Exact Minimum Cycle Times for Finite State Machines

William K.C. Lam *
Hewlett-Packard Co. Palo Alto, CA

Robert K. Brayton Alberto L. Sangiovanni-Vincentelli
Department of EECS, University of California, Berkeley

Abstract

In current research, the minimum cycle times of finite state machines are estimated by computing the delays of the combinational logic in the finite state machines. Even though these methods deal with false paths, they ignore the sequential and periodic nature of minimum cycle times, and hence may give pessimistic results. In this paper, we first prove conditions under which combinational delays are correct upper bounds on minimum cycle times. Then, we present a sequential approach to compute the minimum cycle times of finite state machines, taking into account the effects of gate delay variations, reachable state space, initial states, unrealizable transitions, multiple cycle false paths, and periodicity of the present state vector sequences. We formulate and solve the problem exactly using Timed Boolean Functions, and give an efficient algorithm to solve for upper bounds of minimum cycle times. The exact formulation with Timed Boolean Functions provides a framework for further improvements on existing algorithms to compute the minimum cycle times. We implemented the algorithm and obtained the tightest bounds known on ISCAS benchmarks. From the experiments, we found that for about 20% of the circuits (not all shown in section 8), combinational delays, e.g. floating, viability, and transition delays, give pessimistic upper bounds for cycle times by as much as 25%.

1 Introduction

Accurately computing the minimum cycle time of a finite state machine is important because it not only directly affects the computation speed of the finite state machine but also aids in the synthesis of high speed sequential machines. All current approaches to this problem estimate the minimum cycle time of a finite state machine by estimating or computing the exact delay of the combinational logic implementing the next state function of the machine; hence, sequential properties pertaining to the machine and affecting its minimum cycle time are not taken into account. Some of these properties are reachable state space, initial states, unrealizable transitions, multiple cycle false paths, and periodicity of the present state vector sequences; without taking into account these factors, results on cycle times may be pessimistic. Here we consider this problem sequentially instead of combinationally. First, we formulate the problem exactly for sequential circuits with edge-triggered latches using Timed Boolean Functions (TBF's); then we provide an algorithm to compute the minimum cycle time. In our method, we include effects of gate delay variations, input waveforms, unreachable and initial states, multiple cycle false paths, and periodicity of input sequences.

2 Previous Approaches: Combinational Delays

Figure 3 shows a block diagram of a general finite state machine. The block labeled combinational logic computes the next state function of the finite state machine. In all previous approaches, the delay of the combinational logic is taken to be an upper bound of the minimum cycle time of the finite state machine. The delay of the combinational logic depends on the type of delay definition used in its computation; examples are single vector delay which includes viability delay and floating delay, and transition delay, [8, 2]. In single vector delay computation, an input vector is applied to a circuit, and the node values in the circuit are assumed conservatively to be arbitrary until the input vector has propagated through. The single vector delay is the latest time the output becomes steady under any input vector. Exact sensitization conditions were shown in [2] and efficient computational algorithms, in [4, 9, 6]. The delay by sequences of vectors, proposed in [6], for a circuit with gate delays variable within bounded intervals is the latest arrival time of the last output transition, when a settled circuit is applied with an arbitrary sequence of vectors with the last vector at $t = 0$. It is shown in [6] that single vector delay is equivalent to delay by sequences of vectors in any circuit with variable gate delays and that both single vector delay and delay by sequences of vectors are invariant under both bounded and unbounded gate delay models. Transition delay of a circuit is the latest arrival time of the last output transition when a pair of input vectors are applied to the circuit at $t = -\infty$ and $t = 0$, [2, 3]. Gate delays are assumed to vary within bounded intervals. [6] uses Timed Boolean Functions to compute the exact transition delays of circuits.

In single vector delay computation, a last vector is assumed, while in minimum cycle time, the input (present state vector) is a periodic sequence. Nevertheless, single vector delays are believed to give correct upper bounds for minimum cycle times, although there is no formal proof of this in the literature. Transition delays give correct upper bounds for the minimum cycle times only if the transition delay is greater than half of the topological delay of the combinational logic. This result was proven in [3] and also, independently, in [7].

3 Minimum Cycle Time is a Sequential Delay

Using delays of the combinational logic of finite state machines as cycle time upper bounds overlooks the sequential nature of cycle times. Some sequential properties of cycle times not considered in combinational delay computations are as follows. The input vector space is assumed to be the entire Boolean space. For instance, in single vector delay, the input vector can be an arbitrary vector, and in transition delay, the input vector pair can be any pair. However, in a finite state machine, the input vector to its combinational logic is its state vector, and this state vector is restricted to this machine's reachable space, which can be a proper subspace of the entire Boolean space. Similarly, if a state is not reachable from another state, then the input vector pair representing the two states is never realizable. A recent work [1] takes advantage of the reachable state space of finite state machines in eliminating long false paths over multiple cycles and reducing cycle times. Even though, minimum cycle times may be not derived from the knowledge of multiple cycle false paths. Another factor is that a single vector or a pair of vectors (e.g. in transition delay) are assumed to be the input to the combinational logic of a finite state machine while the true inputs are sequences of vectors arriving at periodic time intervals.

It is conceivable that the upper bounds by single vector delays and transition delays may be improved by considering only the vectors in the reachable state space and the realizable transitions; however, single vector delays are inherently conservative and transition delays are limited by being greater than half of the longest topological delays. The formulation in this paper gives the exact minimum cycle times: including all sequential don't cares, bounded gate delays, false paths,

* Supported by Fannie and John Hertz Foundation and SRC under contract 93-DC-008, whose supports are gratefully acknowledged.

reachable state space, etc.

To include these sequential factors in computing minimum cycle times, we use Timed Boolean Functions to formulate the problem exactly and derive algorithms thereby.

DEFINITION 1 *Timed Boolean Functions*

1. *A* **binary signal space** $B(t)$ *is a collection of mappings* $f : R \mapsto B$, *where* R *is the set of real numbers and* $B = \{0, 1\}$.

2. *A* **Timed Boolean Function (TBF)** *is any function with domain* $B^n(t)$ *and range* $B(t)$. *For analysis on most digital circuits, the following subset of TBF's is sufficient.*

   TBF $F : B^n(t) \mapsto B(t)$ *satisfies the following properties.*

   - Identity. *The identity function* F *(i.e.* $F(v)(t) = v(t),\ v(t) \in B(t)$*) is a TBF.*
   - Closed under Boolean operations. *If* $G : B^{n_1}(t) \mapsto B(t)$ *and* $H : B^{n_2}(t) \mapsto B(t)$ *are TBF's, then,* $\overline{G}, G \cdot H, G + H$ *are also TBF's.*
   - Closed under argument transformation. *If* $F(t)$ *is a TBF, then, for any function* $\phi : R^n \mapsto R$, $F(\phi)$ *is also a TBF.*
   - Closed under composition. *If*

     $$G = G(\ldots, x_i(g_{i1}), \ldots, x_i(g_{im}), \ldots), i = 1, \ldots, n_1,$$

     and $H = H(t)$ *are TBF's, where* $G : B^{n_1}(t) \mapsto B(t)$, $H : B^{n_2}(t) \mapsto B(t)$, *and* $g_{ij} : R^n \mapsto R$, *then*

     $$G \circ H = G(\ldots, H(g_{i1}), \ldots, H(g_{im}), \ldots)$$

     is also a TBF.

3.1 Modeling Timing Behavior with TBF

Before representing a circuit by a TBF, each component of the circuit needs to be modeled by a TBF. Here, we only illustrate through examples the modeling process for some commonly encountered gates.

1. *Gates whose delays are characterized by a single delay for each input-output pair.* The complex gate shown in Figure 1(a) has three inputs; input x_i has a delay τ_i to the output. This gate is modeled with the TBF:

   $$y(t) = \overline{x}_1(t - \tau_1) + x_2(t - \tau_2) + x_3(t - \tau_3).$$

2. *Buffers with different rising and falling delays.* Let τ_r and τ_f be the rising and falling delays, respectively. If $\tau_r > \tau_f$, then the buffer can be modeled as:

   $$y(t) = x(t - \tau_r) \cdot x(t - \tau_f).$$

   and if $\tau_r < \tau_f$, the buffer can be modeled as:

   $$y(t) = x(t - \tau_r) + x(t - \tau_f).$$

3. *Gates with different rising and falling delays for each input-output pair.* Rising (falling) delay is the delay when the output is rising (falling). Each input is modeled by a buffer with different rising and falling delays; and the "functional block" assumes zero delay. The overall TBF for the gate is obtained through the usual functional composition. An example of an OR gate is shown in Figure 1(b). Input 1 has a rising delay of 1 and a falling delay of 2, while input 2 has a rising delay of 4 and a falling delay of 3. The buffer modeling input 1 is $x_1(t-1) + x_1(t-2)$ and input 2 is $x_2(t-4) \cdot x_2(t-3)$. Therefore, the OR gate is

   $$x_1(t-1) + x_1(t-2) + x_2(t-4) \cdot x_2(t-3).$$

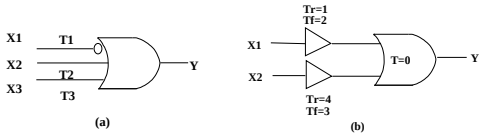

Figure 1: Modeling With TBF

4. *Edge triggered D-flipflop with a common clock of period* P. Let Q, D, d be the output, the data input, and the delay of the flip flop, respectively; then the flipflop is represented by

   $$Q(t) = D\left(P \cdot \left\lfloor \frac{t-d}{P} \right\rfloor\right)$$

   where $\lfloor x \rfloor$ = the greatest integer not exceeding x.

Note that memory elements (e.g. edge triggered D-FF) are represented without feedback; its memory effect is captured by the greatest integer function $\lfloor x \rfloor$. Being able to characterize memory element enables TBF's to represent sequential circuits with complete timing information.

3.2 Synchronous Circuit Formulation With Timed Boolean Function

Once all components of a circuit are represented by TBF's, the TBF for the circuit can be derived by identifying the timed variables corresponding to the ports connected to the same net. For synchronous sequential circuit, the combinational part of the circuit is first formulated with TBF's, then composed with the TBF's for the memory elements to obtain the TBF representation for the entire synchronous sequential circuit. We illustrate this with an example.

Example 1 *In Figure 2, the delay for each gate is shown inside the gate.*

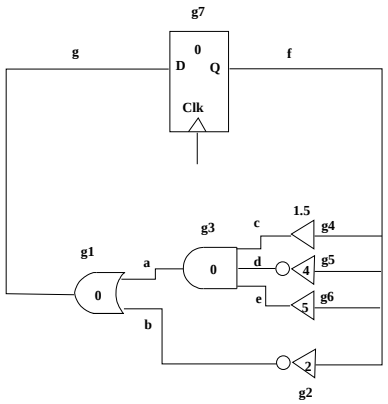

Figure 2: A Synchronous Sequential Circuit

First, we formulate the combinational part of the circuit with TBF's. Each gate is represented by a TBF, as follows.

$$\begin{aligned} g(t) &= \underline{a}(t) + b(t) \\ b(t) &= f(t-2) \\ a(t) &= c(t)d(t)e(t) \\ c(t) &= \underline{f}(t-1.5) \\ d(t) &= f(t-4) \\ e(t) &= f(t-5) \end{aligned}$$

We can also flatten above equations to a two level representation, as follows.

$$\begin{aligned} a(t) &= \underline{f}(t-1.5)\overline{f}(t-4)f(t-5) \\ b(t) &= f(t-2) \end{aligned}$$

Therefore,

$$g(t) = f(t-1.5)\bar{f}(t-4)f(t-5) + \bar{f}(t-2)$$

The TBF for the D flip flop is

$$f(t) = g(\lfloor \frac{t}{\tau} \rfloor \tau)$$

where τ is the cycle time of the synchronous sequential circuit.

Now, compose the two set of TBF's to obtain:

$$g(t) = g(\lfloor \frac{t-1.5}{\tau} \rfloor \tau)\bar{g}(\lfloor \frac{t-4}{\tau} \rfloor \tau)g(\lfloor \frac{t-5}{\tau} \rfloor \tau) + \bar{g}(\lfloor \frac{t-2}{\tau} \rfloor \tau) \quad (1)$$

This equation represents the complete functionality and timing information of the synchronous sequential circuit shown in Figure 2.

Comments:

1. For combinational circuits, when each circuit component is represented by a TBF having time argument of the form $t - h_i$, h_i is a constant, then the TBF for the circuit has only the time arguments of the form $t - h_i$.

2. The TBF's for synchronous sequential circuits with cycle time τ can be derived systematically as described below. Assume that all external inputs to the circuit are synchronized to the clock as shown in Figure 3. The TBF's for the combinational logic have the general form:

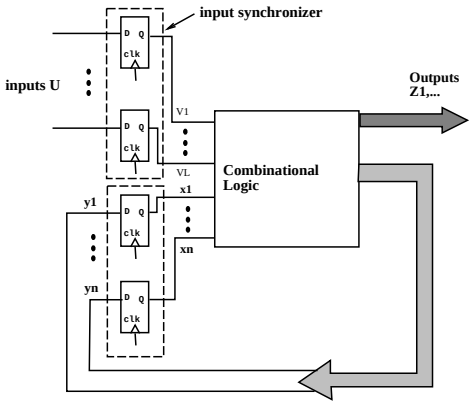

Figure 3: Block Diagram of a Synchronous Sequential Circuit

$$y_i(t) = f_i(x_1(t-h_{i1}), ..., x_n(t-h_{in}), v_1(t-h_{r_{i1}}), ..., v_l(t-h_{r_{il}})) \quad (2)$$

We can treat v_i's as states of the circuit, hence,

$$y_i(t) = f_i(x_1(t-h_{il}), ..., x_s(t-h_{is})),$$

where h_{ij} is the delay from ith flip flop's output to the jth flip flop's data input. Incorporating the flip flops' TBF's, we obtain:

$$y_i(t) = f_i(y_1(\lfloor \frac{t-h_{i1}-d_{f_1}}{\tau} \rfloor \tau), ..., y_s(\lfloor \frac{t-h_{is}-d_{f_s}}{\tau} \rfloor \tau),$$

$$y_i(t) = f_i(y_1(\lfloor \frac{t-k_{i1}}{\tau} \rfloor \tau), ..., y_s(\lfloor \frac{t-k_{is}}{\tau} \rfloor \tau), \quad (3)$$

where d_{f_i} is the delay of the ith flip flop. $y_{n+1}, ..., y_s$ are the external inputs. Therefore, $k_{ij} = h_{ij} + d_{f_j}$ is the delay around the loop from the jth flip flop's input to the ith flip flop's input. Therefore,

$$y_i(n\tau) = f_i(y_1(n\tau + \lfloor \frac{-k_{i1}}{\tau} \rfloor \tau), ..., y_s(n\tau + \lfloor \frac{-k_{is}}{\tau} \rfloor \tau)).$$

Normalizing to τ, the nth value of $y_i(t)$ is:

$$y_i(n) = f_i(y_1(n + \lfloor \frac{-k_{i1}}{\tau} \rfloor), ..., y_s(n + \lfloor \frac{-k_{is}}{\tau} \rfloor)).$$

4 Definition of Minimum Cycle Time

We want to define the minimum cycle time of a finite state machine in terms of the machine's I/O behaviors, instead of its combinational delay. If D_s is the minimum cycle time of a finite state machine, then we require that the finite state machine operate correctly at any clock period greater than or equal to D_s. Thus,

DEFINITION 2 *1. If $y(n, \tau)$ is a TBF for the outputs of a synchronous sequential circuit, where τ is a cycle time, the* **minimum cycle time** *of the circuit is the minimum D_s such that*

$$y(n, \tau) = y(n, D_s)\ \forall \tau > D_s,\ \forall n$$

2. Let L be the maximum value of the time constant in $y(n, \tau)$. Obviously,

$$y(n, \tau) = y(n, L)\ \forall \tau > L,\ \forall n$$

We call $y(n, L)$ the **steady state TBF** *of the machine.*

Example 2 *Assume that the gates' delays in Figure 2 are fixed constants as indicated and that the output is f, the output of the latch. We want to compute the circuit's minimum cycle time.*

First, we derive the circuit's TBF. The TBF for the latch with delay 0 is: $f(t) = g(\lfloor \frac{t}{\tau} \rfloor \tau)$. The TBF for the circuit is obtained by composing TBF's for each component,

$$g(t) = g(\lfloor \frac{t-1.5}{\tau} \rfloor \tau)\bar{g}(\lfloor \frac{t-4}{\tau} \rfloor \tau)g(\lfloor \frac{t-5}{\tau} \rfloor \tau) + \bar{g}(\lfloor \frac{t-2}{\tau} \rfloor \tau).$$

To find the minimum clock period, we need to look at $g(t)$ at $n\tau$. Substituting $t = n\tau$ and normalize to τ, we get:

$$g(n) = g(n + \lfloor \frac{-1.5}{\tau} \rfloor)\bar{g}(n + \lfloor \frac{-4}{\tau} \rfloor)g(n + \lfloor \frac{-5}{\tau} \rfloor) + \bar{g}(n + \lfloor \frac{-2}{\tau} \rfloor)$$

which will be denoted by $g(n, \tau)$. To find the minimum clock period, we decrease τ from ∞ until $g(n, \tau) \neq g(n, \infty)$. From the equation, we only need to examine the values of τ at which the argument of some term(s) in $g(n, \tau)$ changes value, e.g. $n + \lfloor \frac{-1.5}{\tau} \rfloor$. The first few τ's need to be examined are 4, 2.5, 2, $\frac{5}{3}$, ... At $\tau = 4$, $g(n, 4) = g(n-1)\bar{g}(n-1)g(n-2) + \bar{g}(n-1) = \bar{g}(n-1)$, which is equal to $g(n, \infty)$. At $\tau = 2.5$, $g(n, 2.5) = g(n-1)\bar{g}(n-2)g(n-2) + \bar{g}(n-1) = g(n, \infty)$. At $\tau = 2$, $g(n, 2) = g(n-1)\bar{g}(n-2)g(n-3) + \bar{g}(n-1) \neq g(n, \infty)$. Therefore, the minimum clock period is 2.5.

It is interesting to calculate the single vector delay and the 2-vector delay of the combinational logic in the circuit and compare them with the minimum cycle time. The single vector delay is 4, and the 2-vector delay is 2. The 2-vector delay gives an overly optimistic and incorrect(!) *upper bound of the minimum cycle time, while the single vector delay gives a* pessimistic, *but correct upper bound. In single vector delay computation, node d and e are assumed to take arbitrary values, hence the single vector delay of 4. In fact, due to the periodic nature of signal at f, the values at node d and e* can not *be arbitrary; therefore, the single vector delay gives a pessimistic upper bound. For the 2-vector delay, since the length of the longest path is 5, there are more than 2 data propagating along that path when the circuit*

operates at a clock period less than 2.5. *Because only two vectors are assumed in 2-vector delay, it is an incorrect upper bound.*

It is interesting to note that the state space is complete; there is only one bit, i.e. two states 0 *and* 1, *and the combinational logic is an inverter, which visits the entire state space. Thus, incorporating reachable state space in computing the combinational delays will not improve the upper bounds.*

5 Combinational Delays and Cycle Times

From example 2, we see that the 2-vector delay gives an incorrect upper bound of cycle time while the single vector delay gives a conservative upper bound. Here we want to examine the relationship between combinational delays and minimum cycle times. A theorem in [6] says that single vector delay is the same as delay by sequences of vectors for most practical circuits. Hence, we will study only the following two types of combinational delays: delay by sequences of vectors and 2-vector delay.

Will the single vector delays, or the delays by sequences of vectors, of the combinational logic in finite state machines always give conservative upper bounds for the minimum cycle times? Some of the concerns are as follows. In delay by sequences of vectors, a last input vector is assumed, while in the setting of finite state machines the inputs are periodic, having no last vector. Then, will this violation of the last vector assumption cause a problem? Further, from example 2, the single vector delay, or the delay by sequences of vectors, is 4 and the longest topological delay is 5; thus, even after the output of the combinational logic has become stable after 4 units of time, there are signals still propagating along the long false path of length 5. If the finite state machine is clocked at a period of 4, will the signals from the next clock period propagate along *short paths* to interact with the present signal still propagating along the long false path and the cause the long false path to become true? This is the short path problem. The following theorem provides a condition under which single vector delays, or delays by sequences of vectors are valid upper bounds of the minimum cycle times. For simplicity, the following discussion assumes that all latches have zero delay.

THEOREM 1 *Let edge-triggered latches have setup and hold times τ_s and τ_h, D^{max}, the maximum delay by sequences of vectors of the combinational logic of a finite state machine, and L^{min}, the length of the shortest path in the combinational logic. Then $D^{max} + \tau_s$ is a correct, by may be conservative, upper bound for the minimum cycle time if $L^{min} \geq \tau_h$.*

Note that if $L^{min} < \tau_h$, single vector delay and delay by sequence of vector of a finite state machine alone will not guarantee correct operation of of the finite state machine.

Example 2 shows how 2-vector delays can give incorrect cycle time upper bounds. A condition under which 2-vector delays always give correct upper bounds is provided in [4] and [7], which is restated as follows.

THEOREM 2 *If the 2-vector delay of the combinational logic of a finite state machine is greater than or equal to half of the topological delay of the combinational logic, then the minimum cycle time of the machine is less than or equal to the 2-vector delay.*

In example 2, the 2-vector delay of 2 is less than half of the topological delay 5, therefore it is not guaranteed by Theorem 2 to give a correct cycle time upper bound. In this case, it gives an incorrect one.

6 Computing Minimum Cycle Times

Using TBF's, the exact minimum cycle time of a finite state machine is the solution of the following mixed Boolean linear program:

$$\begin{array}{rcl} D_s & = & \max \tau \\ y(n,\tau) & \neq & y(n,L) \\ d_i^{min} \leq & d_i & \leq d_i^{max} \end{array}$$

We call this program a mixed Boolean linear program because it involves deciding equality of two Boolean functions, i.e. $y(n,\tau) \neq y(n,L)$ and computing the linear programming problems induced by the linear inequalities.

In general, a finite state machine is characterized by an output TBF and a state TBF, i.e.

$$\begin{array}{rcl} y(n,\tau) & = & f(\ldots, x(n + \lfloor \frac{-k_i}{\tau} \rfloor), \ldots, u(n + \left\lfloor \frac{-k_j}{\tau} \right\rfloor), \ldots) \\ x(n,\tau) & = & g(\ldots, x(n + \lfloor \frac{-k_i}{\tau} \rfloor), \ldots, u(n + \left\lfloor \frac{-k_j}{\tau} \right\rfloor), \ldots) \end{array}$$

where $x(n + \lfloor \frac{-k_i}{\tau} \rfloor)$ and $u(n + \lfloor \frac{-k_i}{\tau} \rfloor)$ are the state and input TBF variables. Usually, k_i's can vary within an interval, due to delay uncertainties in manufacturing. In this section, we assume k_i's are constants. Variable k_i's are considered in section 7.

To find the minimum cycle time , τ is decreased from L; and at each τ, $y(n,\tau)$ is compared with the steady state TBF $y(n,L)$. The minimum cycle time is the minimum τ such that for some n $y(n,\tau) \neq y(n,L)$. For a particular τ, we want to find a condition C for which we can check easily that C is satisfied if and only if $y(n,\tau) = y(n,L), \forall\ n$. Deciding this equality is equivalent to decide whether two finite state machines are equivalent and this can be done by reducing both machines to minimal machines and comparing them. An algorithm for reducing finite state machines to minimal machines can be found in [5]. However this explicit method takes too much memory space for most practical circuits. Therefore, we present a relatively simple sufficient condition which assume all state variables are observable.

DEFINITION 3 *A* **state sufficient** *condition C_x is:*

1. $x(n,\tau) = x(n,L), \forall\ n$.
2. $y(n,\tau) = y(n,L), \forall\ n$.

By including $x(n,\tau) = x(n,L), \forall\ n$, condition C_x becomes a sufficient condition only; so the minimum cycle time computed by checking whether C_x is satisfied is not the exact minimum cycle time for a finite state machine such that $y(n,\tau) = y(n,L), \forall\ n$, but for some n $x(n,\tau) \neq x(n,L)$, i.e. $x(n,\tau)$ and $x(n,L)$ are equivalent with respect to outputs. Yet, checking $x(n,\tau) = x(n,L), \forall\ n$ is very simple and once $x(n,\tau) = x(n,L), \forall\ n$ is assured $y(n,\tau) = y(n,L), \forall\ n$ is easily checked.

6.1 Checking State Sufficient Condition C_x

For a given τ, the TBF's can be written as:

$$\begin{array}{l} y(n) = f(\ldots, x(n - m_i), \ldots, u(n - m_j), \ldots) \\ x(n) = g(\ldots, x(n - m_i), \ldots, u(n - m_j), \ldots) \end{array}$$

We rewrite the steady state TBF's as:

$$\begin{array}{l} \hat{y}(n) = f(\ldots, \hat{x}(n-1), \ldots, u(n-1), \ldots) \\ \hat{x}(n) = g(\ldots, \hat{x}(n-1), \ldots, u(n-1), \ldots) \end{array}$$

Let m be the maximum of the m_i's. Consider checking $x(n) = \hat{x}(n), \forall\ n$. We start by comparing BDD's of $x(n)$ and $\hat{x}(n)$ for $1 \leq n \leq m$. For $1 \leq n \leq m$, some $x(n - m_i)$'s have values equal to the initial values; thus, we need to check for each such n. This is the basis step in a mathematical induction. If all the BDD pairs are equal, we proceed with the intermediate step in the induction. If the equality is true, we replace $x(n - m_i)$ in $x(n)$ by $\hat{x}(n - m_i)$, and make all arguments of x's in both $x(n)$ and $\hat{x}(n)$ equal by iteratively substituting $\hat{x}(n) = g(\ldots, \hat{x}(n-1), \ldots, u(n-1), \ldots)$ until all arguments are equal to $n - m$. Now both $x(n)$ and $\hat{x}(n)$ have the same $\hat{x}$ TBF variables, and we construct their BDD's. Their BDD's are equal if and only if $x(n) = \hat{x}(n), \forall\ n$.

Once $x(n) = \hat{x}(n)$ is assured, checking $y(n) = \hat{y}(n)$ is done in the same way. We start by checking $y(n) = \hat{y}(n)$ for $1 \leq n \leq m$. Then, replace $x(n - m_i)$ in $y(n)$ by $\hat{x}(n - m_i)$, and make their arguments equal by iteratively substituting $\hat{x}(n) = g(\ldots, \hat{x}(n-1), \ldots, u(n-1), \ldots)$. Then BDD's of $y(n)$ and $\hat{y}(n)$ are compared.

In summary,

Decision Algorithm (6.1) on Condition C_x

Given a TBF at a particular τ

$$y(n) = f(\ldots, x(n - m_i), \ldots, u(n - m_j), \ldots)$$
$$x(n) = g(\ldots, x(n - m_i), \ldots, u(n - m_j), \ldots)$$

and a steady state TBF

$$\hat{y}(n) = f(\ldots, \hat{x}(n-1), \ldots, u(n-1), \ldots)$$
$$\hat{x}(n) = g(\ldots, \hat{x}(n-1), \ldots, u(n-1), \ldots)$$

Let m be the maximum of m_i's.

1. Deciding $x(n, \tau) = x(n, L), \forall\ n$.
   (a) Compare BDD's of $x(n)$ and $\hat{x}(n)$, for $1 \leq n \leq m$. If all BDD pairs are equal, continue. Otherwise, $x(n) \neq \hat{x}(n),\ \forall n$.
   (b) Replace $x(n - m_i)$ in $x(n)$ by $\hat{x}(n - m_i)$.
   (c) Make all arguments of $\hat{x}$'s in both $x(n)$ and $\hat{x}(n)$ equal by iteratively substituting $\hat{x}(n) = g(\ldots, \hat{x}(n-1), \ldots, u(n-1), \ldots)$.
   (d) If the BDD's for $x(n)$ and $\hat{x}(n)$ are equal, $x(n) = \hat{x}(n), \forall\ n$.
2. Deciding $y(n, \tau) = y(n, L), \forall\ n$.
   (a) Compare BDD's of $y(n)$ and $\hat{y}(n)$, for $1 \leq n \leq m$. If all BDD pairs are equal, continue. Otherwise, $y(n) \neq \hat{y}(n),\ \forall n$.
   (b) Replace $x(n - m_i)$ in $y(n)$ by $\hat{x}(n - m_i)$.
   (c) Make all arguments of $\hat{x}$'s in both $y(n)$ and $\hat{y}(n)$ equal by iteratively substituting $\hat{x}(n) = g(\ldots, \hat{x}(n-1), \ldots, u(n-1), \ldots)$.
   (d) If the BDD's for $y(n)$ and $\hat{y}(n)$ are equal, $y(n) = \hat{y}(n), \forall\ n$.

THEOREM 3 (Correctness) *If the decision algorithm (6.1) is affirmative then the state sufficient condition C_x is satisfied.*

Therefore, for a particular τ, if the decision algorithm (6.1) is affirmative, this τ is a valid clock period; otherwise, we conservatively assume this τ is invalid.

7 Variable Gate Delays and Interval Algebra

If k_i's are constants, we find the minimum cycle time by sweeping τ starting from $\tau = L$. At $\tau = L$, all terms $\left\lfloor \frac{-k_i}{\tau} \right\rfloor$'s are -1, and the TBF's are the steady state TBF's. Now we decrease τ until some term(s) $\left\lfloor \frac{-k_i}{\tau} \right\rfloor$ changes value. Then, we decide whether this τ is valid by resorting to the decision algorithm on C_x. An upper bound to the minimum cycle time is the minimum τ such that the decision algorithm fails. Effectively, we divide the τ-axis into intervals with points $\bigcup_i \{\frac{k_i}{n}, n = 1, \ldots\}$. Within each such interval, $\left\lfloor \frac{-k_i}{\tau} \right\rfloor$ is a constant. Thus, searching the entire τ-axis reduces to searching at a set of points, each of which is in one of those partitioning the τ-axis into intervals of constant $\left\lfloor \frac{-k}{\tau} \right\rfloor$'s.

In reality, k_i's vary within intervals due to delay variations in manufacturing. Analysis with variable k_i's is similar except it deals with intervals instead of numbers. First, we define some algebraic operations on intervals.

DEFINITION 4 *1. Denote $\left\lfloor \frac{-k_i}{\tau} \right\rfloor$ where $k_i \in [k_i^{min}, k_i^{max}]$ by $\left\lfloor \frac{-I_{k_i}}{\tau} \right\rfloor$ where $I_{k_i} = [k_i^{min}, k_i^{max}]$ is an interval.*

2.

$$\left\lfloor \frac{-I_{k_i}}{\tau} \right\rfloor = \{j : \left\lfloor \frac{-k_i^{max}}{\tau} \right\rfloor \leq j \leq \left\lfloor \frac{-k_i^{min}}{\tau} \right\rfloor\}$$

3. Define $\Phi(.)(\tau)$ to be the Cartesian product of $\left\lfloor \frac{-I_i}{\tau} \right\rfloor$ at τ, i.e.

$$\Phi(I_1, \ldots, I_n)(\tau) = \prod_i \left\lfloor \frac{-I_i}{\tau} \right\rfloor$$

4. $\sigma = (\sigma_1, \ldots, \sigma_n) \in \Phi(I_{k_1}, \ldots, I_{k_n})(\tau)$ is **feasible**, *if there is an τ such that*

$$\left\lfloor \frac{-k_i}{\tau} \right\rfloor = \sigma_i,\ k_i \in [k_i^{max}, k_i^{min}],\ i \in [1, n]$$

Equivalently, the following system of inequalities is satisfiable.

$$\begin{array}{ccc} \frac{-k_i}{\sigma_i} & \leq \tau \leq & \frac{-k_i}{\sigma_i + 1} \\ k_i^{min} & \leq k_i \leq & k_i^{max} \\ & & i = 1, \ldots, n \end{array}$$

Analogous to dividing the τ-axis into intervals such that $\left\lfloor \frac{-k}{\tau} \right\rfloor$ is constant within an interval, we divide the τ-axis into the coarsest intervals such that $\left\lfloor \frac{-I_{k_i}}{\tau} \right\rfloor$ remains the same within a such interval. To search, we sweep τ starting from $\tau = L$. In an interval, if $\left\lfloor \frac{-I_{k_i}}{\tau} \right\rfloor$ has more than one element, each element is a possible value of $\left\lfloor \frac{-k_i}{\tau} \right\rfloor$. Thus, for an interval to be a range of valid τ, the decision algorithm (6.1) on C_x must be affirmative for all possible combinations of $\Phi(I_{k_1}, \ldots)(\tau)$, unless the combinations that fail the decision algorithm are infeasible. When there are feasible combinations that fail the decision algorithm, an upper bound on the minimum cycle time is the maximum of the linear programmings induced by these combinations. Symbolically, let Ω be the set of combinations that fail the decision algorithm. Then, an upper bound for the minimum cycle time , $\bar{D}_s$, can be found in the following linear program.

$$\bar{D}_s = \max_{\sigma \in \Omega} \tau(\sigma)$$

$$\begin{array}{ccc} \tau(\sigma) & = & \max\ t \\ \left\lfloor \frac{-k_i}{\tau} \right\rfloor & = & \sigma_i \\ k_i^{min} \leq & k_i & \leq k_i^{max} \\ & & i = 1, \ldots, n \end{array}$$

Equivalently,

$$\bar{D}_s = \max_{\sigma \in \Omega} \tau(\sigma)$$

$$\begin{array}{ccc} \tau(\sigma) & = & \max\ t \\ \tau(-\sigma_i - 1) + \epsilon \leq & k_i & \leq \tau(-\sigma_i) \\ k_i^{min} \leq & k_i & \leq k_i^{max} \\ & & i = 1, \ldots, n \end{array}$$

where ϵ is an arbitrarily small positive number. But k_i is the sum of delays of the gates on the path associated with k_i, i.e. $k_i = \sum_i d_i$, where d_i is the ith gate's delay. Thus, the linear programming is:

$$\bar{D}_s = \max_{\sigma \in \Omega} \tau(\sigma)$$

$$\begin{array}{ccc} \tau(\sigma) & = & \max\ t \\ \tau(-\sigma_i - 1) + \sigma \leq & \sum_i d_i & \leq \tau(-\sigma_i) \\ d_i^{min} \leq & d_i & \leq d_i^{max} \\ & & i = 1, \ldots, n \end{array}$$

Dividing the τ-axis into the coarsest intervals such that $\lfloor \frac{-I}{\tau} \rfloor$ remains the same within any interval can be done by dividing the axis with the points $\{ \lfloor \frac{k^{min}}{n} \rfloor, n, i = 1, \ldots \} \bigcup \{ \lfloor \frac{k^{max}}{n} \rfloor, n, i = 1, \ldots \}$. A method to speed up the search for minimum cycle time is to record the combinations of $\Phi(.)(\tau)$, and skip the ones that have been considered.

8 Experimental Results

We implemented the above algorithm and ran all ISCAS benchmarks on a DECstation 5000. In these benchmarks, we assumed that the gate delays varied from 90% to 100% of their respective maxima. For circuits s526, s526n, s820, s832, s953, s15850, and s38584, their combinational delays, e.g. single vector and transition (2-vector), give pessimistic upper bounds of their minimum cycle times by as much as 25%, while for the rest, the upper bounds on minimum cycle times are equal to their single vector delays and transition delays. These circuits, e.g. s526 and etc, consist of about 20% of the benchmark suite. The fact that many benchmarks have equal bounds on the minimum cycle time and single vector delays and transition delays might have three implications. First, the ISCAS benchmarks we ran simply do not contain good samples of designs in this respect; second, most current designs do not make use of the sequential aspects of minimum cycle times; or third, single vector delays are tight upper bounds for minimum cycle times.

The following table gives selected benchmarks; those not given have equal topological delays, single vector delays, transition delays, and the bounds on minimum cycle time . In the following table, the second column shows the topological delays of the combinational circuits of the finite state machines, while the third and the fourth columns show the single vector delays [3, 9], and the respective CPU times in seconds. The fifth and the sixth columns show the exact transition delays and their respective CPU times [6]. The seventh and the last columns give the upper bounds on the minimum cycle times, and the corresponding CPU times.

Note that the single vector delays and the transition delays are exactly the same for all the circuits, including those whose minimum cycle times are shorter than their single vector delays. This implies that the tighter bounds on minimum cycle times result from sequential aspects, which can not be discovered by combinational delays like single vector and transition delays.

In the last circuit $s38584$, the minimum cycle time is less than a fourth of the topological delay; so multiple cycles of inputs exist on the long paths if the circuit operates at the minimum cycle time, and the periodic nature of inputs plays a role in this circuit, further contributing to the pessimism of combinational delays. In addition, a correct upper bound given by 2-vector delay computation can only be as tight as a half of the topological delay, i.e. 189.2, larger than the actual minimum cycle time by more than 200%.

Circuit	Top. D	Float	CPU	Trans.	CPU	MCT	CPU
s444	22.8	22.8	0.36	22.8	0.2	22.8	0.31
s526‡	22.5	22.5	0.26	22.5	0.15	18.4	10.5
s526n‡	23.4	23.4	0.26	23.4	0.14	18.8	8.03
s641§	42.7	42.5	1.97	42.5	1.44	42.5	1.53
s713§	44.5	43.4	1.49	43.4	10.73	43.4	12.23
s820‡	29.6	29.6	0.48	19.6	0.28	27.9	1.5
s832‡	29.1	29.1	0.51	29.1	0.3	28.8	1.49
s953‡	29.7	29.7	0.44	29.7	0.26	28.2	2.42
s1196§	37	35.8	0.94	35.8	0.54	35.8	0.87
s1238§	42.9	41	3.09	41	1.69	41	2.22
s1423	119.8	119.8	3.26	119.8	1.91	119.8	51.13
s1494	36.2	36.2	0.73	36.2	0.44	36.2	0.69
s5378§	42.4	42	11.71	42	7.64	42	50.62
s9234§	58.4	56.7	215.65	56.7	211.88	–†	–
s15850§‡	128.8	127.4	1047.02	127.4	814.79	127.2†	–
s35932	436.3	436.3	29.26	436.3	19.21	436.3	49.93
s38417	128.8	128.8	205.34	128.8	132.08	–†	–
s38584	378.4	–†	–	–†	–	82.0†	–

†: memory out; the last value is reported.

‡: single vector and transition delays give pessimistic upper bounds.
§: topological delays > single vector, transition delays.

9 Conclusions and Future Directions

In current research, the minimum cycle times of finite state machines are estimated by computing the delays of the combinational logic in the finite state machines. Many of these use sophisticated new methods for dealing with false paths. However, these methods ignore the sequential and periodic nature of minimum cycle times, and hence may give pessimistic results. First we proved conditions under which combinational delays are correct upper bounds on minimum cycle times. Then we presented a sequential approach to computing the minimum cycle times of finite state machines, taking into account the effects of gate delay variations, reachable state space, initial states, unrealizable transitions, multiple cycle false paths, and periodicity of the present state vector sequences. From the ISCAS benchmarks, we showed that in almost 20% of all ISCAS circuits, their combinational delays, e.g. floating (viability), and transition delays, give pessimistic upper bounds for cycle times by as much as 25%. The problem of computing the exact minimum cycle times of finite state machines was formulated and solved using Timed Boolean Functions; then, we gave an efficient algorithm to solve for upper bounds on minimum cycle times. The exact formulation with Timed Boolean Functions provides not only a framework for further improvements on existing algorithms on computing the minimum cycle times, but also opens the possibility of bringing these analysis techniques into the synthesis of high speed sequential circuits. Extensions to circuits with level-sensitive latches are another direction for the future.

References

[1] P. Ashar, S. Dey, and S. Malik. Exploiting multi-cycle paths in the performance optimization of sequential circuits. *IEEE/ACM International Conference on Computer Aided Design, '92*, Nov., 1992.

[2] H. C. Chen and D. H. Du. Path sensitization in critical path problem. *1990 ACM Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, 1990.

[3] S. Devadas, K. Keutzer, and S. Malik. Certified timing verification and transition delay of a logic circuit. *Proc. of the 29th Design Automation Conference*, June, 1992.

[4] S. Devadas, K. Keutzer, and S. Malik. Delay computation in combinational logic circuits: Theory and algorithms. *IEEE/ACM International Conference on Computer-Aided Design*, Nov. 1991.

[5] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata, Languages and Computation*. Addison-Wesley, 1979.

[6] W. Lam, R. Brayton, and A. Sangiovanni-Vincentelli. Circuit delay models and their exact computation using timed boolean functions. *IEEE/ACM Design Automation Conference '93*, 1993.

[7] W. Lam, R. Brayton, and A. Sangiovanni-Vincentelli. Minimum cycle time of synchronous circuit with bounded delays. *UC Berkeley ERL memorandum: UCB/ERL M92/56*, May 1992.

[8] P. McGeer and R. Brayton. Provably correct critical paths. *The Proceedings of the Decennial Caltech VLSI Conference*, 1989.

[9] P. McGeer, A. Saldanha, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. Timing analysis and delay-fault test generation using path recursive functions. *IEEE International Conference on Computer-Aided Design*, pages 180–183, Nov. 1991.