IEEE International Conference on Computer-Aided Design. November, 1990.

checkT$_c$ and *minT$_c$* : 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

Abstract

We introduce two 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 multiphase 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 in the design of a 4ns gallium arsenide micro-supercomputer.

1 Introduction

We present algorithms for timing verification and optimal clocking based on a new *timing model* of synchronous digital circuits. The new model extends our earlier work in [1] to handle short- as well as long-path propagation. It assumes that the circuits are logically (functionally) correct, and focuses only on capturing their latching, synchronization and propagation properties. The model 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 significantly more versatile than previous models. In particular, on the issue of level-sensitive latches which has received a great deal of attention recently (e.g. [2]–[6]), the model offers a general and accurate, yet simple, treatment. Briefly, the salient features of the new model are:

- 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 paper we only consider timing macromodels for D-type level-sensitive latches.
- 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.

The model applies to a synchronous sequential circuit containing l synchronizers (latches or flip-flops) controlled by a k-phase clock. Space limitations preclude a detailed development of the model here. Instead, a summary of its parameters, variables, and constraints is given in Table 1, and interested readers are referred to [7, 8] for a more comprehensive treatment. We devote the remainder of this paper to a discussion of the model's application to timing verification (Sec. 2) and optimal clocking (Sec. 3) and its use in the design of a micro-supercomputer (Sec. 4).

2 Timing Verification

The 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, and then checking if this solution satisfies the latching constraints. 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 latches 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 latch i, Alg. 2.1 incorporates the above approach and forms the basis of the timing verification procedure

Parameters	
p_i :	clock phase controlling latch i
S_i :	setup time of latch i
H_i :	hold time of latch i
δ_i :	min delay of latch i
Δ_i :	max delay of latch i
δ_{ij} :	min delay from latch i to latch j
Δ_{ij} :	max delay from latch i to latch j

Variables	
T_c :	cycle time
T_{p_i} :	width of active interval of phase p_i
$E_{p_jp_i}$:	forward phase shift from phase p_i to phase p_j
a_i :	earliest signal arrival time at latch i
A_i :	latest signal arrival time at latch i
d_i :	earliest signal departure time from latch i
D_i :	latest signal departure time from latch i

General System Timing Constraints (GSTC)	
Latching:	$a_i \geq H_i$
	$A_i \leq T_c - S_i$
Synchronization:	$d_i = \max(a_i, T_c - T_{p_i})$
	$D_i = \max(A_i, T_c - T_{p_i})$
Propagation:	$a_i = \min_j(d_j + \delta_j + \delta_{ji} - E_{p_jp_i})$
	$A_i = \max_j(D_j + \Delta_j + \Delta_{ji} - E_{p_jp_i})$

Table 1: Timing Model

Algorithm 2.1: Timing Verification in *checkT*$_c$ *1*

```
1.  for ($i = 1, \cdots, l$) {
2.      $V_i^S = V_i^H = 0$ ;
3.      $D_i^0 = T_c - T_{p_i}$ ;
4.      $d_i^0 = T_c - S_i$ ;
5.  } /* for */
6.  $m = 0$;
7.  repeat
8.      $done$ = true ;
9.      $m = m + 1$ ;
10.     for ($i = 1, \cdots, l$) {
11.         for ($j = 1, \cdots, l$) {
12.             $A_i^m = \max_j(D_j^{m-1} + \Delta_j + \Delta_{ji} - E_{p_jp_i})$ ;
13.             $a_i^m = \min_j(d_j^{m-1} + \delta_j + \delta_{ji} - E_{p_jp_i})$
14.         } /* for */
15.         if ($A_i^m \geq T_c - S_i$) {
16.             $V_i^S = A_i^m - T_c + S_i$ ;
17.             $A_i^m = T_c - S_i$
18.         }
19.         if ($a_i^m \leq H_i$) {
20.             $V_i^H = H_i - a_i^m$ ;
21.             $a_i^m = H_i$
22.         } /* if */
23.         $D_i^m = \max(A_i^m, T_c - T_{p_i})$ ;
24.         $d_i^m = \max(a_i^m, T_c - T_{p_i})$ ;
25.         if (($D_i^m \neq D_i^{m-1}$) or ($d_i^m \neq d_i^{m-1}$))
26.             $done$ = false ;
27.     } /* for */
28. until $done$;
```

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 and A_i (d_i and a_i) during the iterative portion of the algorithm (lines 7–28) are guaranteed to be non-decreasing (non-increasing). 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 17) and minimum (line 21) 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, 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 12,13,23,24) 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. 2.1 to identify the *critical* path segments in the circuit. The long path from latch j to latch i is considered critical if $D_j + \Delta_j + \Delta_{ji} - E_{p_jp_i} = T_c - S_i$. Similarly, the short path from latch j to latch i is critical if $d_j + \delta_j + \delta_{ji} - E_{p_jp_i} = H_i$. Furthermore, the *slacks* in the other (non-critical) path segments can be easily found.

3 Calculation of Optimal Cycle Time

The minimum clock cycle time can be found by solving the following optimization problem:

Program PG: Optimal Cycle Time—GSTC
Minimize T_c
Subject to GSTC (Table 1)

Following the approach suggested in [1], we solve this problem by first transforming the nonlinear program PG to an associated linear program PXG by *relaxing* the min and max constraints. However, unlike the more restrictive case in [1], 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, and an approach similar to Algorithm MLP in [1] cannot be guaranteed to yield an optimal solution to program PG under all circumstances. In fact 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 latch i is *reduced* by the corresponding violation amount V_i^H. Specifically, the hold time constraints, $a_i \geq H_i$, are replaced by:

$$a_i \geq H_i - V_i^H \tag{1}$$

Finally, With these constructions, it is now possible to obtain the optimal solution of program PG by applying the sensitivity analysis techniques of linear programming [9, 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 latch 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 3.1 $T_{c,min}^{(G)} = T_{c,min}^{(XG)} + \sum_{i=1}^{l} \pi_i V_i^H$.
where π_i denotes the optimal value of the *dual* variable corresponding to the ith modified hold time constraint in (1). 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. 3.1.

Algorithm 3.1: Optimal Clocking in *$minT_c$ 1*

```
1.  Solve PXG; Denote solution by $D_i^0$ and $d_i^0$
2.  $T_{c,min}^{G} = T_{c,min}^{XG}$ ;
3.  $m = 0$;
4.  repeat
5.      $done$ = true ;
6.      $m = m + 1$ ;
7.      for ($i = 1, \cdots, l$) {
8.          for ($j = 1, \cdots, l$) {
9.              $A_i^m = \max_j(D_j^{m-1} + \Delta_j + \Delta_{ji} - E_{p_j p_i})$ ;
10.             $a_i^m = \min_j(d_j^{m-1} + \delta_j + \delta_{ji} - E_{p_j p_i})$
11.         } /* for */
12.         if ($a_i^m \leq H_i$) {
13.             $V_i^H = H_i - a_i^m$ ;
14.             $a_i^m = H_i$
15.         } /* if */
16.         $D_i^m = \max(A_i^m, T_c - T_{p_i})$ ;
17.         $d_i^m = \max(a_i^m, T_c - T_{p_i})$ ;
18.         if (($D_i^m \neq D_i^{m-1}$) or ($d_i^m \neq d_i^{m-1}$))
19.             $done$ = false ;
20.     } /* for */
21. until done;
22. for ($i = 1, \cdots, l$)
23.     $T_{c,min}^{G} = T_{c,min}^{G} + \pi_i V_i^H$ ;
```

4 Example

The timing model and algorithms introduced in this paper have been implemented in two experimental CAD tools: *checkT$_c$* which examines a circuit for adherence to a specified clock schedule, and reports on setup and hold time violations; and *minT$_c$* which determines 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[10]. Figure 1 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. 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 to obtain the short- and long-path delays for the chip crossings 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 latch. The output of *minT*$_c$ also includes waveforms of the signals arriving at the inputs of each latch in the MCM. Five of these waveforms, corresponding to the labeled latches in the block diagram, are shown below the clock waveforms.

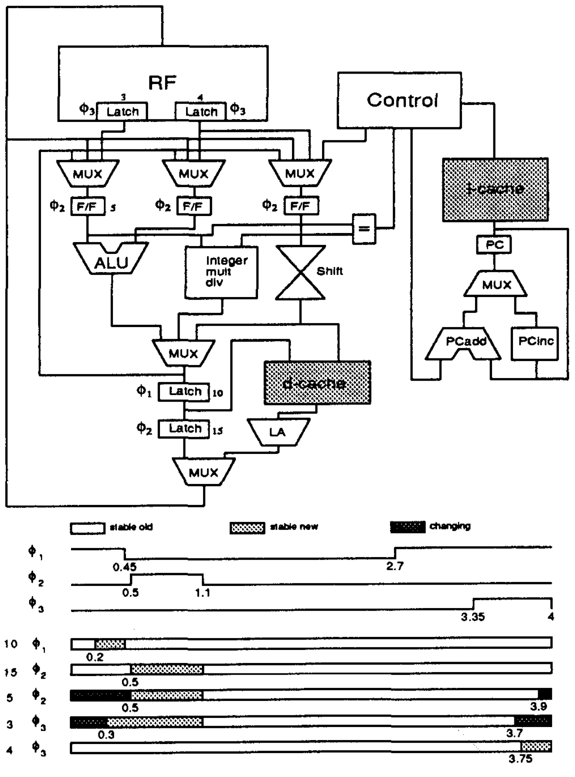

Figure 1: GaAs MIPS Datapath and Timing Waveforms

5 Conclusions and Future Work

The timing model presented in this paper 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 presented in this paper. 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 of algorithms 2.1 and 3.1. 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] 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.

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

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

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

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

[6] 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.

[7] 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.

[8] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "Optimal Clocking of Synchronous Systems," Technical Report CSE-TR-65-90, University of Michigan, Dept of EECS, Ann Arbor, MI 48109-2122, 1990.

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

[10] 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, 1990.