Logic Characterization Vehicle Design Reflection via Layout Rewiring

Phillip Fynan*, Zeye Liu*, Benjamin Niewenhuis*, Soumya Mittal*, Marcin Strojwas†, and R. D. (Shawn) Blanton*

*Advanced Chip Testing Laboratory (www.ece.cmu.edu/~actl/)
Department of Electrical and Computer Engineering
Carnegie Mellon University
†PDF Solutions Inc.

Abstract—Continued scaling of semiconductor fabrication processes has made achieving yield targets increasingly difficult. The design and fabrication of various types of test vehicles is one approach for enabling fast yield learning. Recent work introduced the Carnegie Mellon logic characterization vehicle (CM-LCV). The CM-LCV design methodology uses regularity and existing testability theory to produce logic-based designs that are both highly testable and diagnosable. For the CM-LCV to be effective for yield learning, it must reflect the design characteristics of actual product layouts. Previous work enables incorporation of a standard-cell distribution derived from product designs into an LCV while simultaneously ensuring optimal testability. In this work, a new method is proposed for constructing a CM-LCV that reflects the design characteristics of a product through rewiring either the entire layout or some portion thereof. Four different approaches for rewiring are examined, and the results of each approach are evaluated using a variety of metrics. Experiment results reveal that a product layout can be easily rewired to construct an LCV with reasonable wirelength with reasonable CPU time. Rewiring has many advantages including the transformation of an actual product front-end to a logic-based test chip that has significant transparency to failure. Consequently, this means that front-end masks from an actual product can be re-used to create an effective LCV that is both more reflective and inexpensive to fabricate.

I. INTRODUCTION

The continued scaling predicted by Moore’s law requires the development of more complex fabrication techniques. For example, the manufacture of smaller wires now requires double patterning, a fabrication step that is vulnerable to bump defects which were previously only cosmetic [1]. Defects adversely impact yield and must be understood and remedied as quickly as possible to ensure billion-dollar investments in new technology are profitable.

For modern technologies, most defects are systematic, and not random. That is, most defects are a function of the design, the process, or some combination thereof. As a result, it is commonplace to fabricate example chips (i.e., test chips) so that systematic defects can be discovered before actual customer products are fabricated [2]. Typically, several different kinds of ‘test chips’ are fabricated to evaluate the manufacturability of various types of structures and devices of differing levels of complexity. Fig. 1 illustrates examples of various test vehicles. Ideal test chips have two characteristics. The first is transparency, which is necessary so that defects can be easily detected and diagnosed. The second is similarity to actual customer chips, since systematic defects may only manifest in more complex designs. Various types of test chips have these two characteristics with varying degrees of efficacy.

The simplest test chips consist of interconnect structures such as comb drives, serpentine wires, or via chains [3]. These structures are quite transparent, but are similar to actual product designs in only the most elementary fashion. However, these structures fulfill a very important role in yield learning since their transparency is useful for characterizing actual defects.

More sophisticated test chips are actual customer designs or portions thereof. Because these test chips actually include the complex and unique geometries of real designs, they are the most likely to uncover systematic defects in customer chips. However, product-like test chips are not inherently transparent, so testability and diagnosability can suffer greatly. There are exceptions, however. SRAM is an excellent test vehicle since it is an actual product, and inherently possesses good test and diagnosis properties. Because of these properties, SRAM is typically fabricated during yield ramping. On the other hand, SRAM is not sufficient since it does not resemble the standard-cell logic used in many product designs due to its regularity and limited use of layers and geometries. Thus, test chips similar to other product designs are also needed.

Another important type of test chip is the logic characterization vehicle (LCV). LCVs focus on standard-cell logic, and are commonly made by taking standard cells or subcircuits from legacy product designs and modifying them to be more transparent to failure. Transparency is conventionally improved by adding design-for-test structures such as scan chains and/or control/test points. However, most existing work on LCVs does not explicitly address specific fault/defect testability or diagnosability of the LCV, or quantify its similarity to product designs.

Many LCV designs are described in previous work. For example, one LCV consists of various circuits with a common test-access mechanism [4]. That LCV uses three test circuits, namely, a square-root implementation that is aimed at uncovering front-end-of-line (FEOL) defects, a ring oscillator with extensive wiring to discover back-end-of-line (BEOL) defects, and an SRAM to capture memory-related defects. The trans-
The CM-LCV achieves testability using the property of C-testability [8]. C-testability means that each FUB can be exhaustively tested with a constant number of tests independent of the size of the array of FUBs. Later work achieved 100% cell-aware coverage [9]. Diagnosability is enhanced by the structure of the array. The two-dimensional array structure of the CM-LCV causes fault effects stemming from a defective module to propagate to all FUBs downstream. Because each FUB implements the same bijective function, errors are guaranteed to propagate. Experiments demonstrate that the CM-LCV can achieve a better diagnostic resolution than can be typically achieved with conventional approaches [6].

The CM-LCV achieves similarity to product designs because each FUB can be implemented with different logic and/or layout. Because each FUB implements the same function, an LCV design can be assembled by selecting a set of FUB implementations from a large library to better reflect the properties of a portion of a product design, an entire product design, or family of product designs. Previous work has focused on reflecting the standard-cell distribution of product designs in the CM-LCV, where a cell distribution is simply the count of each type of cell in the design, (i.e., the count of NAND gates, OR gates, inverters, etc.). By synthesizing various FUB implementations to form a FUB library with the standard-cell functions of interest, any cell distribution can be reflected.

However, a cell distribution is only one aspect of design reflection. Design reflection is the more general idea that an LCV should possess characteristics of actual product designs so that it is vulnerable to the same defects as the product. Because the manufacturability of a design will depend on its layout patterns, an LCV then should use similar layout patterns found in product designs. The CM-LCV can indirectly capture similar patterns by using the same standard cells as used by actual product designs, along with the same orientation of cells, and the same cell neighborhoods. Prior work on the CM-LCV design only reflected the cell distribution and did not take cell orientation or cell neighborhood into account.

Reflecting cell neighborhoods is the more challenging problem. A cell neighborhood is defined to be the combination of cells (including orientation and reflection) that surround a cell of interest. Prior work has explored the impact of cell neighborhoods on manufacturability [10], [11]. For example, Fig. 2 shows an AND-OR (AO22X4) gate with an AND-OR-INVERT (AOI22X2) gate above, an AND (AND2X1) gate to the left, etc. Reflecting cell neighborhoods is challenging due to both the number of cell neighborhoods and the fact that most are unique, appearing only once in a given product layout. Incorporating all of these product cell neighborhoods into an LCV is further complicated by the fact that existing place-and-route flows do not provide a direct mechanism for constraining the cell neighborhoods used in a design.

Table I shows the results of a cell neighborhood analysis on the CM-LCV arrays generated for three different ITC99 [12] benchmark circuits. Each LCV is constructed to match the cell
In this work, a method to construct a CM-LCV that reflects the cell distribution, orientation, and neighborhoods of a product design is described. In order to achieve better reflection, the method rewires the cells in a product layout to form a CM-LCV. The method takes as input the product layout. The routing between cells is removed, leaving only the standard cells. The method proceeds to connect the cells together into a LCV, culminating in a mapping between the layout cells and the LCV netlist. The resulting design can then be routed with a commercial tool.

The rewiring method produces an LCV with superior ability to detect static defects in standard cells, while maintaining the same cell placement as the product layout. Since the cell placement of the product layout is unchanged, the cell neighborhoods will be preserved. Additionally, the CM-LCV will perfectly reflect layers used by the standard cells, specifically the FEOL layers and metal-1. Consequently, this means that the front-end masks of an actual product can be re-used to create an effective LCV, thus significantly reducing costs of LCV fabrication. Incorporating testability, diagnosability, and now cell neighborhoods within the CM-LCV design methodology ensures a characterization vehicle that can better aid in yield learning.

However, rewiring has limitations. The CM-LCV is only designed to detect static faults; delay faults will be addressed in future work. The reflection is limited to layers used in the standard cells, namely metal-1 and below. Because the remaining layers are recreated during rewiring, patterns in those layers may not match those present in the product design. However, since the CM-LCV is currently focused on the standard cells, the mismatch in metal-2 and above is less of a concern. Furthermore, this method only reflects one product layout, while other products can have very different layouts. However, the advantage offered by rewiring can be maximized by targeting the flagship product.

The rest of this paper describes the details of our approach for layout rewiring to construct a CM-LCV with the aforementioned characteristics. Specifically, Section II provides an overview of the CM-LCV, and the flow used to implement a CM-LCV is described. Next, in Section III, the rewiring flow is presented, including four FUB assignment approaches and the intra- and inter-FUB wiring procedures. Section IV demonstrates the efficacy of layout rewiring using one industrial and three benchmark circuits. Finally, conclusions and future work are presented in Section V.

II. CM-LCV

As noted in the introduction, transparency is critical for test chips used in yield learning. The CM-LCV methodology selects a transparent logical function and implements it using product-like physical features. This approach is driven by the insight that the manufacturing process is only sensitive to the physical characteristics of a design (i.e., its physical layout) and not the functionality that the layout implements. This section is thus divided into two parts, the first of which describes the logic testing theory that informs the CM-LCV design, followed by a description of the CM-LCV implementation flow.

A. Regular Circuit Test

Regular circuits are a class of circuits composed of identical blocks that are interconnected in a uniform fashion. Examples of regular circuits include one-dimensional arrays, convergent and divergent trees, and two-dimensional arrays. In 1973, Friedman described the necessary and sufficient conditions for one-dimensional array circuits (without vertical outputs) to be what he called C-testable [8]. A regular circuit is C-testable if every block-level input pattern (IP) fault [13] can be tested for every block regardless of the size of the circuit with a constant number of test patterns. In other words, C-testability implies that the function of every circuit block can be completely verified with a fixed number of test patterns. The necessary and sufficient conditions for C-testability in one-dimensional arrays with vertical outputs [14] and tree circuits [15] have been derived, however the conditions for two-dimensional array C-testability remain an open problem despite extensive studies (e.g., [16]–[18]).

In addition to the total number of tests being fixed independent of the circuit size, the cardinality of the constant test

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Total cell neighborhoods</th>
<th>Unique cell neighborhoods</th>
<th>CM-LCV reflecting benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Total cell neighborhoods</td>
</tr>
<tr>
<td>b17</td>
<td>12,854</td>
<td>12,617</td>
<td>9,072</td>
</tr>
<tr>
<td>b18</td>
<td>40,073</td>
<td>39,593</td>
<td>31,329</td>
</tr>
<tr>
<td>b19</td>
<td>79,742</td>
<td>78,519</td>
<td>63,093</td>
</tr>
</tbody>
</table>

TABLE I

Cell Neighborhood Analysis for Various Benchmark Circuits and Corresponding CM-LCV Implementations
Fig. 3. Application of a single test to a 5×3 array of FUBs. The FUB located at the intersection of the second row and third column produces an erroneous response (represented using an expected/observed notation) that propagates through the array and is observed at the array boundary.

set tends to be minimal or near minimal. Thus, the C-testable property of regular circuits is particularly appealing for use in a test chip as it provides strong guarantees concerning the number of tests and fault testability while preserving flexibility with respect to the size of the circuit.

B. CM-LCV Architecture

A CM-LCV is a collection of one or more two-dimensional FUB arrays along with their test access mechanisms. A two-dimensional array is selected as it provides improved diagnosability due to the diverging fanout for FUB outputs. Although the necessary conditions for two-dimensional array C-testability are unknown, a simple approach that ensures C-testability is to use a FUB function that is bijective. A bijective Boolean function is a lossless function with equal numbers of inputs and outputs where each unique set of input values maps to a unique output set of values. It can be easily shown that a two-dimensional array that has an $n$-input bijective module function is C-testable for all IP faults with the minimum number of test patterns, namely $2^n$. Fig. 3 illustrates a test applied to a FUB array with errors propagating from a single defective FUB (shown highlighted).

Additional improvements to the array diagnosability can be achieved by applying additional constraints on the bijective FUB function. In [6], a more strict property called VH-bijectivity is described. VH-bijectivity ensures that an error signal that arrives on only one input port of a FUB is propagated to both the vertical and horizontal output ports. An array composed of VH-bijective FUBs results in near-ideal, FUB-level diagnostic resolution for defects confined to a single FUB regardless of the array size. Similar diagnosis outcomes are also achievable in many cases for multiple defective FUBs.

C. CM-LCV Implementation

Having determined the function and structure of the CM-LCV, there remains the problem of implementing it in a manner that reflects the properties of actual product designs. The portion of the CM-LCV design flow, shown in the shaded section of Fig. 4, begins with a library of standard cells for a given technology. In the first step, the logical functions are extracted from the standard-cell library to form a logic library consisting only of functions. Note that a typical standard-cell library contains numerous logic functions implemented as cells with varying drive strengths; for example, there may exist a two-input NAND with nominal drive strength (denoted typically as NAND2X1) and a two-input NAND with twice the drive strength (e.g., NAND2X2). Both of these cells implement the same two-input NAND logic function. The resulting logic-function library is used to repeatedly synthesize a given FUB function, resulting in a library of FUB implementations. Each of these FUB implementations is examined for its standard-cell usage, fault coverage, and diagnosability characteristics.

A different analysis is performed on the product design or family of product designs. The standard-cell usage and fault coverage characteristics measured from the product designs are used in addition to input from the product designers to determine the objectives for the CM-LCV. These targeted objectives are combined with the data collected from the FUB implementations in the FUB library to formulate a matrix optimization problem. This problem is then fed to a solver, which identifies a set of FUBs from the library, called a FUB template, that best achieves the given objectives. This FUB template can then be instantiated any number of times to construct an array of arbitrary size that maintains the desired design, test, and diagnosis properties.

III. PRODUCT LAYOUT REWIRING

To rewire the product layout into a CM-LCV, a flow is developed that begins with a product layout and FUB template (as described in the previous section), then generates a new netlist as illustrated in Fig. 4. The steps in the flow consist of assigning cells from within the existing product layout to FUBs, original wire elimination, intra-FUB rewiring, and finally inter-FUB rewiring. In the following sections, a distinction is drawn between layout cells, which are defined as the physical cells in the layout; and FUB cells, which are defined as the cell instances from the FUB netlists. During rewiring, the FUB cells may be assigned to different layout
cells, but the layout cells never deviate from their original placement.

In the first step, each FUB from the template must be assigned to a set of layout cells such that the FUB has the standard cells of the proper types for its implementation. While any assignment of the proper cells would be sufficient, this step aims to assign cells in close proximity to the same FUB. In product designs created with place-and-route, cells close to each other are more likely to be connected. Thus, the resulting LCV reflects wiring similar to that found in product designs. Bias towards assigning FUBs to a compact area ensures the LCV will contain local wiring within FUBs and longer wires between them. For this step, four different approaches are developed and compared.

The next step is trivial and simply eliminates the original wiring between layout cells. The cell neighborhoods of the product layout remain unchanged, resulting in a layout that exactly reflects the cell neighborhoods of a chosen product. The third step is intra-FUB rewiring, which connects the cells within the product together to form individual FUBs. To connect the cells together, each specific FUB cell must be assigned to an element of the set of layout cells assigned to the FUB in the first step.

Finally, inter-FUB rewiring must be completed. In this step, the FUBs are interconnected into an array to form the LCV. A netlist is created from the new intra-FUB and inter-FUB wires, which is given to a commercial routing tool to generate new BEOL layers. Unused layout cells are either left unconnected or have their inputs tied to ground or $V_{dd}$.

In the following, details of each of the non-trivial steps of the rewiring flow are described. Particular emphasis is placed upon the four approaches presented for assigning product-layout standard cells to the FUBs.

A. FUB Assignment

Each FUB is a logic netlist, that is, a simple network of interconnected standard cells that implement a bijective function. FUB assignment for a given logic netlist involves identifying cell types in the layout that match those used in the netlist. There are several objectives in FUB assignment that are targeted. First, we want to ensure that the cells used for a given FUB are in close proximity so that the overall area of the FUB is minimized. Small FUBs ensure that FUB diagnosis is localized to a small area of the test chip. Second, we want to limit the intra-FUB interconnect as much as possible. Limited intra-FUB interconnect ensures wiring is localized and thus product-like which overall makes failure due to interconnect less likely. Finally, it is desired that overlap among FUBs is minimized. Minimizing overlap reduces the likelihood that a single-site of failure will lead to multiple faulty FUBs, thus hindering diagnosis. FUB assignment is also complicated by the fact that some types of cells are restricted to certain areas of the chip. For example, NAND2 and AOI21 may be well distributed throughout the layout, but this may not be the case for the OR2 and AOI222. This means that FUB assignment for netlists that use either OR2 or AOI222 will be more challenging. Four different methods are presented: commercial-tool based, greedy, geometric, and analytic.

1) Commercial Tool: Commercial EDA tools (e.g., Synopsys IC-compiler, Cadence Encounter Digital Implementation [19], etc.) include flat and hierarchical design planning, placement, routing, and optimization that enable the user to revise the routing and change the functionality of the original design. Of particular interest for this paper is the post-mask engineering change order (ECO) flow, which allows the user to modify the design logic by making changes to the routing (BEOL). This makes the ECO flow a straightforward solution to the FUB assignment problem.

The ECO flow consists of three phases. In the first phase, wires between standard cells within the original product layout are eliminated. In the second phase, the commercial tool takes the new design logic, in this case the gate-level netlist of the FUB array, and assigns each standard cell in the new design logic to the location of a cell of the same type in the layout produced in phase one. The standard-cell assignment of the ECO flow is driven by a timing constraint, which promotes smaller wirelength and matches the constraints used in actual product designs. The final phase routes the assigned standard cells together to complete the CM-LCV.

The primary benefit of the ECO approach for FUB assignment is that it is available in all place-and-route tools, which means there is no need to spend effort constructing a new approach for this problem. The primary disadvantage of the ECO approach is that it is sub-optimal compared to a dedicated approach, since an ECO flow is not intended to rewire entire designs.

2) Greedy: The main idea of this approach is to iterate through all of the FUBs in order to find the one that fits best in a randomly-selected region of the chip, and repeating this process until all the FUBs have been assigned. The criterion for the best-fitting FUB is the area required for the FUB, which is equated to the convex hull of the positions of all the standard cells used by the FUB. The FUB with the least area is deemed to be the best fitting. Obviously, other metrics could be easily used instead for identifying the best-fitting FUB.

The first step in the greedy approach is to sort each cell type in the unassigned FUBs by frequency and select a random cell of the most common cell type as the starting location. All the FUBs that contain this cell type are analyzed and the best-fitting FUB (i.e., the one with the smallest convex-hull area) is assigned during that iteration. Analysis of each FUB involves looking up the numbers and types of cells required to implement that FUB, identifying the required number of cells for each cell type in the design that are nearest to the starting location, and finding the convex hull of the resulting collection of cells. To identify the nearest cells of each cell type, an R-tree is employed, which is a data structure that provides fast lookup of locations that are in close proximity. Once the best-fitting FUB is identified the list of FUB assignments is updated and the process is repeated.

The primary advantage of the greedy approach is that it is simple and straightforward. Its main disadvantage, like
any greedy approach, is that execution could lead to a local minima, i.e., the total area of the FUB array may not be optimized. Furthermore, the convex-hull area may not be the best metric for identifying the optimal FUB to place at a given location.

3) Geometric: The geometric approach uses spatial information to solve the FUB assignment problem. This approach consists of two phases. The first phase is trivial, in that the layout is divided into sections and each section is assigned a FUB implementation from the template that best matches the standard cells within that section. In the second phase, all of the standard cells in the product layout are iteratively partitioned and assigned to FUBs in an order determined by the section assignments in the first phase.

The first phase of the geometric approach determines an approximate spatial location for each FUB in the layout. This phase begins with dividing the layout along both the x and y axes into a number of sections that approximately equal the number of FUBs in the array. The next step is to assign a FUB implementation to each section. To accomplish this, two parameters are defined: first, mismatch $M$ is a vector that represents how well a given FUB implementation fits into a given section for each of the $n$ types of logic cells. Vector $M$ is defined as the number of cells present in a given section minus the number of cells needed for a given FUB implementation, with negative values indicating that the FUB implementation requires more cells than are present in that section of layout. Second, the scalar $q$ quantifies the cost of a FUB assignment to a section. $q$ is defined as:

$$q = \sum_{i=1}^{n} \left\{ \begin{array}{ll} kM_i^2 & \text{if } M_i < 0 \\ M_i & \text{if } M_i \geq 0 \end{array} \right.$$  

Note that $q$ penalizes a deficit of standard cells more highly (the term is squared and multiplied by weight factor $k$) than a surplus of standard cells for a given FUB assignment. A $q = 0$ ($q \approx 0$) is preferable (ideal) as it indicates a better match between the FUB implementation and the layout section.

Given $M$ and $q$, the iterative process of assigning the FUB implementations to layout sections is accomplished using a straightforward application of the simulated annealing algorithm. Initially each layout section is randomly assigned a FUB implementation and a total $Q$ is calculated as the sum of the $q$ for each assignment. During simulated annealing, random pairs of FUB assignments are swapped and the net change in $Q$ is calculated. FUB swaps are initially accepted at a high rate regardless of the change in $Q$, however, as simulated annealing proceeds the likelihood of a swap being accepted decreases until only swaps that improve $Q$ are accepted. This process is known as cooling, and the cooling schedule adopted here is described in [20].

The second phase uses the FUB assignments from the first phase to guide the assignment of standard cells to FUB implementations. This is accomplished through a recursive function that operates on all of the cells of one type at a time. This function takes a layout region, a list of standard cells, and an axis ($x$ or $y$). It partitions the spatial region along the axis such that the FUBs that have been assigned to the sections within each of the two subregions require approximately the same number of cells of the given type. The cell list is then sorted by position along the given axis and divided into two sublists that satisfy the requirements for the corresponding subregions. This function then calls itself for each subregion and sublist pair with the opposite axis. If only one FUB implementation remains within the region, the function then assigns all cells in the list to that FUB implementation. Note that in both phases, the original placement of layout standard cells never changes, so the cell neighborhoods of the product design are preserved.

The geometric approach has several advantages. The first is speed; many of the computations required are relatively simple and can be performed in parallel. A second advantage is that the geometric approach naturally attempts to implement each FUB using the cells found in a localized region of the layout. This results in a tendency towards solutions with compact FUBs that are distributed throughout the layout, which may be desirable in some situations.

There remain, however, disadvantages to the geometric approach. The primary disadvantage is that the layout sections created in the first phase may be incompatible with the optimal solution for a given problem. For example, some FUB implementations may be larger than others, or may require specific combinations of cells that do not occur in a localized region of the provided layout. For these cases, the geometric approach is unsuitable due to the mismatch between the FUB implementation requirements and the layout sections chosen.

4) Analytic: This approach leverages ideas from analytic placement algorithms for ASICs. Analytic placement algorithms find placements for standard cells by iteratively solving an optimization problem to minimize the wirelength of the design. Early iterations are approximate, that is, they do not produce legal placements (e.g., some cells may overlap or be outside the placement region boundaries). Later iterations improve both the wirelength and the legality of the placement. After the optimization completes, the placement must be refined to find the best local placement and ensure the placement is completely legal [21].

For FUB assignment, the analytic approach similarly minimizes wirelength. It begins with a list of all of the cells used by the FUBs (the FUB-cell list) and a corresponding list of assigned locations for each cell. The analytic approach proceeds by alternating between two steps: legalization and optimization. In the legalization step, each cell in the FUB-cell list is assigned to a legal location as determined by the positions of the cells in the layout. In the optimization step, the assigned cell locations are adjusted to minimize the wirelength of the FUBs. In both these steps, the original placement of layout cells never changes. While FUB cells may move in the optimization step, they will be assigned to pre-existing layout cells after the legalization step.

The design is legalized by recursive partitioning. For each cell type, the FUB-cell list is partitioned by the assigned
locations at the median location in one direction (vertical or horizontal). Likewise, the cells in the layout are also partitioned at their median location by a cutline. Each partition of FUB cells is assigned to the corresponding partition of layout cells. This legalization is repeated recursively in alternating directions until the number of cells remaining falls below an arbitrary threshold (64 is used in the implementation). Each cell in the FUB-cell list is then assigned to the nearest unassigned layout cell location in the same partition in some random order. In the end, each cell in the FUB cell list has a legal assignment to a layout cell location after the legalization step has completed.

The optimization step minimizes the wirelength between the cells. Wirelength is modeled as quadratic wirelength, which is the squared Euclidean distance between the pins. Wires connecting more than two pins are represented as a clique. Each wire with more than two pins is replaced by pairwise wires that fully connect all the pins. Because this representation creates many more wires, each new wire is weighted by $1/(n-1)$, where $n$ is the number of pins in the original wire. While this model will not perfectly match actual wires, quadratic wirelength is used because it forms a convex quadratic optimization problem that can be solved efficiently [21].

Without constraints, the optimization step would place all FUB cells in the center of the product layout. To promote a more useful placement, each FUB cell is attached to its last legal location by a pseudo-wire, an approach used in [22]. The pseudo-wire ensures each FUB cell is not moved very far unless there is significant advantage. In addition, all FUB cells are connected by a wire to compress them together into a smaller area. The resulting quadratic optimization problem has a solution given by a sparse system of linear equations, which is solved using the conjugate-gradient method.

The legalization and optimization steps are repeated until either the number of iterations exceeds a maximum limit or the wirelength does not improve. Once done, legalization is performed once more to generate a final legal assignment.

The main advantage of this method is that it uses properties of the intra-FUB connections to guide FUB assignment in order to approximate a global optimum. The primary drawback of this approach is complexity as it requires many computationally-intensive iterations.

### B. Intra-FUB and Inter-FUB rewiring

After FUB assignment, each FUB has been assigned to a set of cells in the product layout. However, two problems remain¹: the cells still must be mapped to instances in the FUB netlist for each FUB, and all of the FUBs must be connected to form the array. In the first problem, called intra-FUB rewiring, cells of the same type can be interchangeably assigned to different instances in the FUB netlist. In the second, called inter-FUB rewiring, the FUB instances can be connected in any order to implement the overall FUB array.

Simulated annealing, as described in the section on the geometric approach to FUB assignment, is again used for both intra-FUB and inter-FUB rewiring. In both problems, wirelength is the quantity optimized. For intra-FUB rewiring, the wirelength between cells within each FUB is minimized. For inter-FUB rewiring, the objective is to minimize the length of wires connecting the FUBs together. Wirelength is modeled using the half-perimeter wirelength (HPWL) model, which is a common objective in ASIC placers [21]. HPWL is calculated as half the perimeter of the bounding box containing all pins connected by the net.

In intra-FUB rewiring, the variable optimized is the total wirelength associated with the assignment of the FUB netlist to cells in the product layout. Because the netlist specifies the inter-cell connections, assigning netlist cells to layout cells implies the creation of wires in the layout. Similarly, inter-FUB rewiring optimizes over the assignment of each FUB to an element in a two-dimensional array. Good FUB assignment means FUBs assigned to standard cells near each other within the layout result in FUBs near each other in the array.

After intra-FUB rewiring is complete, each cell in the netlist corresponds to a cell in the layout. Upon completion of inter-FUB rewiring, the FUBs are connected to form an array. The final output is a complete netlist for the CM-LCV that is ready to be routed by any available commercial tool.

### IV. Experiment

All four rewiring approaches are evaluated using four different designs. Three of the designs are ITC99 benchmark circuits [12] implemented using a commercial 65nm standard-cell library. The fourth design is an industrial layout implemented using a 7nm standard-cell library. The size of these designs (number of standard cells and area) are shown in Table II.

A FUB template for each design is obtained using the flow described in Section II. In order to demonstrate the feasibility of the rewiring techniques, the FUB templates are optimized for utilization rather than fault coverage. Because the three ITC99 designs share the same standard-cell library, the same FUB library can be used. A separate FUB library is created however for the industrial design. In each library, the same two bijective functions are used. One is a 4-bit function; the other is a 6-bit function. The use of two different functions allows better utilization of layout cells by providing a more diverse library of FUBs. From the template, two FUB arrays are created, one for each bijective function. The dimensions of each FUB array are derived from the template so that each array is as close to square as possible. The sizes of each template and the dimensions of each array are also shown in Table II.

As can be expected, the FUB arrays for a given design does not use every single cell in the layout. The utilization of cells within each product layout is reported. Utilization is high, consistently above 70%, and extremely high for the industrial design at 91%. Cell reflection is shown in Fig. 5.

¹The ECO flow is an exception because the commercial tool automatically maps layout cells to FUB netlist instances. Thus the intra-FUB rewiring is already accomplished when using the ECO flow.
Some cell types are underrepresented in the FUB template, notably more complex cells such as AOI22X1. This can be attributed to the difficulty in synthesizing a large variety of FUB implementations that use these more complex cells without significant numbers of basic cells such as INVERTERs, NANDS, and NORs. Each product design and FUB template also has an IP fault coverage and diagnostic coverage [23] calculated. The coverages do not include unutilized layout cells, which are not tested. While the existence of untested layout cells may seem concerning, the cells that are tested are more exhaustively tested, leading to better characterization of defects. As Table III shows, in every case, both fault and diagnostic coverage are better for the CM-LCV than for the original design. Higher fault and diagnostic coverages are achievable by optimizing the FUB templates for testability and diagnosability instead of utilization.

Once the FUB templates have been created, each layout is rewired to form a CM-LCV. Each of the four approaches presented in the previous section are used to create a FUB assignment for each of the four layouts, then intra- and inter-FUB wiring is performed for all approaches except ECO since it handles both types of wires inherently. Finally, a commercial routing tool is used to route the resulting netlist into the final LCV layout. Despite the large changes to the routing in the rewired design, the routing process did not encounter problems with congestion.

Table IV presents the results of the comparison experiment. Each LCV is evaluated using several different metrics. The first is FUB area, which measures the sum of the areas of the convex hulls of all the cells in each FUB. This area gives an indication of how likely it is that FUBs will overlap. Greater FUB overlap increases the likelihood that a defect will affect multiple FUBs, making diagnosis more difficult. Note that total FUB area can be larger than the layout area since FUBs may overlap; that is, overlapped area is not subtracted from the total area calculation. The second metric is wirelength, which measures the total length of all wires in the design. An LCV with shorter wirelength is preferable since it will likely be more compact, and less susceptible to interconnect defects. Wirelength is calculated by the commercial routing tool, except for the industrial design where it is instead estimated by half-perimeter wirelength (HPWL). The third metric is the compute time required for each approach, as measured on a machine with 64 CPU cores running at 2.2 GHz with 1,009 GB of RAM. Additionally, the measured wirelengths of the original product design and of an LCV made using the same template without restrictions on the standard cell placement are provided for comparison.

First, the wirelengths of the rewired LCVs are compared for each approach. ECO performs worst because it is simply not engineered to rewire an entire design. The greedy approach performs second best on the smallest two designs (b17 and b18), but falls into third place for the two larger designs. The smaller designs are likely easier to optimize, so the greedy approach performs well. The geometric approach performs second best on the larger designs and third best on the smaller. The analytic approach performs best on all designs, and thus is the superior approach for rewiring.

FUB area follows the same general pattern as wirelength, with a few differences. Again, ECO performs worst. For b17, the greedy approach performs better than the others, followed by the geometric approach. For b18 and b19, the geometric is best, followed by the analytic. Finally, for the industrial design, the analytic is best, followed by the geometric. The greedy approach performs better on the smallest design, the geometric approach does best on the middle designs, and the analytic only performs well on the largest. Despite having second or third best wirelength, in many cases the geometric approach has better FUB area, suggesting that it generates a more compact FUB in general, than the other approaches.

Comparing execution time, ECO is consistently the fastest in terms of total time. However, this advantage stems from not having to perform intra- and inter-FUB wiring as separate steps. Just comparing the FUB assignment time reveals that ECO takes much more time to make similar decisions. The analytic and greedy approaches both scale poorly compared to the geometric approach, which is often the fastest. The total
time differs considerably due to the wide variance of time taken by the intra- and inter-FUB wiring, due to the stochastic nature of simulated annealing.

The wirelengths of the LCVs resulting from most of the approaches compare quite well to the lengths extracted from the original designs. The one exception is ECO: for b19, it produces a wirelength that is 9× larger than the total length of the original design. The wirelength for b17 is less for the analytic than the original design, likely because utilization is low, allowing more room for routing and enabling decreased wirelength. As another point of comparison, an unconstrained version of the FUB array is constructed, where cells are placed-and-routed using conventional approaches. Running place-and-route on the FUB array produces wirelengths that are less than half the original design wirelengths. This is an expected outcome given that the FUB array is a two-dimensional array of blocks that can be easily placed-and-routed optimally through FUB abutment that mimics the two-dimensional regularity.

Other comparisons are timing and the distribution of unused
cells over the layout area. The critical path of all the rewired designs is at most 4.3x worse than the original design. However, for the purposes of this paper, timing is not a concern since the CM-LCV can be tested at whatever speed is necessary. Finally, the unused cells in the rewired design are reasonably well distributed across the layout. While unused cells that are of the most common cell types are evenly distributed over the layout, unused cells that are of uncommon types are clustered into smaller areas. Thus, defects that only appear in certain regions of the die may be unobserved in uncommon cells, but should still be observed in common cells.

V. Conclusion

This paper has presented a flow for rewiring existing product designs into a highly transparent logic-based test chip called the CM-LCV. The rewiring flow presented in this paper has a number of advantages. The first advantage is that many design characteristics are incorporated ‘for free’ by reusing layout, whereas other techniques require extra effort to ensure that the design characteristics of interest are properly represented. For example, the cell neighborhoods of the generated CM-LCV perfectly match the actual product layout since they are preserved during rewiring. A second advantage of this flow is that the resulting CM-LCV is guaranteed to be highly testable and diagnosable. Each FUB is completely testable for all FUB-level IP faults with a minimum number of test vectors, whereas the high logical fanout causes the resulting circuit to be highly diagnosable. Finally, rewiring of a new-technology flagship product means that existing front-end masks can be utilized for systematic-defect yield learning. Disadvantages of this test-chip design flow include the lack of utilization of all cells in the rewired layout as well as the creation of a completely new BEOL which may not be reflective of product layouts.

Future work includes improving the utilization through modifications to the FUB template generation flow and exploring the possibility of allowing the FUB assignment approach to influence the FUB template. Reductions in wirelength and FUB area will also be pursued by further improving the FUB assignment approaches and incorporating additional ideas from literature on FPGA and ASIC placement and routing. Future work will also address other types of defects such as delay defects. The ultimate goal is to use this flow to create and fabricate test chips in a modern process. To this point, we are currently working with an industrial partner on a tape-out and volume production of a CM-LCV design in 7 nm technology.

REFERENCES


Paper 10.1 INTERNATIONAL TEST CONFERENCE 10