Statistical Timing Driven Partitioning for VLSI Circuits

Abstract – In this paper we present statistical-timing driven partitioning for performance optimization. We show that by using the concept of *node criticality* we can enhance the Fiduccia-Mattheyses (FM) partitioning algorithm to achieve more than 20% improvements in terms of timing, among partitions with the same cut size. By incorporating mechanisms for timing optimization at the partitioning level, we facilitate wire-planning at high levels of the design process. This is a different perspective than the traditional approaches that use techniques such as gate duplication, retiming and buffer insertion for delay optimization in later stages of the design process. Experimental results show that significant improvements in terms of delay (up to 40%) can be obtained.

1. Introduction

Partitioning is an early and very important step during the design process not only for the fact that it makes successive design steps like placement, floorplanning and routing manageable, but also because it influences the overall performance of the circuit [1]. The conventional objective of partitioning has been to minimize the number of connections between partitions (cut size or congestion). However, because timing has started to be dominated by the wiring delay, it is imperative to account for timing during partitioning.

Timing-driven partitioning approaches can be categorized into two classes. The first class includes top-down approaches that use netlist alteration (date duplication, buffer insertion, etc.) in order to meet some delay constraint while minimizing the cut size. They are usually based on the Fiduccia-Mattheyses (FM) recursive min-cut partitioning method followed by placement algorithms [14] [17]. Timing optimization is done by minimizing the delay of the most critical path.

The second class includes bottom-up clustering-based approaches. They can be augmented by netlist alteration or min-cut algorithms. As examples [10], [12] and [15] use node labeling and replication for clustering with minimum number of clusters along the critical paths. The focus is on delay improvement but the cut size is not considered and sometime gate replication can be massive. The approach in [20] presents the optimization of both delay and power. Retiming and gate duplication for cut size/delay tradeoff can be used during [4] or after [14] partitioning. Detailed surveys on partitioning can be found in [1], [7].

For the purposed of timing-driven iterative partitioning, statistical timing analysis [3] [6] [8] [9] is a better alternative than static timing analysis for two reasons. First, statistical timing analysis is more robust and can capture variations in gate and wire delays due to fabrication variations and changes in the supply voltage and temperature [3] [6] [8] [9]. Second, static timing analysis is a critical path-based technique. It has to simultaneously observe all the critical paths in the circuit. Furthermore, is should consider the effect of false paths, which in turn is difficult and can result in huge computation times. In contrast, statistical timing analysis assigns to notion of criticality to nodes, and not paths. Hence, it is able to globally optimize the timing in a short period of time.

Until now, all previous timing-driven partitioning approaches used static timing analysis. Due to the advantages of the statistical timing analysis, we propose to use the statistical timing analysis [3], [6] incorporated in two modified versions of the FM algorithm [5]. This methodology incorporates a better gate and wire delay models at the partitioning level and provides better estimates of the timing of the circuit.

Our main contribution is as follows:

- We first propose the statistical timing driven partitioning. Using the concept of criticality (defined in Section 2) we are able to obtain more realistic estimates of the delay of all internal gates and all POs in reasonable computational times.
- We show that considering statistical timing analysis is worthwhile and motivate and formulate the optimization problem.
- We propose two modified FM algorithms for recursive bi-partitioning using a combination of timing and cut size for optimization. Our approach is a different way of performing timing-driven optimization: we drive the partitioning such that the best timing is achieved without performing any netlist alteration techniques (gate duplication, retiming or buffer insertion) which increase circuit area; though such techniques can be independently applied to enhance the timing results of the circuit that is partitioned by our method. Our approach enables us to understand the trade-off between cut size and circuit delay, which is not observable with critical-path based methods [14].

The remainder of the paper is organized as follows. Section 2 briefly presents the concept of criticality within the framework of statistical timing analysis. Section 3 motivates our approach and formulates the optimization problem. Section 4 describes the two different strategies for statistical timing driven recursive bi-partitioning.

Experimental results are presented in Section 5. We conclude the paper by suggesting further research directions, in Section 6.

2. Statistical Timing Analysis

In this section we present the idea of criticality within the framework of statistical timing analysis versus static timing analysis. Such a notion will help in presenting the motivation behind our approach in Section 3. The idea of static timing analysis is to compute the slack for every gate based on the difference between latest arrival times and the required arrival times. Furthermore, each gate is assumed to have a constant delay value. However, in reality there are several uncertainties in both gate and wire delays. Such sources are fabrication variations, estimation error of wire capacitance and resistance, uncertainties of wire capacitance during physical design, changes in the supply voltage and temperature, diversity in signal waveforms, etc. [6] [13] [18]. These uncertainties are modeled in statistical timing analysis by considering gate and wire delays as stochastic variables with certain means and standard deviations. Different approaches of statistical timing analysis have been proposed [8] [11]. We adopt the statistical timing analysis method proposed in [3] and later improved in [6] due to the introduction of the criticality concept that fits well into the partitioning framework, as we will see later.

The idea of statistical timing analysis is that gate and wire delays are modeled as statistical variables. Generally, for an n-input gate (see Fig.1.a), under the assumption of stochastic independence of the inputs, the *maximum* latest arrival time at all inputs can be modeled with a normal distribution whose probability density function is [6]:

$$f_{maxPIs}(x) = \sum_i^n \left[f_i(x) \cdot \prod_{j \neq i}^n F_j(x) \right] \tag{1}$$

where f_i and F_j are the probability density function (pdf) and cumulative density function (cdf) of input i respectively.

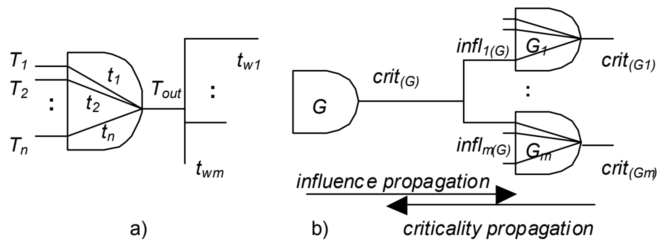

Fig.1 a) General gate b) Influence and criticality concepts

Since the internal gate delay (with the approximation that the delay from each input to output is the same) is also normally distributed, the gate output delay is calculated as the sum of two normal distributions: the maximum of all inputs and the internal gate delay. Wire delays are also considered stochastic variables. Hence, we can compute the probability density function of the overall circuit delay by computing the pdf of each PO. The notions of *influence* and *criticality* [6] are defined in statistical timing analysis to serve as the concept of slack in static timing analysis. In what follows we briefly present the idea of influence and criticality. The term between brackets in equation (1) represents the following probability:

$$f_i(x) \cdot \prod_{j \neq i}^n F_j(x) = P(T_i + t_i = x) \cdot \prod_{j \neq i}^n P(T_j + t_j \leq x) \tag{2}$$

The probability $P(T_i+t_i=x)$ expresses the magnitude of the *influence* that the i-th input gives to f_{maxPIs} at x. The influence $infl_i$ is defined as the influence proportion of the i-th input in the range $x>x_1$ as follows:

$$infl_i = C_1 \cdot \int_{x_1}^{\infty} f_i(x) \cdot \prod_{j \neq i}^n F_j(x) \cdot \exp(C_2 x) dx \tag{3}$$

where C_1 is a normalization coefficient to satisfy $\sum_i infl_i = 1$ and C_2 is a constant to emphasize the region of large arrival time. Criticality is meant to represent the timing criticality at each gate, i.e. the contribution to the circuit delay of the paths that pass through that gate, and is computed using the following equation (see Fig.1.b):

$$crit(G) = \sum_j^m infl_{i(G)} \cdot crit(G_j) \tag{4}$$

Equation (4) defines influence $infl_{i(G)}$ as how much the $i(G)$-th input affects the timing at gate G_j for $x \geq x_1$. In other words, $infl_{i(G)}$ represents how easily the timing criticality back-propagates from gate G_j to gate G. All influences are computed by propagation from primary inputs (PIs) towards POs. Criticalities are computed by back-propagation from POs towards PIs. *The gate(s) with the largest criticality in a circuit is the most critical in terms of timing since its contribution to the circuit output delays is the most significant among all gates in the circuit.* Details can be found in [3], [6].

The complexity of this statistical timing analysis and the calculation of all criticalities in the circuit are linear with respect to the circuit size, which makes it appealing especially for large circuit sizes.

In the next section, we motivate our criticality-based timing analysis in the context of partitioning for performance and formulate the statistical timing driven partitioning for delay minimization.

3. Motivation and Problem Formulation

In this section we motivate our proposed statistical-timing driven partitioning for performance and formulate the problem that we try to solve in this paper.

To further motivate the use of statistical timing analysis in partitioning, we consider circuit C17 as shown in Fig.2.a.

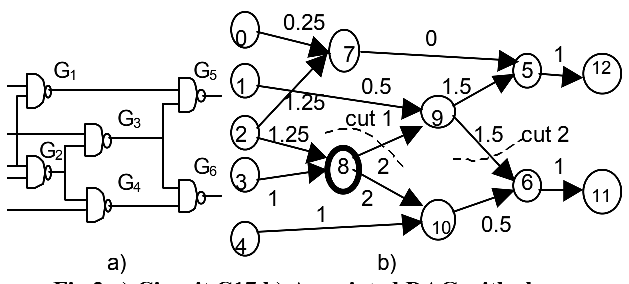

Fig.2 a) Circuit C17 b) Associated DAG with shown criticalities

Its associated directed acyclic graph (DAG) is shown in Fig.2.b, which also shows all computed criticalities at the output of all gates. Recalling the physical meaning of criticality, Fig.2.b shows that the most critical gate in this circuit is G2 (vertex 8 in DAG). That is, gate G2 contributes mostly to the output delays because its criticality, which equals 2, is the largest among all gates in the circuit

If the standard FM algorithm were used to partition this circuit, there would be no difference between the case when the hyperedge {8, 9, 10} is cut and the case when the hyperedge {9, 5, 6} is cut. That is because in the standard FM algorithm the cost function is simply the cut size and the output delays are assumed to be the same no matter which hyperedges are cut along the most critical path, as long as the number of cut hyperedges are the same. So, in a static timing analysis framework, cuts 1 and 2, shown in Fig.2.b, are equivalent. However, we simulated the two cases with HSpice. Fig.3 shows the waveform at the output of the gate G6 in both cases.

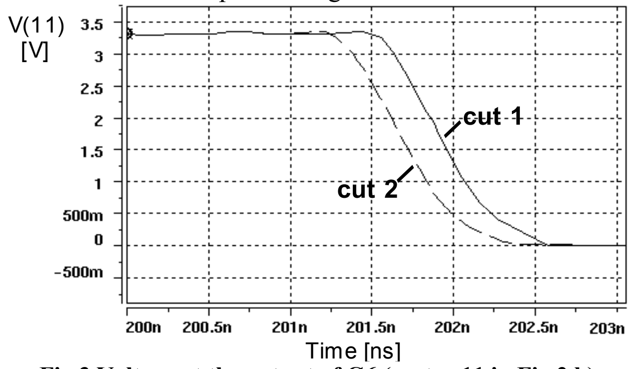

Fig.3 Voltage at the output of G6 (vertex 11 in Fig.2.b)

We modeled the interconnections with the RC lumped model for a 0.18u copper process technology (unit length resistance r=0.115, unit length capacitance c=0.00015).

As it can be seen when the hyperedge with largest criticality is cut (cut 1) the output delay is larger than the case where a hyperedge with a smaller criticality is cut (cut 2).

The above discussion leads us to the conclusion that we need an appropriate timing measure to compare different partitions. We propose to use the criticality concept to drive the partitioning such that both delay and cut size are minimized. By using criticality in the cost function of our mincut-based partitioning algorithm, we use more global information, in contrast to previous net-delay based techniques, which suffer from exploiting only local information. We can formulate the following statistical timing driven partitioning problem.

Problem Formulation: Given a circuit represented by a directed graph G=(V, E), where V is the set of all vertices and E is the set of all hyperedges, and given delays for all vertices (gates) as pdfs, find a k-way partitioning such that the worst delay of any of the POs, the overall cut criticality and the cut size (i.e. congestion) are minimized.

In the next section, we will present two different strategies, both modifications of the standard FM, that try to solve the above problem.

4. Statistical Timing Driven Partitioning Approaches

In this section we justify the use of FM algorithm as the starting point for the development of our proposed statistical timing-driven partitioning algorithms. Our approach uses a modified version of the FM algorithm. Not only the speed and the quality of partitioning were the reasons behind our choice, but also the fact that it can easily incorporate modifications of the cost function allowing us to implement more variations. Additionally, the FM algorithm adapts well to recursive partitioning.

In what follows the reader is assumed to be familiar with the FM algorithm and its data structure [5]. The pseudo-code of the modified FM algorithm is as follows:

```
1. while (stopping criterion not met) {   // this is a pass
2.     Up-date criticalities   // Str.I and Str.II
3.     Partition(V, E) // generate the random  initial partition
4.     Find critical cells   // Str.II
5.     while (unlocked non-critical cells) {   // "non-critical" for Str.II only
6.         Find best cell
7.         Compute cutcrit   // Str.I
8.         Compute cutcrit gain of best cell   // Str.II
9.         Lock best cell
10.        }
11.    Find prefix up to which cutcrit is min with min decay in cut size   // Str.I
12.    Find prefix up to which cut size is min   // Str.II
13.    Swap all best cells up to prefix
14.    Unlock all cells
15.    }
16.    Up-date criticalities and find Delay
```

Through extensive experiments we found that the following two simple strategies are fast and generate satisfactorily accurate solutions:

- *Strategy I.* During each pass, we record the cut-criticalities of the cells that are moved. When choosing the sequence of moves to accept, we pick the one that offers minimum cut-criticality at no more than a user set percentage decay in cut size, compared with the standard FM.

• *Strategy II.* We first order all hyperedges in non-increasing magnitude of their weights, i.e. criticalities, and then constrain the standard FM algorithm not to cut any of the critical hyperedges. A hyperedge is *critical* if it is among the first DONOTCUT% (set by the user) of all hyperedges ordered in non-increasing order of their associated criticalities. All vertices connected by critical hyperedges are called *critical* cells.

In the next section we will present the experimental setup for both proposed strategies that implement the corresponding modified FM algorithms and report experimental results.

5. Experimental Results

In this section we first present the delay model, and the flow diagram of our modified FM algorithms and then we report the experimental results.

5.1. Delay Model

The accuracy of our methodology depends on the delay model that is used. At the early stage of partitioning there is no layout information available. Because of that, we cannot estimate the wire delays with high accuracy. On the other hand, gate delays are known for any particular technology. They can be taken from technology files characterizing all gates in the cell library.

Our delay model has two components. The first component is the gate delay. For all gates we consider a typical intrinsic delay that is given for a typical input transition and a typical output net capacitance. This delay is actually the mean value of the pdf associated to the gate delay. For all pdfs associated to the gates, we consider a standard deviation of 15% from its mean. This standard deviation is realistic for today's technologies [18] though smaller than 25%, which was considered in [9].

The second component is the wire delay. We use the Elmore delay to model the wire delay. The Elmore delay for an edge *e* (an edge corresponds to the wire connecting the net source to one of its fanout sinks) is given by:

$$Delay(e) = R_e(\frac{C_e}{2} + C_t) \tag{5}$$

where R_e is the lumped wire resistance, C_e is the lumped wire capacitance, and C_t is the total lumped capacitance of the source node of each net. To compute R_e and C_e we need an estimate of the length of each edge. For that, we use the statistical net-length estimation proposed in [21]. The average length of a net, connecting *m* cells enclosed in a rectangular area whose width is *a* and whose height is *b*, is given by:

$$L_{av} \approx (\alpha \cdot m^{\gamma} - \beta)\frac{a \cdot b}{a+b} + (a+b) \tag{6}$$

where α, β, and γ are fitting parameters computed in [21] as $\alpha \approx 1.1$, $\beta \approx 2.0$, $\gamma \approx 0.5$. During recursive partitioning, when a net is cut, it is assigned a pre-computed wire delay that will be used to re-compute all delays on the paths that include that net. The earlier a net is cut during recursive partitioning, the larger the back-annotated wire delay will be. In our case, any net that is cut during the first bi-partitioning step (see Fig.4) is assumed to be bounded by a rectangular area which is the same as the chip area and for simplicity we consider an aspect ratio equal to 1. At the second partitioning level, *a* and *b* (see Fig.4) have different values that will ensure a smaller delay than that assigned during a previous partitioning level. The delay of each net is set only the first time that it is cut. In our experiments we consider a 0.18u copper process technology (unit length resistance r=0.115, unit length capacitance c=0.00015).

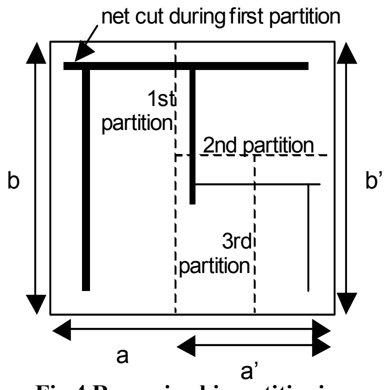

Fig.4 Recursive bi-partitioning

5.2. The Experimental Setup

Our proposed recursive bi-partitioning engines follow the setup shown in Fig.5. The engine takes as input the gate netlist of the circuit. Then, criticalities for all gates are computed as explained in Section 2. Recursive bi-partitioning, based on the standard FM algorithm or using our proposed strategies, is performed. During recursive partitioning we can up-date all criticalities by back-annotating delays to wires corresponding to cut edges and then propagating the effect of the delay forward to compute the new criticalities on all affected hyperedges at the current level of recursive bi-partitioning. Finally, we choose the best partitioning in terms of timing, overall cut size, and overall cut-criticality.

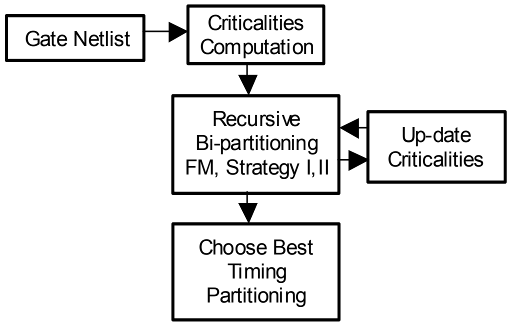
Gate Netlist
Criticalities Computation
Recursive Bi-partitioning FM, Strategy I,II
Up-date Criticalities
Choose Best Timing Partitioning

Fig.5 The experimental setup

5.3. Experimental Results

Because our approach is based on statistical timing analysis, which is different from all previous approaches and because we do not use netlist alteration in order to meet a timing requirement (we simply provide a mechanism to efficiently distinguish among different partitions), we cannot make a fair comparison between our results and previous works.

However, in what follows we provide experimental results that we obtained with the standard FM algorithm and with the two proposed modified algorithms. All experiments were performed on a 450MHz Pentium III PC with 128KB of memory.

First we optimize all benchmarks using the classic *script.rugged* script in SIS [16]. Then, we run the partitioning algorithms 10 times and record the average values for cut size and cut criticality. These values are shown in Table 1. The cut size is the total number of cut hyperedges and the cut criticality is the summation of all criticalities of all cut hyperedges. We use the cut criticality (cutcrit in the tables) as an indicator directly related to the output delay because, as we saw in the motivation section, the fact that hyperedges with large criticalities are not cut helps reduce the output delay. The average delay and the computation times (in seconds, rounded to the closest integer) are also shown in Table 1.

We can see that, both modified algorithms perform better than the standard FM algorithm in terms of delay. However, that is at the expense of a slightly increased cut size. For example, for benchmark *rd84* we obtained a delay of 11.37 using the FM algorithm as opposed to 10.97 and 11.26 obtained with the modified algorithms. The cut size is 269 for the case of the FM algorithm and 261 and 273 for the case of the modified algorithms. The maximum delay difference between the FM approach and the second algorithm is about 40% for the benchmark *apex2*. The second modified FM algorithm offers a delay improvement of 15.64% on average at 34.6% increase in cut size relative to the standard FM algorithm. The first modified FM algorithm offers insignificantly decreased delay with the same cut size. In the majority of the cases, the modified algorithms provide better cut criticalities, which indicates improved circuit delays when these modified algorithms are used.

We note that this difference tends to be more significant for larger circuits, because the larger the circuit, the larger the search space, and thus the better the chances of finding a partition with minimum timing. The difference between minimum and maximum delays can be significant as we can see in Table 2, where we present the minimum and the maximum values recorded during the ten runs.

Since the second proposed algorithm performs slightly better compared to the first one, we studied the delay variation (see Fig.6 and Fig.7) when the percentage of edges that are not allowed to be cut during partitioning is varied between 0% and 7%. Note that the case 0% corresponds to the simple FM algorithm. As we can see, there exists an optimum value for DONOTCUT of about 1-5% for each circuit, which ensures significant delay shifts. We mention that the same behavior is manifested by all benchmarks. The following observations are in order:

- Using criticality within the framework of timing driven partitioning can result in significant delay improvements.
- Timing improvement is significant even when the delay difference is relatively small. That is due to the fact that for all the experimented circuits the largest wire delay is not much larger than the delay of a common cell. This means that for instance in the case of circuit *cordic* in Table 1, a difference of 2.75 between the delay obtained with the standard FM algorithm and that obtained using the second modified algorithm tells us that the most critical path is not cut so many times with the partition obtained with the second modified algorithm. Hence, we expect the delay improvement to be more for larger circuits.
- We noticed that the internal structure of the circuit is very important. Circuits with many large fanout nets are difficult to optimize for delay because most of the partitions are similar. An example is benchmark *des* (see Table 2). We can see that, all ten runs resulted in the same timing.

We conclude this section with an observation about the weak aspect of our methodology. The assumption of event independence in the derivation of equation (1) is erroneous because in reality circuits present spatial and temporal correlations due to fanout reconvergence and input pattern dependencies. However, it was shown in [3] and [19] that for circuits with more than three logic levels the error due to that assumption is less than 5%.

6. Conclusion

In this paper we proposed statistical-timing driven partitioning. We showed that, by using the concept of criticality, it is possible to efficiently distinguish between partitions in terms of timing. We proposed two modified versions of the standard FM algorithm, which offered better timing results at the expense of an increase in cut size. These algorithms are able to constrain the optimization process in the FM algorithm such that the cut-criticality is minimized and thus, the delay of all POs is decreased.

We are currently working on speeding up the criticality and delay up-date at any partitioning level by simplifying influence computation using look up tables and on implementing multilevel counterparts of all recursive algorithm versions discussed in this paper.

Table 1:Recursive partitioning, balance ratio=0.48, 10 random run, minimum timing reported

		FM				Str.I				Str.II			
Circuit	No. of cells	Cut size	Cutcrit	Delay	CPU (s)	Cut size	Cutcrit	Delay	CPU (s)	Cut size	Cutcrit	Delay	CPU (s)
rd73	114	59	19.4	5.89	0	58	16.59	5.87	0	58	14.13	5.8	0
9symml	145	69	12.6	11.38	0	68	10.99	11.2	0	72	11.1	11.19	0
alu4	276	120	53.73	17.62	0	120	51.91	17.5	0	141	47.73	16.56	0
C1355	293	124	352.27	11.4	0	123	320.16	11.07	0	124	308.77	11.4	0
t481	365	249	13.91	12.54	0	251	13.34	12.53	2	257	13.13	12.32	0
rd84	481	269	33.42	11.37	1	261	30.26	10.97	4	273	29.74	11.26	1
table3	686	426	185.26	16.57	1	421	168.7	15.92	10	444	142.7	16.22	1
mul8	820	164	202.4	94.39	2	161	182.07	92.65	12	176	199.47	98.94	2
cordic	856	383	18.29	41.33	2	361	15.62	40.53	12	437	14.59	38.58	2
apex2	2388	979	33.01	133.42	11	965	30.13	130.66	186	1444	14.59	79.82	12
des	2557	424	1493.1	193.71	16	446	1502	193.71	354	971	703.18	173.13	16
	Average	**296.9**	**219.76**	**49.96**	**3**	**294**	**212.8**	**49.32**	**52.7**	**399.7**	**136.28**	**43.2**	**3.09**

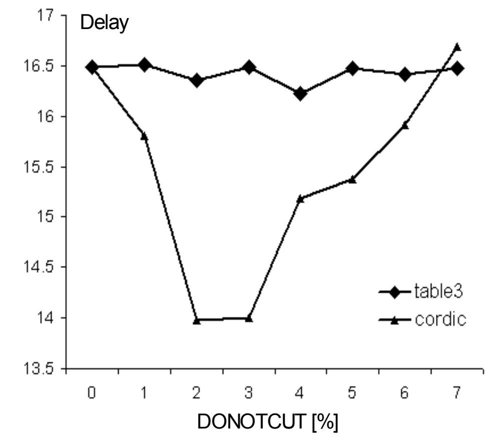

Fig.6 DONOTCUT influence on timing

Table 2: Recursive partitioning, balance ratio=0.48, minimum and maximum timings

			FM				Str.I				Str.II			
			Minimum		Maximum		Minimum		Maximum		Minimum		Maximum	
Circuit	No. of cells	PI/PO	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
rd73	114	7/3	5.72	0.23	6.05	0.22	5.71	0.23	6.08	0.22	5.68	0.23	5.96	0.23
9symml	145	9/1	11.03	0.48	11.74	0.43	11.02	0.39	11.44	0.45	10.85	0.46	11.62	0.41
alu4	276	10/6	17.11	1.03	18.47	1.18	16.77	1.1	18.36	1.23	16.01	0.99	16.87	1.08
C1355	293	41/32	11.12	0.59	12.21	0.61	10.5	0.55	11.65	0.6	11	0.6	12.02	0.6
t481	365	16/1	11.73	0.81	14.34	1.01	11.31	0.81	14.34	1.03	11.64	0.84	13.1	0.92
rd84	481	8/4	10.94	0.51	12.06	0.54	9.92	0.53	11.84	0.48	10.43	0.55	11.85	0.47
table3	686	14/14	14.75	0.72	17.74	0.95	14.56	0.89	19.37	1.05	15.04	1.04	17.34	1.18
mul8	820	32/16	83.68	6.98	99.46	8.31	87.65	7.32	98.48	8.23	88.8	7.42	105.1	8.78
cordic	856	23/2	37.76	3.14	44.76	3.71	35.64	3.02	42.11	3.44	37.27	3.05	42.6	3.55
apex2	2388	39/3	132.75	11.33	135.33	11.63	124.71	10.71	134.79	10.76	78.48	6.49	80.38	6.63
des	2557	256/245	193.71	17.2	193.71	17.21	193.71	17.21	193.71	17.21	173.13	14.3	173.13	14.3

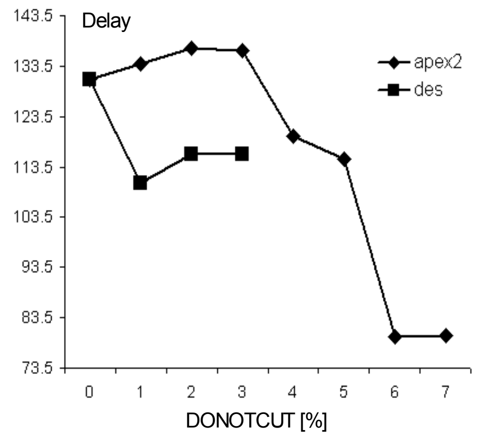

Fig.7 DONOTCUT influence on timing

References

[1] C.J. Alpert, A.B. Kahng, 'Recent Developments in Netlist Partitioning: A Survey', *Integration: the VLSI Journal*, 1995.

[3] M. Berkelaar, 'Statistical Delay Calculation, a Linear Time Method', *Proc. TAU*, 1997.

[4] J. Cong, S.K. Lim, C. Wu, 'Performance Driven Multi-level and Multiway Partitioning with Retiming', DAC2000.

[5] C.M. Fiduccia, R.M. Mattheyses, 'A Linear-Time Heuristic for Improving Network Partitions', *Proc. ACM/IEEE DAC*, 1982.

[6] M. Hashimoto, H. Onodera, 'A Performance Optimization Method by Gate Resizing Based on Statistical Static Timing Analysis', *IEICE Trans. Fundamentals*, Dec. 2000.

[7] S. Hauck, G. Borriello, 'An Evaluation of Bipartitioning Techniques', *Chapel Hill Conference on Advanced Research in VLSI*, 1995.

[8] H.-F. Jyu, S. Malik, S. Devadas, K.W. Keutzer, 'Statistical Timing Analysis of Combinational Logic Circuits', *IEEE Trans. VLSI Systems*, June 1993.

[9] H.-F. Jyu, S. Malik, 'Statistical Delay Modeling in Logic Design and Synthesis', *Proc. ACM/IEEE DAC*, 1994.

[10] E.L. Lawer, K.N. Levitt, J. Turner, 'Module Clustering to Minimize Delay in digital Networks', *IEEE Trans. Computers*, 1969.

[11] J.-J Liou, K.-T Cheng, S. Kundu, A. Krstic, 'Fast Statistical Timing Analysis By Probabilistic Event Propagation', *Proc. ACM/IEEE DAC*, 2001.

[12] J. Minami, T. Koide, S. Wakabayashi, 'An Iterative Improvement Circuit Partitioning Algorithm under Path Delay Constraints', *IEICE Trans. Fundamentals*, Dec. 2000.

[13] S.R. Nassif, 'Modeling and Forecasting of Manufacturing Variations', *Proc. ACM/IEEE ASPDAC*, 2001.

[14] S.-L Ou, M. Pedram, 'Timing-driven Partitioning Using Iterative Quadratic Programming', http://atrak.usc.edu/~massoud/, Publications section (Coming attractions!)

[15] R. Rajaraman, D.F. Wong, 'Optimum Clustering for Delay Minimization', *IEEE Trans.CAD*, Dec. 1995.

[16] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, A. Sangiovanni-Vincentelli, 'SIS: A System for Sequential Circuit Synthesis', *Technical Report UCB/ERL M92/41*, University of California, Berkeley, May1992.

[17] M. Shih, E.S. Kuh, 'Quadratic Boolean Programming for Performance-driven System Partitioning', *Proc. ACM/IEEE DAC*, 1993.

[18] D. Sylvester, 'Measurement Techniques and Interconnect Estimation', *SLIP00*, 2000.

[19] S. Tsukiyama, M. Tanaka, M. Fukui, 'A Statistical Static Timing Analysis Considering Correlations Between Delays', *Proc. ACM/IEEE ASPDAC*, 2001.

[20] H. Vaishnav, M. Pedram, 'Delay Optimal Partitioning Targeting Low Power VLSI Circuits', *IEEE Trans. CAD*, June 1999.

[21] P. Zarkesh-Ha, J.A. Davis, J.D. Meindl, 'Prediction of Net-Length Distribution for Global Interconnects in a Heterogeneous System-on-a-Chip', *IEEE Trans. VLSI Systems*, Dec. 2000.