ACCURATE AUTOMATIC TIMING CHARACTERIZATION OF STATIC CMOS LIBRARIES

by

Sunil P. Khatri, Alberto Sangiovanni-Vincentelli
And Robert K. Brayton

Memorandum No. UCB/ERL M98/58

10 October 1998

COVER

ACCURATE AUTOMATIC TIMING CHARACTERIZATION OF STATIC CMOS LIBRARIES

by

Sunil P. Khatri, Alberto Sangiovanni-Vincentelli
And Robert K. Brayton

Memorandum No. UCB/ERL M98/58

10 October 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Accurate Automatic Timing Characterization of Static CMOS Libraries

Sunil P. Khatri (linus@ic.eecs.berkeley.edu) *
Alberto Sangiovanni-Vincentelli (alberto@ic.eecs.berkeley.edu) *
Robert K. Brayton (brayton@ic.eecs.berkeley.edu) *

10 October 1988

Abstract

For accurate timing analysis of high performance digital designs, it is important to use a gate delay model which can capture delays accurately and efficiently. Traditional gate delay models are expected to be inadequate for this purpose, since they typically assign a separate delay or range of delays per gate pin. However, in reality, the state of the other pins of a gate strongly affects the delay of the gate. This effect has not been efficiently accounted for to date.

In the *vector-pair* based gate delay model, each vector pair which causes the gate output to change is assigned a unique delay value. This delay model is powerful, and can be easily retrofitted into existing timing tools. However, the problem with this gate delay model is that there can be $N^{on} \cdot N^{off}$ unique gate delays, where N^{on} is the cardinality of the on-set of the gate function, and N^{off} is that of the off-set.

In our scheme, we use the vector-pair based delay model. Vector pairs that give rise to similar delay values are grouped together so as to reduce the total number of unique delay values required.

Using this idea, we present a technique to automatically and efficiently characterize a cell library using the above delay model. The input to our method is a transistor level net-list of all the gates of a static CMOS cell library. From each such cell, output transition time estimates are generated for each vector pair, after an analysis of the transistor net-list. We then group together vector pairs with similar delays, so that the total number of unique delay values is reduced. Vector pairs whose estimates are within 20% of each other are grouped together. Preliminary experiments on a set of common library cells show that the actual delays of the vector pairs in each group are usually within 20% of each other, with a low standard deviation. The grouping gives rise to an average reduction of 66% in the number of unique delay values required to characterize the gate.

Our technique has direct applications for library-less synthesis or constant-delay synthesis techniques that are currently being proposed, as well as for accurate timing analysis.

1 Introduction

The computation of delay for combinational circuits is a well-studied problem which, until recently, was considered solved. Efficient exact methods have been devised for computing the delays of acyclic combinational circuits ([1], [2], [3], [4] among others).

Traditional gate delay models have either assigned a fixed delay to a gate output, or a range of delays in some interval $[d_{min}, d_{max}]$. Some gate delay models allow each pin to have a unique delay value or range. However, with the feature sizes of integrated circuits shrinking to sub-micron levels, it is now commonly accepted that existing methods

*CAD Research Group, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720. This research was funded under the Semiconductor Research Corporation Grant SRC-324-040.

for modeling delays are inadequate for deep sub-micron circuits (See, for instance, [5], [6] and other related papers in the TAU '97 Workshop). The reasons for this can be attributed to:

- First, design at higher clock speeds requires more accurate modeling of circuit delay, therefore, more sophisticated gate delay models are needed. Such an accuracy cannot be prvided by the pin-delay models that are commonly utilized today.
- Second, circuit effects such as the presence or absence of transitions on other inputs of a gate significantly affect the output delay, which is not captured in traditional gate delay models.
- With layout feature sizes in the deep sub-micron region, the capacitance between a wire and its minimally spaced neighbors is a significant fraction of the total capacitance of the wire. This causes the delay of the wire to be strongly affected by the presence or absence of transitions on its neighboring wires. To accurately estimate delay variations due to the effect of capacitive coupling between neighboring wires (also referred to as *cross-talk*), highly accurate delay information is needed.

There has been some prior work in devising more accurate delay models which are input vector pair dependent [7], [8]. However, such a vector pair based delay model can result in a large number of unique delay values. In this paper, our gate delay model is a vector pair based model as well. Our main contribution is a method for accurate and fast delay estimation for each vector pair. Based on this analysis, vector pairs with estimated delay values within a fixed percentage from each other are grouped together. This gives rise to a highly accurate delay model, while controlling the number of unique delay values required.

Our scheme is implemented in a static CMOS cell library characterization tool. Given the transistor level net-lists (represented in SPICE [9]) of the cells in a library, the tool calculates delay estimates of vector pairs of each cell. Next it performs the grouping of vector pairs with similar delays. Following this, it constructs a SPICE input waveform corresponding to a few of the vector pairs from each group. SPICE is now called on each of the library cells, and the output delays are extracted and reported.

An accurate hands-free cell library characterization scheme like ours has many applications.

- For high performance digital designs with deep sub-micron feature sizes, it is felt that existing delay models are inadequate [5, 6]. With our accurate delay model and the automatic characterization tool, the accuracy of timing analysis tools will increase significantly.
- Our scheme can form the inner loop of a library-free synthesis algorithm such as [10]. Also, the constant delay synthesis algorithm suggested in [11] can benefit from an on-the-fly cell characterization scheme such as ours.
- It can be used for accurate timing characterization of any cell library. By grouping together vector pairs with similar delays, the resulting timing information can be compactly represented as well.

The paper is organized as follows. Section 2 discusses some previous work in this area. In Section 3 we describe our method to do automatic cell library characterization. Section 4 presents experimental results, while section 5 summarizes our work, and discusses avenues for future research.

2 Previous Work

The gate delay models employed by conventional timing analysis methods have been somewhat simplistic. The most primitive of these is the *unit delay* model, where each gate is assumed to have a delay of one time unit. This was supplanted by the *fixed* delay model, where gates can have different but constant delays. Both of these models are too coarse as they do not allow for variations in gate delay. The *min-max* delay model addresses this problem to some

degree by allowing variations in the delay of a gate: transitions at a gate input are reflected at its output with a delay in the range $[d_{min}, d_{max}]$. In the *pin-delay* model, different input-to-output delays can be specified for different input pins of a gate. Neither the min-max model nor the pin-delay model are powerful enough for an accurate analysis because they can not express the dependency of gate delay on specific input values, or on sequences of input values.

Motivated by this need for an accurate gate delay model, the authors of [7] introduced a delay model in which different vector pairs are assigned distinct delays. However, this model requires a significantly larger number of transitions to model delays. For a 3-input NAND gate, the authors of [7] state that 109 transitions would be required in their gate model.

Later, [8] introduced another vector-pair based gate delay model. This delay model is based on the fact that in practice gate delays have a strong dependence on the input vectors applied. This is illustrated by means of the static CMOS NAND gate shown in Figure 1. For this gate, under the input sequence $a = b = 1 \rightarrow a = 1, b = 0$, let the gate delay be $t^r_{11\rightarrow 10}$. The superscript r (f) indicates that the output is rising (falling), and the subscript represents the applied input vector sequence. Similarly, under the input sequence $a = b = 1 \rightarrow a = 0, b = 0$, let the gate delay be $t^r_{11\rightarrow 00}$. In the second case, the delay of the gate is smaller because, after the inputs transition, both transistors T1 and T2 are on, effectively doubling the current to charge the capacitance of node *out*. In the first case, only transistor T2 is on after input b has switched, hence the rising transition is slower.

The pin delay model does not distinguish between these two input sequences, and hence is not accurate enough for high performance designs. Some delay models allow the specification of separate rising and falling delays for the gate output. This is a useful property to model, but in the absence of input sequence dependence of the output delay, this model alone is not very useful.

Consider any gate whose transistor level representation has n paths from the output node to *vdd*, and m paths from the output node to *gnd*. Assuming that all paths to *vdd* and *gnd* have the same effective size of transistors, the scheme of [8] will give rise to n distinct rising delays, and m distinct falling delays. However, as we will see in Section 3, a more general and accurate grouping of delays is possible if the analysis takes into account the values of internal parasitic capacitances of the gate.

3 Our Approach

As mentioned earlier, we adopt the *vector-pair* based delay model that was used in [7] and [8]. Consider a gate F with output f and n inputs $x_1, x_2, \cdots, x_n$. The function f is a mapping $f : B^n \rightarrow B$. So each *vertex* or *vector* in the input space B^n is mapped to either the "1" value or the "0" value. The *on-set* V^1_F of f is the set of vectors which evaluate to the "1" value, while the *off-set* V^0_F consists of vectors which evaluate to "0".

A *vector pair* p of f is defined as a pair of vectors, one of which is in the on-set of f, and the other in the off-set. Hence there are $2 \cdot |V^1| \cdot |V^0|$ vector pairs for f. If $p =< v_1, v_2 >$, then the application of vector v_1 followed by vector v_2 will result in a change in the value of f. The *delay of the vector pair p* is defined as the delay for the value of f to change when vector v_2 is applied after vector v_1.

A set $g = \{p_1, p_2, \cdots, p_k\}$ of vector pairs is called a *group* of vector pairs. Vector pairs are grouped together if they all share a property of interest.

3.1 Problem Definition

Vector-pair based static CMOS cell library characterization can be defined as follows: We are given a transistor level description of a static CMOS cell library L with gates $F_1, F_2, \cdots F_m$. For each $F_i \in L$, construct delay groups $g_1, g_2, \cdots g_q$ such that for each group g_j, the delay of its constituent vector pairs p_k are within some percentage P of each other. For each delay group, report the average delay of its constituent vector pairs.

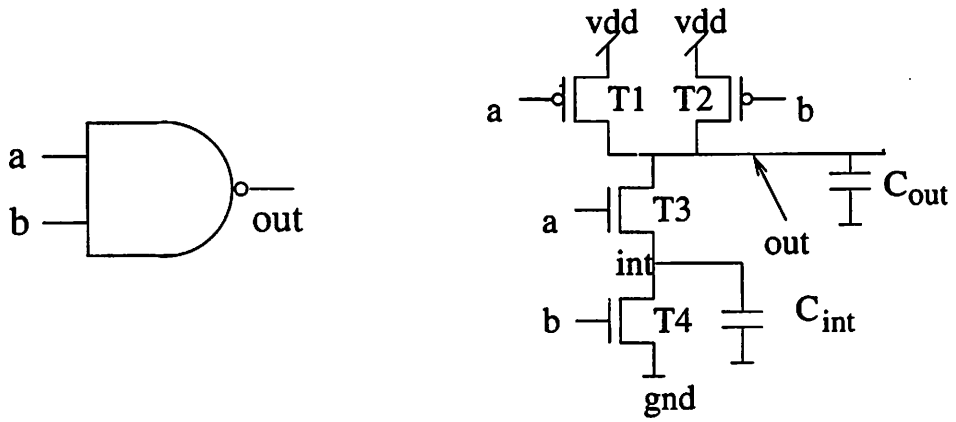


Figure 1: NAND gate: Transistor Level Description

3.2 Details of the Scheme

The outline of our procedure for vector-pair based static CMOS cell characterization is shown in Figure 2. The input to the procedure is a transistor-level SPICE [9] net-list of the cell being analyzed. This net-list has transistor connectivity and size information. The individual steps of the procedure are detailed below.

As a working example, we will consider the 2-input nand gate shown in Figure 1. Here we assume that each of the 4 transistors have $w = 0.1\mu$m and $l = 0.1\mu$m, and are implemented in a 0.1μm process technology, with $Vdd = 1.2$ volts. The output is loaded by a capacitance of value 2.5×10^{-15} Farads. All input signals are modeled as ramp signals with rise and fall times of 40 ps. Detailed analysis and results for this example are shown in Table 1, and will be used to illustrate our procedure.

3.2.1 Logical Analysis of the Cell

The first step is to logically analyze the transistor-level net-list of the gate. For each node n in the net-list, the logical condition under which the node is driven to a "1" and "0" value is computed, and called V_n^0 and V_n^1 respectively. Note that for the output node, $V^1 \cup V^0 = 1$, but this is not necessarily the case for internal nodes of the gate.

Next, we construct vector pairs for the cell. Each vector pair p_i that causes a transition in the output is listed. There are a total of $2 \cdot |V^1| \cdot |V^0|$ such pairs. Since library cells have a small number of inputs, this number is a manageable one.

In our example, V_{int}^0 is computed by listing all paths to *gnd*. This simply gives $V_{int}^0 = b$. Similarly, V_{int}^1 is computed by listing paths to *vdd*. There are two such paths, one through T1 and T3, and the other through T2 and T3. For T1 to be conducting, $a = 0$ is required, since T1 is a PMOS transistor. For T3 to conduct $a = 1$ is required, and for T2 to conduct, $b = 0$. Hence $V_{int}^1 = (a \cdot \overline{a}) + (a \cdot \overline{b}) = a \cdot \overline{b}$.

There are 6 vector pairs for the NAND gate of our example. These are $< 00,11 >$, $< 01,11 >$, $< 10,11 >$, $< 11,00 >$, $< 11,01 >$ and $< 11,10 >$. These are listed in column 2 of Table 1.

3.2.2 Delay Estimation of the Cell

To estimate the delay D_{eff} of a vector pair, we first compute the *effective resistance* R_{eff} and *effective capacitance* C_{eff} for the vector pair. Then D_{eff} is estimated as the product of R_{eff} and C_{eff}.

Computing Effective Resistance R_{eff}: For an NMOS transistor with $w = 0.1\mu$m and $l = 0.1\mu$m, or a PMOS transistor with $w = 0.2\mu$m and $l = 0.1\mu$m, we determined from SPICE that the effective on-resistance was approximately 10K Ω. This number scales inversely with w.

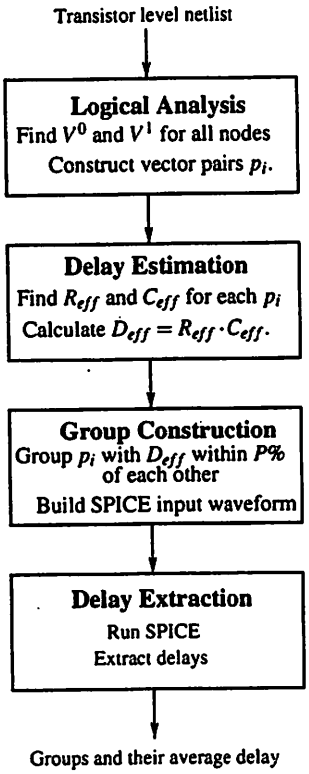


Figure 2: Vector-pair based Cell Library Characterization

Consider the vector v. The effective resistance for this vector is determined by considering each conducting transistor to be a resistor, with a value determined as above. The equivalent resistance to *vdd* or *gnd* is the effective resistance for v.

Given a vector pair $p =< v_1, v_2 >$, we first find the effective resistances R_{v_1} and R_{v_2} for both its constituent vectors. The effective resistance for p is R_{v_2}. In case the equivalent resistor network of R_{v_1} contained two or more series connected transistors, we increase the effective resistance by 50%. This is to account for the fact that series connected transistor structures typically have a longer turn-off time, effectively increasing the output delay.

The values of effective resistance for each vector pair in the NAND2 example are listed in column 3 of Table 1.

Computing Effective Capacitance C_{eff}: For a PMOS or NMOS transistor with $w = 0.1\mu$m and $l = 0.1\mu$m, we determined from SPICE that the source and drain diffusion capacitances are approximately 90 $\times 10^{-18}$ F. In our method to estimate the effective capacitance, we assume that this number scales with w.

Group	Vector Pair	R_{eff} (Ω)	C_{eff} ($\times 10^{-18}$ F)	D_{eff} (ps)	Actual Delay (ps)
1	$< 11,01 >$	30K	2770	83.1	60.43
1	$< 11,10 >$	30K	2950	88.5	65.80
2	$< 00,11 >$	20K	2950	59.0	58.68
2	$< 01,11 >$	20K	2770	55.4	49.42
2	$< 10,11 >$	20K	2950	59.0	53.24
3	$< 11,00 >$	15K	2770	41.6	37.32

Table 1: Working Example - NAND2 cell

The effective capacitance C_{eff} of a vector pair p is computed as follows. For both the input vectors in a vector pair $p =< v_1, v_2 >$, we compute the logic value of each internal parasitic node n. The logic functions V_n^0 and V_n^1 extracted in Section 3.2.1 are used for this. If $V_n^0(v) = 1$ ($V_n^0(v) = 1$) then the node is driven to a logic "1" ("0"). If $V_n^0(v) = 0$ and $V_n^1(v) = 0$, then the node is in a high-impedance state.

If n has the same value (i.e. "0", "1" or high-impedance) under both vectors v_1 and v_2, then the parasitic node does not transition, and hence has no contribution to the effective capacitance of p. If node n is in a high-impedance state under vector v_2, it does not contribute to the effective capacitance of p either. This is because it does not change its value when the vector pair p is applied. In all other cases the parasitic capacitance of n is added to the effective capacitance for p.

In the NAND gate example of Table 1, consider the vector pair $< 00, 11 >$. The node *int* is at a high-impedance state when $a = b = 0$, and is driven to a "0" value when $a = b = 1$. However, just before the second vector is applied, the increasing gate voltage of transistors $T3$ and $T4$ capacitively couples into *int*, instantaneously increasing its voltage while it is still in a high-impedance state. This coupled voltage must be discharged by the $a = b = 1$ vector. Hence the parasitic capacitance of *int* is added to the effective capacitance for this vector pair.

From the values of R_{eff} and C_{eff}, the effective delay for vector pair p is computed as $D_{eff} = R_{eff} \cdot C_{eff}$.

Using the values of internal and output capacitances as described earlier, the values of effective capacitance for each vector pair in the NAND2 example are listed in column 4 of Table 1.

3.2.3 Construction of Groups

Based on the D_{eff} estimates of each vector pair p, we construct delay groups. First, we sort the vector pairs in decreasing order of D_{eff}. Now vectors that are within $P\%$ of the maximum delay estimate are assigned to the first group. This is repeated among the remaining vectors, until each vector pair is in a group. It is possible to have delay groups with a single vector pair in it. For examples with three or more inputs, P is chosen to be 20%, in order to keep the number of groups small. For two-input examples, P was chosen to be 10%.

Note that it is possible that one vector pair in some group causes the output to rise, while another causes the output to fall. Our scheme does not distinguish between falling and rising delays; as long as two delays are within $P\%$ of each other, they will belong to the same group.

Based on the groups created, a SPICE input waveform file is written, which is used in the next step. For each group, a small number of vector pairs are written to this file.

Using the values of D_{eff} from column 5 of Table 1 and $P = 10\%$, we get three delay groups, which are shown in column 1 of the same table.

3.2.4 Back-end Tasks

Finally, SPICE is run on the transistor level net-list of the gate. The SPICE input waveforms from the previous step are used. After this, the output delays are extracted automatically and reported for each group.

The delay data which is reported for the gate can be compactly stored using a variation of the concept of Reduced Ordered Binary Decision Diagrams (ROBDDs) [12]. This step was not implemented in this paper.

In order to validate our scheme, we perform the spice simulation for all the vector pairs. The grouped vector pairs are analyzed for the mean and variance of their delay, to provide a figure of effectiveness for the scheme. It is expected that when the groups are listed in decreasing order of estimated delays, the actual average delays of the groups are in decreasing order as well, and the standard deviation of each group is small.

The results obtained by running SPICE on each vector pair for the NAND2 example are shown in column 6 of Table 1.

4 Experimental Results

We implemented the ideas described in this paper to characterize the delays of a static CMOS standard cell library. Our code reads in the SPICE [9] net-list containing all the cells of the library, and performs logical analysis, delay estimation, grouping and delay extraction as described in Section 3. About 750 lines of *perl* code was written to perform all the above tasks. The experiments were run on a 200 MHz Pentium MMX based machine running the Linux operating system.

The static CMOS cells we used in our experiment are shown in Figure 3. All transistors of all cells had a length $l = 0.1\mu$m. The width of each transistor is shown in Figure 3. The particular ratioing of transistors chosen results in approximately equal rise and fall delays for each gate, and is typically used in industrial cell libraries.

The mean and standard deviation of the rising and falling delays for each pin in the pin delay model were also computed for the above library . Althought these are not reported here for brevity, it is observed that for gates with larger numbers of inputs (especially NAND4, NOR4, and NOR3), the standard deviations of some pin to output delays were almost half the value of the mean delay. This indicates that the pin delay model is not very effective for high performance designs.

Characterization results are described in Tables 2 through 9. For these tables, each row corresponds to a delay group. The cardinality of each group is reported on the last column. The first column reports the average estimated delay for the vector pairs corresponding to that row. These delays are within $P\%$ of each other by design. The second column represents the standard deviation of the estimated delays of the corresponding group. The third column represents the percentage maximum variation of any estimated delay in a group from the group mean. The fourth column reports the actual delay for that group, averaged over all its constituent vector pairs. The fifth column represents the percentage maximum variation of any actual delay in a group from the group mean. The sixth column represents the standard deviation of the actual delays of vector pairs constituting any group.

In order to keep the number of groups reasonably small, a larger P value was used for cells with three or more inputs. For two-input cells, P was chosen to be 10%, and for other cells, P was 20%. As a result, we find that group sizes are larger for examples with larger numbers of inputs.

For Tables 2 and 5, the value of P that was used was 10%. For these tables, we note that the maximum variation of the actual delay of any vector pair from the group mean is always less than 10%. In fact, the maximum variation is usually much less than 10%, because the maximum variation of the estimated delay of any vector pair from the group estimated mean is usually small. For Tables 3, 4, 6, 7, 8 and 9, the value of P is 20%. Again we notice that the maximum variation of the delay of any vector pair from the group mean is usually well within 20%. This suggests that our estimated delay is a good predictor of the actual delay.

In all tables, we list groups in decreasing order of their average estimated delay value. We note that the average actual delays for all the tables also follow the same decreasing order, again suggesting that our delay estimator is a good one.

The standard deviation of the actual delays of all groups is usually pretty low in comparison to the difference in average actual delays of two adjacent delay groups , suggesting that the delays within a group are tightly clustered together, and separated from the delays of other groups. Also the standard deviation of estimated delays is in most cases, only slightly smaller than the standard deviation of actual delays.

Note that the average estimated delay for a group and the average actual delay are reasonably in agreement. The difference in values of these two numbers are not important, however. The estimated delay is simply used to bin the different delays into groups. As a result, it is important that the average estimated delay is a good *predictor* of the average actual delay for the group. The actual delay value for each group will be generated running by SPICE on a small number of candidate vector pairs from that group, as described in Section 3.2.3.

Note also that the total number of groups is kept low by means of the choice of P. In effect, the number of distinct delays in our scheme is very low. In this sense, our scheme compares favorably with simpler gate delay models, while at the same time providing much higher accuracy.

Examples with larger numbers of vector pairs exhibit a larger maximum variation and standard deviation of actual delays, as expected. For such examples, the maximum variation and standard deviation of estimated delays are larger as well. This is attributed to the larger number of vector pairs in these examples.

Estimated Delay			Actual Delay			
average	max. variation	std dev	average	max. variation	std dev	group size
85.800	3.1469	2.700	63.117	4.2592	2.688	2
57.800	4.1522	1.697	53.778	9.1105	3.799	3
41.550	0.0000	0.000	37.322	0.0000	0.000	1

Table 2: Experimental Results for NAND2 Cell

Estimated Delay			Actual Delay			
average	max. variation	std dev	average	max. variation	std dev	group size
95.250	8.5039	6.614	76.686	13.0241	7.656	3
65.814	11.7213	3.934	58.564	17.8399	5.448	7
44.925	6.0100	1.909	42.882	8.5383	2.643	3
29.047	0.0000	0.000	33.252	0.0000	0.000	1

Table 3: Experimental Results for NAND3 Cell

Estimated Delay			Actual Delay			
average	max. variation	std dev	average	max. variation	std dev	group size
112.800	9.5745	8.818	99.208	10.0212	7.845	3
80.862	12.7854	4.421	69.826	15.9246	4.969	13
63.300	10.9005	4.951	55.804	15.0992	5.006	4
47.760	6.7839	2.645	48.290	11.3568	3.892	5
31.297	8.6262	1.559	37.182	9.2565	2.118	4
22.800	0.0000	0.000	31.501	0.0000	0.000	1

Table 4: Experimental Results for NAND4 Cell

The quantity $R = 100 \cdot (1 - (\#groups)/(2 \cdot |V_{out}^1| \cdot |V_{out}^0|))$ for each cell is shown in Table 10. This quantity represents the percent reduction in the number of delay values needed to characterize the library using our scheme, over the scheme where each vector pair has a separate delay value. Note that for the library chosen, an average reduction of 66% is obtained.

5 Conclusions and Future Work

We introduce an accurate technique for automatic static CMOS cell library characterization. The delay model we utilize is the vector-pair delay model. For each vector pair, we first compute a delay estimate. Vector pairs with estimated delay values within $P\%$ of each other are combined into groups. We showed the effectiveness of our scheme on some common library cells. The effectiveness of our scheme is illustrated by the fact that average delays of these groups decrease when the groups are placed in decreasing order of estimated delay, and the maximum delay

Estimated Delay			Actual Delay			
average	max. variation	std dev	average	max. variation	std dev	group size
56.400	0.0000	0.000	57.133	0.0000	0.000	1
45.600	0.0000	0.000	45.281	0.0000	0.000	1
37.600	0.0000	0.000	43.897	2.3421	1.028	2
30.400	0.0000	0.000	37.035	0.0000	0.000	1
22.800	0.0000	0.000	29.187	0.0000	0.000	1

Table 5: Experimental Results for NOR2 Cell

Estimated Delay			Actual Delay			
average	max. variation	std dev	average	max. variation	std dev	group size
73.950	10.9533	8.100	81.619	8.5139	6.949	2
50.893	13.7404	4.741	57.552	7.6878	3.185	7
33.013	0.2651	0.088	41.718	5.9745	2.492	2
24.825	0.0000	0.000	34.358	1.9209	0.660	2
16.548	0.0000	0.000	27.956	0.0000	0.000	1

Table 6: Experimental Results for NOR3 Cell

Estimated Delay			Actual Delay			
average	max. variation	std dev	average	max. variation	std dev	group size
107.700	10.0279	10.800	122.615	7.1042	8.711	2
74.285	13.0372	6.528	78.837	16.9730	6.202	13
50.638	6.0479	1.907	61.288	2.1166	0.972	4
37.033	3.3303	0.872	48.851	18.9278	6.619	3
26.412	4.9765	0.759	38.559	5.6284	1.505	4
17.898	0.0000	0.000	29.903	6.2655	1.428	3
13.425	0.0000	0.000	24.883	0.0000	0.000	1

Table 7: Experimental Results for NOR4 Cell

Estimated Delay			Actual Delay			
average	max. variation	std dev	average	max. variation	std dev	group size
64.604	11.5761	3.946	55.116	9.5537	3.529	7
47.272	10.7416	2.098	44.747	11.5875	2.788	9
35.506	11.8465	2.391	40.676	20.7973	5.196	8
23.648	10.6863	1.283	31.533	19.3521	3.223	6

Table 8: Experimental Results for AOI21 Cell

Estimated Delay			Actual Delay			
average	max. variation	std dev	average	max. variation	std dev	group size
46.697	12.1060	3.825	46.311	23.9473	6.452	16
32.503	12.0085	1.949	38.232	23.0460	4.999	8
22.350	8.0537	1.273	31.992	21.9203	3.422	6

Table 9: Experimental Results for OAI21 Cell

Cell	R^*
NAND2	50.00
NAND3	71.43
NAND4	80.00
NOR2	16.67
NOR3	64.29
NOR4	76.67
AOI21	86.67
OAI21	90.00

Table 10: Percentage Reduction in Number of Delay Values (* R= $100 \cdot (1 - (\#groups)/(2 \cdot |V^1_{out}| \cdot |V^0_{out}|))$

variation within any group is usually less than P%. Also the standard deviation of the group delays is low, suggesting that our model for delay estimation is an effective one. An average reduction of 66% is obtained by grouping delays in this fashion.

We wrote a tool which, given a transistor level SPICE net-list of all cells in the library, computes the groups and their average delays This accurate delay data can be used for highly accurate timing analysis for high-performance designs. It can also be used as the delay estimation technique for library-less synthesis techniques or for constant-delay synthesis techniques that have been reported recently.

In the future, we plan to incorporate a variety of methods to estimate the delay of a vector pair, so as to have a tradeoff between efficiency and accuracy. Also, we plan to devise implicit analysis techniques to enumerate the vector pairs. This will enable the scheme to process cells with a far greater number of inputs.

References

[1] S. Devadas, K. Keutzer, and S. Malik, "Computation of floating mode delay in combinational circuits: Theory and algorithms," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 1913–1923, Dec 1993.

[2] W. Lam and R. Brayton, *Timed Boolean Functions- A Unified Formalism for Exact Timing Analysis*. Kluwer Academic Publishers, 1994. ISBN 0-7923-945402.

[3] P. McGeer, A. Saldanha, R. Brayton, and A. Sangiovanni-Vincentelli, *Logic Synthesis and Optimization*, ch. Delay Models and Exact Timing Analysis, pp. 167–189. Kluwer Academic Publishers, 1993.

[4] H. Yalcin and J. Hayes, "Hierarchical timing analysis using conditional delays," in *Proc. IEEE/ACM Int'l Conf. on Computer-Aided Design, ICCAD '95*, pp. 371–377, 1995.

[5] V. Chandramouli, J. Whittemore, and K. Sakallah, "Afta: A delay model for functional timing analysis," in *Proceedings of the 1997 ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pp. 5–14, 1997. Austin, Texas.

[6] S. Taşıran, Y. Kukimoto, and R. Brayton, "Computing delay with coupling using timed automata," in *Proceedings of the 1997 ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pp. 232–244, 1997. Austin, Texas.

[7] J. Frößl and T. Kropf, "A new model to uniformly represent the function and timing of mos circuits and its application to vhdl simulation," in *European Design and Test Conference (EDTC)*, pp. 343–348, Mar 1994.

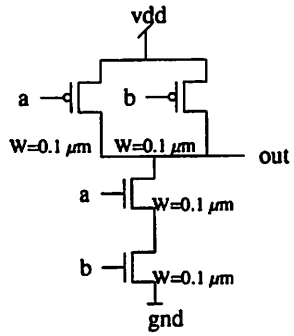
[8] S. Taşıran, S. Khatri, S. Yovine, R. Brayton, and A. Sangiovanni-Vincentelli, "A timed automaton-based method for accurate computation of circuit delay in the presence of cross-talk," in *Proceedings of Formal Methods in Computer Aided Design*, Nov 1998. To Appear.

[9] L. Nagel, "Spice: A computer program to simulate computer circuits," in *University of California, Berkeley UCB/ERL Memo M520*, May 1995.

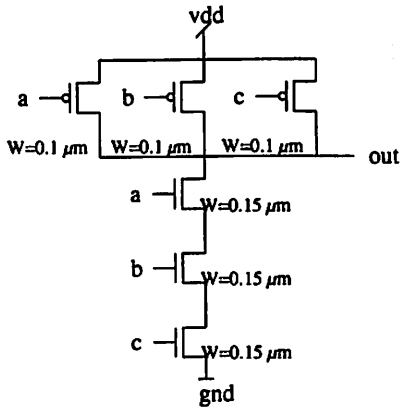
[10] S. Gavrilov, A. Glebov, S. Pullela, S. Moore, A. Dharchoudhury, R. Panda, G. Vijayan, and D. Blaauw, "Library-less synthesis for static CMOS combinational logic circuits," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 658–662, Nov 1997.

[11] R. Otten and R. Brayton, "Planning for performance," in *Proceedings of the Design Automation Conference*, pp. 122–127, Jun 1998.

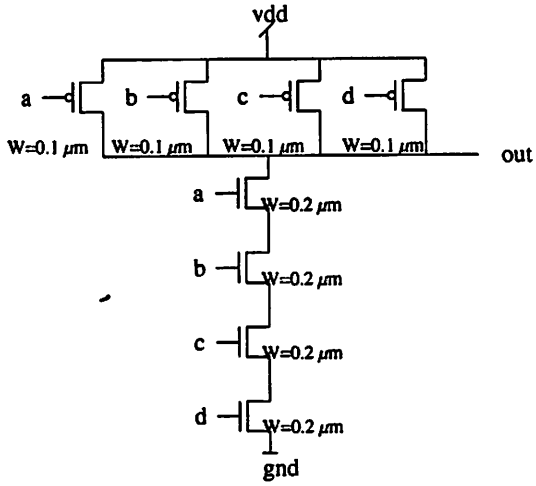
[12] R. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers*, vol. C-35, pp. 677–691, Aug. 1986.



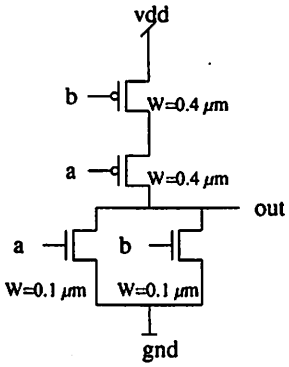
a) NAND2



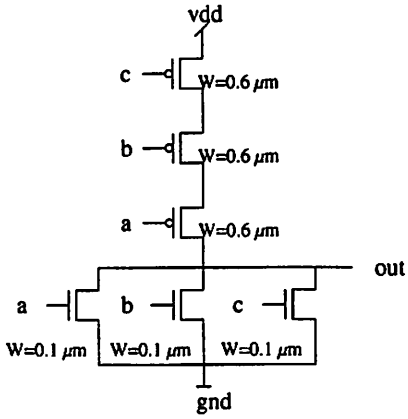
b) NAND3



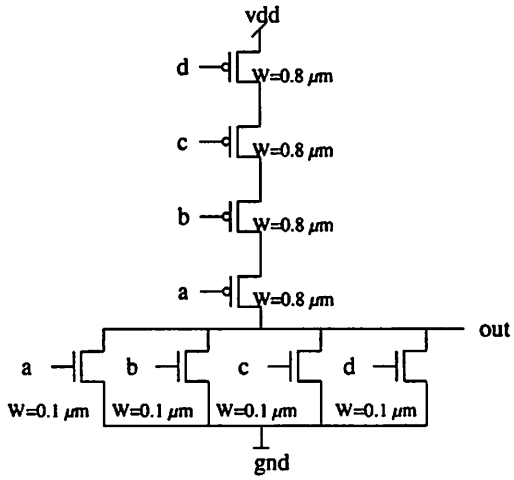
c) NAND4



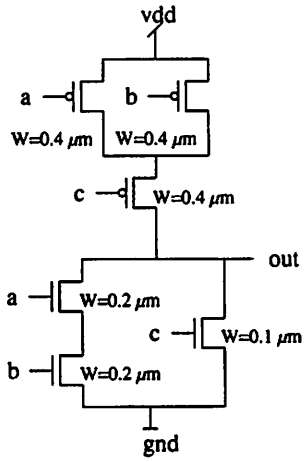
d) NOR2



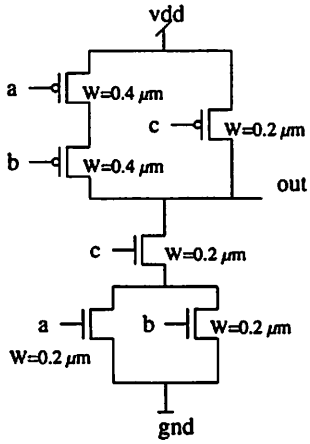
e) NOR3



f) NOR4



g) AOI21



h) OAI21

Figure 3: Cell Library