Computer Science
University of Manchester

Variable Delay Timing Analysis of Logic Circuits

Behnam Bani-Eqbal

Technical Report UMCS-94-12-3

Variable Delay Timing Analysis of Logic Circuits*†

Behnam Bani-Eqbal

Department of Computer Science
University of Manchester
Oxford Road, Manchester, UK.
behnam@cs.man.ac.uk

Abstract

We propose a new simulation algorithm for combinational circuits which accurately models delay variations in the gates. The algorithm can also be used for the timing verification of asynchronous circuits, eg hazard detection, race/oscillation of sequential circuits, etc. The basic idea is to assign a time variable to each signal transition (edge). The gate delays translate to algebraic relations between the variables. As the arrival order of the edges to a gate can be uncertain, so the simulation result of the gate can diverge. Branching signal paths are introduced to deal with this problem. However the simulation process later often does not depend on earlier branching. The algorithm takes advantage of such cases. We believe that this leads to considerable efficiency improvements.

Recent technical reports issued by the Department of Computer Science, Manchester University, are available by anonymous ftp from `ftp.cs.man.ac.uk` in the directory `pub/TR`. The files are stored as PostScript, in compressed form, with the report number as filename. They can also be obtained on WWW via URL `http://www.cs.man.ac.uk/csonly/cstechrep/index.html`. Alternatively, all reports are available by post from The Computer Library, Department of Computer Science, The University, Oxford Road, Manchester M13 9PL, UK.

†This work was supported by the Science and Engineering Research Council Under Grant GR/F 09570.

1 Introduction

In this paper we propose a new simulation algorithm for combinational circuits which accurately models delay variations in the gates. The same algorithms can be used for the timing verification of asynchronous logic circuits, eg hazard detection of combinational circuits, race/oscillation of asynchronous sequential circuits, etc.

In min/max logic simulation [1] gate delays are specified by a range where the output of a gate is treated as unknown. However, it is well known that the simulation results obtained from min/max are often too pessimistic due to the reconvergence in the circuit [1]. It is therefore important to develop algorithms which can accurately model gate delay uncertainty. The recent work in this area closest to our proposal is the Time-Symbolic Simulation work by Ishiura et al [2]. As in that work, we assign a separate variable to each signal transition (edge) in the simulation. The gate delays translate to algebraic relations between the variables. Since the arrival order of the edges at a gate can be uncertain, so the simulation result of that gate can diverge. In [2], branching signal patterns are introduced to deal with this unavoidable problem. However this technique can be improved, as it is often the case that the simulation process further on may not depend on an earlier branching, ie that the some subtrees of the simulation result tree can be identical. We will take advantage of this and introduce graphs of signals. We will show that this can lead to efficiency improvements.

2 Models

The actual delays of gates are not constant due to variations in the manufacturing process. Usually they are specified to lie within a range of values with a minimum and maximum. In the min/max logic simulation of a combinational circuit, the output of a gate is taken as unknown during the min/max range. However the unknown states will build up further in the circuit, resulting in pessimistic analysis. This is because the input to several gates may be fanned out from the same gate and therefore have the same delay element [1]. As in [2], we solve this problem by having a delay variable for each gate. we can have an expression for the precise time of the occurrence of each edge and we then propagate the edges through the circuit. However we encounter our main problem: there is no queue of edges since their time is uncertain. So the conventional discrete event simulation method does not work. The algorithm used in [2] (the T-algorithm) tries all the orderings compatible with the available timing information. This is organised on a gate depth order and results in the branching of the simulation event tree. However nodes in the tree can be identical. Consider the following example:

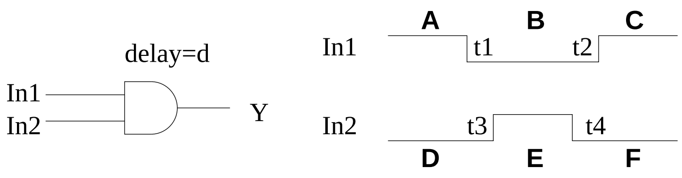


Figure 1.

where the t_i denotes the time of the edge i and lets assume that we know definitely $t_1, t_3 < t_2, t_4$ The T-algorithm produces the following result:

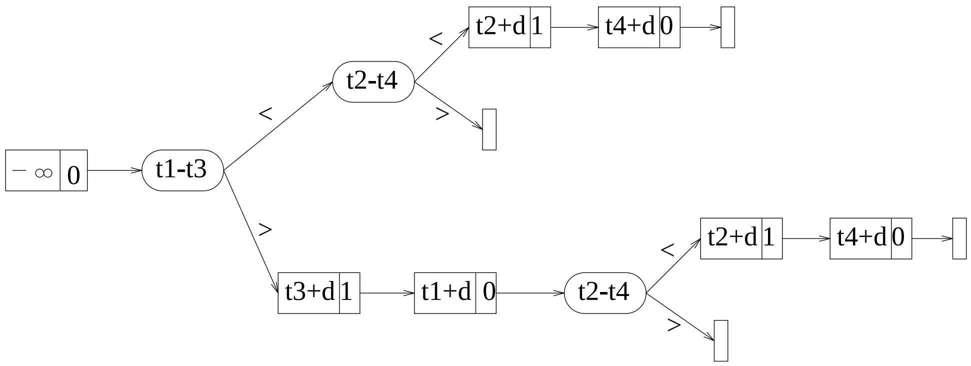


Figure 2.

The notation $[t|l]$ means that the signal value staring from the time t is at the level l, and (E) is a condition node: if $E < 0$ then take the upper branch, otherwise take the lower one. Clearly the graph can be simplified to:

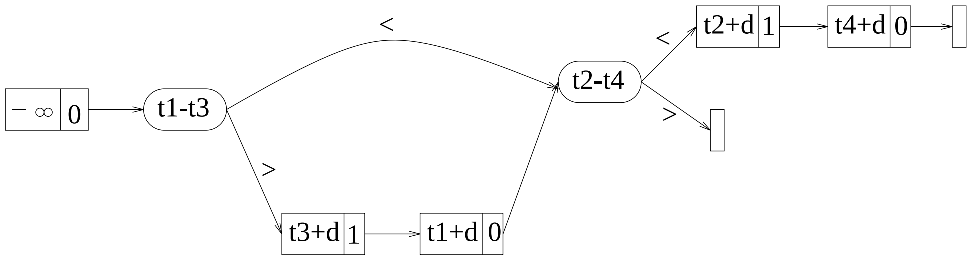


Figure 3.

We can have a more complicated situation. Consider the same circuit where we know only $t_3 < t_2$. The T-algorithm will produce:

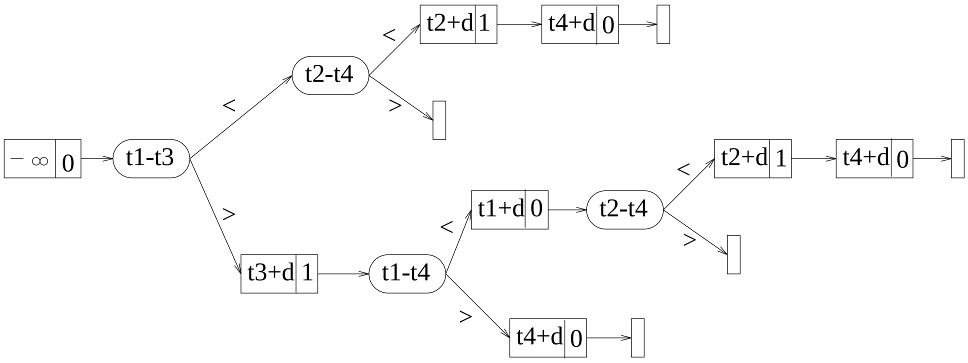


Figure 4.

This time sharing on two branches is possible, but the lower joins after the $t_2 + d, 0$ transition on the common subtree. The shared structure is:

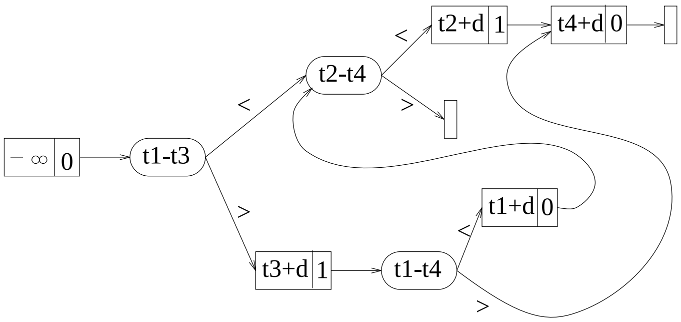


Figure 5.

It should be clear from these examples that the event-tree has some potential for sharing subtrees. The problem is to recognise a sharable subtree. Now any subtree is a sub-run of the simulation from some point on in time, so the sharable trees are identical sub-runs. To maximise the sharing, we must split the simulation process into as many simple steps as possible, so that a major proportion can be shared. The simplest step is one with no edges on the inputs at all! Thus the inputs must be split into its constant signal intervals. Since the edge times are in general uncertain, a simple simulation step takes any two intervals which can overlap, and propagates them to produce the output interval. This output holds for the duration of the overlap deferred by the (uncertain) delay of the gate. With this analysis the sharing of the simulations is automatic, as the output intervals can be immediately connected in a graph (rather than a tree).

We apply this method to the above examples. Let the intervals be marked as in figure 1. Figure 6 is the output for the first example. The notation $AD + d$ above (0) means that the signal is at 0 during the overlap of A, D shifted by d. Figure 7 is the result of the second example.

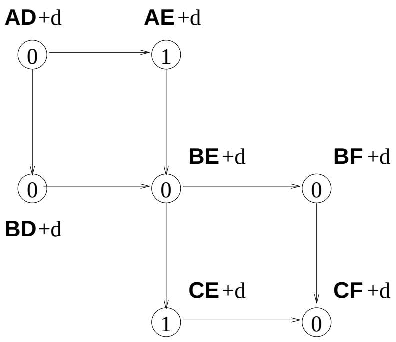


Figure 6.

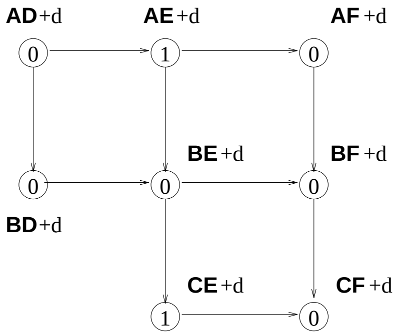


Figure 7.

Notice that the interval CD is missing since $t_3 < t_2$. Our algorithm, the shared time symbolic simulation (STSS) algorithm, keeps signals as graphs like above. A node in the graph represents a time interval, and stores at least three values: the start and the finish time variables and the signal level. As with the T-algorithm, a path through the event graph is a possible run of the circuit simulation. In the next section we will give a more formal presentation of the STSS.

3 Simulation Algorithm

Every wire in the circuit has an associated signal graph (SG). A signal graph is a directed acyclic graph with nodes labelled by a time interval and a signal value of 0 or 1 (this can be extended to three value logic if necessary). The time interval is stored by its start and finish time variables. All information regarding time variables, e.g. min/ max values or linear relations to other variables, are kept by the simulator.

At the start of the simulation, the SG's of the input wires are initialised. A typical input waveform and its SG are:

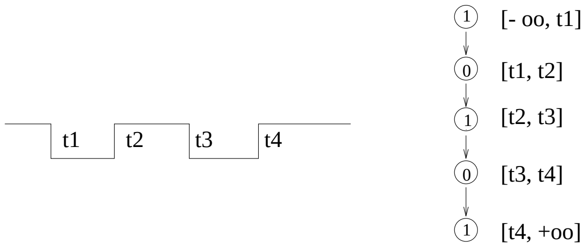


Figure 8.

The t_i are the times of the input edges. The times are specified to the simulator as a system of inequalities of the form $a < t_i < b$ or $a < t_i - t_j < b$. The notation $[t_i, t_j]$ denotes the interval of time from t_i to t_j. The simulator propagates SG's through the circuit on the gate depth level basis. The basic step is as follows:-
Consider a gate with a delay of d and two input wires whose SG's are S_1 and S_2. Let

$I_1 = (t_1, u_1, l_1)$, $I_2 = (t_2, u_2, l_2)$ be two nodes in the SG's. The simulator will generate a new node $I = (t, u, l)$ and record the following properties:

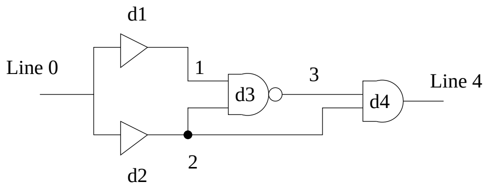
$$t = max(t_1, t_2) + d,\ u = min(u_1, u_2) + d,\ l = \text{gate-function}(l_1, l_2)$$

The generated nodes are connected in the obvious way. The simulator uses linear programming to solve any constraints. For example, the node I above survives only if $t < u$ is feasible. If not the node is not generated at all. A relation like $t < u$ translates into relations like $t_1 < u_2$, etc. which involve previous time variables. Also they are of a similar form. Thus they can be rewritten further. This process terminates with a system of inequalities in the input variables of the form $t_i - t_j < c$. Finally the consistency of these together with the original input set is checked by a specialised linear programming package. We notice that all the inequalities involve only two variables and are of the special form $a < t_i - t_j < b$. We solve such a system by the special algorithms described in [5] or [4].

We have implemented the STSS algorithms in the 'C' language on a SUN SparcStation. We display the signal graphs as traditional signal waveforms. See figure 9 below for an example. Each node (t, u, l) is displayed as a line segment at level l with the maximum extent ie. from the smallest possible value of t to the largest value of u. We shade in the portions of the waveform where the signal can be either 0 or 1. This displays looks exactly like the output of traditional min/max simulators.

4 Examples

We apply our technique to the circuit example described in [2], figure 5. The circuit and our output are shown in figure 9.



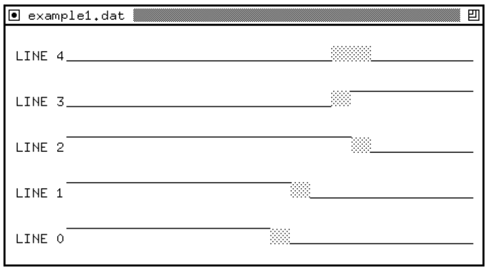


Figure 9.

The delay parameters are $0.8 < d_1 < 1.2$, $3.8 < d_2 < 4.2$, $1.8 < d_3 < 2.2$, $d_4 = 0$ and $0 < t_1 < 1$ where t_1 is the input edge time.

We have also simulated the more difficult examples from [3], namely the Kanehara-A and the Kanehara-O examples, and found our simulator to produce the correct results. The STSS algorithm can also be applied to the hazard detection of asynchronous circuits. This is done by splicing the feedback lines into new input and output lines. The resulting combinational circuit is verified for the input stimuli by feeding the new input wires values from the state transition diagram. We have done this for the T-flip flop in [2] and obtained similar results.

5 Conclusions

We have introduced a new approach to the accurate timing verification of logic circuits and have shown that the method can be used to detect correctly the presence of hazards in such circuits. We have implemented the algorithms and applied it successfully to circuits found in the literature.

Currently the simulator can only handle combinational circuits. However the extension to sequential circuits is straightforward. We are developing the software to deal with feedback lines and state diagrams. We are extending the input parser to read VERILOG hardware designs, so we should be able to test future versions of the simulator on real industrial designs.

References

[1] M. A. Breuer, A. A. Friedman, Diagnosis and Reliable Design of Digital Systems. Computer Science Press, 1976.

[2] N. Ishiura, M. Takahashi, S. Yajima, Time-Symbolic Simulation for Accurate Timing Verification of Asynchronous Behaviour of Logic Circuits. Proceeding of the 26th ACM/IEEE Design Automation Conference 1989.

[3] N. Ishiura, Y. Deguchi, S. Yajima, Coded Time-Symbolic Simulation Using Shared Binary Decision Diagram Proceedings of the 27th ACM/IEEE Design Automation Conference.

[4] B. Aspvall, Y. Shiloach, A Polynomial Algorithm for Solving Systems of Linear Inequalities with Two Variables per Inequality. SIAM journal of Computing,Vol9,No 4, Nov 1980.

[5] V. Pratt, Solving systems of inequalities of the form $t_i - t_j < a$. Unpublished Report, 1978.