# Automated Multi-Cycle Symbolic Timing Verification of Microprocessor-based Designs*

Anurag P. Gupta, Daniel P. Siewiorek
ECE Department, Carnegie Mellon University, Pittsburgh, PA 15213

*Abstract* – **Timing verification ascertains whether timing checks on components in a circuit are satisfied given component delay models. This paper addresses timing verification of microprocessor-based designs for which previous approaches are shown to be inadequate. It introduces the concept of sequential path tracing – tracing paths through both space *and time* – that forms the basis of the MTV tool. MTV has the following novel features:**

- **unlike previous approaches, it considers *sequential* behavior *together* with timing and handles sequential sensitizability and multi-cycle paths automatically;**
- **it does not require a predefined clock schedule and can handle circuits with conditional or gated clocks, multiple unrelated clocks, asynchronous set/reset, and power-up initialization;**
- **it generates *symbolic* constraints between timing attributes of components that can be efficiently re-used for small circuit changes or by a synthesis/optimization tool; symbolic constraints also enable common ambiguity removal.**

**Experimental results demonstrate that MTV takes only a few CPU minutes to generate symbolic constraints for each of several microprocessor-based designs.**

## I. Introduction

A circuit is an interconnection network of components. Each component responds to changes in values at its inputs by placing appropriate values at its outputs after some timing delay. Also, each component may impose certain restrictions on the ordering of changes in value at its inputs for correct operation. For example, a D flip flop requires that data be setup a certain time before and held a certain time after the clocking signal. Given models of the delays through all components, a timing verifier ascertains whether these restrictions or *timing checks* are satisfied for all components in the circuit.

This paper concentrates on timing verification of *microprocessor-based designs*. Example circuits are microprocessors connected to memory chips, other peripherals, and bus interfaces, including glue logic. Functionality and delay information for these components is limited to that available in databooks, which support the use of bus-interface modeling and the min-max delay models. There is a growing interest in designs with these off-the-shelf components with the increased emphasis on reducing design cost and time to market. However, previous timing verification approaches are not adequate since these designs have several multi-cycle paths (i.e. paths that are allowed more than one clock cycle to propagate), asynchronous set/clear, and lack separate clock signals with a predefined schedule (e.g. relationship between multiple clock phase signals).

Previous approaches and their limitations are discussed in Section II. Sequential path tracing is introduced in Section III as a novel approach for addressing these limitations. Since sequential paths are paths through both space *and time*, they handle multi-cycle paths automatically. Sequential path tracing has been implemented in MTV, a Multi-cycle Timing Verification tool. MTV includes two major subtasks: state transition graph (STG) generation sketched briefly in Section IV and constraint generation detailed in Section V. Another unique feature of MTV is that instead of reasoning with numeric values and indicating whether a timing check is satisfied or not, it uses *symbolic* delay values and generates constraints that must be satisfied by the symbolic values. The main disadvantage of an explicit STG is its potentially exponential size; however MTV is computationally practical since the worst case does not happen for microprocessor-based designs[1] as demonstrated by experimental results in Section VI.

*This work has been supported by the Engineering Design Research Center, an NSF Engineering Research Center, and the Computer-Aided Design Center.

## II. Previous Approaches

### A. Classification

Previous timing verification approaches are broadly classified into static or dynamic depending on whether they use a value-independent approach or not. The classification in Figure 1 is more detailed and is based on two distinct axes – whether input test vectors are needed or not, and whether the behavior of components is considered or not. Traditional static approaches are towards the bottom-left and dynamic approaches are towards the top-right of the table.

- **Simulation-based approaches** have the advantage of modeling signal interactions and logic behavior accurately but also the disadvantages of incompleteness, requiring the user or tool to generate test patterns, and higher computation cost. One reason for greater cost is the need for repeating the simulation with different delay values (since worst case delays may not be the maximum values). Some approaches mitigate the problem by reusing results in repeated fixed delay simulations [4], using ambiguity delay simulation [14, 8], or a combination of fixed delay simulation and path tracing [5].
- In contrast, delay analysis or **path tracing** approaches [10] ensure that the length of the longest (shortest) path in the combinational network between two synchronizing elements (i.e. flip flops or latches) is less (more) than some known relationship between the corresponding clocking signals at the synchronizing elements.
- Subsequently, approaches that considered combinational behavior to determine **static sensitizability** of paths and avoid reporting false paths were developed [3].
- Recently, approaches that consider combinational behavior and timing together to determine dynamic or **true sensitizability** of paths have been developed [6, 7].
- **Causality graph**-based approaches [13] are similar to path tracing except that they trace paths in a causality graph of events in the circuit instead of the combinational circuit structure.

These approaches (except simulation) concentrated on combinational behavior and are referred to as **Combinational Path Tracing (CPT)**. Since they did not reason with the *sequential* behavior, they could be overly pessimistic (as illustrated in the next section). Recent approaches that do consider sequential behavior are listed below:

- Path tracing has been combined with state transition graph **(STG) analysis** [12] for a specific class of designs – CPUs partitioned

[1]In contrast, STGs of gate or register transfer level circuits are too large to be represented explicitly and/or take too much time to be built.

|  | **No behavior considered** | **Behavior and timing considered separately** |  | **Behavior and timing considered together** |  |
|---|---|---|---|---|---|
|  |  | **Combinational** | **Sequential** | **Combinational** | **Sequential** |
| **Input test vectors needed** |  |  |  | Simulation-based approaches<br>Repeated fixed delay simulation [4]<br>Minmax simulation [14]<br>Minmax simulation + Event graphs [8]<br>Fixed delay simulation + Path tracing [5] |  |
| **No input test vectors** | Path tracing [10] | Path tracing + static sens. [3]<br><br>Causality graph [13] | Path tracing + STG analysis [12] | Path tracing + true sens. [6, 7] | Transformations on OEgraphs [1]<br><br>Sequential path tracing [MTV] |

Figure 1: Classification of Timing Verification Approaches.

into datapath and controller. This approach is limited to fully synchronous designs where clock gating (and hence multi-cycle paths) is allowed only in the datapath using signals generated by the controller that takes the op-code bits as input.

- **Transformation on OEgraphs** [1] which model behavior and timing together is applicable to a general class of designs. However, the transformation rules are adhoc and do not adequately handle the large degree of concurrency encountered for a large number of components.
- **Sequential path tracing** described in this paper also considers behavior and timing together. While it generates and analyzes the STG (similar to [12]), it is generally applicable to any interconnection of finite state machines with delays, and more importantly, it does not assume fully synchronous operation, predefined clock phase signals or a predetermined clock schedule; all these factors are critical for applicability to microprocessor-based designs.

Another significant issue addressed by other researchers [15] is the ability to handle level-sensitive latches (since coupling of inputs and outputs when the latch is transparent results in cyclic relations which are difficult to solve). MTV can handle latches only in limited cases (where no feedback loops with latches exist), but these cases have been adequate for microprocessor-based designs encountered so far[2]. Again, previous approaches that allow latches also ignore the sequential behavior of the design (except for the clock schedule provided by the user) and have the limitations described next.

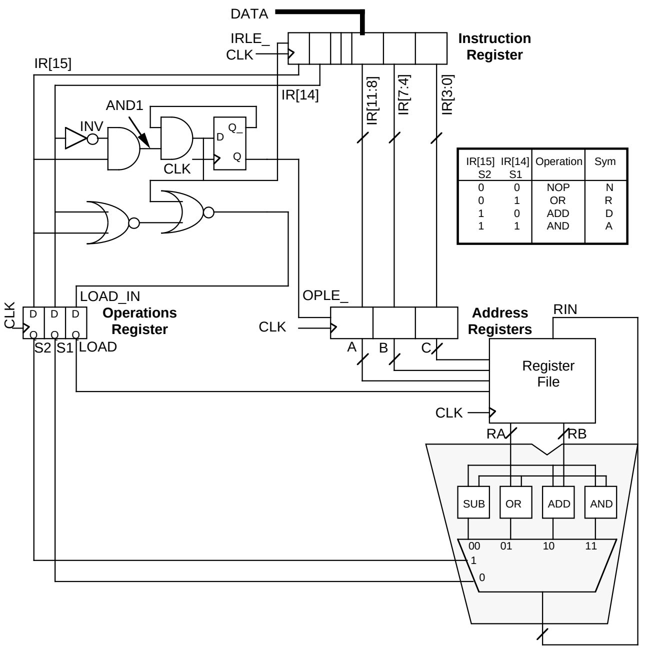
## B. *Limitations of CPT approaches*

The limitations of CPT are illustrated using the simple CPU example in Figure 2. The datapath has a three-ported register file and an ALU. The controller loads the instruction register from the data bus, decodes it, and loads the operations and address registers in the next cycle. The ALU is a standard part from a library with the delay for the SUB operation larger than that of the ADD operation which is larger than that for the AND and OR operations. The ALU has not been specifically designed for for this particular CPU. In particular, the SUB operation of the ALU is never used by this CPU since it does not support a SUB instruction. Also, the controller has been designed such that the ADD operation is allowed two clock cycles to execute. Clearly, the designer (human or a synthesis tool) has this information, but it is not directly available to the timing verifier.

### Sequential sensitizability

Consider the data setup check for the RIN port on the register file; the check is valid only when LOAD = 1. CPT indicates that the longest path is the SUB-path from the address registers A (or B) through the register file through the ALU for the SUB operation. CPT checks the sensitizability of the SUB-path and back-propagates the sensitization criterion to result in S2, S1, LOAD = 001; since

[2]Note that unlike simulation where one can not ensure that a long enough sequence has been simulated, MTV can analyze delays of paths that span loops in the STG in a manner similar to cross-frame constraints [11]; these extensions are not straightforward (since MTV does not assume a predetermined clock schedule) and need further investigation.



| IR[15] S2 | IR[14] S1 | Operation | Sym |
|---|---|---|---|
| 0 | 0 | NOP | N |
| 0 | 1 | OR | R |
| 1 | 0 | ADD | D |
| 1 | 1 | AND | A |

Figure 2: A simple CPU example.

no conflicts are detected in the back-propagation, CPT considers the SUB-path as a true sensitizable path and generates the constraint:

Clock period $\geq$ Delay of SUB-path + Setup time of register file

But this constraint is erroneous! Attempting to back-propagate the S2, S1, LOAD values to the inputs of the flip flops (i.e. look back one clock cycle) quickly generates a conflict. Thus, the SUB-path is *not* sensitizable if back-propagation is done in one previous clock cycle. Looking beyond the synchronizing elements for the sensitizability of the path is termed **sequential sensitizability**. We would need to propagate backwards to the initial reset state to ensure that a path is sequentially sensitizable.

In general, *combinational sensitizability* assumes that all input vectors to the combinational logic can be applied, which might be incorrect since some states might be unreachable from the reset state. Only recently, other researchers have exploited this property to prune multi-cycle false paths[2].

### Multi-cycle paths

The next longest path is the ADD-path (similar to the SUB-path but for the ADD operation through the ALU) giving the constraint:

Clock period $\geq$ Delay of ADD-path + Setup time of register file

The path starts with a new value being loaded in the address registers at a clock edge when OPLE_ = 0 and terminates at a clock edge with LOAD = 1 when the result is written back to the register file. The above constraint implicitly assumes that there is

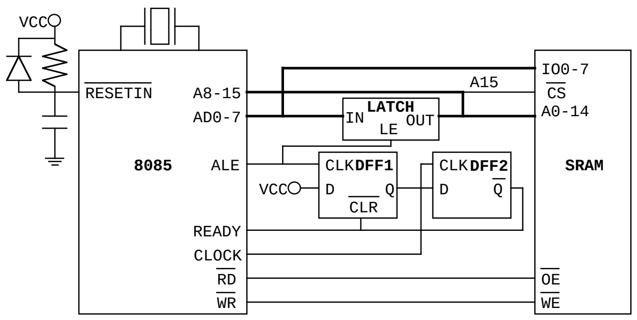


Figure 3: Simple Intel 8085-based design.

one clock period between these clock edges. However, sensitizing S2, S1, LOAD = 101 in the previous clock cycle yields OPLE$_-$ = 1 in the previous clock cycle, requiring yet another clock cycle for OPLE$_-$ = 0. Thus, there are in fact two clock periods between the clock edge in which OPLE$_-$ = 0 and S2, S1, LOAD = 101, resulting in the correct constraint:

$$2 \times \text{Clock period} \geq \text{Delay of ADD-path + Setup time of reg file}$$

Paths that have several clock cycles to propagate signals are termed **multi-cycle paths** (i.e. paths whose input is stable for more than one clock cycle before the output is required). No previously reported timing verification approach addresses multi-cycle paths automatically for a general circuit.

**Symbolic vs Numeric Delay Attributes**

Timing verifiers use one of several component delay models – fixed, unbounded, min-max, mean-variance, or slew-rate-based. Each model represents the delay as one or more parameters. Numeric values for these parameters are used by most timing verifiers. So if some component is replaced with a faster or slower version, the verifier has to rerun. Incremental timing verifiers mitigate this limitation by reusing previous results. Our approach is to treat delay parameters as symbolic variables. Symbolic constraints could be generated once; on each iteration different numeric values could be substituted into these symbolic constraints to ascertain if the circuit satisfies the timing checks. Symbolic constraints could be used in the reverse manner – obtaining the fastest possible clock given a set of circuit delays. The constraints could also be used in the synthesis process during component selection. Symbolic delays enables the modeling of blocks that are not yet designed; the generated symbolic constraints would provide design constraints for the block. Only one of the existing timing verifiers [1] generates symbolic constraints. MTV uses min-max delays, but can be extended to other delay models.

**Microprocessor-based Designs**

The inability to handle sequential sensitizability and multi-cycle paths is especially critical for microprocessor-based designs. Consider the circuit in Figure 3 which has an Intel 8085 connected to a static memory chip. Since the data bus is multiplexed with the lower byte of the address, a latch is used to demultiplex these signals. The D flip flops generate a single wait-state and the RC network generates a power-on reset pulse. The read transaction to memory is allowed 7 half clock cycles for address delay, and 5 half clock cycles for read delay, where the number depends upon the D flip flop circuitry and the 8085 bus interface. Note that the design uses asynchronous clear for the first D flip flop (a manufacturer recommended design style!), has timing checks relative to WE and ALE (not just CLOCK), and several unreachable states (e.g. READY is always asserted in 8085 bus-state Tw). All these circuit features cannot be supported by previous approaches (except simulation). The next section introduces MTV's approach for handling such circuits.

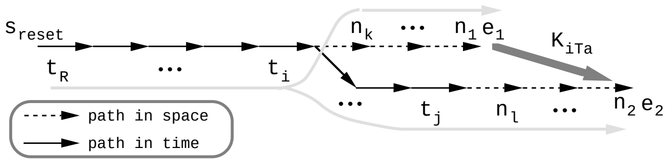


Figure 4: Constraint generation using sequential path tracing.

## III. Sequential Path Tracing

CPT stops tracing paths at the synchronizing elements. *Sequential path tracing* continues tracing beyond the synchronizing elements to the next or previous state and continues regular path tracing in the combinational network for the next or previous time step[3]. These paths are through *both space and time* and are termed **sequential paths** – a term borrowed from sequential test pattern generation [14]. Each sequential path starts with a sequence of transitions denoted by $t_x \ldots t_y$ (which is the sub-path in time and can be null) followed by a sequence of nets denoted by $n_p \ldots n_q$ (which is the sub-path in space for the last transition in the sub-path in time). The delay of a sequential path is the sum of the delays of the combinational sub-paths and delays of each state transition in the path. For synchronous edge-triggered designs, the delay of each transition would be one clock cycle. For other designs, the delay is not as simple and is computed by the procedure in Section V.C.

Consider a timing check:

$$(e_1 \text{ when } C_1) \text{ BEFORE } (e_2 \text{ when } C_2) \text{ ATLEAST } (\kappa_{iT_a})$$

that requires an event $e_1$ under condition $C_1$ to occur before event $e_2$ under condition $C_2$ by at least some timing attribute $\kappa_{iT_a}$. For example, the register file in Figure 2 has the check:

$$(\text{RIN V}) \text{ BEFORE } (\text{CLK 1 when LOAD} = 1) \text{ ATLEAST } (\kappa_{\text{RF},\text{T}_\text{s}}).$$

The timing verifier must ensure that when conditions are satisfied,

$$(\text{Earliest } e_2) - (\text{Latest } e_1) \geq \kappa_{iT_a}$$

The corresponding constraints are generated as below (see Figure 4):

1. find all sequential paths $t_R \ldots t_i n_k \ldots n_1$ from the reset state to event $e_1$ on net $n_1$.
2. find all sequential paths $t_i \ldots t_j n_l \ldots n_2$ to event $e_2$ on net $n_2$ such that $e_1$ does not happen again along this path.
3. for each pair of paths, the timing check is satisfied if (Earliest $e_2$) - (Latest $e_1$) $\geq$ $\kappa_{iT_a}$ which gives the relation:

   $$p_{t_i \ldots t_j} + d_{t_j n_2} - D_{t_i n_1} \geq \kappa_{iT_a}$$

   where $p_{t_i \ldots t_j}$ is the minimum delay of path from $t_i$ to $t_j$, $d_{t_j n_2}$ is the minimum delay of combinational path to $n_2$ relative to start of $t_j$, and $D_{t_i n_1}$ is the maximum delay of combinational path to $n_1$ relative to start of $t_i$.

Since only paths that start from the reset state are traced, all paths considered are sequentially sensitizable. Also, in the case where the combinational path to $e_1$ is activated multiple clock cycles before the combinational path to $e_2$, path $t_i \ldots t_j$ includes the appropriate number of state transitions in addition to the combinational path, and the constraint has the correct number of clock cycles computed automatically. Thus, this approach does not have any of the limitations described previously.

The above procedure requires repeated explorations of the STG. MTV builds the STG once and uses it for all subsequent processing. Some might consider building an *explicit* STG a throwback to early formal verifiers, but we have found the explicit representation necessary to support circuits with timing-dependent logic behavior [9].

[3]Looping may occur if the same state is reached again; such paths are allowed if one can exit the loop for some primary input combination.

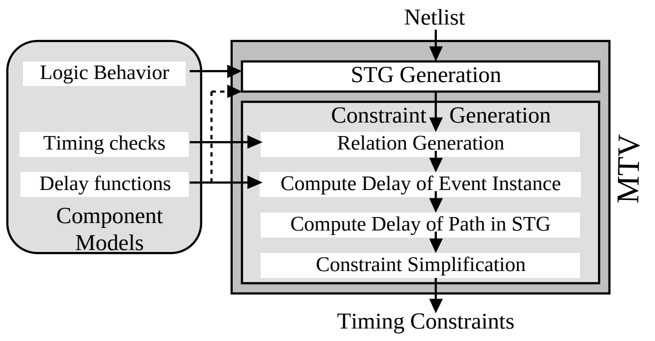


Figure 5: MTV System Diagram.

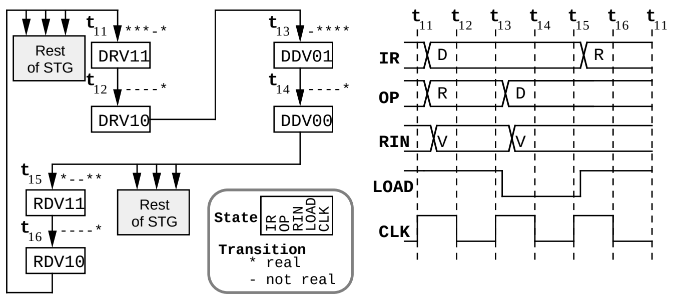


Figure 6: State Transition Graph for the simple CPU example.

The overall organization of MTV is illustrated in Figure 5. The netlist and component logic behavior are used in STG generation as sketched in Section IV. The STG is used to generate constraints in four steps detailed in Section V. Due to space limitations, we will avoid unnecessary details of the model of a microprocessor and other complex VLSI components, and use simpler gate-level components to explain the MTV approach. Application to microprocessor-based designs is demonstrated in Section VI.

## IV. STG Generation

The circuit consists of an interconnection of combinational and sequential components. For each sequential component, changes in its internal state and outputs are caused by changes on one or more of its inputs that are termed **trigger events** on **trigger nets**. For example, for a positive-edge triggered D flip flop, the net connected to the CLK input is the only trigger net and its transition to 1 is the trigger event. Note that the overall circuit may have any number of trigger nets, which may be outputs of other components or primary inputs. Unlike a traditional STG for synchronous circuits where transitions correspond to events on a global clock, transitions in our enhanced STG correspond to events on one of the trigger nets. The definition of STG below is based on the notion of the *circuit state* in a logic simulator – it includes the value of all nodes in the circuit and future values of these nodes on the event queue; each transition corresponds to selecting an event from the front of the queue and propagating the new events. This concept of state is simplified into the definition below based on the reasoning that we are interested in the net values only at discrete points in time when trigger events happen; the non-trigger nets are assumed to settle to the stable values – this assumption limits the class of circuits but also drastically reduces the complexity of the STG [9]; timing constraints ensure that the assumption is indeed satisfied.

The **state** of the circuit is defined as a set of values of all internal states of components, all nets, and values to be scheduled on the trigger nets. A trigger event is activated in a state if the scheduled value on the net is different from the current value. Given a state $s$ and an activated trigger event $e$, the next state $ns$ is the result of evaluating the sequential elements that have trigger $e$ and evaluating all combinational elements in the fanout[4]. The STG contains transitions from $s$ to $ns$ for all possible input vectors. An event on a net is defined to be **real** in a transition if the value of the net is not stable in that transition; there are flags with each transition to indicate real events for each net. Note that a change in the value is sufficient, but not necessary for an event to be real (since hazards may occur as determined by delay functions [9], see V.B for an example). STG generation starts with the reset state that has all nets (except the power supply) at unknown value; hence, timing during power-up initialization sequence is also verified.

Figure 6 shows a portion of the STG generated for the CPU example; a timing diagram is provided to illustrate events in one path in the STG. For example, in transition $t_{13}$, CLK changing from 0 to 1 is the trigger event and IR stays stable at operation D (ADD), OP changes from operation R (OR) to D (ADD), RIN changes from some valid value (V) to another V, and LOAD changes from 1 to 0; all events in the transition happen sometime after the trigger event; the string _**** with $t_{13}$ indicates that there is no real event on IR (i.e. it is stable), and there are real events on all other nets (OP, RIN, LOAD, and CLK).

## V. Constraint Generation

Multiple relations are generated for each check:

$$(e_1 \text{ when } C_1) \text{ BEFORE } (e_2 \text{ when } C_2) \text{ ATLEAST } (\kappa_{iT_a})$$

The relations contain delays of event instances that need to be simplified into functions of timing attributes of components in the design. This process may introduce delays of paths in the STG into the relation, which in turn need to be simplified into functions of timing attributes of the components. Each of these steps is described below. Terminology used in this section is summarized in Figure 7.

### *A. Relation Generation*

For each transition $t$ in which $e_1$ is real and $C_1$ is satisfied, there are two possibilities for the next real instance of $e_2$ where $C_2$ is satisfied:

- it is in $t$ after $e_1$: The relation is:

$$d_{te_2} - D_{te_1} \geq \kappa_{iT_a}$$

- it is in transition $t_2$ after sequence of transitions $t \ldots t_2$: The relation is:

$$p_{t \ldots t_2} + d_{t_2 e_2} - D_{te_1} \geq \kappa_{iT_a}$$

For the data setup timing check on the register file in Figure 2 and the portion of STG in Figure 6, relations are generated as below:

- for $t_{11}$, RIN V is real; next CLK 1 when LOAD=1 is in $t_{13}$ after path $t_{11}t_{12}t_{13}$; hence relation is:

$$p_{t_{11}t_{12}t_{13}} + d_{t_{13},\text{CLK}} - D_{t_{11},\text{RIN}} \geq \kappa_{\text{RF},\text{T}_\text{s}}$$

- for $t_{12}$, RIN V is is not real;
- for $t_{13}$, RIN V is real; next CLK 1 when LOAD=1 is in $t_{11}$ after path $t_{13}t_{14}t_{15}t_{16}t_{11}$; hence relation is:

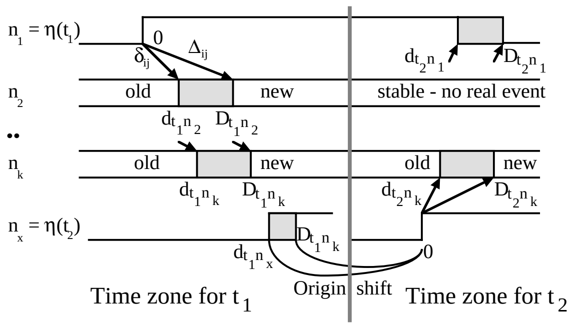
$$p_{t_{13}t_{14}t_{15}t_{16}t_{11}} + d_{t_{11},\text{CLK}} - D_{t_{13},\text{RIN}} \geq \kappa_{\text{RF},\text{T}_\text{s}}$$

At this stage, each relation is in terms of variables – delays of event instances and delay of paths. They are to be computed next and substituted into the relation to generate the constraint.

### *B. Delay of an Event Instance*

An event on a net is caused by the component whose output pin is connected to that net. Its delay can be computed using functions provided in the component's model that express the output delay in terms of the component's timing attributes and delay of input events.

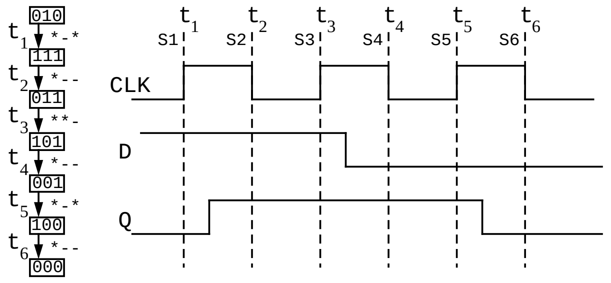
[4]Feedback between combinational elements is not supported.



| | |
|---|---|
| **Parameters** | |
| $\delta_{ij}$ | minimum value of attribute $j$ of component instance $i$ |
| $\Delta_{ij}$ | maximum value of attribute $j$ of component instance $i$ |
| $\kappa_{ij}$ | min. value of timing check attribute $j$ of component instance $i$ |
| **Variables** | |
| $d_{tn}$ | earliest change in net $n$ in transition $t$ relative to start of $t$ |
| $D_{tn}$ | latest stable time of net $n$ in transition $t$ relative to start of $t$ |
| $p_{t_1 \ldots t_k}$ | minimum time between start of transition $t_1$ and $t_k$ |
| $P_{t_1 \ldots t_k}$ | maximum time between start of transition $t_1$ and $t_k$ |
| **Functions** | |
| $\eta(t)$ | trigger net for transition $t$ |
| $s_{\text{from}}(t)$ | state from which transition $t$ begins |
| $s_{\text{to}}(t)$ | state to which transition $t$ ends |
| **Component delay fns** in terms of time of occ. of i/p events & $\delta_{ij}$ or $\Delta_{ij}$ | |
| $C(i)$ | input cubes for component instance $i$ |
| Edge-triggered component instance $i$ | |
| $\omega_{icn}$ | minimum delay of output $n$ under input conditions $c$ |
| $\Omega_{icn}$ | maximum delay of output $n$ under input conditions $c$ |
| Level-sensitive or combinational component instance $i$ | |
| $\omega^{\text{off}}_{icn}$ | earliest change on output $n$ when input conditions $c$ are changed |
| $\Omega^{\text{on}}_{icn}$ | latest output $n$ is stable when input conditions $c$ are established |

Figure 7: Terminology.

| Trigger CLK 1 | | | $\omega_{\text{DFF,c,Q}}$ | $\Omega_{\text{DFF,c,Q}}$ |
|---|---|---|---|---|
| Inputs | | Output | | |
| D | Q$_{\text{old}}$ | Q$_{\text{new}}$ | | |
| 0 | 0 | At rest | - | - |
| 0 | 1 | 0 | $d_{t,\text{CLK}} + \delta_{\text{DFF},\text{T}_{\text{hl}}}$ | $D_{t,\text{CLK}} + \Delta_{\text{DFF},\text{T}_{\text{hl}}}$ |
| 1 | 0 | 1 | $d_{t,\text{CLK}} + \delta_{\text{DFF},\text{T}_{\text{lh}}}$ | $D_{t,\text{CLK}} + \Delta_{\text{DFF},\text{T}_{\text{lh}}}$ |
| 1 | 1 | At rest | - | - |

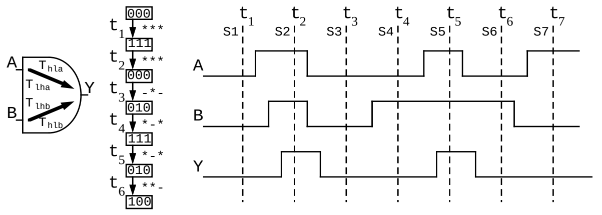


| Transition $t$ | $d_{t\text{Q}}$ | $D_{t\text{Q}}$ |
|---|---|---|
| $t_1$ | $d_{t,\text{CLK}} + \delta_{\text{DFF},\text{T}_{\text{lh}}}$ | $D_{t,\text{CLK}} + \Delta_{\text{DFF},\text{T}_{\text{lh}}}$ |
| $t_2$ | Not real | Not real |
| $t_3$ | Not real | Not real |
| $t_4$ | Not real | Not real |
| $t_5$ | $d_{t,\text{CLK}} + \delta_{\text{DFF},\text{T}_{\text{hl}}}$ | $D_{t,\text{CLK}} + \Delta_{\text{DFF},\text{T}_{\text{hl}}}$ |
| $t_6$ | Not real | Not real |

Figure 8: Delay expression computation example for a D flip flop.

For edge-triggered sequential component $i$, the model provides delay expressions $\omega_{icn}$ and $\Omega_{icn}$ for each output $n$ under different input condition $c$. The delay expression of the output event instance

| Inputs | | O/p | $\omega^{\text{off}}_{\text{AND,c,Y}}$ | $\Omega^{\text{on}}_{\text{AND,c,Y}}$ |
|---|---|---|---|---|
| A | B | Y | | |
| 0 | ? | 0 | $d_{t\text{A}} + \delta_{\text{AND},\text{T}_{\text{lha}}}$ | $D_{t\text{A}} + \Delta_{\text{AND},\text{T}_{\text{hla}}}$ |
| ? | 0 | 0 | $d_{t\text{B}} + \delta_{\text{AND},\text{T}_{\text{lhb}}}$ | $D_{t\text{B}} + \Delta_{\text{AND},\text{T}_{\text{hlb}}}$ |
| 1 | 1 | 1 | $\min(d_{t\text{A}} + \delta_{\text{AND},\text{T}_{\text{hla}}}, d_{t\text{B}} + \delta_{\text{AND},\text{T}_{\text{hlb}}})$ | $\max(D_{t\text{A}} + \Delta_{\text{AND},\text{T}_{\text{lha}}}, D_{t\text{B}} + \Delta_{\text{AND},\text{T}_{\text{lhb}}})$ |



| Transition $t$ | $d_{t\text{Y}}$ | $D_{t\text{Y}}$ |
|---|---|---|
| $t_1$ | $\max(d_{t_1\text{A}} + \delta_{\text{AND},\text{T}_{\text{lha}}}, d_{t_1\text{B}} + \delta_{\text{AND},\text{T}_{\text{lhb}}})$ | $\max(D_{t_1\text{A}} + \Delta_{\text{AND},\text{T}_{\text{lha}}}, D_{t_1\text{B}} + \Delta_{\text{AND},\text{T}_{\text{lhb}}})$ |
| $t_2$ | $\min(d_{t_2\text{A}} + \delta_{\text{AND},\text{T}_{\text{hla}}}, d_{t_2\text{B}} + \delta_{\text{AND},\text{T}_{\text{hlb}}})$ | $\min(D_{t_2\text{A}} + \Delta_{\text{AND},\text{T}_{\text{hla}}}, D_{t_2\text{B}} + \Delta_{\text{AND},\text{T}_{\text{hlb}}})$ |
| $t_3$ | Not real | Not real |
| $t_4$ | $d_{t_4\text{A}} + \delta_{\text{AND},\text{T}_{\text{lha}}}$ | $\max(D_{t_4\text{A}} + \Delta_{\text{AND},\text{T}_{\text{lha}}}, D_{t_4\text{B}} + \Delta_{\text{AND},\text{T}_{\text{lhb}}})$ |
| $t_5$ | $\min(d_{t_5\text{A}} + \delta_{\text{AND},\text{T}_{\text{hla}}}, d_{t_5\text{B}} + \delta_{\text{AND},\text{T}_{\text{hlb}}})$ | $D_{t_5\text{A}} + \Delta_{\text{AND},\text{T}_{\text{hla}}}$ |
| $t_6$ | Not real | Not real |

Figure 9: Delay expression computation example for an AND gate.

depends on the current input condition (i.e. input condition cube that the current state is within) as given below:

$$d_{tn} = \omega_{icn} \mid_{s_{\text{to}}(t) \subseteq c} \qquad D_{tn} = \Omega_{icn} \mid_{s_{\text{to}}(t) \subseteq c}$$

Figure 8 provides the model of a D flip flop and illustrates delay expressions for an example STG.

For combinational components[5], the delay is computed by a novel scheme. The function of the combinational element is represented as a list of cubes for the ON and OFF sets. With each cube $c$, there is a delay expression for the latest the cube turns ON and earliest the cube turns OFF, denoted by $\Omega^{\text{on}}_{icn}$ and $\omega^{\text{off}}_{icn}$ respectively. The delay expressions of the real output event in transition $t$ are:

$$d_{tn} = \max_{c \in C(i), s_{\text{from}}(t) \subseteq c} \omega^{\text{off}}_{icn}$$ i.e. latest all cubes in previous state turn off;

$$D_{tn} = \min_{c \in C(i), s_{\text{to}}(t) \subseteq c} \Omega^{\text{on}}_{icn}$$ i.e. earliest some cube in next state turns on.

Figure 9 provides the model for an AND gate and delay expressions for the output for several transitions in a sample STG portion (which is also illustrated by the timing diagram). The delay expressions are also used during STG generation to detect false hazards (when $d_{tn} > D_{tn}$ e.g. in transition $t_6$ of Figure 9). Numeric min-max delay values may be needed for hazard detection resulting in preconditions for the validity of the STG.

In some cases, the delay of the output event in a transition might depend on the delay of an input event in a previous transition. This phenomenon of **hidden causation** is particularly significant when there is a large difference in the delays from two inputs. For example, consider the case where ADDR is input to a SRAM chip and OE is asserted on the next clock cycle; the DATA output stays tristated in the first clock cycle (i.e. no effect due to ADDR event) and becomes stable after OE is asserted in the second clock cycle; however, the delay of DATA being stable depends upon not only the delay of the

[5]Level-sensitive sequential components are similar except that their outputs are allowed to stay at rest.

OE event but also on the delay of the previous ADDR event (since the delay from ADDR to DATA is usually much larger than that from OE and the difference might be larger than one clock cycle). Such *hidden* causation is not accurately addressed by most simulation-based approaches, but is correctly handled by the procedure above. For example in Figure 9, the expression for $D_{t_4\mathrm{Y}}$ includes $D_{t_4\mathrm{B}}$. Now, in $t_4$, the event on B is not real and the extra step below is used to compute its delay.

For a virtual event $e$ in $t$, all backward paths $BP(t, e)$ in the STG are found such that for each $t_1 \ldots t \in BP(t,\ e)$:

- event $e$ in transition $t_1$ is real, and
- for all transitions $t_i$ within $t_1 \ldots t$, event $e$ is not real.

There is a set of delay expressions for the virtual event as follows:

$$D_{te} = \{D_{t_1 e} - p_{t_1 \ldots t} \mid t_1 \ldots t \in BP(t,\ e)\}$$

In the example in Figure 9, $D_{t_4\mathrm{B}} = D_{t_3\mathrm{B}} - p_{t_3 t_4}$

### C. Delay of a Path in the STG

A path is a sequence of transitions in the STG. The delay of the path is the time between the trigger of the first transition to the trigger of the last transition. If the next value on $\eta(t_2)$ is scheduled in transition $t_1$, then $t_1$ *trigger-causes* $t_2$ and the minimum delay of any path from $t_1$ to $t_2$ is $d_{t_1\eta(t_2)}$. A trigger causal chain $t_1 \ldots t_i t_{i+1} \ldots t_n$ of a path is an ordered list of transitions within the path such that $t_i$ trigger-causes $t_{i+1}$, $1 \le i < n$. The delay of the trigger causal chain is:

$$c_{t_1 \ldots t_n} = \sum_{i=1}^{n-1} d_{t_i \eta(t_{i+1})}$$

If $TCC(\pi)$ denotes the set of all possible maximal trigger causal chains in path $\pi$,

$$p_\pi = \max_{t_1 \ldots t_n \in TCC(\pi)} c_{t_1 \ldots t_n},$$

since the path must be at least as long as the chains it contains. The algorithm for computing $TCC(\pi)$ for multi-synchronous designs is omitted due to space restrictions. For synchronous designs, the computation is straightforward since for designs with no races between triggers, $TCC(\pi) = \{\pi\}$. For the example from Section V.A,

$$\begin{aligned} p_{t_{13}t_{14}t_{15}t_{16}t_{11}} &= d_{t_{13}\mathrm{CLK}} + d_{t_{14}\mathrm{CLK}} + d_{t_{15}\mathrm{CLK}} + d_{t_{16}\mathrm{CLK}} \\ &= 4 \times \delta_{\mathrm{CLKGEN},\mathrm{T_{halfcyc}}} \end{aligned}$$

### D. Constraint Simplification

A set of constraints are generated by substituting expressions for delay of an event instance and delay of a path into the relations generated for each timing check. These constraints are reduced to a canonical form. For the register file timing check, the relation derived for transition $t_{13}$ in Section V.A results in the constraint:

$$\begin{aligned} 4 \times \delta_{\mathrm{CLKGEN},\mathrm{T_{halfcyc}}} - \max( & \Delta_{\mathrm{A.DFF},\mathrm{T_p}} + \Delta_{\mathrm{RF},\mathrm{T_a}} + \Delta_{\mathrm{ALU},\mathrm{T_{add}}}, \\ & \Delta_{\mathrm{B.DFF},\mathrm{T_p}} + \Delta_{\mathrm{RF},\mathrm{T_a}} + \Delta_{\mathrm{ALU},\mathrm{T_{add}}}, \\ & \Delta_{\mathrm{S2.DFF},\mathrm{T_{plh}}} + \Delta_{\mathrm{ALU},\mathrm{T_{s2s1bar}}}, \\ & \Delta_{\mathrm{S1.DFF},\mathrm{T_{phl}}} + \Delta_{\mathrm{ALU},\mathrm{T_{s1}}}) \ge \kappa_{\mathrm{RF},\mathrm{T_s}} \end{aligned}$$

which is reduced to the conjunction of the following constraints in the canonical form:

$$\begin{aligned} 4 \times \delta_{\mathrm{CLKGEN},\mathrm{T_{halfcyc}}} - \Delta_{\mathrm{A.DFF},\mathrm{T_p}} - \Delta_{\mathrm{RF},\mathrm{T_a}} - \Delta_{\mathrm{ALU},\mathrm{T_{add}}} - \kappa_{\mathrm{RF},\mathrm{T_s}} &\ge 0 \\ 4 \times \delta_{\mathrm{CLKGEN},\mathrm{T_{halfcyc}}} - \Delta_{\mathrm{B.DFF},\mathrm{T_p}} - \Delta_{\mathrm{RF},\mathrm{T_a}} - \Delta_{\mathrm{ALU},\mathrm{T_{add}}} - \kappa_{\mathrm{RF},\mathrm{T_s}} &\ge 0 \\ 4 \times \delta_{\mathrm{CLKGEN},\mathrm{T_{halfcyc}}} - \Delta_{\mathrm{S2.DFF},\mathrm{T_{plh}}} - \Delta_{\mathrm{ALU},\mathrm{T_{s2s1bar}}} - \kappa_{\mathrm{RF},\mathrm{T_s}} &\ge 0 \\ 4 \times \delta_{\mathrm{CLKGEN},\mathrm{T_{halfcyc}}} - \Delta_{\mathrm{S1.DFF},\mathrm{T_{phl}}} - \Delta_{\mathrm{ALU},\mathrm{T_{s1}}} - \kappa_{\mathrm{RF},\mathrm{T_s}} &\ge 0 \end{aligned}$$

Duplicate and dominated constraints are removed. For example, constraint $a - b \ge 0$ algebraically dominates $a - b + c \ge 0$ if $c$ is known to be non-negative; hence the latter constraint is deleted. Also, if the delays to the two events in the timing check are correlated since they pass through the same causation link in a component, then the constraint would have $\delta_{ij}$ and $\Delta_{ij}$ (i.e. max and min of the same timing attribute) which can be canceled out; thus symbolic constraints also enable correct handling of common ambiguity (other verifiers would be overly pessimistic). Symbolic expression manipulation rules are used for all these transformations. The final output is a minimal simplified set of constraints for each timing check.

### E. Preconditions

Note that delay expressions used so far were in terms of symbolic minimum and maximum delay attributes. To determine the ordering between events (during relation generation and also during STG generation), delay expressions of two event instances have to be compared. In some comparisons, numeric min-max values of the delays might have to be used. The relationships satisfied by the numeric values are stored as preconditions for the validity of the STG and the generated constraints.

TABLE I: EXPERIMENTAL RESULTS SUMMARY

| Design | S | T | Tg | TC | C | P | CPU time (s) | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | MTV | MATH |
| **CPU ex** (Figure 2) | 30 | 62 | 2 | 18 | 54 | 0 | 141.2 | 130.7 |
| **6809-based designs** | | | | | | | | |
| SRAM@0w | 37 | 53 | 3 | 16 | 20 | 4 | 58.5 | 55.0 |
| SRAM@1w | 42 | 58 | 3 | 16 | 20 | 4 | 65.6 | 61.4 |
| SRAM@0w, SRAM@1w | 42 | 58 | 3 | 32 | 41 | 5 | 130.9 | 124.6 |
| SRAM@1w, SIO@0w | 43 | 60 | 3 | 35 | 39 | 5 | 87.3 | 81.4 |
| **8085-based designs** | | | | | | | | |
| SRAM@0w, SIO@0w | 34 | 47 | 3 | 41 | 43 | 3 | 50.6 | 47.3 |
| SRAM@0w, SRAM@0w | 34 | 47 | 3 | 46 | 42 | 3 | 58.8 | 55.9 |
| SRAM@1w (Figure 3) | 42 | 52 | 5 | 28 | 26 | 7 | 47.3 | 43.8 |
| SRAM@1w, SRAM@1w | 67 | 92 | 5 | 50 | 46 | 8 | 129.6 | 122.0 |
| SRAM@0w, SRAM@1w | 101 | 124 | 6 | 50 | 52 | 15 | 388.5 | 332.7 |
| **80188-based designs** | | | | | | | | |
| SRAM@0w | 40 | 53 | 3 | 27 | 25 | 2 | 37.3 | 34.8 |
| SRAM@0w, SRAM@0w | 62 | 99 | 3 | 49 | 50 | 3 | 90.7 | 85.0 |
| SRAM@0w, PAL | 65 | 105 | 3 | 27 | 26 | 5 | 86.9 | 80.9 |
| SRAM@0w, ATbus (no IO) | 312 | 423 | 8 | 46 | 43 | 22 | 1608.9 | 1368.1 |
| SRAM@0w, ATbus | 343 | 458 | 10 | 46 | 49 | 22 | 2619.5 | 2011.3 |

## VI. EXPERIMENTAL RESULTS

MTV has been implemented in C++ with the symbolic expression manipulation and simplification in MATHEMATICA [16]. MTV takes as input an EDIF netlist and component models from a library and generates a set of constraints for each check in the component models. Results of using MTV for several example circuits are summarized in Table I. For each circuit, the table provides the number of states and transitions in the STG generated by MTV, the number of triggers in the circuit, the number of distinct timing checks, the number of constraints generated for the checks, the number of preconditions, and the total MTV CPU time and that for MATHEMATICA on an Ultrix DEC5000/200. Note that a large fraction of total MTV time is spent in MATHEMATICA, underscoring the large processing overhead for symbolic delays. No comparisons are included since no other tool can automatically handle microprocessor-based designs with multi-cycle paths. Observe that the STG size does not grow exponentially with the number of components for microprocessor-based designs; this can be explained by the bus-oriented design style in which the complexity increases linearly with the number of components on the bus. The experiments show that MTV is computationally feasible for embedded controller boards. Workstation boards may have upto an order of magnitude more components (after removing multiple instances of identical components); current data shows that the CPU time per timing check increases linearly with the product of the num-

TABLE II: DETAILS OF PATHS AND MTV-GENERATED CONSTRAINTS FOR SOME EXAMPLES

| Design | Timing Check | Logical Paths | | | Simplified | | Output Constraints | |
|---|---|---|---|---|---|---|---|---|
| | | # total | # False paths | | # symbolically identical | # dominated | # total | # paths (# half-cycles) |
| | | | Comb. | Seq. | | | | |
| CPU ex Figure 2 | RFILE setup | 22 | 0 | 5 | 3 | 0 | 14 | 9 (2), 5 (4) |
| | IRLE_ DFF setup | 6 | 0 | 0 | 0 | 2 | 4 | 4 (2) |
| | IRLE_ DFF hold | 6 | 0 | 0 | 0 | 3 | 3 | 3 (2) |
| | LOAD_IN DFF setup | 10 | 0 | 0 | 4 | 0 | 10 | 8 (2), 2 (4) |
| 8085 + SRAM @1wait state Figure 3 | Proc data setup | 7 | 0 | 1 | 0 | 1 | 5 | 1 (5), 3 (7), 1 (8) |
| | Mem data setup | 7 | 0 | 6 | 0 | 0 | 1 | 1 (5) |
| | Mem data hold | 7 | 0 | 6 | 0 | 0 | 1 | 1 (1) |
| | Mem addr setup (start write) | 3 | 0 | 0 | 0 | 0 | 3 | 3 (2) |
| | Mem addr setup (end write) | 3 | 0 | 0 | 0 | 0 | 3 | 3 (7) |
| | Mem addr hold | 3 | 0 | 0 | 0 | 1 | 2 | 2 (1) |
| | Mem write pulse width | 1 | 0 | 0 | 0 | 0 | 1 | 1 (5) |
| | Bus contention†: Mem Z BEFORE Proc V | 4 | 0 | 3 | 0 | 0 | 1 | 1 (1) |
| | Bus contention†: Proc Z BEFORE Mem V | 6 | 0 | 5 | 0 | 0 | 1 | 1 (0) |

† These checks are introduced automatically when there is more than one driver on a net.

ber of transitions and number of trigger nets; hence, we project that MTV would be suitable for workstation designs also.

Let us consider a few examples in more detail as shown in Table II. For each check on the components in the circuit, the table gives: the total number of logical paths; the number of these paths that are false since they are not combinationally or sequentially sensitizable; the number of paths that are simplified out since they are symbolically identical or dominated; the number of MTV-generated constraints which is the total number of paths minus the number of false and simplified paths. Some of these constraints involve multi-cycle paths (i.e. other than 2 half-clock cycles); the number of paths together with the number of half-clock cycles they are allowed to propagate is listed. Several rows of the table have sequentially unsensitizable false paths and multi-cycle paths underscoring the usefulness of MTV.

Consider the first row of Table II for the register file setup timing check in Figure 2. There are 22 logical paths from a rising clock event to RIN of the register file: 4 from CLK to A through register file and ALU (one for each ALU operation), similarly 4 from B, 4 from register file CLK to RA through ALU, similarly 4 through RB, 2 from rising S2 through ALU to RIN (one for each Boolean value of S1), similarly 2 for falling S2, 1 from rising S1 through ALU, and similarly 1 from falling S1. All of these paths are combinationally sensitizable. However, since the SUB-path through the ALU is not sequentially sensitizable, one path each from A, B, RA, RB, and the path from falling event on S2 when S1 = 0 is false. Also, since the delay from CLK to RA and RB is identical for the register file, symbolically identical constraints for 3 additional paths can be removed. The number of remaining constraints is (22 - 0 - 5 - 3) = 14, which is the number of constraints reported by MTV. Since the controller allows two clock cycles for the ADD-operation in the ALU, 5 paths, one each from A, B, RA (or RB), rising S2, and falling S1 is a multi-cycle path; this fact is automatically computed by MTV. Other rows can be explained similarly.

## VII. CONCLUSIONS

This paper has shown that by ignoring the sequential behavior of components in the design, previous static timing verification approaches could be overly pessimistic by reporting sequentially unsensitizable paths and assuming that all paths are limited to one clock cycle. These limitations make them inadequate for microprocessor-based designs. Sequential path tracing handles both these limitations and forms the basis of the MTV approach. MTV handles multi-cycle paths automatically, does not require a predefined clock schedule, and handles conditional or gated clocks, unrelated clocks, asynchronous set/reset, and power-up initialization. Another novel feature of MTV is that it generates symbolic constraints between timing attributes instead of determining if a set of numeric values of the timing attributes satisfy all timing checks; symbolic processing also enables common ambiguity removal and could be used to account for correlated delays. Experimental results demonstrate that MTV takes only a few CPU minutes for moderately-sized microprocessor-based designs.

## REFERENCES

[1] T. Amon and G. Borriello. An approach to symbolic timing verification. *29th DAC*, pages 410–413. 1992.

[2] P. Ashar, S. Dey, and S. Malik. Exploiting multi-cycle false paths in the performance optimization of sequential circuits. *ICCAD*, pages 510–517. 1992.

[3] J. Benkoski, E. V. Meersch, L. Claesen, and H. De Man. Efficient algorithms for solving the false path problem in timing verification. *ICCAD* , pages 44–47. 1987.

[4] J. Benkoski and A. J. Strojwas. A new approach to hierarchical and statistical timing simulation. *IEEE Trans. on CAD*, CAD-6(6):1039–1052, Nov. 1987.

[5] Cadence Design Systems. *Veritime Reference Manual*, 1989.

[6] H. C. Chen and D. H. C. Du. Path sensitization in critical path problem. *ICCAD*, pages 208–211. 1991.

[7] S. Devadas, K. Keutzer, and S. Malik. Delay computation in combinational logic circuits: Theory and algorithms. *ICCAD*, pages 176–179. 1991.

[8] D. Doukas and A. S. LaPough. CLOVER: A timing constraints verification system. *28th DAC*, pages 662–667. 1991.

[9] A. P. Gupta. *Timing Verification of Microprocessor-based Designs*. PhD thesis, ECE Department, Carnegie Mellon University, 1994.

[10] R. B. Hitchcock, Sr. Timing verification and the timing analysis program. *19th DAC*, pages 446–456. 1982.

[11] A. Ishii and C. E. Leiserson. A timing analysis of level-clocked circuitry. *Adv. Research in VLSI; 6th MIT Conf.*, pages 113–130, 1990.

[12] M. Kawarabayashi, N. Shenoy, and A. Sangiovanni-Vincentelli. A verification technique for gated clock. *30th DAC*, pages 123–127. 1993.

[13] A. R. Martello, S. P. Levitan, and D. M. Chiarulli. Timing verification using HDTV. *27th DAC*. 1990.

[14] A. Miczo. *Digital Logic Testing and Simulation*. Harper and Row, 1986.

[15] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. Analysis and design of latch-controlled synchronous digital circuits. *27th DAC*, pages 111–117. 1990.

[16] S. Wolfram. *Mathematica: A system for doing mathematics by computer*. Addison Wesley, 2 edition, 1991.