1993

Capacitance estimation of standard cell circuit design

Chih-kuo Liang
Lehigh University

Capacitance Estimation of Standard Cell Circuit Design

by

Chih-kuo Liang

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Electrical Engineering And Computer Science

Lehigh University

May 15, 1993

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science.

May 18, 1993
Date

Thesis Advisor

Chairperson of Department

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my advisor, Professor Richard Decker, for his guidance during the work and writing of this thesis.

I would also like to thank Dr. Chi-Chang Liaw, my supervisor at AT&T Bell Laboratories, for his support and valuable comments.

During the work I benefited from discussions and suggestions from many colleagues at my Laboratory. Special thanks go to Dr. Khe-Sing The who gave me a lot of helpful suggestions especially in the area of floorplanning. Dr. Shyuan Wang also provided me many valuable informations in cell placement and routing area.

Most computer support was provided by AT&T Bell Laboratories through the Computer Aided Engineering And Design Department.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In current microelectronic industry, the most popular capacitance estimation method used in pre-layout methodologies is the one called NETCAP[1] capacitance estimation. Because of the introduction of floorplanning to the timing-driven layout system, a new scenario based on the floorplan information to proceed more accurate capacitance estimation is possible now. This paper describes NETCAP capacitance estimation and proposed a methodology to proceed floorplan-based capacitance estimation. A floorplan-based capacitance estimate program is implemented to incorporate with an existing CAD layout tool to test the methodology. Three study cases are selected to proceed the experiments for the implementation of the program. The estimated capacitance values from the above two methods in the study cases are compared with the accurate capacitance values from post-layout processes, i.e. compared with the results of a reference technique based on final layout routing. In the three study cases, the NETCAP capacitance estimation achieved 0.283 pf[2] (27.29 %) mean estimation error[3] with the standard deviation 2.479 pf, whereas the floorplan-based capacitance estimation achieved 0.143 pf (13.61 %) mean estimation error[4] with the standard deviation 0.645 pf.

1. NETCAP is a program developed and widely used at AT&T Bell Labs, It is also known as Autoroute program in Bell Labs. see Section 1.1.
2. pf: pico-farad, the capacitance unit which is used throughout the paper.
3. see Table 9. Statistics of NETCAP Residual.
4. see Table 8. Statistics of Floorplan-based Residual.

1. Introduction

In timing simulation processes, the capacitance values of all nets in the circuit are important informations for the computation (post-layout simulation) or estimation (pre-layout simulation) of timing properties.

Net capacitance is composed of terminal capacitance and routing capacitance. The sum of these two components is the total net capacitance. The terminal capacitance is due to the source and destination cells connected by the net, whereas the routing capacitance is due to the interconnection of the net itself. The terminal capacitance of each cell is accessed from the description of the cell type.[1] In post-layout methodologies, the routing capacitance of each net can be accurately extracted by the layout tool from the completed layout. In pre-layout methodologies, since the placement and routing of cells affects the routing capacitance are not known yet, the routing capacitance can only be approximately estimated.

In current microelectronic industry, the most popular capacitance estimation method used in pre-layout methodologies is the one called NETCAP capacitance estimation. Because of the introduction of floorplanning to the Timing-Driven layout system,[5] a new scenario based on the floorplan information to proceed more accurate capacitance

5. Timing-Driven layout system is an integrated system at AT&T Bell Labs for standard cell circuit design.

estimation is possible now. This paper describes NETCAP capacitance estimation and proposed a methodology to proceed floorplan-based capacitance estimation. A floorplan-based capacitance estimate program is implemented to incorporate with an existing CAD layout tool[6] to test the methodology. Three study cases are selected to proceed the experiments for the implementation of the program. The estimated capacitance values from the above two methods in the study cases are compared with the accurate capacitance values from post-layout processes, i.e. compared with the results of a reference technique based on final layout routing.

1.1 NETCAP Capacitance Estimation

The simplest estimation to the routing capacitance is called NETCAP which uses the net connectivity information only.[2] The NETCAP estimation makes the assumption that the routing capacitance is function of the fan-in and fan-out of the net. It does not take into account the actual wire length of the interconnect but simply counts the number of terminals to which the net connects and applies a step-wise linear function to one less than this count number. The number of wires required for a net is the number of terminals on the net minus one because a simple net which connects one source to one destination connects two terminals.

6. The CAD tool is a standard cell placement and route tool called LTX. It is widely used at AT&T Bell Labs.

To the first order, the NETCAP capacitance is proportional to the number of terminals on the net minus one. But this is too pessimistic for high fan-out nets such as clock driver nets. A user specified step-wise linear function permits the routing capacitance for such nets to be estimated more accurately.

As described above, NETCAP capacitance estimation does not take into account the actual wire length of the net and is not accurately enough. However, in pre-layout processes, there is no layout at all and the net connectivity is the only information can be used. The method provides a quick reference without going through the layout processes.

1.2 Floorplan-based Capacitance Estimation

Recently, a floorplanning process is introduced to the timing-driven layout system. In the system, a block-based floorplanner is implemented to optimize the block areas and the total wire length among multiple blocks with a "soft" block representing a set of cells grouped by a partition.[3]

After the placement of the blocks by the floorplanner, a capacitance estimation can be proceeded based on the floorplanning information and produce the estimated capacitances for the pre-layout simulation.[4] Since the estimation uses the floorplanning results, it is called floorplan-based capacitance estimation.

From the floorplanning results, the location of the blocks are determined. The length of wires connecting the blocks can be estimated to produce capacitance estimation.[5] The capacitance values within a block can be estimated by applying the empirical formula we derived from some experiments or a theoretical model called neighborhood density equations estimation technique.[6]

More details about the theory and implementation of the floorplan-based capacitance estimation will be given. An overview of the design process is described first so that the reader will have a clear picture about the existing CAD layout tool and the necessary components to proceed floorplan-based capacitance estimation.

2. Overview of the Layout Design Process

The layout design process starts from the hierarchical netlist view of the design. The netlist file contains the connectivity information which indicates the interconnections of the cells without any physical wire shapes. Generally speaking, the objective of the layout design process is to create the layout from the netlist view of the design. The layout of the design is created by the layout engineer with the help of some CAD layout tools. Before a netlist file can be used by the layout tool, since it is a hierarchical netlist, a procedure called logic partition is needed to partition and flatten the connectivity. This procedure decides the grouping of cells and is proceeded by a tool called logic partitioner. The layout tool reads the partitioned netlist file and retrieves the geometries of the used cells from the cell library to create an unplaced layout. With the unplaced layout, the procedure of buffer placement can be proceeded to place all the I/O buffers first. The floorplanning is followed after the buffer placement. The floorplanning decides the locations of the cell groups, whereas the location of each cell is obtained by the next procedure called cell placement. After the cell placement is done, the routing procedure can be proceeded to complete the layout of the design. The flowchart of the layout design process is illustrated in Figure 1. The five components, logic partition, buffer placement, floorplanning, cell placement, and routing are described below.

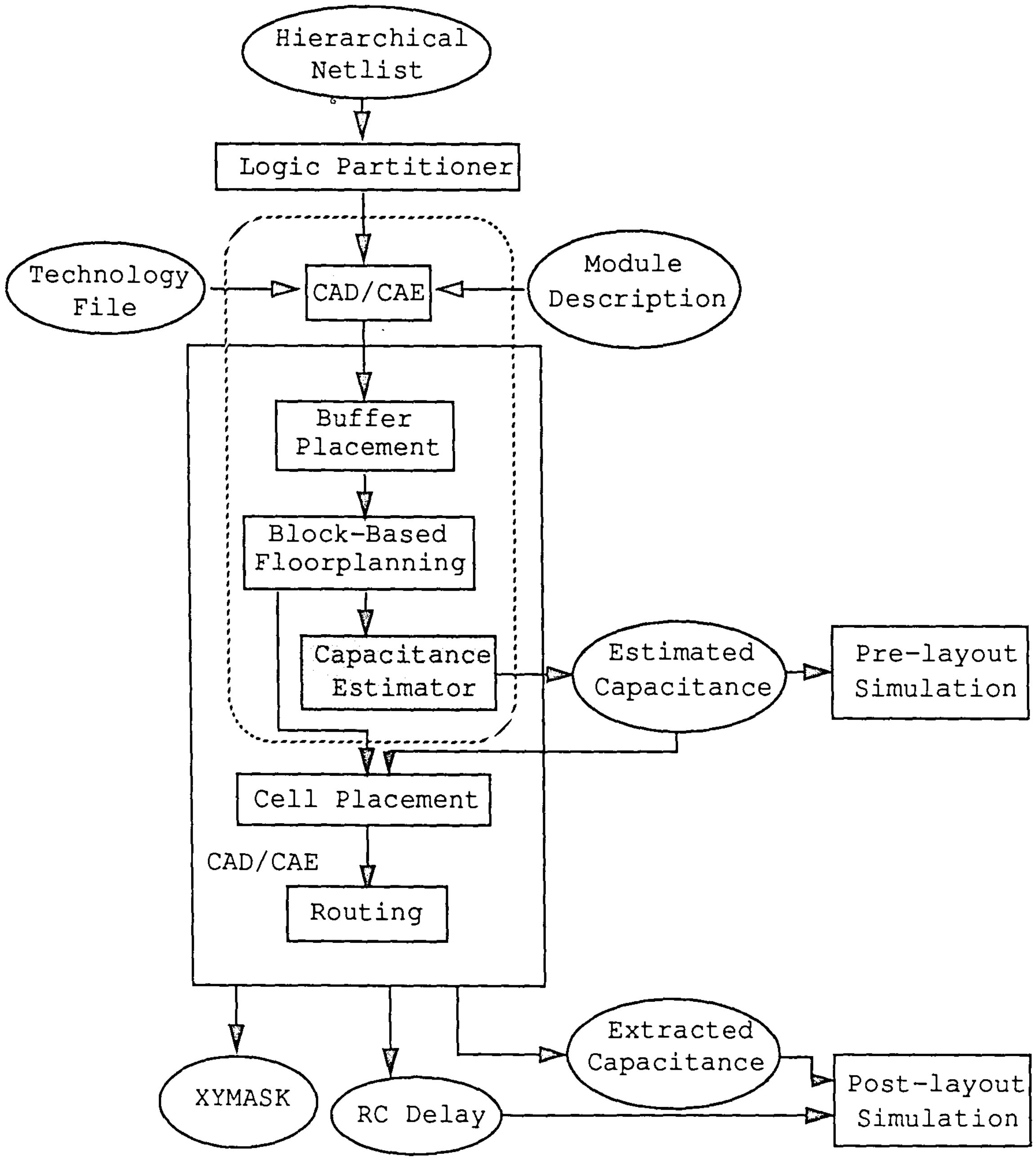

Figure 1. Timing-Driven Layout Using Floorplanner

2.1 Logic Partition

The logic partitioner partitions a circuit into several disjoint modules with the objective of minimizing both the number of modules and the number of interconnections among them.[7] The model of the unpartitioned circuit, i.e. the flattened netlist, consists of an interconnection of cells. Every cell is characterized by its relative size, which reflects the size of that cell relative to other cells. The partitioner expects the component size data as a size library, which is a simple text file consisting of a sequence of lines. A size library file of the standard cells is extracted from the polycell library system and is provided for the process. With the cell size information and nr_module, the user specified number of modules, the logic partitioner applys bottom-up clustering algorithm to group the cells into nr_module modules. It also produces the information of the number of cells, the number of pins, and the number of nets in every module.

2.2 Buffer Placement

A buffer placement tool is used to create and verify the placement of I/O buffers and power/ground pins. It is used to verify the acceptability of the buffer placement by the user, or to synthesize an appropriate buffer placement aiming at minimizing the noise voltages due to the ground bounce. The tool provides built-in algorithms to quantitatively and efficiently derive the switching noise characteristics of a given buffer placement with a package and a process technology specified by the user. It is started with an initial I/O pinlist extracted from the netlist. The process is proceeded by editing the pinlist, choosing a package, computing inductances, and synthesizing the near-optimal buffer placement. Different configurations made by rearranging the pin placement and adding power and ground pins are interactively entered and analyzed. The final result of the process is written to a text file which will be read by the floorplanner.

2.3 Floorplanning

The floorplanner has the capability to generate layout view from a given netlist view, technology parameters, and cell library. The objects in the database are polycells, hard blocks, and I/O buffers. Since the netlist view is partitioned by the logic partitioner, the polycells are clustered into soft blocks when the layout view is generated by the floorplanner. In addition to generating the layout view, the floorplanner is used to determine the shapes and positions of the soft blocks as well as the positions and orientations of the hard blocks and I/O buffers.[8] It is a tool for obtaining realistic block level placement, and producing accurate interconnect parasitics to be used for timing verification. The floorplanner contains a set of automatic and interactive procedures. The essential automatic procedures are for buffer placement, block placement. The interactive procedures include commands to hard place and soft place, to move, and to reshape the blocks. Utilities are also provided for viewing the graphical representation of the design.

The I/O buffers are placed by the floorplanner according to the buffer placement file created by the buffer placement tool described above. Based on the shape constraints of each block and the connectivity among blocks, the block placement procedure automatically places all the blocks and determines the shapes of all the soft blocks. The objective is to minimize the total area of the chip as well as the total interconnection wire length.

2.4 Cell Placement

A CAD layout tool[7] is used to complete the layout after the floorplanning. The floorplanning only determines the placement and alignment of chip components within the overall chip boundary whereas the cells within each soft blocks are placed by the tool with the simulated annealing algorithm using the net length and capacitance as the cost function. The tool also provides some options to merge the specified soft blocks into polyrows[8] with or without changing the overall cell grouping. The boundary outlines of the soft blocks from the floorplanning are no longer existed because the layout is represented by polyrows, hard blocks, and I/O buffers after the cell placement.

2.5. Routing

After the cell placement, the same CAD tool is used to divide the space left between the chip components into routing modules that will eventually contain routing. A process

7. The CAD tool is a standard cell placement and routing tool called LTX which is used to perform all the experiments of this paper.
8. A polyrow is a group of polycells in a row shape.

called loose routing is first used to determine the global paths that connect the chip modules together. The tool looks at the connectivity of each module, assigns connections to the routing modules based on the shortest wire length and other signal performance properties. After loose routing, a process called placement improvement can be proceeded based on the loose routing information. The last process is called fine routing which creates the actual routing with the information of various communication paths provided above.

3. Incorporating Capacitance Estimation in The Layout Design Process

A verification of the layout is needed after the layout design process. The verification is a process called timing simulation. There are two kinds of timing simulation, pre-layout simulation and post-layout simulation. The pre-layout simulation uses the net capacitances derived from NETCAP capacitance estimation and therefore is before the layout design process. However, the capacitance estimation can not be accurate enough because no layout information is used by the NETCAP capacitance estimator. In post-layout simulation, the net capacitances provided as input information are computed from the wire length of the complete layout. Suppose the post-layout simulation found that the timing properties are not acceptable, the whole layout design process has to be redone starting from the logic partition.

However, a capacitance estimation can be incorporated in the layout design process after the floorplanning before the cell placement. With the approach, the layout design process can be split into two major processes, floorplanning and layout completion, the cell placement and routing which are the most CPU-dense and time-consuming procedures. Under the scenario, the designers are expected to proceed the first process with a floorplanner to produce floorplans with sufficient placement resolution and use the estimated net capacitances in pre-layout simulation. The layout engineers are expected to proceed the second process with these floorplans and the capacitance files as reference and produce the final layouts. In general, the designers do buffer placement and floorplanning, estimate routing capacitances from the floorplan, run simulation, tighten the capacitance values and re-simulate, and eventually include the floorplan and the capacitance file in the package to the layout engineer. The design scenario is aimed at reducing the pre-layout engineering cost and maintaining a smooth post-layout process.

4. Floorplan-based Capacitance Estimation

The accuracy of the net capacitance estimate depends on the estimate of the routing wire length of the signal net. Applying the wire length and wire width with the unit area capacitance and unit fringe capacitance, the capacitance of each signal net can be estimated without the chip completely laid out. The main challenge in the implementation lies in how to get practical estimate for the total wire length of each signal net after the blocks are placed by the block-based floorplanner. The total wire length of a signal net can be estimated from two parts. One is the wire length from the interconnections among fixed terminals (of buffers or hard blocks) and soft blocks which contain source terminals of the signal net. The other is the wire length of the signal net within each soft block.

4.1 Estimate Wire Length among Multiple Blocks

After block placement, modules on the chip are handled as three categories: soft blocks, hard blocks and I/O buffers. The terminal locations of each hard block and I/O buffer are known and can be used directly to estimate the wire length. However, the location of a terminal within a soft block is not determined yet. We use the center point of the soft block to approximate the actual terminal location.

With all the terminal locations of a signal net, four different near-optimal rectilinear Steiner Trees can be calculated by growing from bottom, top, left and right. The shortest tree can be chosen as the geometric representation of the net, from which the estimated wire length of the signal is obtained.

However, hard blocks are blockage for Steiner trees since they are not penetrative by routing wires. In order to obtain a more accurate wire length, a maze-running algorithm is needed. It can be achieved by constructing the minimal Steiner tree, then check against the blockage. If there is no penetration, the Steiner tree can be used to estimate the wire length. If penetration occurs, the Lee's algorithm[9] will be used to modify the tree and estimate the wire length more accurately.

4.2 Estimate Wire Length within a Soft Block

The wire length within a soft block can be estimated by applying a theoretical model called neighborhood density equations estimation technique presented by a paper in 1992 DAC.[9] For several industrial circuits tested, the technique achieved average estimation error of 9.0% with a maximum deviation of +16.3%, which compares favorably with other techniques previously proposed. The paper concludes that U_m, the average wire length of m-pin net is given by

$$U_m = 2(m^2 + 2m-2)/3m * \frac{1}{\beta}$$

where β is the equilibrium parameter. Therefore,

$$U_m = (m^2 + 2m-2)/3m * U_2$$

where $U_2 = 2 / \beta$. Since the capacitance C_m is in proportion to U_m, it is appropriate to represent C_m as

$$C_m = (m^2 + 2m-2)/3m * C_2$$

The value C_2 can be derived from some experiments on the 0.9 micron CMOS technology, with which the timing-driven placement was fine tuned. The experiments are described in the next chapter.

9. see Reference [5].

5. How to Produce Capacitance Estimator

In order to apply the above theory to produce the floorplan-based capacitance estimator, appropriate data are needed in addition to code writing. The estimated capacitance of a net is composed of the capacitance within each soft block and the capacitance among multiple soft blocks of the net. Through the data of some experiments, the empirical formula of C_2 can be derived. The experimental data can be used to verify the above theoretical formula or to derive our own empirical formula of C_m. The capacitance within each soft block is estimated by C_2 and C_m. A program[10] is implemented to estimate the wire length with the algorithm of Steiner tree for the capacitance among multiple soft blocks.

5.1 Procedure of Experiments

The procedures of performing the experiments are summarized as follows:

- Find an appropriate pure polycell chip.
- Extract the netlist of the chip.
- Make 15, 20, and 25 nearly-equal logic partitions of the chip. Each number of partitions is one study case.

10. see Appendix.

- Perform the floorplanning of each study case.
- Completely Lay out each study case.
- Extract the capacitances of 2-pin, 3-pin, 4-pin, and 5-pin nets from each study case.
- Group the capacitances of 2-pin, 3-pin, 4-pin, and 5-pin nets in each soft block for each study case.
- Analyze the correlation of the capacitances of 2-pin, 3-pin, 4-pin, and 5-pin nets versus the soft block size for each study case.
- Analyze the correlation of m-pin capacitances versus 2-pin capacitances.
- Analyze the correlation of the three study cases.

5.2 Case Study

A chip with 6558 polycells and 8716 signal nets is selected. The layout of the chip is illustrated in Figure 2. In order to simplify the analysis of m-pin signal capacitances, the I/O buffers are ignored from our experiments. A netlist is extracted from the 44 polyrows of the chip. Based on the netlist, the following study cases are proceeded.

5.3 Study Case I -- 15 Soft Blocks

A logic partition with 15 modules was performed from the netlist. The statistics of the partition is listed in Table 1. The floorplan of the partition is illustrated in Figure 3.

5.4 Study Case II -- 20 Soft Blocks

A logic partition with 20 modules was performed from the same netlist. The statistics of the partition is listed in Table 2. The floorplan of the partition is illustrated in Figure 4.

5.5 Study Case III -- 25 Soft Blocks

A logic partition with 25 modules was performed from the same netlist. The statistics of the partition is listed in Table 3. The floorplan of the partition is illustrated in Figure 5.

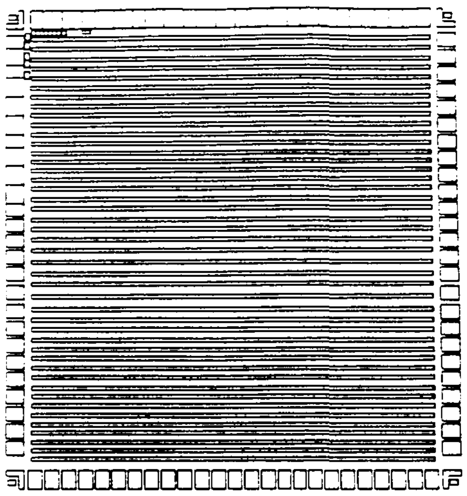
Figure 2. Chip Layout of the Case Study

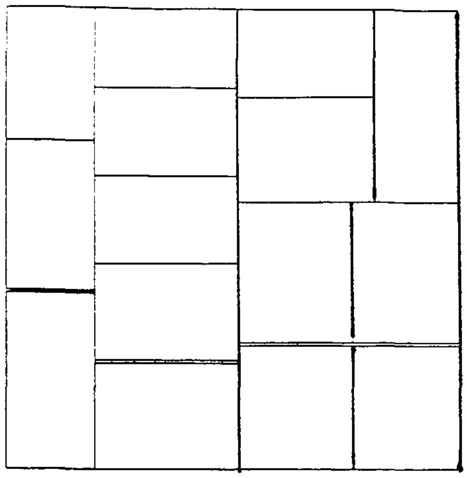
Figure 3. Floorplan of Study Case I

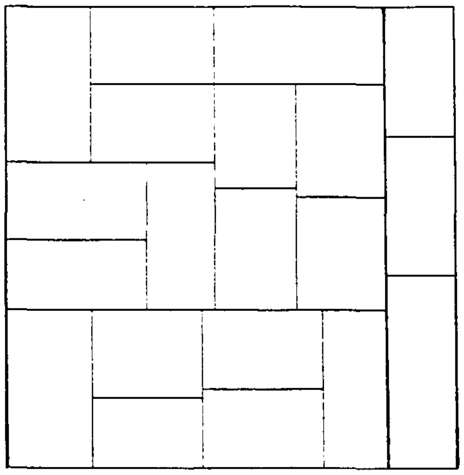
Figure 4. Floorplan of Study Case II

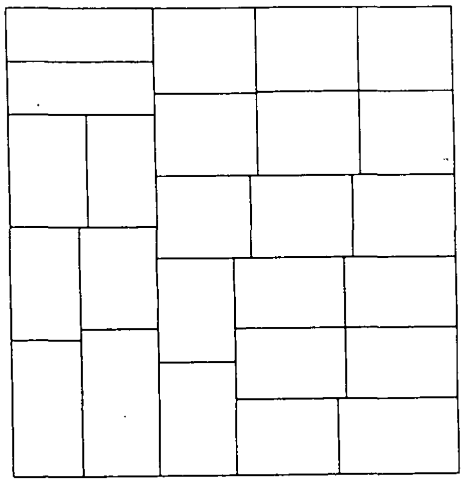
Figure 5. Floorplan of Study Case III

Cluster No.	Devices	Size	Pins	Buried Nets	Boundary Nets
1	349	3971	116	431	116
2	477	4572	119	543	119
3	468	5004	219	510	219
4	470	4661	133	546	129
5	327	3737	98	417	98
6	370	3956	113	446	113
7	417	4428	150	503	150
8	469	4600	152	539	150
9	391	4389	168	474	168
10	405	4570	154	511	150
11	605	4883	281	657	233
12	512	4944	253	517	223
13	454	4938	222	519	221
14	409	4697	193	450	193
15	435	4896	183	516	183

Devices: 6558 Nets: 8716
Total size: 68246
Target average size: 4550
Size factor: 2.72727
Smallest size: 2
Total pins: 2554
Total buried nets: 7579

Table 1. Statistics of Case I - Logic Partition with 15 Soft Blocks

Cluster No.	Devices	Size	Pins	Buried Nets	Boundary Nets
1	232	2915	95	290	95
2	313	3177	97	370	95
3	378	2723	225	274	197
4	345	3002	300	328	294
5	259	3258	97	323	97
6	352	3518	117	410	115
7	284	3358	131	342	131
8	358	3481	140	401	140
9	379	3447	153	417	127
10	404	3632	218	468	199
11	350	3437	203	369	203
12	312	3650	100	397	100
13	331	3699	119	404	119
14	310	3547	151	328	151
15	347	3444	125	390	122
16	320	3617	145	381	145
17	309	3638	105	388	105
18	287	3462	123	337	123
19	371	3507	112	429	111
20	317	3734	145	378	145

Devices: 6558 Nets: 8716
Total size: 68246
Target average size: 3413
Size factor: 2.72749
Smallest size: 2
Total pins: 2901
Total buried nets: 7424

Table 2. Statistics of Case II - Logic Partition with 20 Soft Blocks

Cluster No.	Devices	Size	Pins	Buried Nets	Boundary Nets
1	201	2675	128	220	128
2	290	2628	109	315	109
3	285	2899	114	325	110
4	269	2919	143	334	128
5	302	2730	207	319	207
6	221	2704	116	236	116
7	316	3001	111	363	93
8	198	2653	110	227	110
9	296	2716	212	284	211
10	295	2596	180	312	180
11	216	2725	117	251	117
12	205	2745	123	222	123
13	254	2692	118	283	118
14	253	2693	108	274	108
15	289	2671	215	300	215
16	311	2721	214	313	202
17	377	2650	274	301	256
18	215	2762	130	247	130
19	305	2666	207	266	193
20	273	2705	81	322	81
21	273	2892	164	288	162
22	222	2695	104	281	99
23	243	2717	110	293	110
24	241	2671	150	264	150
25	208	2720	126	249	126

Devices: 6558 Nets: 8716
Total size: 68246
Target average size: 2730
Size factor: 2.72727
Smallest size: 2
Total pins: 3671
Total buried nets: 7089

Table 3. Statistics of Case III - Logic Partition with 25 Soft Blocks

5.6 Analyses of Capacitance versus Soft Block Size

It is desirable to find out an empirical formula of the 2-pin signal capacitance from the soft block size. Since the width and the height of the soft block is known, we could apply the following non-linear model

$$C_2 = (\alpha x + \gamma y)\eta$$

where $0 < \alpha < 1$ weights the relative contributions of horizontal wirings and γ weights the relative contributions of vertical wirings, and η strongly correlates to the capacitance per unit length wiring.

Some histograms of the 2-pin net capacitance distribution are illustrated in Figure 6. The pattern shows most 2-pin net capacitances are less than 0.1 pf no matter which soft block they are in. This pattern is valid for all three study cases. The statistics about the correlation between the mean capacitance of each soft block and the block width and height from study case I is illustrated in Figure 7. The statistics from study case II is illustrated in Figure 8. Figure 9 illustrates the statistics of study case III.

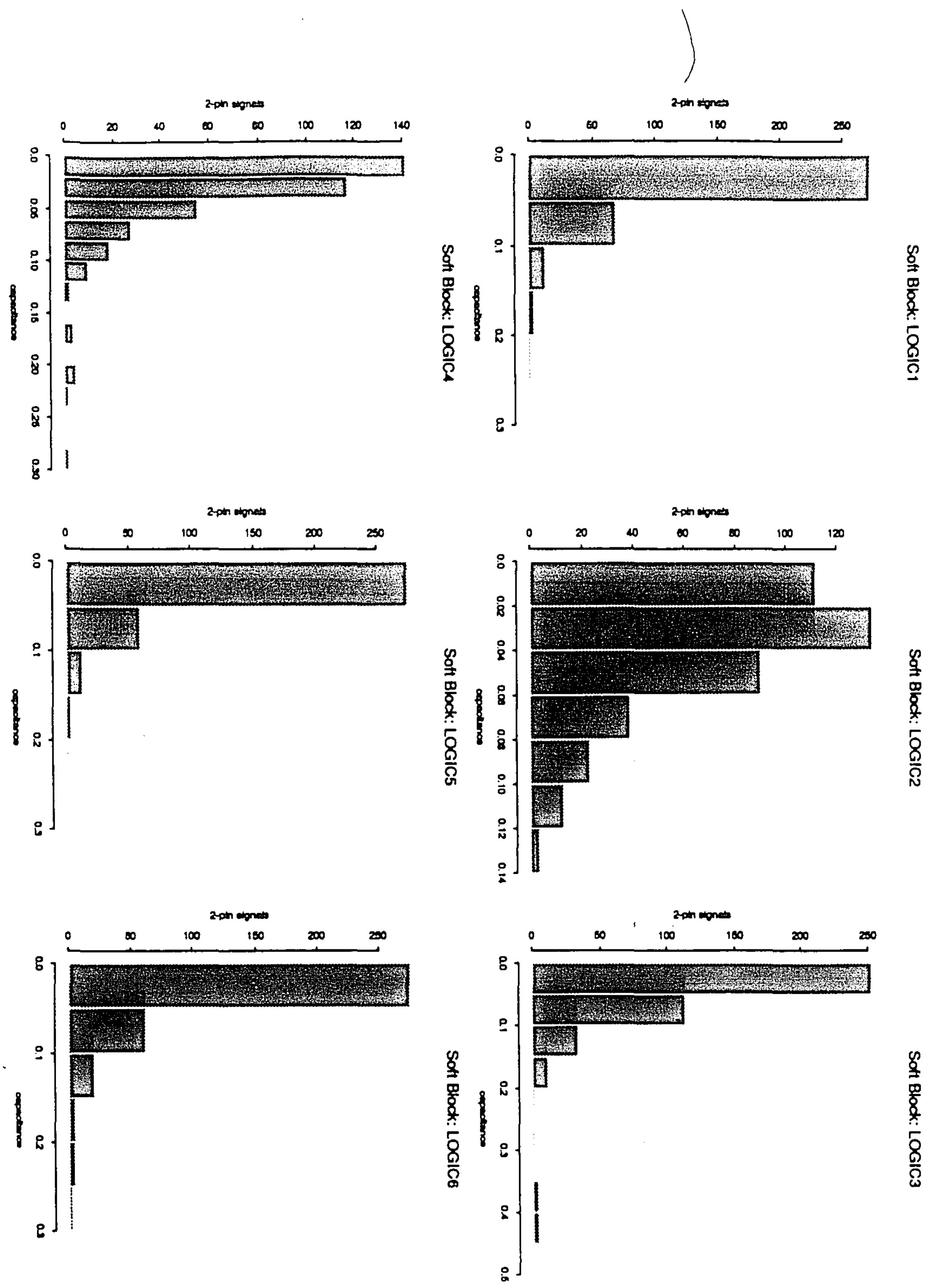

Figure 6. Histograms of 2-pin Net Capacitance Distribution

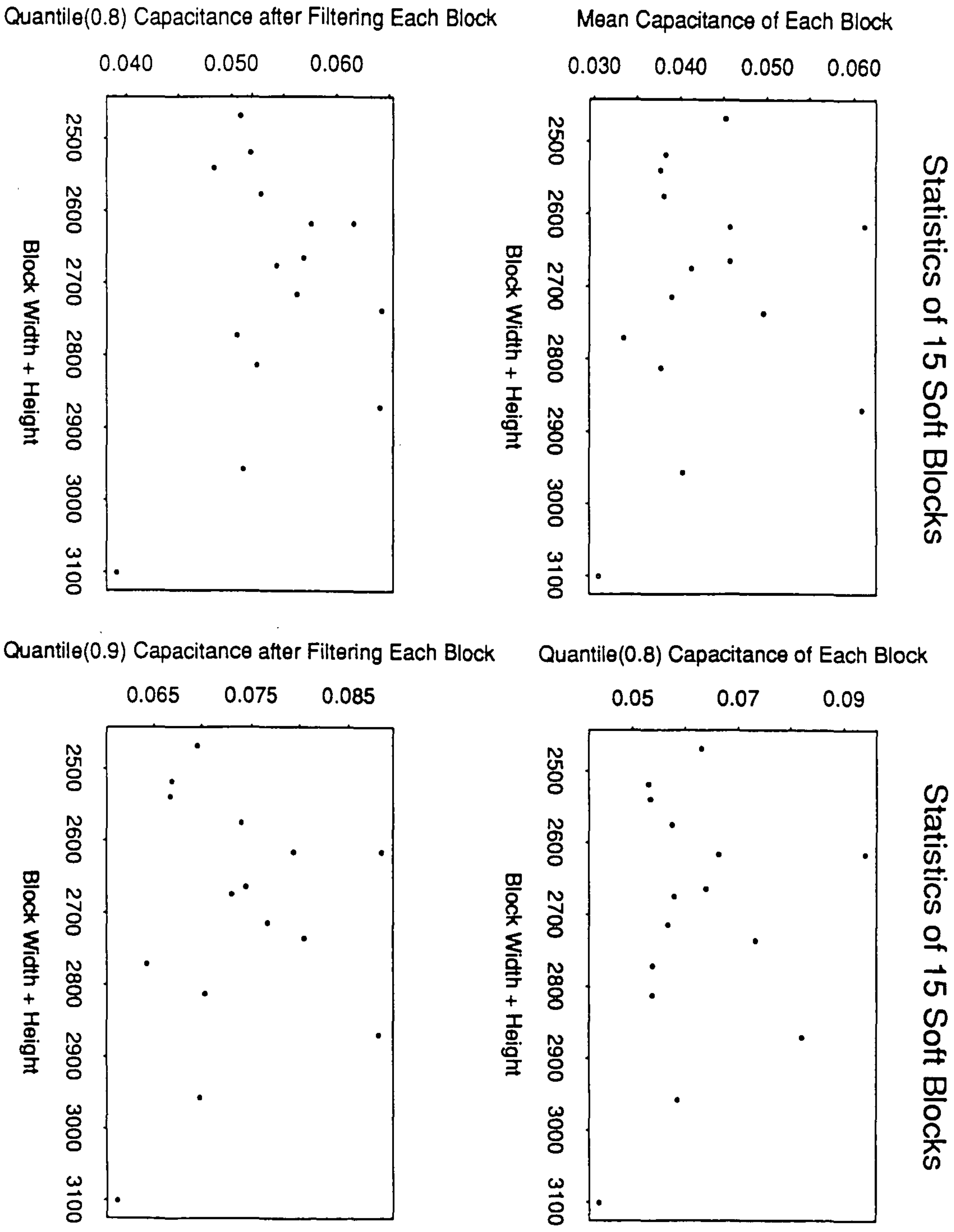

Figure 7. Case I - Correlation of Mean Capacitance versus Soft Block Size

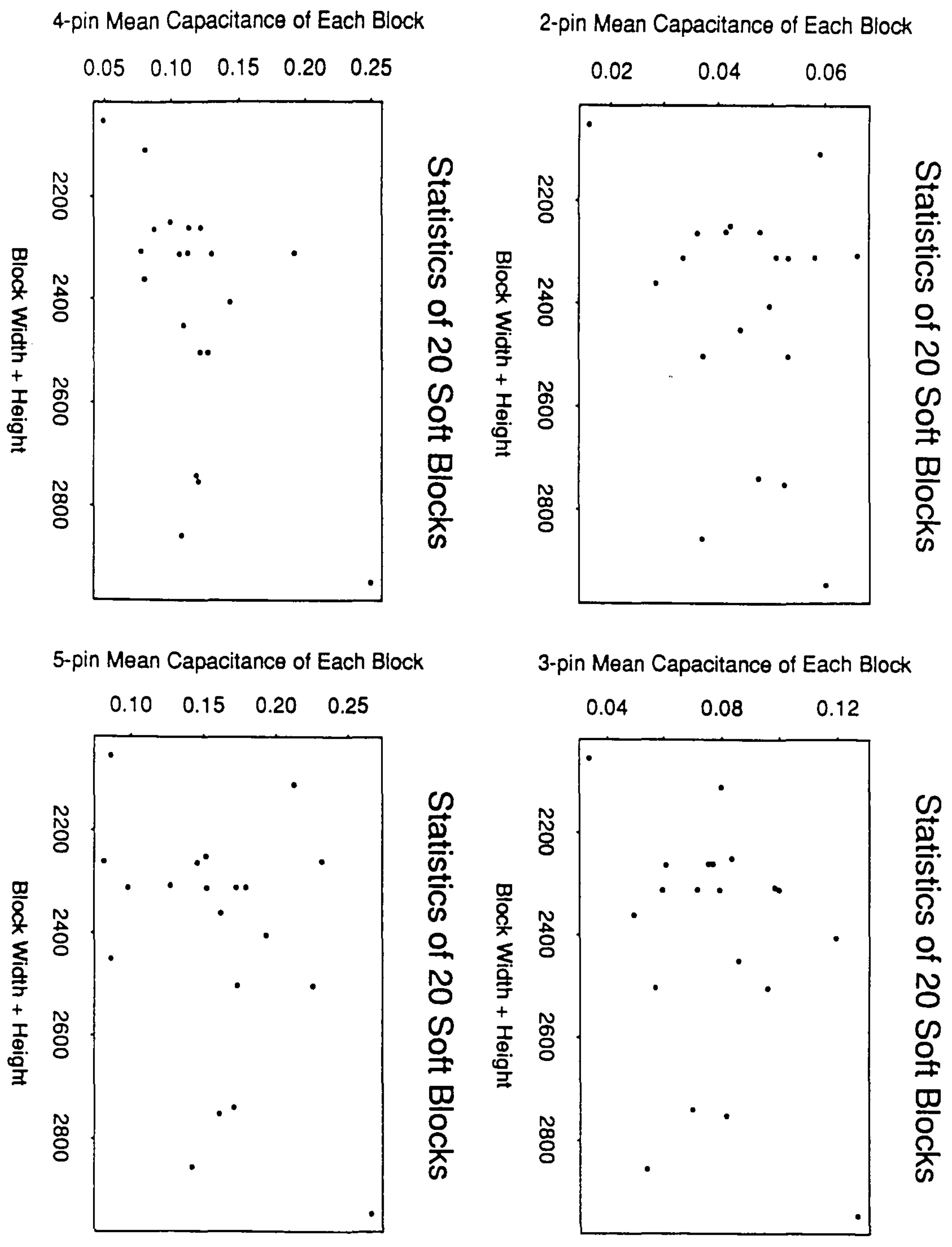

Figure 8. Case II - Correlation of Mean Capacitance versus Soft Block Size

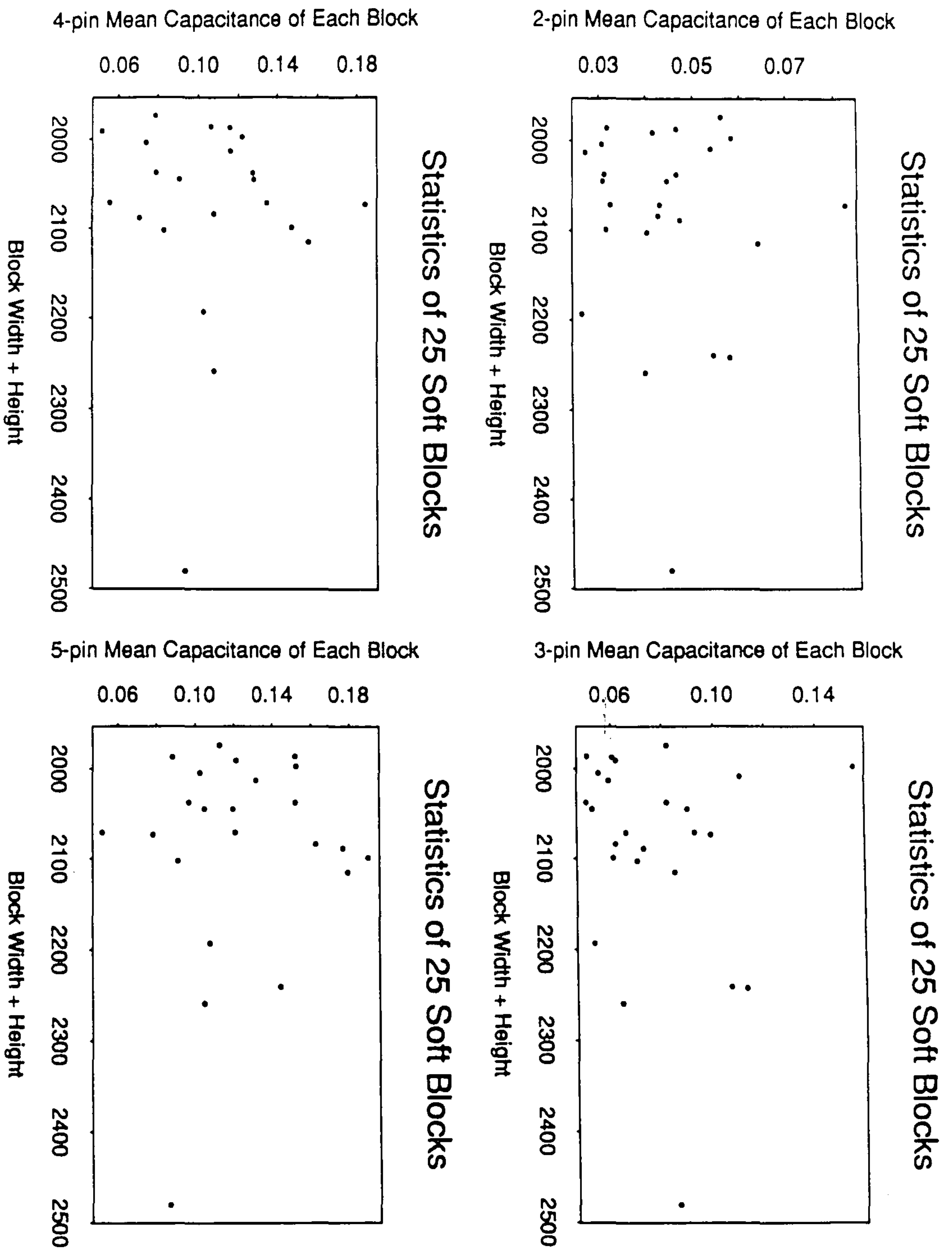

Figure 9. Case III - Correlation of Mean Capacitance versus Soft Block Size

From Figure 6, 7, 8, and 9, it is obvious that there is no direct correlation between the 2-pin signal capacitance and the block size. The reason is that the algorithm of the cell placement, the simulated annealing, functions so well that most cells constructing the 2-pin net are very close and the distance between the two cells is independent of the block size. The attempt of characterizing C_2 as a function of the dimensions (x,y) of the soft block is invalid.

5.7 Analyses of 2-pin Capacitance versus m-pin Capacitance

A lot of analyses were performed to verify the correlation between the 2-pin capacitance and the m-pin capacitance. Figure 10 illustrates the mean 2-pin, 3-pin, 4-pin, and 5-pin capacitances of each soft block from study case I. The same plot from study case II is illustrated in Figure 11. Figure 12 illustrates the plot from study case III.

From Figure 10, 11, and 12, it can be seen that although there seems no correlation among mean capacitances of soft blocks, the correlation between m-pin capacitance and 2-pin capacitance is obvious. A summary statistics of the three study cases is shown in Table 4. In addition to the three study cases, another case without partitioning, i.e., the traditional procedure without floorplanning was performed to make the comparison with the three different partitions. The statistics of the case is also provided in Table 4. In order to derive C_m from C_2, two linear least squares fitting with the following mathematical models are proceeded:

$C_m = a * C_2$
$C_m = b * C_2 + c$
The results are listed in Table 5 and the values from the theoretical model
$C_m = (m^2 + 2m-2)/3m * C_2$

are listed in Table 6 for the comparison.

Case No.	Number of Soft Blocks	m-pin	Number of Nets	Capacitance Mean (pf)
Case I	15	2-pin	6041	0.04313
		3-pin	781	0.08126
		4-pin	152	0.10618
		5-pin	121	0.15895
Case II	20	2-pin	5957	0.04596
		3-pin	760	0.07795
		4-pin	150	0.11768
		5-pin	112	0.16118
Case III	25	2-pin	5731	0.04441
		3-pin	701	0.07330
		4-pin	127	0.10532
		5-pin	88	0.12660
Case IV	1 (No Partition)	2-pin	6391	0.04352
		3-pin	925	0.11265
		4-pin	181	0.11158
		5-pin	219	0.15402

Table 4. Statistics of Four Study Cases

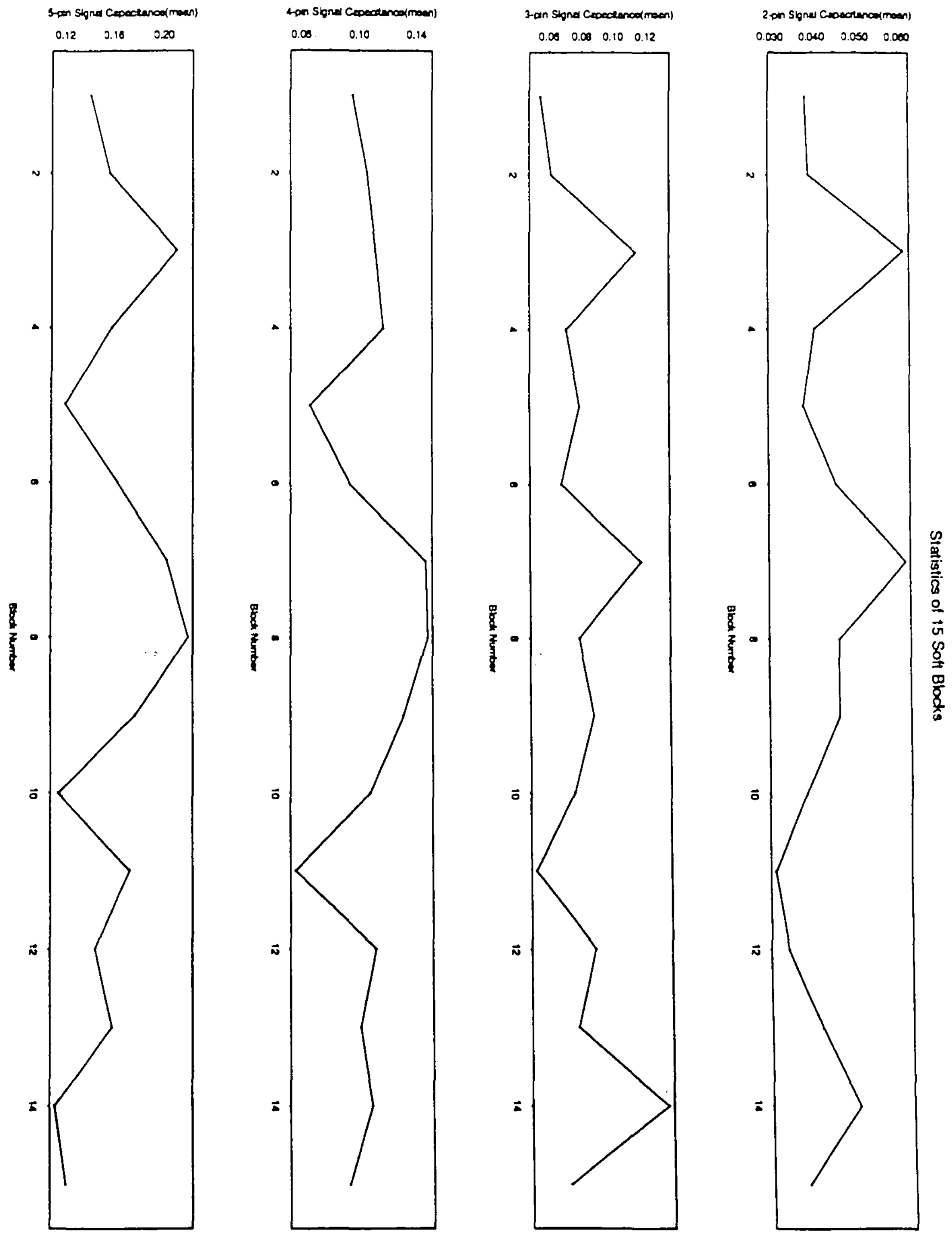

Figure 10. Case I - Correlation of 2-pin Capacitance versus m-pin Capacitance

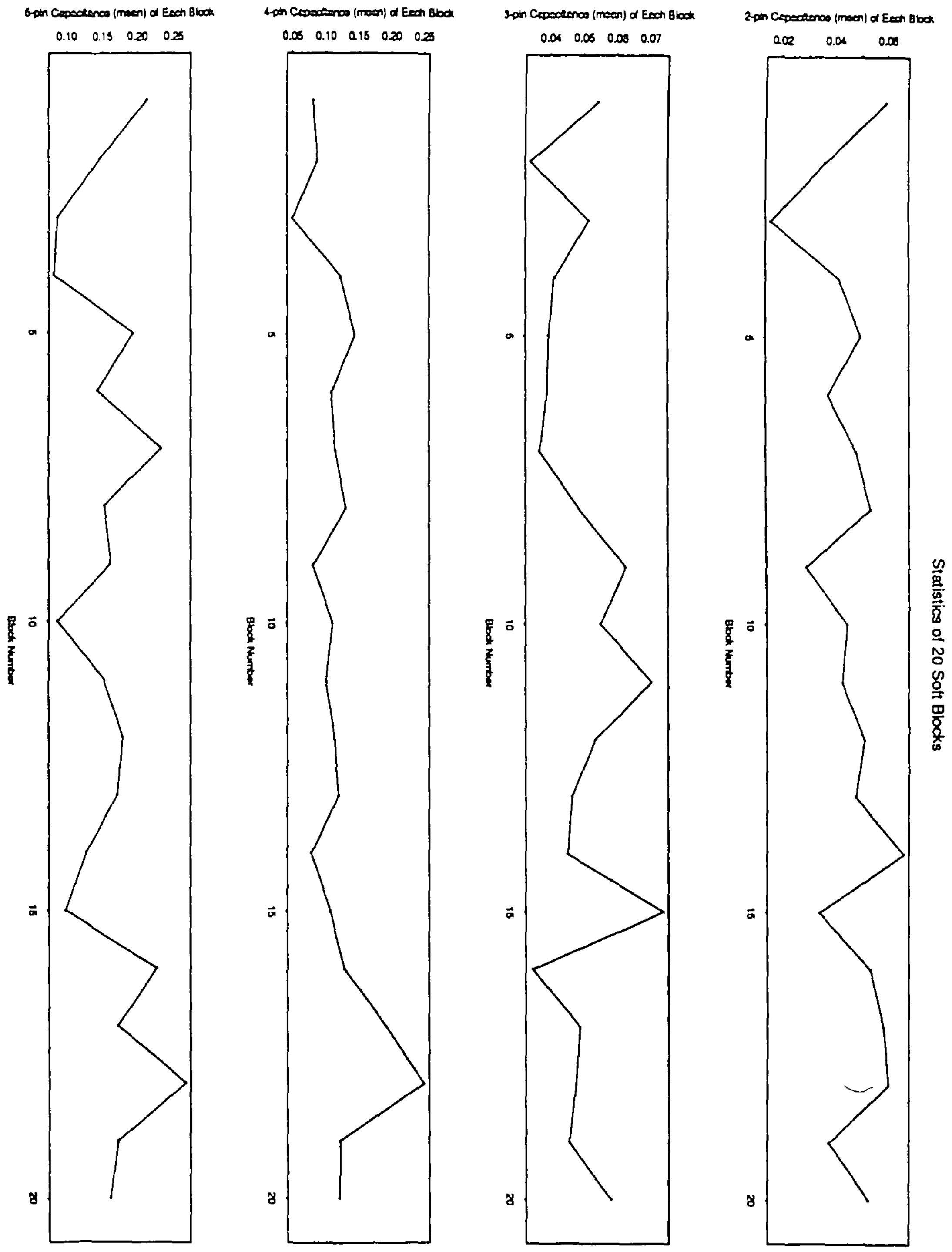

Figure 11. Case II - Correlation of 2-pin Capacitance versus m-pin Capacitance

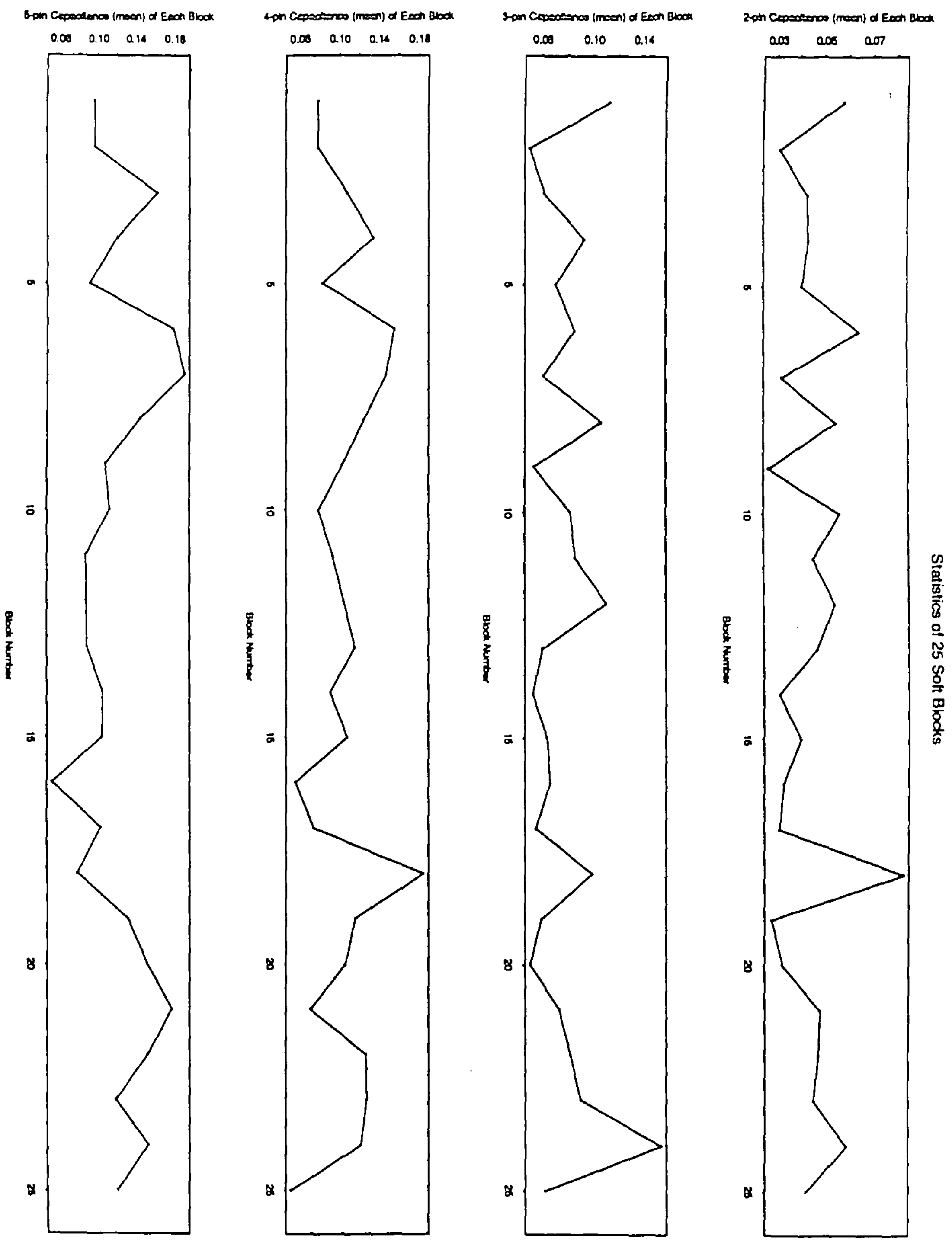

Figure 12. Case III - Correlation of 2-pin Capacitance versus m-pin Capacitance

Case No.	Number of Soft Blocks	m-pin	$C_m = a * C_2$	$C_m = b * C_2 + c$	
			a	b	c
Case I	15	3-pin	1.83333	2.02387	- 0.00604
		4-pin	2.43397	1.72674	0.03170
		5-pin	3.62507	2.09390	0.06863
Case II	20	3-pin	1.68667	1.54160	0.00710
		4-pin	2.51436	1.80854	0.03456
		5-pin	3.42814	2.22213	0.05905
Case III	25	3-pin	1.71817	1.35234	0.01766
		4-pin	2.37217	1.31011	0.04990
		5-pin	3.01128	0.85301	0.09668
Case IV	1 (No Partition)	3-pin	2.58831	▨	▨
		4-pin	2.56383	▨	▨
		5-pin	3.53891	▨	▨

Table 5. Empirical Values from Four Study Cases

Theoretical Model	$C_m = d * C_2$	3-pin	4-pin	5-pin
	$d = \frac{m * m + 2m + 2}{3m}$	1.44444	1.83333	2.20000

Table 6. Numerical Values from Theoretical Model

6. Capacitance Empirical Formula

6.1 2-pin Capacitance Empirical Formula

From Table 4, C_2 value is around 0.044 pf in each study case. It is appropriate to use the average value of three study cases to represent C_2. However, the objective of the capacitance estimator is not only to provide the overall estimated capacitance, but also to warn the designer those signals whose capacitances exceed the specification of the chip design. In other words, the worst case is more interesting to us. To serve the objective, the technique of quantile is more appropriate to us than the mean value. A data analysis tool S-PLUS[11] is used by the paper to do the statistical analyses. The package provides the quantile function by an algorithm which linearly interpolates between order statistics of a vector, assuming that the i-th order statistic is the i / (vector length) quantile.[10] Therefore, quantile(0.70) means the value is greater than 70% of the values in the vector. Different quantile values, based on 0.7, 0.8, 0.85, 0.9, and 0.95 are derived from each study case and listed in Table 7 for references.

11. S-PLUS is a statistic software product from Statistical Sciences, Inc.

Case No.	m-pin	Number of Nets	Empirical Quantiles				
			0.70	0.80	0.85	0.90	0.95
Case I	2-pin	6041	0.04773	0.06120	0.07309	0.09005	0.12072
	3-pin	781	0.09162	0.11561	0.13597	0.15916	0.19434
	4-pin	152	0.12684	0.14353	0.15809	0.18440	0.22520
	5-pin	121	0.19134	0.22264	0.24603	0.28855	0.35374
Case II	2-pin	5957	0.04901	0.06616	0.07860	0.09718	0.13665
	3-pin	760	0.08687	0.10765	0.12470	0.14658	0.18674
	4-pin	150	0.13734	0.16416	0.17824	0.20572	0.24622
	5-pin	112	0.20492	0.24221	0.27474	0.29874	0.33730
Case III	2-pin	5731	0.05051	0.06502	0.07525	0.09418	0.12524
	3-pin	701	0.08753	0.10398	0.11469	0.13568	0.17103
	4-pin	127	0.12131	0.15428	0.16363	0.18108	0.19302
	5-pin	88	0.15111	0.16989	0.18142	0.20407	0.22606
Case IV	2-pin	6391	0.04516	0.06227	0.07412	0.09199	0.12874
	3-pin	925	0.12753	0.17571	0.20456	0.24332	0.31177
	4-pin	181	0.13403	0.14846	0.16951	0.19948	0.23855
	5-pin	219	0.17086	0.22682	0.25172	0.27698	0.32664

Table 7. Quantile Values of Three Study Cases

It is decided to use quantile(0.8) to represent C_2 in the capacitance estimator to reach the 80% confidence level. Although larger quantile values can reach higher confidence level, it is too pessimistic if the capacitance is too high. The average value of quantile(0.8) from the three study cases is 0.064127 pf and is used in the capacitance estimate program.

6.2 m-pin Capacitance Empirical Formula

Comparing the empirical results of C_3, C_4, and C_5 with the theoretical model C_m, The theoretical values are too low. It may result in an optimistic capacitance estimation. Since C_3, C_4, and C_5 are derived, it is decided to use their quantile(0.8) values as the empirical values. For m higher than 5, the theoretical formula is applied in the capacitance estimate program. The average value of quantile(0.8) of C_3 from the three study cases is 0.10908 pf. C_4 is 0.15399 pf and C_5 is 0.21158 pf by the same approach.

7. Results of the Capacitance Estimator

7.1 The Capacitance Estimate Program

In the database of the floorplan created by the floorplanner, each signal contains a linked list of connected terminals. From the linked list, the soft blocks contain the terminals of the signal can be retrieved. An ASCII file is generated by the floorplanner to provide the information of the soft blocks to the capacitance estimator. The file contains the signal name and the center locations of the soft blocks which are connected through the signal.

A capacitance estimate program is implemented to use the file as input and generate the estimated capacitance of each signal. The program checks the center location of input blocks for the repeated numbers. If a block is repeated m times, C_m is calculated as described above to obtain the capacitance within the block. With the center locations of the input blocks, four different near-optimal rectilinear Steiner Trees, growing from bottom, top, left and right, are constructed. The shortest length of the four trees is chosen as the external wire length. The capacitance among multiple blocks is estimated from the external wire length by applying the unit capacitance[12] and unit fringe capacitance[13]. A constant 0.9 micron is chosen as the wire width for the experiments. Both values of the unit capacitance and fringe capacitance are from the 0.9 micron CMOS technology. The capacitance estimate program is attached in Appendix.

7.2 Result of Floorplan-base Capacitance Estimation

The floorplan-based capacitance estimate program was tested with the three study cases. The estimated capacitances are compared with the computed capacitances from post-layout, i.e. compared with the values accurately computed by the real wire length applying to the corresponding unit capacitance. Only the signals which connect through multiple soft blocks are used for the comparison because the other signals consist of C_m only. The histograms of the residual (estimate error) of the three study cases are shown in Figure 13. From Figure 13, it can be seen that the shape of the residuals is a normal distribution and most residuals are small.

12. Unit capacitance 0.00007025 pf / square microns is used which is the average value of polysilicon and metal material.
13. Unit fringe capacitance 0.00005513 pf / micron is used.

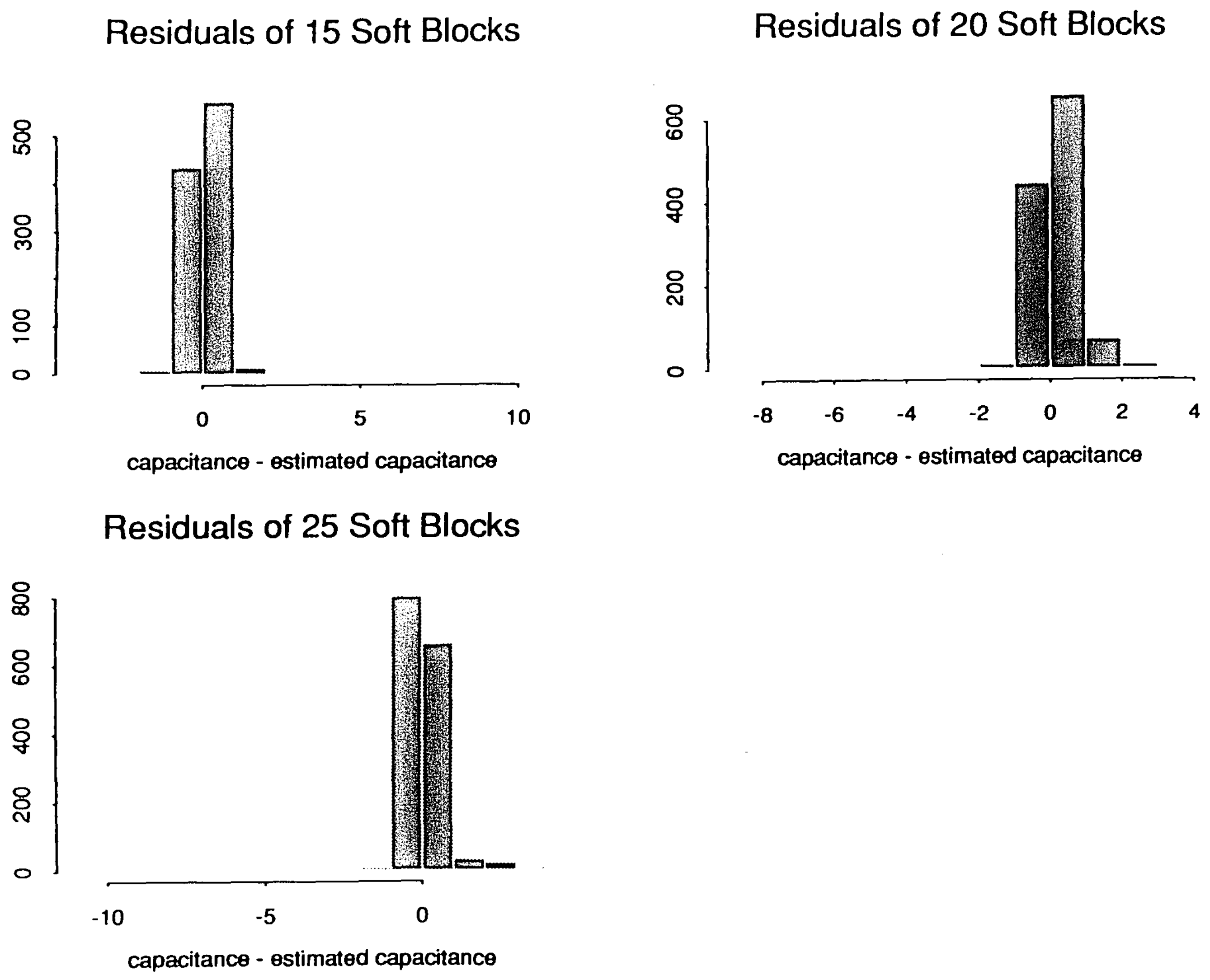

Figure 13. Floorplan-based Residual Histograms of Three Study Cases

The statistics of the comparison are listed in Table 8.

Case No.	Number of Signals	Layouted Mean (pf)	Estimated Mean (pf)	Residual Mean (pf)	Standard Deviation (pf)	Residual / Layouted (%)
Case I	1052	1.09346	0.98264	0.11082	0.73240	10.14
Case II	1177	1.11335	0.88498	0.22837	0.60054	20.51
Case III	1491	0.89253	0.80176	0.09077	0.60085	10.17
Average	1240	1.03311	0.88979	0.143	0.645	13.61

Table 8. Statistics of Floorplan-based Residual

In general, the floorplan-based capacitance estimation achieved 0.143 pf (13.61 %) mean estimation error with the standard deviation 0.645 pf.

7.3 Result of NETCAP Capacitance Estimation

A similar analysis from the NETCAP capacitance estimation was performed. The NETCAP capacitance is estimated by the following formula

capacitance = (number of terminals - 1) * 0.09 pf

The statistics from NETCAP capacitance estimation are listed in Table 9.

Case No.	Number of Signals	Layouted Mean (pf)	NETCAP Mean (pf)	Residual Mean (pf)	Standard Deviation (pf)	Residual / Layouted (%)
Case I	1052	1.09346	0.81428	0.27918	2.65120	25.53
Case II	1177	1.11335	0.77560	0.33776	2.53551	30.34
Case III	1491	0.89253	0.66054	0.23198	2.24964	25.99
Average	1240	1.03311	0.75014	0.283	2.479	27.29

Table 9. Statistics of NETCAP Residual

In the three study cases, the NETCAP capacitance estimation achieved 0.283 pf (27.29 %) mean estimation error with the standard deviation 2.479 pf.

8. Conclusion

The theoretical foundation of the proposed methodology of the floorplan-based capacitance estimation is established by the four experiments (study cases) of a chip layout. The empirical formula required by the capacitance estimator are derived from the four experiments. With the implementation of the capacitance estimate program, the comparison between the NETCAP program and the floorplan-based capacitance estimation was performed.

The result of the comparison shows the NETCAP capacitance estimation achieved 0.283 pf (27.29 %) mean error with the standard deviation 2.479 pf, whereas the floorplan-based capacitance estimation achieved 0.143 pf (13.61 %) mean error with the standard deviation 0.645 pf. The floorplan-based capacitance estimation doubles the accuracy in the mean estimation error and reduces the standard deviation to one-fourth.

REFERENCES

1. Cadd User's Guide, A. D. Schapira, AT&T Bell Laboratories, August 5, 1992
2. Cell-Level Delay Calculator Release 1.0, Reference Manual 1.0, A. D. Schapira, F. C. Chang, AT&T Bell Labs, August 22, 1991
3. A New Algorithm For Floorplan Design, D. F. Wong, Department of Computer Sciences, University of Texas at Austin, 1986 DAC pp. 101-107
4. Timing-Driven Layout Using Floorplanner, Chi-Chang Liaw, AT&T Bell Labs, July 10, 1992
5. An Implementation Plan of Incorporating Capacitance Estimation in The HCPA System for Timing-Driven Layout, C. K. Liang, AT&T Bell Labs, September 28, 1992
6. A Wire Length Estimation Technique Utilizing Neighborhood Density Equations, Takeo Hamada, Chung-Kuan Cheng, and Paul M. Chau, University of California at San Diego, 1992 DAC Paper 5.3
7. Partition User's Manual, Miron Abramovici, AT&T Bell Labs, March 1992
8. An Algorithm For Hierarchical Floorplan Design, K. S. The, Department of Computer Sciences, University of Texas at Austin, ICCAD 1989, pp. 484-487
9. Combinatorial Algorithms for Integrated Circuit Layout, Thomas Lengauer, University of Paderborn, Applicable Theory in Computer Science, A Wiley-Teubner Series
10. The New S Language, Richard Becker, John Chambers, Allan Wilks, AT&T Bell Labs.

APPENDIX

```
/*
 * This is a program to estimate capacitance value from the block
 * information of the signals after the floorplanning process
 */
#include <stdio.h>

typedef struct sblk {
      float xc, yc; /* center location of block */
      float xh1;
      float xh2;
      float yh;
      float xv;
      float yv1;
      float yv2;
      struct sblk *fptr; /* forward pointer for processing */
      struct sblk *bptr; /* backward pointer for processing */
      float sortVal; /* value to be used for sorting */
      int index; /* index number */
} BLOCK;

#define UNIT_CAP 0.00007025 /* 0.9 CMOS unit capacitance */
#define FRINGE_CAP 0.00005513 /* 0.9 CMOS unit fringe capacitance */
#define WIDTH 0.9 /* 0.9 CMOS default wire width */

#define C2 0.064127 /* 2-pin capacitance quantile(0.8) */
#define C3 0.109080 /* 3-pin capacitance quantile(0.8) */
#define C4 0.153990 /* 4-pin capacitance quantile(0.8) */
#define C5 0.211580 /* 5-pin capacitance quantile(0.8) */

#define MAXBLK 8000 /* maximum number of blocks of a signal */
#define INF 1073741823

#define MAX(A,B) ((A)>(B) ? (A) : (B))
#define MIN(A,B) ((A)<(B) ? (A) : (B))

/* return rectlinear distance between two segments a, b */
#define RECTLINEAR(xa1,ya1,xa2,ya2,xb1,yb1,xb2,yb2)
      (MAX(0,MAX(xa1-xb2,xb1-xa2))+MAX(0,MAX(ya1-yb2,yb1-ya2)))

/* return 1 if block b1 is the same as block b2 */
#define IS_SAME_BLOCK(b1,b2)
```

```
     (abs((b1).xc-(b2).xc) < 0.5 && abs((b1).yc-(b2).yc) < 0.5)

BLOCK block1[MAXBLK]; /* block array */
float blk_xc[MAXBLK], blk_yc[MAXBLK]; /* block center coordibates */
int nr_index; /* number of index */

extern void qsort();

/* main program to read input file and write output file */
main(argc, argv)
int argc;
char *argv[];
{
     FILE *fd1, *fd2;
     char sigName[80];
     int i, j, nr_block;
     float sigValue, estimate();

     if(argc < 3) {
       fprintf(stderr,
        "Usage: %s cap_input_file cap_output_file0, argv[0]);
       exit(1);
     }
     if(!(fd1 = fopen(argv[1], "r"))) {
       fprintf(stderr, "Can't open cap_input_file0, argv[1]);
       exit(1);
     }
     if(!(fd2 = fopen(argv[2],"w"))) {
       fprintf(stderr, "Can't open cap_output_file to write0,
       argv[2]);
       fclose(fd1);
       exit(1);
     }
     while((i= fscanf(fd1,"%s",sigName)) == 1) {
          i = 0;
          while((j= fscanf(fd1,"%d %f %f", &nr_block, &blk_xc[i],
          &blk_yc[i++])) == 3);
          if(nr_block < 2)
               sigValue = 0.0;
          else
               sigValue = estimate(nr_block, blk_xc, blk_yc);
          fprintf(fd2, "%s %.6f", sigName, sigValue);
          if(sigValue > 0) {
               for(i=1; i <= nr_index; i++)
```

```
                fprintf(fd2, " %d",block1[i].index);
            }
            fprintf(fd2, "0);
        }
        fclose(fd1);
        fclose(fd2);
        exit(0);
}

/* process capacitance estimation */
 float
estimate(nr_block, blk_xc, blk_yc)
int nr_block;
float blk_xc[], blk_yc[];
{
        BLOCK tmp_blk;
        int i, j, same;
        float internalCap, externalCap, wireLength;
        float estimateExternal(), int_capacitance(), ext_capacitance();

        /* build block1[] array */
        block1[1].xc = blk_xc[0];
        block1[1].yc = blk_yc[0];
        block1[1].index = 1;
        nr_index = 1;
        for(j=1; j < nr_block; j++) {
            same = 0;
            tmp_blk.xc = blk_xc[j];
            tmp_blk.yc = blk_yc[j];
            for(i=1; i <= nr_index; i++) {
                if(IS_SAME_BLOCK(tmp_blk, block1[i])) {
                    same = i;
                    break;
                }
            }
            if(same)
                ++(block1[same].index);
            else {
                block1[++nr_index].xc = blk_xc[j];
                block1[nr_index].yc = blk_yc[j];
                block1[nr_index].index = 1;
            }
        }
```

```
    /* estimate internal capacitance */
    internalCap = 0.0;
    for(i=1; i <= nr_index; i++) {
        if(block1[i].index > 1) /* m-pin signal */
            internalCap += int_capacitance(block1[i].index);
    }

    /* estimate external capacitance */
    wireLength = estimateExternal(nr_index, block1);
    externalCap = ext_capacitance(wireLength);
    return(internalCap + externalCap);
}

/* return internal capacitance value */
 static float
int_capacitance(m)
int m;
{
    if(m < 2) return(0.0);
    switch (m) {
    case 2: return(C2);
    case 3: return(C3);
    case 4: return(C4);
    case 5: return(C5);
    default:
          return(C2*(m*m+2*m+2.)/(3.*m));
    }
}

/* return external capacitance value */
 static float
ext_capacitance(wireLength)
float wireLength;
{
    double dcap;
    float fcap;
    dcap = UNIT_CAP * wireLength * WIDTH +
         FRINGE_CAP * 2 * (wireLength + WIDTH);
    fcap = dcap;
    return(fcap);
}

/* compare function for qsort() */
 static int
```

```
cmpfunc(blk1, blk2)
BLOCK *blk1, *blk2;
{
     if(blk1->sortVal < blk2->sortVal) return(-1);
     if(blk1->sortVal > blk2->sortVal) return(1);
     return(0);
}

/* estimate external wire length
 * Algorithm:
 *  For all the given block[n] array,
 *  four different near-optimum rectilinear Steiner trees
 *  are calculated in the IBM-style:
 *              growing from bottom,top,left, and right.
 *  The shortest tree is chosen as the representation of the net
 *  and the length of the net is returned.
 */
 static float
estimateExternal(n, block)
int n;
BLOCK block[];
{
     BLOCK wp[MAXBLK]; /* temporary array for qsort() */
     BLOCK *ts;
     register BLOCK *tt, *tr, *wpi;
     register int i, k;
     int kk,flg,rdir,rd,nb,cmpfunc();
     float bwire, bdist,dd,dx,dy,twire,xx,yy;

     /* copy input array onto wp */
     for(i=1; i <= n; i++) wp[i] = block[i];

     bwire=INF; /* initialize the best wire length */

     /* directions to build up the tree:
       0=from bottom, 1=from top, 2=from left, 3= from right */
     for(k=0; k < 4; k++) {
       twire=0;  /* total wire for this direction of the tree */
       /* initialize data and prepare for sorting */
       for(i=1; i <= n; i++) {
           wpi= &wp[i];
           wpi->index=0;
           wpi->fptr=NULL;
           wpi->bptr=NULL;
```

```
    wpi->xh1=wpi->xh2=wpi->xv=wpi->xc;
    wpi->yh=wpi->yv1=wpi->yv2=wpi->yc;
    switch(k){
    case 0:
          wpi->sortVal=wpi->yc;
          break;
    case 1:
          wpi->sortVal= -wpi->yc;
          break;
    case 2:
          wpi->sortVal= wpi->xc;
          break;
    case 3:
          wpi->sortVal= -wpi->xc;
          break;
    default:
          return(1);
    }
}

/* sort the terMINal records according to sortVal */
qsort(&(wp[1]),(unsigned)n,sizeof(BLOCK),cmpfunc);

/* setup the starting point */
wp[1].index = 1;

/* add blocks one by one */
/* i=1 is already part of the tree */
for(i=2; i <= n; i++) {
    wpi= &wp[i];
    if(wpi->index == 0) {
      xx=wpi->xc;
      yy=wpi->yc;
      bdist=INF; /* best distance so far */
      for(tt= &(wp[1]); tt!=NULL; tt=tt->fptr){
          dd=RECTLINEAR(tt->xh1,tt->yv1,tt->xh2,
              tt->yv2,xx,yy,xx,yy);
          if(bdist>dd){
              bdist=dd;
              tr=tt;
      }
    }
    twire += bdist;
    dx=MIN(xx,tr->xh2);
```

```
dx=MAX(dx,tr->xh1);
dy=MIN(yy,tr->yv2);
dy=MAX(dy,tr->yv1);
wpi->xh1=MIN(dx,xx);
wpi->xh2=MAX(dx,xx);
wpi->yv1=MIN(dy,yy);
wpi->yv2=MAX(dy,yy);
switch(k){
case 0:
case 1:
    wpi->yh=yy;
    wpi->xv=dx;
    break;
case 2:
case 3:
    wpi->yh=dy;
    wpi->xv=xx;
    break;
default:
    return(2);
}

/* remove sections that are under the new segment */
/* always leave wp[1] in, pointers are simple then */
tr= &(wp[i]);
for(tt=wp[1].fptr;tt!=NULL;tt=ts) {
    ts=tt->fptr;
    if(tt!=tr) {
        flg=0;
        switch(k) {
        case 0:
          if(tt->xh1>=tr->xh1&&tt->xh2
            <=tr->xh2&&tt->yh<=tr->yh)flg=1;
          break;
        case 1:
          if(tt->xh1>=tr->xh1&&tt->xh2
            <=tr->xh2&&tt->yh>=tr->yh)flg=1;
          break;
        case 2:
          if(tt->yv1>=tr->yv1&&tt->yv2
            <=tr->yv2&&tt->xv<=tr->xv)flg=1;
          break;
        case 3:
          if(tt->yv1>=tr->yv1&&tt->yv2
                  <=tr->yv2&&tt->xv>=tr->xv)flg=1;
                 break;
               default:
                 return(3);
               }
               if(flg==1){
                 tt->bptr->fptr=tt->fptr;
                 if(tt->fptr!=NULL)
                   tt->fptr->bptr=tt->bptr;
               }
           }
         }
       }
   }
   if(twire<bwire){
       /* remember the better solution */
       bwire=twire;
   }
 }

 /* display the tree
 printf("resulting tree:0);
 for(i=1;i<=n;i++)
   printf("x=%f y=%f xh1=%f xh2=%f yh=%f xv=%f yv1=%f yv2=%f0,
   wp[i].xc,wp[i].yc,wp[i].xh1,wp[i].xh2,wp[i].yh,wp[i].xv,
   wp[i].yv1,wp[i].yv2);
 */
 return(bwire);
}
```

VITA

Chih-kuo Liang was born in PingTung, Taiwan, on November 22, 1952. He entered National Cheng Kung University, Tainan, Taiwan, where he received the Bachelor of Science degree in Civil Engineering in June 1975. After his two-year military service as an instructor in the Army Engineering School, he went back to National Cheng Kung University and received the Master of Science in Surveying and Photogrammetry in June 1979. From August 1979 to September 1981, he was an Assistant Researcher in Chung Shan Institute of Science and Technology, Taiwan. In September 1981, he entered the Graduate School of The Ohio State University at Columbus Ohio. From January 1982 to March 1985, he was a Graduate Research Associate in the Department of Geodetic Science and Surveying and received the Master of Science in Geodetic Science and Surveying in December 1983 from The Ohio State University. He was with Intergraph Corporation in Huntsville, Alabama as a Senior System Engineer from March 1985 to June 1988. From August 1985 to June 1988, he was a part-time Instructor in Alabama A&M University, Department of Computer Science, Normal, Alabama. Since June 1988, he has been a Member of Technical Staff at AT&T Bell Laboratories. His work is in the area of circuit layout automation with Computer Aided Engineering and Design.

END

OF

TITLE