Transistor-Level Timing Analysis Using Embedded Simulation

Pawan Kulshreshtha, Robert Palermo, Mohammad Mortazavi, Cyrus Bamji*, Hakan Yalcin

Cadence Design Systems Inc., San Jose, CA 95134, USA
*Canesta Inc., Santa Clara, CA 95054, USA

Abstract

A high accuracy system for transistor-level static timing analysis is presented. Accurate static timing verification requires that individual gate and interconnect delays be accurately calculated. At the sub-micron level, calculating gate and interconnect delays using delay models can result in reduced accuracy. Instead, the proposed method calculates delays through numerical integration using an embedded circuit simulator. It takes into account short circuit current and carefully chooses the set of conditions that results in a tight upper bound of the worst case delay for each gate. Similar repeating transistor configurations of gates in the circuit are automatically identified and a novel interpolation based caching scheme quickly computes gate delays from the delays of similar gates. A tight object code level integration with a commercial high speed transistor-level circuit simulator allows efficient invocation of the simulation.

1. Introduction

Static timing analysis (STA) allows quick and comprehensive timing verification of large circuits. Compared to simulation, STA is much faster and with the exception of false paths, guarantees the identification of the critical paths. Simulation, on the other hand, is impractical for large circuits because simulators are typically slow and finding the right input vectors to excite the critical paths is very difficult.

STA has three main steps: (1) calculating delays of individual gates (and interconnect), (2) adding up the delays of the gates to obtain the path delays for the entire circuit, (3) verifying the circuit constraints by checking whether certain signal transitions occur before/after certain other transitions. This paper deals with the issue of calculating delays of individual gates.

It is possible to approximate the gate delay in terms of transistor sizes, output loading and input slew without calculating the exact output waveform. This idea has led to numerous methods for gate delay calculation using formulas and table look-up methods [1,6]. In effect, these methods offer a closed form solution to the non-linear system of equations describing the gate behavior and are commonly used in delay calculators. However, as features sizes become smaller, existing approximations for this solution become increasingly inaccurate because new considerations, neglected in previous models, must be taken into account. Enhanced models that attempt to rectify this accuracy deficiency become complex, unwieldy and are often heuristic formulas whose results under certain conditions are questionable [2]. For high accuracy and reliability, closed form expression for delay can no longer be used. Therefore, it has become necessary to go back to the method of solving non-linear equations via numerical integration, i.e., using a circuit simulator. Circuit simulation is now a mature field and efficient techniques that trade accuracy for speed have been proposed [3]. One approach that uses simulation for delay calculation is presented in [4]. However, it requires multiple simulations to calculate a single pin-to-pin gate delay. The method proposed here also uses a circuit simulator for delay calculation. Our method has the following key features: (1) Master based simulation to allow simulation of small subcircuits separately and to avoid circuit reloading, (2) Fanout reduction to reduce the number of masters generated, (3) Worst case delay calculation using a single simulation, (4) Tight integration of STA with the simulator to reduce simulation time, and (5) A novel caching scheme to minimize the number of simulations. With the introduction of these features, the result is a high accuracy STA environment with the flexibility approaching that of a model based STA and the speed that is orders of magnitude faster than conventional simulation.

2. Overview

The block diagram of the STA system is shown in Figure 1. After reading the input netlist, the netlist processor creates the internal representation of the design and the master extractor generates *masters*, which are unique channel-connected

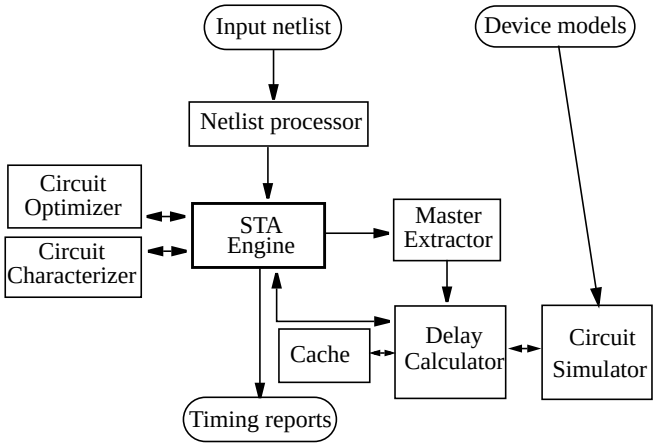


Figure 1. Block diagram of STA with embedded simulator.

components (CCCs). The delay calculator pre-loads these masters into an event-driven transistor-level simulator, EMU2 [3]. To calculate the delay of an instance, the delay calculator sets the parameters of the corresponding master along with its input conditions, and queries the cache, which stores the results of all previous simulations. If a match is found, the cache returns the output waveform. Otherwise, the simulator is run to calculate the output waveform. The caching scheme is based on interpolation of the simulation results of the same master configuration with similar parameters.

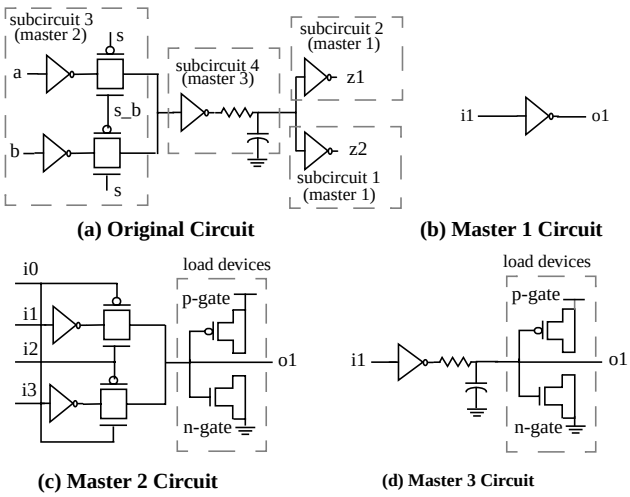
A built-in incremental timing capability, allows quick recalculation of the circuit delays affected by local circuit modifications. This feature enables various applications, including circuit optimization and block characterization, to be linked into this system to form a comprehensive transistor-level timing solution.

3. Subcircuit Extraction

For a small circuit, it is feasible to simulate the entire circuit and calculate its delays. However, it is either impossible or computationally very expensive to simulate large circuits as a whole. To handle large circuits efficiently, our method partitions the circuit into small subcircuits and then simulates each subcircuit individually. The subcircuits are the CCCs extracted from the circuit. The task of subcircuit identification is done by the master extractor which traverses the input netlist and creates a new master each time a new basic subcircuit (CCC) is found. It uses a pattern recognition algorithm [5] to match the same basic subcircuits. Once all the masters are found, they are preloaded into the simulator for efficiency. Figure 2(a) shows an example circuit with its CCCs identified. The masters extracted from this circuit are shown in Figure 2(b), 2(c), and 2(d). Note that RC interconnect networks are part of the masters.

The masters are parameterized in order for each one to represent all the CCCs having the same basic subcircuit. Each master has the following parameterizable attributes: device width (W), device source area (AS), device drain area (AD), device source perimeter (PS), device drain perimeter (PD), wire resistance values and node wire capacitance values. Each time a particular CCC needs to be simulated, first the parameters of the master corresponding to this CCC are set, and then the master is simulated.

In order to decrease the number of masters that need to be created, the loading (fanout) devices for each master output port are reduced to two FET devices (p-gate and n-gate) with equivalent parameters as shown in Figure 2(c) and (d). The equivalent device gate capacitance is the sum of the gate capacitances of all the devices connected to the output node. Hence, the parameters (AS, AD, PS, PD) for each equivalent FET are approximated by the sum of the corresponding parameters of the similar (P or N) fanouts. The length (L_{eq}) and width (W_{eq}) of each equivalent FET are approximated as:



(a) Original Circuit **(b) Master 1 Circuit**

(c) Master 2 Circuit **(d) Master 3 Circuit**

Figure 2. The original circuit and the extracted masters

$$L_{eq} = \sum_{i=1}^{n} L_i / n \qquad and \qquad W_{eq} = \sum_{i=1}^{n} (W_i \times L_i) / L_{eq}$$

4. Subcircuit Simulation

Traditionally, the delay of a gate can be found by simulating the gate with a set of input vectors. However, even for small circuits, this method requires multiple simulations. Our method uses a single simulation to calculate the worst case delay by carefully choosing the input excitations and the internal node initial conditions.

The timing behavior of each subcircuit or gate in the circuit is represented internally by a set of *arcs*, [6] corresponding to the causal relationships between its inputs and outputs. For an arc, all the devices through which the output is charged/ discharged are called *arc devices*, the path from the supply to the arc output node through the arc devices is called the *arc path* and the arc device driven by the arc input is called the *trigger device*. In Figure 3(a), for the arc from the rising transition of i0 to the falling transition of o1, devices m6, m5, m7 and m8 are arc devices, the path GND-n1-n2-o1 is the arc path and the device m5 is the trigger device. The calculation of arc delays for the entire circuit is done in a levelized manner, proceeding from the circuit inputs to its outputs, so that slews are available at all arc inputs. When an arc delay is being calculated, the corresponding master parameters are set in the simulator through an API. These master parameters, including transistor sizes, wire resistances, and node wire capacitances, are obtained from the subcircuit surrounding the arc.

A waveform with a single transition (rise or fall) is applied to the switching input of a subcircuit. For a primary input, a two-point waveform is derived from the input slew. For an intermediate node, the output of the driving gate produces the input waveform. Normally, a single input of a subcircuit is allowed to switch. However, in case of transmission gates, both the FET gate inputs switch for increased accuracy.

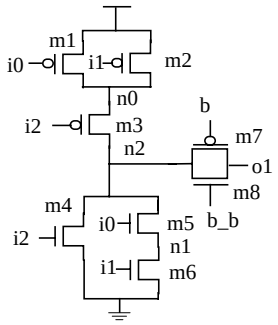
Switching only one FET gate may cause the output to fail to switch completely. Also, one of the transmission gate input waveforms is delayed by the difference in arrival times between two gate inputs.

The fixed voltages on the side inputs and the initial voltages on internal nodes are set to give the worst case delay by maximizing the number of nodes, and hence the capacitance, to be charged/discharged. The algorithm to find the worst case excitation voltages, first sets the default excitations for all the nodes in the master subcircuit to the arc output initial state. It then turns ON all the devices on the arc path and if necessary, overwrites the default initial voltages on internal nodes connected to supply or ground. Finally, it traverses each device on paths from arc output node to supply and ground and turns it ON, if it does not enable a parallel path to supply or ground. For a given arc, the excitation voltages can be found using the following procedure:

/*All the excitation voltages set on master inputs are either fixed voltages (for the side inputs) or input waveforms (for the driving inputs) and the rest are initial voltages (for the internal nodes). */

```
SetArcExcitations(arc) {
  /* Set default excitation */
  for (each node in arc_master_subcircuit)
    set node Excitation = arc_output InitialState
  SetArcDeviceExcitations(arc)
  /* Set the remaining pass-gates to OFF, if possible */
  for (each pass-transistor device in arc_master_subcircuit) {
    if (device_gate_node is not set)
      set device_gate_node Excitation that turns OFF the device
    if (device has complementary_device &&
      complementary_device_gate_node is not set)
      set complementary_device_gate_node Excitation that turns
        OFF the device
  }
  if (arc output is rising) {
    /* Set pullup device excitations first, it will automatically set the
       excitations for the complementary pulldown devices */
    SetNonArcDeviceExcitations(arc, VDD)
    /* Set pulldown device excitations for structures which are
       non-complementary */
    SetNonArcDeviceExcitations(arc, GND)
  } else {
    SetNonArcDeviceExcitations(arc, GND)
    SetNonArcDeviceExcitations(arc, VDD)
  }
}

SetArcDeviceExcitations(arc) {
  for (each arc_device)
    if (device is trigger_device) {
      set device_gate_node Excitation = InputWaveform
      if (device is pass-transistor &&
        device has complementary_device)
        set complementary_device_gate_node Excitation
          = InputWaveform
    } else {
      set device_gate_node Excitation that turns ON the device
      if (device is on arc_path between supply node (vdd or gnd) and
        trigger_device) {
        /* Overwrite default initial voltages */
        set device_source_node Excitation = supply
        set device_drain_node Excitation = supply
      }
    }
}
```



(a) Master subcircuit with single gate

Nodes	Arc i0↑-> o1↓	Arc b↓ -> o1↑
i0	RISE	GND
i1	VDD	VDD
i2	GND	GND
b	GND	FALL
b_b	VDD	RISE
n0	VDD	VDD
n1	GND	GND
n2	VDD	VDD
o1	VDD	GND

(b) Excitation voltages

Figure 3. Example circuits and their their excitation voltages

```
SetNonArcDeviceExcitations(arc, supply_node) {
  for (each path from arc_output_node to supply_node)
    for (each device on path)
      if (device_gate_node is not set)
        if (making the device ON does not make a parallel path ON)
          set device_gate_node Excitation that turns ON the device
        else
          set device_gate_node Excitation that turns OFF the device
}
```

If the master subcircuit has multiple gates connected through a complex pass-gate structure, there may be side paths driving the arc output node. The excitation voltages for nodes in the side path are determined by propagating output node excitation through turned ON pass-gates and the driving devices. This method results in absolute worst case excitations for most circuits. The circuit types supported include static CMOS, pass-gates, latches and domino gates. Figure 3(a) shows the master subcircuit with a single gate and Figure 3(b) shows the excitation voltages of the circuit for two arcs.

Once the master parameters are set, the simulator is called. Its dynamic regionization [7] and event-based algorithm provide fast yet accurate simulations (<5% accuracy and 10-50X faster vs. SPICE). In this work, enhancements have been made to provide for dynamically controllable simulation with a callback mechanism, and master-based simulation to avoid circuit reloading. The simulator's tight integration into the STA environment allows the simulation to be run only for the period long enough to calculate the delay and output slew, thus enhancing the performance. Finally, the delay and slew values are calculated from the input and output waveforms.

5. Caching

The concept of global caching is to save, or cache data relative to specific simulations with the intent of using that data to derive estimated results for other prospective simulations. The goal is to substantially reduce the number of simulations required during execution. The keys to caching are that cache retrieval must be efficient and the retrieved result must be very close to the result that would have occurred if simulation were performed.

A simulation can be considered as a function $S(p_i)$ where p_i

are the various inputs to the simulation with *S* being the result, i.e., output waveform. The input parameters to the simulation are the master subcircuit, the input node excitations, the internal node initial conditions, the device sizes (W, AS, AD, PS, PD), the node capacitances, the wire resistances, and the output node. These input parameters can be classified into two types. First there are the discrete or fixed type parameters such as master subcircuit, nodes, and initial conditions. Simulations that differ on any of these parameters are fundamentally different simulations. The other parameters can be classified as variable parameters. Incremental changes in these parameters result in incremental differences in the simulation results. Thus, the inputs to every simulation can be represented as a *point P*, whose coordinates are the input parameters, and thus having the form (pf_j, pv_k), where the pf_j are the fixed type parameters and the pv_k are the variable type parameters.

For the purpose of caching, input waveforms are represented by three values, those being fall to rise time (t_{fr}), time to threshold (t_{thr}), and threshold offset from first input waveform (t_{off}). On the other hand, output waveforms, which are the results of simulations, are stored in the cache essentially intact. They undergo a reduction that eliminates redundant points along contiguous segments of the piecewise-linear waveform whose slopes are within a pre-set tolerance. This reduction preserves waveform integrity, and typically results in a 50%-75% reduction in waveform size. Using the following definitions for waveform (wf)

$t_{vlow}(wf)$: *wf* low voltage time (normally 10%)

$t_{vthresh}(wf)$: *wf* threshold voltage time (normally 50%)

$t_{vhigh}(wf)$: *wf* high voltage time (normally 90%)

the formulae for the input waveform representation are

$$t_{fr}(wf) = t_{vhigh}(wf) - t_{vlow}(wf),$$
$$t_{thr}(wf) = t_{vthresh}(wf) - min(t_{vlow}(wf), t_{vhigh}(wf)),$$
$$t_{off}(wf) = t_{vthresh}(wf) - t_{vthresh}(wf_{input1})$$

In order for retrieval to be efficient, points are partitioned into multi-dimensional rectangular grids, called point classes. The grid point function $G(P)$ is used to determine the point class that *P* should be placed in.

The result $G(P) = (g(pf_j), g(pv_k))$ is determined as follows. For fixed parameters, $g(pf_j) = pf_j$. For variable parameters, each parameter type has a pre-defined parameter range array, $A[0...N]$, with $A[0] = 0$.

$$\begin{aligned} g(pv_k) &= m && \text{if } A[m] \le pv_k < A[m+1] \text{ and } 0 \le pv_k < A[N] \\ &= N && \text{if } pv_k \ge A[N] \\ &= -g(-pv_k) && \text{if } pv_k < 0 \end{aligned}$$

For example, the range array for capacitance is {0, $1x10^{-14}$, $3.2x10^{-14}$, $1x10^{-13}$, $3.2x10^{-13}$, $1x10^{-12}$, $5x10^{-12}$}). So for capacitance value pv = 75 FF, $g(pv) = 2$.

Before a simulation is performed, the input parameters for the prospective simulation are used to create a point *P*. In order to avoid a simulation, there must exist a point *Q*, in the same point class as *P*, that is very close to *P*. The formula used for calculating *closeness* is a weighted normalized RMS of the differences between the variable coordinates of the points. The formula for closeness between *P* and *Q* is

$$C(P,Q) = (\Sigma(((pv_k - qv_k)\, w_k/r_k)^2) / \Sigma(w_k^2))^{1/2}$$

where r_k, the range size for pv_k, is given by

$$\begin{aligned} r_k &= A[|g(pv_k)| + 1] - A[|g(pv_k)|] && \text{if } |g(pv_k)| < N \\ &= A[N] - A[N-1] && \text{if } |g(pv_k)| = N \end{aligned}$$

and w_k is the relative weighting of the parameter type of pv_k. The weightings of 1.0 for time values, 0.7 for device sizes, 0.7 for capacitances, 0.3 for resistances and 0.1 for areas were determined to yield the best results.

Benchmarking revealed that points must be very close for cached results to be close enough to use in lieu of simulation. Our implementation offers 4 levels of cache usage, with cache level 2, for example, requiring closeness values, $C(P,Q) \le .003$, to result in cached results within 3% of simulation.

Once a close point *Q* is found, the slope (*S*) of the delay function along vector $\overrightarrow{QP}$ needs to be computed. By multiplying this slope *S*, with $\left|\overrightarrow{QP}\right|$, the difference between *delay(P)* and *delay(Q)* can be calculated, i.e. *delay(P)* - *delay(Q)* = $S^*\left|\overrightarrow{QP}\right|$. This delay difference will be henceforth denoted as $\Delta(P,Q)$. *S* can be calculated in terms of the slope of the delay functions on each of the primary axes of the space in which the points reside. If $\vec{V}$ is the vector whose coordinates are these slopes, the expression for the delay difference becomes: $\Delta(P,Q) = \vec{V} \cdot \overrightarrow{QP}$.

Note that $\vec{V}$ points in the direction of maximum slope at *Q*. To determine $\vec{V}$, the cached points near *Q* are used. For each such Q_m near *Q*, the slope of the delay function along $\overrightarrow{Q_mQ}$ can be readily computed:

$$slope = (delay(Q) - delay(Q_m)) / \left|\overrightarrow{Q_mQ}\right|$$

Using these delay slopes, a modified Gram-Schmidt Orthonormalization [8] routine is applied to calculate the slope of the delay function along each of the primary axes, resulting in the slope vector $\vec{V}$.

Once $\vec{V}$ is calculated, the difference in delay is computed, i.e. $\Delta(P,Q) = \vec{V} \cdot \overrightarrow{QP}$. Note that $\Delta(P,Q)$ can be computed even if some of the coordinates of $\vec{V}$ are unknown. Specifically, the coordinates of $\vec{V}$ for the axes for which $\overrightarrow{QP}$ is null, are not needed. Once $\Delta(P,Q)$ has been calculated, the resulting waveform *S(P)* can be derived from the waveform *S(Q)*.

There is a direct relationship between the use of cached results and the reduction of run time for delay calculation on a design. For example, if half of the simulations can be avoided by use of cached results, then there is virtually a 50% run time reduction. Each of the four cache retrieval levels offer different expected accuracy, those being within 1%, 3%, 6%, and 10% of simulation respectively.

6. Experimental Results

Table 1 shows the timing analysis results for the ISCAS-85 benchmarks and three industrial circuits. The gate-level ISCAS-85 benchmarks were mapped to transistor level using a sample library. The remaining circuits are transistor-level custom blocks: a portion of a datapath block (ckt1), an ALU (ckt2), and a large multiplier (ckt3). Transistor count for each circuit is given in the table. A full timing analysis was performed for each circuit. Included in the table are the run times in seconds (RunT) and the longest path delays in nanoseconds (Delay) obtained with caching levels 0 (no caching), 2 and 4. Notice that the run time goes down on average by 40% for caching level 2, with the maximum reduction being 96% for c6288, which has a very regular structure consisting of a 2-D array of full adders. The average run time reduction is 47% for caching level 4, with the maximum reduction being, again, 96% for c6288. The run time reduction is generally higher for larger circuits, indicating the effectiveness of the cache. As for the path delays, the level 2 results are on average within 0.19% of those of level 0, with the maximum difference being 4%. The level 4 results are on average within 0.25% of those of level 0, with the maximum difference being 6%. These results illustrate that the accuracy loss due to caching is minimal.

Experiments were also performed to compare the accuracy of EMU2 against a commercial SPICE simulator. For each circuit, the longest path identified with EMU2 was simulated with SPICE using the worst-case conditions described in Section 5. The EMU2 calculated path delays were found to differ from SPICE by less than 1%. Given the speed advantage of EMU2 over SPICE, it is clear that the proposed approach results in considerable reduction in computational effort with a minimal loss in accuracy.

7. Conclusions

A method has been presented for transistor-level static timing analysis, emphasizing on five salient features: (1) master-based simulation to allow simulation of small subcircuits separately, (2) fanout reduction to reduce the number of masters generated, (3) worst case delay calculation in a single simulation and (4) tight integration with a circuit simulator to reduce simulation time, (5) a caching scheme to reduce the number of simulations. The experimental results suggest that the system can process large circuits with the accuracy and speed required by today's high performance designs. As part of the future work, the following will be investigated:

Circuit	Trans. count	Cache level 0		Cache level 2		Cache level 4	
		RunT	Delay	RunT	Delay	RunT	Delay
c432	784	658	6.92	441	6.91	428	6.91
c499	1364	901	10.98	707	10.95	674	10.96
c880	1802	415	6.33	296	6.34	255	6.28
c1355	2196	511	7.24	389	7.22	300	7.20
c1908	3878	863	9.13	566	9.13	470	9.10
c2670	5684	1259	10.80	771	10.84	598	10.86
c3540	7822	1668	12.99	1062	12.91	826	12.87
c5315	11308	2504	12.22	1595	12.20	1222	12.30
c6288	10112	2756	32.78	107	32.73	97	32.73
c7552	15512	3530	7.65	2583	7.66	1973	7.64
ckt1	7973	2750	85.74	2138	89.29	2018	90.50
ckt2	10527	4930	30.70	3471	30.57	2774	30.57
ckt3	29556	45262	9.55	9892	9.53	9454	9.52
Avg. diff. from cache level 0		-	-	-40%	0.19%	-47%	0.25%

Table 1: Timing analysis results with different caching levels.

- RC interconnect reduction techniques to reduce the RC interconnect networks to a simpler model [9]. This will further reduce the number of masters extracted and will also make simulations more efficient.
- Support masters with multiple CCCs, using user specified patterns for special circuits with feedback.

Acknowledgements

The authors would like to thank Prof. Karem Sakallah of University of Michigan for valuable discussions and contributions.

References

[1] J. Ousterhout, "A Switch-Level Timing Verifier for Digital MOS VLSI", IEEE Trans. on CAD, July 1985, pp. 336-349.

[2] A. Hirata, H. Onodera, K. Tamaru, "Proposal of a Timing Model for CMOS Logic Gates Driving a CRC π Load", Proc. of ICCAD, November 1998, pp. 537-544.

[3] B. Ackland, R. Clark, "Event-EMU: An Event Driven Timing Simulator for MOS VLSI Circuits", Proc. of ICCAD, Nov. 1989, pp. 80-83.

[4] V. Rao, J. Soreff, T. Brodnax, R. Mains, "EinsTLT: Transistor Level Timing with EinsTimer", Proc. of Int. Workshop on Timing Issues in the Spec. and Syn. of Digital Systems (TAU), 1999, pp. 1-6.

[5] M. Ohlrich, C. Ebeling, E. Ginting, L. Sather, "SubGemini: Identifiying Subcircuits using a Fast Subgraph Isomorhism Algorithm", Proc. of IEEE/ACM DAC, 1993, pp. 31-37.

[6] J. Cherry, "A CMOS Timing Analyzer", Proc. of IEEE/ACM DAC, 1988, pp. 148-153.

[7] M. L. Yu, B. Ackland, "VLSI Timing Simulation with Selective Dynamic Regionization", 1994 ACM 0-89791-690-5/94/0011, pp. 195-199.

[8] G. Strang, "Introduction to Applied Mathematics", Wellesley-Cambridge Press., 1986.

[9] F. Dartu, L.T. Pileggi, "TETA: Transistor-Level Engine for Timing Analysis", Proc.of IEEE/ACM DAC, 1998, pp. 595-598.