Automated Analysis of Timing Faults in Synchronous MOS Circuits. *

E. Vanden Meersch, L. Claesen, H. De Man †

Interuniversity Micro Electronics Center IMEC, Kapeldreef 75, B-3030 Leuven, Belgium.

Abstract

In this paper, an analysis is presented of timing faults in synchronous MOS circuits. A rule-based method is used to transform the network of MOS transistors, obtained from the physical layout by extraction, into a network of unidirectional subcircuits. Each subcircuit is characterised by a logic model, which is used to accurately derive the timing constraints. The timing model of the circuit, required by this algorithm, is derived from the created subcircuit structure. The logic model allows to eliminate false edges in the signal propagation graph. The resulting timing verifier can be used for a wide range of static and dynamic nMOS and CMOS circuits.

1 Introduction

Timing verifiers have become recognized tools for the verification of the performance of electronic designs at transistor level [1,2,11,12,4,13,7]. Commonly, they calculate for each node in a given circuit the settling time, or the latest point in time at which a signal transition can take place. These delay times are calculated independent of specific input excitations as is in contrast to simulation which can only be performed for specific input patterns.

Timing analysis and verification for MOS circuits is traditionally performed in a number of steps: (1) signal flow modeling and partioning in MOS transistors (2) delay modeling [12,3,8,10,9] and critical delay path analysis. Instead of using hardcoded heuristics to determine the signalflow as in [4,11,12], the SLOCOP timing verifier [13] relies on rule based circuit partioning [19] whereby signal flow as well as logic functionality is modeled for subcircuits to be extracted. This partioning also allows to identify combinatorial blocks as well as registers and latches in MOS circuits. In contrast of using crude approximative models, SLOCOP uses circuit simulation [13] for delay determination of individual subcircuits.

A timing verifier calculates settling times for each node in combinational circuits. It is up to the designer to draw conclusions about whether or not the circuit can be inserted in a synchronous system without causing timing faults. More advanced timing verification tools allow to directly analyse synchronous circuits. The important problem that arises here is to determine when the calculated settling times with respect to the different transitions of the clock actually imply a timing fault in the circuit.

Important work in the direction of automating the timing verification for synchronous circuits has been done in [7]. Szymanski describes how the timing analysis of synchronous circuits, irrespective of the number of clocks, can be performed by applying a PERT critical path evaluation of the timing model during each phase of the clock cycle. This is repeated over a number of clock cycles until the settling times converge.

In combination with this analysis algorithm, a rigourous method is needed to derive the constraints on the settling times which are to be verified. In this paper, it will be shown how a set of constraints can be derived which cover timing faults due to the improper timing of signals at inputs of level sensitive latches and precharged circuits. Also the specific timing constraints related to the use of conditional clocks are dealt with in a unified way. In figure 1, two situations are shown which lead to a timing fault in the circuit. In the first example, the latch output has not assumed the correct value before the start of the 'hold phase'. The signal A arrives too late to be gated in the latch. In the second example, the precharged output of the gate unintentionally discharges. Both timing faults are seemingly unrelated. It can however be seen that in both examples the final output state is a storage (or memory) state which stores the wrong logic value.

The basic observation that, when a memory state (M) can exist during a certain phase of the clock period, it must store the correct logic value, will be generalised and put in a form that is suitable for computer verification.

In Section 2 the logic model of a subcircuit will be defined as an expression of four operators. The possible output states that are considered are High, Low, Error and Memory (1,0,E,M). In Section 3, the timing constraints will be formulated. Section 4 deals with the timing analysis algorithm, which is needed to verify the timing constraints. Section 5 discusses the problem of compatibility of logic states at different nodes in the circuit. It will be shown that a detection of statically incompatible logic states is needed to avoid false timing errors and to solve the false path problem in the timing analysis. More extensive information on the algorithms in this paper is available in [19].

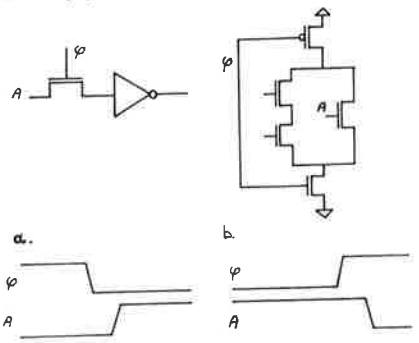


Figure 1: Examples of timing faults in clocked circuits.

2 Logic modeling of subcircuits.

The SLOCOP program uses a rule-based subcircuit partioning [14]. In the rule base, the possible generic subcircuits are described manually in the LEXTOC language [14] for a given design style. Togther with this partioning the logic model for the subcircuits is specified.

From the logic point of view, the exact voltage at the input or output of a subcircuit is not relevant. For each technology, there exists a function $D(v) : R \rightarrow \{0, 1, E\}$, which maps a voltage v onto one of three states: **0** : logic low, **1** : logic high, **E** : error. The **E** indicates an indetermined signal state that is not accepted by the logic model that is build of the subcircuit.

For a given subcircuit s, the following notation is introduced to denote inputs and output of a subcircuit s and the logic states at these nodes: $\beta_{out}(s) = \{n_0\}, \beta_{in}(s) = \{n_1, n_2, \ldots n_N\}, D(v(n_0)) = y, D(v(n_i)) = x_i, X = [x_1 \ldots x_N]^T$.

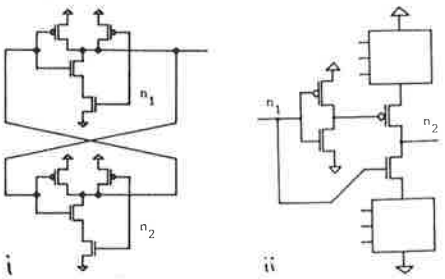
*Work sponsored by the EC under the ESPRIT-1058 project

†Professor at Kath. Univ. Leuven

The output of an arbitrary subcircuit of MOS transistors can, in accordance with previous work [5] be characterised by four states, which lead to a logic model as a function: $F : \{0,1,E,M\}^N \rightarrow \{0,1,E,M\}$ with:

$F(X) \in \{0,1,E\} \Rightarrow F(X) = y$
$F(X) = M \Rightarrow$ y is not determined by X

Physically, the M state indicates that the output of the subcircuit is in a bistable state. This means that both an electrically stable 0 or 1 state can exist at the output, together with these input states. Common ways to implement a bistable state is by electrical isolation of a node or



i. (**select** ((**not** n1) (**and** n1 (**not** n2))) ((**not** n1) n2))
ii. (**select** ((**and** n1 (**tf** n2 vdd)) (**and** n1 (**tf** n2 gnd))) (n1 (**not** n1)))

Figure 3: Subcircuit with a corresponding logic model specification.

by a loop with an even number of inversions. Figure 2 shows examples of subcircuits whose output state is M.

Because of the introduction of the M and E state, the basic operators of boolean algebra are not sufficient to describe the function F(X) and must be extended. We will illustrate the extension of the AND operation:

conjunction (AND)

IF $x_i = E$ $(1 \leq i \leq N)$ THEN $x_1 \wedge x_2 \ldots \wedge x_N = E$
ELSE IF $x_i = 0$ $(1 \leq i \leq N)$ THEN $x_1 \wedge x_2 \ldots \wedge x_N = 0$
ELSE IF $x_i = 1$ $(i = 1 \ldots N)$ THEN $x_1 \wedge x_2 \ldots \wedge x_N - 1$
OTHERWISE $x_1 \wedge x_2 \ldots, \wedge x_N = M$

The definitions of the operators OR, NOT and SELECT is given in [19]. The SELECT operator is necessary as it is the only operation which can evaluate to E or M when all its operands are 0 or 1.

In order to be able to specify the function F(X) of *generic classes* of subcircuits, an additional construct in the rule base language is necessary: the *transmission function* **tf** between two nodes in a subcircuit is a boolean function of the subcircuit inputs which evaluates to 1 if a conducting transistor path between the two nodes exists. The transmission function avoids that logic equations have to be specified in the rule-base for each instance of a specific type of gate (for example for all pull up and pull down configurations in a static CMOS gate).

In figure 3, an example of a subcircuit is shown together with the specification of the logic model F(X), as specified for a specific design style in the LEXTOC rule base for CMOS design. It describes the function of an RS type flip flop build out of two CMOS nand gates.

Automatic methods for determing logic expressions of MOS subcircuits have been presented recently [17,18] and should be investigated for avoiding the need for manual specification of logic expressions in the rule base.

3 Timing constraints in synchronous circuits

Timing constraints in synchronous circuits follow from the presence of subcircuits which can have an M state at the output during certain phases of the global clock period. By definition of the M state, the actual logic value at the subcircuit output is determined by the preceding *sequence* of events at the subcircuit inputs. The purpose of the timing constraints is to formulate conditions under which this sequence of events can or will lead to an incorrect logic value for the M state at the subcircuit output. First the notation will be presented to denote clocks and clock waveforms. Next, we introduce the concept of 'settling time' of a node. Finally, the timing constraints are discussed and formulated in terms of the settling times of the nodes in the circuit.

To the clock inputs of a synchronous circuit, periodic waveforms are applied. The following notation is used:

Clocks: $nc_i \in N2, \quad i = 1 \ldots q$
Logic state of the clocks: $D(v(nc_i)) = xc_i \in \{0,1\}$
$XC = [xc_1 \ \ldots \ xc_q]^T$
Clock waveform: $XC_1 \ \ldots \ XC_p$
The time point at which $XC_{i-1} \rightarrow XC_i$ is called tc_i ($tc_1 = 0$)

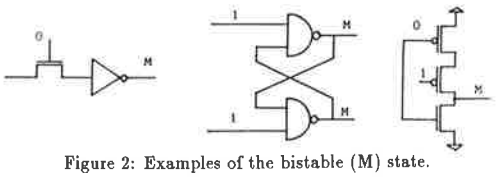


Figure 2: Examples of the bistable (M) state.

Each maximum time interval (tc_i, tc_{i+1}) during which the clocks are stable at the value XC_i is called a clock phase.

For an arbitrary node n in the circuit, the settling times during a certain clock phase are defined as the latest possible time points at which a signal transition can take place after the beginning of the clock phase, under the assumption that the states of the clock would be kept at the value of this phase. The following notation is used:

$U(n)$: latest 0 to 1 transition time at node n
$D(n)$: latest 1 to 0 transition time at node n (1)

The settling times of internal nodes in the circuit are determined by the settling times of the primary inputs. The timing constraints will be expressed in terms of these settling times. In the following section, the algorithm to calculate $U(n)$ and $D(n)$ for each node in the circuit and for each clock phase will be discussed.

A timing fault in a circuit occurs when the M state at a certain node and during a certain clock phase does not store the intended logic value. This condition imposes constraints on the settling times at certain nodes in the circuit.

In order to formulate the constraints, it must be observed that for each subcircuit, during a certain clock phase XC_i, a number of inputs $\hat{n}$ have no determined logic state, while the others are determined by the states of the clocks. If all possible combination of states $\hat{X}$ are applied at the inputs $\hat{n}$, the set Θ_i of potential states (0,1,E or M) at the output of the subcircuit during the i-th clock phase is obtained:

$\exists \ \hat{X} : \ F(X) = 0 \Leftrightarrow 0 \in \Theta_i$
$\exists \ \hat{X} : \ F(X) = 1 \Leftrightarrow 1 \in \Theta_i$
$\exists \ \hat{X} : \ F(X) = M \Leftrightarrow M \in \Theta_i$

The following timing constraints can then be formulated, which must hold for each subcircuit and each clock phase:

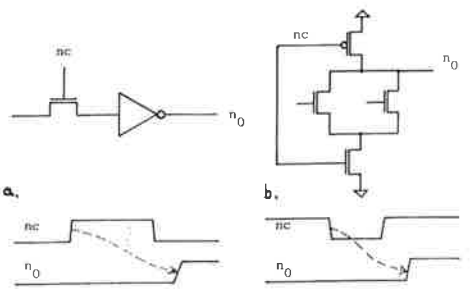


Figure 4: Timing faults covered by the first timing constraint.

Timing constraint 1.

If $M \in \Theta_i \wedge \Theta_{i-1} \setminus \Theta_i \neq \{\}$ **Then** $\begin{matrix} 1 \in \Theta_{i-1} \setminus \Theta_i \Rightarrow U(n_0) < tc_i \\ 0 \in \Theta_{i-1} \setminus \Theta_i \Rightarrow D(n_0) < tc_i \end{matrix}$

If a determined boolean state (1 or 0) can exist during a certain clock phase but not during the next clock phase, it must have settled before the start of the next phase. This is based on the consideration that this state may have to be stored by the M state during the second phase. Figure 4 shows two examples of timing faults which can occur when the constraints are not satisfied. In the first case (4a), the output of the latch has not reached the intended state before the latch 'closes'. In the second case (4b), the output of the dynamic gate is not yet precharged before the end of the precharging phase of the clock period.

Timing constraint 2.

If $M \in \Theta_i \ \wedge \ \Theta_i \setminus \{M\} \neq \{\}$ Then $\forall \ \hat{X}_1, \hat{X}_2$ *during* XC_i

$$\begin{cases} F(X_1) \neq M \\ F(X_2) = M \\ X_2 = X_1^{\bar{t}} \\ x_{1,t} = 0 \end{cases} \Rightarrow U(n_t) < tc_i \qquad \begin{cases} F(X_1) \neq M \\ F(X_2) = M \\ X_2 = X_1^{\bar{t}} \\ x_{1,t} = 1 \end{cases} \Rightarrow D(n_t) < tc_i \ ^{1}$$

If during a certain clock phase both an M state and a determined boolean state are possible at the output, no input transition must take

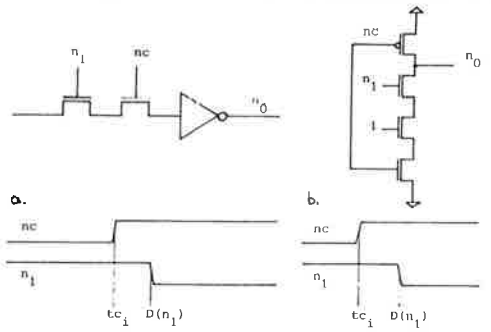


Figure 5: Timing faults covered by the second timing constraint.

place at the subcircuit during this clock cycle that changes the output from the determined boolean state to the M state. This is based on the consideration that the purpose of the M state can be to store the value of the previous clock cycle. Figure 5 shows timing faults which are excluded by the second timing constraint. Signal *nc* is directly influenced by the primary clocks. n_1 is an input signal to the subnetwork. In 5a, there is a time interval during which an unintended state can 'slip' through the latch, because the condition signal A is not stable in time. In the case of 5b, the output of the precharged gate may discharge because the inputs have not all stabilized at the end of the precharging phase of the clock period.

4 Timing analysis of combinational circuits

The goal of timing analysis is to calculate for each node n in the timing model the settling times for a logic high and low signal, respectively denoted as in (1). In a combinational circuit, we assume that for each node n with $|\alpha_{out}(n)| = 0$, the values $U(n)$ and $D(n)$ are given. These values represent the settling times of the inputs of the circuit.

The values of $U(n)$ and $D(n)$ for all other nodes can then be obtained by evaluating all subcircuits in topological order (i.e. a subcircuit is not handled before all the subcircuits that drive it) as follows:

$$U(n_0) = MAX \begin{pmatrix} U(n_i) + \text{ delay } n_i \rightarrow n_0 \text{ with } T_i = 0 \rightarrow 1 \text{ and } T_o = 0 \rightarrow 1 \\ D(n_i) + \text{ delay } n_i \rightarrow n_0 \text{ with } T_i = 1 \rightarrow 0 \text{ and } T_o = 0 \rightarrow 1 \end{pmatrix}$$

$$D(n_0) = MAX \begin{pmatrix} U(n_i) + \text{ delay } n_i \rightarrow n_0 \text{ with } T_i = 0 \rightarrow 1 \text{ and } T_o = 1 \rightarrow 0 \\ D(n_i) + \text{ delay } n_i \rightarrow n_0 \text{ with } T_i = 1 \rightarrow 0 \text{ and } T_o = 1 \rightarrow 0 \end{pmatrix}$$

[1] $X^{\bar{t}}$ is defined in the next section as a vector of boolean values which is the same as vector X with only a different value for entry t

where the maximum is taken over all edges $n_i \rightarrow n_0$ and T_i, T_o denote the input and output transition of the edge. This algorithm is known as the PERT algorithm [16] and used in several existing timing verifiers.

The analysis of synchronous circuits is different from the analysis of combinational circuits because the timing model is no longer an acyclic graph. Therefore, the PERT algorithm can not be used. However, during each clock phase the propagation conditions of a number of edges in the graph are in logic contradiction (see Section 5) with the states of the clocks during this phase. For a synchronous circuit, it can safely be assumed that elimination of these edges from the graph yields an acyclic graph during each clock phase. This is in the assumption that small cycles can be taken up in subnetworks that are described in the rule base. Under this condition, it is possible to repeat the PERT evaluation described in the previous section for each clock phase. Each evaluation gives the worst case settling times $U(n)$ and $D(n)$ for every node n in the circuit, in response to a rising or falling edge of the clock waveforms. The algorithm is summarized below:

REPEAT
  FOR every phase in the clock period **DO**
    Eliminate edges from the SPG with incompatible conditions;
    Determine $U(n)$ and $D(n)$ by the sensitizable critical path algorithm on the remaining graph [15];
    Verify timing constraints
  ENDFOR;
UNTIL convergence: U and D are incremented with exactly one clock period for all nodes.

The advantage of this algorithm lies in the absence of any assumptions about the synchronising actions of the clocks. Signals are not assumed to be stable at latch inputs or outputs at certain time points of the clock period [7]. The algorithm correctly handles signals that propagate through latches during the time interval that these are in 'sample' mode.

5 Compatibility of logic states

Both the timing model and the timing constraints are derived from the logic model of the subcircuits. These subcircuits are defined for a specific design style in the LEXTOC rule base description. It will be shown in this section that another benefit follows from the availability of this logic model in the timing verifier. Consider the following situation: A logic state (0 or 1) can be imposed on an arbitrary number of nodes in the network. If the states are arbitrarily chosen, it is possible that they can never occur simultaneously in the circuit, because of logic incompatibility (e.g. a 0 state may be assigned to both the input and output of an invertor). We will denote a set of (node, logic state) pairs with the symbol Λ. It is then possible to define a boolean function $\varepsilon(\Lambda)$, which determines the compatibility of the set Λ:

$$\varepsilon(\Lambda) = 1 \Leftrightarrow \Lambda \text{ is not contradictory}$$
$$\varepsilon(\Lambda) = 0 \Leftrightarrow \Lambda \text{ is contradictory}$$

An implementation of the function ε is given below, based on the D-algorithm which has been adapted to the set of operators used in the logic model of the subcircuits. Essentially, this algorithm for propagation of logic states in a network of logic operators is similar to event-driven logic simulation at gate level, except for the fact that signals can also propagate from the output of an operator to the input. Below, the evaluation procedure of the operators is given.

- $y = -x$
  1. IF x is known, set y to $\overline{x}$
  2. IF y is known, set x to $\overline{y}$
- $y = x_1 \wedge x_2 \ldots \wedge x_N$
  1. IF $x_i = 1$ for all i, set y to 1
  2. IF y=1 , set x_i to 1 for all i
  3. IF $x_i = 0$ for any i, set y to 0
  4. IF x_j is unknown and,
     $x_i = 1$ for all i≠j and,
     $y = 0$, set x_j to 0

- $y = x_1 \vee x_2 \ldots \vee x_N$
  1. IF $x_i = 0$ for all i, set y to 0
  2. IF y=0 , set x_i to 0 for all i
  3. IF $x_i = 1$ for any i, set y to 1
  4. IF x_j is unknown and,
     $x_i = 0$ for all i≠j and,
     $y = 1$, set x_j to 1
- $y = (s_1, s_2 \ldots, s_N) | (d_1, d_2 \ldots, y_N)$
  IF $s_i = 1$ for any i,
  1. set s_j to 0 for all $j \neq i$
  2. IF d_i is known, set y to d_i
  3. IF y is known, set d_i to y

The procedure determines all necessary implications of logic states at inputs and output of each operator. Logic contradiction is detected when both a 1 and a 0 state are implied at the same node. This causes the function ε to return 0. Note that only the 0 and 1 state are to be considered, as the M state does never cause any implications and an implication of an E state is taken to be equivalent to contradiction.

We will demonstrate the use of the function $\varepsilon(\Lambda)$ with three applications in the timing verifier:

1. Elimination of edges during each clock phase In the timing analysis algorithm, a number of edges can be eliminated from the timing model during each phase of the clock period. Each edge E in the graph has a number of propagation conditions associated to it, denoted with $\Lambda(E)$. Whether an edge is to be eliminated or not can be established by evaluation of the function ε for the propagation conditions of the edge, together with the states of the clocks during that particular clock phase:

$$\begin{cases} \Lambda(E) \\ (nc_i, xc_i) \quad i = 1..q \end{cases}$$

This is in contrast to the method used in LEADOUT [7] where simulation is used to determine the relevant transitions under each clock phase.

2. Elimination of false timing constraints In Section 3, it is explained how timing constraints are derived from the logic model of the subcircuits. In practice this requires the evaluation of the subcircuit expression for all possible input vectors X of each subcircuit during each phase of the clock period. If applied literally, the two rules give rise to many false timing constraints. These false constraints can be eliminated by verifying the logic consistency of each tested input vector by evaluation of the function ε.

An example is given in figure 6. During each phase of the clock period, M is a potential output state of the multiplexor, which gives rise to a number of timing constraints. However, the invertor excludes the M state and hence the timing constraints. This can be detected by application of the function ε to all tested input vectors of subcircuit (ii), to verify whether they are indeed possible.

3. Elimination of false paths

The remaining subgraphs of the causality graph in each clock phase are assumed to be acyclic, as all cycles are assumed to be detected by the rule base. In each such subgraph, the largest sensitizable paths have to be found. However due to the fact that timing verification is signal value independent it can occur that paths found by a PERT analysis can not logically be excited and that the path found is false. In order to avoid this, the logical compatibility for the edges in a path have to be taken into account. In [15] efficient algorithms, as implemented in SLOCOP, which effectively solve these problems are described.

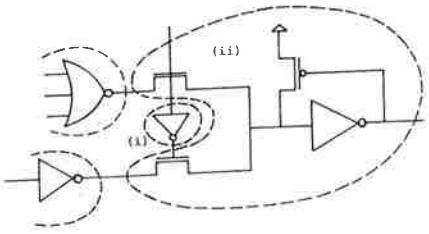


Figure 6: Partitioning of a multiplexor circuit.

6 Conclusions

An alternative view on timing faults in synchronous MOS circuits, and the way these faults can be verified has been presented. To our knowledge, all existing timing verifiers ignore the logic behaviour of the circuitry to be analysed. We showed that the availability of a logic model allows to automatically derive a detailed timing model and the timing constraints which are to be verified. Additional advantages of the logic modeling lie in the possibility to rule out false timing errors and exclude false paths from the analysis. Because of the rule-based subcircuit partitioning, the SLOCOP timing verifier has been used for a wide range of static and dynamic nMOS and CMOS circuit designs.

References

[1] Robert B. Hitchcock, *Timing Verification and the Timing Analysis Program* Proc. of the 19th DA Conf., 1982, pp.594-604.

[2] David E. Wallace, Carlo H. Sequin, *Plug-in Timing Models for an Abstract Timing Verifier*, Proc. 23rd DA Conf. 1986, pp. 683-689.

[3] Seung H. Hwang, Young H. Kim, A.R. Newton, *An accurate delay modeling technique for switch-level timing verification*, 23rd DA Conf., June 29- July 2, 1986, pp. 227-233.

[4] Norman P. Jouppi, *TV: an nMOS Timing Analyzer*, Proceedings of the Third Caltech VLSI Conference, 1983, pp. 72-85.

[5] G. Ditlow, W. Donath, A. Ruehli, *Logic Equations for MOSFET Circuits*, ISCAS-83, IEEE, 1983, pp. 752-755.

[6] *Diagnosis of automata failures: A calculus and a new method*, IBM J. Res. Develop., Oct. 1966, pp. 278-281.

[7] Thomas G. Szymanski, *LEADOUT: A Static Timing Analyzer of MOS Circuits*, Proc. IEEE ICCAD '86, Nov. 1986, pp. 130-133.

[8] Mark D. Matson, Lance A. Glasser, *Macromodeling and optimisation of digital MOS VLSI circuits*, IEEE Transactions on Computer-Aided Design, Vol. CAD-5, No. 4, October 1986, pp. 659-678.

[9] D. Etiemble, V. Adeline, Nguyen H. Duyet, J.C. Ballegeer, *Microcomputer Oriented Algorithms for Delay Evaluation of MOS Gates*, Proceedings of the 21st Design Automation Conf. pp. 358-364.

[10] Zhong L. Mo, Michael R. Lightner, *A Two Parameter Delay Model for Switch Level Simulation*, 1984.

[11] John K. Ousterhout, *Crystal: a Timing Analyzer for nMOS VLSI Circuits*, Proc. Third Caltech VLSI Conf., 1983, pp. 58-69.

[12] John K. Ousterhout, *Switch-Level Delay Models for Digital MOS VLSI*, Proc. 21st DA Conf., 1984, pp. 542-548.

[13] E.Vanden Meersch, L.Claesen, H.De Man, *SLOCOP: a Timing Verification Tool for Synchronous CMOS Logic*, ESSCIRC '86, pp.205-207.

[14] H. De Man, I. Bolsens, E. Vanden Meersch, J. Van Cleynenbreughel, *DIALOG, An Expert Debugging System for MOSVLSI Design*, IEEE Trans. on CAD, CAD-4, No.3, June 1985, pp. 303-311.

[15] J. Benkoski, E.Vanden Meersch, L.Claesen, H.De Man, *Efficient Algorithms for Solving the False Path Problem in Timing Verification*, Proceedings IEEE ICCAD conference, Santa Clara, November 1987.

[16] Shimon Even, *Graph Algorithms*, Computer Science Press, 1979.

[17] R.E.Bryant, *Algorithmic Aspects of Symbolic Switch Network Analysis*, IEEE Trans. Computer-Aided Design of Integrated Circuits, July 1987.

[18] R.E.Bryant, *Boolean Analysis of MOS Circuits*, IEEE Trans. Computer-Aided Design of Integrated Circuits, July 1987.

[19] E. Vanden Meersch, L.Claesen, H.De Man, *Automated Analysis of Timing Faults in Synchronous MOS Circuits - Extended report*, Report ESPRIT1058/D8750, IMEC Leuven Belgium, July 1987.