, and continue until no more cycles remain.

4 Experiments

Inter-latch and cross-latch variations of the optimization algorithms of Section 2 were evaluated using ISCAS89 benchmark circuits. The original ISCAS89 circuits were synchronized using edge-triggered devices and a single-phase clock. To obtain a variety of level-sensitive circuit structures, we transformed the benchmarks in three ways:

1. by replacing edge-triggered devices with level-sensitive latches and considering the late signal constraints only (hold time constraints are ignored). These circuits have names beginning with the letter “s”, e.g., *s953*.

2. by replacing edge-triggered devices with pairs of level-sensitive latches controlled by alternate phases of a two-phase clock. The circuits were then retimed to minimize cycle time using a procedure similar to that of Ishii et al. [15]. These circuits have names beginning with the letter “t”.

3. by using a doubling transformation described by Szymanski [11]. These circuits have names beginning with the letter “d”.

The circuits used ranged in size from 21 latches and 158 gates (*s382*) to 1642 latches and 15902 gates (*d13207*) Each was controlled by a symmetric clock, and all two-phase clocks were required to be non-overlapping. Since the algorithms we consider involve modifying circuit delays to satisfy a fixed clock schedule, this restriction is a convenience only and does not affect the generality of the approach.

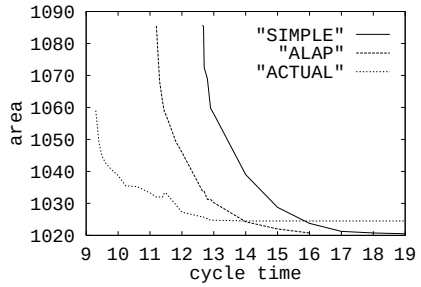
Parts were obtained from the Texas Instruments 1-μ CMOS standard cell library [4]. During retiming transformations, each part was assumed to be implemented using the smallest variant in the library and since existing retiming algorithms do not allow for load-dependent delays, the retimed examples were obtained by assuming that each gate drove a constant number of standard loads. All other analyses included actual loading effects along with standard TI pre-layout estimators for interconnect capacitance.

We sought to compare results obtained using inter-latch optimization with those of cross-latch optimizations using the algorithms of Section 2. Each algorithm was implemented using late signal constraint graphs. The same implementations performed inter-latch or cross-latch optimization, depending on the presence of $A_i \rightarrow D_i$ arcs in the graph.

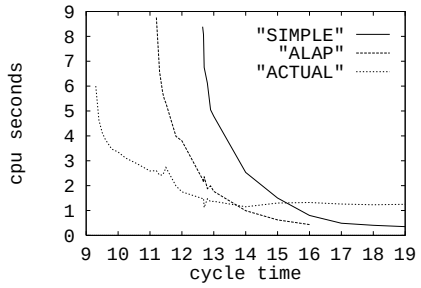
The two part selection algorithms can be used to explore the relationship between the area and the minimum cycle time of a circuit. Each circuit is capable of operating at a variety of speeds, depending on the sizing of its component parts. Assuming all parts are at their minimum size, we can compute a certain minimum cycle time for the circuit. In many cases it is possible to reduce this minimum by adding area to the circuit in the form of larger part variants. As a result, we expect an inverse relationship between area and minimum cycle time.

Figure 3-a shows the area vs. minimum cycle time relationship for benchmark *t953* obtained using the TILOS algorithm. The SIMPLE curve represents simple inter-latch optimization. The ALAP and ACTUAL curves show results of cross-latch optimizations breaking loops using the respective methods. For ACTUAL, initial times were obtained from a sizing using the SIMPLE strategy to first reduce latch-to-latch delays as much as possible. The arrival and departure times at breakpoints were then computed at the minimum cycle time of the presized circuit. Figure 3-b shows the CPU seconds required by each approach on a lightly loaded DEC-station 5000/120 (the ACTUAL curve does not include the constant additional time required for presizing). CPU times are directly related to the amount of additional area required; optimizations requiring larger amounts of additional area require a proportionately larger number of iterations of the TILOS algorithm.

Because of the additional flexibility allowed by trading time across latches, the area-delay curve for the ALAP and ACTUAL approaches are below and to the left of the SIMPLE curve. Because less additional area is required, the running times for these optimizations are also less than those for the SIMPLE strategy. Interestingly, the ACTUAL strategy was superior at small cycle times but was unable to find minimal areas at large cycle times, probably because the optimized starting point introduced



a. Area vs. target cycle time



b. CPU time required to reach target cycle time

Figure 3: Optimization of *t953* with TILOS algorithm

an arrival time constraint that could not be satisfied by the minimum-area circuit.

Similar curves were found for Hinsberger and Kolla's algorithm and are omitted due to space constraints. We observed Hinsberger and Kolla's algorithm to be slightly faster, perhaps because it optimizes an entire path at a time. The TILOS algorithm found smaller implementations for a given cycle time, but the difference was small. Both algorithms show similar improvements when cross-latch optimizations are used, and both produce better results when "initialized" with the timing of a pre-sized circuit.

Table 2 summarizes additional experiments using the TILOS and Hinsberger-Kolla algorithms. Each benchmark circuit was optimized with a target cycle time of T_m, the minimum cycle time reachable using inter-latch optimization. For the cross-latch optimizations, loops were broken using the ALAP method. In the table, the first column identifies benchmark circuits and remaining columns list ratios of additional area and optimization time required to reach the target cycle times for each circuit. In all cases, these ratios were less than one, indicating that the cross-latch optimization approaches required less additional area and less CPU time to reach the same cycle time.

5 Conclusions

We see three general benefits of cross-latch optimization. First, it *is simple to implement*. Each algorithm we examined was formulated using general CPM networks.

circuit	TILOS		Hinsberger-Kolla	
	$\frac{\Delta A_{\text{ALAP}}}{\Delta A_{\text{SIMPLE}}}$	$\frac{CPU_{\text{ALAP}}}{CPU_{\text{SIMPLE}}}$	$\frac{\Delta A_{\text{ALAP}}}{\Delta A_{\text{SIMPLE}}}$	$\frac{CPU_{\text{ALAP}}}{CPU_{\text{SIMPLE}}}$
s382	0.38	0.52	0.32	0.57
s444	0.33	0.51	0.39	0.49
s526	0.26	0.64	0.17	0.35
s953	0.32	0.40	0.38	0.62
s1423	0.42	0.69	0.58	0.22
s9234	0.60	0.74	0.53	0.68
s13207	0.75	0.92	0.62	0.89
t382	0.49	0.71	0.45	0.78
t444	0.32	0.43	0.24	0.57
t526	0.29	0.50	0.24	0.53
t953	0.20	0.29	0.23	0.58
t1423	0.43	0.72	0.58	0.23
t9234	0.58	0.77	0.48	0.70
t13207	0.75	0.99	0.71	1.01
d382	0.26	0.35	0.31	0.56
d444	0.33	0.41	0.37	0.50
d526	0.23	0.42	0.15	0.44
d953	0.29	0.35	0.36	0.64
d1423	0.32	0.56	0.46	0.19
d9234	0.56	N/A[a]	0.60	0.60
d13207	0.73	N/A	0.19	0.38
averages	0.44	0.57	0.42	0.56

Table 2: Experiments conducted at T_m

a. unavailable ratios are due to running times outside the measurable range.

The extended formulation incorporates level-sensitive latch timing behavior and only requires the additional step of breaking cycles in the constraint graph.

Second, cross-latch optimization *produces better results*. In all cases examined, the additional flexibility of cross-latch optimization found solutions of equal or better quality to those of inter-latch optimization.

Third, cross latch optimization *adds no significant computational cost*. The only extra computation required is the inexpensive loop-breaking step. In all cases examined, cross-latch optimizations required *less* time than comparable inter-latch optimizations. The running times of these algorithms depend on the difficulty of satisfying timing constraints; more accurately modeling latch timing eases these constraints, allowing the algorithms to more quickly find a feasible solution.

To these we add the following limitation: cross-latch optimization *requires a target clock schedule*. Traditional critical path methods minimize cycle time by maximizing slack, increasing the margin on the setup constraints. This margin will have a varying influence on the cycle time, depending on the time budgets of the related paths. Inter-latch optimization does not necessarily require a target clock. If the time budgets of all paths are equal, then the minimum cycle time can be obtained by maximizing slack regardless of the cycle time target.

We believe that many other CPM-based optimizations can be similarly extended to perform cross-latch optimization. Other areas for research include evaluation of techniques for loop breaking and development of guidelines for their use. Better optimization solutions may be found using iterative approaches that modify breakpoint arrival and departure times during optimization. Incremental timing analysis would reduce running times. Finally, since the loop breaking modifications fundamentally restrict the solution space, approaches which can be directly used on cyclic constraint graphs could allow further improvement.

References

[1] K. Lockyer and J. Gordon, *Critical Path Analysis and other Project Network Techniques*, Pitman, 1991.

[2] T. I. Kirkpatrick and N. R. Clark, "PERT as an Aid to Logic Design", *IBM Journal of Res. and Dev.*, vol. 10, no. 2, p. 135-141, March 1966.

[3] R. B. Hitchcock, Sr., G. L. Smith, and D. D. Cheng, "Timing Analysis of Computer Hardware", *IBM Journal of Res. and Dev.*, vol. 26, no. 1, p. 100-105, January 1982.

[4] Texas Instruments, *TSC 700 Series 1-micron CMOS Standard Cells*, SRSS035B-D3857, 1992.

[5] J. P. Fishburn and A. E. Dunlop, "TILOS: A Posynomial Programming Approach to Transistor Sizing," in *ICCAD-85 Digest of Technical Papers*, p. 326-328, 1985.

[6] J. M. Shyu, A. Sangiovanni-Vincentelli, J. P. Fishburn, and A. E. Dunlop, "Optimization-Based Transistor Sizing," *IEEE Journal of Solid-State Circuits*, 23(2), p. 400-409, April 1988.

[7] S. Lin, M. Marek-Sadowska, and E. S. Kuh, "Delay and Area Optimization in Standard-Cell Design," in *Proc. Design Automation Conf.*, p. 349-352, 1990.

[8] U. Hinsberger and R. Kolla, "A Cell-Based Approach to Performance Optimization of Fanout-Free Circuits," *IEEE Trans. on Computer-Aided Design*, 11(10), p. 1317-1321, October 1992.

[9] U. Hinsberger and R. Kolla, "Cell Based Performance Optimization of Combinational Circuits," in *Proc. European Conf. on Design Automation*, p. 594-599, 1990.

[10] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. *checkT_c* and *minT_c*: Timing Verification and Optimal Clocking of Synchronous Digital Circuits, in *ICCAD-90 Digest of Technical Papers*, p. 552-555, 1990.

[11] T. G. Szymanski, "Computing Optimal Clock Schedules," In *Proc. Design Automation Conf.*, p. 399-404, 1992.

[12] T. M. Burks, K. A. Sakallah, and T. N. Mudge, "Identification of Critical Paths in Circuits with Level-Sensitive Latches", in *ICCAD-92 Digest of Technical Papers*, p. 137-141, 1992.

[13] W. Chuang, S. S. Sapatnekar, and I. N. Hajj, "A Unified Algorithm for Gate Sizing and Clock Skew Optimization to Minimize Sequential Circuit Area," in *Proc. Design Automation Conf.*, p. 220-223, 1993.

[14] R. M. Karp, "Reducability Among Combinatorial Problems," in R.E. Miller and J. W. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Press, New York, p. 85-103, 1972.

[15] A. Ishii, C. E. Leiserson, and M. C. Papaefthymiou, "Optimizing Two-Phase Level-Clocked Circuitry," in *Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference*, p. 245-264, 1992.