Diagnosis of Optical Lithography Faults with Product Test Sets

Munkang Choi* and Linda Milor
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332

Abstract— **As semiconductor technology advances into the nanoscale era and more functional blocks are added into systems on chip (SoC), within-die variation is increasing, resulting in path delay faults. A component of within-die variation comes from optical lithography, such as the optical proximity effect, lens aberrations, and flare. This paper presents a methodology to generate test sets to diagnose the sources of within-die variation. Specifically, a delay fault diagnosis algorithm is developed to link failing signatures to physical mechanisms and to distinguish among different sources of within-die variation. The algorithm relies on layout-dependent timing analysis, path enumeration, test pattern generation, and correlation of pass/fail signatures to diagnose lithography-caused delay faults. The effectiveness in diagnosis is evaluated for ISCAS85 benchmark circuits, and implementation issues are discussed.**

I. Introduction

Circuits manufactured with the same fabrication process flow exhibit variation in performance due to non-uniformity in process conditions. This variation is quantified by specifying a distribution of process parameters, such as transistor channel length, width, oxide thickness, etc. This distribution can be broken down into lot-to-lot, wafer-to-wafer, field-to-field, and within-field variability. Circuit designers account for such variability through worst case analysis [1]. However, within-field variability, which characterizes parameter variation within a die (chip), is not accounted for by worst case analysis, and consequently, circuits are vulnerable to yield loss due to such variability (within-die variation).

Unfortunately, within-die variation is increasing for advanced technologies. Consequently, it is important for manufacturers to diagnose the causes of within-die variation. The purpose of this paper is to provide such a methodology.

A. Physical Causes of Within-Die Variation

An important source of within-die variation is variation from lithography. Such variation impacts circuit delay.

The physical origin of within-die variation from lithography includes the optical proximity effect [2], lens aberrations [3], and flare [4]. Each of these effects result in distinct fault signatures and lead to distinct correction methods. For example, local mask correction can be used to counter the proximity effect [2]. Spatial mask correction can counter lens aberrations [5]. And, dummy feature insertion can create uniform mask density and eliminate the effect of flare [4]. Since all of these techniques increase mask cost, it is important to optimize the use of correction based on product requirements. Therefore, we must diagnose the physical origin of failures and choose the proper correction strategy.

B. Existing Approaches to Detecting Within-Die Variation

Process variations are parametric faults, which are diagnosed with correlation analysis. Specifically, the correlation is determined between the yield for a collection of wafers and the average measurement for all test structures in the scribe line. If the correlation is high for a specific test structure, the parameter associated with the test structure is the likely cause of yield variation. For example, it may be determined that low yielding wafers are associated with high resistance vias.

If variation exhibits patterns within a wafer, correlations are performed between yield in specific wafer sectors and test structure measurements to identify the cause of variation.

The specific faults considered in this paper are not easily diagnosed with correlation analysis, because faults caused by lithography cause variations within a reticle, rather than within a wafer. Typically, only a single copy of each test structure is included in each reticle. Therefore, the scribe line does not have sufficient granularity to detect such variation. Moreover, even if the scribe line were populated with sufficient test structures covering variation within a reticle, the range of sources of variation from lithography is sufficiently large that multiple copies of many test structures in many positions would be required, which is not practical. Therefore, this paper attempts to determine if we can diagnose such faults via product tests.

* Munkang Choi is currently with Synopsys, CA.

C. Path Delay Faults

Within-die variation from lithography causes a circuit's speed of operation to be reduced. Such variation can be thought of as a distributed manufacturing defect that cumulatively increases delays within circuit paths. Therefore, the path delay fault model [6] addresses this failure mode.

Path delay faults are sets of paths that can be sensitized by a transition or sets of transitions at the primary inputs. It has been shown that only a subset of such paths need to be tested to guarantee temporal correctness of a circuit [7], and this set is independent of component delays. The set of paths that need to be tested to guarantee temporal correctness is called the set of primitive delay faults. Several approaches have been developed to identify single and multiple primitive delay faults [8]-[10] and to generate the appropriate test sets for these faults. However, these methods are applicable to moderately sized circuits, as indicated in [8]. Sharma *et al.* [11] propose to overcome this problem by covering delay faults on robustly untestable critical paths by robustly testing their longest possible segments that are not covered by any of the testable critical paths.

Many of the paths associated with primitive delay faults have delays that are much less than the clock period under normal operation. Moreover, paths associated with primitive delay faults may not be the longest paths in a circuit. Delay faults on such paths can only be detected during normal operation if the delay fault is large. Consequently, in order to detect smaller delay faults, it is desirable to find a set of longest paths. Several papers have been published relating to the selection of the longest critical paths [11]-[16]. Most papers have focused on selecting paths to ensure topological coverage. Specifically, in [12],[13], paths are selected to cover each gate in the circuit, i.e. the set of longest paths through each gate are found. As a result, small delay defects associated with a gate can be detected. To ensure testability, these methods check path sensitization, but do not consider whether their tests are associated with primitive faults. Moreover, in [11]-[15], the calculated delays of the critical paths are based on discrete-valued timing models that don't take into account the signal propagation effect (signal transition slope dependency). Wang *et al.* [16] introduced the concept of path correlation in critical path selection using a statistical timing model. The statistical timing framework has the potential to properly deal with coupling noise, temperature gradients, power supply gradients, and across-chip linewidth variation [17], but determining the characterization (probability distribution functions and their correlations) of the underlying transistors and wires is a major unsolved challenge [17],[18].

D. Delay Fault Diagnosis

Diagnosis is the process of identifying the cause of failure. There are a variety of causes for delay faults. They include local failure mechanisms, such as resistive shorts and opens, and other mechanisms, such as crosstalk-induced delay, delay due to power supply noise, and delay due to process variations. Traditionally, delay fault diagnosis has focused on local failure mechanisms. Specifically, the goal of diagnosis is the localization of physical defects in failing circuits, in order to identify the root cause. Localization involves analyzing input vectors and output responses to determine the defect location.

Methods for localization can be classified as cause-effect and effect-cause analysis [19]. Cause-effect analysis pre-computes faulty behavior based on the assumed fault model and stores the information in a fault dictionary. The behavior of a failing chip is compared with the fault dictionary to identify the most probably faults. Effect-cause analysis involves searching backwards from the failing outputs, deducing internal values, to identify locations of probable faults.

Underlying the diagnosis process is the fault model. A variety of fault models exist that differ from each other based on fault complexity (stuck-at vs. resistive opens or shorts), temporality (static or dynamic), and cardinality (single or multiple). Many papers have addressed diagnosis for a variety of fault models, including stuck-at faults, bridging faults, and even Byzantine defects [20], where defects result in intermediate voltage levels at a gate output and the corresponding fanout branches are associated with different logic values due to the different logic thresholds of subsequent gates.

D.1. Gate Delay Fault Diagnosis

Resistive shorts and opens, crosstalk, and power supply noise are among the failure mechanisms that may not cause stuck-at failures, but rather may cause single or multiple transistor delay faults. Several papers [21]-[23] have presented diagnostic methodologies to isolate delay faults associated with gates (gate delay faults and transition delay faults). Girard *et al.* [21] proposed a method to diagnose gate delay faults based on critical path tracing. The method involves logic simulation only, together with tracking of transitions for sets of patterns. If a transition results in a failing output, the gates in the paths sensitized by the transition are stored as potential sites for gate delay faults. The method accounts for potential glitches through the use of six-valued logic simulation. Wang *et al.* [22] improves the resolution of transition delay fault diagnosis through pruning impossible fault candidates using circuit timing information. Krstic *et al.* [23] goes beyond this approach by linking diagnosis to statistical timing analysis. Specifically, in [23] the delay fault potentially associated with each gate is assumed to be probabilistic, together with the rise and fall times of the gates. Statistical timing is combined with the probabilistic fault model to construct a fault dictionary to provide probabilities of failure for all input transitions and faults. However, the delay distribution of each circuit element is assumed to be known, together with correlations among elements, and statistical characterization information for all instances is not easily available [17],[18].

D.2. Path Delay Fault Diagnosis

Gate delay fault diagnosis focuses on localizing the cause of a delay fault. However, some failures may be due to small delay variations in a number of gates that accumulate to produce a delay fault. Path delay fault diagnosis addresses the isolation of such faults. Specifically, path delay fault diagnosis involves locating input-output paths in a chip that cause the delay fault. Several methods have been proposed to address this problem [24]-[27].

Diagnosis procedures generally start with a complete fault list, which is pruned by analyzing the applied tests and responses. In the case of path delay fault diagnosis, the set of potential faults is all sensitizable paths in a circuit. Therefore, the initial set of faults is exponentially large. When random tests are applied to a circuit, such tests sensitize a number of single and multiple path delay faults. Pant *et al.* [24] address this problem using an effect-cause approach where, first, the set of paths sensitized by each failing vector is determined, and, second, those paths that have been robustly tested by other passing vectors (guaranteed to be delay fault free) are removed from consideration. The result is the suspect set.

Padmanaban *et al.* [25] improved this approach by further pruning the suspect set by eliminating paths (single and multiple path delay faults) that pass validatable non-robust tests.

To further guide diagnosis Sivaraman *et al.* [26] and Krstic *et al.* [27] introduce a statistical framework. To aid in diagnosis, Sivaraman *et al.* [26] limits test patterns to those that provide single multipath robust tests. And, since each of the test vector pairs has incompletely specified inputs, the unspecified inputs are set to minimize the number of primary inputs that have transitions so that the number of side paths that get sensitized are minimized. Then, given a set of failed tests, the sets of sensitized paths are determined. For each sensitizable path, a model of process parameter variations is used together with Monte Carlo analysis to find statistical distributions of slack for each path and to weight potential sites for delay faults. Therefore, sites for likely faults are selected if the corresponding path is sensitized by a test which violates a timing constraint, and sites are more probable fault sites if tighter timing constraints are placed on the paths through them. In this way, the method in [26] provides better feedback about the location of faults than [24],[25].

Like [26], Krstic *et al.* [27] propose a similar path delay fault diagnostic framework, involving three steps. First, effect-cause analysis identifies a suspect set through logic analysis of failing patterns. Second, cause-effect analysis reduces the suspect set through statistical timing simulation in the presence of various error sources (modeling errors, single-site random size timing errors, etc.). And third, the failure mechanism is linked to potential error sources by comparing simulation results from a collection of circuit instances to the fault dictionary and voting among the faults.

The problem with these approaches is that they just pinpoint a sub-path responsible for circuit failure, and not the underlying physical cause. Hence, these algorithms must be followed with physical analysis in order to provide useful information. What is needed is physical evidence of the cause of failure so that appropriate action can be taken.

D.3. Linking Diagnostic Results to Physical Failure Mechanisms

In order to lead to corrective actions, diagnostic procedures must go beyond identifying the failing path to determining the physical mechanism causing the failure. To this end, several papers have proposed test pattern generation to detect crosstalk-induced delay [28]-[30], power supply noise [30],[31], and resistive open and short defects [30]. In Chen *et al.* [28], it is demonstrated that crosstalk can lead to delay faults, and test patterns are generated for a set of user-supplied single crosstalk-induced delay faults. Krstic *et al.* [29] extends this work by adding methods to select crosstalk faults based on performance sensitivity analysis. Moreover, once the paths have been selected, a genetic algorithm is used to find the test patterns.

Similarly, Krstic *et al.* [31] identifies path delay faults associated with power supply noise through performance sensitivity analysis, with a statistical dynamic timing analysis framework. Patterns are found that sensitize the faults using a genetic algorithm, which assigns unspecified primary inputs such that the power supply noise impact on the delays of signals is maximized.

Finally, in addition to crosstalk-induced and power supply noise-induced delay faults, Liou *et al.* [30] considers interconnect delays coming from resistive open and short defects. Again, a similar methodology is used to identify faults and to select test patterns to detect the faults.

This paper is similar to these papers in that it aims to design test patterns for specific failure mechanisms which can be used to activate specific sources of delay faults. It is different because the focus is not on detecting design issues (crosstalk, power supply noise), but instead targets detection of process problems (within-die variation from lithography). Like crosstalk and power supply noise, within-die variation is not simulated during conventional design, and therefore designs are vulnerable to yield loss as a result. Moreover, process monitors in the scribe lines cannot be used for diagnosis. Hence, all of these failure mechanisms are difficult to diagnose.

This paper differs from the above papers in that the focus is not just to detect the cause of failure but also to use the generated test patterns to provide diagnostic information of the physical causes of failure to the chip manufacturer. Hence, the set of failures is directly linked to corrective actions.

E. A Summary of This Paper

This paper proposes a method to diagnose the physical origins of faults from lithography. The methodology includes the design of test sets to detect each of the failure mechanisms. These test sets are used to construct a dictionary of pass/fail patterns associated with each fault.

To design such test sets, it is first necessary to enumerate critical paths. We have implemented a tool that provides efficient path enumeration while considering the transition slope (slew) dependency. The output of the tool is the most significant critical paths of the fault-free circuits, with delays longer than a fixed threshold. Test patterns are then generated for these critical paths using a commercial tool.

Next, fault simulation is performed for each delay fault caused by imperfect lithography, via dynamic timing analysis and using the extracted test patterns. To activate a fault, it is necessary to determine how various sources of within-die variation affect circuit performance. Thus we have implemented a tool to involve layout-dependent within-die variation in dynamic timing analysis. The tool inputs layout-dependent information relating to the physical neighborhood, location, and mask density in the vicinity of each transistor and interconnect segment. It updates transistor and interconnect geometries based on the physical neighborhood, feature location, and mask density. The revised delays for each test pattern and each fault are determined. These delays are compared against a fixed delay, which is a function of the test frequency to generate pass/fail patterns for each fault. The results are for all test patterns and all faults are stored in a dictionary. Diagnosis involves correlating an observed pass/fail pattern with those in the dictionary.

F. Organization of the Paper

This paper is organized as follows. In Section II, we introduce the lithography delay fault model, and in Section III, we present a layout-dependent timing analysis flow based on this fault model. In Section IV, we discuss the diagnosis methodology to determine the cause of lithography faults. In Section V, we present experimental results relating to diagnosis, in Section VI, we discuss implementation of the methodology, and in Section VII, we summarize our results.

II. Modeling Optical Lithography-caused Delay Faults

Three types of delay faults are caused by imperfect lithography: the proximity effect [2], Coma [3], and lens aberrations [3]. They are summarized in [32] in detail. In this section, we present the modeling of their layout dependencies. We focus on the gate layer since circuit speed is most sensitive to this layer [32].

A. The Proximity Effect

The proximity effect causes linewidths in dense areas to be different than linewidths in isolated areas, as well as line-end shortening, and corner rounding. The proximity effect is caused by variations in light intensity during exposure of the photoresist, resulting from the presence of neighboring features. This intensity variation modifies the exposure of photoresist on gate edges, which in turn translates into systematic variation in gate CDs. Thus, the gate critical dimension (CD) is a function of its neighborhood. We account for the neighborhood by determining the distance to the nearest poly geometry on the left and on the right of each transistor gate. Each transistor, therefore, has two labels, the distance to the nearest poly geometry on the left and the distance to the nearest poly geometry on the right, assuming a vertical orientation. Labels for horizontal transistors correspond to distances to the nearest poly geometry above and below the feature. These two labels combine to determine the category of each gate.

The distances to the left and to the right are labeled as n1 to n5, where n1 is the minimum poly spacing, and n5 is the largest distance. These distance categories have been chosen arbitrarily, but they conform to common distances seen in a layout, i.e. minimum poly spacing, minimum poly spacing if the space contains a contact, etc. The distance categories are illustrated in Figure 1. Table I illustrates the case where dense patterns (n1n1) are 10% larger than isolated patterns. As can be seen from the table, intermediate patterns are interpolated.

B. Coma

Coma is a lens aberration that depends on both the neighborhood and location. Coma becomes severe when making use of resolution enhancement techniques, such as phase shift masks (PSM) and off-axis illumination (OAI). Analyzing Coma requires that we distinguish between features to the left and features to the right of a specific pattern, since patterns with asymmetric categories are printed on the wafer differently. We do this by prescribing an order to the two labels that define the category of each gate. For example, Coma may cause transistors (n5n1) with dense features on the right and isolated features on the left to have larger CDs than transistors (n1n5) with dense features on the left and isolated features on the right. Table II illustrates this case for a 10% range of variation.

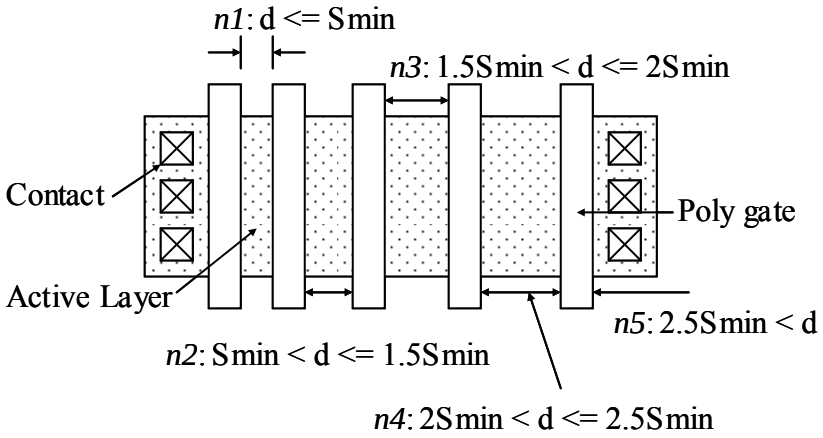


Fig. 1. Distance categories for poly gates. Smin is the minimum space design rule between two poly lines with no contacts in between them.

TABLE I
Proximity Effect ([%])

Category	**n1**	**n2**	**n3**	**n4**	**n5**
n1	10.00	8.75	7.50	6.25	5.00
n2	8.75	7.50	6.25	5.00	3.75
n3	7.50	6.25	5.00	3.75	2.50
n4	6.25	5.00	3.75	2.50	1.25
n5	5.00	3.75	2.50	1.25	0.00

TABLE II
Coma ([%])

Left \ Right	**n1**	**n2**	**n3**	**n4**	**n5**
n1	5.00	3.75	2.50	1.25	0.00
n2	6.25	5.00	3.75	2.50	1.25
n3	7.50	6.25	5.00	3.75	2.50
n4	8.75	7.50	6.25	5.00	3.75
n5	10.00	8.75	7.50	6.25	5.00

C. Lens Aberrations

Lenses have imperfections which can be described by aberrations. Lens aberrations create optical path differences for each ray through the lens. Accounting for lens aberrations involves determining the location of the pattern in the layout. In our examples, we model variation from lens aberrations as a CD gradient across the chip.

III. Layout-Dependent Timing Analysis

Faults from within-die variation due to lithography modify the gate critical dimensions (CDs) based on neighboring gates in the layout (the proximity effect), the placement of the cells in the layout (lens aberrations), and the density of features on the mask (flare). Thus, we need to take these factors into account in timing analysis.

The conventional timing analysis tool has two classes: Class GATE_TABLE and GATE_INSTANCE. Class GATE_TABLE provides the technology information, such as input pin capacitance, delay and leakage tables, of each gate cell. Class GATE_INSTANCE describes the topology of the circuit (connections of instances of the gate cells). Objects of class GATE_TABLE are created by reading the technology library, and objects of class GATE_INSTANCE are generated from the hardware description language (HDL) file, which in our case is verilogHDL. For example, if a circuit has 2000 gates, which are classified into 30 types, then 2000 objects of class GATE_INSTANCE and 30 objects of class GATE_TABLE are required for timing analysis. The components that make up conventional timing analysis are shown in Figure 2.

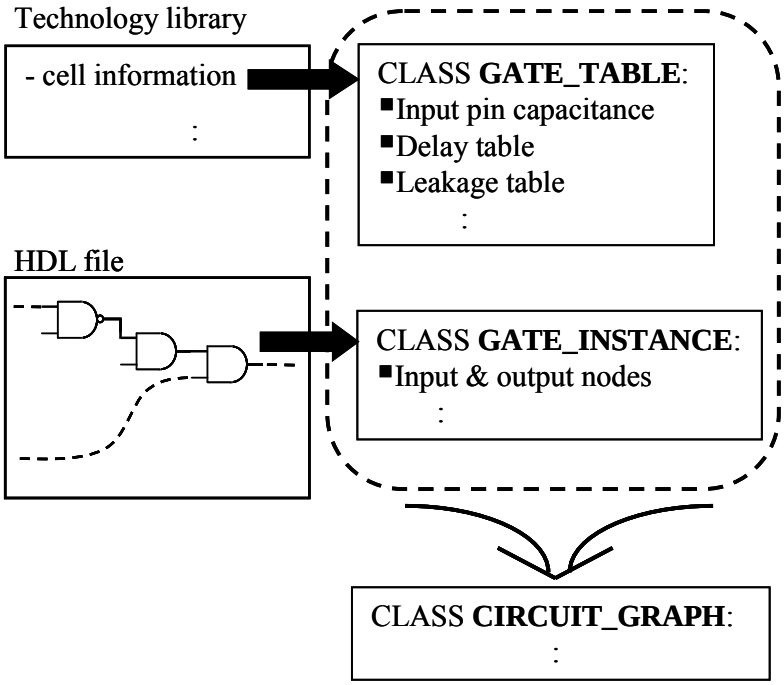


Fig. 2. Inside the conventional timing analysis flow.

The layout-dependent timing analysis flow adds an array GATE_Tr in class GATE_TABLE, pattern density tables in class CIRCUIT_GRAPH, and new variables to contain the location of the gate in class GATE_INSTANCE, as shown in Figure 3. Class GATE_Tr contains the proximity effect information, gate length, gate width, and transistor pin connections in the gate cell, as shown in Figure 4.

The goal of the layout-dependent timing analysis flow is updated critical path delays, which involves updating delays of cell instances. The delays of cell instances are a function of the CDs of gates within the instances, which in turn depend on layout features. Therefore layout data is extracted and fed into the timing analyzer, together with data on variations as a function of layout features (proximity effect, Coma, lens aberrations, flare). Based on this information, it is then straightforward to generate a new gate cell netlist just by writing the modified gate length and the other variables in GATE_Tr into class GATE_TABLE. The link between detailed transistor data in GATE_Tr and physical cell characteristics in GATE_TABLE requires delay re-characterization of the gate cell. This can be determined through various methods, including using Hspice simulation, analytic gate cell delay models [33], and efficient dynamic simulation [34]. Our work has used *Avant! Hspice* [35]. Gate characterization is followed by dynamic timing analysis, which will be discussed in the next section.

.

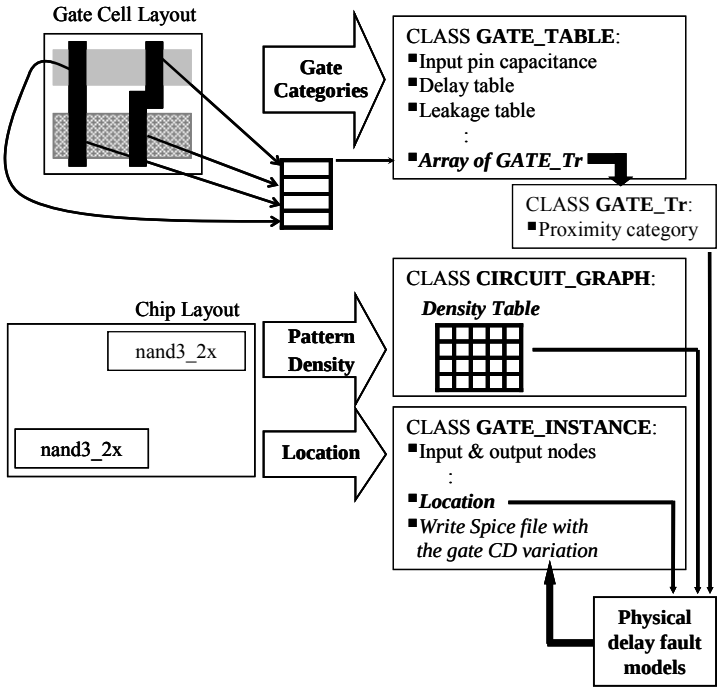


Fig. 3. Inside layout-dependent timing analysis flow.

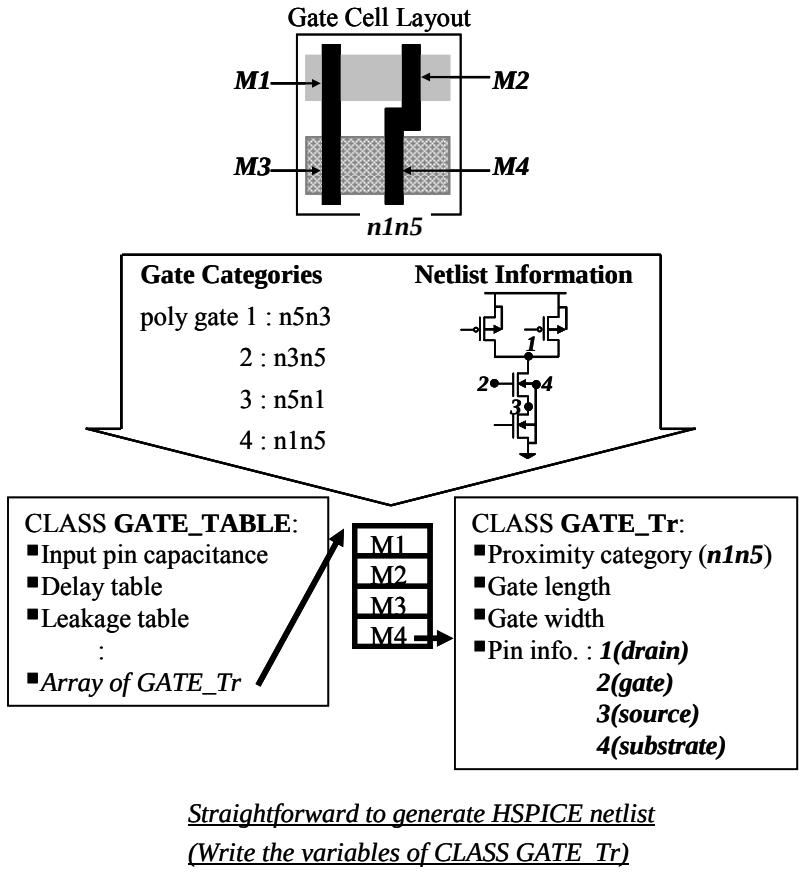


Fig. 4. Generation of the modified gate cell netlist, which includes neighborhood information for each transistor.

IV. Lithography Impacted Delay Fault Diagnosis

The delay fault diagnosis methodology begins by extracting the most significant critical paths through efficient path enumeration of the fault-free circuit. Path enumeration takes into account the transition time dependency and uses a depth first search algorithm, which is improved by pruning the search space. Next, test patterns are generated for the critical paths using a commercial tool, which in our case is *Synopsys Design Compiler* [36]. Using the resulting test patterns, faults are activated, and dynamic timing analysis is performed by applying the extracted test patterns to obtain delays. The delays are compared with a fixed threshold to determine if the test pattern generates a pass or fail. Pass/fail data is collected for all test patterns and all faults to construct the fault dictionary. Observed pass/fail patterns are compared with those in the dictionary through correlation to link the observed signature to a physical mechanism.

A. Path Enumeration

One disadvantage of the path delay fault model is that practical circuits have a very large number of paths. One of the ISCAS '85 benchmark circuits [37], c6288, has 10^{20} paths. Thus we focus on a set of most significant critical paths, under the assumption that other paths are unlikely to affect circuit speed. This may not be the case for delay faults caused by resistive defects, which can cause a large delay in a localized area. It is acceptable for faults caused by within-die variation because such faults cause many small deviations which when combined together result in faults. Thus we reduce the computational effort required to handle a huge number of paths by restricting paths to ones with delays over a specified threshold.

Several papers [12]-[15] provide algorithms to enumerate the longest paths. In [14],[15], the K longest paths are enumerated in the circuit while pruning the search space with the maximum possible delay. In [12],[13], on the other hand, the longest paths are selected such that they cover each gate in the circuit. In these approaches the search space is pruned by the maximum delay constraint, and the paths are checked to ensure sensitizability in order to eliminate any false paths. In all of these approaches the maximum delay to the sink is used for pruning in order to enhance efficiency. However, if the delay constraint is, say, 90% of the maximum circuit delay, these algorithms may have problems with memory management, since the number of paths satisfying this constraint can be extremely large. In [15], in order to limit the number of paths that need to be stored, the algorithm aims to extract the K longest paths. To this end, each node in the circuit graph is associated with a K array, storing the K longest paths to that node. This, however, could be a very large array and could still lead to memory problems.

Our work improves the depth first search (DFS) algorithm by pruning the search space through backward signal propagation, as used in static timing analysis [38]. As in [38], it includes the signal propagation effect, which is important for timing analysis, as will be explained in Section IV.C.

B. Some Definitions

Combinational circuits have a plurality of inputs and outputs. Combinational circuits may be represented as graphs, where gates are associated with edges and interconnect is associated with the nodes. Therefore, the signals flow from the input nodes to the output nodes through the gates (edges). By convention, adding a source node *s* and a sink node *f* to the graph makes the handling of the boundary conditions easier. Therefore, we modify the circuit graph accordingly. In addition, here are useful definitions relating to the circuit graph.

Definition 1: A timing graph is defined as a directed graph having one source and one sink node: $G = \{N, E, n_s, n_f\}$, where $N = \{n_1, n_2, \ldots, n_k\}$ is a set of nodes, $E = \{e_1, e_2, \ldots, e_l\}$ is a set of edges, $n_s \in N$ is a source node, and $n_f \in N$ is a sink node. Each edge $e \in E$ is simply an ordered pair $e = (n_i, n_j)$ of nodes, where $n_i, n_j \in N$.

Definition 2: A path P of a timing graph $G = \{N, E, n_s, n_f\}$ is a sequence of its nodes $P = (n_a, n_b, \ldots, n_z)$ such that each pair of adjacent nodes n_i and n_j has an edge $e_{ij} = (n_i, n_j)$.

Definition 3: The path delay d_P of path P is defined as $\sum_{e_{ij} \in P} d_{ij}(s_i)$, where $d_{ij}(s_i)$ is a delay of an edge e_{ij} on path P with input transition time s_i, and the summation is over all edges belonging to path P. The edge delays are also a function of the loading capacitance, but we have not denoted loading capacitance as an argument since it is fixed for a specific network.

In order to understand the sequence of operations through a path, a graph is divided into layers. The layers begin at the inputs and end at the outputs. Most static timing analysis algorithms progress from the inputs sequentially through the layers of the graph until reaching the outputs. Backward signal propagation, on the other hand, starts at the outputs and progresses through the layers to the inputs.

Definition 4: A node, n_i, is connected to a set of adjacent nodes, n_j, on successive layers by edges, e_{ij}. The set of nodes connected to node, n_i, on successive layers is called $SUCC(i)$.

C. Signal Propagation Effect

Previous path enumeration algorithms [12]-[15] assume a fixed edge delay (delay between each of the inputs and the output of the gate). The gate cell delay is actually a function of the transition time of the input signal and the loading capacitance. In fact, when two signals arrive at an input, the one that arrives first can determine the longest path if the transition time (slope) is longer. This is why we need to take into account the signal propagation effect.

The loading capacitance is fixed for a specific network. Therefore, the transition time is only dependent on the path being investigated. In fact, if two signals are propagating to a single input, *i*, of a gate, S_{ai} and S_{bi}, the edge delays through the gate, d_{ai} and d_{bi}, respectively, are different as shown in Figure 5. The result is distinct delays between node *i* and the sink. To accurately find the critical paths, we then need to take into account all signal transition times within a path. If we do this by brute force, the complexity of the problem of finding the K critical paths becomes exponential.

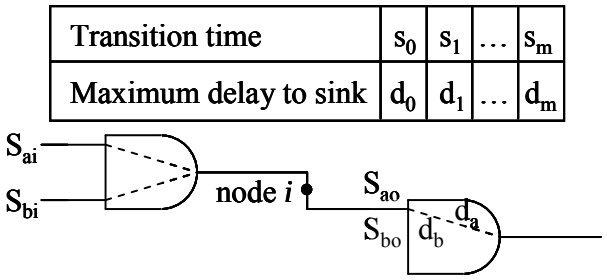


Fig. 5. Transition time dependency of the maximum delay from node *i* to the sink (outputs), where s_{ai} and s_{bi} are distinct signals arriving at node *i*.

The estimation of critical path delay in static timing analysis suffers from a similar problem. In [38], this problem is addressed by backward signal propagation to determine the slack at each node. Based on the initial computations of slack, this approach maintains a table of the maximum delay to the sink as a function of the transition time at each node, as shown in Figure 5. As illustrated in Figure 5, each node in the graph stores a table. This table indicates the delay to the sink as a function of transition time.

We borrow backward signal propagation to overcome the drawback of previous path enumeration approaches. Specifically, we start at the outputs and progress towards the inputs by creating a table for each preceding node until we reach the inputs. To create the table, we consider a specific transition time at node, n_i. We then look up the resulting delay for the successive edge, e_{ij}. We also look up the delay to the sink for the successive node, n_j, which has been computed previously. We add these delays together, and take the maximum among all successive nodes, to create the entries in the table for the given transition time. This process is repeated for each transition time. Figure 6 shows the backward signal propagation algorithm.

The table at each node stores a set of discrete values of input transition times and delays to the sink. We have assumed six representative transition times at each node, as in [38], although this choice is arbitrary. Because we obtain the delay by interpolating the delay table, more transition times make the delay estimate more accurate, but more transition times also increase computation time and consume more memory.

D. Improved Depth First Search (DFS) Algorithm with Search Space Pruning

As we have the table of the maximum delay to the outputs as a function of the transition time at each node, the DFS algorithm to determine all critical paths with a delay greater than a threshold can be easily upgraded. While proceeding with DFS, we start at the inputs and estimate the maximum delay for all paths emanating from each node. At each node, if the delay is less than the threshold delay, searching for paths emanating from that node is terminated, as shown in Figure 7. All complete paths that reach a primary output are saved on the hard disk and are not kept in memory. Figure 8 shows the pseudo code of the pruned DFS path enumeration algorithm.

Once the paths are determined, they are converted to a test set using a commercial ATPG tool which determines the primary input signals that can sensitize the path and the resulting values at the primary outputs. The test set is then compacted to minimize the number of patterns required to sensitize the longest paths.

Initialize the delay table of the sink node n_f as a function of the transition time.

{Backward propagation}
Visit a node layer in reverse topological order.
For each node *i* in the same order
{
For each edge $e_{ij} = (n_i, n_j)$, where $j \in SUCC(i)$.
{
Propagate the delay table from n_j through e_{ij}
using $D_{ij}(s) = D_j(s_{ij}(s)) + d_e(s)$,
where $s_{ij}(s)$ = transition time at n_j,
$D_j(s_{ij}(s))$ = maximum delay of n_j to sink,
$d_e(s)$ = edge e_{ij} delay.
}
Establish the delay table of n_i
using $D_i(s) = \max(D_{ij}(s), D_{ik}(s), \dots)$.
}

Fig. 6. Backward delay_table_to_sink propagation.

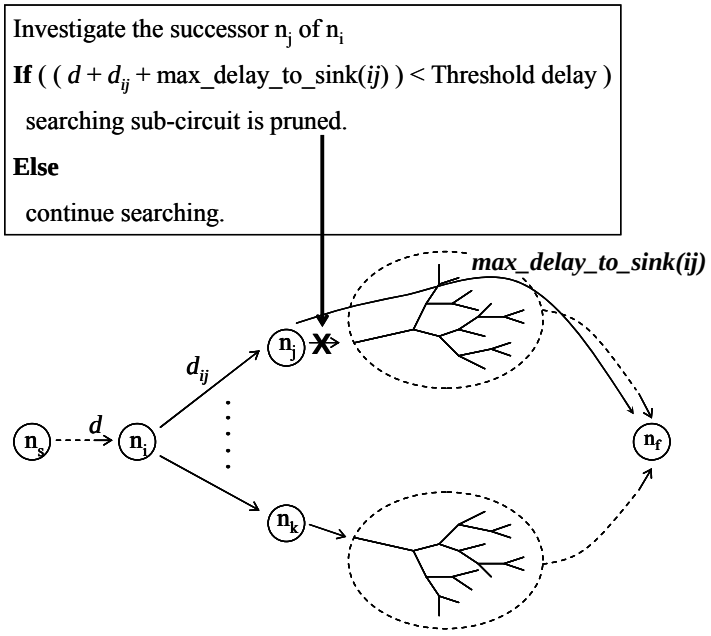


Fig. 7. Pruning in the DFS path enumeration algorithm. The maximum delay to a sink (output) is stored at each node. If this delay is below a threshold at a specific node, enumeration of paths that involve branches beyond that node is terminated.

```
T = threshold delay, looking for paths over T in G.

{Create the source node s and the sink node f.}
{Compute the maximum delays to sink.}
{Sort the successors of each node.}

{Path enumeraton using pruned DFS algorithm}
 prunedDFS (s);

 prunedDFS (Node i)
 {
     If ( i = sink node )
    save path information;
    return;

  For each  j ∈ SUCC(i)
  {
   If ( ( d + d_ij + max_delay_to_sink(j) ) < T )
     Return;
   Else
     save arrival time ( d + d_ij ) and transition time;
     prunedDFS (j);
  }
 }
```

Fig. 8. Pruned DFS path enumeration algorithm pseudo code.

E. Dynamic Timing Analysis for Fault Simulation

Path enumeration has not taken into account details about the signal train. Therefore, we use dynamic timing analysis to verify temporal correctness of the selected test patterns and to perform fault simulation.

In dynamic timing analysis the signal is propagated through the gates in topological order, just as with static timing analysis. The signal train is input into each gate. The signal train consists of the arrival time train, the transition time train, and the logic state train, as shown in Figure 9.

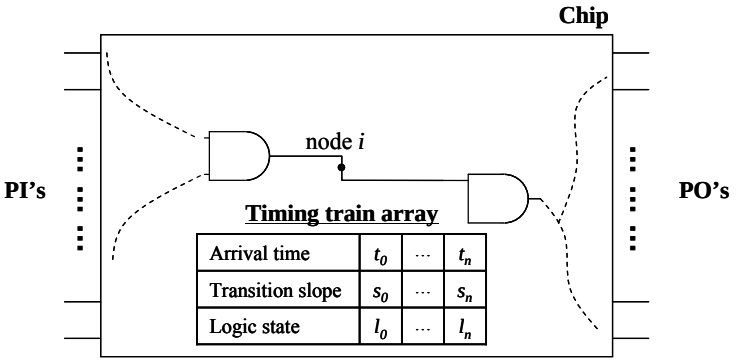


Fig. 9. A node has signal train for dynamic timing analysis.

The input signals to the gate are propagated to the output, as in Figure 10. First, the output arrival time and logic state trains are obtained with zero gate delay. Second, for each transition of the output, it is determined which input changes the output, and the gate delay and transition times caused by each input transition are obtained from the gate cell delay table. Finally, the output arrival time train is updated by adding the delay, and the transition time train at the output is established.

Fault simulation relies on dynamic timing analysis. To simulate a fault, the revised delay tables that reflect faulty behaviors are generated. The dynamic timing analyzer then inputs the revised faulty delay tables and test patterns to determine the delay associated with each test pattern.

In our implementation of dynamic timing analysis, we do not consider signal coupling between interconnect lines (crosstalk) [39], data dependent delays [40]-[42], and the closeness dependency of input transitions.

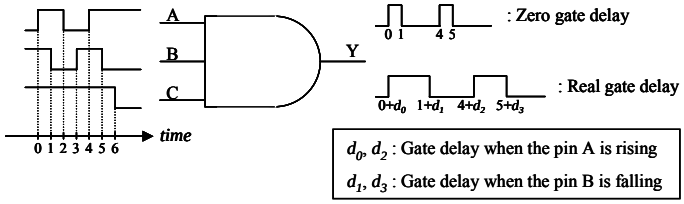


Fig. 10. Dynamic signal propagation with a signal train.

F. Fault Diagnosis

The patterns selected for test application have delays greater than a threshold. For our examples, this threshold was set to 90% of the maximum delay. Therefore, for the 90% threshold, all selected test patterns, *t,* have delay $d_t > 0.9 \cdot d_{\max}$. Then, if the clock frequency is increased to $f = 1/0.9d_{\max}$, all of the selected patterns, with delay greater than 90% of the maximum delay will fail for fault-free circuits. Even in the presence of die-to-die variation, because die-to-die parameter variation is almost 100% correlated, all of the selected patterns will fail. However, if the circuit contains a fault caused by within-die variation, it is possible that some of the patterns will pass. The patterns that pass are associated with specific faults, and each fault is associated with a pass/fail signature for the applied test patterns. Consequently, the test methodology involves determining the maximum delay, $d_{\max}$, for each circuit instance, applying test patterns at $f = 1/0.9d_{\max}$, and determining the passing patterns, where $d_{\max}$ is a function of process parameters and the fault being considered. Faults are detectable if there is at least one passing pattern. Faults are diagnosed by correlating the pass/fail signatures, i.e. if the pass/fail patterns match those in the dictionary.

V. Experimental Results

The methodology described in the previous section has been implemented in Java on a Solaris 2.8 running on a Sun E-450 Server with four CPUs and 4GB RAM. Experiments have been performed on ISCAS85 circuits [37]. Table III shows the execution time for critical path extraction and test pattern simulation. It also shows the number of critical paths, true paths, test vectors, and the maximum delay for each circuit under fault-free conditions.

TABLE III
Execution Times and Experimental Results

Circuit	DFS Time [sec]	Pruned DFS Time [sec]	Paths*	True Paths**	Test Vectors	Maximum Dynamic Delay [nsec]	Test Pattern Simulation Time [sec]
c432	16.5	8.2	16,598	40	40	5.85	42.8
c499	2.4	0.7	1340	181	62	2.81	25.8
c880	3.8	0.2	112	95	63	3.72	55.6
c1355	976.1	229.8	229,660	4441	332	3.71	148.8
c1908	185.0	25.9	25,232	1025	942	4.61	1581.6
c2670	395.7	379.0	383,839	18	18	5.15	50.7
c3540	7417.4	233.9	159,305	225	225	6.58	483.3
c5315	486.2	79.8	71,410	1874	1874	6.04	7948.6
c7552	248.4	10.9	11,095	139	139	4.98	835.7

*Paths with delays over 90% maximum static circuit delay (c2670: over 70%)
** Sensitizable Paths

In the table, the DFS time refers to the time required to enumerate all paths with a delay above 90% of the maximum circuit delay, except for c2670, where a 70% threshold is used. The pruned DFS time is the execution time when pruning is used, as described in Section IV.D. For c2670, the threshold was set to 70% to obtain a reasonable number of true paths. We can see that the execution time for the pruned DFS algorithm is 10 times faster than DFS on average.

Circuits c432 and c2670 did not get much reduction in execution time due to pruning. In the case of c2670, 56% of the total paths have delays over 70% of the maximum circuit delay. In the case of c432, 20% of the total paths have delays over 90% of the maximum circuit delay. In other words, both of these circuits have a large number of long paths. Besides these two circuits, the other circuits have many short paths. Therefore, the search space was reduce to below 20% of the paths in these circuits, and pruning was effective.

Table III also lists the number of "true paths". It's well known that ISCAS '85 circuits are poorly testable, except for c880 [43], where almost all paths are true paths.

For each true path, test vectors are simultaneously extracted. The test vectors are compacted by eliminating unknowns in the input patterns. The number of test vectors is shown in Table III. Finally, fault simulations are performed for each test pattern, using dynamic timing simulation. The dynamic fault simulation time required to generate the entire fault is shown in the last column of Table III.

The optical lithography faults tested in the experiment are shown in Table IV. In order to generate the fault dictionary for pass/fail signatures a maximum variation of 10% of the gate length is assumed for each fault.

TABLE IV
Physical Origins of Faults Considered

Physical Origin Code	Optical Effects
0	Optical proximity effect (large n1n1, small n5n5)
1	Coma (large n5n1, small n1n5)
2	Lens aberrations (Left->Right)
3	Optical proximity effect (Reverse trend)
4	Coma (Reverse trend)
5	Lens aberrations (Right->Left)
6	Lens aberrations (Bottom->Top)
7	Lens aberrations (Top->Bottom)

Before we look at diagnosability of the causes of within-die variation, we need to look at detectability. To be detectable, at least one pattern has to pass tests applied at frequency $f = 1/0.9d_{\max}$. The results are shown in Figure 11. This figure indicates that detectability increases for larger ranges of variation, i.e. 15% variation is more easily detectable than 5% variation. Moreover, some circuits displayed poor detectability, while within-die variation faults in others appear to be detectable. To better understand the detectability results, delay distributions for a selected set of circuits are shown in Figure 12. It appears that larger circuits show improved detectability compared to smaller circuits, indicating potentially improved results for large circuits. Secondly, the circuits for which larger numbers of test vectors could be generated had improved detectability. Specifically, more than 900 test vectors were generated for c1908 and c5315, and these circuits displayed the best detectability. Detectability versus the number of test vectors is illustrated in Figure 13.

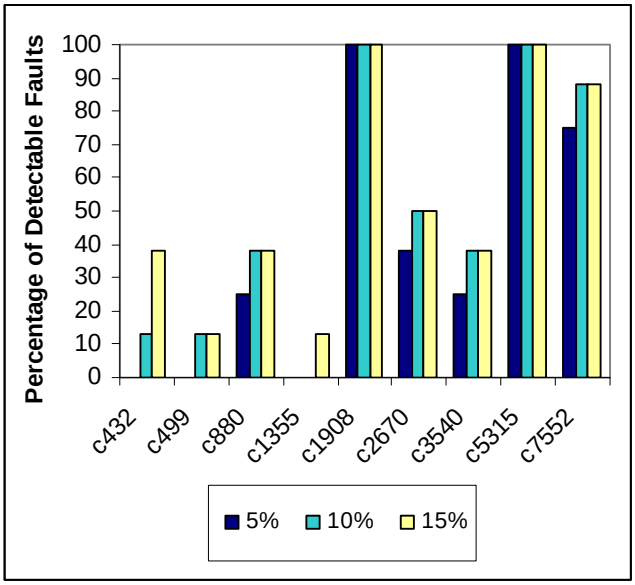
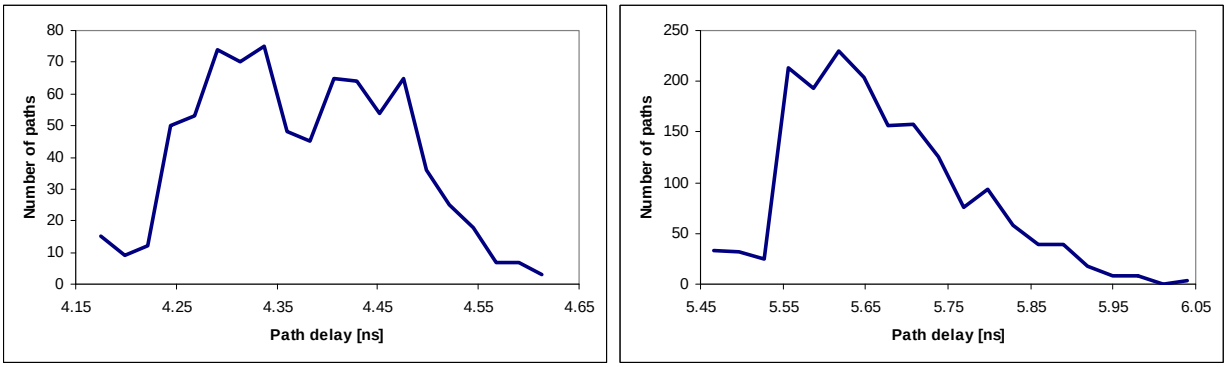
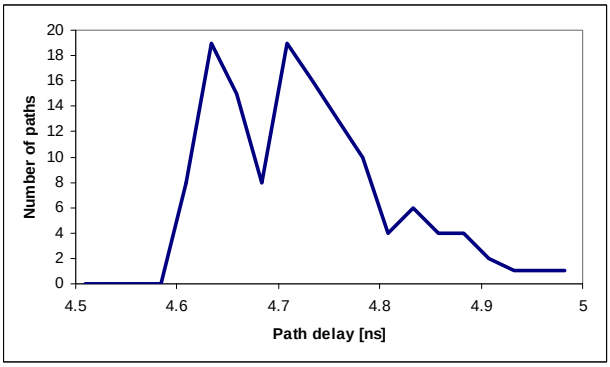


Fig. 11. Percentage of detectable faults as a function of range of within-die variation (5%, 10%, 15%).

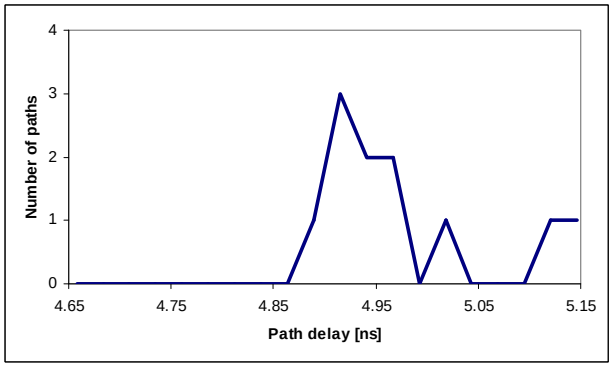


(a) c1908

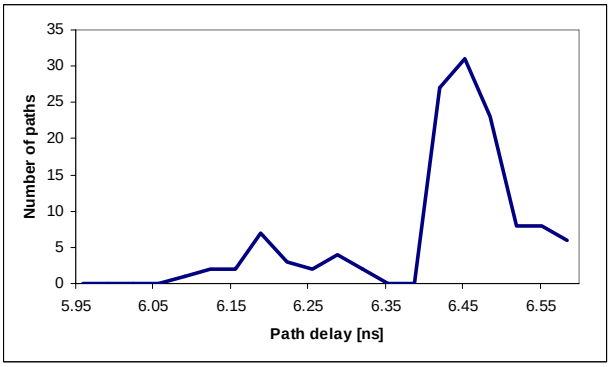
(b) c5315



(c) c7552



(d) c2670



(e) c3540

Fig. 12. Delay distribution for some ISCAS '85 circuits in the delay range from $0.9 \cdot d_{max}$ to d_{max}. Most path delays in c2670 and c3540 are crowded closer to d_{max}. Since the test method requires passing vectors, with delay $< 0.9 \cdot d_{max}$, these circuits require very large amonts of within die variation to generate passing vectors. The path delays in c1908, c5315, and c7552 are distributed evenly between $0.9 \cdot d_{max}$ and d_{max}.

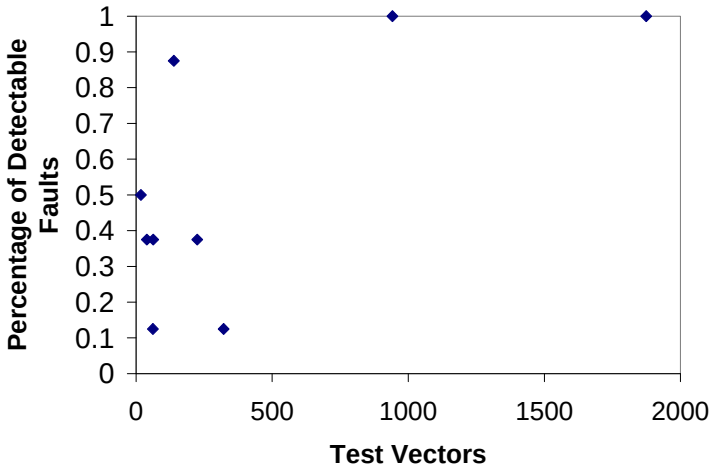


Fig. 13. Detectability vs. the number of test vectors for 15% range of variation.

Diagnosis requires that we distinguish among different faults. In order to do this, the correlation between the pass/fail patterns associated with the fault to be diagnosed must be significantly higher than correlations with other faults in the dictionary. We first consider constructing the dictionary containing faults associated with a 10% range of variation, and testing each circuit instance at frequency $f = 1/0.9d_{\max}$, where $d_{\max}$ is the maximum operating frequency of the faulty circuit instance. The patterns in the dictionary are denoted p_{ik}, where i is the physical origin code and $k = 10\%$, the range of variation of the fault. In order to evaluate diagnosability, we then suppose that a circuit contains a fault, j, where j is the physical origin code, or size l, where l is the range of variation, and determine the pass/fail pattern, p_{jl}, for tests applied at frequency $f = 1/0.9d_{\max}$, where $d_{\max}$ is the maximum operating frequency of the faulty circuit instance. Correlations are then performed with all patterns in the dictionary, i.e. we compute $\rho(p_{jl}, p_{ik})$. The fault is correctly diagnosed if $\rho(p_{jl}, p_{ik}) > \max_{i \neq j} \rho(p_{jl}, p_{ik})$, where $k = 10\%$ and l is an arbitrary range of variation. Otherwise, we say that a fault is mis-diagnosed.

In our examples, we considered three circuits where faults with a 10% range of variation are mostly detectable: c1908, c5315, and c7552. Figures 14, 15, and 16 compare $\rho(p_{jl}, p_{ik})$ and $\max_{i \neq j} \rho(p_{jl}, p_{ik})$ for c1908, c5315, and c7552, respectively. These figures indicate that faults with a 10% and 15% range of variation are diagnosable for c1908 and c5315. Diagnosability decreases for a 5% range of variation. Diagnosability for c7552 is less successful, because the number of test vectors is much smaller than c5315 and c1908. As a result, the set of vectors that can be used to distinguish among the faults is too small.

In order to attempt to improve diagnosability for smaller ranges of variation, we have considered adding pass/fail patterns associated with a 5% range of variation to the dictionary. As a result, the dictionary contains patterns, p_{ik}, where i is the physical origin code and $k = 5\%$ and 10%, the range of variation of the fault. Then, we determine p_{jl} for an arbitrary fault with physical origin code j and range of variation l and check to see if the fault is correctly diagnosed by determining if $\max_{k=5\%,10\%} \rho(p_{jl}, p_{ik}) > \max_{i \neq j, k=5\%,10\%} \rho(p_{jl}, p_{ik})$ where $k = 5\%$ and 10% and l is an arbitrary range of variation. Figures 17 and 18 compare $\max_{k=5\%,10\%} \rho(p_{jl}, p_{ik})$ and $\max_{i \neq j, k=5\%,10\%} \rho(p_{jl}, p_{ik})$ for c1908 and c5315, respectively. These figures indicate that diagnosability of 5% faults in improved, as the expense of a few of the 15% faults.

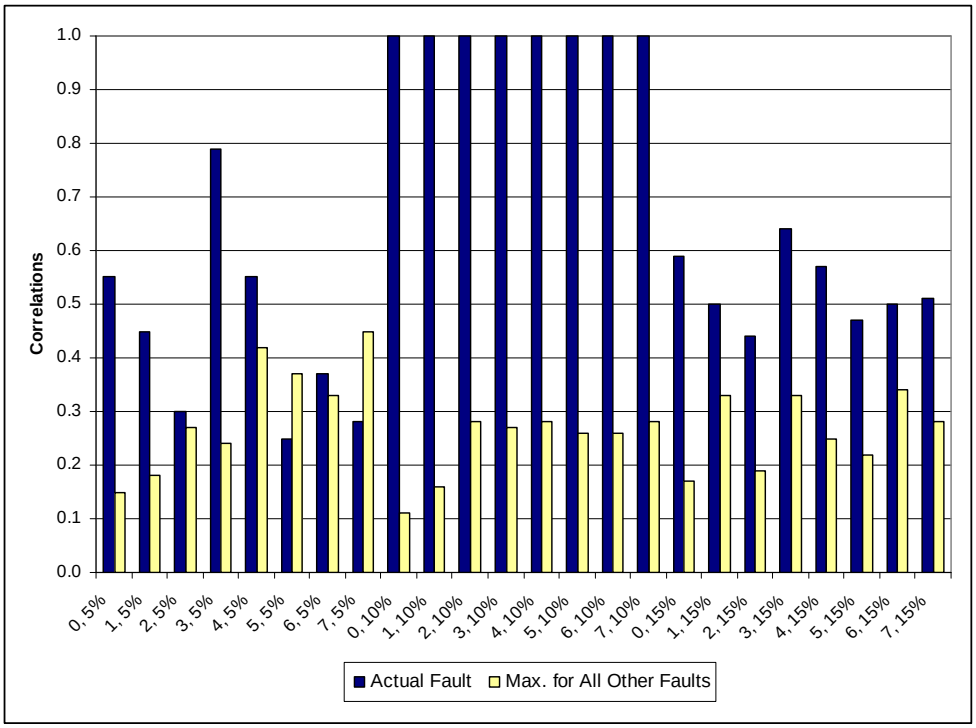


Fig. 14. Correlations between pass/fail patterns for faults as a function of range of within-die variation (5%, 10%, 15%) for c1908. The labels indicate the physical origin code and the range of variation of the fault. Each pair compares the correlation are the actual fault in the dictionary (where the range of variation is 10%) and the maximum correlations between pass/fail patterns for all other faults in the dictionary (excluding the actual fault).

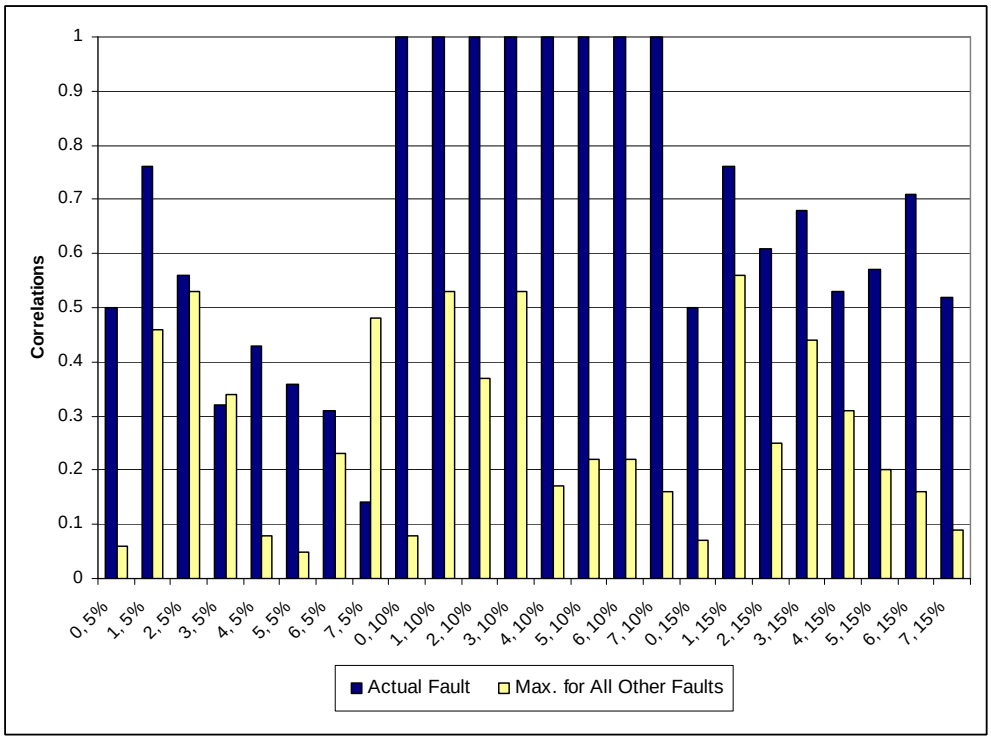


Fig. 15. Correlations between pass/fail patterns for faults as a function of range of within-die variation (5%, 10%, 15%) for c5315. The labels indicate the physical origin code and the range of variation of the fault. Each pair compares the correlation are the actual fault in the dictionary (where the range of variation is 10%) and the maximum correlations between pass/fail patterns for all other faults in the dictionary (excluding the actual fault).

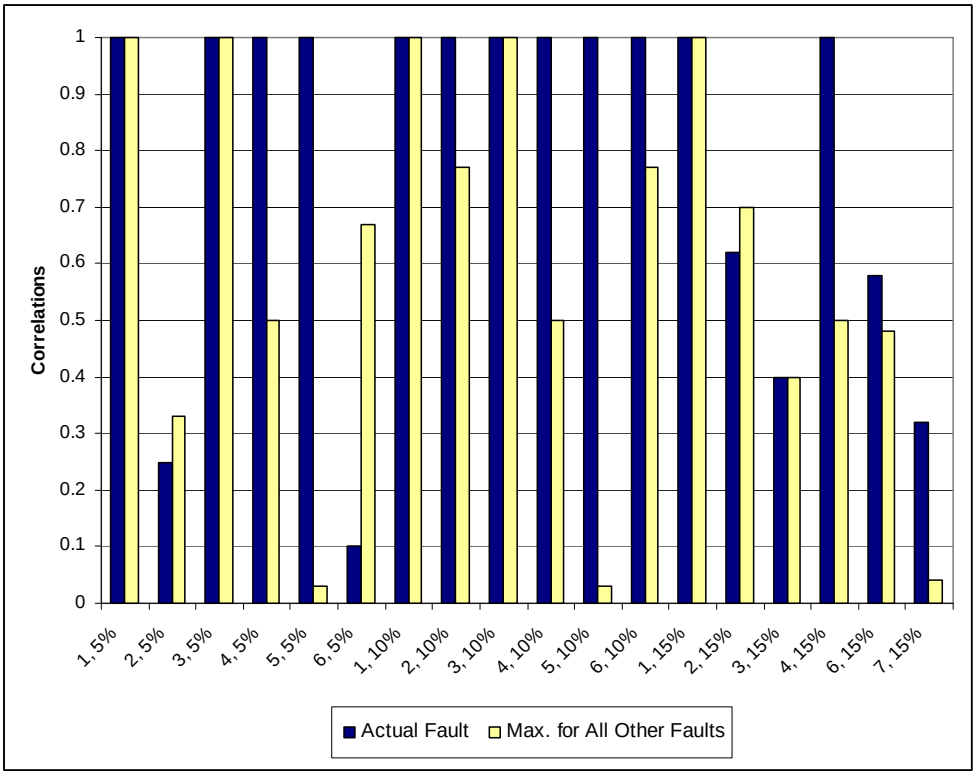


Fig. 16. Correlations between pass/fail patterns for detectable faults as a function of range of within-die variation (5%, 10%, 15%) for c7552. The labels indicate the physical origin code and the range of variation of the fault. Each pair compares the correlation are the actual fault in the dictionary (where the range of variation is 10%) and the maximum correlations between pass/fail patterns for all other faults in the dictionary (excluding the actual fault).

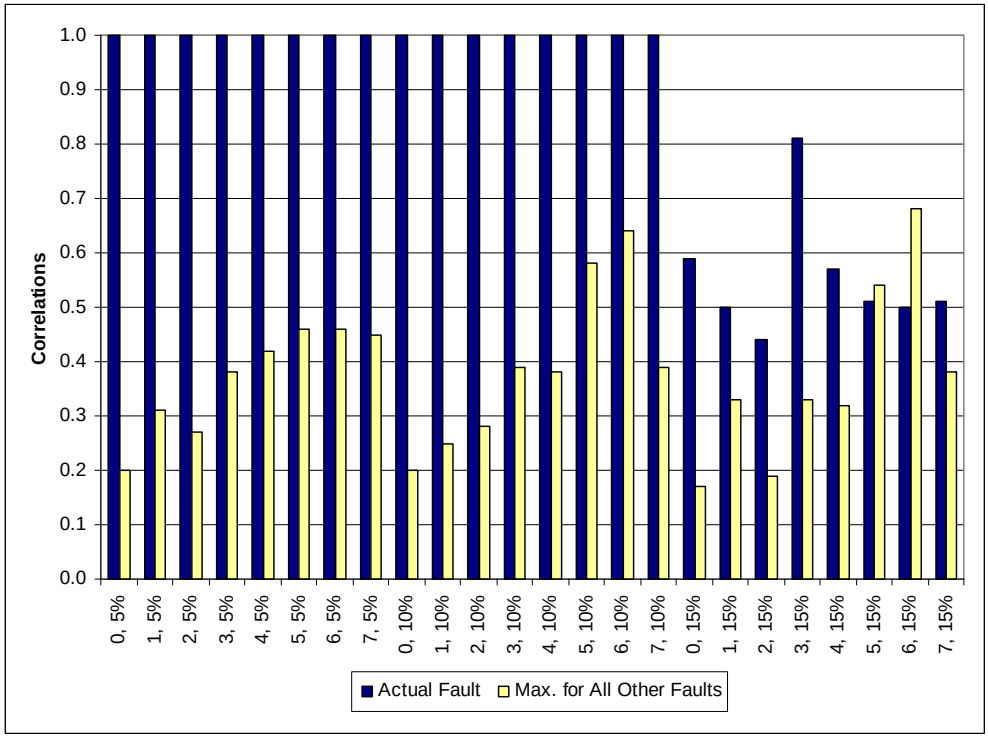


Fig. 17. Correlations between pass/fail patterns for faults as a function of range of within-die variation (5%, 10%, 15%) for c1908. The labels indicate the physical origin code and the range of variation of the fault. Each pair compares the correlation are the actual fault in the dictionary (where the range of variation is 10%) and the maximum correlations between pass/fail patterns for all other faults in the dictionary (excluding the actual fault).

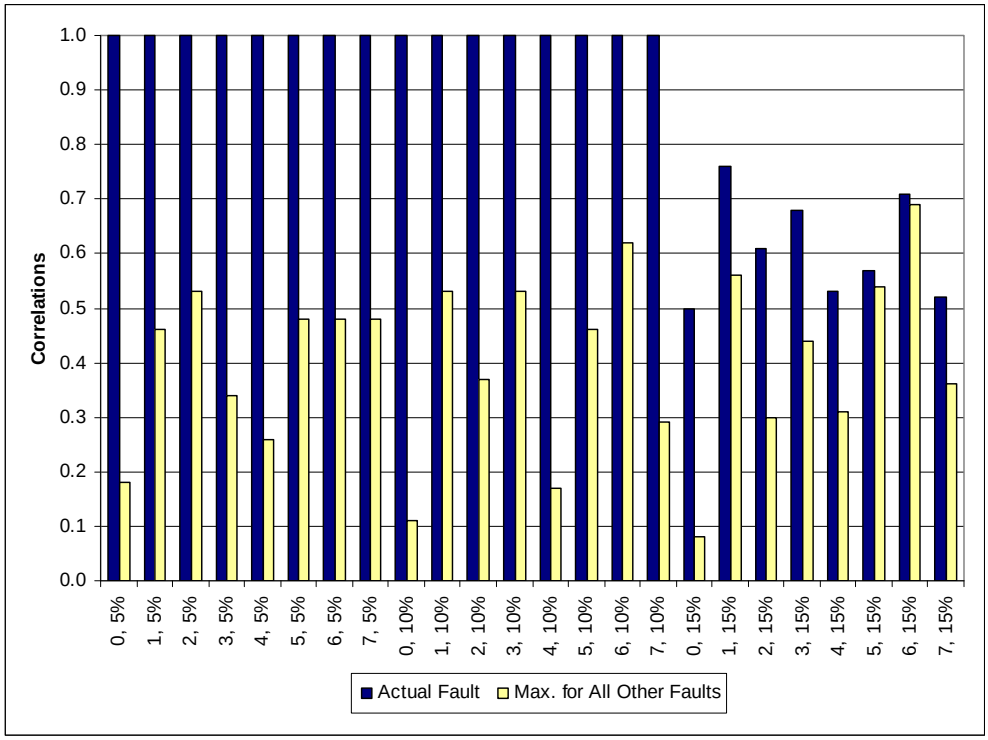


Fig. 18. Correlations between pass/fail patterns for faults as a function of range of within-die variation (5%, 10%, 15%) for c5315. The labels indicate the physical origin code and the range of variation of the fault. Each pair compares the correlation are the actual fault in the dictionary (where the range of variation is 10%) and the maximum correlations between pass/fail patterns for all other faults in the dictionary (excluding the actual fault).

VI. Implementation Issues

The application of delay test patterns in scan-based designs has several issues. First, we must consider the method for test pattern application, and second, we must consider clock generation.

Test pattern application requires a sequence of two patterns, one to initiate the logic into a known state and a second to trigger the targeted transitions in the circuit. Transitions are launched from a path's starting point and captured at the path's end point. It is straightforward to scan in the initial vector, and capture the output in scan cells. The challenge is the application of the second vector at the input scan cells. There are two approaches to generating the second vector: launch-off-shift and broad-side. Both of these approaches suffer limitations, in that they restrict the set of patterns that can be applied. In the launch-off-shift approach, the second vector is required to be the next (i.e. one bit shift) pattern in the scan chain. This creates shift dependencies, and is limited by correlations between bits in the first and second vectors. In the broad-side approach, the second vector is applied through the logic. This approach is restricted by the ability to generate the required second vector through the incoming logic cone.

Implementation of delay test requires precise control of the time interval between launch and capture clocks, there the interval between the launch and capture clocks is governed by the required operating frequency. In general, at-speed test at wafer probe is most easily achieved by generating the clock on chip with a phase-locked loop, and many chips incorporate phase-locked loops for internal clock generation. However, test application may suffer from excessive power dissipation. And, in wafer probe, an appropriate cooling environment may not be available. The alternative is to operate test at a much lower frequency, except for the targeted transition being testing, i.e. the transition between launch and capture. This requires generating two test clock signals with a will controlled interval between launch and capture signals. One approach to achieve this is through a delay-locked loop (DLL). We have designed such a delay-locked loop using 0.18um technology. The DLL produces delayed versions of the core clock. Details of the design are described in the following sections.

A. Circuit Architecture

A DLL can provide multiple phase-shifted clock signals from which the two desired test clocks can be chosen. Moreover, the negative feedback mechanism of a DLL ensures that the delay of the entire VCDL is always one clock period of the input signal. This delay does not vary with process, supply voltage, and temperature (PVT).

The architecture of the proposed DLL clock generator is shown in Figure 19. Low power, wide lock range, short locking time, and low jitter are the focuses of the design. Wide locking range allows the design to be used for a wide variety of applications, since the DLL will lock to a wide set of frequencies. Short locking time aids in test execution, since if the locking

time is short, large numbers of cycles to not have to be wasted before making a measurement. Finally, low jitter enables accurate measurement.

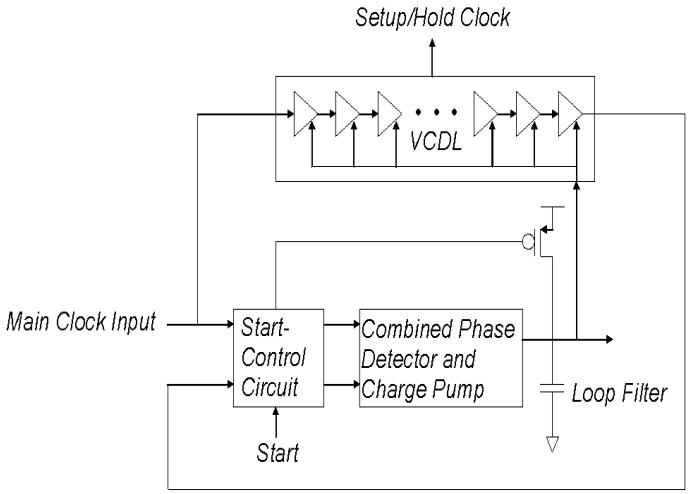


Fig. 19. Architecture of the proposed DLL clock generator.

The DLL clock generator is composed of a combined phase detector and charge pump circuit, a loop filter, a VCDL, and a start-control circuit. In the DLL, the input clock signal propagates through the VCDL and develops phase shift at every delay stage. The phase shift (or delay) of each stage is controlled by the voltage of the loop filter. The phase of the output signal from the end of the delay line is compared with the phase of the input clock signal in the phase detector. The phase error information is then transferred to the charge pump to adjust the voltage of the loop filter which, in turn, changes the delay of the VCDL. Due to such a negative feedback mechanism, the phase error is gradually reduced until it finally approaches zero. At that time, the delay of the entire VCDL becomes exactly one clock period, and the voltage of the loop filter is stabilized, which indicates that delay lock has been established.

B. Component Design

The phase detector (PD) and the charge pump (CP) are two of the most critical components within a DLL. The task of the phase detector is to detect the phase difference between the reference signal and the VCDL output signal. The charge pump performs the function of adjusting the voltage of the loop filter in response to the detected phase error information.

B.1. Phase Detector and Charge Pump

In practical designs, it is desirable to integrate the phase detector and the charge pump together in order to minimize die area and power dissipation, as well as to reduce jitter. Such an approach was previously taken in [44]. However the design in [44] suffers from two drawbacks. Firstly, the charging or discharging current varies during the charging or discharging process, which may slow down the DLL locking process. Secondly, the charging and discharging currents are not very well matched, which can result in higher phase noise and greater clock skew. We address these two problems by using fixed charging and discharging currents, as well as a high-precision current mirror to improve the matching between the charging and discharging currents.

The phase detector detects the phase difference between the output signal from the VCDL and the input reference signal. An XOR gate implements the phase detector function. It generates UP and DOWN signals indicating if the frequency needs to be increased or decreased. This signal is input to the charge pump in terms of pulses. The charge pump adjusts the voltage of the loop filter and thereby alters the VCDL delay according to the phase error information from the phase detector. In principle, the CP consists of two controlled switches, one current source, one current sink, as shown in Figure 20, where the switches are controlled by the UP and DOWN pulses. The charge pump adds or subtracts charge from the loop filter (capacitor). This charging or discharging process continues until lock is achieved. After lock is achieved, equal charging and discharging pulses continue in order to minimize jitter [45].

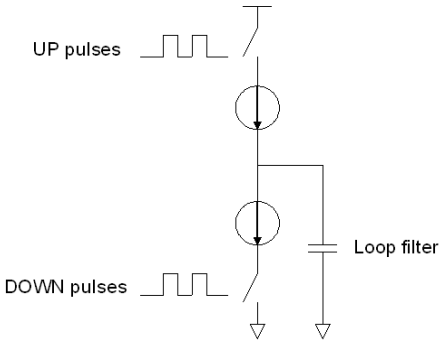


Fig. 20. A simplified phase detector and charge pump diagram.

In the design of the charge pump, an important choice must be made between a single-ended topology and a differential topology. A single-ended topology has the advantages of smaller area and lower power dissipation. However, a single-ended topology is more vulnerable to supply and substrate noise compared to a differential topology. Our DLL uses a single-ended topology because of its simpler structure and lower power dissipation.

In general, there are three different configurations for single-ended charge pumps: switching in the source, switching in the drain, and switching in the gate [46]. Among the three configurations, switching in source is preferred due to its simpler structure, lower power dissipation, and faster switching time [46]. Furthermore, it is known that for CMOS circuits, current switching provides a faster switching speed than voltage switching if all other conditions are the same [47].

Based on the above considerations, a combined phase detector and charge pump circuit has been designed which provides a fast switching speed and excellent charging and discharging current matching. Moreover, the proposed structure introduces very little charge injection and clock feedthrough. As a result, phase noise is reduced. The circuit is shown in Figure 21.

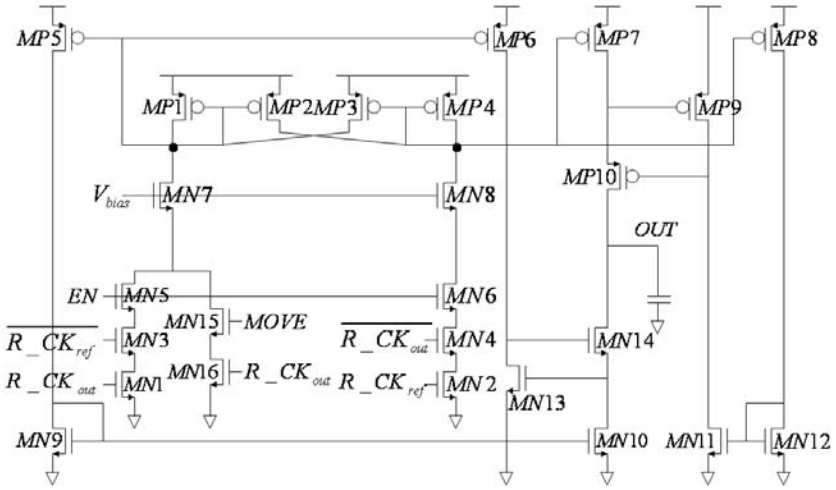


Fig. 21. Combined PD and CP circuit.

The circuit is composed of three functional blocks: the phase error detection block, which includes transistors MN1 to MN6 and MN15 to MN16; the current switching block, which includes transistors MN7 to MN8 and MP1 to MP4; and the current mirror block, which includes transistors MN9 to MN14 and MP5 to MP10. The operation of the circuit is described as follows.

The phase error detection block is composed of three AND gates, shown in Figure 22. Each AND gate is composed of two or three series-connected NMOS transistors. During the locking process, one of the AND gates is turned on corresponding to the phase relationship between R_Ck_{REF} and R_Ck_{out}. The corresponding current source, MN7 or MN8, is activated and a fixed current is generated and mirrored to charge or discharge the loop capacitor C. Such an implementation of the AND gates generates simultaneous charging and discharging current pulses I_{up} and I_{down} for equal durations during every clock cycle after the loop is locked (i.e., in the steady state), which are needed to avoid a dead-band region in the phase detector and thus to avoid additional input tracking jitter. The simultaneous pulses I_{up} and I_{down} must be identical as well as very narrow in order to avoid possible disturbances of the loop filter voltage in steady state. The current pulses generated by the

series transistor structure are significantly narrower than what is possible with a voltage signal [44]. Moreover, the current switching block further narrows the current pulses. The simulated current pulses in steady state are shown in Figure 23.

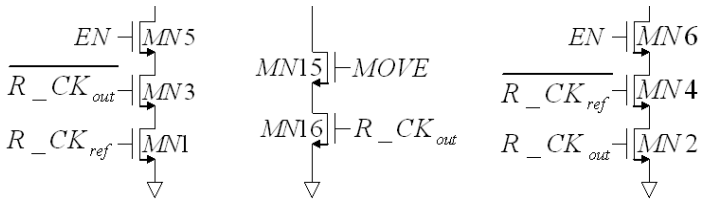


Fig. 22. Three pseudo AND gates.

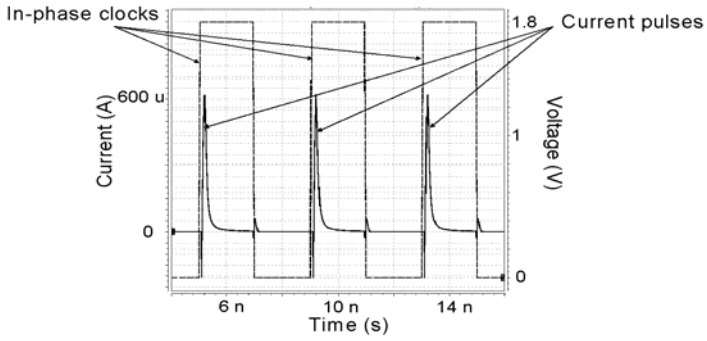


Fig. 23. Simultaneous current pulses in steady state.

The three AND gates are controlled by the two clock signals, R_Ck_{ref} and R_Ck_{out}, plus their complements, as well as MOVE and EN, which are used to select the time window during which the AND gates become active. The MOVE and EN signals are generated by the circuits shown in Figure 24.

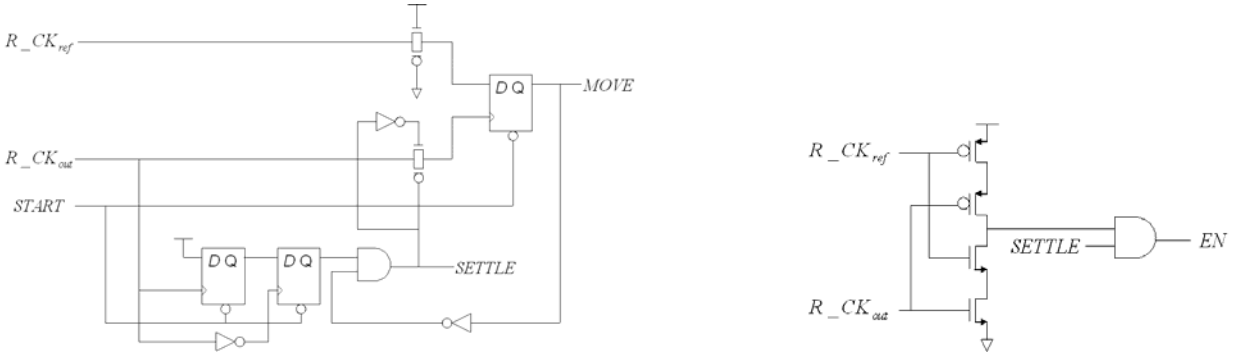


Fig. 24. Generation of the SETTLE, MOVE, and EN signals: (a) MOVE and SETTLE and (b) EN.

If the phase difference between R_Ck_{ref} and R_CK_{out} is less than half of the clock period, the values of MOVE and SETTLE are LOW and HIGH, respectively. In this case, the AND gate controlled by MOVE, and R_Ck_{out} will be disabled. The other two AND gates work together with the current sources MN7 and MN8.

As can be seen in Figure 24, the enable signal is asserted as soon as both R_Ck_{ref} and R_Ck_{out} become low and remains asserted until both of them become high. In response to the asserted EN signal, the phase error detection block starts its operation and generates output pulses with the same width as the phase difference between R_Ck_{ref} and R_Ck_{out}. The pulses turn on the corresponding current source to start the charging or discharging process. Due to the negative feedback mechanism, the phase difference becomes smaller and smaller, eventually approaching zero.

On the other hand, if the phase difference between R_Ck_{ref} and R_Ck_{out} is greater than half the clock period, the MOVE signal is asserted and the SETTLE signal becomes LOW. Because the MOVE signal is HIGH, the AND gate which is

controlled by MOVE and R_Ck_{out} takes over. This AND gate, together with the current source MN7 produces a discharging current to increase the phase delay of R_Ck_{out}, i.e. to reduce the phase difference between R_Ck_{ref} and R_Ck_{out} from greater than half the clock period to less than half the clock period. Once the phase difference between R_Ck_{ref} and R_Ck_{out} becomes less than half the clock period, the MOVE signal is deasserted and the other two AND gates take over. Owing to the above mechanism, the DLL achieves lock as long as $D_{VCDL}(\min)$ is less than one clock period T_{REF}, which is significantly wider than existing designs.

The second functional block is the current switching block. The charging and discharging currents are supplied by two identical current sources with the same PMOS loads. Two pull-up transistors, MP2 and MP3, are added to further enhance the switching speed. The current switching happens when charging immediately follows discharging, or vice versa. For example, in the case that the discharging process happens immediately after the charging process, the current source MN7 is turned on to supply the discharging current I_{down}. At the same time, both MP1 and MP2 will be turned on since they have the same source-gate voltage. Transistor MP1 is used to conduct the discharging current I_{down}, while MP2 is used to charge the gate of MP4 so that MP4 can be turned off very quickly. In this way, the transition from the charging process to the discharging process is reduced. A similar scenario happens when charging immediately follows discharging.

The third functional block is the current mirror block. High-precision current mirrors are used so that the charging current I_{up} and the discharging current I_{down} can be matched very well. Transistors MN13 and MP9 are used to enhance the matching accuracy and to boost the output impedance. Specifically, the introduction of current branch MN13 guarantees that the drain-source voltages of MN9 and MN10 are exactly the same. Similarly, the introduction of current branch MP9 guarantees that the drain-source voltages of MP7 and MP4 are exactly the same. Therefore, MN9 and MN10 have the same gate-source voltage as well as the same drain-source voltage. Consequently, the currents flowing through MN9 and MN10 are exactly equal to each other. Similarly, the currents flowing through MP7 and MP4 are also the same.

The combined phase detector and charge pump circuit was simulated with 0.18 μm TSMC process models in HSPICE. Figure 25 illustrates the charge pump output voltage in a simulated pump-up process, assuming a fixed phase error.

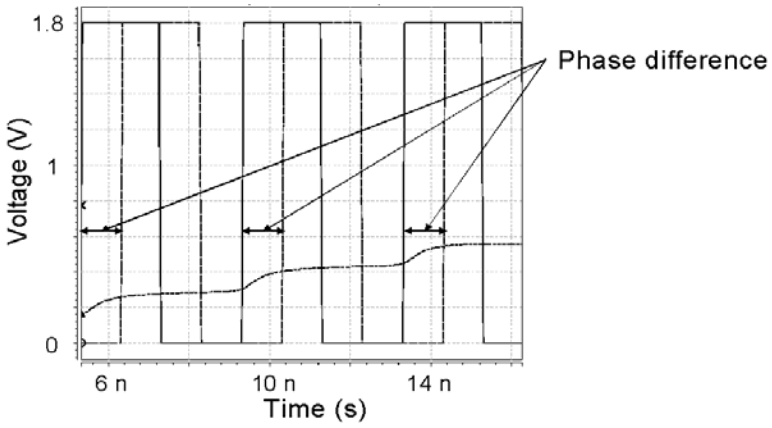
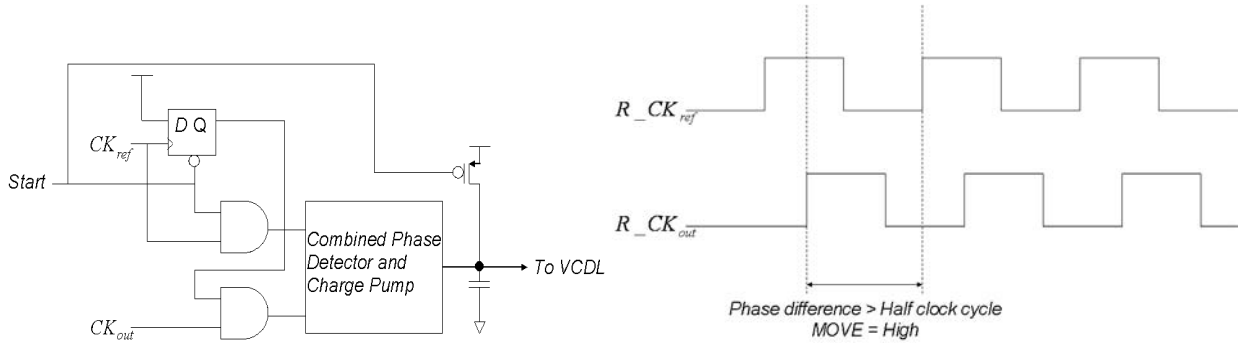


Fig. 25. A simulated pump-up process.

B.2. Start-Control Circuit

The operation of the DLL clock generator is started by the assertion of a START signal. Before the DLL begins the locking process, the START signal is low and the initial voltage of the loop filter is set to a maximum. Consequently, the initial VCDL delay is set to a minimum. After the DLL starts the locking process, the delay of the VCDL gradually increases until it becomes equal to one clock period of the input reference signal. As discussed before, setting the VCDL delay to a minimum in the beginning helps the DLL avoid the false locking problem and extends the lock range. The other purpose of the start-control circuit is to preprocess the Ck_{ref} and Ck_{out} waveforms for the subsequent phase detector. The output signals, R_Ck_{ref} and R_Ck_{out}, from the start-control circuit become inputs to the succeeding PD+CP. The phase difference between Ck_{ref} and Ck_{out} is the same as the phase difference between R_Ck_{ref} and R_Ck_{out}. The start-control circuit is shown in Figure 26.

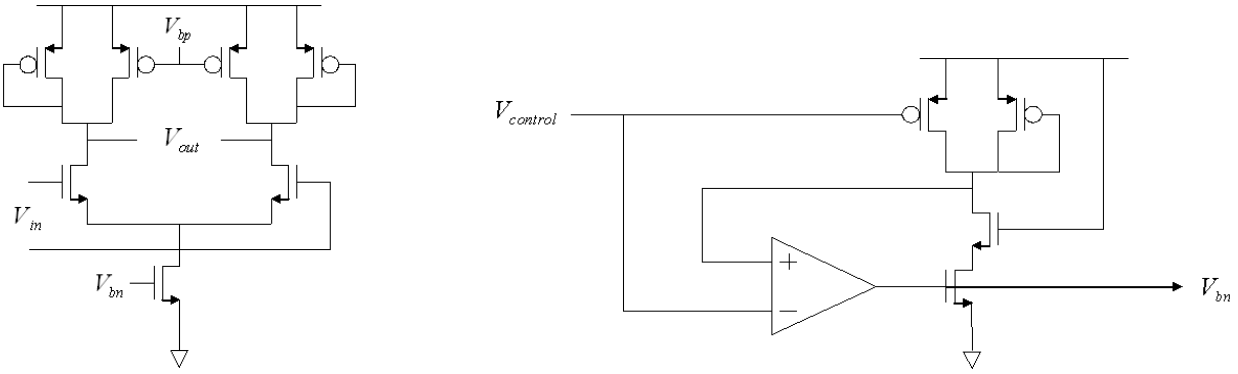


(a) **(b)**

Fig. 26. Start-control circuit: (a) circuit design and (b) waveforms.

B.3. Voltage-Controlled Delay Line

In order to minimize the sensitivity to supply and substrate noise and to achieve a wide tuning range, the delay stage developed in [45] is used in the DLL. The delay stage is built with a differential topology using symmetrical loads and replica-feedback biasing [45]. Figure 27 illustrates the VCDL delay stage.



(a) **(b)**

Fig. 27. VCDL delay stage: (a) delay stage and (b) replica-feedback biasing for the VCDL.

Simulation shows that for an 8-stage VCDL, $D_{VCDL}(\min)$ is approximately 180ps. Therefore, the DLL is able to operate as long as the operating frequency is less than 700MHz for nominal device characteristics, supply voltage, and room temperature. However, the output signal is distorted when V_{bp} is close to V_{dd}. Since V_{bp} is generated from the CP control voltage, $V_{control}$, the usable range of the control voltage, $V_{control}$ is limited, which effectively limits the operating frequency range in practice. Therefore, due to the output swing limitation, the actual operating frequency range is approximately 160MHz to 700MHz for nominal device characteristics, supply voltage, and room temperature.

In the DLL clock generator, the correctness of the generated setup clock and hold clock signals relies on the matching between the delay stages. In order to improve matching, a shift-averaging VCDL [48] is used in the design. The shift-averaging technique equalizes the delay of each delay stage as well as improves the duty cycle of the generated clock signals [48]. This technique requires the VCDL to have even number of delay stages. The shift-averaging VCDL is shown in Figure 28.

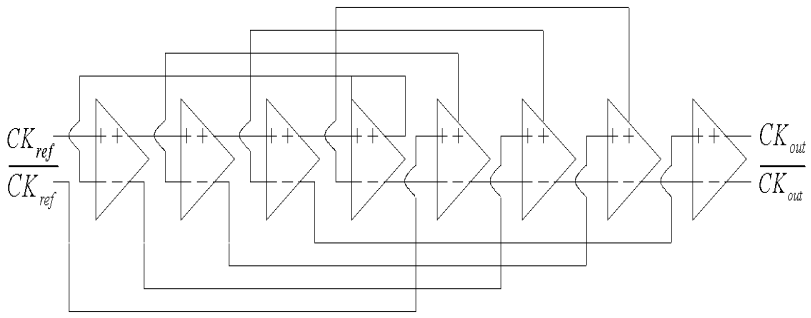


Fig. 28. Eight-stage shift averaging DLL.

A design trade-off exists in the number of delay stages in the VCDL. More delay stages can enhance the phase resolution of the VCDL output signals. On the other hand, fewer delay stages may boost the high end of the DLL operating frequency range. Eight stages are used in our DLL clock generator.

Since the DLL can only generate clocks from the outputs of a finite number of delay stages, the number of available phase shifts is limited by the number of delay stages in the VCDL. As a result, we are limited to the generation of delays that are multiples of the phase delay of a single delay cell. In other words, the target specifications influence the choice of the number of stages used in the VCDL. For example, if the operating frequency is 250MHz, the delays that can be generated are 0.5ns, 1.0ns, 1.5ns, 2.0ns, 2.5ns, 3.0ns, and 3.5ns.

C. Circuit Performance under Process, Supply Voltage and Temperature Variations

Extensive simulations have been done to verify the performance of the circuit under process, supply voltage, and temperature variations. Table V summarizes the performance of the circuit. Process variations are assumed to be limited to variations in device performance, modeled as fast, typical, and slow n-channel and p-channel transistors. The standard commercial temperature range of 0^{o} C to 70^{o} C was considered, together with supply voltage variations from 1.53V to 2.07V. In our analysis of process corners, we considered four circuit performances: jitter, static phase error, lock range, and lock time.

TABLE V
DLL Performance Summary

Technology	TSMC 0.18um 1P6M CMOS
Power Supply	1.8V
Active Die Area	$0.06\,mm^2$
Worst Case Operating Frequency	180MHz – 610MHz
Charge Pump Current	10uA
Loop Bandwidth	0.07* ω_{REF}
Worst Case Lock-In Time	13 cycles @ 250MHz

C.1.. Jitter

Jitter refers to the random variations in the period of the output clock signal. Assuming that t_n is the time point when the *n*th minus-to-plus zero crossing happens, the *n*th clock cycle period is then $T_n = t_{n+1} - t_n$. Ideally, the clock signal has a clock period of $\overline{T}$. The difference ΔT_n between T_n and $\overline{T}$ ($\Delta T_n = T_n - \overline{T}$) is an indicator of jitter [49]. Absolute jitter $\Delta T_{abs}(N)$ represents the accumulated jitter in the first *N* clock cycles. Cycle jitter is the long term RMS value of ΔT_n:

$$\Delta T_c = \lim_{N\to\infty} \sqrt{\frac{1}{N}\sum_{n=1}^{N} \Delta T_n^2}\ .$$

Cycle jitter represents the long term average effect of clock cycle fluctuation. Cycle-to-cycle jitter is the RMS difference between two consecutive clock cycles:

$$\Delta T_{cc} = \lim_{N\to\infty} \sqrt{\frac{1}{N}\sum_{n=1}^{N} (T_{n+1} - T_n)^2}$$

All of the above three parameters describe jitter characteristics. In general, absolute jitter is used to describe a PLL as there is significant jitter accumulation in a PLL. For other timing circuits, including DLLs, the other two parameters are usually used [49]. In this work, we use cycle-to-cycle jitter to quantify jitter.

The jitter of a DLL's output signal has several sources. They include jitter of the input reference signal, jitter contributed by the VCDL, and jitter due to switching noise on power supplies and ground bounce [50]. Ideally, because of our application, where we would like to capture any delays due to noise in the core clock, we would like to transfer all noise from the input reference clock to the output. However, as mentioned earlier, this is not possible, since the DLL performs a low pass filter function and stability requirements dictate the bandwidth. However, as it is important that the DLL does not introduce additional jitter, the focus of our analysis is on the jitter caused by the delay stages of the VCDL.

For active MOSFET transistors, device noise mainly consists of thermal noise and flicker noise. As proposed in [51], to simulate device noise, an equivalent white noise source and an equivalent flicker noise source is associated with each transistor. Equivalent noise voltages are converted into equivalent noise currents and injected into the signal path. Transcient analysis is performed to find the zero crossing points, from which cycle-to-cycle jitter. The results over process, supply voltage, and temperature are shown in Table VI.

TABLE VI

Cycle-to-Cycle Jitter of the DLL Output Signal @ 250MHz (in ps)

	Room Temperature and Vsupply=1.8V	**Temperature= 0^o C and Vsupply=2.07V**	**Temperature= 0^o C and Vsupply=1.53V**	**Temperature= 70^o C and Vsupply=2.07V**	**Temperature= 70^o C and Vsupply=1.53V**
TT	25.2	13.6	14.5	53.3	55.2
FF	32.8	15.8	15.6	56.4	57.4
SS	38.2	16.7	16.9	58.1	57.8
FS	37.0	16.0	16.9	56.5	57.1
SF	36.3	16.2	16.5	56.6	57.5

Process sensitivities can be best visualized by fitting a statistical model [52]. The model data vs. temperature and supply are shown in Figure 29. It can be seen that jitter increases as a function of temperature. This is because for the TSMC 0.18um process, thermal noise dominates flicker noise when the operating frequency is higher than 10MHz, and the spectral density of thermal noise increases linearly with temperature.

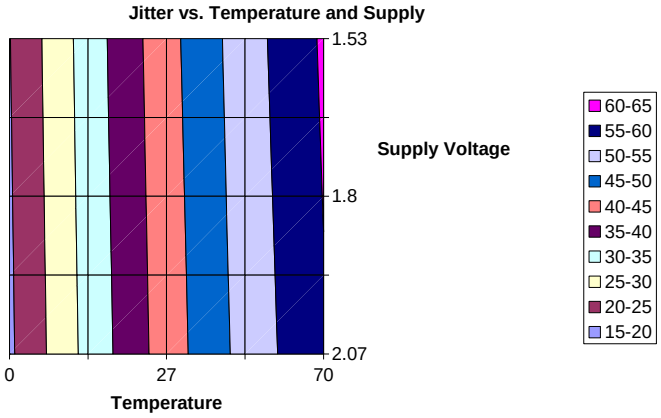


Fig. 29. Cycle-to-cycle jitter vs. supply and temperature for typical transistors.

C.2. Static Phase Error

Static phase error refers to the phase difference between the output signal of the last stage of the VCDL and the input reference signal. In the ideal case, after the DLL is locked, the phases of these two signals should be perfectly matched. However, due to the limited phase resolution of the PD+CP, some static phase error exists. Table VII shows the static phase error of the DLL as a function of device speed, power supply voltage, and temperature.

The results can be visualized in Figure 30. Sensitivities to device speed, power supply voltage, and temperature were found by analyzing the data, fitting a model, and checking for significance of model coefficients. It can be seen from the figure that static phase error increases for slow n and p-channel devices and is very sensitive to device speed. Static phase error increases with temperature, as well. Static phase error becomes larger for slower devices because slow switching in the PD and CP circuit may lead to the voltage of the loop filter (capacitor) to be in an equilibrium state when phase alignment has not been achieved. High temperature also contributes to large static phase error because the transistor mobility is inversely proportional to temperature. Low mobility causes transistors to switch slowly, similar to the behavior of slow devices.

TABLE VII
Static Phase Error @ 250MHz (in ps)

	Room Temperature and Vsupply=1.8V	Temperature= 0^o C and Vsupply=2.07V	Temperature= 0^o C and Vsupply=1.53V	Temperature= 70^o C and Vsupply=2.07V	Temperature= 70^o C and Vsupply=1.53V
TT	80	70	80	80	80
FF	60	60	60	70	70
SS	110	100	100	110	110
FS	80	70	80	90	90
SF	70	70	70	80	90

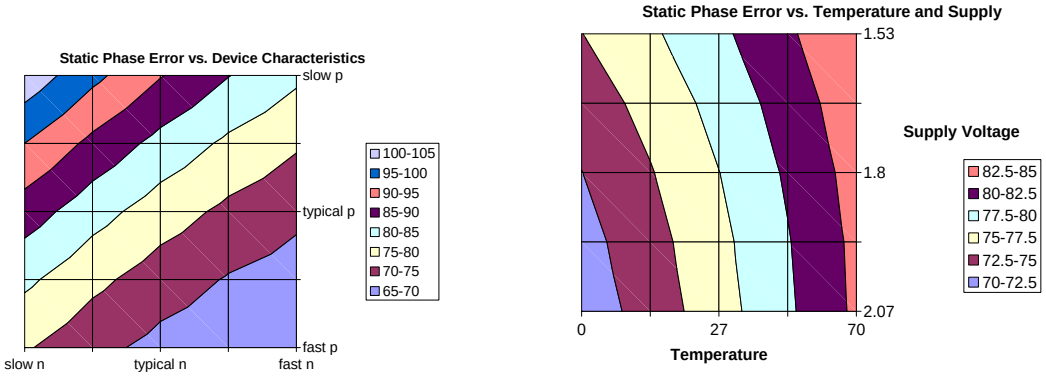


Fig. 30. Static phase error vs. device characteristics and temperature and supply voltage.

C.3. Lock Range

Lock range refers to the frequency range in which a DLL is able to achieve lock. The lock range for our DLL is shown in Table VIII as a function of device speed, power supply, and temperature.

TABLE VIII
Lock Range (in MHz)

	Room Temperature and Vsupply=1.8V	Temperature= 0^o C and Vsupply=2.07V	Temperature= 0^o C and Vsupply=1.53V	Temperature= 70^o C and Vsupply=2.07V	Temperature= 70^o C and Vsupply=1.53V
TT	160-700	170-810	150-610	180-850	150-630
FF	170-700	180-820	150-620	180-850	160-650
SS	160-680	170-800	150-610	170-840	140-610
FS	160-690	170-810	150-610	170-840	150-620
SF	160-680	180-810	150-610	170-840	150-620

The lower and upper lock range limits are limited for different reasons. Lower lock range has a small sensitivity to the power supply voltage, since the supply voltage limits the usable range of the CP control voltage. However, this sensitivity does not exceed the noise level in the data.

On the other hand, the upper limit of the lock range is sensitive to device speed, power supply voltage, and temperature are shown in Figure 31. The slow NMOS and slow PMOS corner, combined with the lower value of supply voltage, provides the lower limit on the lock range, since slow transistors and a lower supply voltage result in a longer minimum VCDL delay. Specifically, slower transistors and lower power supply values results in slower charging/discharging currents of the VCDL.

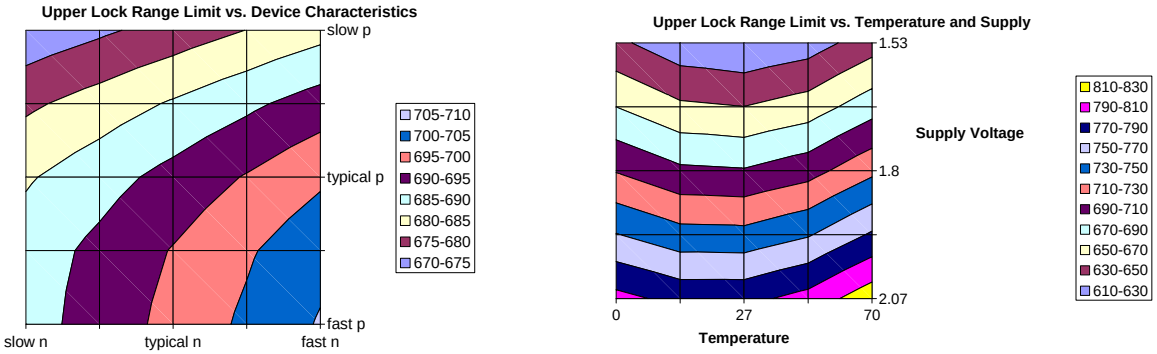


Fig. 31. Upper lock range limit vs. device characteristics and temperature and supply voltage.

C.4. Lock Time

Lock time refers to the time interval that the DLL takes to achieve lock. Table IX lists the lock times under different temperature, supply, and process variations. Fitting the data with a statistical model revealed no significant sensitivities. The average lock time is 12.4 clock cycles and the worst case simulated value is 13 clock cycles.

TABLE IX
Lock Time at Lowest Operating Frequency (in cycles)

	Room Temperature and Vsupply=1.8V	**Temperature= 0^o C and Vsupply=2.07V**	**Temperature= 0^o C and Vsupply=1.53V**	**Temperature= 70^o C and Vsupply=2.07V**	**Temperature= 70^o C and Vsupply=1.53V**
TT	11	12	12	13	12
FF	12	13	13	12	12
SS	12	13	13	13	13
FS	11	12	13	13	12
SF	12	12	13	13	12

C.5. Impact on Test Accuracy

Two parameters that directly impact measurement accuracy are static phase error and jitter. We use the statistical characteristics of these two parameters to find a 95% confidence bound on error. Specifically, the mean error is given by the static phase error, and the standard deviation is computed from jitter. The results are shown in Table X. These errors are upper bounds in measurement error if only a single measurement is used. If, however, multiple measurements are averaged, then static phase error, as shown in Table VII, defines the upper bound on measurement error. In this case, the nominal measurement error is 80ps, and the maximum over process, temperature, and supply variations is 110ps.

TABLE X
95% Confidence Bound on Measurement Error (in ps)

	Room Temperature and Vsupply=1.8 V	**Temperature= 0^o C and Vsupply=2.07V**	**Temperature= 0^o C and Vsupply=1.53V**	**Temperature= 70^o C and Vsupply=2.07V**	**Temperature= 70^o C and Vsupply=1.53V**
TT	115	89	100	154	157
FF	105	82	82	148	150
SS	163	123	123	191	190
FS	131	92	103	168	169
SF	120	92	93	158	170

D. Implementation

The BIST circuit has been implemented and fabricated with a TSMC 0.18um one-poly six-metal CMOS process (CL018). The full chip layout is shown in Figure 32. The layout focused on three issues that are critical for analog designs: (a) matching, (b) substrate and power supply noise, and (c) minimizing crosstalk.

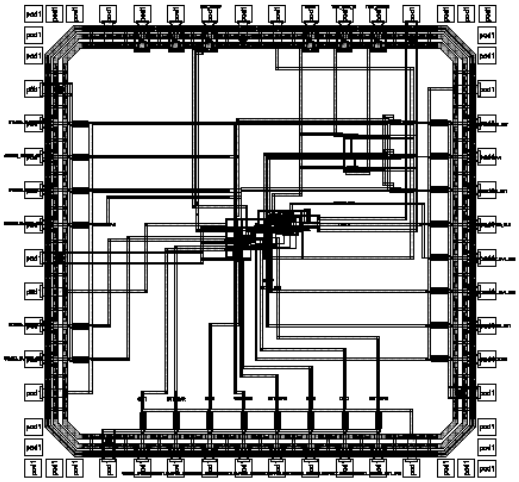


Fig. 32. Layout of the chip.

VII. Conclusions

We have presented a methodology to diagnose the physical origin of path delay faults caused by known imperfections in optical lithography. Our methodology involves layout-dependent timing analysis, taking into account deterministic within-die variation (gate length variation as a function of the local neighborhood, location in the reticle, and pattern density). Critical paths are extracted by pruned DFS and used to establish test patterns and pass/fail patterns associated with each fault. The results are stored in a dictionary. Observed faults are matched with simulated pass/fail patterns to diagnose the physical origin of within-die variation and to help us properly allocate mask correction effort.

Implementation of the delay test methodology requires analysis of methods to input two pattern sequences into the scan chain. Two approaches have been discussed: launch-off-shift and broad-side. Both limit the set of possible two pattern sequences that can be applied. In addition, delay test implementation is limited by the clocking methodology. At-speed test with an on-chip phase lock loop is one approach, but is limited by power dissipation. The alternative is to generate two clocks that are separated by a fixed phase shift. Our approach to the design of such clocks, using a delay-locked loop, which are invariant to temperature, supply voltage, and process, has been discussed.

Acknowledgment

The authors thank the Semiconductor Research Corporation for their financial support, under Task 1176.001.

References

[1] S.R. Nassif, A.J. Strojwas, and S.W. Director, "A methodology for worst-case analysis of integrated circuits," *IEEE Trans. Computer-Aided Design,* vol. 5, no. 1., pp. 104-113, Jan. 1986.

[2] H. J. Levinson, *Principles of Lithography*. SPIE PRESS, 2001.

[3] T.A. Brunner, "Impact of lens aberrations on optical lithography," *IBM J. Research and Development,* vol. 41, pp. 57-67, Jan./March 1997.

[4] C. A. Mack, "Measuring and Modeling Flare in Optical Lithography," *Proc. SPIE,* vol. 5040, pp. 151-161, 2003.

[5] M. Orshansky, L. Milor, and C. Hu, "Characterization of spatial-intra-field gate CD variability, its impact on circuit performance, and spatial mask-level correction," *IEEE Trans. Semi. Manufacturing,* vol. 17, no. 1, pp. 2-11, Feb. 2004.

[6] G. L. Smith, "Modeling for delay faults based upon paths," *Proc. Int. Test Conf.,* 1985, pp. 342-349.

[7] W. Ke and P.R. Menon, "Synthesis of delay-verifiable combinational circuits," *IEEE Trans. Computers,* vol. 44, no. 2, pp. 213-222, Feb. 1995.

[8] R.C. Tekumalla and P.R. Menon, "Identification of primitive faults in combinational and sequential circuits," *IEEE Trans. Computer-Aided Design,* vol. 20, pp. 1426-1442, Dec. 2001.

[9] M. Sivaraman and A.J. Strojwas, "Primitive path delay faults: Identification and their use in timing analysis," *IEEE Trans. Computer-Aided Design,* vol. 19, pp. 1347-1362, Nov. 2000.

[10] A. Krstic, K.-T. Cheng, and S.T. Chakradhar, "Primitive delay faults: Identification, testing, and design for testability," *IEEE Trans. Computer-Aided Design,* vol. 18, pp. 669-684, June 1999.

[11] M. Sharma and J.H. Patel, "Testing of critical paths for delay faults," *Proc. Int. Test Conf.,* 2001, pp. 634-641.

[12] M. Sharma and J. H. Patel, "Finding a small set of longest testable paths that cover every gate" *Proc. Int. Test Conf.*, 2002, pp. 974-982.

[13] W. Qiu and D.M.H. Walker, "An efficient algorithm for finding the K longest testable paths through each gate in a combination circuit," *Proc. Int. Test Conf.*, 2003, pp. 592-601.
[14] S.H.C. Yen, D.H.C. Du, and S. Ghanta, "Efficient algorithms for extracting the K most critical paths in timing analysis," *Proc. Design Automation Conf.,* 1989, pp. 649-654.
[15] S. Kundu, "An incremental algorithm for identification of longest (shortest) paths," *Integration the VLSI Journal*, pp. 25-31, 1994.
[16] L.-C. Wang, J.-J. Liou, and K.T. Cheng, "Critical path selection for delay fault testing based upon a statistical timing model," *IEEE Trans. Computer-Aided Design,* vol. 23, pp. 1550-1565, Nov. 2004.
[17] C. Visweswariah, "Death, taxes and failing chips," *Proc. Design Automation Conf.,* 2003, pp. 343-347.
[18] S.R. Nassif, D. Boning, and N. Hakim, "The care and feeding of your statistical static timer," *Proc. Int. Conf. on Computer-Aided Design,* 2004, pp. 138-139.
[19] M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design,* IEEE Press, 1990, ch. 12.
[20] X. Wen *et al.,* "On per-test fault diagnosis using X-fault model," *Proc. Int. Conf. on Computer-Aided Design,* 2004, pp. 633-640.
[21] P. Girard, C. Landrault, and S. Pravossoudovitch, "A novel approach to delay-fault diagnosis," *Proc. Design Automation Conf.,* 1992, pp. 357-360.
[22] Z. Wang, *et al.,* "Delay-fault diagnosis using timing information," *IEEE trans. Computer-Aided Design,* vol. 24, pp. 1315-1325, Sept. 2005.
[23] A. Krstic *et al.,* "Delay defect diagnosis based upon a statistical timing model – the first step," *IEE Proc. – Computers and Digital Techniques,* vol. 150, pp. 346-354, Sept. 2003.
[24] P. Pant, *et al.*, "Path delay fault diagnosis in combinational circuits with implicit fault enumeration," *IEEE Trans. Computer-Aided Design,* vol. 20, pp. 1226-1235, Oct. 2001.
[25] S. Padmanaban and S. Tragoudas, "An implicit path-delay fault diagnosis methodology," *IEEE Trans. Computer-Aided Design,* vol. 22, pp. 1399-1408, Mar. 2003.
[26] M. Sivaraman and A. J. Strojwas, "Path delay fault diagnosis and coverage-A metric and an estimation technique," *IEEE Trans. Computer-Aided Design,* vol. 20, pp. 440-457, Mar. 2001.
[27] A. Krstic *et al.,* "Diagnosis-based post-silicon timing validation suing statistical tools and methodologies," *Proc. Int. Test Conf.,* 1999, pp. 558-567.
[28] W.-Y. Chen, S.K. Gupta, and M.A. Breuer, "Test generation for crosstalk-induced delay in integrated circuits," *Proc. Int. Test Conf.,* 1999, pp. 191-200.
[29] A. Krstic, *et al.,* "Delay testing considering crosstalk-induced effects," *Proc. Int. Test Conf.,* 2001, pp. 558-567.
[30] J.-J. Liou, *et al.*, "Modeling, testing, and analysis for delay defects and noise effects in deep submicron devices," *IEEE Trans. Computer-Aided Design,* vol. 22, pp. 756-769, Jun. 2003.
[31] A. Krstic, Y.-M. Jiang, and K.-T. Cheng, "Pattern generation for delay testing and dynamic timing analysis considering power-supply noise effects," *IEEE Trans. Computer-Aided Design,* vol. 20, pp. 416-425, Mar. 2001.
[32] M. Choi and L. Milor, "The impact on circuit performance of deterministic within-die variation in nanoscale semiconductor manufacturing," *IEEE Trans. Computer-Aided Design,* vol. 25, pp. 1350-1367, July 2006.
[33] A. Chatzigeorgiou, S. Nikolaidis, and I. Tsoukalas, "A modeling technique for CMOS gates," *IEEE Trans. Computer-Aided Design,* vol. 18, pp. 557-575, May 1999.
[34] Y.-H. Shih, Y. Leblebici, and S. M. Kang, "ILLIADS: A fast timing and reliability simulator for digital MOS circuits," *IEEE Trans. Computer-Aided Design,* vol. 12, pp. 1387-1402, Sep. 1993.
[35] Star-Hspice Manual, Avant!, 1998.
[36] Design Compiler Reference Manual, Synopsys, 2000.
[37] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinatorial benchmark circuits," *Proc. Int. Symp. Circuits and Systems*, 1985, pp. 695–698.
[38] D. Lee, V. Zolotov, and D. Blaauw, "Static timing analysis using backward signal propagation," *Proc. Design Automation Conf.,* 2004, pp. 664-669.
[39] W.-Y. Chen, S.K. Guota, and M.A.Breuer, "Analytical models for crosstalk excitation and propagation in VLSI circuits," *IEEE Trans. Computer-Aided Design,* vol. 21, pp. 1117-1131, Oct. 2002.
[40] S.-Z. Sun, D.H.C. Du, and H.-C. Chen, "Efficient timing analysis for CMOS circuits considering data dependent delays," *IEEE Trans. Computer-Aided Design,* vol. 17, pp. 546-552, June 1998.
[41] A. Pierzynska and S. Pilarski, "Pitfalls in delay fault testing," *IEEE Trans. Computer-Aided Design,* vol. 16, pp. 321-329, March 1997.
[42] C.T. Gray, *et al.,* "Circuit delay calculation considering data dependent delays," *Integration, the VLSI Journal,* pp. 1-23, 1994.
[43] 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, pp. 1323-1335, Oct. 1991.
[44] R. Farjad-Rad *et al*., "A low-power multiplying DLL for low-jitter multigigahertz clock generation in highly integrated digital chips," *IEEE J. Solid-State Circuits,* vol. 37, pp. 1804-1812, Dec. 2002.

[45] J. Maneatis, "Low-jitter process-independent DLL and PLL based on self-biased techniques," *IEEE J. Solid-State Circuits,* vol. 31, pp. 1723-1732, Nov. 1996.
[46] W. Rhee, "Design of high performance CMOS charge pumps in phase-locked loops", *Proc. Intl. Symp. on Circuits and Systems,* vol. 1, 1999, pp. 545-548.
[47] *High Speed Design Techniques*, Analog Devices Technical Reference Books, Analog Devices, Inc., 1996.
[48] H-H. Chang, C-H. Sun, and S-I. Liu, "A low-jitter and precise multiphase delay-locked loop using shifted averaging VCDL," *Proc. Int. Solid-State Circuits Conf.*, Feb. 2003, pp. 177-180.
[49] M. Mansuri and C-K. K. Yang, "Jitter optimization based on phase-locked loop design parameters," *IEEE J. Solid-State Circuits,* vol. 37, pp. 1375-1382, Nov. 2002.
[50] G. Chien and P.R. Gray, "A 900-MHz local oscillator using a DLL-based frequency multiplier technique for PCS applications," *IEEE J. Solid-State Circuits,* vol. 35, pp. 1996-1999, Dec. 2000.
[51] C.W. Zhang, X.Y. Wang, and L. Forbes, "Simulation techniques for noise and timing jitter in electronic oscillators," *IEE Proc. Circuits Devices Syst.,* vol. 151, no. 2, pp. 184-189, April 2004.
[52] B.E.P. Box, W.G. Hunter, and J.S. Hunter, *Statistics for Experimenters,* John Wiley & Sons, New York, 1978.