ECO Optimization Using Metal-Configurable Gate-Array Spare Cells

Hua-Yu Chang, Iris Hui-Ru Jiang, *Member, IEEE,* and Yao-Wen Chang, *Fellow, IEEE*

***Abstract*—Due to the rapidly increasing design complexity in modern IC designs, metal-only engineering change order (ECO) becomes inevitable to achieve design closure with a low respin cost. Traditionally, preplaced redundant standard cells are regarded as spare cells. However, these cells are limited by predefined functionalities and locations, and they always consume leakage power despite their inputs being tied off. To overcome the inflexibility and power overhead, a new type of spare cells, called metal-configurable gate-array spare cells, are introduced. In this paper, we address a new ECO problem, which performs design changes using metal-configurable gate-array spare cells. We first study the properties of this new ECO problem and propose a new cost metric, aliveness, to model the capability of a spare gate array. Based on aliveness and routability, we then develop two ECO optimization frameworks, one for timing ECO and the other for functional ECO. Experimental results show that our approach delivers superior efficiency and effectiveness.**

***Index Terms*—Engineering change order, gate array, mixed integer linear programming**

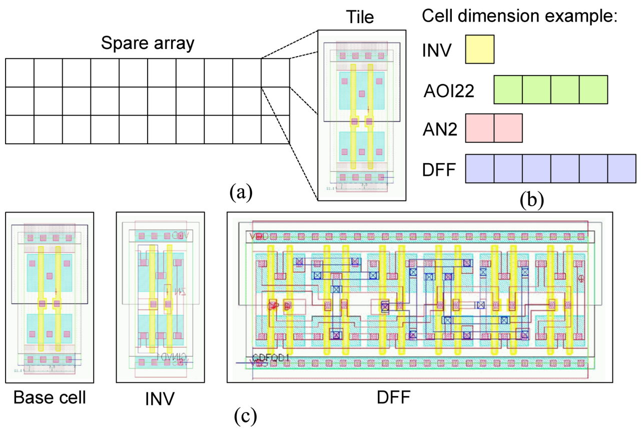


Fig. 1. Metal-configurable gate-array spare cells. (a) Spare array. (b) Predefined cell dimensions. (c) Configuring metal layers to realize a specific functionality.

I. Introduction

DUE TO THE rapidly growing design complexity, some timing and functional failures might not be detected until late design stages. To remedy these late-found failures in a short turn-around time and with a low respin cost [1], metal-only engineering change order (ECO) realizes incremental design changes by only metal-layer modifications. Consequently, metal-only ECO becomes an essential process in the modern IC design flow.

Metal-only ECO has been extensively studied in recent literature [2]–[12]. To enable metal-only ECO, these works use preinserted redundant standard cells as spare cells. To avoid gate floating, the inputs of these redundant cells should be connected to tie cells for ESD reliability. Once a design failure is detected, proper spare cells are activated by rewiring their inputs and outputs. However, the weaknesses of using standard cells as spare cells are twofold: First, these standard spare cells are limited in functionality, quantity, and location. This inflexibility may result in unfixed failures due to a shortage of proper spare cells. Second, these standard spare cells and tie cells always draw leakage current. Since leakage is of particular importance not only to hand-held mobile devices but also to standby circuit operations, the power overhead limits the allowable number of inserted spare cells.

To overcome the inflexibility and power overhead issues, a new type of spare cells, called metal-configurable gate-array spare cells, are proposed [14]–[18]. The gate-array spare cells are developed by combining gate arrays and structured ASICs [19]. As shown in Fig. 1, a block of gate-array spare cells, named a spare array, is an array of tiles. Each tile is composed of unwired transistors, which consume no power. A designated functionality can be formed by configuring the metal layers on top of several consecutive tiles. Hence, a spare array can provide a flexible resource to realize ECO for neighboring gates.

In a related work [20], Chen *et al.* propose reconfigurable decoupling capacitance (decap) cells which can be programmed as functional cells. These configurable decap cells are inserted according to IR drop consideration, and thus

Manuscript received January 21, 2013; revised May 3, 2013; accepted June 25, 2013. Date of current version October 16, 2013. This work was supported in part by IBM, Synopsys, TSMC, and NSC of Taiwan under Grants NSC 100-2221-E-002-088-MY3, NSC 99-2221-E-002-207-MY3, 99-2221-E-002-210-MY3, 101-2220-E-009-044, and NSC 101-2628-E-009-012-MY2. An earlier version of this paper was presented at the ACM/IEEE Design Automation Conference 2012 [13]. The Associate Editor coordinating the review process was Dr. Prashant Saxena.

H.-Y. Chang is with the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan (e-mail: huayu.chang@gmail.com).

I. H.-R. Jiang is with the Department of Electronics Engineering and Institute of Electronics, National Chiao Tung University, Hsinchu 30010, Taiwan (e-mail: huiru.jiang@gmail.com).

Y.-W. Chang is with the Department of Electrical Engineering and Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan and the Research Center for Information and Technology Innovation, Academia Sinica, Nangang, Taipei 11529, Taiwan (e-mail: ywchang@cc.ee.ntu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2013.2272540

they are separated instead of clustered as arrays. However, a complex function may be realized by connecting several decap cells. As shown in Fig. 2(a), the internal connection among decap cells induces unwanted timing degradation. On the other hand, as shown in Fig. 2(b), selecting consecutive tiles in a spare array can realize a required function without timing degradation. They greedily fix timing violations from the gate with the maximum loading capacitance by configuring extra decap cells as resized gates or buffers. They do not have a global view to manage all decap cells, thus leaving some timing violating paths unfixed. Consequently, their greedy heuristic is suitable for small and local revisions.

Based on the above facts, using metal-configurable gate-array spare cells to accomplish ECO is practical and promising for modern IC designs. In addition, timing satisfaction is essential for ECO. In this paper, we first introduce a new problem of timing ECO optimization using metal-configurable gate-array spare cells, where timing violations are fixed by gate sizing and buffer insertion, implemented by configuring spare arrays.

This new ECO problem is quite different from the conventional one. There are two issues to consider for spare arrays: fragmentation and congestion. A fragmented spare array is adverse to ECO because unabutted free tiles may not form the required function. On the other hand, the input and output pins of the allocated cells within a spare array should be connected to external cells, so a high pin density within a spare array and long input/output nets may incur congestion. Hence, when realizing ECO using spare arrays, we shall keep spare arrays alive, the free tiles should be capable of implementing as many functions as possible, and routable, the congestion should be well-controlled.

To fully utilize the capability of spare arrays, we consider aliveness, routability, and timing satisfaction in our timing ECO optimization framework. We collect gates on timing violating paths and check if there exist gates whose timing can be improved. Most timing critical gates are extracted from these improvable gates. With a global view, we insert buffers or size the critical gates by appropriate spare arrays with aliveness, routability, and timing considerations. Finally, spare arrays are further packed to reduce wirelength. This procedure is repeated until no timing violations or no fixable gates can be found. We solve the spare array assignment and packing by mixed-integer linear programming (MILP).

On the other hand, functional rectification is prevalent in ECO. Based on the functional flexibility of spare arrays, we extend our MILP formulation to functional ECO optimization. Consider a set of ECO patches that describe the logic difference between the original design and the revised specification. We extract available spare arrays and generate technology remapping candidates for each patch. We then adequately configure spare arrays to implement these patches with aliveness and routability consideration. Similarly, we solve the spare array assignment and packing of functional ECO optimization by MILP.

Our contributions are summarized as follows.

1) We address new ECO problems. To overcome the inflexibility and power overhead of using standard cells as spare cells, we address new problems of timing and functional ECO optimization using metal-configurable gate-array spare cells.
2) We model the aliveness of spare arrays. To avoid fragmentation, we shall keep spare arrays alive. To achieve this goal, we propose the aliveness metric to model the capability of each spare array by a piecewise linear function, which can be incorporated into our MILP.
3) We adopt iterative MILP for timing ECO optimization. It might not be feasible to consider all gates on timing violating paths in one single MILP for large-scale designs. In addition, the estimated timing improvement may not be accurate enough, and thus more than one MILP may be needed. As a result, we use a set of independent and small MILPs to fix timing critical gates. Experimental results show that this reduction delivers superior efficiency and effectiveness.
4) We extend our MILP formulation to functional ECO optimization. Because of functional flexibilities, spare arrays are suitable for functional ECO optimization. Our MILP can handle spare array assignment and packing well not only for timing but also for functional ECO optimization.
5) We can simultaneously consider standard spare cells and spare arrays. In addition to spare arrays, a design typically contains standard spare cells released from earlier ECO runs or inserted along with spare arrays. Simple ECO tasks can be done by spare cells, while difficult ones can be realized by spare arrays. A standard spare cell can be viewed as a special spare array. Through adequate modifications, our framework is readily extended to handle a mix of standard spare cells and spare arrays.

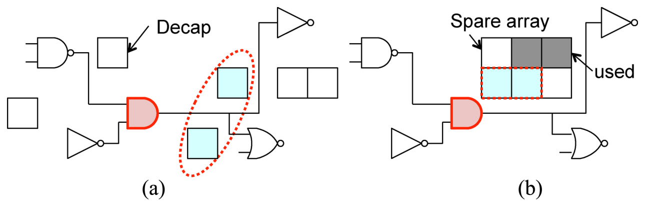


Fig. 2. Reconfigurable decap cells versus metal-configurable gate-array spare cells. Consider that the shaded AND cell is required to be resized by two decap cells or two tiles. (a) Additional wiring between decap cells incurs extra delay. (b) Tiles in a spare array are adequately selected to realize the gate sizing.

The remainder of this paper is organized as follows. Section II describes the cost metrics (including aliveness) and problem formulation for timing ECO optimization. Section III gives the generic and reduced MILP formulations for timing ECO using metal-configurable gate-array spare cells. Section IV presents our timing ECO optimization framework based on iterative MILP and demonstrates how to handle a mix of standard spare cells and spare arrays. Section V extends to functional ECO optimization. Section VI shows our experimental results. Section VII concludes this paper.

II. Problem Formulation and Cost Metrics

In this section, we introduce spare arrays, detail the cost metrics, and give the problem formulation of timing ECO optimization.

A. Spare Arrays

By combining gate arrays and structured ASICs [19], metal-configurable gate-array spare cells are proposed to provide a flexible resource to realize metal-only ECO [14]–[18]. As shown in Fig. 1(a), a spare array is an array of tiles. Each tile (base cell) is composed of unwired transistors, which consume no power. A designated functionality is formed by configuring the metal layers on top of several tiles.

To accommodate a sufficient and flexible resource of spare cells for metal-only ECO, spare arrays are scattered over the layout at the placement stage and/or filled into empty spaces after the placement stage [18]. Furthermore, to facilitate timing/power characterization and mask generation, the dimension of a certain cell type is predefined. Since an irregular shape might incur performance degradation, the shape of a functional cell is typically regular. In addition, to keep the same driving strength, a functional cell generated by spare array tiles is somewhat slower and less area-efficient than a standard spare cell. Even so, if such a functional cell is close to a gate that should be fixed, this cell is definitely faster than a standard spare cell far away because interconnect dominates gate delay. On the other hand, tie cells are necessary when standard spare cells are used; their area overhead should be considered as well. Hence, the area deficiency of spare arrays is minor.

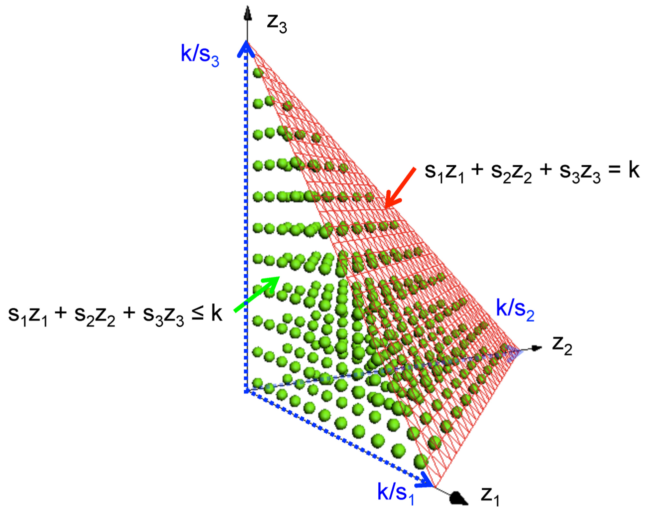
To facilitate metal-only ECO synthesis, a spare array cell library is constructed. As mentioned above, the cell dimension in current technology is set to multiple consecutive tiles in a single row [Fig. 1(b)]. For example, as shown in Fig. 1(c), an inverter uses one tile, while a flip-flop uses six tiles. In addition to the dimension, a spare array cell library also stores timing/power characteristics and layout of each cell type. The timing model is based on Synopsys' Liberty library [21]. The delay and output transition of a cell depend on its input transition and output capacitance, and these values are characterized by lookup tables. The output capacitance of a cell includes its output-pin capacitance, the input-pin capacitance of its fanout gates, and the wire loading. The wire loading is proportional to the wirelength of its output net.

B. Aliveness

We propose a new metric, aliveness, to model the capability of a spare array. Given a spare array cell library which contains m different sizes of functional cells; each size of the functional cells occupies s_i tiles. Consider a spare array with k free tiles. Let z_i denote the number of cells of size s_i that are implemented by this spare array, $z_i \in N$. We have

$$\begin{aligned} s_1z_1 + s_2z_2 + \cdots + s_mz_m &\leq k \\ z_i \geq 0, \forall 1 \leq i &\leq m. \end{aligned} \tag{1}$$

The linear equation $s_1z_1 + s_2z_2 + \cdots + s_mz_m = k$ defines a hyperplane in the m-dimensional space. The distance between the origin to the intersection point of this hyperplane and axis z_i is k/s_i. The hyperplane and m axes form a convex polytope. Each integer point inside the polytope represents a feasible combination of cell sizes that the spare array can implement. Fig. 3 shows an example for a library with three different cell sizes. Since a spare array is desired to be capable of implementing as many functions as possible, we model the aliveness of a spare array by the total number of integer points within this polytope. Furthermore, we use the volume of this polytope to approximate this number. Hence, aliveness is defined as follows.



z_3
k/s_3
$s_1z_1 + s_2z_2 + s_3z_3 = k$
$s_1z_1 + s_2z_2 + s_3z_3 \leq k$
k/s_2
z_2
k/s_1
z_1

Fig. 3. Aliveness. The aliveness of a spare array is modeled by the volume of the polytope defined by the hyperplane and the three axes.

Definition 1: The aliveness of a spare array is defined as the volume of the polytope given by inequality (1).

C. Routability

Congestion may incur unwanted detours, thus making the aforementioned wire loading computation inaccurate. Hence, it is necessary to consider routability during ECO. First, the layout is divided into uniform and non-overlapping bins to construct the routing grid graph. In the graph, a node represents a bin, and an edge connects each pair of adjacent bins. Each edge is associated with a routing capacity, which is the number of routing tracks available for nets passing through the corresponding bin boundary. The routability of an edge is computed by the difference between its capacity and the number of occupied tracks (density), while the routability of a bin is thus the total routability values on the edges of its boundaries. We adopt the routing model proposed by Hsu *et al.* in [24], since it is efficient yet sufficiently accurate. Each net is first decomposed into two-pin nets by FLUTE [22] which is a fast and accurate rectilinear Steiner minimal tree (RSMT) algorithm. Each two-pin net is then routed by upper-L and lower-L patterns with 50% probability for each direction [Fig. 4].

D. Problem Formulation

Different from standard spare cells, we shall avoid fragmentation and congestion when using spare arrays. Hence, we

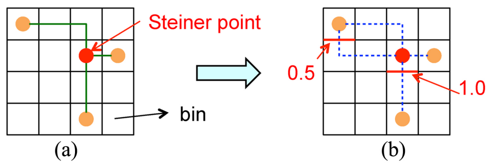


Fig. 4. Routability. (a) One net is decomposed into two-pin nets by FLUTE [22]. (b) Each two-pin net is routed by upper-L and lower-L patterns. Each direction has 50% probability.

consider aliveness and routability in our problem formulation. In addition, timing satisfaction is essential for ECO. In this paper, consequently, we first focus on timing ECO optimization. Typically, timing violations are fixed by gate sizing and buffer insertion. Gate sizing is an operation that changes the driving strength of some cell on a timing violating path, while buffer insertion is an operation that inserts a buffer along a timing violating path.

The timing ECO problem is thus formulated as follows.

Problem: Given a placed design, timing violating paths, the placement of spare arrays and the corresponding cell library, select tiles from spare arrays to perform gate sizing and/or buffer insertion such that the timing constraint is satisfied, and the aliveness and routability of spare arrays are maximized.

III. Single MILP Formulation for Timing ECO

In this section, we give two mixed integer linear programming (MILP) formulations of spare array assignment for timing ECO using metal-configurable gate-array spare cells.

A. Generic Single MILP

First of all, we show the generic single MILP formulation of spare array assignment for timing ECO using metal-configurable gate-array spare cells. Consider all timing violating paths and their related paths (the paths that fork or join with some timing violating path). The notation used in this MILP formulation is as follows.

1) G': set of gates on all timing violating and their related paths.
2) S: set of spare arrays, a spare array means a maximal set of consecutive free tiles in a single row.
3) H: set of timing violating paths and their related paths.
4) M_i: set of fanin gates of gate i.
5) $s^b_{i,j}$: required size of the inserted buffer if gate i is assigned to spare array j.
6) $s^g_{i,j}$: required size of the sized gate if gate i is assigned to spare array j
7) k_j: the number of free tiles of spare array j.
8) x^0_i: 0-1 variable indicating if gate i remains unmodified.
9) x^z_i: 0-1 variable indicating if gate i is a buffer and bypassed.
10) $x^b_{i,j}$: 0-1 variable indicating if gate i is assigned to spare array j and buffer insertion is performed.
11) $x^g_{i,j}$: 0-1 variable indicating if gate i is assigned to spare array j and gate sizing is performed.
12) a_j: aliveness of spare array j.
13) $r^b_{i,j}$: routability by assigning gate i to spare array j and performing buffer insertion.
14) $r^g_{i,j}$: routability by assigning gate i to spare array j and performing gate sizing.
15) r_j: routability contributed by spare array j.
16) t^r_h: timing requirement of path h.
17) d^0_i: original delay of gate i.
18) d^z_i: composed delay of gate i if gate i is a bypassed buffer.
19) $d^b_{i,j}$: composed delay of gate i by assigning gate i to spare array j and performing buffer insertion.
20) $d^g_{i,j}$: composed delay of gate i by assigning gate i to spare array j and performing gate sizing.
21) p^d_j: the pin density bound of spare array j.
22) p_i: pin count of gate i.
23) α, β: user-specified parameters used to trade between aliveness and routability. $\alpha + \beta = 1$.

Based on the above notation, the generic single MILP formulation can be written as follows:

$$\textit{maximize} \quad \alpha \sum_{j\in S} a_j + \beta \sum_{j\in S} r_j$$

$$\textit{subject to} \quad a_j = f(k_j - \sum_{i\in G'} (x^b_{i,j}s^b_{i,j} + x^g_{i,j}s^g_{i,j})), \forall j \in S \tag{2}$$

$$r_j = \sum_{i\in G'} (x^b_{i,j}r^b_{i,j} + x^g_{i,j}r^g_{i,j}), \forall j \in S \tag{3}$$

$$\sum_{i\in G'} (x^b_{i,j}s^b_{i,j} + x^g_{i,j}s^g_{i,j}) \le k_j, \forall j \in S \tag{4}$$

$$x^0_i + x^z_i + \sum_{j\in S} (x^b_{i,j} + x^g_{i,j}) = 1, \forall i \in G' \tag{5}$$

$$\sum_{i\in G'} (2x^b_{i,j} + x^g_{i,j}p_i) \le p^d_j, \forall j \in S \tag{6}$$

$$\sum_{j\in S} (x^b_{m,j} + x^g_{m,j}) + \sum_{j\in S} x^g_{i,j} \le 1, \forall i \in G', m \in M_i \tag{7}$$

$$\sum_{i\in h} (x^0_i d^0_i + x^z_i d^z_i + \sum_{j\in S} (x^b_{i,j}d^b_{i,j} + x^g_{i,j}d^g_{i,j})) \le t^r_h, \forall h \in H \tag{8}$$

$$x^0_i, x^z_i, x^b_{i,j}, x^g_{i,j} \in \{0, 1\}, \forall i \in G', j \in S. \tag{9}$$

The objective function is to maximize the weighted sum of aliveness and routability. If the input design is short of free tiles in spare arrays or will undergo many ECO runs, we prefer a larger α; if the input design is highly congested, we prefer a larger β. Equations (2) and (3) define the aliveness and routability of each spare array, respectively. Based on Definition 1, the aliveness of a spare array is defined by the volume of the polytope formed by inequality (1). To incorporate the volume computation into our MILP, we use a piece-wise linear function f to approximate the volume. Since the input and output pins of the allocated cells within a spare array should be connected to external cells, long input/output nets and a high pin density within a spare array may incur congestion. The routability r_j of a spare array j is used to optimize the total congestion values induced by its related input/output nets. $r^b_{i,j}$ (respectively, $r^g_{i,j}$) is computed by $c_{\max} - c^b_{i,j}$ (respectively, $c_{\max} - c^g_{i,j}$), where $c^b_{i,j}$ (respectively,

$c^g_{i,j}$) denotes the sum of congestion values of input/output nets induced by assigning gate i to spare array j for buffer insertion (respectively, gate sizing), and $c_{\max} = \max_{i,j} \max(c^b_{i,j}, c^g_{i,j})$. The congestion value of a net is the number of bin boundaries crossed based on FLUTE [22], e.g., the congestion value of the net shown in Fig. 4(a) is 6.0. Constraint (4) ensures that the total sizes of allocated cells do not exceed the number of free tiles for each spare array. Constraint (5) describes that exactly one of the following options is selected for each gate: unmodified, buffer bypassing, buffer insertion, or gate sizing. In addition to the routability optimized by the objective function, (6) limits the pin density of each spare array [24]. Based on the shielding effect[1] [2], (7) avoids performing buffer insertion and/or gate sizing on directly connected gates at the same time. Constraint (8) guarantees timing satisfaction for all timing violating paths and their related paths.

The number of paths considered in (8) is exponential to the gate count. Hence, the generic MILP formulation is impractical. In the subsequent subsection, we apply a reduction technique to reduce the complexity.

B. Reduced Single MILP

In this subsection, we adopt block-based timing analysis to reduce the MILP complexity. Most of variables used in the reduced single MILP formulation are the same as the generic one. We introduce the following three additional variables.

1) N: set of primary outputs or pseudo primary outputs (flip-flops) with timing violations.
2) t^r_i: timing requirement of gate i.
3) t^a_i: arrival time of gate i.

Therefore, the reduced single MILP formulation can be written as follows:

$$\begin{aligned}
&\textit{maximize} \quad \alpha \sum_{j\in S} a_j + \beta \sum_{j \in S} r_j \\
&\textit{subject to} \quad a_j = f\Big(k_j - \sum_{i\in G'} (x^b_{i,j}s^b_{i,j} + x^g_{i,j}s^g_{i,j})\Big), \forall j \in S && (10)\\
&r_j = \sum_{i\in G'} (x^b_{i,j}r^b_{i,j} + x^g_{i,j}s^g_{i,j}), \forall j \in S && (11)\\
&\sum_{i\in G'} (x^b_{i,j}s^b_{i,j} + x^g_{i,j}s^g_{i,j}) \le k_j, \forall j \in S && (12)\\
&x^0_i + x^z_i + \sum_{j\in S}(x^b_{i,j} + x^g_{i,j}) = 1, \forall i \in G' && (13)\\
&\sum_{i\in G'}(2x^b_{i,j} + x^g_{i,j}p_i) \le p^d_j, \forall j \in S && (14)\\
&\sum_{j\in S}(x^b_{m,j} + x^g_{m,j}) + \sum_{j\in S} x^g_{i,j} \le 1, \forall i \in G', m \in M_i && (15)\\
&t^a_m + x^0_i d^0_i + x^z_i d^z_i + \sum_{j\in S}(x^b_{i,j}d^b_{i,j} + x^g_{i,j}d^g_{i,j}) \le t^a_i \\
&\forall i \in G', m \in M_i && (16)\\
&t^a_i \le t^r_i, \forall i \in N && (17)\\
&x^0_i, x^z_i, x^b_{i,j}, x^g_{i,j} \in \{0,1\}, \forall i \in G', j \in S. && (18)
\end{aligned}$$

[1]The shielding effect means that sizing a gate or inserting a buffer influences only the delays of its fanin and fanout gates [2].

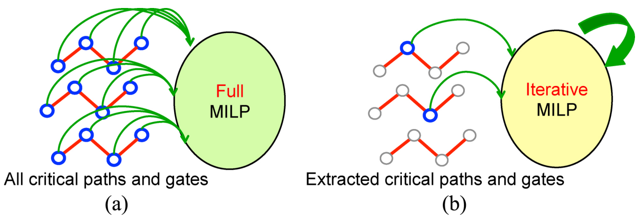

Full
MILP
All critical paths and gates
(a)
Iterative
MILP
Extracted critical paths and gates
(b)

Fig. 5. (a) Single MILP versus (b) iterative MILP.

Constraint (8) in the generic MILP formulation is transformed to (16) and (17). Based on block-based timing analysis, (16) describes that the arrival time of each gate i should be greater than or equal to the arrival time of its fanin gate plus its composed delay. Constraint (17) guarantees that the arrival time of the endpoint of each path satisfies the timing requirement. It can be seen that the number of variables and constraints considered in (16) and (17) are linear with the number of gates and wires. However, even with the reduction technique, our results show that the reduced single MILP formulation still exceeds the capability of a typical MILP solver [30].

IV. Our Timing ECO Optimization Framework

In this section, we detail our timing ECO optimization framework.

A. Overview

We propose our timing ECO framework to achieve timing closure using metal-configurable gate-array spare cells. To fully utilize the capability of spare arrays, we consider aliveness, routability, and timing satisfaction in our framework. As revealed in Section III, it might not be feasible to model all gates on timing violating paths into one single MILP for large-scale designs, due to the high complexity of MILP's. [Fig. 5(a)] Therefore, it is necessary to develop more effective reduction techniques to reduce the number of variables and constraints. In addition, the estimated timing improvement may not be accurate enough, and thus the optimality cannot be preserved and more than one MILP may be needed. As a result, we resort to iterative MILP in our timing ECO framework, where a set of independent and small MILPs are computed. [Fig. 5(b)] To minimize the deviation from the optimality, we try to reduce the number of iterations by relaxing the timing constraints to the objective function of MILPs. It is clear later that this approach delivers superior efficiency and effectiveness.

Fig. 6 gives the overview of our timing ECO framework. First, timing analysis reports timing violating paths. Second, we collect gates on these paths and check if there exist gates whose timing can be improved by gate sizing or buffer insertion. Third, timing critical gates are extracted from these fixable gates. Fourth, we assign only extracted critical gates (instead of all gates on timing violating paths and their related paths) to appropriate spare arrays with aliveness, routability, and timing considerations. Fifth, spare arrays are further packed to reduce wirelength. This procedure is repeated until no timing violations or no fixable gates can be found.

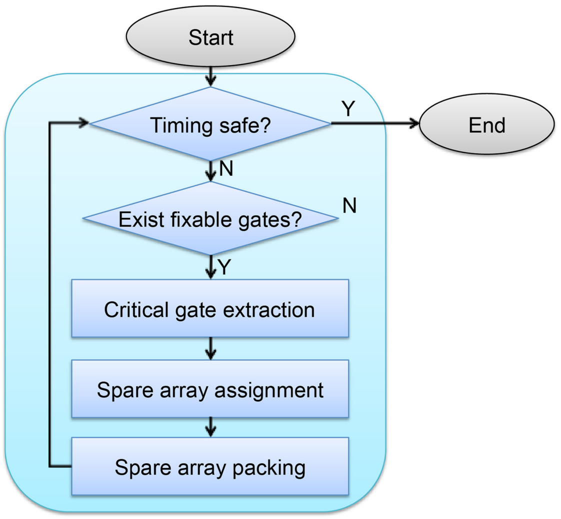


Fig. 6. Overview of our timing ECO framework.

B. Fixable and Critical Gate Extraction

To effectively reduce the problem size for MILP, we extract only timing critical gates from timing violating paths. Chang *et al.* propose a new metric of timing criticality—fixability—which can accurately identify the most timing critical gates along the timing violating paths [11]. Hence, in this paper, we extract the timing critical gates based on fixability.

First of all, we check if there exist gates on the timing violating paths whose timing can be improved by gate sizing or buffer insertion. Considering a gate and a spare array, the configuration (either gate sizing or buffer insertion) resulting in the best timing improvement is recorded for subsequent steps. Second, we calculate the fixability for these fixable gates. Finally, timing critical gates are extracted from these fixable gates based on the method proposed in [11].

To further reduce the problem size, for each investigated gate, we consider only the spare arrays located inside its bounding polygon. The bounding polygon defined in [2] specifies a search region so that the spare arrays outside the bounding polygon are negligible. In addition, the query of spare arrays is accelerated by using R-trees [23] to efficiently categorize the neighboring resource.[2]

C. Spare Array Assignment

As shown in Fig. 7, after extracting critical gates, we assign appropriate spare arrays to perform gate sizing or buffer insertion for these gates. Since we adopt iterative MILP, we relax the timing satisfaction constraint to the objective function. Therefore, at each iteration, the objective function of spare array assignment is to maximize the aliveness, routability, and timing improvement.

The complete notation used in the iterative MILP formulation for spare array assignment is listed as follows.

1) G: set of critical gates.
2) S: set of spare arrays, a spare array means a maximal set of consecutive free tiles in a single row.

[2]An R-tree is a height-balanced tree which is widely used for indexing spatial objects such as points, rectangles, or polygons.

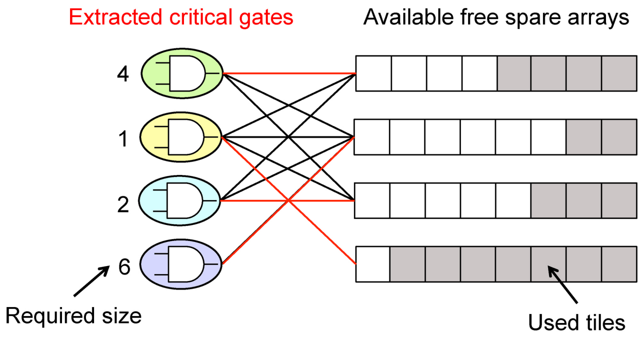


Fig. 7. Spare array assignment.

3) $s_{i,j}$: required size of either the sized gate or the inserted buffer if gate i is assigned to spare array j; this value is obtained from critical gate extraction described in Section IV-B.
4) k_j: the number of free tiles of spare array j.
5) $x_{i,j}$: 0-1 variable indicating if critical gate i is assigned to spare array j. For critical gate i, only the spare arrays within its bounding polygon are considered.
6) a_j: aliveness of spare array j.
7) $r_{i,j}$: routability by assigning gate i to spare array j.
8) r_j: routability contributed by spare array j.
9) $t_{i,j}$: timing improvement by assigning gate i to spare array j.
10) t_j: timing improvement contributed by spare array j.
11) p_j^d: the pin density bound of spare array j.
12) p_i: pin count of gate i. $p_i = 2$ if buffer insertion is performed.
13) α, β, γ: user-specified parameters used to trade among aliveness, routability, and timing improvement. $\alpha+\beta+\gamma=1$.

Based on the above notation, the spare array assignment problem can be formulated as follows:

$$\begin{aligned}
&\textit{maximize} && \alpha\sum_{j\in S} a_j + \beta\sum_{j\in S} r_j + \gamma\sum_{j\in S} t_j \\
&\textit{subject to} && a_j = f(k_j - \sum_{i\in G} x_{i,j}s_{i,j}), \forall j \in S && (19)\\
& && r_j = \sum_{i\in G} r_{i,j}, \forall j \in S && (20)\\
& && t_j = \sum_{i\in G} t_{i,j}, \forall j \in S && (21)\\
& && \sum_{i\in G} x_{i,j}s_{i,j} \le k_j, \forall j \in S && (22)\\
& && \sum_{j\in S} x_{i,j} = 1, \forall i \in G && (23)\\
& && \sum_{i\in G} x_{i,j}p_i \le p_j^d, \forall j \in S && (24)\\
& && x_{i,j} \in \{0,1\}, \forall i \in G, j \in S. && (25)
\end{aligned}$$

The objective function is to maximize the weighted sum of aliveness, routability, and timing improvement. If the input design is short of free tiles in spare arrays or will undergo many ECO runs, we prefer a larger α; if the input design is highly congested, we prefer a larger β; if the input design

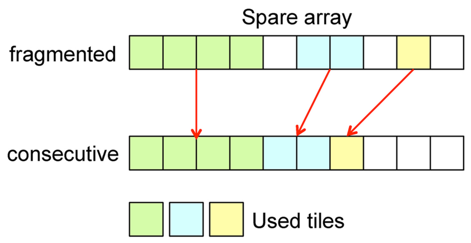


Fig. 8. Fragmented versus consecutive packing. A fragmented spare array is adverse to ECO.

incurs large timing violations, we prefer a larger γ. Based on our experiments, the desirable ranges of parameters are reasonably large. Hence, we may set $\alpha = \beta = \gamma$ by default. Equations (19), (20), and (21) define the aliveness, routability, and timing improvement of each spare array, respectively. Similar to (2), we use a piece-wise linear function f to approximate the volume of the polytope formed by Inequality (1). Similar to (3), the routability r_j of spare array j is used to optimize the total congestion values induced by its related input/output nets. $r_{i,j}$ is either $r^g_{i,j}$ or $r^b_{i,j}$ according to the configuration recorded for assigning gate i to spare array j. Thus, $r_{i,j}$ is computed by $c_{\max} - c_{i,j}$, where $c_{i,j}$ denotes the sum of congestion values of input/output nets induced by assigning gate i to spare array j, and $c_{\max} = \max_{i,j} c_{i,j}$. The timing improvement $t_{i,j}$, relaxed from the timing satisfaction constraint, is computed assuming the resized gate or inserted buffer i is located at the center of spare array j. Similar to (4), (22) ensures that the total sizes of allocated cells do not exceed the number of free tiles for each spare array. Constraint (23) guarantees that each critical gate is assigned to exactly one spare array. Similar to (6), in addition to the routability optimized by the objective function, (24) limits the pin density of each spare array [24]. It can be seen that because only critical gates are considered in the iterative MILP formulation for spare array assignment, the problem size is greatly reduced compared with the generic and reduced single MILP formulations described in Section III. By introducing timing improvement to the objective function, we can reduce the number of iterations and thus minimize the deviation from the optimality. Later, our results show that the number of iterations of our iterative MILP is quite small, and the overall runtime is short.

D. Spare Array Packing

After spare array assignment, each spare array is assigned with a set of critical gates to which gate sizing or buffer insertion will be applied. Spare array packing further determines the actual tiles allocated for each of these critical gates with minimum wirelength.

Since a spare array is much smaller than a bin size, different packings do not influence routability, but affect aliveness and wire loading. First of all, as shown in Fig. 8, if allocation is not in continuity, there is some gap between allocated tiles. Consecutive allocation is better in terms of aliveness, and thus we have the following property that can be proved by an exchange argument.

Theorem 1: There is an aliveness optimal packing with no fragmentation.

Proof: Without loss of generality, assume a fragmented spare array with two groups of free tiles is aliveness optimal. These two groups have k_1 and k_2 free tiles, and k_3 allocated tiles are in between them. Consider a new packing for this spare array where k_3 allocated tiles are swapped with k_2 free tiles. The new packing has $k = k_1 + k_2$ free tiles. The alivenss of the new packing is as follows.

$$f(k) \geq f(k_1) + f(k_2). \tag{26}$$

Therefore, the new packing is also aliveness optimal. ■

Based on the property, we adopt consecutive allocation for spare array packing. Furthermore, we formulate an MILP for each spare array to determine the actual allocation with wirelength minimization. For spare array $j*$, we pack the tiles for the gates whose $x_{i,j*} = 1$ after spare array assignment. The notation used in the MILP formulation for spare array packing is as follows.

1) G: set of critical gates assigned to the investigated spare array.
2) T: set of free tiles in the investigated spare array.
3) s_i: the required size of gate i. s_i is determined by spare array assignment, $s_i = \sum_j s_{i,j} x_{i,j}$.
4) $y_{i,j}$: 0-1 integer variable that denotes if gate i is assigned to tile j. When $y_{i,j} = 1$, gate i occupies tiles $j, j+1, \ldots, j+s_i-1$.
5) $w_{i,j}$: wirelength for gate i assigned to tile j. $w_{i,j}$ counts the external connections induced by input/output nets since the wirelength between tiles within the same spare array is negligible.

Based on the above notation, the spare array packing problem can be written as follows:

$$\begin{aligned} &\textit{minimize} \sum_{i \in G} \sum_{j \in T} w_{i,j} y_{i,j} \\ &\textit{subject to} \sum_j y_{i,j} = 1, \forall i \in G \end{aligned} \tag{27}$$

$$\sum_{i \in G} \sum_{k=j-s_i+1}^{j} y_{i,k} = 1, \forall j = 1, \ldots, \sum_{i \in G} s_i \tag{28}$$

$$\sum_{i \in G} \sum_{k=j-s_i+1}^{j} y_{i,k} = 0, \forall j = \sum_{i \in G} s_i + 1, \ldots, |T|. \tag{29}$$

The objective function is to minimize the total wirelength. Constraint (27) ensures that each critical gate is assigned to exactly one position. Constraints (28) and (29) guarantee that each tile is occupied by at most one gate. Based on Theorem 1, tiles 1 to $\sum_{i \in G} s_i$ should be occupied, and from tile ($\sum_{i \in G} s_i +$ 1) to tile $|T|$ should not be occupied. When $y_{i,j} = 1$, gate i occupies tiles $j, j+1, \ldots, j+s_i-1$, as shown in Fig. 9. Hence, tile j is occupied by gate i if $y_{i,j-s_i+1} = 1, \ldots,$ or $y_{i,j} = 1$.

E. Extension to Mixed Standard Spare Cells and Spare Arrays

In this subsection, we discuss how to extend our timing ECO framework to utilize a mix of standard spare cells and spare arrays. Standard spare cells and spare arrays may co-exist in a

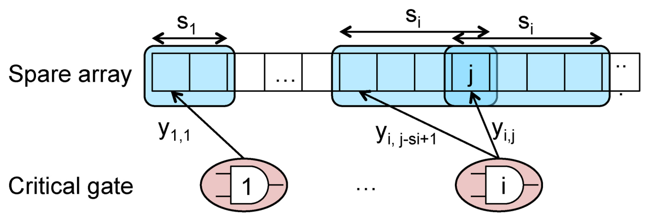


Fig. 9. Spare array packing. Each tile can be occupied by at most one gate.

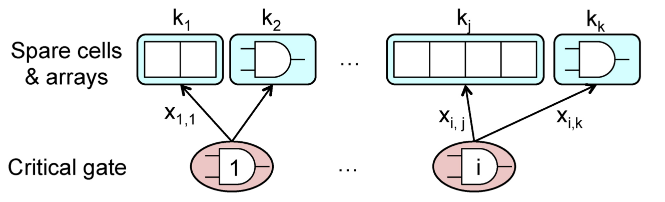


Fig. 10. Spare array assignment considering standard spare cells and spare arrays. A standard spare cell can be viewed as a special spare array with a fixed functionality.

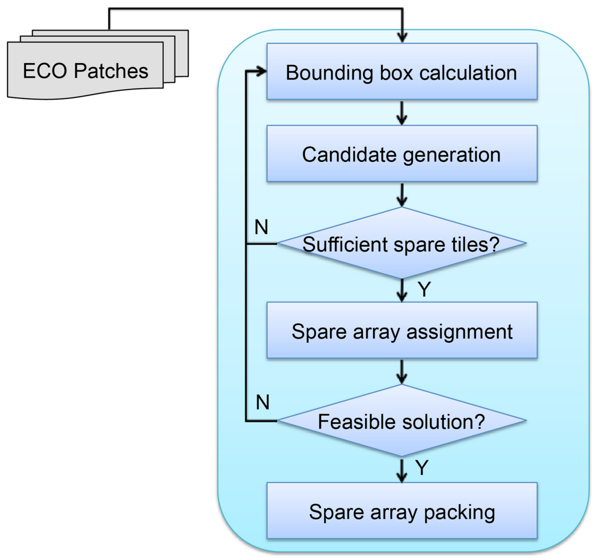


Fig. 11. Overview of our functional ECO framework.

design to afford resources for ECO. In some designs, standard spare cells may be inserted along with spare arrays at the placement stage. In addition, a design might need to undergo many ECO runs. Hence, some standard cells are freed up by ECO. At later ECO runs, the freed-up cells can be recycled and serve as spare cells. Simple ECO tasks can be done by spare cells, while difficult ones can be realized by spare arrays.

Basically, a standard spare cell can be viewed as a special spare array with a fixed functionality. Hence, for spare array assignment, as shown in Fig. 10, if spare cell k matches the configuration determined for gate i (Section IV-B), $x_{i,k}$ is created, and $s_{i,k} = k_k = k$'s cell area. Otherwise, we delete $x_{i,k}$ or set $x_{i,k} = 0$. For spare array packing, selected standard spare cells do not need packing. At this step, only spare arrays are processed. Based on the above modifications, our framework is readily extended to handle a mix of standard spare cells and spare arrays.

V. Extension to Functional ECO Optimization

Metal-only functional ECO is a practical and effective process for functional rectification and/or specification revision in modern IC design flow. However, conventional standard spare cells are limited in functionality, quantity, and location. This inflexibility may result in unfixed functional failures due to a shortage of proper spare cells. In contrast, spare arrays can flexibly be configured to designated functionalities. This flexibility makes spare arrays as superior resources for metal-only functional ECO. In this section, we extend our work to functional ECO optimization.

A. Problem Formulation and Overview

The functional ECO problem is formulated as follows.

Problem: Given a placed design, ECO patches, the placement of spare arrays and the corresponding cell library, select tiles from spare arrays to perform functional changes such that the aliveness and routability of spare arrays are maximized.

The ECO patches describe the logic difference between the original design and the revised functionality/specification. These patches are generated by logic difference tools [25]–[28]. Similar to timing ECO, we consider aliveness and routability in our functional ECO framework. Fig. 11 shows the overview of our functional ECO framework. First, the bounding box of each patch is calculated to collect available spare arrays. Second, resynthesis and technology remapping are performed for each patch to generate multiple remapping candidates. Third, the candidate gates are assigned to appropriate spare arrays with aliveness and routability considerations. Fourth, spare arrays are further packed to reduce wirelength. If the MILP fails to find a feasible spare array assignment, we enlarge the bounding box and generate more remapping candidates of each patch. This procedure is iterated until a feasible solution is found or no more solution candidates can be generated.

The following subsections detail the first three steps of our functional ECO framework. Here, spare array packing is the same as timing ECO. Based on the modifications made in Section IV-E, we can also handle functional ECO using a mix of standard spare cells and spare arrays.

B. Bounding Box Calculation

To reduce the problem size, we define the bounding box of an ECO patch and then consider only the spare arrays inside the bounding box.

Definition 2: The bounding box of an ECO patch is defined as the bounding box covering the primary inputs and primary outputs of this ECO patch.

Fig. 12 shows an example with two ECO patches, u_1 and u_2. Only the spare arrays inside the union of all bounding boxes will be considered in the following steps.

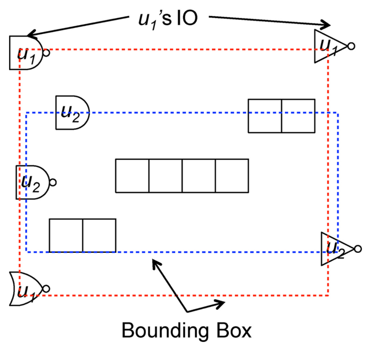


Fig. 12. Bounding boxes of ECO patches.

C. Candidate Generation

We adopt ABC [29] to perform technology remapping for each ECO patch according to the spare array cell library. We force ABC to restructure the AIG of each ECO patch, and thus several remapping solution candidates are generated. For example, in Fig. 13, there are two ECO patches, and each patch has two solution candidates. Patch u_1 can be remapped to candidate v_1 (one AND plus one NAND gate) or candidate v_2 (one INV and two NAND gates).

After that, to ensure that there are sufficient spare tiles, we check if the maximum number of required tiles is greater than the number of free tiles. If free tiles are insufficient, the bounding boxes will be enlarged to collect more available spare tiles. This procedure is repeated until sufficient spare tiles are collected or the union of bounding boxes reaches the circuit size.

D. Spare Array Assignment for Functional ECO

We solve the spare array assignment problem for functional ECO optimization by one MILP. The notation used in this MILP formulation is as follows.

1) G: set of gates in all solution candidates generated by technology remapping.
2) S: set of spare arrays, a spare array means a maximal set of consecutive free tiles in a single row.
3) U: set of ECO patches.
4) V_u: set of solution candidates of ECO patch u.
5) $s_{u,v,i}$: required size of gate i which belongs to patch u's solution candidate v.
6) k_j: the number of free tiles of spare array j.
7) $x_{u,v}$: 0-1 variable indicating if patch u is assigned to solution candidate v.
8) $x_{u,v,i,j}$: 0-1 variable indicating if gate i which belongs to patch u's solution candidate v is assigned to spare array j.
9) a_j: aliveness of spare array j.
10) $r_{u,v,i,j}$: routability by assigning gate i which belongs to patch u's solution candidate v to spare array j.
11) r_j: routability contributed by spare array j.
12) p^d_j: the pin density bound of spare array j.
13) $p_{u,v,i}$: pin count of gate i which belongs to patch u's solution candidate v.

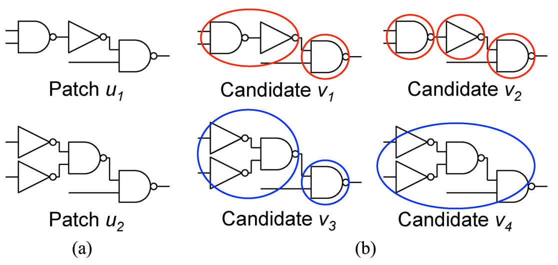


Fig. 13. Example of ECO patches.

14) α, β: user-specified parameters used to trade between aliveness and routability. $\alpha + \beta = 1$.

Based on the above notation, the spare array assignment problem for functional ECO optimization can be formulated as follows:

$$\begin{aligned}
&\textit{maximize} \quad \alpha \sum_{j\in S} a_j + \beta \sum_{j \in S} r_j \\
&\textit{subject to} \quad a_j = f(k_j - \sum_{i\in G} x_{u,v,i,j} s_{u,v,i}), \forall j \in S && (30)\\
&r_j = \sum_{i \in G} x_{u,v,i,j} r_{u,v,i,j}, \forall j \in S && (31)\\
&\sum_{i\in G} x_{u,v,i,j} p_{u,v,i} \le p^d_j, \forall j \in S && (32)\\
&\sum_{v \in V_u} x_{u,v} = 1, \forall u \in U && (33)\\
&\sum_{j\in S} x_{u,v,i,j} = x_{u,v}, \forall i \in G && (34)\\
&\sum_{i\in G} x_{u,v,i,j} s_{u,v,i} \le k_j, \forall j \in S && (35)\\
&x_{u,v} \in \{0,1\}, \forall u \in U, v \in V_u && (36)\\
&x_{u,v,i,j} \in \{0,1\}, \forall u \in U, v \in V_u, i \in G, j \in S. && (37)
\end{aligned}$$

The objective function is to maximize the weighted sum of aliveness and routability. If the input design is short of free tiles in spare arrays or will undergo many ECO runs, we prefer a larger α; if the input design is highly congested, we prefer a larger β. Based on our experiments, the desirable ranges of parameters are reasonably large. Hence, we may set $\alpha = \beta$ by default. Similar to the timing ECO framework, (30) and (31) define the aliveness and routability, respectively. Constraint (32) limits the pin density of each spare array. Constraint (33) ensures that exactly one solution candidate is selected for each ECO patch. Constraint (34) guarantees that each gate is assigned to exactly one spare array if the corresponding solution candidate is selected. Constraint (35) ensures that the total size of allocated cells do not exceed the number of free tiles for each spare array.

VI. Experimental Results

Our approach was implemented in the C++ programming language on a platform with a 2.53 GHz Intel Core 2 Duo T9400 CPU and 4 GB memory. The CPLEX [30] was applied to solve the formulated MILPs. We did two sets of

TABLE I
COMPARISON BETWEEN [20] AND OUR FRAMEWORK

Circuit name	Initial								Greedy [20]					Ours					
	Gate count	Spare arrays	Cycle (ns)	#Critical paths	Max #gate	#Gate passed	WNS (ns)	TNS (ns)	TNS (ns)	#Ite.	Runtime(s) CPU	STA	Total	TNS (ns)	#Ite.	Runtime(s) ILP	CPU	STA	Total
Industry1	28,927	1.0%	38	16	164	2,604	1.1	9.8	0.0	1	0.44	0.29	0.73	0.0	1	0.43	0.55	0.49	1.47
Industry2	200,504	0.5%	40	80	178	13,627	10.8	312.0	0.0	32	65.29	2.02	64.31	0.0	2	4.18	4.19	2.44	10.80
Industry3	91,107	0.5%	37	27	173	4,059	19.3	319.0	8.7	542	327.68	0.90	328.57	0.0	2	4.26	1.34	1.13	6.72
Industry4	18,932	1.0%	18	22	85	1,278	6.8	70.0	17.4	366	0.51	1.02	1.53	0.0	1	0.74	0.31	0.18	1.23
Industry5	38,011	0.5%	18	137	72	9,160	2.8	161.0	76.3	3,054	254.78	0.75	255.53	0.0	27	9.68	8.82	1.07	19.56
Ratio													16.43						1.00

TABLE II
ILP STATISTICS OF TIMING ECO

Circuit	Generic Single MILP			Reduced Single MILP				Ours	
name	#Path	#Constraint	#Variable	#Node	#Edge	#Constraint	#Variable	#Constraint	#Variable
Industry1	>3,129,688	-	-	20,775	33,870	114,060	63,886,516	2,751	3,875
Industry2	>2,229,778	-	-	60,920	99,053	>3,390,520	>376,328,130	11,422	16,089
Industry3	>2,099,791	-	-	21,614	34,830	>1,237,160	>75,500,300	14,656	20,646
Industry4	5,508	16,460	2,507,511	1,447	1,914	9,228	2,509,403	9,289	13,082
Industry5	32,836	90,405	13,932,274	12,052	16,199	56,762	13,913,746	8,607	4,011

TABLE III
FUNCTIONAL ECO COMPARISON BETWEEN [8] AND OUR FRAMEWORK

Case name	Statistics				[8] ($\alpha = 0.5$)		Ours ($\alpha = 0.5$)		Ours ($\alpha = 1.0$)		Ours ($\alpha = 0.0$)	
	#Patch	#Candidate gate	Spare arrays	Gate count	A / R	Runtime (s)	A / R	Runtime (s)	A / R	Runtime (s)	A / R	Runtime (s)
Case1	2	15	1.0%	28,591	13,218 / 965	1.24	13,536 / 946	1.80	13,602 / 470	1.44	13,414 / 1000	1.66
Case2	1	435	0.3%	28,591	66 / 23,276	33.36	146 / 24,599	7.85	294 / 21,882	8.52	12 / 24,631	4.23
Case3	3	500	0.5%	28,591	1,150 / 3,169	366.30	7,782 / 3,573	292.24	7,798 / 2,935	512.96	7,636 / 3,579	12.22
Ratio					0.77	1.33	1.00	1.00				

A: Aliveness, R: Routability, $\beta = 1.0 - \alpha$.

experiments to evaluate our timing ECO and functional ECO frameworks.

A. Timing ECO Optimization

The experiments were conducted with five industrial designs, where the spare arrays are uniformly filled into the layout. Because of the flexibility of spare arrays, only 0.5% to 1% of the chip area is occupied by spare arrays. (Note that this ratio is very low, compared with 2% to 5% for standard spare cells.) Each spare array contains 3×10 tiles. A cell in the spare array cell library incurs an average 20% timing degradation compared with the cell in the standard cell library with the same driving strength. The statistics of these circuits are summarized in Table I, including the benchmark name (Circuit name), the number of gates in each design (Gate count), the number of available spare cells (#Spare cells), the clock period (Cycle), the number of timing violating paths (#Critical paths), the total number of gates passed by the critical paths (#Gate passed), the worst negative slack (WNS), and the total negative slack (TNS).

We tried to feed the generic and reduced single MILP formulations into CPLEX, but they both failed to generate solutions due to out of memory/time problems. Table II lists the statistics of generic single MILP, reduced single MILP and iterative MILP, including the number of related paths in each design (#Path), the number of nodes (#Node), the number of edges (#Edge), the number of constraints (#Constraint), and the number of variables (#Variable). It can be seen that the complexity of single MILPs is extremely large compared with the iterative MILP.

We implemented the greedy heuristic proposed by Chen *et al.* in [20] with adequate modifications to handle the spare arrays. Chen *et al.* reprogram decap cells to fix as many timing violations as possible under the IR drop constraint. For fair comparison, we remove the IR drop constraint and use spare arrays instead of decap cells. In our implementation, all free tiles in spare arrays can be used to fix timing violations. For each timing violating path, they greedily fix timing from the gate with the largest output loading. At each iteration, for the investigated gate, they find the spare array that can improve the delay of this gate best (either by gate sizing or by buffer insertion). The search region is set to the bounding box defined by this gate and its fanin/fanout gates. They then configure the selected spare array and perform the incremental static timing analysis (STA). Because of the greedy assignment, some timing violations cannot be fixed well because the best candidates have been occupied in previous iterations.

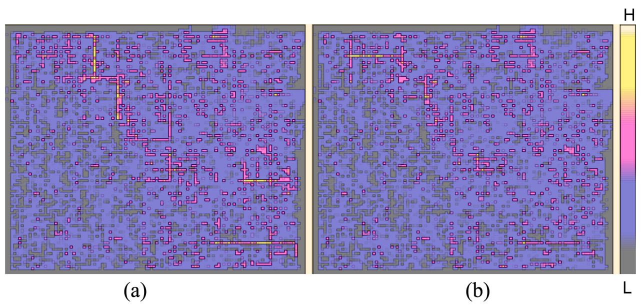


Fig. 14. Routing congestion map for Industry5 after timing ECO is applied. H (L) indicates high (low) congestion. (a) Without routability consideration. (b) With routability consideration.

Table I compares the above heuristic [20] with our iterative MILP in terms of the resulting TNS, the number of iterations (#Ite.), and the running times (Runtime). "STA" means the runtime consumed by incremental STA, "ILP" means the runtime used by MILP, and "CPU" means the runtime consumed by the remaining part. For [20], "#Ite" represents the number of iterations actually applied, while for our approach, "#Ite." represents the number of MILPs generated. Although not shown here, based on our experiments, the desirable ranges of parameters are reasonably large. Hence, we set $\alpha = \beta = \gamma$ in these experiments. First of all, we can successfully fix all timing violations because we consider aliveness during ECO optimization, while [20] fails for Industry3, Industry4, and Industry5 because their method does not have a global view to manage all spare arrays. Second, our iterative MILP is very efficient, achieving a 16.43X speedup. Fig. 14 shows the routing congestion map for Industry5. It can be seen that ECO with routability consideration indeed results in better routability, thus facilitating subsequent rewiring.

B. Functional ECO Optimization

For functional ECO, one industrial design was used. We applied three different sets of functional ECO patches to the industrial design. Table III shows the statistics of the ECO patches, including the number of patches (#Patch), the total number of candidate gates of these patches (#Candidate gate), the ratio of spare arrays spread in the design (Spare arrays), and the gate count of the design (Gate count). Among the three cases, Case1 demonstrates the situation when the functional ECO is applied on the congested area, Case2 shows an example when the number of available spare arrays is low after many ECO runs, and Case3 illustrates a larger case.

We implemented the state-of-the-art work [8] with proper modifications as the baseline. For each ECO patch, the work [8] generates a solution candidate and allocates nearest spare arrays. Then, pair-wise swapping between allocated spare arrays is applied for better aliveness and routability. If the candidate is infeasible, another candidate is generated and the procedure is repeated. As listed in Table III, our proposed functional ECO framework outperforms [8] with average 23% A/R improvement and 1.33X speedups. Interestingly, the runtime of Case2 and Case3 by [8] are somewhat longer because of the time-consuming pair-wise swapping.

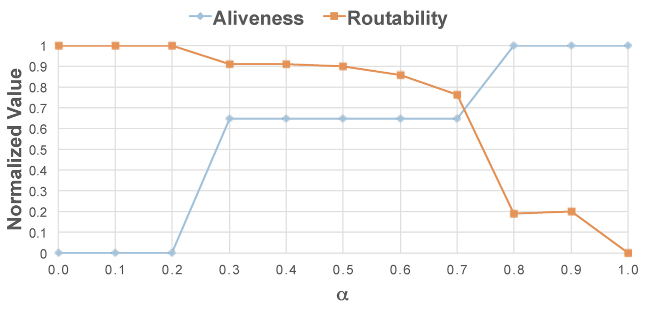


Fig. 15. Aliveness and routability under different parameter setting for functional ECO Case1.

Table III also compares three settings of the weight of aliveness and routability for each case, [considering both aliveness and routability ($\alpha = 0.5$), considering only aliveness ($\alpha = 1.0$), and considering only routability ($\alpha = 0.0$)]. It can be seen that our functional ECO framework can solve Case1 and Case2 pretty well. Case3 takes longer to be finished, but the runtime is still acceptable. Fig. 15 shows the impact of different parameter settings on aliveness and routability for functional ECO Case1. It can be seen that the desirable ranges of parameters are reasonably large. Hence, we set $\alpha = \beta = 0.5$ in our experiments.

VII. Conclusion

Traditionally, preplaced redundant standard cells are regarded as spare cells for metal-only ECO. To overcome the inflexibility and power overhead, we introduced a new problem of ECO optimization using metal-configurable gate-array spare cells. We first studied the properties for this new ECO problem and proposed the aliveness metric to model the capability of a spare gate array. We then adoptd iterative MILP to solve the new timing ECO problem with aliveness, routability, and timing consideration. We further extended to consider functional ECO optimization using spare arrays. Moreover, our ECO frameworks were readily extended to handle a mix of standard spare cells and spare arrays. Experimental results showed that our approach delivered superior efficiency and effectiveness.

References

[1] A. Balasinski, "Optimization of sub-100-nm designs for mask cost reduction," *SPIE J. Micro/Nanolith. MEMS MOEMS*, vol. 3 no. 2, pp. 322–331, Apr. 2004.

[2] K.-H. Ho, Y.-P. Chen, J.-W. Fang, and Y.-W. Chang, "ECO timing optimization using spare cells and technology remapping," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 29, no. 5, pp. 697–710, May 2010.

[3] S.-Y. Fang, T.-F. Chien, and Y.-W. Chang, "Redundant-wires-aware ECO timing and mask-cost optimization," in *Proc. ICCAD*, Nov. 2010, pp. 381–386.

[4] K.-H. Ho, J.-H. R. Jiang, and Y.-W. Chang, "TRECO: Dynamic technology remapping for timing engineering change orders," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 31, no. 11, pp. 1723–1733, Nov. 2012.

[5] Y.-M. Kuo, Y.-T. Chang, S.-C. Chang, and M. Marek-Sadowska, "Spare cells with constant insertion for engineering change," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 28, no. 3, pp. 456–460, Mar. 2009.

[6] N.A. Modi and M. Marek-Sadowska, “ECO-map: Technology remapping for post-mask ECO using simulated annealing,” in *Proc. ICCD*, Oct. 2008, pp. 652–657.

[7] I. H.-R. Jiang and H.-Y. Chang, “ECOS: Stable matching based metal-only ECO synthesis,” *IEEE Trans. Very Large Scale (VLSI) Syst.*, vol. 20, no. 3, pp. 485–497, Mar. 2012.

[8] S.-L. Huang, C.-A. Wu, K.-F. Tang, C.-H. Hsu, and C.-Y. Huang, “A robust ECO engine by resource-constraint-aware technology mapping and incremental routing optimization,” in *Proc. ASP-DAC*, Jan. 2011, pp. 382–387.

[9] C.-P. Lu, M. C.-T. Chao, C.-H. Lo, and C.-W. Chang, “A metal-only-ECO solver for input-slew and output-loading violations,” *IEEE Trans. on Computer-Aided Design Integr. Circuits Syst.*, vol. 29, no. 2, pp. 240–245, Feb. 2010.

[10] H.-Y. Chang, I. H.-R. Jiang, and Y.-W. Chang, “Simultaneous functional and timing ECO,” in *Proc. DAC*, Jun. 2011, pp. 140–145.

[11] H.-Y. Chang, I. H.-R. Jiang, and Y.-W. Chang, “Timing ECO optimization via Bézier curve smoothing and fixability identification,” *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 31, no. 12, pp. 1857–1866, Dec. 2012.

[12] X. Wei, W.-C. Tang, Y. Diao, and Y.-L. Wu, “ECO timing optimization with negotiation-based re-routing and logic re-structuring using spare cells,” in *Proc. ASP-DAC*, Jan. 2012, pp. 511–516.

[13] H.-Y. Chang, I. H.-R. Jiang, and Y.-W. Chang, “Timing ECO optimization using metal-configurable gate-array spare cells,” in *Proc. DAC*, Jun. 2012, pp. 802–807.

[14] T. Petit (STMicroelectronics), “Important ECOs implementation using gate-array-like mask configurable cells,” in *Proc. Cadence CDNLive! EMEA User Conf.*, May. 2011.

[15] ARM Artisan Physical IP, SC12 Std. Cell Library ECO Kit, and High Performance TSMC 40nm G.

[16] L. Ciccarelli, L. Cali, M. Innocenti, C. Mucci, V. Nardone, M. Pizzotti, and P. Rohilla, “Base cell for engineering change order (ECO) implementation,” U.S. Patent 7,965,207 B2, Jun. 2011.

[17] L.-C. Tien, “Method for reducing layers revision in engineering change order,” U.S. Patent 7,137,094 B2, Nov. 2006.

[18] G. S. Tsapepas, D. A. Webber, and M. H. Wood, “Spare gate array cell distribution analysis,” U.S. Patent 7,676,776 B2, Mar. 2010.

[19] K.-C. Wu and Y.-W. Tsai, “Structured ASIC, evolution or revolution?,” in *Proc. ISPD*, Apr. 2004, pp. 103–106.

[20] H.-T. Chen, C.-C. Chang, and T. Hwang, “New spare cell design for IR drop minimization in engineering change order,” in *Proc. DAC*, Jul. 2009, pp. 402–407.

[21] “Liberty: The EDA library modeling standard,” [Online]. Available: http://www.opensourceliberty.org/

[22] C. Chu and Y.-C. Wong, “FLUTE: Fast lookup table-based rectilinear Steiner minimal tree algorithm for VLSI Design,” *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 27, no. 1, pp. 70–83, Jan. 2008.

[23] A. Guttman, “R-trees: A dynamic index structure for spatial searching,” in *Proc. SIGMOD*, Jun. 1984, pp. 47–57.

[24] H.-K. Hsu, S. Chou, T.-H. Lin, and Y.-W. Chang, “Routability-driven analytical placement for mixed-size circuit designs,” in *Proc. ICCAD*, Nov. 2011, pp. 80–84.

[25] S. Krishnaswamy, H. Ren, N. A. Modi, and R. Puri, “DeltaSyn: An efficient logic difference optimizer for ECO synthesis,” in *Proc. ICCAD*, Nov. 2009, pp. 789–796.

[26] B.-H. Wu, C.-J. Yang, C.-Y. Huang, and J.-H. R. Jiang, “A robust functional ECO engine by SAT proof minimization and interpolation techniques,” in *Proc. ICCAD*, Nov. 2010, pp. 729–734.

[27] K.-F. Tang, C.-A. Wu, P.-K. Huang, and C.-Y. Huang, “Interpolation-based incremental ECO synthesis for multi-error logic rectification,” in *Proc. DAC*, Jun. 2011, pp. 146–151.

[28] S.-L. Huang, W.-H. Lin, and C.-Y. Huang, “Match and replace—A functional ECO engine for multi-error circuit rectification,” in *Proc. ICCAD*, Nov. 2011, pp. 383–388.

[29] Berkeley Logic Synthesis and Verification Group. (2012). “ABC: A System for Sequential Synthesis and Verification,” [Online] Available: http://www.eecs.berkeley.edu/alanmi/abc/

[30] IBM ILOG CPLEX Optimizer [Online] Available: http://www.ilog.com/products/cplex/

Hua-Yu Chang received the B.S. degree from National Chengchi University, Taipei, Taiwan, in 1998, and the M.S. degree from National Chiao Tung University, Hsinchu, Taiwan, in 2001, both in computer Science. He is currently pursuing the Ph.D. degree in electronic design automation at the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan.

He has 11 years of experience in networking and computer graphics. His current research interests include physical design and logic synthesis.

Iris Hui-Ru Jiang (M’07) received the B.S. and Ph.D. degrees in electronics engineering from National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 1995 and 2002, respectively.

From 2002 to 2005, she was with VIA Technologies, Inc., New Taipei, Taiwan. She is currently an Associate Professor with the Department of Electronics Engineering and the Institute of Electronics, NCTU. Her current research interests include physical design optimization and interaction between logic design and physical synthesis.

Dr. Jiang is a member of the Phi Tau Phi Scholastic Honor Society.

Yao-Wen Chang (S’94–A’96–M’96–SM’12–F’13) received the B.S. degree from National Taiwan University (NTU), Taipei, Taiwan, in 1988, and the M.S. and Ph.D. degrees in computer science from the University of Texas at Austin, Austin, USA, in 1993 and 1996, respectively.

He is currently an Associate Dean of the College of Electrical Engineering and Computer Science and a Distinguished Professor of the Department of Electrical Engineering, NTU. He has been associated closely with industries in these areas. He has co-edited one textbook on electronic design automation and co-authored one book on routing and over 220 ACM/IEEE conference/journal papers in these areas. His current research interests include VLSI physical design and manufacturability.

Dr. Chang was a recipient of four awards at the 50th ACM/IEEE DAC in 2013 for the First Most Papers in the Fifth Decade, Most Prolific Author (at least six papers) in a Single Year in 2012–2013, DAC Prolific Author Award (40 Club), etc. He is a First Place holder of four recent contests: the 2012 DAC Placement Contest, the 2012 ACM ISPD Discrete Gate Sizing Contest, the 2011 IEEE CEDA PATMOS Timing Analysis Contest, and the 2009 ISPD Clock Tree Synthesis Contest. He is a five-time winner of the ISPD contests on placement, global routing, clock network synthesis, and discrete gate sizing, and a recipient of six Best Paper Awards (ICCD, etc.) and 23 Best Paper Award nominations from DAC (five times), ICCAD (four times), ISPD (five times), etc., in the past ten years. He has also received many research and teaching awards, such as the Distinguished Research Award (highest honor) from National Science Council of Taiwan (twice), the IBM Faculty Awards (three times), CIEE Distinguished EE Professorship, MXIC Young Chair Professorship, and Distinguished (highest honor)/Excellent Teaching Awards from NTU (eight times). He is currently an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS and an Editor of the IEEE DESIGN AND TEST OF COMPUTERS. He has served as the General and Steering Committee Chairs of ISPD, and the program chairs of ICCAD, ASP-DAC, FPT, and ISPD. He is on the IEEE CEDA Executive Committee, the ICCAD Executive Committee, the ASP-DAC Steering Committee, and the ACM/SIGDA Physical Design Technical Committee. He has served on the Technical Program Committees of major EDA conferences, including ASP-DAC, DAC, DATE, FPL, FPT, GLSVLSI, ICCAD, ICCD, ISPD, SLIP, SOCC, and VLSI-DAT. He has served as an independent Board Director of Genesys Logic, a Technical Consultant of Faraday, MediaTek, and RealTek, a Chair of the EDA Consortium of the Ministry of Education, Taiwan, and a member of the Board Governors of Taiwan IC Design Society.