Spare-Cell-Aware Multilevel Analytical Placement *

Zhe-Wei Jiang[1], Meng-Kai Hsu[1], Yao-Wen Chang[12], and Kai-Yuan Chao[3]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, TW
[2]Department of Electrical Engineering, National Taiwan University, Taipei, TW
[3]Enterprise Microprocessor Group, Intel Corporation, Hillsboro, OR
{crazying, kaie}@eda.ee.ntu.edu.tw; ywchang@cc.ee.ntu.edu.tw; kaiyuan.chao@intel.com

ABSTRACT

Post-silicon validation has recently drawn designers' attention due to its increasing impacts on the VLSI design cycle and cost. One key feature of the post-silicon validation is the use of spare cells. In the literature, most existing works focus on developing new delicate spare cell structures. On the other hand, the placement of spare cells has a crucial impact on the design cycle and cost of the post-silicon debugging; however, there exists not much work on this placement problem. In this paper, we propose *the first* spare-cell-aware analytical placement framework which predicts the spare cell requirement and considers spare cell insertion during global placement. We also propose a multilevel spare cell insertion technique which provides a more efficient spare cell planning and a better control of quality impact due to spare cell insertion. To guide the selection of available spare cell positions during insertion, we propose a mixed-integer-linear-programming formulation to determine the optimal spare cell positions. Experimental results show that our algorithm can averagely achieve 17–33% and 1.77–2.61X better quality of spare cell insertion than that of the existing spare cell insertion algorithms, UniSpare [10] and PostSpare [22, 26], on the tested real designs with 1–5% spare cell insertion rates.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids [Placement and Routing]; J.6 [**Computer-Aided Engineering**]: Computer-Aided Design

General Terms

Algorithms, Performance

Keywords

Physical Design, Spare Cells, Placement

1. INTRODUCTION

Due to the increasing complexity of modern VLSI designs, more and more errors escape pre-silicon validation. Those errors can only be found after a chip is manufactured, and thus *post-silicon debugging* is required to make the manufactured design work correctly. According to [3], the post-silicon debugging now contributes averagely 35% of the development cycle. It is also pointed out in a recent EE Times article [14] that "post-silicon debugging can cost $15 to $20 million and take six months to complete." These phenomenons have clearly indicated the urgent need of new methodologies to reduce the design cycle and cost of post-silicon debugging.

There are two different ways of performing post-silicon debugging. The first way is to reuse the transistor masks and change only the metal layer masks, which is called *metal fix* [10,15]. Since the transistor masks are usually much more costly to be respun than the metal layer masks, changing only the metal masks can effectively save the post-silicon debugging cost. The other way to perform post-silicon debugging is through *Focus Ion Beam (FIB)* to modify individual chips. The FIB technique has the capability to cut unwanted electrical connections, or to deposit conductive material in order to make a connection after the chip is manufactured. Both of the above two techniques require *spare cells* to be placed before manufacturing in order to perform the desired debugging steps.

Up to present, not much work discusses the spare cell placement problem. The spare cell placement usually follows the intuitive guidelines of placing spare cells close to potentially buggy regions or modules. In the literature, most existing works focus on developing new delicate spare cell structures [5–7, 13, 19, 21–26]. Among these works, one strategy, called *PostSpare*, of inserting spare cells after design placement was proposed by Yee [26] and Payne [22]. However, the PostSpare strategy might not be able to insert spare cells among blocks[1], while blocks are sometimes packed due to performance optimization and buffer insertion. Thus the spare cells will be inserted around all blocks and might be far from the blocks which also require the spare cell. Recently, Chang *et al.* provided a comprehensive survey on existing spare cell insertion techniques in [10]. They also proposed a new spare cell insertion strategy, called *UniSpare*, to place spare cells among the chip uniformly before circuit placement. Such a strategy is expected to reduce the average distance from blocks to spare cells (one most commonly used metric for spare cell insertion quality). However, according to our observation, since the blocks that require the identical spare cell type may not be uniformly distributed in the chip, and the distribution cannot be predicted before placement, the UniSpare strategy may insert spare cells far from those blocks that require them. Figure 1 illustrates such difficulties faced by the PostSpare and UniSpare strategies.

To overcome the limitations and drawbacks of the previous spare cell insertion strategies, we propose *the first* spare-cell-aware analytical placement framework which predicts the spare cell requirement and preserves whitespace for later insertion during global placement. Such a framework enables the opportunities of the optimization process to search for an optimal placement solution with the consideration of spare cell insertion. Afterwards, the spare cells are inserted after the legalization and detailed placement stages, which eases the potential problems with unknown block distributions, and makes the quality impact analysis and control easier. We summarize our major contributions as follows:

*This work was partially supported by ITRI, Springsoft, Synopsys, TSMC, and National Science Council of Taiwan under Grant No's. NSC 97-2221-E-002-237-MY3, NSC 96-2628-E-002-249-MY3, NSC 96-2628-E-002-248-MY3.

DAC 2009, July 26 - 31, 2009, San Francisco, California, USA.

[1]Note that to distinguish from spare cells, we use "blocks" to represent the regular cells within a design in this paper.

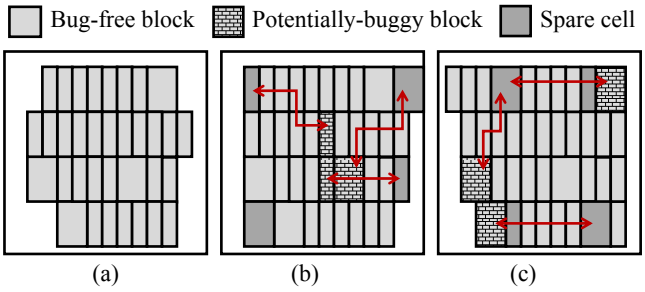

Figure 1: Illustration of existing spare cell insertion strategies. The blocks and spare cells with the identical size are of the identical type. (a) The original placement without spare cell insertion. (b) The PostSpare strategy places spare cells around the packed blocks, but the inserted spare cells are far from the central blocks within the placement region. (c) The UniSpare strategy inserts spare cells between blocks, but the inserted spare cells may be far from those blocks which require them, due to the lack of block distribution information.

- This is *the first* work in the literature to consider the spare cell insertion within the analytical placement framework. The proposed *cluster expansion* and *density constraint determination* can effectively preserve whitespace during global placement for later spare cell insertion.
- This paper proposes a multilevel spare cell insertion technique, which provides a more efficient spare cell planning according to known block distribution and gives a better control of quality impact due to spare cell insertion.
- We propose a mixed-integer-linear-programming formulation to determine the optimal spare cell positions with known block positions. Those optimal positions can provide a good guidance for the selection of available spare cell positions during spare cell insertion.

The remainder of this paper is organized as follows. Section 2 reviews the analytical placement framework used in this paper. The proposed spare-cell-aware global placement techniques and multilevel spare cell insertion are explained in Section 3. Section 4 reports the experimental results. Finally, the conclusions are given in Section 5.

2. REVIEW ON THE ANALYTICAL PLACEMENT FRAMEWORK

The circuit placement problem can be formulated as a hypergraph $H = (V, E)$ placement problem. Let vertices $V = \{v_1, v_2, ..., v_n\}$ represent blocks and hyperedges $E = \{e_1, e_2, ..., e_m\}$ represent nets. Let x_i and y_i be the respective x and y coordinates of the center of block v_i. The circuit may contain some *preplaced blocks* which have fixed x and y coordinates and cannot be moved. We intend to determine the optimal positions of movable blocks so that the total wirelength is minimized, and there is no overlap among blocks. The placement problem is usually solved in three stages, (1) global placement, (2) legalization, and (3) detailed placement. Global placement evenly distributes the blocks and finds the best position for each block to minimize the target cost (e.g., wirelength). Then, legalization removes all overlaps. Finally, detailed placement refines the solution.

We summarize the notations used to explain this framework in Figure 2.

x_i, y_i	center coordinate of block v_i
w_b, h_b	width and height of bin b
P_b	base potential (area of preplaced blocks) in bin b
D_b	potential (area of movable blocks) in bin b
M_b	the maximum potential in bin b
$t_{density}$	target placement density

Figure 2: Notation used in this paper.

To evenly distribute the blocks, we divide the placement region into uniform non-overlapping bin grids. Then, the global placement problem can be formulated as a constrained minimization problem as follows:

$$\begin{array}{ll} \min & W(\mathbf{x}, \mathbf{y}) \\ \text{s.t.} & D_b(\mathbf{x}, \mathbf{y}) \leq M_b, \quad \text{for each bin b,} \end{array} \tag{1}$$

where $W(\mathbf{x}, \mathbf{y})$ is the wirelength function, $D_b(\mathbf{x}, \mathbf{y})$ is the potential function that is the total area of movable blocks in bin b, and M_b is the maximum allowable area of movable blocks in bin b. M_b can be computed by $M_b = t_{density}(w_b h_b - P_b)$, where $t_{density}$ is a user-specified target density value for each bin, w_b (h_b) is the width (height) of bin b, and P_b is the *base potential* that equals the preplaced block area in bin b. Note that M_b is a fixed value as long as all preplaced block positions are given and the bin size is determined.

The wirelength $W(\mathbf{x}, \mathbf{y})$ is defined as the total half-perimeter wirelength (HPWL). Since $W(\mathbf{x}, \mathbf{y})$ is not smooth and non-convex, it is hard to minimize it directly. Thus, several smooth wirelength approximation functions are proposed, such as quadratic wirelength [12,18], L_p-norm wirelength [9, 17], and log-sum-exp wirelength [8,16,20]. The log-sum-exp wirelength model,

$$\gamma \sum_{e \in E} (\log \sum_{v_k \in e} \exp(x_k/\gamma) + \log \sum_{v_k \in e} \exp(-x_k/\gamma) + \log \sum_{v_k \in e} \exp(y_k/\gamma) + \log \sum_{v_k \in e} \exp(-y_k/\gamma)), \tag{2}$$

proposed in [20], achieves the best result among these three models [9]. When γ is small, log-sum-exp wirelength is close to the HPWL [20].

Since density $D_b(\mathbf{x}, \mathbf{y})$ is neither smooth nor differentiable, mPL [9] uses inverse Laplace transformation to smooth the density, while APlace [16] and NTUplace3 [11] use the bell-shaped function for each block to smooth the density. We express the function $D_b(\mathbf{x}, \mathbf{y})$ as $D_b(\mathbf{x}, \mathbf{y}) = \sum_{v \in V} P_x(b, v) P_y(b, v)$, where P_x and P_y are the overlap functions of bin b and block v along the x and y directions. In this paper, we adopt the bell-shaped potential function [16] p_x to smooth P_x. By doing so, the non-smooth function $D_b(\mathbf{x}, \mathbf{y})$ can be replaced by a smooth one, $\hat{D}_b(\mathbf{x}, \mathbf{y}) = \sum_{v \in V}^{n} c_v p_x(b, v) p_y(b, v)$, where c_v is a normalization factor so that the total potential of a block equals its area. Besides, in order to reduce the "valleys" generated by the bell-shaped function, we use the Gaussian function to further smooth the base potential [11].

The quadratic penalty method is used to solve Equation (1), implying that we solve a sequence of unconstrained minimization problems of the form

$$\min \quad W(\mathbf{x}, \mathbf{y}) + \lambda \sum_b (\hat{D}_b(\mathbf{x}, \mathbf{y}) - M_b)^2 \tag{3}$$

with increasing λ's. The solution of the previous problem is used as the initial solution for the next one. We solve the unconstrained problem in Equation (3) by the conjugate

gradient (CG) method. Further, we apply the dynamic step size approach in [11] to speed up the process of minimizing Equation (3).

3. MULTILEVEL SPARE-CELL-AWARE PLACEMENT

In this paper, we propose the spare-cell-aware multilevel analytical placement. Figure 3 summarizes the flow chart of the proposed algorithm extended from the analytical placement framework introduced in Section 2. Note that due to the increasing complexity of modern circuit designs, the multilevel framework is usually applied on the analytical placement to improve the scalability. Since in the upper level of global placement, blocks are grouped into clusters, we propose the *cluster expansion* to provide a rougher prediction of spare cell insertion to each cluster before solving the analytical placement formulation. Then in the finest level of global placement, with known block distribution, we propose the *density constraint determination* to transform the spare cell requirement into density constraints, which achieves a more accurate whitespace preservation for spare cell insertion. Finally, after legalization and detailed placement, the *multilevel spare cell insertion* is proposed to insert spare cells to near-block and less-quality-impact positions. In this section, we will introduce the details of all proposed techniques.

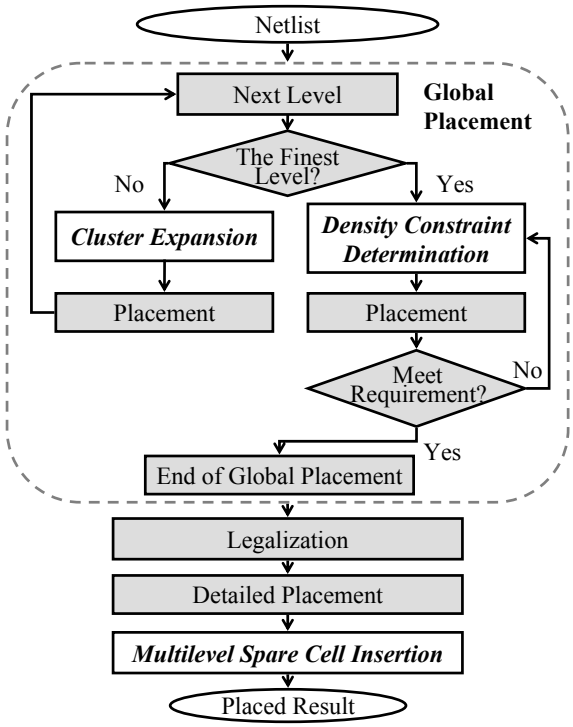

Figure 3: The proposed flow of our spare-cell-aware multilevel analytical placement.

3.1 Cluster Expansion

The multilevel framework applies a two-stage technique of bottom-up coarsening followed by top-down uncoarsening. During the coarsening stage, the blocks are clustered level by level to reduce the number of movable blocks. There have been many famous clustering techniques proposed in the literature, e.g., the best-choice clustering [4] and the first-choice clustering [9]. The clustering process continues until the number of blocks is reduced significantly.

After clustering, the analytical placement problem is solved by the conjugate gradient method at each level of the uncoarsening stage. However, since in the uncoarsening stage, each cluster contains multiple movable blocks, and the exact placement within the cluster remains unknown, it is relatively hard to preserve whitespace for individual blocks within each cluster directly. Therefore, in the upper level of uncoarsening, we evaluate the total area of the spare cell requirement for each cluster, and attach the area to the corresponding cluster when solving the analytical placement formulation.

In most cases, the spare cell requirement is determined according to designers' experience. The requirement can be specified by the number or the insertion rate of each type of spare cells. Without loss of generality, in this paper, we adopt the insertion rate to represent the spare cell requirement. Let r_k represents the given required insertion rate of spare cells of type t_k. For a block v_i, let $T(v_i)$ represent its required spare cell type, and $R(v_i)$ represent the required insertion rate of $T(v_i)$; in other words, if the required spare cell type of v_i is t_k, then $T(v_i) = t_k$ and $R(v_i) = r_k$. Let $Area(t_k)$ represent the area of the spare cell of type t_k. For a cluster C, its required area $Area_s(C)$ for spare cell insertion can be computed by

$$Area_s(C) = \sum_{v_i \in C} R(v_i) \times Area(T(v_i)). \tag{4}$$

Then when solving the analytical placement formulation (Equation (1)) for the current level, $Area_s(C)$ is attached to cluster C when computing the potential function $D_b(\mathbf{x}, \mathbf{y})$. By doing so, we can preserve the whitespace for spare cell insertion during the upper levels of uncoarsening, and thus give the global placement engine more chance to optimize its objectives with the consideration of spare cell insertion.

3.2 Density-Constraint Determination

When the uncoarsening stage reaches the finest level, the optimization process of the analytical placement framework is directly applied on blocks of the circuit instead of clusters. Therefore, with known block positions, we can now provide a more accurate spare cell prediction to the global placement engine. Since the density constraints give a better control of whitespace than expanding block sizes, in the finest level, we propose to transform the spare cell requirement into the density constraints in Equation (1) to further plan the whitespace during global placement.

In the analytical placement framework reviewed in Section 2, the potential function $D_b(\mathbf{x}, \mathbf{y})$ and the maximum allowable potential M_b are determined according to pre-divided uniform non-overlapping bins. To preserve whitespace for spare cell insertion through density constraints, we need to modify M_b for each bin according to its spare cell requirement. Similar to the cluster expansion technique, each block will contribute the required area for spare cell insertion to the corresponding bins, which is proportional to the overlapping area between the block and the bin. For a density bin b, its required area $Area_s(b)$ for spare cell insertion can be written as

$$Area_s(b) = \sum_{v_i \in V} \left(R(v_i) \times Area(T(v_i)) \times \frac{O_x(v_i, b) \times O_y(v_i, b)}{Area(v_i)} \right), \tag{5}$$

where $O_x(v_i, b)$ and $O_y(v_i, b)$ are the amounts of the overlaps between block v_i and bin b along the x and y directions, respectively, and $Area(v_i)$ is the area of block v_i. Then the maximum allowable potential $\hat{M}_b$ with the spare cell consid-

eration can be updated by $\hat{M}_b = M_b - Area_s(b)$, and the analytical placement formulation will be transformed to

$$\begin{aligned} \min \quad & W(\mathbf{x}, \mathbf{y}) \\ \text{s.t.} \quad & D_b(\mathbf{x}, \mathbf{y}) \leq \hat{M}_b \quad \text{for each bin b.} \end{aligned} \tag{6}$$

Solving this formulation can thus preserve whitespace for spare cell insertion. Note that this update for the maximum allowable potential will be performed several times during the conjugate gradient optimization process to monitor the block position change and provide a more accurate prediction of spare cell insertion.

3.3 Multilevel Spare Cell Insertion

Since the spare cell insertion always desires as less impact on the placement quality as possible, the spare cell insertion should therefore be performed after the placement is legalized and fully optimized, which eases the potential problems with unknown block distributions, and makes the quality impact analysis and control easier. Further, because the spare cell requirement may not uniformly distribute within the chip boundary, a whole-chip analysis is required to determine an efficient spare cell planning. Therefore, in this paper, we propose a multilevel spare cell insertion technique which provides a more efficient spare cell planning according to known block distribution and a better control of quality impact due to spare cell insertion.

For each given insertion rate r_k of spare cell type t_k, the total number of required spare cells of type t_k can be easily computed by the total number of blocks that require spare cells of type t_k multiplied by r_k. After obtaining the required number of spare cells, we recursively partition the placement region into uniform sub-regions. Each time the partitioning is performed, the required number of spare cells is also divided and assigned to two sub-regions in proportion to the number of blocks belonging to each sub-region that require spare cells of type t_k. Such an allocation strategy can make sure that the spare cell distribution will be similar to the block distribution. A slicing tree is constructed at the same time to record the cut directions and the number of spare cells assigned to each sub-region. The recursive partitioning process continues until the sub-region is small enough or the number of assigned spare cells to this sub-region equals 0 or 1.

After the construction of the partitions and the slicing tree, we insert the spare cells in a bottom-up fashion. For each sub-region located in the leaf of the slicing tree, we first compute the optimal position for each inserted spare cell according to the blocks that require spare cells of type t_k. Then each row within this sub-region is explored to find available spare cell positions with their quality impacts under a given threshold. Then to achieve a lower average distance from blocks to spare cells, those positions are chosen in an increasing order of the minimum distance to the optimal positions. If some spare cells fail to be inserted within this sub-region, they are reserved and will be inserted again in the upper level of the slicing tree within the union region of this sub-region and those corresponding to its siblings.

Figure 4 gives an example of the multilevel spare cell insertion. Figure 4(a) shows the initial partitions and the block distribution within each partition. Figure 4(b) illustrates the allocation process along the slicing tree. Assume that there are 10 spare cells of type t_k to be inserted in this region. In the root of the slicing tree, the number of allocated spare cells is divided to 7 and 3 because the two sub-regions contain 35 and 15 blocks that require spare cells of type t_k respectively. The partitioning process continues to allocate spare cells into all sub-regions. Note that the recursive partitioning stops at sub-region G since only one spare cell is allocated in G. Figure 4(c) illustrates the bottom-up spare cell insertion process. All spare cells are inserted into their assigned sub-regions. However, in this example, one spare cell failed to be inserted in sub-region A, and another spare cell failed to be inserted in sub-region D. They are reserved and will be inserted into the union of sub-regions A and B and the union of sub-regions C and D respectively, as shown in Figure 4(d).

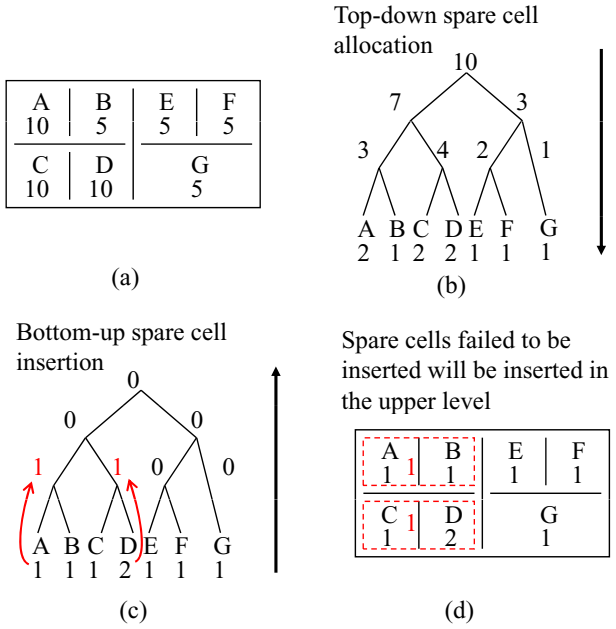

Figure 4: (a) Number of blocks that require spare cells of type t_k in each sub-region. (b) Illustration of the top-down spare cell allocation. The numbers on the tree nodes indicate the amounts of assigned spare cells. (c)(d) Bottom-up spare cell insertion. Spare cells will be inserted in an upper level on the slicing tree if there is no available position in their corresponding sub-region.

3.4 Determination of Optimal Spare Cell Positions

The optimal positions that minimize the average distance from blocks to spare cells can provide a good guidance for the selection of available spare cell positions in our multilevel spare cell insertion. For a sub-region that contains exactly one spare cell, its optimal position is at the gravity center of all blocks that require it. However, when a sub-region contains more than one spare cells, the determination of the optimal spare cell positions becomes much more complex. For a given sub-region, let V_P represent the set of blocks in this sub-region that require spare cells of type t_k, and S represent the set of to-be-inserted spare cells of the same type. For a block $v_p \in V_P$ and a spare cell $s_q \in S$, their distance is represented by $d(v_p, s_q)$. Then, with the objective of minimizing the average distance from placed blocks to spare cells, the determination of optimal spare cell positions can be formulated as

$$\begin{aligned} \min \quad & (\textstyle\sum_{V_P} d_p^{min})/|V_P| \\ \text{s.t.} \quad & d_p^{min} = \min_S d(v_p, s_q), \forall v_p \in V_P, \end{aligned}$$

where d_p^{min} represents the distance from block v_p to its clos-

est spare cell, and $|V_P|$ denotes the size of set V_P. However, it is hard to solve such a formulation directly, since it tries to minimize the minimum of functions.

To cope with such difficulty, we introduce a binary variable $\gamma_{p,q}$ for each pair of (v_p, s_q). If $\gamma_{p,q}$ is set to 1, it means that the closest spare cell for block v_p is s_q. We can then rewrite the formulation as follows:

$$\begin{aligned} \min \quad & (\textstyle\sum_{V_P} d_p^{min})/|V_P| \\ \text{s.t.} \quad & d_p^{min} \geqslant d(v_p, s_q) + (\gamma_{p,q} - 1) \cdot D_{max}, \forall p, q \qquad (7) \\ & \textstyle\sum_q \gamma_{p,q} = 1, \forall p \qquad (8) \\ & \gamma_{p,q} \in \{0, 1\}, \forall p, q, \end{aligned}$$

where D_{max} is a constant representing the maximum possible distance from blocks to spare cells, and is set to the summation of the width and height of the given sub-region. For each block v_p, the inequality (8) constrains that exactly one $\gamma_{p,q}$ will be set to 1. Then in inequality (7), the chosen $\gamma_{p,q}$ will constrain that d_p^{min} is greater or equal to the chosen $d(v_p, s_q)$, while others with value 0 make no influence on d_p^{min}, because D_{max} is always larger or equal to $d(v_p, s_q)$. Since our objective tends to minimize d_p^{min}, the assignment of $\gamma_{p,q}$'s will tend to select the closet spare cell s_p for each block v_p, which makes d_p^{min} equal to the exact minimum distance from block v_p to all spare cells. Such a formulation can be solved by *mixed integer linear programming (MILP)*, and has the capability to find the optimal spare cell positions and the closest spare cell assignment at the same time. Since the sub-regions are usually small and contain few spare cells, say less than 5 in practice, at the leaf nodes of the constructed slicing tree by the multilevel spare cell insertion, the optimal spare cell positions can thus be determined within reasonable running time.

4. EXPERIMENTAL RESULTS

We conducted several experiments to justify the effectiveness of the proposed method. Our algorithm has been integrated into NTUplace3 [11], which is a leading academic placer for the large-scale mixed-size designs and is available to the public. It should be noted that the proposed algorithm is very flexible and can also be integrated into other placers that are based on a similar framework introduced in Section 2 with slight modifications. We compare our proposed spare cell insertion algorithm with two previously proposed ones, PostSpare [22, 26] and UniSpare [10]. We also integrated both algorithms into NTUplace3. All the experiments were performed on the same PC workstation with eight Intel Xeon 2.5 GHz CPUs and 26 GB memory. To demonstrate the effects of the spare cell insertion on real designs, we picked nine largest OpenCores [2] circuits in the IWLS 2005 benchmark suite [1]. The benchmark statistics are given in Table 1.

Table 1: Statistics of the OpenCores circuits used in this paper.

Circuit Name	#Movs	#Nets	Util (%)
ac97_ctrl	11855	11947	70.01
aes_core	20795	21055	70.01
des_perf	98341	98576	70.01
ethernet	46771	46889	70.01
mem_ctrl	11440	11560	70.03
pci_bridge32	16816	16989	70.02
usb_funct	12808	12967	70.01
vga_lcd	124031	124133	70.01
wb_conmax	29034	30165	70.01

4.1 Quality Comparison of Spare Cell Insertion

To demonstrate the effectiveness of our proposed algorithm, we compare the quality of spare cell insertion in this experiment. One most commonly used metric to measure the quality of spare cell insertion is the *average spare distance*. Let $d_{min}(v_i, t_k)$ represent the minimum distance from block v_i to spare cells of type t_k. Then the average spare distance d_{avg} for a placement solution can be computed by

$$d_{avg} = \frac{\sum_{v_i \in V} d_{min}(v_i, T(v_i))}{|V|}, \qquad (9)$$

where $|V|$ indicates the total number of blocks in this circuit. Table 2 reports the average spare distances of all three compared spare cell insertion algorithms for 1–5% spare cell insertion rates. We list the exact values of d_{avg} of our proposed algorithm, and for the UniSpare and PostSpare algorithms, we list the ratios of their d_{avg} to that of our algorithm. Among all tested insertion rates, our algorithm can consistently achieve 17–33% better average d_{avg} than UniSpare due to its lack of block distribution information before placement, while our algorithm applies the multilevel spare cell insertion to place spare cells close to blocks that require them. Compared with PostSpare, since it can hardly insert spare cells between packed blocks without pre-allocated whitespace, the average spare distance d_{avg} of PostSpare is about 1.77–2.61X worse than that of our algorithm.

4.2 HPWL Comparison after Spare Cell Insertion

In this experiment, we compare the HPWL degradation due to spare cell insertion for each algorithm. Table 3 lists the HPWLs for NTUplace3 without spare cell insertion, and the ratios of the HPWLs for our algorithm and UniSpare to those of NTUplace3, among 1–5% spare cell insertion rates. Note that the results of PostSpare are not listed here since its HPWLs are all the same with those of NTUplace3, and its average spare distance d_{avg} is much worse than that of our algorithm and UniSpare. Since the spare cell insertion methods for our algorithm and UniSpare add extra cells between the placed blocks, they often induce HPWL overheads in order to improve the quality of spare cell insertion[2]. Under the identical spare cell insertion rate, since UniSpare distributes spare cells uniformly into the chip, part of the spare cells will be placed far from the placed blocks. Therefore, the number of spare cells inserted between blocks of UniSpare should be less than that of our algorithm, which inserts all spare cells between placed blocks. Consequently, the HPWL overhead of UniSpare should be less than that of our algorithm. However, according to Table 3, the average HPWL overhead of our algorithm (ranges from 2% to 4%) is always less or equal to that of UniSpare (ranges from 3% to 5%) among all tested insertion rates. Such a phenomenon implies that our proposed spare-cell-aware analytical placement framework provides accurate spare cell prediction, and thus enables the opportunities of the global placement engine to optimize its original objectives with the additional consideration of spare cell insertion.

5. CONCLUSIONS

In this paper, we have proposed the first spare-cell-aware analytical placement framework. A multilevel spare cell in-

[2]Due to the unstability of the nonlinear analytical placement formulation, NTUplace3 may sometimes find a slightly better HPWL placement solution with the existence of pre-placed spare cells from UniSpare, or with the use of cluster expansion and density constraint determination for our algorithm.

Table 2: The comparison of the average spare distance (d_{avg}).

Circuit Name	1% Insertion Rate			2% Insertion Rate			3% Insertion Rate			4% Insertion Rate			5% Insertion Rate		
	Ours d_{avg} ($\times 10^5$)	Uni-Spare Ratio	Post-Spare Ratio	Ours d_{avg} ($\times 10^5$)	Uni-Spare Ratio	Post-Spare Ratio	Ours d_{avg} ($\times 10^5$)	Uni-Spare Ratio	Post-Spare Ratio	Ours d_{avg} ($\times 10^5$)	Uni-Spare Ratio	Post-Spare Ratio	Ours d_{avg} ($\times 10^5$)	Uni-Spare Ratio	Post-Spare Ratio
ac97_ctrl	2.94	1.33	1.87	2.05	1.30	2.23	1.70	1.53	2.43	1.48	1.45	2.60	1.34	1.46	2.81
aes_core	2.50	1.32	2.23	1.75	1.30	2.71	1.47	1.31	2.80	1.27	1.28	2.88	1.17	1.26	3.16
des_perf	3.26	1.17	3.29	2.22	1.23	4.15	1.93	1.16	4.47	1.75	1.10	4.88	1.73	0.95	4.69
ethernet	2.91	1.30	3.19	2.18	1.30	3.54	1.89	1.30	3.61	1.59	1.35	3.85	1.51	1.28	3.80
mem_ctrl	2.93	1.31	1.90	2.24	1.31	2.35	1.96	1.20	2.40	1.68	1.21	2.92	1.58	1.18	2.87
pci_bridge32	3.06	1.26	2.27	2.14	1.32	2.61	1.87	1.24	2.51	1.67	1.28	2.57	1.49	1.26	2.71
usb_funct	3.01	1.28	2.06	2.17	1.19	2.50	1.86	1.19	2.75	1.72	1.16	2.92	1.46	1.23	3.31
vga_lcd	3.40	1.07	4.82	2.74	0.96	5.33	2.00	1.13	6.77	2.32	0.80	5.49	2.37	0.68	5.05
wb_conmax	2.16	1.88	3.29	1.53	1.37	3.78	1.35	1.30	4.37	1.23	1.24	4.03	1.19	1.21	4.08
average	-	1.33	2.77	-	1.25	3.24	-	1.26	3.57	-	1.21	3.57	-	1.17	3.61

Table 3: The comparison of HPWL ratios.

Circuit Name	NTUplace3 HPWL ($\times 10^9$)	1% Insertion Rate		2% Insertion Rate		3% Insertion Rate		4% Insertion Rate		5% Insertion Rate	
		Ours	UniSpare	Ours	UniSpare	Ours	UniSpare	Ours	UniSpare	Ours	UniSpare
ac97_ctrl	1.16	1.02	1.02	1.01	1.01	1.02	0.98	1.02	1.02	1.04	1.03
aes_core	1.98	1.02	1.05	1.04	1.05	1.03	1.06	1.05	1.06	1.03	1.07
des_perf	14.08	1.07	1.04	1.13	1.05	1.10	1.23	1.11	1.14	1.14	1.05
ethernet	8.62	1.01	1.04	1.04	1.03	1.03	0.96	0.97	0.98	0.98	0.98
mem_ctrl	1.30	1.03	1.09	1.03	1.10	1.02	1.11	1.05	1.11	1.04	1.13
pci_bridge32	1.86	1.00	0.94	0.98	0.96	1.02	0.97	1.00	0.96	1.01	0.95
usb_funct	1.35	1.03	1.05	1.05	1.06	1.04	1.04	1.06	1.04	1.05	1.03
vga_lcd	29.71	0.98	1.00	0.97	1.06	0.97	1.02	0.99	1.02	1.03	1.03
wb_conmax	3.21	1.01	1.02	0.98	1.02	1.02	1.04	1.04	1.03	1.03	1.05
average	-	1.02	1.03	1.03	1.04	1.03	1.05	1.03	1.04	1.04	1.04

sertion technique has also been presented with proven capability of providing an efficient spare cell planning and an excellent control of quality impact due to spare cell insertion. To guide the selection of available spare cell positions during insertion, we have also proposed a mixed-integer-linear-programming formulation to determine the optimal spare cell positions. Experimental results have shown that our algorithm can achieve a much better efficiency of spare cell insertion than existing algorithms with only slight quality overhead.

6. REFERENCES

[1] *IWLS 2005 Benchmarks*. http://iwls.org/iwls2005/benchmarks.html.

[2] *OpenCores*. http://www.opencores.org.

[3] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller. A reconfigurable design-for-debug infrastructure for SoCs. In *Porc. of DAC*, 2006.

[4] C. Alpert, A. Kahng, G.-J. Nam, S. Reda, and P. Villarrubia. A semi-persistent clustering technique for VLSI circuit placement. In *Proc. of ISPD*, pages 200–207, 2005.

[5] C. Bingert, C. D. Gorsuch, O. G. Mercado, A. K. Myers, J. A. Schadt, and B. W. Yeager. US patent 6,600,341: Integrated circuit and associated design method using spare gate islands. 2003.

[6] M. Brazell and A. Essbaum. US patent 6,993,738: Method for allocating spare cells in auto-place-route blocks. 2006.

[7] P. Chaisemartin. US patent 6,586,961: Structure and method of repair of integrated circuits. 2003.

[8] T. Chan, J. Cong, J. Shinnerl, K. Sze, and M. Xie. mPL6: Enhanced multilevel mixed-size placement. In *Proc. of ISPD*, 2006.

[9] T. Chan, J. Cong, and K. Sze. Multilevel generalized force-directed method for circuit placement. In *Proc. of ISPD*, 2005.

[10] K.-H. Chang, I. L. Markov, and V. Bertacco. Reap what you sow: Spare cells for post-silicon metal fix. In *Porc. of ISPD*, 2008.

[11] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang. NTUplace3: A high-quality mixed-size analytical placer considering preplaced blocks and density constraints. In *Proc. of ICCAD*, 2006.

[12] H. Eisenmann and F. M. Johannes. Generic global placement and floorplanning. In *Proc. of DAC*, 1998.

[13] C. M. Giles. US patent 6,650,139: Modular collection of spare gates for use in hierarchical integrated circuit design process. 2003.

[14] R. Goering. Post-silicon debugging worth a second look. *EE Times*, February 5, 2007.

[15] D. Josephson. The good, the bad, and the ugly of silicon debug. In *Porc. of DAC*, 2006.

[16] A. B. Kahng and Q. Wang. Implementation and extensibility of an analytic placer. *IEEE Trans. on CAD*, 24(5), 2005.

[17] A. B. Kahng and Q. Wang. A faster implementation of APlace. In *Proc. of ISPD*, 2006.

[18] M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. Gordian: VLSI placement by quadratic programming and slicing optimization. *IEEE Trans. on CAD*, 10(3), 1991.

[19] D. Lee. US patent 5,696,943: Method and apparatus for quick and reliable design modification on silicon. 1997.

[20] W. C. Naylor, R. Donelly, and L. Sha. US patent 6,301,693: Non-linear optimization system and method for wire length and dealy optimization for an automatic electric circuit placer. 2001.

[21] Z. Or-Bach. US patent 6,756,811 B2: Customizable and programmable cell array. 2004.

[22] R. L. Payne. US patent 5,959,905: Cell-based integrated circuit design repair using gate array repair cells. 1999.

[23] J. A. Schadt. US patent 6,404,226 B1: Integrated circuit with standard cell logic and spare gates. 2002.

[24] A. Vergnes. US patent 6,791,355 B2: Spare cell architecture for fixing design errors in manufactured integrated circuits. 2004.

[25] J. Wong, D. Chiang, and J. Tolentino. US patent 6,255,845 B1: Efficient use of spare gates for post-silicon debug and enhancements. 2001.

[26] C. L. Yee, S. Aji, and S. Rusu. US patent 5,623,420: Method and apparatus to distribute spare cells within a standard cell region of an integrated circuit. 1997.