Yield-Driven, False-Path-Aware Clock Skew Scheduling

Jeng-Liang Tsai, Dong Hyun Baik, Charlie Chung-Ping Chen, and Kewal K. Saluja
University of Wisconsin-Madison

Editor's note:
This article proposes clock skew scheduling as a tool to address causes of performance-related circuit yield loss. It is an interesting example of how managing circuit-level parameters can have a direct impact on yield metrics and, therefore, a clear example of the direction of DFM research.
—Juan-Antonio Carballo, IBM Corporate Strategy, Venture Capital Group

SEMICONDUCTOR TECHNOLOGY ADVANCES have enabled designers to integrate more functionality in a single chip. As design complexity increases, many new design techniques are developed to optimize chip area and power consumption, as well as performance. Traditionally, yield improvement has been achieved through process improvement. However, in deep-submicron technologies, process variations are difficult to control. As a result, many design decisions significantly affect yield. Therefore, designers should consider yield-related issues during the design phase.

Timing failure is a major cause of yield loss for high-performance circuits. Although designers have used clock skew scheduling to increase operation frequency, there is little research addressing its impact on yield. We propose a novel clock-skew-scheduling scheme that improves yield without sacrificing performance. The scheme achieves this by combining accurate path delay information using our efficient sensitizable-critical-path search algorithm and a proportional slack distribution heuristic. Our experimental results show substantial yield improvement in many of the ISCAS89 and ITC99 benchmark circuits, and in one case improvement is as high as 50%.

Existing methods

In a zero-skew design, the circuit's longest path delay limits the shortest clock period. Properly assigning clock arrival times to each sequential element or introducing clock skew at various storage elements can help operate a circuit at a higher clock frequency.[1,2] Because timing failure is the major cause of yield loss, proper clock skew assignment for performance and yield is extremely important. Previous works attempted to reduce susceptibility to delay defects caused by process variations by finding a clock skew schedule that minimizes the number of paths with small slack.[3,4] However, as we show in this article, these approaches might not always improve yield, and the researchers did not verify their results with a yield model.

Clock period optimization

To increase operation frequency, designers have adopted techniques such as circuit retiming and pipelining to balance path delays in different parts of the circuit. Because path delays usually cannot be perfectly balanced, the application of clock skew scheduling can further optimize the clock period.[1-3] Figure 1a shows an example circuit with three flip-flops. The maximum and minimum path delays between FF_i and FF_j are D_{ij} and d_{ij}, respectively. CP is the clock period, and T_{setup} and T_{hold} are the FF setup and hold times. Without clock skew scheduling and with zero slack, this design will have $CP = \max\{D_{ij}\} + T_{setup}$. For this example, the value of CP is 4 where $T_{setup} = T_{hold} = 0$. However, to determine the optimal clock period with clock skew scheduling, we first define the clock arrival time to FF_i as T_i and the clock skew between FF_i and FF_j as $s_{ij} = T_i - T_j$. By absorbing the FF delays as part of the path delays, we can write the hold time and setup time constraints as

$$s_{ij} \geq T_{\text{hold}} - d_{ij}$$

and

$$s_{ij} \leq CP - D_{ij} - T_{\text{setup}}.$$

There are also two sets of constraints stemming from the clock skew definition $s_{ij} = T_i - T_j$. Path constraints require the sum of clock skews for paths having the same starting and ending flip-flop to be the same. Cycle constraints require the sum of clock skews of all cycles to be zero. Since all the constraints are linear, we can calculate the optimal clock period with linear programming solvers.[1]

Deokar and Sapatnekar use a graph-theoretic approach to solve the optimization problem.[2] They create a timing graph by replacing the hold time constraint of s_{ij} with an *h*-edge with cost $-(T_{\text{hold}} - d_{ij})$ from FF_i to FF_j, and by replacing the setup time constraint of s_{ij} with an *s*-edge with cost $(CP - D_{ij} - T_{\text{setup}})$ from FF_j to FF_i. Figure 1b shows the timing graph of the circuit in Figure 1a with zero T_{hold} and T_{setup}. Clock period *CP* is feasible if the timing graph contains no negative cost cycles. Therefore, we can obtain the optimal clock period through a binary search between $[0, \max\{D_{ij}\} + T_{\text{setup}}]$ by applying the Bellman-Ford algorithm to the timing graph. In Figure 1b, the optimal clock period is 3, and the three *s*-edges form a zero-cost cycle. This is substantially better than a zero-skew clocking scheme.

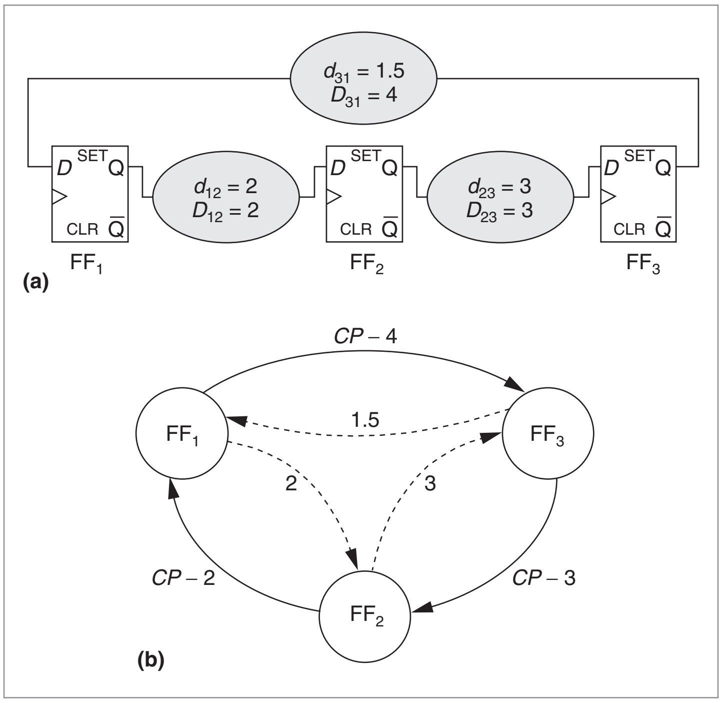

Figure 1. Example circuit (a) and its timing graph (b).

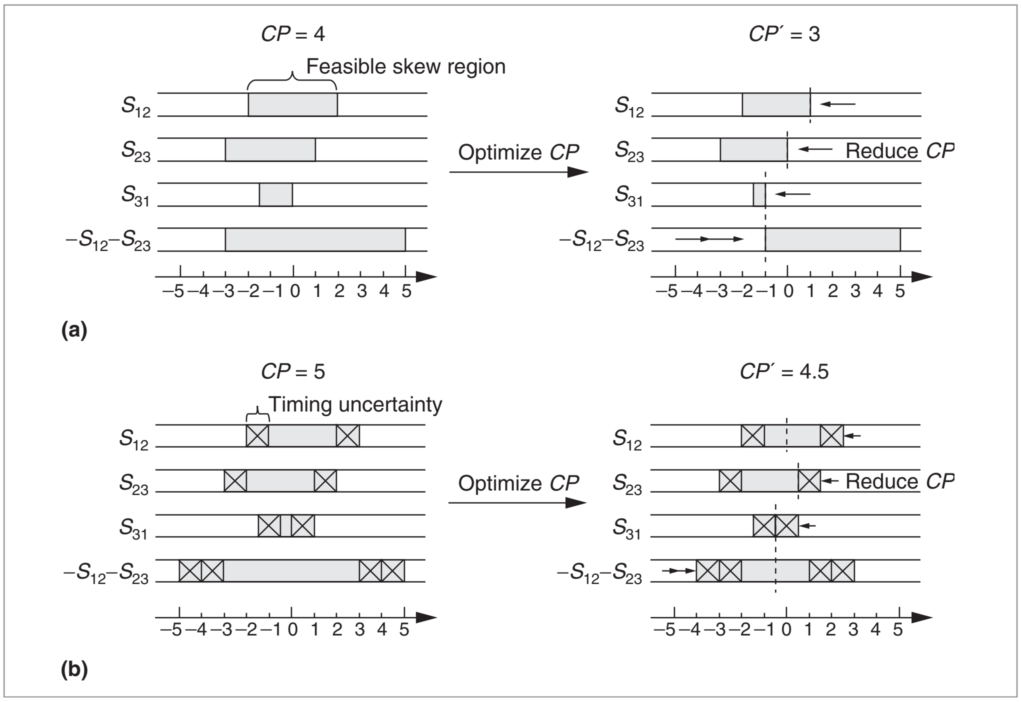

Figure 2. Clock period optimization: without timing uncertainty consideration (a) and with timing uncertainty consideration (b).

Clock skew optimization

The optimal clock period and the clock skew schedule just discussed will result in low-yield designs because many skew values are on the upper or lower bounds of their feasible range and have zero slack. If the feasible skew region (FSR) of s_{ij} is $[T_{\text{hold}} - d_{ij}, CP - D_{ij} - T_{\text{setup}}]$, the choice of clock period affects only the region's right boundary. Figure 2a shows the FSRs of Figure 1 for clock period $CP = 4$; the FSRs of s_{12}, s_{23}, and s_{31} are [–2, 2], [–3, 1], and [–1.5, 0]. When *CP* is reduced, the right boundaries of the FSRs of s_{12}, s_{23}, and s_{31} move to the left while the left boundary of $-s_{12}-s_{23}$ moves to the right. Cycle constraint $s_{31} = -s_{12}-s_{23}$ requires the FSR of s_{31} to overlap with the range of $-s_{12}-s_{23}$ for a feasible solution. Therefore, optimal clock period CP' is 3 (1 time unit less than CP), and clock skew schedule $(s_{12}, s_{23}, s_{31}) = (1, 0, -1)$, indicated by the dashed line, has zero slack.

Process variation introduces timing uncertainty to path delays. Assume that the maximum timing uncertainties of the minimum and maximum delays are both ±1 time unit. A simple way to guarantee that the clock skew schedule will have slack at least as large as the maximum timing uncertainty, is to preallocate timing margins of 1 time unit at both ends of the FSRs and then perform clock period optimization. In Figure 2b, the dashed line indicates one of the feasible clock skew schedules $(s_{12}, s_{23}, s_{31}) = (0, 0.5, -0.5)$, and optimal clock period CP' is 4.5. The slacks on (e_{12}, e_{23}, e_{31}) are (2, 1, 1). This approach increases the optimal clock period by an amount no less than the preallocated timing margin (1.5× in this example). Allocating a timing margin of maximum timing uncertainty is undesirable because the actual timing uncertainty cannot be accurately obtained, the maximum timing uncertainty is too pessimistic, and the ratio of clock period increase versus timing uncertainty can be large. In practice, product requirements usually determine the clock period, and we must find a clock skew schedule that maximizes yield.

A simple observation suggests that, to maximize slack, skew values should be chosen as close as possible to the middle points of their FSRs. Neves et al. formulate the skew optimization problem as a least-square error problem in which the error is defined as the difference between the skew and the middle point of the permissible range.[4] However, the formulation might reduce the slack of some skew values to zero to minimize the total error. Therefore, the resulting clock skew schedule might not be optimal in terms of yield. Albrecht et al. adopted a minimum-balance algorithm to maximize the slack of all skew values iteratively.[3] In each iteration, a parametric shortest-path algorithm solves a slack optimization problem that maximizes the minimum slack. The algorithm contracts the critical cycle that has the minimum average slack into a single vertex and repeats the process until all skew values are assigned. This algorithm ensures that the slacks along the timing-critical cycle are the same. However, it does not consider path delay differences between cycle edges, and its implementation is complex. Nevertheless, this slack-balancing clock-skew-scheduling algorithm is the most plausible method in the existing literature.

Yield-driven clock skew scheduling

In contrast to existing methods, our clock-skew-scheduling scheme takes path delays and false-path information into consideration to improve yield. First we solve the slack optimization problem using a minimum-mean-cycle algorithm. Then we use a new slack distribution method that distributes slack along the most timing-critical cycle proportional to path delays. Finally, we use an efficient sensitizable-critical-path search algorithm for clock skew scheduling.

Iterative slack optimization

To solve the clock skew optimization problem, we must

- identify the circuit's most timing-critical cycle,
- distribute the slack along the cycle,
- freeze the clock skews on the cycle, and
- repeat the process iteratively.

Each slack optimization problem is equivalent to a minimum-mean-cycle problem, and a well-known method for solving this problem is Karp's algorithm.[5] We adopted a faster minimum-mean-cycle algorithm to solve the clock skew optimization problem.[6]

After we determine the critical cycle's skews, the available slacks for the rest of the graph change accordingly. To determine the optimal slacks and skews for the rest of the graph, we replace the critical cycle with super vertex v'. In-edge e_{uv} from outside vertex u to cycle member v is replaced by in-edge $e_{uv'}$ with cost $w(e_{uv}) - T_v$. Out-edge e_{vu} is replaced by out-edge $e_{v'u}$ with cost $w(e_{vu}) + T_v$. Multiple edges from u to v' or from v' to u can exist after edge substitutions. Only the edge with the lowest cost in each direction must remain because edge cost is the upper bound of the corresponding skew variable. We can optimize the reduced graph again. The process continues until the graph is reduced to a single super vertex.

Figure 3 shows the clock skew optimization steps with $CP = 4.5$. It involves two minimum-mean-cycle runs, and the final clock skew schedule is $(s_{12}, s_{23}, s_{31}) = (0.75, -0.25, -0.5)$. The slacks on (e_{12}, e_{23}, e_{31}) are (1.75, 1.75, 1). Compared with Figure 1b, which has slacks of (2, 1, 1), this is a better schedule in terms of yield. Because this algorithm distributes slack evenly along the critical cycle, we refer to it as *Even*.

Slack distribution based on Gaussian model

The timing uncertainty of a long combinational path is usually larger than that of a shorter path. Therefore, the even slack distribution along timing-critical cycles performed by *Even* is not optimal for yield if path delays along the cycles are not the same. A gate's delay distribution usually matches a Gaussian distribution well. Observe that the sum of $n(\mu, \sigma)$ Gaussian distributions is an $(n\mu, n^{1/2}\sigma)$ Gaussian distribution. As a rule of thumb, we would prefer to distribute slack along the most timing-critical cycle according to the square root of each edge's path delays. We justify this heuristic on the ground that the longest paths require relatively longer slack for correct functionality and improved yield. To achieve this, we can update the weights of the *s*-edges with cost $CP - (D_{ij} + \alpha D_{ij}^{1/2}\sigma) - T_{\text{setup}}$ and *h*-edges with cost $-(T_{\text{hold}} - (d_{ij} - \alpha d_{ij}^{1/2}\sigma))$, where α ensures a minimum timing margin for each timing constraint.

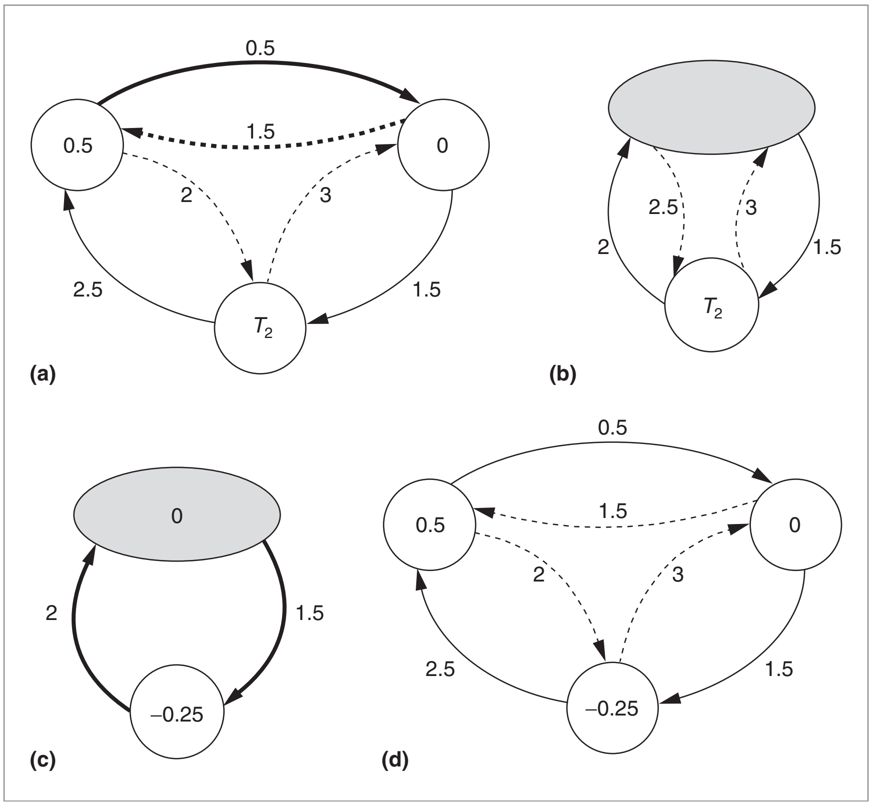

Figure 3. Clock skew optimization using a minimum-mean-cycle algorithm: detection of minimum mean cycle (a), cycle contraction (b), repetition of minimum-mean-cycle detection (c), and the final clock-skew schedule (d).

This is equivalent to preallocating timing margins of $\alpha D_{ij}^{1/2}\sigma$ and $\alpha d_{ij}^{1/2}\sigma$ to the right and left of the FSR for s_{ij}. We gradually increase α and use the Bellman-Ford algorithm to detect whether *CP* is still feasible. After we find the maximum value of α, the edges along the most timing-critical cycle will have slacks equal to the preallocated timing margins. Finding the maximum value of α requires multiple runs of the Bellman-Ford algorithm, and each run is as expensive as solving a slack optimization problem. However, many edges in a circuit have sufficiently large slack. Therefore, we can perform proportional slack distribution only for the most timing-critical cycle. We assign the rest of the skews by iteratively solving slack optimization problems as described earlier. We refer to the algorithm described here as *Prop*.

Finding sensitizable critical paths

We know that many paths in the circuit are functionally unsensitizable (false paths) and that they don't affect the circuit's timing behavior.[7-9] Thus, to obtain superior clock skew schedules, we must find the sensitizable (true) maximum and minimum path delay for all pairs of flip-flops. Although there are numerous studies on identifying false paths[9] or sensitizable long paths,[10,11] none attempts to find the maximum and minimum delays of sensitizable paths between all possible flip-flop pairs.

We developed a new algorithm to efficiently identify sensitizable longest and shortest paths between each pair of flip-flops. For conciseness, we describe the proposed method for finding the longest sensitizable path only; the same method can be applied to finding the shortest sensitizable path. We treat source flip-flops as primary inputs (*PI*s) and destination flip-flops as primary outputs (*PO*s) in the following discussion.

Dynamic delay path tree. First, we develop a data structure called a dynamic delay path tree (DPT) to efficiently identify functionally sensitizable paths. The DPT dynamically grows and prunes itself during the search process. The main purpose of using the DPT is to avoid explicit enumeration and sensitization of all structural paths that are candidates for the sensitizable longest path. Fuchs, Fink, and Schulz proposed a similar structure, called a path tree,[8] but its properties and purpose are different from those of the DPT.

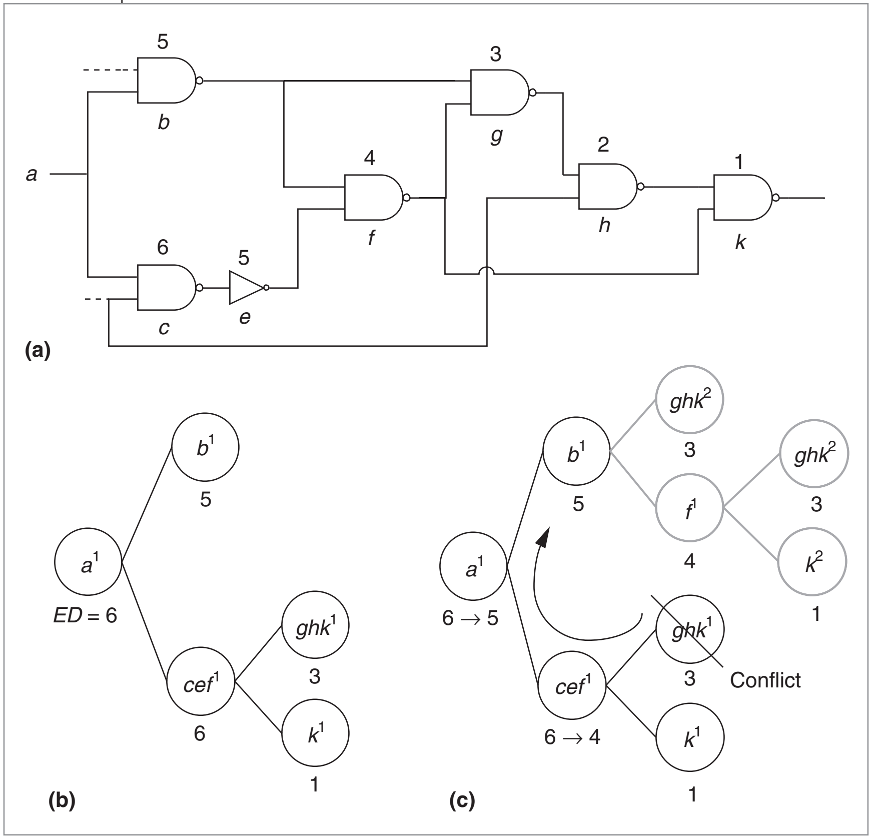

Figure 4. Dynamic delay path tree (DPT) example: circuit (a), DPT (b), and traversal redirection (c).

```
FindPath(DPT_NODE N)
(1) Check sensitizability of partial path N
        if success, longest sensitizable path is found
        if conflict is found, mark N as conflict and
backtrack
(2) Recursive advance in DPT
        Grow DPT by adding child node as needed
        FindPath( Child node with the maximum ED )
(3) Redirection of traversal
        Depending on result of recursion,
            Update EDs, change direction of traversal by
            backtracking and/or choosing other child
```

Figure 5. Pseudocode for finding the longest sensitizable path.

To describe DPT, we define a stem as a gate that has more than one fan-out branch, and a branch as a gate that has at least one input signal from a stem. A partial path is a path from a stem to a branch, and a subpath is a sequence of partial paths from *PI* to a certain partial path. Figure 4 shows an example circuit and DPT, which we use here to describe the DPT structure. Each DPT node is associated with a partial path and a subpath from *PI* to the corresponding partial path. A node's name represents the list of gates in the partial path. Superscripts uniquely identify a DPT node that ends with the same partial path but has different subpaths.

Before creating the DPT nodes and starting the search process, we calculate the expected delay (*ED*) for each gate. A gate's *ED* represents the maximum possible delay from that gate to *PO*. Whenever we create a new DPT node, we use the *ED* of the first gate in the corresponding partial path for the *ED* of that DPT node, and the *ED* guides the search process. When the process finds the longest true path, it modifies the *ED* value to the true path's delay. In Figure 4a, the initial *ED* value appears above each gate in the circuit.

Sensitizable-critical-path search algorithm. For a given *PI* and *PO* pair, the algorithm extracts the intersection of *PI*'s fan-out cone and *PO*'s fan-in cone from the circuit to define possible paths between the *PI*/*PO* pair. Then, the algorithm performs the search process twice for up and down transitions at *PI*. Figure 5 gives the algorithm's pseudocode.

For the selected *PI*, the algorithm first creates the root node and then initiates the recursion by calling the FindPath procedure on the root DPT node. FindPath contains three steps:

1. *Check sensitizability.* Step 1 of Figure 5 uses functional sensitization criteria[9] and local mandatory value assignments to test functional sensitizability of the partial path for DTP node *N*. If a conflict is found during sensitization, the algorithm marks *N* as a conflict, and the procedure backtracks to its parent DPT node. Once it completes the sensitization step, the algorithm checks whether the DPT is fully grown to *PO* to terminate the recursion process.
2. *Recursive advance in DPT.* With successful partial-path sensitization, the algorithm adds child nodes of the currently processed node to the DPT and chooses the child with maximum *ED* to be traversed recursively toward the targeted *PO*. Figure 4b shows that the sensitizability check is performed in the sequence of $a^1 \rightarrow cef^1 \rightarrow ghk^1$ first. Thus, the DPT

grows only toward the path that can potentially be extended to the longest sensitizable path.

3. *Traversal redirection.* Depending on the result of recursive FindPath on the child DPT node, the algorithm changes traversal direction. Assume that the node ghk^1 found by the process has conflict and cannot be sensitized. First, the memory allocated for ghk^1 will be freed because this node will not be rechecked for sensitizability. Consequently, the *ED*s of all its predecessors will be updated as in Figure 4c. Also, while updating the predecessors' *ED*s, the algorithm finds that the search process direction must be changed at a^1 because path *acefghk*, originally expected to have a delay of 6, is a false path. Then, the algorithm backtracks down to a^1 and chooses b^1 as the next child to traverse. In this backtracking procedure, the memory-allocated sensitization information for cef^1 is not discarded because cef^1 will need retraversal if node b^1 is marked as a conflict or if the *ED* of b^1 becomes less than 4. Whenever the algorithm revisits a node, it restores the sensitization information, thus avoiding a repeated computation.

As previously mentioned, the search process is guided only toward the potentially longest sensitizable paths without explicit enumeration. Also, if many subpaths are shared by several longest paths, we can avoid repeated sensitization efforts for shared subpaths by using the DPT structure. We can easily modify the algorithm to find the *K* longest or *K* shortest sensitizable paths by traversing the DPT until it finds *K* sensitizable paths.

Using false-path information to improve yield

To reduce chip cost, different combinational paths usually share logic gates and their intermediate outputs when possible. However, this sharing creates many false paths that are unsensitizable. Clearly, some of the structurally longest and shortest paths can be false paths. As a result, path delay information without knowledge of false paths might be too pessimistic. By combining path delay information from our sensitizable-critical-path search algorithm with *Prop*, we can find a clock skew schedule likely to improve yield even further. Our experimental results substantiate this. We call the combined algorithm for false-path-aware proportional slack distribution *fp-Prop*.

Earlier we demonstrated a method for slack distribution based on the approximated path delay uncertainties $D_{ij}^{1/2}\sigma$ and $d_{ij}^{1/2}\sigma$. The simple approximation on path delay uncertainties should work well if the circuit doesn't have reconvergent paths and each gate delay is an independent random variable. However, delays of two gates are usually correlated if the distance between them in the layout is small. To better account for reconvergent paths and correlations between delay variables, we can use path-based statistical static timing analysis (SSTA) to obtain more accurate path delay distributions. We can extend *Prop* and *fp-Prop* to account for path delay uncertainties from the distributions and replace $D_{ij}^{1/2}\sigma$ and $d_{ij}^{1/2}\sigma$ with these new values to perform slack distribution.

Path-based SSTA requires a list of paths as input. Because there can be an exponential number of paths in a circuit, we must select a subset of paths for analysis. Our sensitizable-critical-path search algorithm can efficiently select the *K* longest or *K* shortest sensitizable paths for path-based SSTA.

Experimental results

Now we introduce a hybrid yield model that uses statistical timing information to speed up Monte Carlo simulation. We obtain the false-path information for benchmark circuits and compare the yields of our three clock-skew-scheduling schemes: slack-balancing scheduling (*Even*), proportional slack distribution scheduling (*Prop*), and false-path-aware proportional slack distribution scheduling (*fp-Prop*).

Statistical-timing-enhanced yield model

To estimate a clock skew schedule's yield, we must generate circuit samples according to the gate delay distributions and check the setup-time and hold-time constraints for all edges in the timing graph. Although the error of Monte Carlo simulation is proportional to $1/N^{1/2}$, where N is the number of samples, and independent of the sample space's dimensions, the time needed to verify a circuit increases significantly as the circuit size grows. Unlike the Gaussian distribution, which extends to $\pm\infty$, gate delay usually varies no more than a few standard deviations from its mean value. Using extend factor k such that $k\sigma$ is a gate's maximum timing uncertainty, we can truncate the Gaussian distribution outside $\mu \pm k\sigma$ and renormalize the distribution to describe a gate's delay distribution. We call this the truncated Gaussian distribution.

Agarwal et al. proposed a statistical timing analysis algorithm that generates the upper and lower bounds of the maximum and minimum path delay distributions.[12] The researchers proved that for a two-input gate with the inputs' probability density functions being f and g and the cumulative density functions being F and G, a

Table 1. Sensitizable critical paths of benchmark circuits.

Circuit	No. of FFs	No. of gates	Str. pairs	Diff.-long	Diff.-short
s1423	74	657	2,235	8	0
s1488	6	653	266	0	0
s1494	6	647	266	0	0
s5378	179	2,779	2,313	0	0
s9234.1	211	5,597	3,260	0	0
s13207.1	638	7,951	4,721	0	0
s35932	1,728	16,065	7,595	1,600	0
s38417	1,636	22,179	34,351	852	72
s38584.1	1,426	19,253	20,444	0	0
b04s	66	512	844	0	0
b05s	34	864	700	93	6
b06	9	43	44	0	0
b07s	49	362	1,155	24	0
b08	21	133	151	2	0
b09	28	129	303	0	0
b10	17	155	165	0	0
b11s	31	437	631	65	23
b12	121	904	1,630	3	0

pessimistic estimate of the latest input arrival time's distribution is $fG + gF$. Likewise, $f(1 - G) + g(1 - F)$ is an optimistic estimate of the earliest input arrival time's distribution. Using this algorithm, we can obtain the bounds of D_{ij} and d_{ij}, $\overline{D_{ij}}$ and $\underline{d_{ij}}$, from the distributions. After clock skew scheduling, we can check whether s_{ij} is between $[T_{\text{hold}} - \underline{d_{ij}}, CP - \overline{D_{ij}} - T_{\text{setup}}]$ for each edge of the timing graph. If the condition is satisfied, e_{ij} is safe in timing under process variation, and we need not calculate D_{ij} and d_{ij} for every circuit sample and verify their timing constraints. Most skews generally satisfy this condition. Therefore, we need only perform Monte Carlo simulation for the combinational paths between a few flip-flop pairs, and the simulation time is significantly reduced (by a factor of 10 ~ 1,000), depending on the circuit size.

Sensitizable critical paths

Here we present the sensitizable longest- and shortest-path search result from the proposed method for ISCAS89 and ITC99 benchmark circuits. Table 1 presents the benchmark circuit statistics. Column "Str. pairs" lists the total numbers of (*PI*, *PO*) pairs that have combinational paths between them. The column "Diff-long" lists the numbers of structural pairs whose sensitizable longest paths are shorter than structural longest paths. The column "Diff-short" lists the numbers of pairs whose sensitizable shortest paths are longer than structural shortest paths.

For many of the ISCAS89 and ITC99 benchmark circuits, we observe that a substantial number of structural pairs have shorter or longer sensitizable path delays. In circuit b11s especially, more than 10% of all long paths have shorter sensitizable path delays and nearly 4% of short paths have longer sensitizable path delays.

Yield computation and relative improvements

We assume that gate delays are independent truncated Gaussian distributions with $(\mu, \sigma) = (1, 0.15)$ and extend factor $k = 3$. For each benchmark circuit, we first obtain the optimal clock period using the algorithm described earlier. In practice, an initial design's timing yield is usually not very high. To make reasonable comparisons, we choose *CP* such that the yield for the *Even* algorithm is between 60% ~ 80%. Table 2 shows the yields of the three clock-skew-scheduling schemes for ISCAS89 and ITC99 benchmark circuits. The "Imp." columns show the yield improvement from implementing *Prop* and *fp-Prop*. The circuits that have sensitizable critical paths as long (short) as the structural longest (shortest) paths do not benefit from false-path-aware scheduling, and the table does not include their yield entries for *fp-Prop*.

Comparing *Even* with *Prop*, we observe that half the circuits have yield differences within ±5%. This is because the path delays along the critical cycle are similar, resulting in similar slack distributions. Thus, we can assume that the three methods are equally effective for all these circuits. However, the yield improvements for the rest of the circuits are significant, ranging from 12% to 53%. Of the eight circuits that have sufficiently many sensitizable critical paths shorter (longer) than the structurally longest (shortest) paths, two (s38417 and b11s) show additional improvement from *Prop* to *fp-Prop*. This result is not surprising because the edges with path delay differences might not contribute additional slacks on the critical cycle.

For b11s, the yield of *Prop* is lower than that of *Even*. Because *Prop* proportionally distributes slack to the critical edges regardless of false-path information, it can allocate more slack for a critical edge with shorter nonfalse path delay. By taking false-path information into consideration, *fp-Prop* achieved 13% improvement over *Even*. Figure 6 shows the three scheduling schemes' yield curves. The yield of *fp-Prop* is always better than that of the other two schemes. Moreover, the improvement rate gradually decreases as the clock period increases. The yield improvement of *fp-Prop* over *Even* is still 6.5% when *CP* is 31.

By proportionally distributing slack along a circuit's critical cycles and using information about the sensitizable longest and shortest path delays, we achieved substantial yield improvement in most of the benchmark circuits. For ISCAS89 circuit s35932, our method achieves a yield of 98.6%, whereas the traditional scheme achieves only 64.3% yield for the same clock period. For ITC99 circuit b11s, we have shown that false-path information is essential for yield-driven clock skew scheduling; without false-path information, traditional as well as proportional slack distribution will provide far lower yield than is possible.

Table 2. Yield comparison of ISCAS89 and ITC99 circuits.

		Even	***Prop***		***fp-Prop***	
Circuit	***CP***	**Yield (%)**	**Yield (%)**	**Imp. (%)**	**Yield (%)**	**Imp. (%)**
s1423	55.73	74.1	75.5	1.9	75.5	1.9
s1488	16.62	72.3	75.4	4.3	NA	NA
s1494	16.62	77.9	78.8	1.2	NA	NA
s5378	22.50	63.8	64.2	0.6	NA	NA
s9234.1	40.86	74.1	83.9	13.2	NA	NA
s13207.1	52.73	60.2	60.2	0.0	NA	NA
s35932	31.96	64.3	98.6	53.3	98.6	53.3
s38417	34.07	74.1	74.7	0.8	89.1	20.2
s38584.1	50.24	72.5	85.8	18.3	NA	NA
b04s	23.88	67.4	82.6	22.6	NA	NA
b05s	47.68	67.2	86.9	29.3	86.9	29.3
b06	4.92	63.1	75.1	19.0	NA	NA
b07s	26.23	74.8	72.7	–2.8	72.7	–2.8
b08	14.90	68.9	65.6	–4.8	65.6	–4.8
b09	7.68	62.4	78.3	25.5	NA	NA
b10	9.71	73.5	73.7	0.3	NA	NA
b11s	29.86	71.5	58.6	–18.0	80.9	13.1
b12	12.12	78.2	87.6	12.0	87.6	12.0

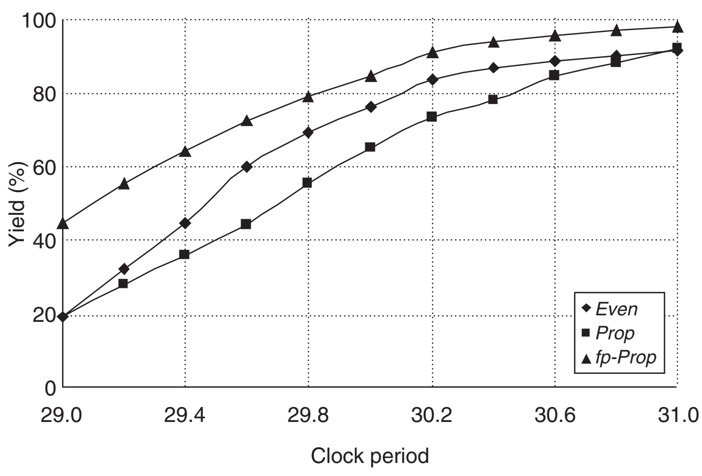

Figure 6. Yield curves of the three clock-skew-scheduling schemes for circuit b11s.

OUR STUDY DEMONSTRATES the advantages of slack distribution methods that account for path length and are aware of false paths in a circuit. However, this approach is static in the sense that it is fixed before the design is implemented in silicon. It is possible to make postsilicon adjustments to the clock skew for a small subset of storage elements to recover some yield loss. Such methods are now under investigation. ■

Acknowledgments

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

■ References

1. J.P. Fishburn, "Clock Skew Optimization," *IEEE Trans. Computers*, vol. 39, no. 7, July 1990, pp. 945-951.
2. R.B. Deokar and S.S. Sapatnekar, "A Graph-Theoretic Approach to Clock Skew Optimization," *Proc. 1994 IEEE Int'l Symp. Circuits and Systems* (ISCAS 94), vol. 1, IEEE Press, 1994, pp. 407-410.
3. C. Albrecht et al., "Cycle Time and Slack Optimization for

VLSI-Chips," *Proc. 1999 IEEE/ACM Int'l Conf. Computer-Aided Design* (ICCAD 99), IEEE Press, 1999, pp. 232-238.

4. J.L. Neves and E.G. Friedman, "Optimal Clock Skew Scheduling Tolerant to Process Variations," *Proc. 33rd Design Automation Conf.* (DAC 96), IEEE CS Press, 1996, pp. 623-628.
5. R.M. Karp, "A Characterization of the Minimum Cycle Mean in a Digraph," *Discrete Mathematics*, vol. 23, no. 3, 1978, pp. 309-311.
6. A. Dasdan and R.K. Gupta, "Faster Maximum and Minimum Mean Cycle Algorithms for System-Performance Analysis," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, Oct. 1998, pp. 889-899.
7. K.-T. Cheng and H.-C. Chen, "Delay Testing for Non-Robust Untestable Circuits," *Proc. Int'l Test Conf.* (ITC 93), IEEE Press, 1993, pp. 954-961.
8. K. Fuchs, F. Fink, and M.H. Schulz, "Dynamite: An Efficient Automatic Test Pattern Generation System for Path Delay Faults," *IEEE Trans. Computer-Aided Design*, vol. 10, no. 10, Oct. 1991, pp. 1323-1335.
9. S. Kajihara et al., "A Method for Identifying Robust Dependent and Functionally Unsensitizable Paths," *Proc. 10th Int'l Conf. VLSI Design*, IEEE Press, 1997, pp. 82-87.
10. W. Qiu and D.M.H. Walker, "An Efficient Algorithm for Finding the *k* Longest Testable Paths through Each Gate in a Combinational Circuit," *Proc. Int'l Test Conf.* (ITC 03), IEEE Press, 2003, pp. 592-601.
11. A. Murakami et al., "Selection of Potentially Testable Path Delay Faults for Test Generation," *Proc. Int'l Test Conf.* (ITC 2000), IEEE Press, 2000, pp. 376-384.
12. A. Agarwal et al., "Statistical Timing Analysis Using Bounds and Selective Enumeration," *Proc. 8th ACM/IEEE Int'l Workshop Timing Issues in the Specification and Synthesis of Digital Systems*, ACM Press, 2002, pp. 29-36.

Jeng-Liang Tsai is a PhD candidate in the Department of Electrical and Computer Engineering, University of Wisconsin-Madison. His research interests include VLSI physical design and optimization with an emphasis on clock design techniques for timing convergence and timing yield improvements. Tsai has a BS and an MS in electrical engineering from National Taiwan University.

Dong Hyun Baik is a PhD candidate in the Department of Electrical and Computer Engineering, University of Wisconsin-Madison. His research interests include design for testability and automatic test pattern generation. Baik has a BE in computer science and engineering from Hanyang University, Seoul, Korea, and an MS in electrical engineering from the University of Wisconsin-Madison.

Charlie Chung-Ping Chen is an assistant professor in the Department of Electrical and Computer Engineering, University of Wisconsin-Madison. He is also an associate professor in the Department of Electrical Engineering, National Taiwan University. His research interests include CAD and microprocessor circuit design with an emphasis on interconnects and circuit optimization, circuit simulation, statistical design, and signal/power/thermal integrity analysis and optimization. Chen has a BS in computer science and information engineering from National Chiao-Tung University, Hsinchu, Taiwan, and MS and PhD degrees in computer science from the University of Texas at Austin.

Kewal K. Saluja is a professor in the Department of Electrical and Computer Engineering, University of Wisconsin-Madison. He teaches courses in logic design, computer architecture, microprocessor-based systems, VLSI design and testing, and fault-tolerant computing. Saluja has a BE in electrical engineering from the University of Roorkee, India, and an MS and PhD in electrical and computer engineering from the University of Iowa, Iowa City. He is a member of Eta Kappa Nu and Tau Beta Pi, a JSPS Fellow, and an IEEE Fellow.

■ Direct questions and comments about this article to Jeng-Liang Tsai, University of Wisconsin-Madison, Dept. of Electrical and Computer Engineering, 1415 Engineering Dr., Madison, WI 53706; jltsai@cae.wisc.edu.