Algorithms for Timing Verification and Optimal Clocking of Synchronous Digital Circuits

By

K.A. Sakallah, T.N. Mudge and O.A. Olukotun

CSE-TR-71-90

THE UNIVERSITY OF MICHIGAN

COMPUTER SCIENCE AND ENGINEERING DIVISION
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
Room 3402 EECS Building
Ann Arbor, Michigan 48109-2122
USA

Algorithms for Timing Verification and Optimal Clocking of Synchronous Digital Circuits

Karem A. Sakallah, Trevor N. Mudge and Oyekunle A. Olukotun
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122

Phone: 313-936-1350
FAX: 313-763-4617
E-mail: karem@razi.eecs.umich.edu

September 6, 1990

Abstract

The need for accurate characterization of the timing behavior of synchronous digital systems has become increasingly evident in the past few years. The continuing drive to design faster digital systems has made the use of CAD tools for timing verification and optimal clocking essential in most system design methodologies. Timing models of synchronous digital circuits are based on the premise that the circuits are logically (functionally) correct, and focus only on capturing their propagation, latching, and synchronization properties. While, in principle, such timing models are much simpler than full-fledged logical models, the increasing use of multi-phase clocking schemes and level-sensitive latching structures has significantly increased their complexity. In this report we describe the algorithms in two new CAD tools, $checkT_c$ and $minT_c$, for timing verification and optimal clocking. Both tools are based on a new timing model of synchronous digital circuits which is: 1) general enough to handle arbitrary multi-phase clocking; 2) complete, in the sense that it captures signal propagation along short as well as long paths in the logic; 3) extensible to make it relatively easy to incorporate "complex" latching structures; and 4) notationally simple to make it amenable to analytic treatment in some important special cases. We are currently using these tools to help design a 4ns gallium arsenide micro-supercomputer.

1 Introduction

This report introduces a *timing model* of synchrounous digital circuits and describes its use in two CAD applications, *timing verification* , and *optimal clocking* . The model assumes that the circuits are logically (functionally) correct, and focuses only on capturing their propagation, latching, and synchronization properties. It can be viewed as a synthesis of ideas from earlier work in the field of timing analysis of synchronous systems. On the other hand, it also includes several key concepts which make it—as will become evident—significantly more versatile than previous models. In particular, on the issue of level-sensitive latches which has received a great deal of attention recently [1]–[10], the model offers a general and accurate, yet simple, treatment.

Briefly, the model can be characterized by the following:

- A clear distinction between *data* and *clock* signals.
- A general treatment of *multi-phase* clocks which underscores the centrality of the common clock cycle T_c and emphasizes the *temporal* rather than the *logical* relations among clock *phases*; in particular, the notions of phase-relative *time zones* and of a *phase-shift operator* eliminate much of the notational clutter which has plagued previous efforts in dealing with "complex" clocking schemes.
- An extensible framework which allows the incorporation of arbitrary *synchronizing* structures, i.e. circuit structures where clock and data signals converge and interact, as long as a timing *macromodel* of these interactions can be provided; in this report we present timing macromodels for D-type latches and flip-flops.
- Completeness, in the sense that the model accounts for signal propagation along the *shortest* as well as the *longest* paths in a circuit.

and, finally,

- Simplicity, through careful attention to notation and the use of variable and parameter symbols with mnemonic value.

We have implemented the model in two prototype CAD tools. The first tool, *check*T_c , is a timing verifier which examines a circuit for adherence to a specified clock schedule, and reports on setup

and hold time violations. The second tool, $minT_c$, is a clocking optimizer which determines the optimal clock schedule (i.e. the schedule with the minimum cycle time) that satisfies all the timing constraints for a given circuit.

The remainder of the report is organized as follows. In Sec. 2 we present a brief review of previous work. The formulation of the proposed timing model is developed in Sec. 3. Section 4 presents the timing verification algorithm used in $checkT_c$. The optimal clocking algorithms used in $minT_c$ are presented in Sec. 5. The use of the tools in the design of a 4ns micro-supercomputer is breifly described in Sec. 6, and we close with conclusions and suggestions for future work in Sec. 7.

2 Previous Work

The timing analysis of digital logic circuits goes back at least to the work of Kirkpatrick in the 1960's [11]. Since then, a great deal of effort has been devoted to the subject. However, much of this work, including Kirkpatrick's original report, was concerned with timing analysis for edge-triggered logic employing simple clocking methodologies. The timing models for this class of circuits are fairly simple because edge-trigerring decouples the timing of flip-flop outputs from the timing of their inputs. In contrast, the timing models for circuits employing level-sensitive latches are coupled and considerably more complex; studies of latch-controlled circuits during that period were, therefore, mostly limited to regular circuit topologies, such as pipelines [12]. It has only been during the last ten years, with MOS VLSI emerging as the leading technology for implementing digital systems, that the timing analysis and design of level-sensitive logic with sophisticated clocking schemes has become important. In this period several authors have addressed the question of level-sensitive latches and multi-phase clocking including McWilliams [13], Agrawal [14], Jouppi [1], Ousterhout [2], Glesner [3], Unger [4], Szymanski [5], Cherry [6], Wallace [7], Ishiura [8], Weiner [9], and Dagenais [10]. Space does not permit us to review these contributions here; interested readers are referred to [15] for additional details.

3 Timing Model

The timing model we present here extends our earlier model in [16] to handle propagation along short paths. In fact, aside from some minor changes in notation, the "new" model is a superset of the earlier one (see Sec. 3.5). The model is based on making a distinction between two types of signals in a synchronous circuit: *clock* signals and *data* signals. Clock signals are used to *regulate* the flow of data signals in the circuit to make sure that no data signal changes any sooner or any later than it is supposed to. This *synchronizing* action of clock signals is predicated on precisely knowing the value of a clock signal at all instants during a clock cycle[1]. The values of data signals, on the other hand, need not be known precisely; it is sufficient for timing purposes to merely know when a data signal is *stable* and when it is *changing* during a clock cycle. Much of the power of timing models, over logical models, is due to this *data-independence* which effectively achieves complete coverage of *all* signal propagation paths in a circuit without having to specify test vectors at the circuit inputs[2].

The distinction between data and clocks leads naturally to three categories of timing relations in a synchronous circuit:

1. Relations among different clock signals, or phases, of a periodic clock.
2. Relations among data and clock signals for various types of synchronizing elements.
3. Relations among different data signals at the inputs and outputs of combinational logic blocks.

The remainder of this section is devoted to the development of each of these three categories of timing relations. Collectively, these relations will be referred to as the *General System Timing Constraints (GSTC)*. A retricted version of these constraints based on assuming that the minimum propagation delays are zero is also derived. This subset of the STC will be called the *Restricted System Timing Constraints (RSTC)*.

[1] Some uncertainty in the value of a clock signal is, however, unaviodable because of finite rise and fall times.
[2] Of course, some logically-impossible paths may also be covered necessitating the inclusion of some degree of data-dependence [1, 2].

3.1 Clocking Model

We define an arbitrary k-phase clock to be a collection of k periodic signals ϕ_1 , ϕ_2 , $\cdots$, ϕ_k — referred to as *phases* — with a common cycle time T_c. The phase signals are applied to the control inputs of synchronizing elements, such as latches and flip-flops. Each phase divides the clock cycle into two intervals: an *active* interval of duration T_i, and a *passive* interval of duration $(T_c - T_i)$. During the active interval of a given phase, the synchronizers it controls are *enabled*; during its passive interval, they are *disabled*. The transitions *into* and *out of* the active interval will be called, respectively, the *enabling* and *latching* edges of the phase. We assume, without loss of generality, that all phases are active high; thus, the enabling and latching edges correspond to the rising and falling transitions of the phase signal. It is important to point out that the phase signals are *not* required to be non-overlapping. In fact, as we will see in Sec. 6, shorter cycle times can be obtained when the clock phases are allowed to overlap.

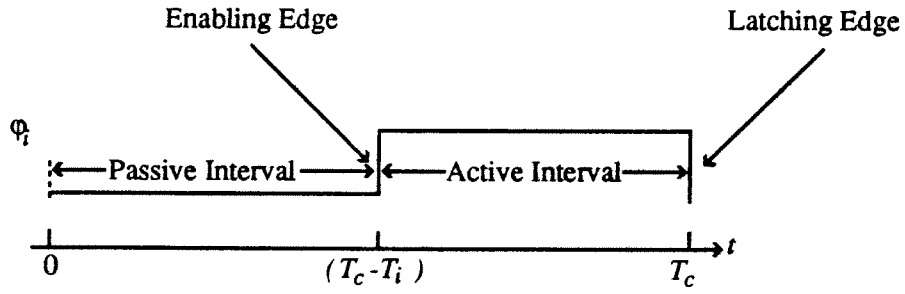

Figure 1: Clock phase ϕ_i and its local time zone

Associated with each phase is a *local time zone*, as shown in Fig. 1, such that its passive interval starts at $t = 0$, its enabling edge occurs at $t = T_c - T_i$, and its latching edge occurs at $t = T_c$. Mathematically, the domain of the local time zone is defined to be the interval $(0, T_c]$ since the start of the current clock cycle coincides with the end of the previous cycle. The temporal relationships among the k phases (i.e. among the different time zones) can now be established by an arbitrary choice of a *global time reference*. We introduce e_i to denote the time, relative to this global time reference, at which phase ϕ_i *ends* (i.e. when its latching edge occurs). We also assume that the phases are *ordered*[3] in this global time reference so that $e_1 \leq e_2 \leq \cdots \leq e_{k-1} \leq e_k$. The global time reference is now *arbitrarily* chosen to coincide with the local time zone of ϕ_k;

[3]This ordering does not imply any restrictions on clocking; it is merely a labeling device for notational convenience.

thus $e_k \equiv T_c$.

Finally, we define a *phase shift* operator:

$$E_{ij} \equiv \begin{cases} (e_j - e_i)\,, & i < j \\ (T_c + e_j - e_i)\,, & i \geq j \end{cases} \tag{1}$$

which takes on positive values in the range $[0, T_c]$. When subtracted from a timing variable in the *current* local time zone of ϕ_i, E_{ij} changes the frame of reference to the *next* local time zone of ϕ_j, taking into account a possible cycle boundary crossing.

3.2 Synchronizer Model

We develop here the timing relations for D-type synchronizers, either latches or flip-flops[4] with three terminals each: data input, data output, and clock input. The latches can be either static (for example cross-coupled NAND gates) or dynamic (for example MOS pass transistors). The circuit is assumed to contain l synchronizers numbered from 1 to l. The ith synchronizer is characterized by the following five parameters:

- p_i: clock phase used to control synchronizer i.
- S_i: setup time of synchronizer i relative to latching edge of p_i.
- H_i: hold time of synchronizer i relative to latching edge of p_i.
- δ_i , Δ_i: minimum and maximum propagation delay of synchronizer i; for latches, propagation is from the data input to the data output; for flip-flops, propagation is from the clock input to the data output.

For timing purposes, it is sufficient to characterize a data signal over one clock cycle by two, possibly simultaneous, events which demark the interval when the signal is switching between its old and new values. For the signal *arriving* at the data input of synchronizer i these two events are defined to occur at $t = a_i$ and $t = A_i$ in the local time zone of phase p_i. The corresponding events of the data signal *departing* from the synchronizer are defined to occur at $t = d_i$ and $t = D_i$. It will be convenient to refer to a_i and A_i as the *early* and *late* arrival times, and to d_i

[4]The model described here applies to negative edge-triggered flip-flops. Models for positive edge-triggered and master-slave flips-flops can be defined similarly.

and D_i as the *early* and *late* departure times. The synchronizing action can now be completely specified in terms of these four event times as follows (see Fig. 2):

- Latch Synchronization:

$$d_i = \max(a_i, T_c - T_{p_i})$$
$$D_i = \max(A_i, T_c - T_{p_i})$$

- Flip-flop Synchronization:

$$d_i = D_i = T_c$$

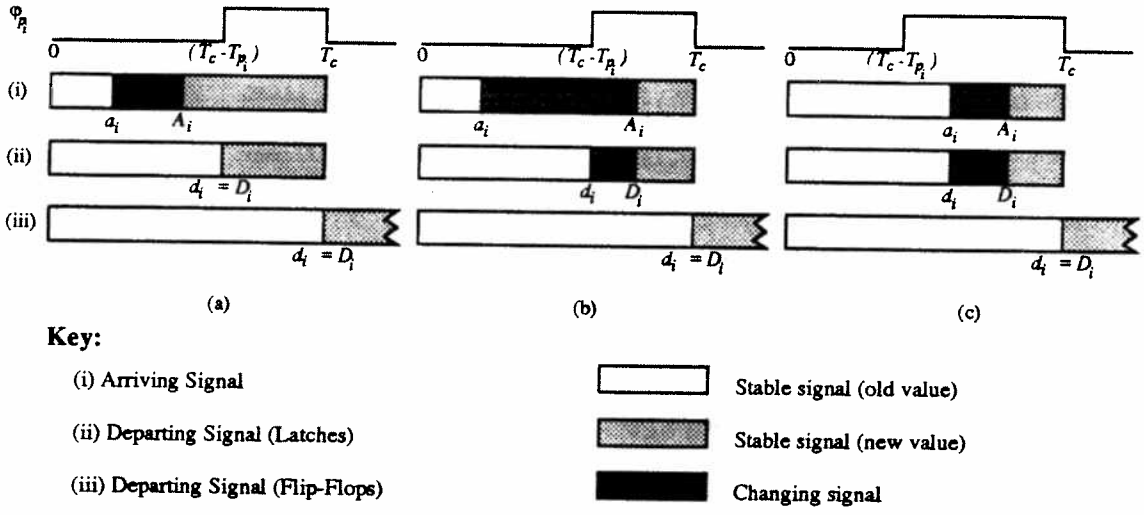

Figure 2: Synchronizing Action of Latches and Flip-Flops

The above equations clearly illustrate the difference between the operation of latches and flip-flops: for flip-flops, the timing of the departing signal is *independent* of the timing of the arriving signal; for latches, the timing of the departing signal is *dependent* on the timing of the arriving signal. It is this dependence which causes much of the complexity in dealing with latch-controlled circuits.

For correct operation, the arriving data signal must satisfy the following hold and setup time requirements (Fig. 3):

$$a_i \geq H_i$$
$$A_i \leq T_c - S_i$$

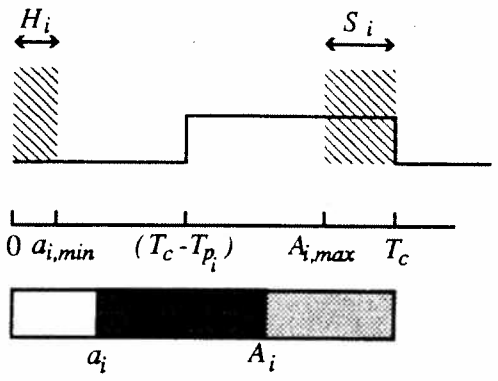

Figure 3: Hold and setup time constraints

3.3 Combinational Logic Model

The combinational logic in the circuit is assumed to have been decomposed into *stages* with clocked inputs and outputs[5] (see Fig. 4), and is characterized for timing purposes by two $l \times l$ matrices whose elements are defined as follows:

- δ_{ij}: minimum propagation delay from the data output of synchronizer i through a combinational logic stage to the data input of synchronizer j; if synchronizers i and j are not directly connected by combinational logic, then $\delta_{ij} \equiv \infty$.

- Δ_{ij}: maximum propagation delay from the data output of synchronizer i through a combinational logic stage to the data input of synchronizer j; if synchronizers i and j are not directly connected by combinational logic, then $\Delta_{ij} \equiv -\infty$.

Both matrices δ and Δ can be simultaneously obtained by a breadth-first or a depth-first critical path algorithm applied to each combinational stage in the circuit[18, Ch. 3].

The temporal model of the combinational logic can now be stated by the following two equations:

$$\begin{aligned} a_i &= \min_j(d_j + \delta_j + \delta_{ji} - E_{p_j p_i}) \qquad i,j = 1,\cdots,l \\ A_i &= \max_j(D_j + \Delta_j + \Delta_{ji} - E_{p_j p_i}) \quad i,j = 1,\cdots,l \end{aligned}$$

[5] Figure 4 is adapted from [17, Fig. 6.7 p. 335].

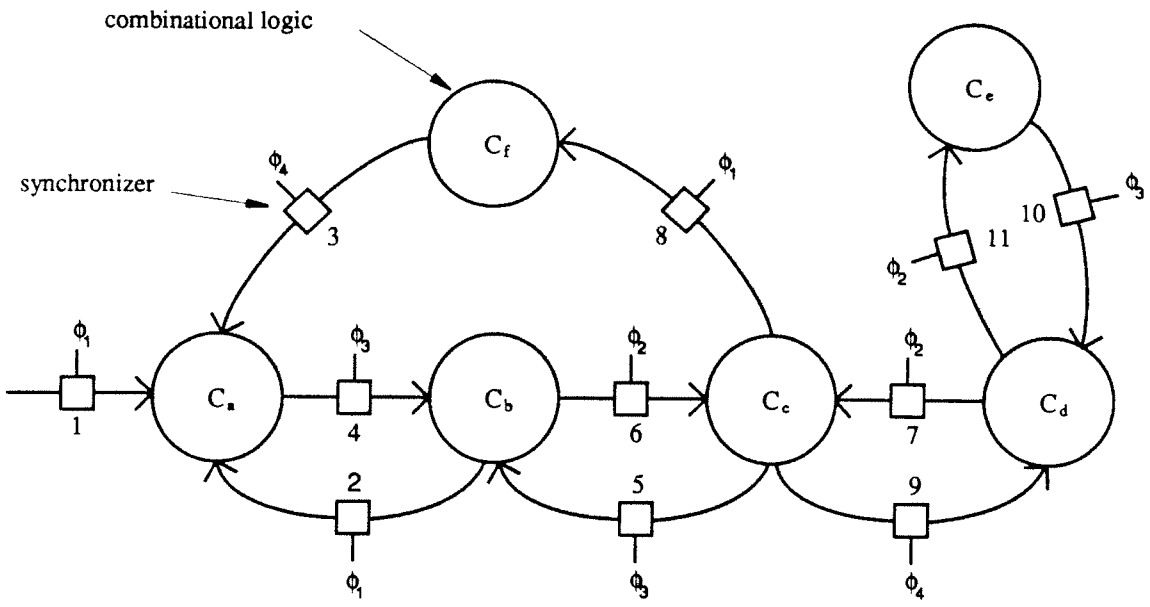

Figure 4: Generalized Synchronous Circuit Model

These two equations express the two arrival time events at synchronizer i in terms of the corresponding departure time events at its *fan-in* synchronizers j. For example, in Fig. 4, $A_7 = \max(D_9 + \Delta_9 + \Delta_{97} - E_{42}, D_{10} + \Delta_{10} + \Delta_{10,7} - E_{32})$. The subtraction of the phase shift operators E_{42} and E_{32} achieves the necessary and unifying change of reference from the local time zones of phases ϕ_4 and ϕ_3 to that of phase ϕ_2.

3.4 General System Timing Constraints

The complete set of general system timing constraints, GSTC, is summarized in Table 1. For ease of reference, the constraints have been organized into five categories, and have been given mnemonic labels.

3.5 Restricted System Timing Constraints

Constraints GC1-GC5 are complete in the sense that they model propagation along both the shortest and longest paths, and insure that hold and setup time requirements are satisfied at all synchronizers. A frequently made simplification is to assume that the minimum propagation delays in the circuit are zero. This assumption makes it possible to replace the short-path constraints by a

GC1. Phase Ordering Constraints:	$e_{i-1} \leq e_i$	$i = 2, \cdots, k$
	$e_k \equiv T_c$	
GC2. Latching Constraints:		
GC2H. Hold Time Constraints:	$a_i \geq H_i$	$i = 1, \cdots, l$
GC2S. Setup Time Constraints:	$A_i \leq T_c - S_i$	$i = 1, \cdots, l$
GC3. Synchronization Constraints:		
GC3L. Latch Synchronization:		
GC3Ld. Early Departure:	$d_i = \max(a_i, T_c - T_{p_i})$	$i = 1, \cdots, l$
GC3LD. Late Departure:	$D_i = \max(A_i, T_c - T_{p_i})$	$i = 1, \cdots, l$
GC3F. Flip-Flop Synchronization:		
GC3Fd. Early Departure:	$d_i = T_c$	$i = 1, \cdots, l$
GC3FD. Late Departure:	$D_i = T_c$	$i = 1, \cdots, l$
GC4. Combinational Propagation Constraints:		
GC4a. Early Arrival:	$a_i = \min_j(d_j + \delta_j + \delta_{ji} - E_{p_j p_i})$	$i, j = 1, \cdots, l$
GC4A. Late Arrival:	$A_i = \max_j(D_j + \Delta_j + \Delta_{ji} - E_{p_j p_i})$	$i, j = 1, \cdots, l$
GC5. Bound Constraints:	$0 \leq e_i \, , \, T_i \leq T_c$	$i = 1, \cdots, k$
	$0 \leq a_i \, , \, A_i \, , \, d_i \, , \, D_i \leq T_c$	$i = 1, \cdots, l$

Table 1: General System Timing Constraints (GSTC)

smaller set of phase non-overlap constraints. This can be readily seen for latch-type synchronizers by substituting $\delta_j = \delta_{ji} = 0$ and $d_j = T_c - T_{p_j}$ in constraint GC4a and combining the result with the hold time inequality, GC2H, to yield:

$$T_c - T_{p_j} - E_{p_j p_i} \geq a_i \geq H_i \tag{2}$$

This constraint is illustrated in Fig. 5 which clearly shows that the next enabling edge of phase p_j must not occur any sooner than H_i after the latching edge of phase p_i. In other words, this constraint insures that the *end* of the active interval of phase p_i is separated from the *beginning* of the active interval of phase p_j by, at a minimum, the hold time of latch i.

We can now write the system timing constraints exclusively in terms of the late arrival and departure variables, without having to worry about short paths. This is facilitated by introducing one last variable, L_{ij}, defined as the set of synchronizers which are controlled by phase ϕ_j, and which also receive one or more data inputs from synchronizers controlled by phase ϕ_i. Thus, in Fig. 4, $L_{24} = \{9\}$, $L_{32} = \{6, 7, 11\}$, and $L_{41} = \emptyset$ (the empty set). The complete set of restricted system timing constraints, RSTC, is summarized in Table 2. As we suggested earlier, the RSTC are basically the same as the constraints reported in [16]. They differ from these earlier constraints in that they:

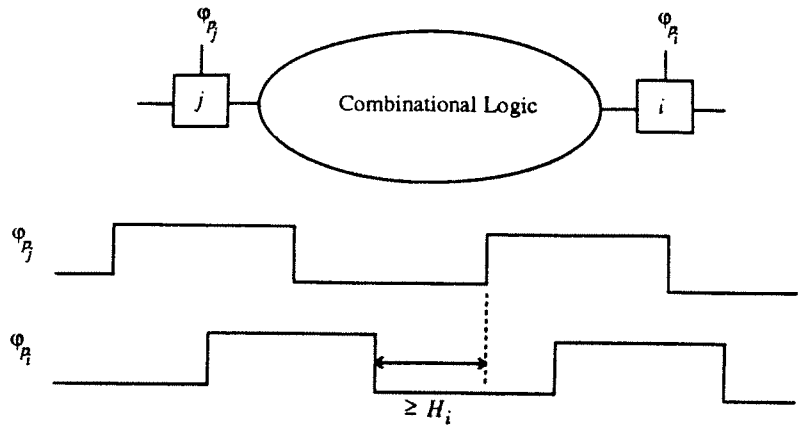

Figure 5: Phase Non-overlap Constraints

RC0. Phase Non-Overlap Constraints:		
	$T_c - T_i - E_{ij} \geq \max_{n \in L_{ij}}(H_n)$	$\forall_{i,j} \ni L_{ij} \neq \emptyset$
RC1. Phase Ordering Constraints:	$e_{i-1} \leq e_i$	$i = 2, \cdots, k$
	$e_k \equiv T_c$	
RC2. Latching (Setup) Constraints:	$A_i \leq T_c - S_i$	$i = 1, \cdots, l$
RC3. Synchronization Constraints:		
RC3L. Latch Synchronization:	$D_i = \max(A_i, T_c - T_{p_i})$	$i = 1, \cdots, l$
RC3F. Flip-Flop Synchronization:	$D_i = T_c$	$i = 1, \cdots, l$
RC4. Combinational (Late Arrival) Propagation Constraints:		
	$A_i = \max_j(D_j + \Delta_j + \Delta_{ji} - E_{p_j p_i})$	$i, j = 1, \cdots, l$
RC5. Bound Constraints:	$0 \leq e_i$, $T_i \leq T_c$	$i = 1, \cdots, k$
	$0 \leq A_i$, $D_i \leq T_c$	$i = 1, \cdots, l$

Table 2: Restricted System Timing Constraints (RSTC)

- use a different frame of reference for time;
- allow for nonzero hold times;

and,

- use A_i instead of D_i in the setup requirements.

In addition, because the RSTC are derived as a special case of the GSTC, we can still obtain from them the values of the early arrival and departure variables. Specifically,

$$d_i = \begin{cases} T_c - T_{p_i} , & i = 1, \cdots, l \quad \text{(latches)} \\ T_c , & i = 1, \cdots, l \quad \text{(flip-flops)} \end{cases} \tag{3}$$

and,

$$a_i = \min_j(d_j - E_{p_j p_i}) \quad i, j = 1, \cdots, l \tag{4}$$

4 Timing Verification

The primary goal of timing verification is to determine if a given circuit can be operated with a specified clock schedule without any setup or hold time violations. This can be conveniently achieved by first finding a solution to the synchronization and propagation constraints (GC3 and GC4), and then checking if this solution satisfies the latching constraints (GC2). With a fixed clock schedule, such a solution can be found by *several* passes of a CPM-like traversal of the circuit graph. Iteration is necessary if the circuit contains latches connected in feedback loops; if the only synchronizers in the circuit are flip-flops, a single traversal would be sufficient.

Defining V_i^S and V_i^H to be the setup and hold time violations at synchronizer i, Alg. 4.1[6] incorporates the above approach and forms the basis of the timing verification procedure in the *checkT$_c$* tool. Several observations should be made about this algorithm:

- The late and early departure times are initialized, respectively, to their minimum (line 3) and maximum (line 4) possible values. With these starting values, successive updates of each D_i (d_i) during the iterative portion of the algorithm (lines 7–27) are guaranteed to be non-decreasing (non-increasing). In other words, for $m = 1, 2, \cdots$, $D_i^m \geq D_i^{m-1}$ and $d_i^m \leq d_i^{m-1}$. This in turn guarantees that $A_i^m \geq A_i^{m-1}$ and $a_i^m \leq a_i^{m-1}$ for $m = 2, 3, \cdots$. The iteration can thus be conveniently viewed as a process of "sliding" each D_i and A_i to the right and each d_i and a_i to the left.

- The late and early arrival times are *clipped*, respectively, at their maximum (line 16) and minimum (line 20) possible values. This clipping has the desireable effect of localizing the corresponding setup or hold violation to the latch in question for better diagnostics.

- The iteration in the algorithm is guaranteed to terminate because:

  1. successive updates of the late departure times are non-decreasing, and are bound from above by $T_c - S_i$; and,

[6] Without loss of generality, we assume for purposes of describing the various algorithms in this reportthat the synchronizers in the circuit are all D-type latches. For circuits containing mixtures of latches and flip-flops, the algorithms should be amended appropriately.

2. successive updates of the early departure times are non-increasing, and are bound from below by $T_c - T_{p_i}$.

- The update formulas in the algorithm (lines 11,12,22,23) have the flavor of a mixed Jacobi/Gauss-Seidel iteration. Other variants are obviously possible. In fact an event-driven update mechanism which only calculates those arrival and departure times which have changed from the previous iteration can be easily implemented. With such an enhancement the cost of the iterative steps can be greatly reduced for large circuits.

- In addition to computing setup and hold time violations, it is possible to extend Alg. 4.1 to identify the *critical* path segments in the circuit. The long path from latch j to latch i is considered critical if $D_j + \Delta_j + \Delta_{ji} - E_{p_j p_i} = T_c - S_i$. Similarly, the short path from latch j to latch i is critical if $d_j + \delta_j + \delta_{ji} - E_{p_j p_i} = H_i$. Furthermore, the *slacks* in the other (non-critical) path segments can be easily found.

5 Calculation of Optimal Cycle Time

The minimum clock cycle time can be found by solving either of the following two optimization problems, depending on which set of system timing constraints is selected:

Program PG: Optimal Cycle Time—GSTC

Minimize T_c

Subject to GSTC (Table 1)

Program PR: Optimal Cycle Time—RSTC

Minimize T_c

Subject to RSTC (Table 2)

These two problems are nonlinear because of the min and max functions in their constraints, implying that their solutions would be difficult to obtain. Fortunately, the nonlinearities in these constraints are mild, and their form suggests an associated, and presumably easier-to-solve, *linear* version of each of the optimization problems. A major result of our work is to show that it is in fact possible to obtain the optimal solutions of PG and PR indirectly by first solving a companion linear program (LP).

Algorithm 4.1: Timing Verification in *check*T_c

```
1.  for (i = 1, ..., l) {                          /* Initialize ... */
2.      V_i^S = V_i^H = 0 ;                        /* ... Setup/Hold Violations */
3.      D_i^0 = T_c - T_{p_i} ;                    /* ... Late Departure Times */
4.      d_i^0 = T_c - S_i ;                        /* ... Early Departure Times */
5.  } /* for */
6.  m = 0;                                         /* ... Iteration Counter */
7.  repeat                                         /* Iterate ... */
8.      done = true ;                              /* Reset Convergence Flag */
9.      m = m + 1 ;                                /* Increment Iteration Counter */
9.      for (i = 1, ..., l) {                      /* Visit All Latches */
10.         for (j = 1, ..., l) {                  /* Update Arrival Times ... */
11.             A_i^m = max_j(D_j^{m-1} + Δ_j + Δ_ji - E_{p_j p_i}) ; /* ... Late Arrival */
12.             a_i^m = min_j(d_j^{m-1} + δ_j + δ_ji - E_{p_j p_i})   /* ... Early Arrival */
13.         } /* for */
14.         if (A_i^m ≥ T_c - S_i) {               /* Check Setup Time */
15.             V_i^S = A_i^m - T_c + S_i ;        /* Calculate Setup Violation */
16.             A_i^m = T_c - S_i                  /* Clip Late Arrival Time */
17.         } /* if */
18.         if (a_i^m ≤ H_i) {                     /* Check Hold Time */
19.             V_i^H = H_i - a_i^m ;              /* Calculate Hold Violation */
20.             a_i^m = H_i                        /* Clip Early Arrival Time */
21.         } /* if */
22.         D_i^m = max(A_i^m, T_c - T_{p_i}) ;    /* Update Late Departure Time */
23.         d_i^m = max(a_i^m, T_c - T_{p_i}) ;    /* Update Early Departure Time */
24.         if ((D_i^m ≠ D_i^{m-1}) or (d_i^m ≠ d_i^{m-1}))   /* Check Convergence */
25.             done = false ;
26.     } /* for */
27. until done;
```

We begin by noting that the min and max constraints can be easily *relaxed* and converted to sets of linear inequalities. For example, the late departure synchronization constraint, RC3L, for the ith latch would be replaced by two linear inequalities as follows:

$$D_i = \max(A_i, T_c - T_{p_i}) \Longrightarrow \begin{cases} D_i \geq A_i \\ D_i \geq T_c - T_{p_i} \end{cases}$$

Similarly, the early arrival constraint for the ith synchronizer, GC4a, would be replaced by l linear inequalities:

$$a_i = \min_j (d_j + \delta_j + \delta_{ji} - E_{p_j p_i}) \Longrightarrow a_i \leq (d_j + \delta_j + \delta_{ji} - E_{p_j p_i})$$

In what follows we will refer to the *relaxed* versions of the GSTC and RSTC by XGSTC and XRSTC. Using the relaxed constraints, we now define the following linear programs:

Program PXG: Optimal Cycle Time—XGSTC

Minimize T_c

Subject to XGSTC

Program PXR: Optimal Cycle Time—XRSTC

Minimize T_c

Subject to XRSTC

5.1 Minimum Cycle Time—The RSTC Case

We show in this section that the minimum cycle time found by solving the nonlinear program PR is the same as that found by solving the linear program PXR. We also present an algorithm for obtaining the optimal solution of PR by a simple modification of the optimal solution of PXR. These results are essentially the same as those we reported in [16] for an earlier formulation of the system timing constraints.

Denoting the optimal values of PR and PXR by $T_{c,min}^{(R)}$ and $T_{c,min}^{(XR)}$, the main result of this section is now simply stated by:

Theorem 5.1 $T_{c,min}^{(R)} = T_{c,min}^{(XR)}$.

Proof: The proof is based on showing that program PR is equivalent to program PXR augmented with extra constraints, and that the optimal value of this augmented linear program is the same as that of program PXR. For the detailed proof steps see [16, Theorem 3.1]. □

This theorem forms the basis for Alg. 5.1 which finds the optimal solution of PR by first solving the linear program PXR. Note that the solution of program PXR (line 1) provides an initial *guess*

Algorithm 5.1: Find Optimal Solution of Program PR

1. *Solve PXR; Let* D_i^0 *be the optimal values of the late departure variables.*
2. $m = 0$; /* *Initialize Iteration Counter* */
3. **repeat** /* *Iterate* $\cdots$ */
4. $done$ = **true** ; /* *Reset Convergence Flag* */
5. $m = m + 1$; /* *Increment Iteration Counter* */
6. **for** $(i = 1, \cdots, l)$ { /* *Visit All Latches* */
7. **for** $(j = 1, \cdots, l)$ /* *Update Late Arrival Times* */
8. $A_i^m = \max_j(D_j^{m-1} + \Delta_j + \Delta_{ji} - E_{p_j p_i})$;
9. $D_i^m = \max(A_i^m, T_c - T_{p_i})$; /* *Update Late Departure Time* */
10. **if** $(D_i^m \neq D_i^{m-1})$ /* *Check Convergence* */
11. $done$ = **false** ;
12. } /* **for** */
13. **until** *done;*
14. **for** $(i = 1, \cdots, l)$ { /* *Update Early Times* $\cdots$ */
15. $d_i = T_c - T_{p_i}$; /* $\cdots$ *Early Departure* */
16. **for** $(j = 1, \cdots, l)$
17. $a_i = \min_j(T_c - T_{p_j} - E_{p_j p_i})$; /* $\cdots$ *Early Arrival* */
18. } /* **for** */

for an iterative process (lines 3–13) similar to, but not exactly the same as, that of Alg. 4.1. The two iterations differ in the following respects:

- The clock schedule found by solving program PXR is guaranteed to satisfy hold time requirements at all latches regardless of the actual values of minimum delays in the circuit. Thus the iteration in Alg. 5.1 need only be concerned with updating the arrival and departure times of long paths.

- The initial values of the late departure times in Alg. 5.1 (line 1) are guaranteed to be larger than or equal to their final values. Thus, unlike Alg. 4.1, successive updates of each D_i and A_i will be non-increasing, i.e. $D_i^m \leq D_i^{m-1}$ and $A_i^m \leq A_i^{m-1}$. This "left sliding", in turn, guarantees that the update process will not lead to setup violations, and no setup checks are necessary.

5.2 Minimum Cycle Time—The GSTC Case

The procedure of finding the minimum cycle time for the GSTC case is complicated by the presence of both min and max functions in the constraints. Thus, unlike the RSTC case, an approach similar to Algorithm 5.1 cannot be guaranteed to yield an optimal solution to program PG under all circumstances. In fact, as we will shortly see, applying such an approach may produce a solution that violates one or more hold time constraints. Fortunately, there is also enough additional information to indicate how this infeasible solution should be modified to make it both feasible and optimal.

Assume, therefore, that we carry out the following two steps:

1. Solve program PXG.
2. Disregarding the hold time constraints, "slide" the solution found in step (1) so that the synchronization and propagation constraints are satisfied.

The solution obtained by this procedure can, then, be interpreted as that of a linear program—call it PXG′—which is identical to program PXG except that the hold time at each synchronizer i is *reduced* by the corresponding violation amount V_i^H. Specifically, the hold time constraints, $a_i \geq H_i$, are replaced by:

$$a_i \geq H_i - V_i^H \quad i = 1, \cdots, l \tag{5}$$

Furthermore, to account for cases when $H_i < V_i^H$, allow the early arrival times, a_i, to be unrestricted in sign. Finally, let π_i denote the optimal value of the *dual* variable[18] corresponding to the ith modified hold time constraint in (5).

With these constructions, it is now possible to obtain the optimal solution of program PG by applying the sensitivity analysis techniques of linear programming [18, Sec. 4.4, p. 160] to the solution of program PXG′. In particular, program PG can be "obtained" from program PXG′ by *increasing* the hold time at each synchronizer i by V_i^H. Denoting the optimal values of programs PG and PXG by $T_{c,min}^{(G)}$ and $T_{c,min}^{(XG)}$, we may therefore conclude that:

Conjecture 5.2 $T_{c,min}^{(G)} = T_{c,min}^{(XG)} + \sum_{i=1}^{l} \pi_i V_i^H$.

This result is stated as a conjecture because we have not yet worked out the detailed proof steps. It assumes that the solution of program PXG′ remains basic feasible after the right-hand-side vector is modified.

The detailed algorithmic steps for computing the minimum cycle time in the GSTC case are given in Alg. 5.2.

Algorithm 5.2: Find Minimum Cycle Time for Program PG

1. *Solve PXG; Let* D_i^0 *and* d_i^0 *be the optimal values of the departure variables.*
2. $T_{c,min}^G = T_{c,min}^{XG}$;
3. $m = 0$; /* *Initialize Iteration Counter* */
4. **repeat** /* *Iterate* $\cdots$ */
5. $done$ = **true** ; /* *Reset Convergence Flag* */
6. $m = m + 1$; /* *Increment Iteration Counter* */
7. **for** $(i = 1, \cdots, l)$ { /* *Visit All Latches* */
8. **for** $(j = 1, \cdots, l)$ { /* *Update Arrival Times* */
9. $A_i^m = \max_j(D_j^{m-1} + \Delta_j + \Delta_{ji} - E_{p_j p_i})$; /* *Late Arrival* */
10. $a_i^m = \min_j(d_j^{m-1} + \delta_j + \delta_{ji} - E_{p_j p_i})$ /* *Early Arrival* */
11. } /* **for** */
12. **if** $(a_i^m \leq H_i)$ { /* *Check Hold Time* */
13. $V_i^H = H_i - a_i^m$; /* *Calc Hold Violation* */
14. $a_i^m = H_i$ /* *Clip Early Arrival Time* */
15. } /* **if** */
16. $D_i^m = \max(A_i^m, T_c - T_{p_i})$; /* *Update Late Departure Time* */
17. $d_i^m = \max(a_i^m, T_c - T_{p_i})$; /* *Update Early Departure Time* */
18. **if** $((D_i^m \neq D_i^{m-1})$ **or** $(d_i^m \neq d_i^{m-1}))$ /* *Check Convergence* */
19. $done$ = **false** ;
20. } /* **for** */
21. **until** *done;*
22. **for** $(i = 1, \cdots, l)$ /* *Calc min cycle time* */
23. $T_{c,min}^G = T_{c,min}^G + \pi_i V_i^H$;

---

6 Example

The timing model introduced in this report has been implemented in two experimental CAD tools: *check*T_c which examines a circuit for adherence to a specified clock schedule, and reports on setup and hold time violations; and *min*T_c which uses algorithms 5.1 and 5.2 to determine the optimal clock schedule. We are using these tools to help in the design of a 250 MHz galluim arsenide micro-supercomputer currently under development at the University of Michigan[19].

Figure 6 illustrates the use of $minT_c$ in the design procedure. At the top of the figure is a simplified block diagram of the micro-supercomputer's CPU and its primary cache subsystem. The CPU implements an existing instruction-set architecture, the MIPS R6000, and has as its main components, a register file of 32 32-bit registers, an ALU, a shifter, and an integer multiply and divide unit. The datapath is 32-bits wide and it is timed with a 3- phase clock. The diagram shows the major logic blocks in the CPU datapath which is implemented as a single chip. The shaded blocks show the instruction cache and the data cache which are implemented as a set of 15 highspeed GaAs 1 K x 32 SRAM chips. The synchronizing elements are a combination of latches and flip-flops (F/Fs, see Fig. 6). To reduce the effects of chip crossings the CPU and the primary caches are integrated into a single multi-chip module (MCM). To avoid slow inter-chip bus protocols, the MCM is timed as one logic entity using $minT_c$ to obtain the optimal clock schedule. To apply $minT_c$ the MCM was first macromodeled in the form of the "synchronous circuit model" shown in Fig. 4. In particular, the short- and long-path delays for the chip crossings were extracted using SPICE simulations.

The optimal clock schedule obtained by $minT_c$ can be seen below the block diagram. The time units are nanoseconds. To simplify the analysis performed by $minT_c$ each of 32-bit wide latches and flip-flops were modeled as a single synchronizer. The output of $minT_c$ also includes waveforms of the signals arriving at the inputs of each synchronizer in the MCM. Seven of these waveforms, corresponding to the labeled synchronizers in the blobk diagram, are shown below the clock waveforms.

7 Conclusions and Future Work

The timing model presented in this report establishes a framework for the study of the dynamic behavior of synchronous digital systems. The model requires that a circuit be decomposed into combinational and sequential parts, a generally difficult task. Furthermore, the dynamic behavior of the sequential structures must be captured in timing "macromodels" similar to those of the D-type latches and flip-flops presented in this report. Both of these steps, decomposition and macromodeling, require further investigation. Other extensions to the model, for example to handle clock skew, are relatively straightforward and are currently being incorporated.

In parallel with the model development effort, we are investigating efficient implementations

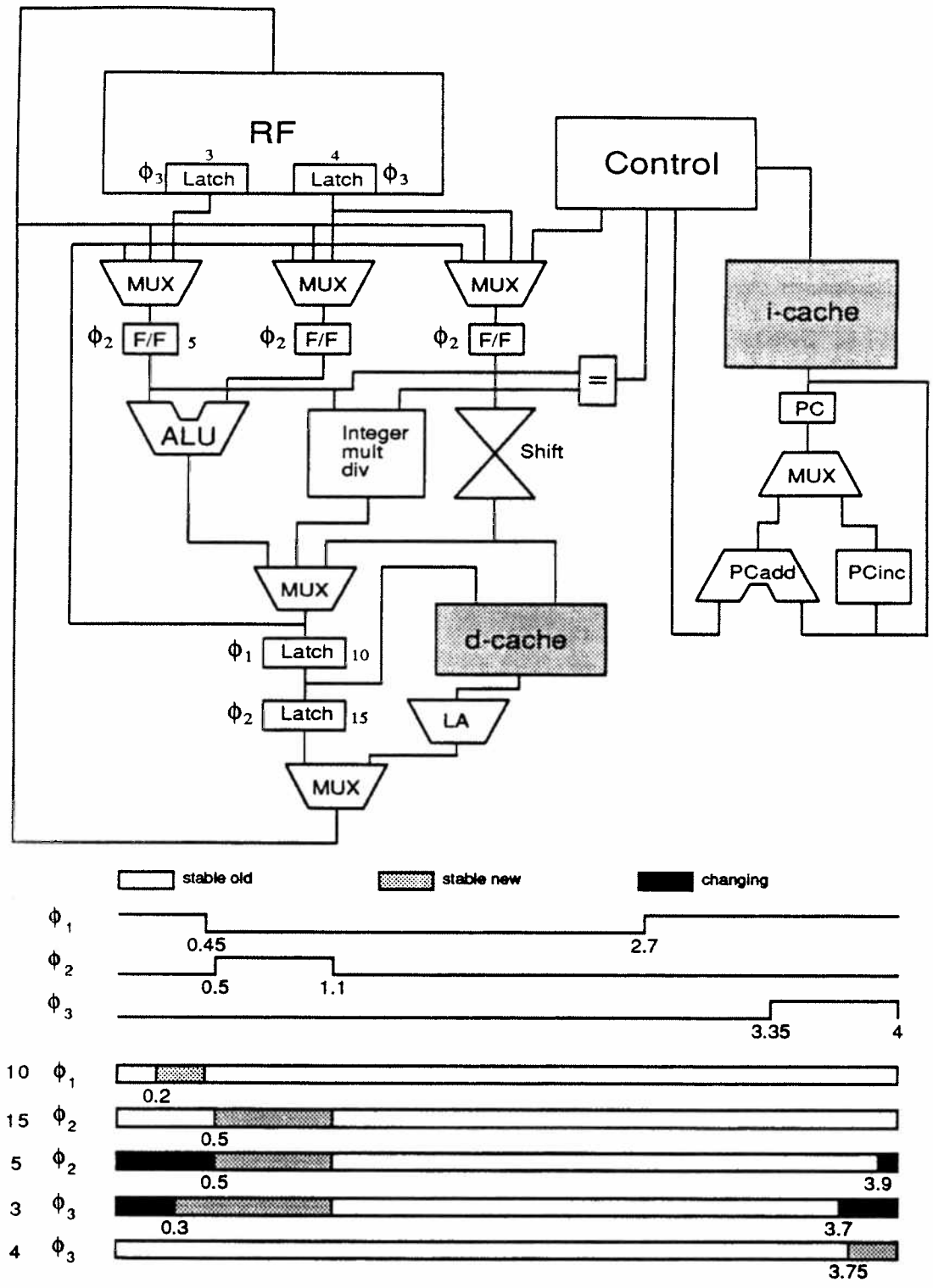

Figure 6: GaAs MIPS Datapath and Timing Waveforms

of algorithms 5.1 and 5.2. We are also looking into the derivation of closed-form design criteria for special circuit structures such as pipelines and CPU data paths.

References

[1] N. P. Jouppi, *Timing Verification and Performance Improvement of MOS VLSI Designs*, PhD thesis, Stanford University, Stanford, CA 94305-2192, October 1984.

[2] J. K. Ousterhout, "A Switch-Level Timing Verifier for Digital MOS VLSI," *IEEE Transactions on Computer-Aided Design*, vol. CAD-4, no. 3, pp. 336–349, July 1985.

[3] M. Glesner, J. Schuck, and R. B. Steck, "SCAT- A New Statistical Timing Verifier in a Silicon Compiler System," in *Proceedings of th 23rd Design Automation Conference*, pp. 220–226, 1986.

[4] S. H. Unger and C.-J. Tan, "Clocking Schemes for High-Speed Digital Systems," *IEEE Transactions on Computers*, vol. C-35, no. 10, pp. 880–895, October 1986.

[5] T. G. Szymanski, "LEADOUT : A Static Timing Analyzer for MOS Cicuits," in *ICCAD-86 Digest of Technical Papers*, pp. 130–133, 1986.

[6] J. J. Cherry, "Pearl : A CMOS Timing Analyzer," in *Proceedings of the 25th Design Automation Conference*, pp. 148–153, 1988.

[7] D. E. Wallace and C. H. Sequin, "ATV: An Abstract Timing Verifier," in *Proceedings of the 25th Design Automation Conference*, pp. 154–159, 1988.

[8] N. Ishiura, M. Takahashi, and S. Yajima, "Time-Symbolic Simulation for Accurate Timing Verification of Asynchronous Behavior of Logic Circuits," in *Proceedings of the 26th Design Automation Conference*, pp. 497–502, 1989.

[9] N. Weiner and A. Sangiovanni-Vincentelli, "Timing Analysis in A Logic Synthesis Environment," in *Proceeding of the 26th Design Automation Conference*, pp. 655–661, 1989.

[10] M. R. Dagenais and N. C. Rumin, "On the Calculation of Optimal Clocking Parameters in Synchronous Circuits with Level-Sensitive Latches," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 3, pp. 268–278, March 1989.

[11] T. I. Kirkpatrick and N. R. Clark, "PERT as an Aid to Logic Design," *IBM Journal of Research and Development*, vol. 10, no. 2, pp. 135–141, March 1966.

[12] B. K. Fawcett, *Maximal Clocking Rates for Pipelined Digital Systems*, Master's thesis, University of Illinois, 1975.

[13] T. M. Mcwilliams, "Verification of Timing Constraints on Large Digital Systems," in *Proceedings of the 17th Design Automation Conference*, pp. 139–147, 1980.

[14] V. D. Agrawal, "Synchronous Path Analysis in MOS Circuit Simulator," in *Proceedings of the 19th Design Automation Conference*, pp. 629–635, 1982.

[15] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "A Timing Model of Synchronous Digital Circuits ," Technical Report CSE-TR-47-90, University of Michigan, Dept of EECS, Ann Arbor, MI 48109-2122, 1990.

[16] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "Analysis and Design of Latch-Controlled Synchronous Digital Circuits ," in *Proceedings of the 27th Design Automation Conference*, 1990.

[17] L. A. Glasser and D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Addison Wesley, 1985.

[18] D. T. Phillips, A. Ravindran, and J. J. Solberg, *Operations Research: Principles and Practice*, John Wiley and Sons, Inc., 1976.

[19] R. B. Brown, J. A. Dykstra, T. N. Mudge, and R. Milano, "A GaAs Micro-Supercomputer: Rationale and Design," Technical Report CSE-TR-42-90, University of Michigan, Dept of EECS, Ann Arbor, MI 48109-2122, October 1990.