Back-End Layout Reflection for Test Chip Design

Zeye Liu
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213, USA
Email: zeyel@andrew.cmu.edu

R. D. (Shawn) Blanton
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213, USA
Email: rblanton@andrew.cmu.edu

Abstract—At advanced technology nodes, complex interactions between layout features and the process can lead to manufacturability issues that reduce yield. Due to the huge number of layout geometries inherent to random logic, logic-only test chips are increasingly employed during yield ramp. This work describes a design methodology that incorporates complex layout geometries into an optimally testable full-flow logic test chip. Experiments comparing properties among test-chip, benchmark, and actual product designs demonstrate the efficacy of the methodology. Specifically, our test chips achieve 100% coverage for various fault models, and on average, incorporate the layout geometries of interest while being 96% less layout area and 63% less wire length compared to various benchmark and product designs.

I. INTRODUCTION

The relentless drive to reduce the size of electronic components has made the semiconductor industry extremely capital intensive [1]. The increase in cost of a new integrated circuit technology is no longer considered extraordinary. In this context, yield is one of the most important metrics for measuring manufacturing performance. In recent years, foundries have allocated 15% or more of total capital expenditures to inspection and review equipment for process control and yield improvement [2]. In addition, the time to increase and stabilize yield become much longer than the past. However, the demands of aggressive time-to-market competition always exist. Thus, there can be substantial economic benefits for quickly enabling higher yields.

Yield losses in semiconductor manufacturing arise from systematic as well as random defects. Random defects are due to contaminant particles, which can be characterized by probability distributions, and can be mitigated through changes made to the process or fabrication equipment [3]. Systematic defects, on the other hand, occur due to manufacturing process, design properties or a combination thereof. As the IC manufacturing process becomes more and more complex, the interaction between design and process becomes less predictable. As a result, certain layout geometries are more sensitive to the non-ideal in the fabrication process than others and thus have an increase likelihood of failure [4]; we refer to such geometries as yield-limiting layout geometries. Unlike random defects, systematic defects can lead to repeated IC failures wherever there are similar layout geometries. Thus, investigating layout geometries not only helps the yield ramp through mitigation of systematic defects, but also improves the manufacturability by enhancing process weak point implied by the yield-limiting layout geometries.

A significant amount of prior work has been devoted to identifying the yield-limiting layout geometries. The conventional method uses full-chip lithography simulation, which has high accuracy but suffers from significant runtime. Particularly, the runtime for lithography simulation on a 5 × 5 mm design at a 32nm technology node can take months to complete [5]. To quickly recognize the yield-limiting layout geometries, two major methodologies exist: pattern matching [6] and machine learning [7][8]. Pattern matching is a direct and fast method but has a significant error rate for layout geometries not characterized a priori. Machine learning techniques attempt to learn hidden relations between layout geometries and their defect characteristics. Although machine learning can improve accuracy as compared to pattern matching method, it too suffers from the false alarm problem that recognizes the manufacturable geometries as yield-limiting layout geometries. However, detection accuracy for yield-limiting layout geometries is highly correlated with the maturity of a manufacturing process, due to the need for a sufficiently large sample of silicon-validated failure data. Therefore, both pattern matching and machine learning alone are insufficient for quickly improving yield of product designs fabricated at the early stages of a new manufacturing process, especially when fast yield ramp leads to the maximal profit.

Conventionally, many types of test chips are used to accelerate the yield ramp during the early development stages of a new manufacturing process, ranging from SRAM, mixed-signal/analog circuit to logic-only test chips. Particularly, logic-only test chips that capture the layout features of random logic are employed to investigate the vast number of the complex layout geometries that result from place-and-route. Logic test chips are a necessity since SRAM uses few geometries which are characterized extensively because of their tremendous repeated use and mixed-signal/analog circuits have geometries that are well controlled due to their custom or semi-custom design methodologies.

Two types of logic-only test chips (short-flow and full-flow test chips) are employed for fast yield ramp. Specifically, short-flow test chips are manufactured using only a subset of the process layers, and are targeted towards either the front end of the line (FEOL) (i.e., transistor and contact layers), or the back end of the line (BEOL) (i.e., metal interconnect). Short-flow test chips are deliberately designed to be simple and uniform so that they can provide timely feedback. Thus, they are unsuitable for monitoring design-process interactions in the presence of complex layout geometries, especially those
involving multiple layers. Work in [9] describes a design methodology to create a short-flow test chip that contains complex BEOL layout geometries. However, short-flow test chips have failed to detect some defects expected to be uncovered, likely due to the missing properties resulting from the accumulated impact of many complex fabrication steps used in the full-flow process. Due to this limitation of the short-flow test chip, full-flow logic test chips are used for yield ramp as well. Full-flow logic test chips are manufactured using all of the process layers, including standard cells and BEOL metal interconnect. Work in [10] describes a full-flow logic test chip consisting of an array of flip-flops with SRAM-like connections (bit lines, word lines, etc.). The authors argue that the flip-flop array can be made to contain characteristics of a product layout through synthesis with a standard-cell library, and the use of conventional place-and-route. But, it remains an open question as to whether the physical similarities between this test chip and product design layouts is sufficient to ensure that the same defect mechanisms are common to both.

The most common type of full-flow logic test chip employed in industry includes sub-circuits (e.g. a floating-point unit) from existing product designs. While the sub-circuits of existing product designs contain actual design features (e.g. use of standard-cell and complex layout geometries), the primary drawback is its transparency to a large universe of failures. A test chip is transparent to failure if it is easily testable and diagnosable for both expected and unexpected defects. To address this shortcoming, this paper proposes a design methodology that incorporates the BEOL layout geometries of interest while ensuring transparency and FEOL layout incorporation [11-16].

The rest of the paper is organized as follows. Section II provides the relevant background and describes the contribution of this work. Section III proposes a design methodology to obtain a logic test chip with various layout geometries of interest while ensuring failure transparency. Experiment results assessing and validating the methodology are presented in Section IV. The final section concludes the paper and describes some directions for future work.

II. RELATED WORK

Due to the limitations of the existing full-flow logic test chips described in the last section, a new approach, called Carnegie-Mellon Logic Characterization Vehicle (CM-LCV), has been introduced [11-16]. The CM-LCV is designed for maximal testability and diagnosability while being sensitive to the defect mechanisms that affect product designs. It is based on the insight that the manufacturing process is sensitive only to the physical features of a design (i.e., its physical layout), and not the logic functionality the layout implements. Thus, the CM-LCV work can be summarized as a highly transparent functionality coupled with a flow for physical implementation that incorporates the layout features of interest. Particularly, work in [11] [12] demonstrated that a two-dimensional array of functional unit blocks (FUBs) that implements a particular

Figure 1. Layout geometry incorporation includes three steps: (a) layout geometry characterization, (b) logic implementation and (c) physical implementation. Rectangle box represents each sub-step, while rounded-corner box indicates an input/output file of the next/last sub-step.
in the library that has nets that are compatible with nets of layout geometries of interest is identified. That a net within a given FUB is compatible with a net in the layout geometry if the former has at least the same number of CPs as the latter. In addition, the testability of each FUB is also determined. From this analysis, a set of FUBs is then selected to form a two-dimensional array (a FUB array) such that each layout geometry can be incorporated into a FUB within the array. The third step is physical implementation. For this step, each FUB within the array is analyzed to identify which nets should be selected to connect the CPs of layout geometry with the aim of minimizing the routing complexity. Commercial place-and-route is then employed to create the layout that incorporates the BEOL layout geometries of interest. Each sub-step is further described in the following sections.

B. Layout geometry characterization

Many approaches can be used to identify BEOL layout geometries of interest. For example, fabless and foundries can specify the BEOL layout geometries that would aid yield ramp. To be more specific, a foundry can specify the layout geometries with complex configuration to examine the process capability [9][17]; a fabless company, on the other hand, can identify the layout geometries that are representative of a particular design or family of designs that will be fabricated in the future. Regardless of which approaches are employed, a set of layout geometries of interest is derived for input to flow of Fig. 1. An example of a BEOL layout geometry is shown in Fig. 2a. A box defines a region for the layout geometry (with a black boundary). In this example, the geometry contains polygons in metal-3 layer (blue), two polygons in metal-2 layer (yellow), a VIA in the via-2 layer (brown) and space (white). The important goal of this work is to preserve the layout geometry of interest, including the polygon, via and space in a full-flow logic test chip, an overall process we term “layout geometry incorporation”.

Database preparation: This task employs topological patterns to describe the layout geometries of interest. A topological pattern consists of a bitmap and two vectors that compactly represent of a layout region (i.e., a region involving one or more layers that contains the features of interest, for example, a net and its surrounding nets). Fig. 2b shows a topological pattern for example layout geometries (Fig. 2a). Particularly, an entry is used to represent whether a location is space or polygon. This representation is extended to multiple layers by increasing the number of values for an entry in the bitmap. In this example, the presence of a polygon in the bottom metal layer, the via layer and the upper metal layer are captured by value ‘1’, ‘2’ and ‘4’ respectively. In addition, each entry in the bitmap also maps to a certain location in the layout. The height and width of each entry in the bitmap is summarized as the two vectors, one for the x dimension and one for the y dimension. In Fig. 2b, the first entry in the bitmap indicates a 190nm x 180nm space region in the upper-left corner of the layout geometry. Since the typical description of design layout is GDS, a commercial EDA tool [18] is used to convert GDS to the topological pattern format.

Net geometry processing: This task characterizes the layout geometry of interest as a vector that summarizes the number of CPs of each net within it. Particularly, Algorithm 1 describes a method that identifies the number of CPs of each net from the topological pattern of the layout geometry.

Algorithm 1 Net Geometry Processing

\textbf{Input:} Bitmap of a layout geometry ($B$), number of layers to be analyzed ($L$)

\textbf{Output:} Vector ($G$) of number of CP of all nets in ($B$)

\begin{align*}
H &\leftarrow \emptyset \quad Q \leftarrow \emptyset \\
& \text{for each entry } b \in B \text{ do} \\
& \quad \text{if } is\_polygon(b,l) \text{ then push } Q[l], b; \\
& \text{end} \\
& \text{end} \\
& \text{for } l = 1 \text{ to } L \text{ do} \\
& \quad \text{while } Q[l] \text{ is not empty do} \\
& \quad \quad q \leftarrow \text{pop } Q[l]; \quad j \leftarrow j + 1; \quad \text{push } H[j], (q,l) \\
& \quad \quad N_{\text{temp}} \leftarrow \text{neighbor}(q) \\
& \quad \quad \text{while } N_{\text{temp}} \text{ is not empty do} \\
& \quad \quad \quad (n,k) \leftarrow \text{pop } N_{\text{temp}}; \quad \text{push } H[j], (n,k) \\
& \quad \quad \quad \text{push } N_{\text{temp}}, \text{ neighbor}(n) \\
& \quad \quad \text{end} \\
& \quad \text{end} \\
& \quad G \leftarrow is\_boundary(H)
\end{align*}

The first step of Algorithm 1 identifies the location of each polygon per layer. Particularly, the function $is\_polygon$ checks whether the entry $b$ at layer $l$ contains a polygon. As a result, for each layer $l$, a list $Q[l]$ identifies which entries contain a polygon. The second step of the Algorithm 1 clusters the entries in a polygon list into sub-groups, where each sub-group describes the geometry of a net. It begins with an arbitrary entry stored in the polygon list $Q[l]$, where $l$ is initially set to the lowest layer. Since a net consists of connected polygons, the $neighbor$ function checks whether a neighboring entry (four in the same layer, and one in layer $l+1$) exists in the polygon lists. If a neighboring entry contains a polygon, an entry $b$ and the corresponding the layer $l$ are stored in a group $H[j]$. It indicates that $j$th net in the bitmap $B$ contains a polygon at location $b$ within layer $l$. Neighboring searching continues until no neighboring entry contains a
polygons. Then, Algorithm 1 continues to identify the j + 1th net that begins with an unvisited entry in the polygon list $Q$, until all entries in list $Q$ are examined. The final step of Algorithm 1 calculates the number of CPs of each net. The function $is\ boundary$ checks the number of polygon entries per sub-group in $H$ that located at the outermost columns and rows in $B$. For example, Algorithm 1 characterizes the layout geometry shown in Fig. 2a as a vector $G = [3, 2]$. The first element in $G$ indicates that the net labeled “P1” has three CPs, while the second element indicates the net labeled “P2” has two CPs.

**C. Logic implementation**

Given a set of the layout geometries of interest, which are characterized by the last step as $\mathcal{G} = \{G_1, G_2, \ldots , G_N\}$, where $N$ is total number of the layout geometries of interest, the second step of Fig. 1 identifies a set of FUBs that can incorporate $\mathcal{G}$. This step begins with a FUB library that has hundreds of thousands of different logic-level implementations of an information-lossless function. Then, a set of the FUB implementations from the FUB library is selected that altogether contain nets that are compatible with the layout geometries $\mathcal{G}$.

**FUB characterization**: Before a set of FUBs can be selected, each FUB in the library must be characterized, FUB characterization determines the number of CPs of each net within a FUB and its testability. As described above, a net with a FUB is compatible with a net in the layout geometry if the former has at least the same number of CPs as the latter. For a FUB net, the number of CPs is equal to the number cell I/Os connects. For example, the four-gate circuit shown in Fig. 3a contains a net labeled “n5” with three CPs and seven other nets with two CPs. It is identified to be able to incorporate the layout geometry shown in Fig. 3b. Specifically, net “n5” is compatible with net “P1” since both have three CPs, and all nets “n1-n8” are compatible with “P2” since they have at least two CPs. Thus, given a FUB library, a set of vectors is created, denoted as $\mathcal{F} = \{F_1, F_2, \ldots , F_M\}$, where $M$ is the total number of FUBs in the library. Similar to the set $\mathcal{G}$, each $F_i \in \mathcal{F}$ is a vector that captures the number of CPs of each net in the i-th FUB.

Because BEOL layout geometries are sensitive to different defect mechanisms, the testability of each FUB is evaluated utilizing ATPG/fault simulation tools to derive the coverage for each fault model. To ensure bridge fault coverage is independent of the physical implementation of a FUB, every net pair is assumed to be a fault site for the 4-way dominant fault model. Moreover, every single I/O path within a FUB is examined to evaluate its robust path delay test properties. The testability analysis of the FUB library results in a vector $R = [R_1, R_2, \ldots , R_M]$. Each $R_i$ corresponds to i-th FUB in the library and tabulates the number of redundant faults for all four fault models. For example, $R_1 = 0$ means the i-th FUB is 100% testable for all four fault models.

**Logic design synthesis**: The last task formulates an optimization problem for selecting FUBs for the FUB array. Particularly, the optimization is formulated to meet two objectives. One objective is to incorporate the layout geometries efficiently so that the overall array layout area is minimized, while the second objective is focused on maximizing testability. Eq. 1 captures these two objectives.

$$\begin{align*}
\text{minimize} & \quad pRx + Ax \\
\text{subject to} & \quad \sum_j Y_{ij} = 1, \quad 1 \leq i \leq N \\
& \quad C_j x_j - \sum_j Y_{ij} S_i \geq 0, \quad 1 \leq j \leq M
\end{align*}$$

In Eq. 1, $p$ is a scalar parameter for tuning the tradeoff between the testability and the layout area; $x$ is a vector that identifies the FUBs for forming the FUB array; and $Y$ is a binary matrix that relates a FUB and a layout geometry. For example, if $Y_{ij} = 1$, indicates that the i-th layout geometry $G_i \in \mathcal{G}$ is compatible with the j-th FUB $F_j \in \mathcal{F}$. Vector $A$ tracks the layout area for each FUB. Parameters $x$ and $Y$ are the quantities being optimized in Eq. 1. Therefore, the solution of Eq. 1 not only selects a set of testable FUBs but also assigns each layout geometry to a selected FUB.

$$a_u = [1(CP_u \geq 1), 1(CP_u \geq 2), \ldots , 1(CP_u \geq k)]' \quad (2)$$

In Eq. 1, the set $\mathcal{G}$ is characterized as a matrix $S$, each column vector $S_i$ tabulates the number of CPs of all nets in i-th layout geometry. Particularly, each $G_i \in \mathcal{G}$ is first extended as a matrix $\alpha = [\alpha_1, \alpha_2, \ldots , \alpha_v]$, where $v$ is the number of nets within $G_i$. Each column vector $\alpha_u \in \alpha$ is defined by Eq. 2. The notation “$1(CP_u \geq k)$” in Eq. 2 means that if the number of CPs of the i-th net in $G$ is not less than $k$, the $k$-th entry of vector $\alpha_u$ is equal to 1. Here, $k$ is the maximum number of CPs of all nets in both set $\mathcal{G}$ and $\mathcal{F}$. Once the matrix $\alpha$ is created, column vector $S_i$ computes the sum of the elements in each row of matrix $\alpha$. The same characterization is applied for the set $\mathcal{F}$, resulting in a matrix $C$. This characterization ensures the dimension consistency of matrix $C$ and $S$. For example, a vector $C_1 = [8, 8, 1]'$ and a vector $S_1 = [2, 2, 1]'$ characterize the number of CPs of the nets in the circuit shown in Fig. 3a; and the layout geometry shown in Fig. 3b, respectively. Since $C_1 \geq S_1$, it indicates the circuit shown in Fig. 3a can incorporate the layout geometry shown in Fig. 3b.

Furthermore, the first term in Eq. 1 $pRx$ drives the solver to maximize testability. The second term $Ax$ minimizes the
overall area of the selected FUBs. The first constraint guarantees every layout geometry is incorporated. The second constraint ensures the selected FUBs have sufficient nets for the geometries in $\mathcal{G}$. The solution $x$ of Eq. 1 identifies a set of FUBs for creating a FUB array. Specifically, a FUB array is constructed by replicating the FUB set as much as required, for the desired array size (e.g., a FUB set consisting of four FUBs would be replicated ten times to produce a 10x4 array, or a 5x8 array, etc.). It should be noted that the properties of the array are completely unaffected by its scaled-up size.

**D. Physical implementation**

The third step of Fig. 1 involves physical implementation of the BEOL layout geometries. Specifically, by solving Eq. 1, a set of FUB $\{F_1, F_2, \ldots, F_m\} \subset \mathcal{F}$ is identified to incorporate the layout geometries $\mathcal{G}$, where $m$ is number of FUB selected to construct the FUB array. For each selected FUB $F_j$, a set of layout geometries $\{G_{1j}, G_{2j}, \ldots, G_{nj}\} \subset \mathcal{G}$ is assigned to be incorporated in $F_j$, where $n$ is number of layout geometries assigned to $F_j$. The first task of physical implementation is to identify which nets in FUB $F_j$ will be used to construct a layout geometry $G_i \in \{G_{1j}, G_{2j}, \ldots, G_{nj}\}$. A second task is then needed to create the final FUB array layout.

One may wonder why it is required that all nets in a layout geometry $G_i$ must be implemented within a single FUB $F_j$? In addition, why are two tasks (logic design synthesis and net selection) necessary to assign a layout geometry to certain nets in a FUB? That is, why not merge the two aforementioned tasks into one formulation within the layout design synthesis task. The explanation is that the runtime for solving a merged optimization problem is unacceptable due to the extremely large solution space. For example, a FUB library usually contains over 10,000 FUBs and each FUB consists in excess a hundred nets. Thus, over a million nets form the solution space for incorporating a given layout geometry. By leveraging two separate tasks as described in this work, the solution space is significantly reduced. Particularly, the solution space for logic design synthesis is limited to the size of the FUB library (e.g., 10,000 FUBs). Moreover nets (at most 200) within a selected FUB form the solution space for identifying specific nets for constructing the desired layout geometry.

**Net selection:** Although a FUB $F_j$ has been selected for incorporating layout geometries $G_i$, the specific nets within $F_j$ has not been selected for the geometry in $G_i$, thus necessitating a net selection task. Particularly, the objective of net selection is to identify the specific nets that the overall routing complexity is minimized for incorporating the layout geometry. The reason for reducing the routing complexity is that the ideal test chip is designed to be sensitive to the parameter of interest while being insensitive to everything else. Thus, besides the layout geometries of interest, the remaining layout geometries in the test chip should be deliberately kept simple and uniform. Increasing routing complexity will cause the overall layout to become more complicated and non-uniform. An optimization problem is formulated for net selection that aims to simplify the routing complexity.

**Figure 4.** An example of selecting a layout geometry for incorporating at (a) the logic level, and (b) the physical level.

Fig. 4 is used to illustrate net selection. The example four-gate circuit of Fig. 3a has nets n5 and n6 selected for incorporating the given layout geometry of Fig. 3b. However, multiple options are available to form the target layout geometry. Specifically, net n5 is compatible with P1 since both have three CPs, and all nets n1-n8 are compatible with P2 since they have at least two CPs. Arbitrary selection of nets however can lead to significant increase in routing complexity. The selection of nets n5 and n6 for incorporating the given layout geometry is motivated by the intuition that if the selected nets are within close proximity in the logic implementation, then they are more likely to be routed closely in the physical layout. Here, the proximity between any two nets in the logic implementation is measured by net adjacency which is defined as the minimal number of standard cells that one net traverses to reach the other net or its side input. For example, the net adjacency of nets n5 and n6 is zero, because net n5 is the side input of net n6; the net adjacency of nets n4 and net n5 is one, since net n4 traverses an OR gate to reach the side input of net n5. Since the net adjacency of n5 and n6 is zero, they are selected to incorporate the target layout geometry; the corresponding layout is shown in Fig. 4b. Fig. 4b shows that the remaining layout geometries (grey) are relatively simple and uniform compared to the target layout geometry (shown in the black boundary box).

\[
\begin{align*}
\text{minimize} & \quad \sum_{i} x_i^T D_j x_i \\
\text{subject to} & \quad E_j x_i - S_i \geq 0, \quad 1 \leq i \leq n \\
& \quad \sum_{i} X_{ki} \leq 1, \quad 1 \leq k \leq t.
\end{align*}
\] (3)

Based on the above mentioned, Eq. 3 is formulated for each FUB $F_j$ in the FUB array $\{F_1, F_2, \ldots, F_m\}$ to identify groups of nets to construct target layout geometries, while minimizing routing complexity. In Eq. 3, the matrix $D_j$ characterizes the net adjacency of any two nets in the $j$th FUB, denoted as $D_j = \{d_{1j}, d_{2j}, \ldots, d_{tj}\}$, where $t$ is the total number of nets within the $j$th FUB; each entry $d_{ku}$ in the $D$ tracks the net adjacency between $k$th and $u$th net. The matrix $E_j$ characterizes the number of CPs for each net within $j$th FUB as a matrix $E = \{e_1, e_2, \ldots, e_t\}$, where $t$ is number of nets in $j$th FUB. Each column vector $e_k$ tabulates the number of CPs of $k$th net in $F_j$ as described in Eq. 2. Each column vector $S_i$ tabulates the number of CPs of all nets within the layout geometry $G_i$ as described in Eq. 1. For each $G_i$, a column vector $x_i$ indicates which net in $F_j$ is selected to form the $i$th layout.
The notation “\(1(\text{net}_i \in G_i)\)” in Eq. 4 means that if the \(i\)th net is selected for \(G_i\), the \(i\)th variable of vector \(x_i\) is equal to 1.

\[
x_i = [1(\text{net}_1 \in G_1), 1(\text{net}_2 \in G_2), \ldots, 1(\text{net}_t \in G_t)]^T \tag{4}
\]

Correspondingly, for the entire set \(\{G_1, G_2, \ldots, G_n\}\), a binary matrix \(X = \{x_1, x_2, \ldots, x_t\}\) is formed. Note that every net in the given FUB is tabulated in the same order for each column vector \(x_i\) in \(X\). Because a net can be used only once, only one entry of each row in \(X\) is non-zero. The matrix \(X\) is the quantity being optimized. The objective function in Eq. 3 evaluates the overall net adjacency of the selected nets that form the layout geometries assigned in \(F_j\). The constraint condition in Eq. 3 guarantees that every layout geometry will be incorporated in \(F_j\) with no net being assigned more than once. The outcome of solving Eq. 3 is \(n\) groups of nets within \(F_j\) for the corresponding layout geometries \(\{G_1, G_2, \ldots, G_n\}\).

**Layout creation:** This task leverages commercial place-and-route to construct layout geometries of interest. To be specific, given a list of the layout geometries \(\mathcal{G} = \{G_1, G_2, \ldots, G_N\}\), a set of nets \(\{\text{net}_1, \text{net}_2, \ldots, \text{net}_t\}\) is identified within some FUBs in the array for forming each \(G_i \in \mathcal{G}\). For each net \(\text{net}_k \in \{\text{net}_1, \text{net}_2, \ldots, \text{net}_t\}\), characteristics of \(\text{net}_k\) have been associated with layout geometries found in \(G_i\). Commercial place-and-route is used to create the net geometry outside \(G_i\), thus preserving the spatial relation of the layout geometry for each \(G_i\). An example is shown in Fig. 4b. The layout geometry of interest (shown in the black boundary box) is incorporated as a pre-defined geometry, place-and-route creates the remaining geometries (grey) that are required for the four-gate circuit shown in Fig. 4a.

**IV. Experiments**

Two different libraries at the 45nm and 65nm technology nodes are utilized for a design experiment. Following the design flow detailed in Section III, experiment input includes a list of BEOL layout geometries. In this work, the layout geometries of interest are derived from a benchmark and a product design. To be specific, two different designs are each separately synthesized, placed and routed using the two libraries. One design, B17, is an ITC99 benchmark circuit. The second design is a sub-circuit of the L2 cache write-back buffer uncore (called L2B) from the OpenSPARC T2 processor design [20]. Once the layouts are created, the layouts are further processed to identify all possible location where are sensitive to systematic defects.

Fig. 5 illustrates several potential defect locations considered. Particularly, ends of every polygon are identified and considered as potential defect locations. For example, an open defect may occur at the end of a polygon where two polygons meet (location marked by a star in Fig. 5a). In addition, the polygon ends that are not connected to another polygon could be a location where the bridge defect occurs (location marked by a circle in Fig. 5a). Then, these polygon ends are projected in four orthogonal directions to discover intersections with other polygons as shown in Fig. 5a using dashed lines. The projection distance is determined by the pitch size of the given process design kit. The location where the intersection happens is deemed a potential bridge location, marked by a triangle in Fig. 5a. In addition, every via location is also identified for defects related to via fabrication (e.g., misalignment). An example location is marked by a star in Fig. 5b.

Once all the potential defect locations have been derived, a window is placed around each location. The choice of window size should ensure that polygons that are likely to interact with each other around the defect location are included. If window size is too large, the polygons it contains may be too far away to affect the defect location. On the other hand, if the window size is too small then it may not contain the polygons necessary to capture the systematic defect mechanism. Instead of having a fixed window size, in this work, multiple window sizes corresponding to metal pitch size are applied. Then, polygons that are contained in the window size form the BEOL layout geometries of interest. In addition, the interaction from vertical direction are captured by including the polygons in consecutive layers. In this experiment, polygons in three consecutive layers (two metal layers and one via layer) are included in the BEOL layout geometries. As a result, hundreds of thousands of BEOL layout geometries are identified. However, in reality, it is impossible to incorporate all BEOL layout geometries due to the limited test chip area. All layout geometries are further process to identify the connecting port (CP). Then, the layout geometries are ranked from high to low according the number of CPs. Hundreds or thousands of top-ranked BEOL layout geometries are selected to evaluate the efficacy of the flow shown in Fig. 1. Layout geometries with a higher number of CPs, is ranked high (i.e., deemed important) because of the difficulty of solving the optimization problem described in Eq. 1 and Eq. 3. Demonstrating that we can easily handle geometries with large numbers of CPs means that other geometries, albeit intricate, but with less CPs can also be easily handled by the flow. Above paragraphs describe one method for identifying BEOL layout geometries of interest. As described in Section III.B, alternative approaches [9][17] can be used as well.

Given the BEOL layout geometries of interest, a FUB library is created that contains more than 18,000 different FUB implementations of an information-lossless function that has

![Figure 5](image-url). An example of potential locations for (a) open and bridge defect and (b) via failure.
six inputs and six outputs equally split between the vertical and horizontal ports. Although five hours of compute time is required to create 18,000 FUB implementations, this is a task that is performed only once. Each FUB implementation is then characterized. The resulting testability and CPs of nets of each FUB is then combined with the top-ranked layout geometries to formulate the optimization described in Eq. 1. Specifically, eight lists of layout geometries with two window sizes (600 nm x 600 nm and 800 nm x 800 nm) are collected from B17 and L2B circuit at 45 nm and 65 nm technology nodes, respectively. Correspondingly, the top-500 and top-1000 ranked layout geometries are identified (resulting in 16 lists) for layout geometry incorporation. Thus, 16 FUB arrays are constructed by solving Eq. 1. Because Eq. 1 is an integer-programming formulation, the Branch and Reduce Optimization Navigator [21] is used to obtain the FUB array by solving Eq. 1. The solver runs on a machine with 64 CPU cores running at 2.2GHz with 1,009GB of RAM. Particularly, the value of parameter p in Eq. 1 is set to 800,000 for guiding the solver to select a FUB array with optimal testability.

Table I provides a detailed testability comparison between the benchmark circuit, the product design, and various FUB arrays using the 45nm technology library. As described in Section I, either the benchmark circuit or the product design can serve as an example of the full-flow logic test chip design. Table I shows that the eight FUB arrays created by Eq. 1 achieve 100% coverage for four fault models (single stuck-at line (SSL), bridge (BF), input pattern (IP) and path delay (PDF)). On the other hand, the benchmark and product do not achieve 100% coverage for any fault model, especially for the IP and PDF models. The low coverage for the IP and PDF fault models can be attributed to the simple fact that the benchmark and product designs are not intentionally designed to be ultra testable as is the case for the test chip. In addition, eight FUB arrays are also constructed using the 65nm technology library. These arrays also achieve 100% coverage for all the considered fault models, but due to page limitation the results are not shown in the Table I. Overall, this result demonstrates that the design flow of Fig. 1 can produce a logic design that are able to incorporate the layout geometries of interest with optimal testability.

For the layout geometries physical implementation, 16 FUB arrays resulting from solving Eq. 1 that achieve optimal testability are used to create the layouts that incorporate the layout geometries of interest. Table II. compares the wire length, the layout area, the “defect site density” and the “unique geometry density” for the arrays, benchmark, and product designs at the 45nm and 65nm technology node. Here, the “defect site density” is defined as the ratio of the total number of identified potential defect sites including open, bridge and via defects illustrated in Fig. 5 to the total layout area. It is used to gauge the simplicity of the layout. For example, a layout with lower defect site density indicates that the overall layout geometry is more simple as compared with a layout with higher defect site density. In addition, the “unique geometry density” is defined as the ratio of the total number of unique layout geometries for a given window size to the total layout area. It is used to gauge the uniformity of the layout. For example, a layout with lower unique geometry density indicates the overall layout is more uniform as compared with a layout with higher unique geometry density. Since the layout geometries are defined by the window size, the same layout will have different unique geometry densities based on the window size. For example, for B17 at the 45nm node, the unique geometry density is 2.65 for a window of 600 nm x 600 nm, and increases to 5.79 for 800 nm x 800 nm.

The first observation from Table II shows that the FUB array has less total wire length, layout area compared with the benchmark and product designs. Particularly, the FUB arrays achieve 93.6% and 63% reduction on average in the wire length and the chip area respectively. These reduction can be explained by the fact that the FUB array is deliberately designed to incorporate the layout geometries of interest and nothing more. Another observation is that the FUB array has a smaller unique geometry density (count per square micrometer) and defect site density (count per square micrometer) compared with the benchmark and product designs. It indicates that, besides the layout geometries of interest, the rest of array layout is more simple and uniform (less routing complex), an indication that the FUB array is more sensitive to the interaction between the process and the layout geometries of interest. A third observation is that increasing the number of layout geometries only leads to a linear increase in the wire length, the layout area and CPU run time. This means the methodology is easily scalable with respect to the number of BEOL layout geometries. Here, the CPU run time is used to


<table>
<thead>
<tr>
<th>Tech node</th>
<th>Benchmark and product design</th>
<th>FUB array</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ckt.</td>
<td>Wire length (mm)</td>
</tr>
<tr>
<td>45nm</td>
<td>B17</td>
<td>310.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2B</td>
<td>360.8</td>
<td>0.024</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65nm</td>
<td>B17</td>
<td>413.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table II: Layout comparison between the various FUB arrays, benchmark and product designs for different technology nodes.

V. CONCLUSION

The increasing need for fast yield ramp at the early stages of a new manufacturing process intensifies the role of logic test chips. Because of the limitations of conventional approaches, this work describes a design methodology for incorporating the BEOL layout geometries of interest into a fully testable full-flow logic test chip. Specifically, the design methodology described here significantly improves the capability of a logic test chip for investigating the BEOL layout geometries in the following ways. (1) Various complex BEOL layout geometries that are sensitive to systematic defect mechanisms that affect product designs can be automatically incorporated directly into the layout. (2) Optimal testability for various fault models is guaranteed. (3) Scalability is achieved since the total wire length, layout area and design effort of the test chips are linearly associated with the number of BEOL layout geometries incorporated. Design experiments demonstrate that BEOL layout geometries extracted from benchmark and product design can be easily incorporated into an optimal testable logic test chip. Our current work is focused on designing a full-flow logic test chip that preserves layout demographics simultaneously from including both the BEOL and FEOL, while being fully transparent to any defects.

REFERENCES