行政院國家科學委員會專題研究計畫　期中進度報告

整合電腦架構及實體佈局共同合成之環境(2/3)

計畫類別： 個別型計畫
計畫編號： NSC93-2215-E-002-022-
執行期間： 93 年 08 月 01 日至 94 年 07 月 31 日
執行單位： 國立臺灣大學電子工程學研究所

計畫主持人： 陳中平

報告類型： 精簡報告

處理方式： 本計畫可公開查詢

中　華　民　國 94 年 5 月 31 日

A Yield Improvement Methodology Using Pre- and Post-Silicon Statistical Clock Scheduling

Jeng-Liang Tsai*, DongHyun Baik*, Charlie Chung-Ping Chen[&] and Kewal K. Saluja*
*University of Wisconsin-Madison
Department of Electrical and Computer Engineering
1415 Engineering Drive
Madison, WI 53706
{*jltsai@cae, dbaik@ece, saluja@engr*}*.wisc.edu*

[&]National Taiwan University
Graduate Institute of Electronics Engineering &
Department of Electrical Engineering
Taipei 106, Taiwan
cchen@cc.ee.ntu.edu.tw

***Abstract*— In deep sub-micron technologies, process variations can cause significant path delay and clock skew uncertainties thereby lead to timing failure and yield loss. In this paper, we propose a comprehensive clock scheduling methodology that improves timing and yield through both pre-silicon clock scheduling and post-silicon clock tuning. First, an optimal clock scheduling algorithm has been developed to allocate the slack for each path according to its timing uncertainty. To balance the skew that can be caused by process variations, programmable delay elements are inserted at the clock inputs of a small set of flip-flops on the timing critical paths. A delay-fault testing scheme combined with linear programming is used to identify and eliminate timing violations in the manufactured chips. Experimental results show that our methodology achieves substantial yield improvement over a traditional clock scheduling algorithm in many of the** $ISCAS89$ **benchmark circuits, and obtain an average yield improvement of** 13.6%**.**

I. Introduction

The performance and yield of sequential circuits can be improved significantly by carefully assigning clock arrival times to each flip-flop. This technique is usually called *clock scheduling* [1], [2] or *clock skew optimization* [3], [4], [5], [6]. Because of process variations, the path delays in each manufactured chip are different, and each chip requires a different clock schedule to achieve its best performance. However, the clock trees in existing designs usually do not have tuning capabilities. Moreover, process variations also change clock arrival times and cause *unintentional clock skews*, which aggravate the timing yield problem.

Previous works attempt to improve the timing yield of a sequential circuit by finding a good clock schedule that maximizes the slacks for all paths in different ways. The authors in [5], [6] formulate the clock skew optimization problem as a least square error problem: where the error is defined as the difference between the assigned skew and the middle point of the *permissible range*. Held et al. [1] and Albrecht et al [2] adopt the minimum balance algorithm [7] to find the clock schedule that yields a *lexicographically maximum* slack vector when the slacks are sorted in nondecreasing order. Both approaches do not take into consideration the statistical behavior of process variation.

Recent clock tree designs have started to incorporate tuning capabilities in order to remove unintentional clock skews or speedup timing convergence. Intel's Itanium^{TM} processors use programmable deskew buffers (DSK) or second level clock buffers (SLCB) to counter unintentional clock skews in order to deliver the pre-determined clock schedule in the manufactured chips [8], [9]. Ginosar et al. [10] propose to insert programmable delay elements to the clock input pins of each intellectual property blocks in system-on-chip designs. When the timing budgets are changed over the design cycles, new clock schedules can be achieved without modifying the clock tree designs, thus speed up the design convergence. Due to the cost of programmable delay elements, both techniques enable clock tuning only on a coarse-grain level. They are used to achieve the clock schedule determined during design time.

Takahashi et al. [11] propose a post-silicon clock timing adjustment method that adjusts the clock arrival times to compensate path delay variations. However, there is no systematic ways to determine the programmable delay element locations or to test and program the delay elements.

We present a novel clock scheduling methodology which reduces process variations induced timing violations through a statistical timing driven clock scheduling algorithm and a post-silicon clock tuning scheme. False-path-aware statistical timing analyses on path delays are performed during the design phase and the information is used to find the optimal clock schedule that maximized the performance yield. A small set of flip-flops that can further improve the timing yield with the help of programmable delay elements is identified and a programmable delay element is inserted at the clock input of each flip-flop. A delay-fault testing scheme is used to detect the actual path delays in a manufactured chip and the optimal delay configuration is obtained by linear programming.

The rest of this paper is organized as follows: in Section II we review our clock scheduling methodology. Section III gives the details of our statistical timing driven clock scheduling algorithm. The benefit of post-silicon clock scheduling, a programmable delay element design, and an insertion algorithm are presented in Section IV. Methods for testing and programming the chips are discussed in Section V, and finally Section VI concludes this work.

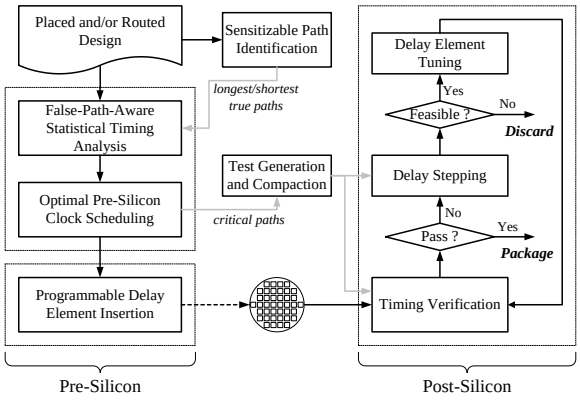

Fig. 1. The proposed clock scheduling methodology.

II. Overview of the Methodology

Our pre-silicon and post-silicon clock scheduling methodology is illustrated in Figure 1. A brief description of each step is as follows:

- **False-path-aware statistical timing analysis**
  It is observed that many structural paths in a circuit are functionally un-sensitizable paths, or false paths [12], [13], [14]. Traditionally, the clock scheduling algorithms obtain the longest and shortest path delays by using static timing analysis tools that usually do not consider false paths. If all the structural longest (shortest) paths between some pairs of flip-flops are false paths, the clock schedule obtained based on this pessimistic expectation of path delays will be sub-optimal. Furthermore, false paths can affect the statistical path delay distributions significantly [15], thus affect the decision on slack allocation. Therefore, it is necessary to perform false-path-aware statistical timing analysis as the first step of clock scheduling.
- **Optimal pre-silicon clock scheduling**
  We improve the clock scheduling algorithm proposed in [1], [2] to take statistical timing information into consideration. The optimal clock schedule for timing yield is obtained.
- **Programmable delay element insertion**
  After pre-silicon clock scheduling, some paths might still have insufficient slacks due to stringent performance requirements. Timing violations can occur in a chip if the path delay variations exceed the pre-allocated slacks. By inserting programmable delay elements at the clock input of the selected flip-flops, timing violations may be corrected by post-silicon clock tuning.
- **Test generation and compaction**
  The timing critical paths identified during clock scheduling might not meet the timing specifications in manufactured chips. Two-pattern tests are generated to excite the longest (shortest) path delays of each timing critical path. Since there may be several structural paths between a pair of flip-flops, test vectors for the k longest (k shortest) true paths need to be generated. Test compaction is performed and a small set of test vectors that can excite multiple longest (shortest) paths simultaneously is obtained.
- **Timing verification**
  Timing closure on the critical paths is delayed until the post-silicon stage for time-to-market and performance requirements. To close the timing the compacted test vectors are applied to excite the longest (shortest) path delays and the outputs are compared against correct values. The chips that pass the tests are ready for the next processing step such as packaging.
- **Delay stepping**
  To perform post-silicon clock scheduling on the chips that failed the timing verification, the actual delays of critical paths must be measured. Test vectors for each critical path are applied repeatedly on different delay configurations and the outputs are compared against the correct value to determine the feasible delay range of each critical path.
- **Delay element tuning**
  The feasible delay ranges on all critical paths are collected. The optimal delay configuration is obtained by linear programming if a feasible configuration exists.

III. Pre-Silicon Clock Scheduling

Pre-allocating timing margins for data paths have been the approach to deal with process variations and inaccuracies of circuit modeling and simulation. To reduce the performance penalty, many clock scheduling algorithms were developed to optimize timing margins. However, most of the existing methods do not take the statistical timing behavior of process variations into account during clock scheduling. We review the techniques for statistical timing analysis and combine our efficient k longest (shortest) true path enumeration algorithm with path-based statistical timing tools to perform false-path-aware statistical timing analysis. A parametric shortest path algorithm is modified to incorporate the statistical timing information to generate the optimal clock schedule.

A. Statistical Timing Analysis

In deep sub-micron designs, statistical timing analysis on path delays is a critical tool used to measuring the growing impacts of process variations. There are two major approaches to obtain path delay distributions. On one hand, a block-based algorithm performs timing analysis by propagating the delay distributions of intermediate nodes to the output in a breadth-first manner. Although the runtime of block-based algorithms is linear to the circuit size, special treatments are required to account for correlations and reconvergence paths. On the other hand, a path-based algorithm can handle correlations and reconvergence of paths. However, the number of paths in a circuit can grow exponentially with respect to the circuit size. Therefore, efficient algorithms are required to select a set of important paths for analysis.

Most of the statistical timing algorithms, either block-based or path-based, do not consider false paths. In other words, the delay distributions are obtained assuming all structural paths are sensitizable. Liou et al. [15] combine a logically

true path selection algorithm with a statistical timer for false-path-aware statistical timing analysis. Runtime is reduced by selecting only the true paths for analysis and the results are more accurate. The logically true path selection algorithm works from PO to PI level by level. New partial paths are created by appending the fan-in gates to the original paths. However, a new partial path will be created only when it is sensitizable and its expected slack is below a predefined threshold. The expected slack is estimated by the worst case statistical timing analysis and is subjected to error. Moreover, the number of true paths the algorithm selects is affected by the threshold value.

For clock scheduling, only the longest and the shortest true paths between pairs of flip-flops are of interest. We develop an implicit path enumeration algorithm which finds the k longest (shortest) true paths efficiently. To find the k longest paths between PI and PO, the expected delay (ED) from each internal node to PO is first calculated. The algorithm traverses from PI to PO by selecting the path with the highest ED and the gates on the selected path are added to a dynamic delay path tree (DPT). For newly added gates, necessary assignments and maximum implications are performed using path sensitization criteria [16] to check their sensitizability. The search continues if sensitization is successful. Otherwise, the EDs are updated as the algorithm backtracks from the current node of the DPT to PI. The algorithm resumes when PO is reached or it backtracks to PI until the k longest true paths are found. The paths are then sent to path-based statistical timers for analysis. By taking the negative expected delay as ED the algorithm finds the k shortest path.

Figure 2 demonstrates our path selection algorithm using a dynamic delay path tree. The numbers on the edges of the DPT are the EDs. The algorithm first searches along $< G1, G2 >$ and backtracks because $G2$ is not sensitizable. The algorithm continues on $< G7, G8, G9 >$ because it has the highest ED from PI. After the second backtracking the algorithm finds the longest true path $< G1, G6, G4 >$. The results of maximum implication are stored in the DPT nodes so that they can be reused when the algorithm backtracks to a previously visited path. Although there are two structurally longest paths, $< G1, G2, G3, G4 >$ and $< G1, G2, G5, G4 >$, they are dropped and never explicitly enumerated as soon as $< G1, G2 >$ is found un-sensitizable.

B. Optimal Clock Scheduling

A sequential circuit can be represented by a directed *circuit graph* $\mathcal{G} = (V, E)$. Each vertex $v \in V$ represents a flip-flop. Each edge $e_{ij} \in E$ represents one or more structural paths from v_i to v_j. The flip-flop delays are counted as parts of the path delays and the setup-time and hold-time of a flip-flop are T_{setup} and T_{hold}. Let T_i be the clock arrival time of v_i and d_{ij} and D_{ij} be the minimum and maximum path delays of e_{ij}, the hold-time constraints are

$$T_i + d_{ij} \quad \geq \quad T_j + T_{hold}. \tag{1}$$

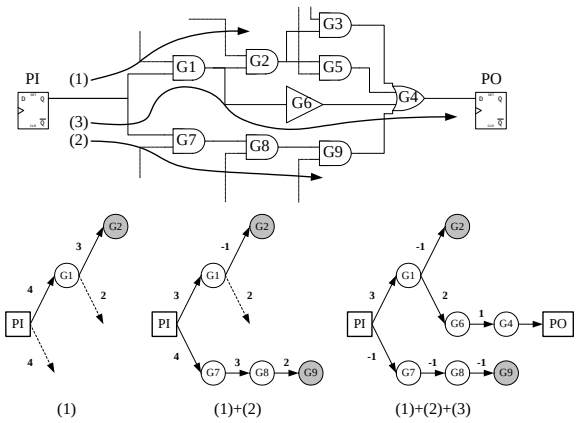

Fig. 2. Implicit enumeration of true paths using dynamic delay path tree.

The setup-time constraints are

$$T_i + D_{ij} \quad \leq \quad CP + T_j - T_{setup}, \tag{2}$$

where CP is the clock period. A feasible clock schedule can be obtained by applying the Bellman-Ford shortest path algorithm [17], [18] on the *timing graph* $\mathcal{G}^{CP}$: replacing each edge e_{ij} in $\mathcal{G}$ with an *h-edge* from v_i to v_j with edge weight $w(e_{ij}) = d_{ij} - T_{hold}$ and an *s-edge* from v_j to v_i with edge weight $w(e_{ji}) = CP - D_{ij} - T_{setup}$.

A clock schedule determined at design time is not always feasible for every manufactured chips since d_{ij} and D_{ij} are random variables. It is important to utilize the statistical timing information of d_{ij} and D_{ij} to find the optimal clock schedule that maximizes the performance yield. As we define the *slack* of e_{ij} in $\mathcal{G}^{CP}$ as $\delta(e_{ij}) = T_i + w(e_{ij}) - T_j$, larger slacks can be achieved through clock scheduling. However, the total slack on a cycle $c =< v_1, v_2, \ldots, v_k, v_1 >$,

$$\delta(c) = \sum_{e_{ij} \in c} \delta(e_{ij}) = \sum_{e_{ij} \in c} w(e_{ij}), \tag{3}$$

is a limited resource and slacks of an edge cannot be arbitrarily increased. Therefore, clock scheduling can be viewed as the process of distributing cycle slacks to edges.

To optimize the timing yield, cycle slacks should be distributed to an edge according to the standard deviation of its associated path delays. We define the *normalized slack* of h-edge e_{ij} as $\lambda_h(e_{ij}) = \frac{T_i + \mu(d_{ij}) - T_{hold} - T_j}{\sigma(d_{ij})}$ and that of s-edge e_{ji} be $\lambda_s(e_{ji}) = \frac{T_j + CP - \mu(D_{ij}) - T_{setup} - T_i}{\sigma(D_{ij})}$. The optimal clock schedule can be obtained by a graph-theoretical algorithm. First, the edge weight of an h-edge e_{ij} in $\mathcal{G}^{CP}$ is replaced by a parametric edge weight of $\mu(d_{ij}) - T_{hold} - \sigma(d_{ij})\lambda$ and the edge weight of an s-edge e_{ji} is replace by a parametric edge weight of $CP - \mu(D_{ij}) - T_{setup} - \sigma(D_{ij})\lambda$. This step yields a *parametric timing graph* $\mathcal{G}^{CP}(\lambda)$. A clock schedule that does not cause any negative cycle in $\mathcal{G}^{CP}(\hat{\lambda})$ guarantees all edges have normalized slacks of at least $\hat{\lambda}$. The *optimal clock scheduling* problem is to find a clock schedule that maximizes λ given a parametric timing graph $\mathcal{G}^{CP}(\lambda)$.

Circuits	Traditional clock scheduling				Optimal clock scheduling				
	# pivot	# contract	yield(%)	CPU(s)	# pivot	# contract	yield(%)	Improvement	CPU(s)
s1488	0	6	72.3	0.4	4	6	75.1	3.9%	0.4
s1494	0	6	77.9	0.4	4	6	78.8	1.1%	0.4
s5378	107	179	63.8	0.4	169	179	64.2	0.6%	0.5
s9234.1	160	210	74.1	0.7	167	210	84.2	13.6%	0.8
s13207.1	276	620	60.2	1.9	397	620	60.2	0.0%	2.0
s35932	1790	1728	64.3	5.3	1108	1728	98.7	53.5%	11.9
s38417	1893	1636	74.0	66.3	1641	1636	88.6	19.7%	79.9
s38584.1	1722	1426	72.8	7.6	1699	1426	84.7	16.3%	9.9

TABLE I

COMPARISONS OF OPTIMAL CLOCK SCHEDULING AND TRADITIONAL CLOCK SCHEDULING.

Held et al. [1] and Albrecht et al. [2] use a *parametric shortest path* algorithm [7] to solve a special case of the problem where $\sigma(\cdot) \triangleq 1$. We adopt the algorithm and assign $\sigma(\cdot)$ using statistical timing information. To find the optimal clock schedule, we start from a feasible clock schedule $\mathbf{T_0}$ obtained by applying the Bellman-Ford shortest path algorithm on $\mathcal{G}^{CP}(\lambda_0)$, where $\lambda_0 = 0$. Since CP is always chosen to be greater than the optimal clock period CP^*, which can be obtained by [4], $\mathbf{T_0}$ always exists. *Path pivoting* [7] is performed and new pairs of $(\lambda_k, \mathbf{T_k})$ are found where $\lambda_k \geq \lambda_{k-1}$. When a zero-cost cycle is detected, the *cycle contraction* [7] step is performed. The zero-cost cycle corresponds to a cycle in the circuit where the normalized slack on the cycle cannot be increased further. The current clock schedule on the cycle is thus optimal and the cycle can be contracted and replaced by a *super vertex*. The timing constraints between the rest of the circuit and the vertices on the cycle are updated and imposed to the super vertex. In other words, the edges from/to the cycle are replaced by new edges from/to the super vertex. When $\sigma(\cdot) \triangleq 1$ multiple edges from v_i to v_j can always be collapsed into one edge by discarding non-dominant ones. In our formulation, multiple edges might need to be kept if their parameter coefficients are different. The path pivoting and cycle contraction steps are repeated until the timing graph becomes a single super vertex.

C. Comparisons

The parametric shortest path algorithm can be viewed as a graph-theoretical version of the simplex algorithm [19], in which a basic feasible solution is moved to another basic feasible solution with increasing objective through row pivoting on the simplex tableau. Young et al. [7] prove that for a graph with $|V| = n$ and $|E| = m$, the parametric shortest path algorithm takes $O(nm + n^2 logn)$ time on the special case where $\sigma(\cdot) \triangleq 1$. Although the worst case running time of the simplex algorithm is exponential in some specially synthesized problems, it usually takes much less time for general problems [20]. We have found that the number of iterations required for obtaining the optimal clock schedule is comparable to the number of iterations used in solving the special case. Furthermore, the performance yield of the optimal clock schedule is significantly increased in many benchmark circuits.

Table I shows the difference of runtime and yield on *ISCAS89* benchmark circuits between our optimal clock scheduling algorithm using false-path-aware statistical timing information and the traditional clock scheduling algorithm with only static structural longest/shortest path delay information. Gaussian distributions with $(\mu, \sigma) = (1, 0.15)$ are used for gate delays and each gate is assumed independent. The clock periods are chosen so that the timing yields of traditional clock schedules are between 60% to 80%. The clock arrival times of all primary inputs and outputs are assumed the same and the corresponding vertices are contracted to a single super vertex before clock scheduling. Monte Carlo simulations are performed and false path information is used to calculate the timing yields for both traditional and optimal clock schedules. The experiments are conducted on a Pentium III 1GHz PC.

D. Clock Skew Uncertainty and Clock Tree Synthesis

In section III-B, unintentional clock skews are not taken into account because the clock tree design may or may not have been completed before clock scheduling. Several options are available in considering unintentional clock skew during clock scheduling. The first option is to design the initial clock tree based on an estimated clock schedule. The initial clock tree is analyzed and the clock skew uncertainties $\sigma(T_i - T_j)$ are added to the coefficients of λ in $\mathcal{G}^{CP}(\lambda)$. After clock scheduling, the wire widths and buffer strengths of the initial clock tree are adjusted for the new clock schedule and $\sigma(T_i - T_j)$ are re-analyzed. It can take several iterations to obtain the clock tree and the optimal clock schedule. Alternatively, clock tree can be designed for zero skew, then it can be analyzed for $\sigma(T_i - T_j)$ and following this the optimal clock schedule can be computed. In the final implementation, flip-flops with different internal delays can be used or additional local clock buffers are inserted [21], [22] to meet the optimal clock schedule computed above. However, flip-flops with internal delays might consume more power than regular flip-flops.

Besides convergence and power consumption issues, clock tree synthesis performed before clock scheduling cannot take advantage of the critical path information that is available after clock scheduling. This can cause serious performance and yield problems. In the case of a balanced H-tree centered at the origin, two flip-flops that are close to the origin but located in the first and the third quadrant have large clock skew uncertainty because they are driven by two completely different clock distribution paths. Thus, in the presence of a

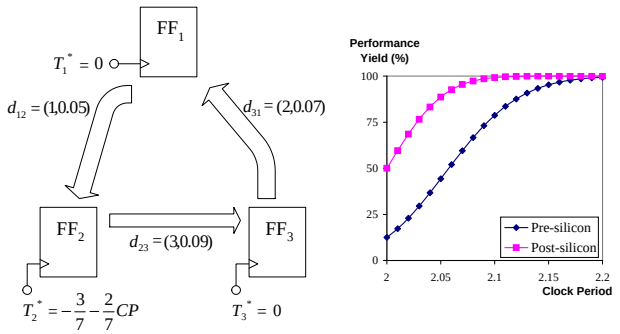

Fig. 3. (Left) a sequential circuit, and (right) timing yields of pre-silicon and post-silicon clock scheduling.

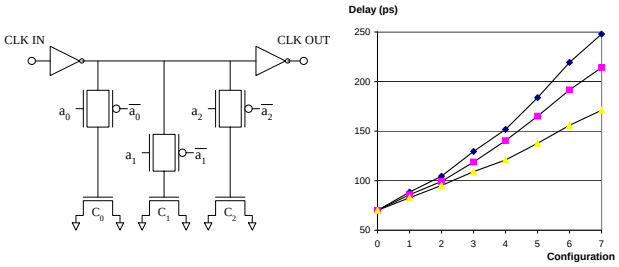

Fig. 4. A programmable delay element circuit and its delay curves for different delay ranges.

critical path between this pair of flip-flops, the timing yield of the whole circuit could be seriously degraded. Therefore, it is necessary to consider the available slacks on each path during clock tree synthesis to control the clock skew uncertainties. The minimum of $\sigma(T_i - T_j)$ is bounded by the physical separation of v_i and v_j. We can first estimate $\sigma(T_i - T_j)$ based on the distance between v_i and v_j, and use the estimations along with the statistical timing information on path delays to perform clock scheduling. After the optimal clock schedule is obtained, we would have the available slacks between every pair of flip-flops. If the available slack between v_i and v_j is abundant, we can allow the subtrees driving v_i and v_j to be merged closer to the clock source to reduce clock tree power. Otherwise, we require two subtrees to be merged earlier to achieve the estimated clock skew uncertainty.

IV. Programmable Delay Element Insertion

The pre-silicon clock scheduling technique tackles timing uncertainty issue by allocating slacks at design time. However, this static approach has its limitations due to two reasons: 1) the processes in new technologies are becoming more difficult to control, and 2) current design trend tends to use fewer gate levels between sequential elements in order to push the clock frequency. As a result, the relative path delay uncertainty is increasing. For example, the delay of an inverter chain with n inverters each having an independent gaussian delay, (μ, σ), is also a gaussian distribution, $(n\mu, \sqrt{n}\sigma)$. The relative path delay uncertainty is $\frac{1}{\sqrt{n}} \times \frac{\sigma}{\mu}$, which increases as n decreases or process uncertainty increases.

The optimal clock schedule for each chip can be determined separately to maximize the performance yield if clock scheduling can be performed after the chips are manufactured. Figure 3 illustrates the benefit of post-silicon clock scheduling over pre-silicon clock scheduling for a sequential circuit containing three flip-flops and three combinational blocks. The path delays of each combinational block are independent gaussian distributions, which are $(1, 0.05)$, $(3, 0.09)$, and $(2, 0.07)$. The optimal pre-silicon clock schedule is also shown in the figure. At a clock period of 2.1, post-silicon clock scheduling achieves a 99.2% performance yield, as oppose to 78.7% with the optimal pre-silicon clock schedule.

We first propose our programmable delay element design and use the clock scheduling information from section III-B to insert delay elements at selected locations to enable post-silicon clock scheduling through fine-grain clock tuning.

A. PDE Design

It is desirable to have delay elements with constant delay steps to facilitate post-silicon clock tuning. In [10], a tapped delay line circuit with a multiplexor to connect the clock output to different points on a clock buffer chain is used. Since the entire buffer chain is switching all the time, this design consumes significant power. Moreover, it requires as many control signals as the number of delay steps. Takahashi et al. [11] use a digital-analog converter to adjust the controlling voltage of a buffer. The adjustable buffer in a $0.18\mu m$ technology gives a minimum of $30ps$ delay step. However, due to the nonlinear characteristic of the design, the delay steps are not equal.

Figure 4 shows the schematic of a programmable delay element design and its delay curves in a $0.13\mu m$ technology with input slew rate at $0.1ns$. The sizes of the transmission gates and their capacitance loads are adjusted to account for the parasitic capacitors and the nonlinearity of the transmission gate resistances. There are three benefits of this design: 1) its delay steps are almost constant, 2) the three control signals are directly used to generate eight different delay configurations, and 3) different delay ranges required at different locations can be achieved by adjusting the loading capacitors. The delay curves of three delay elements tuned for delay ranges of $100ps$, $140ps$, and $180ps$ are shown in Figure 4. The power consumption of the largest delay element operating at $180ps$ is roughly twice as much as that of a D-type flip-flop.

B. Methodology for PDE Insertion

Let $\mathbf{T}^*$ be the optimal clock schedule, $\lambda^*(e_{ij})$ be the normalized slack of e_{ij} given by $\mathbf{T}^*$, and $\lambda_{min}(v)$ be the minimum λ^* of all in-edges and out-edges of v. All the paths connected from/to v have normalized slacks of at least $\lambda_{min}(v)$. A vertex v or an edge e is *timing critical* if $\lambda_{min}(v) < \lambda_{th}$ or $\lambda^*(e) < \lambda_{th}$, where λ_{th} is the normalized slack threshold.

Circuits	Flip-flop counts according to $\lambda_{min}(v)$						
	1∼2	2∼3	3∼4	4∼5	5∼6	6+	<6
s1488	2	2	1	1	0	0	100%
s1494	1	3	1	0	1	0	100%
s5378	0	0	0	0	6	173	3.4%
s9234.1	0	3	9	2	5	191	9.0%
s13207.1	0	0	0	8	0	612	1.3%
s35932	0	0	0	128	160	1440	16.7%
s38417	0	0	111	36	8	1481	9.5%
s38584.1	0	17	48	24	12	1325	7.1%

TABLE II

FLIP-FLOP COUNTS ACCORDING TO THE MINIMUM NORMALIZED SLACKS OF THE CONNECTED PATHS.

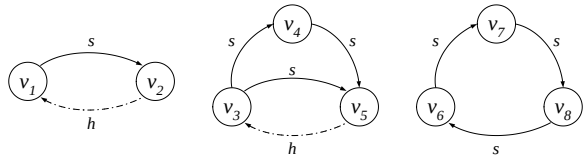

Fig. 5. The timing critical subgraph of $\mathcal{G}^{CP}$ with 3 components.

Table II shows the flip-flop counts according to $\lambda_{min}(v)$. Most of the circuits have less than 17% timing critical flip-flops at $\lambda_{th} = 6$ except the two small circuits, $s1488$ and $s1494$. Tradeoffs between clock tuning capability and area/power can be made by choosing λ_{th}. For example, if we choose $\lambda_{th} = 5$ the number of candidate locations for delay element insertion in $s35932$ drops from 16.7% to 7.4%.

The timing critical vertices and edges form a subgraph of $\mathcal{G}^{CP}$, which consists of zero or more strongly connected components. Figure 5 shows a subgraph with three components, in which h-edges are represented by dashed arcs and s-edges are represented by solid arcs. A straight-forward delay element insertion algorithm inserts a delay element for each timing critical flip-flop, which requires eight delay elements in this example. A better approach is to calculate the *opportunity* of each vertex, which is defined as the performance yield improvement it can achieve by inserting a delay element at this vertex.

Two important metrics are required to evaluate the opportunity: the first is $\lambda_{min}(v)$ and the second is the path delay correlations. A larger $\lambda_{min}(v)$ corresponds to less timing violation probabilites of the paths connected to v, and less yield improvement headroom. Path correlations affect the possibility of correcting a timing violation by post-silicon clock tuning. Strong correlations between path delays can come from several sources. For example, d_{ij} and D_{ij} may be strongly correlated because the longest structural paths and shortest structural paths share common logic gates. Paths located close together can also exhibit strong correlations on their delays due to intra-die and inter-die variations.

The closed form solution of the opportunity can be obtained for a critical component containing one cycle. For a critical component with multiple cycles, one can use Monte Carlo simulation to evaulate the opportunity, or use $\lambda_{min}(v)$ and path delay correlations for estimation. Consider a case in which all path delays are strongly correlated and setup-time violations occur simultaneously on all s-edges in Figure 5. In Figure 5(a), it may be possible to remove the setup-time violation by increasing T_1 or decreasing T_2 without causing hold-time violation on e_{21}. In Figure 5(b), it may be possible to increase T_3 and T_4 together, or decrease T_4 and T_5 together without causing hold-time violation on e_{53}. However, in Figure 5(c) it is not possible to remove the setup-time violations by post-silicon clock tuning. Therefore, the opportunities of v_6, v_7, and v_8 can be lower than that of $v_1 \sim v_5$ even if their λ_{min} values are lower.

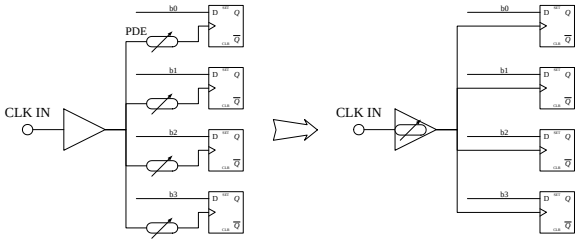

Fig. 6. Reducing number of programmable delay elements.

Further delay element reduction can be achieved by exploring the circuit structure. For example, the path delays of the bit lines in a data bus can be highly correlated if they are routed in close proximity. As shown in Figure 6, we can use a single delay element for the whole bus instead of a separate delay element for each bit.

V. POST-SILICON CLOCK TUNING

Post-silicon clock scheduling can further improve the timing yield by removing timing violations in manufactured chips. It can also speed up time-to-market by avoiding rigorous circuit simulations on all process variation corners to achieve timing closure on the last few timing critical paths. Post-silicon timing verification is the first step to close the timing on the critical paths. Multiple two-pattern tests for each critical path are applied to excite the actual longest (shortest) path delays. Test compaction is performed to reduce the test application time.

The chips that pass the timing verification are ready for the next manufacturing step. Chips with timing violations on the paths without PDEs are discarded. The rest of the chips need to go through post-silicon clock scheduling. First, the feasible delay range of each critical path is obtained through delay stepping. A feasible solution can be generated by linear programming.

A. Test Generation and Compaction

Two-pattern tests which have single or multiple transitions can be used to check for timing violations. Conventional two-pattern test generators typically generate either robust or non-robust two-pattern test sets [23], [24]. Robust tests are guaranteed to detect timing failures on the target path regardless

Circuits	# FF pairs	# total true paths	# critical true paths	Ratio (%)
s1488	266	3848	180	4.7
s1494	266	3902	176	4.5
s5378	2313	48258	2503	5.2
s9234.1	3260	407574	16371	4.0
s13207.1	4721	1488814	24569	1.7
s35932	7595	236290	10368	4.4
s38417	34351	3677596	115736	3.1
s38584.1	20444	1400198	11611	0.8

TABLE III

NUMBER OF TIMING CRITICAL PATHS WITH SLACKS LESS THAN 6σ.

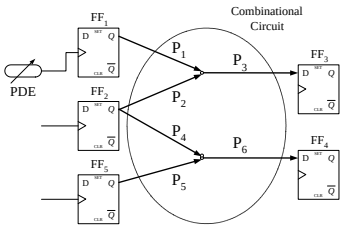

Fig. 7. Example of requirement on test diagnostic capability.

of the signal arrival times of the side inputs. Whereas non-robust tests can only guarantee the fault detection when no other path-delay fault is present. However, robust and non-robust tests do not provide diagnostic capability and a test may produce incorrect output even if the path under test does not have timing fault. Consider the example in Figure 7. A two-pattern test for $< P_2, P_3 >$ might excite the delay fault on P_1. As a result, incorrect result can be captured at FF_3 even if $< P_2, P_3 >$ does not have path delay fault. Although this effect is desirable for conventional delay fault testing, it is not suitable for post-silicon clock tuning. If timing failure exists only on $< P_1, P_3 >$, it may be possible to adjust the PDE of FF_1 to correct the timing violation. Therefore, it is important to distinguish the timing failure on $< P_1, P_3 >$ from that of $< P_2, P_3 >$. A single transition test provides the required diagnostic capability. However, single transition test may not exist for some paths. Delay fault diagnosis techniques [25], [26] can be used or DFT techniques are required to make the path single transition testable for these paths.

Table III shows the number of sensitizable structural paths that are timing critical for $\lambda_{th} = 6$, or have slacks less than 6σ. It is evident that less than 5% of the true paths need to be tested. The numbers of critical true paths in Table III are the upper bounds for the numbers of single transition tests since a single transition test usually can test multiple paths. Static or dynamic compactions can reduce the number of tests as long as the reduced test set provides the same diagnostic capability. Fault diagnosis is required only for the paths connected to PDEs, or the *tunable paths*. Conventional robust or non-robust tests can replace single transition tests to examine non-tunable critical paths to achieve further compaction if they don't cause ambiguity on tunable paths.

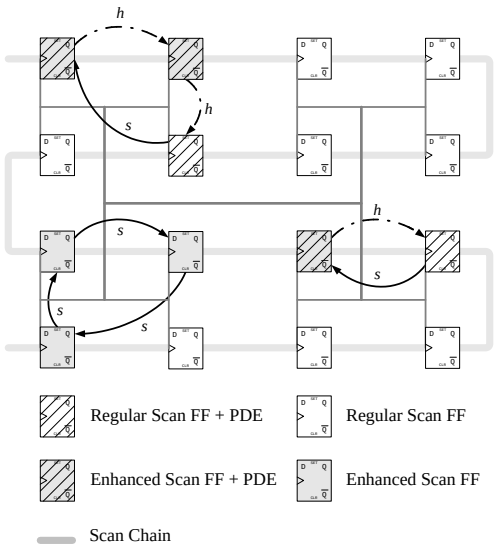

Fig. 8. The delay fault testing scheme for post-silicon clock scheduling.

B. Timing Verification

During timing verification, the tests for non-tunable paths are applied first. Since timing failure on non-tunable paths can not be recovered by post-silicon clock tuning, the chips that failed in this step can be discarded. Next, the tests for tunable paths are applied. The chips that pass the test can be sent to the next manufacturing step and the remaining chips need to go through post-silicon clock scheduling.

Figure 8 shows the delay fault testing scheme for post-silicon clock scheduling. The circuit contains 16 flip-flops that are driven by an H-tree. Each critical component is driven by the same clock branch to reduce the unintentional clock skew. Only timing critical edges are shown in the figure with solid arcs as s-edges and dashed arcs as h-edges.

Four types of flip-flops are used depending on the requirements of two-pattern test capability and tuning capability. First, the flip-flops that are to launch transitions must be able to hold two values. The enhanced scan filp-flop which can hold current value during scan operation (SHFF) [27] can be used for this purpose. Since non-critical paths do not need to be tested, enhanced scan flip-flops are used only at the sources of the critical paths. Note that an s-edge e_{ij} corresponds to a long path from v_j to v_i. A flip-flop on a critical cycle but not a source of any critical path can be a regular scan flip-flop.

Filp-flops with tuning capability are selected by the PDE insertion method described in section IV. Three additional storage elements are attached to each PDE and those storages can be included or separated from the regular scan chain.

C. Delay Stepping and PDE Tuning

The actual longest (shortest) path delay of each timing critical path can be measured by applying single transition tests from its source and observe its outputs. For an h-edge e_{ij} the PDE delays of v_i and v_j are first set to their maximum and minimum to give it the largest slack. If the outputs

contain errors, no feasible configuration exists and the chip is discarded. Otherwise the PDE delay of v_i is reduced and the PDE delay of v_j is increased until an error is detected.

Once all the path delays are known, a set of PDE configurations can be calculated using traditional clock scheduling algorithms with extra constraints on the tuning range of each PDE. Linear programming can be used to solve such problems. If a feasible solution exists, the configurations are scanned into the PDEs and the chip undergoes another timing verification step. The second timing verification is necessary because the delay steps of each PDE may not be constant and rounding the solutions of linear programming into discrete PDE configurations introduces errors. Branch-and-bound algorithms can be used to perturb the configurations and reduce the yield loss due to these errors.

VI. Conclusion and Future Work

We present a pre-silicon and post-silicon clock scheduling methodology that improves the timing yield through the use of false-path-aware statistical timing analysis and post-silicon clock tuning. Compared to a traditional clock scheduling algorithm, our methodology achieves an average of 13.6% yield improvement for benchmark circuits. We plan to integrate clock tree synthesis into our methodology. We would also like to extend our methodology to removing cross-talk delay faults and other types of timing faults through post-silicon clock tuning.

VII. Acknowledgement

This work was partially funded by National Science Foundation under grants CCR-0093309 & CCR-0204468 and National Science Council of Taiwan, R.O.C. under grant NSC 92-2218-E-002-030.

References

[1] Stephan Held, Bernhard Korte, Jens Maberg, Matthias Ringe, and J. Vygen. Clock scheduling and clocktree construction for high performance asics. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, pages 232–239, 2003.

[2] C. Albrecht, B. Korte, J. Schietke, and J. Vygen. Cycle time and slack optimization for VLSI-chips. In *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pages 232–238, 1999.

[3] John P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, 39(7):945–951, July 1990.

[4] Rahul B. Deokar and Sachin S. Sapatnekar. A graph-theoretic approach to clock skew optimization. In *Proceedings of the 1994 IEEE International Symposium on Circuits and Systems*, volume 1, pages 407–410, May 1995.

[5] José Luis Neves and Eby G. Friedman. Optimal clock skew scheduling tolerant to process variations. In *Proceedings of the 33rd annual conference on Design automation conference*, pages 623–628, 1996.

[6] Ivan S. Kourtev and Eby G. Friedman. Clock skew scheduling for improved reliability via quadratic programming. In *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pages 239–243, 1999.

[7] N. E. Young, Robert E. Tarjan, and J. B. Orlin. Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21:205–221, 1991.

[8] Utpal Desai, Simon Tam, Robert Kim, Ji Zhang, and Stefan Rusu. Itanium processor clock design. In *Proceedings of the 2000 international symposium on Physical design*, pages 94–98, 2000.

[9] Jason Stinson and Stefan Rusu. A 1.5GHz third generation Itanium-2 processor. In *Proceedings of the 40th conference on Design automation*, pages 706–709, 2003.

[10] R. Ginosar, Y. Elboim, and A. Kolodny. A clock tuning circuit for system-on-chip. In *Proceedings of the second ACiD-WG workshop of the european commission's fifth framework programme*, 2002.

[11] E. Takahashi, Y. Kasai, M. Murakawa, and T. Higuchi. A post-silicon clock timing adjustment using genetic algorithms. In *Digest of technical papers of the 2003 symposium on VLSI circuits*, pages 13–16, 2003.

[12] Kwang-Ting Cheng and H-C. Chen. Delay testing for non-robust untestable circuits. In *International Test Conference*, pages 954–961, Oct 1993.

[13] Karl Fuchs, Franz Fink, and Michael H. Schulz. DYNAMITE: An efficient automatic test pattern generation system for path delay faults. *IEEE Transactions on Computer-aided design*, 10(10):1323–1335, Oct 1991.

[14] Seiji Kajihara, Kozo Kinoshita, Irith Pomeranz, and Sudhakar M. Reddy. A method for identifying robust dependent and functionally unsensitizable paths. In *10th International Conference on VLSI Design*, pages 82–87, Jan 1996.

[15] Jing-Jia Liou, Angela Krstic, Li-C. Wang, and Kwang-Ting Cheng. False-path-aware statistical timing analysis and efficient path selection for delay testing and timing validation. In *Proceedings of the 39th conference on Design automation*, pages 566–569, 2002.

[16] K.-T. Cheng and H.-C. Chen. Delay testing for non-robust untestable circuits. In *Proceedings of the International test conference*, pages 954–961, Oct 1993.

[17] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 6(1):87–90, 1958.

[18] L. R. Ford and D. R. Fulkerson. Flows in networks. *Princeton University Press*, 1962.

[19] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, N.J., 1963.

[20] Daniel Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 296–305, 2001.

[21] Simon Tam, Stefan Rusu, Utpal Nagarji Desai, Robert Kim, Ji Zhang, and Ian Young. Clock generation and distribution for the first IA-64 microprocessor. *IEEE Journal of Solid-State Circuits*, 35(11):1545–1552, Nov 2000.

[22] David Harris and Sam Naffziger. Statistical clock skew modeling with data delay variations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(6):888–898, Dec 2001.

[23] C. J. Line, S. M. Reddy, and S. K. Sahni. On delay fault testing in logic circuits. *IEEE Transactions on Computer-aided design*, CAD-6(5):694–703, Jan 1987.

[24] K. Fuchs, M. Pabst, and T. Rossel. RESIST: A recursive test pattern generation algorithm for path delay faults considering various test classes. *IEEE Transactions on Computer-aided design*, 13(12):1550–1562, Dec 1994.

[25] S. Padmanaban and S. Tragoudas. An implicit path-delay diagnosis methodology. *IEEE Transactions on Computer-aided design*, 22(10):1399–1408, Oct 2003.

[26] P. Girard, C. Landrault, and S. Pravossoudovitch. An advanced diagnostic method for delay faults in combinational faulty circuits. *J. Electron. Testing*, 6(3):277–293, 1995.

[27] S. DasGupta, R. G. Walther, and T. W. Williams. An enhancement to LSSD and some applications of LSSD in reliability. In *Proceedings of the International test symposium*, pages 32–34, Jun 1981.