Reap What You Sow: Spare Cells for Post-Silicon Metal Fix

Kai-hui Chang
University of Michigan
EECS Department
Ann Arbor, MI 48109
changkh@umich.edu

Igor L. Markov
University of Michigan
EECS Department
Ann Arbor, MI 48109
imarkov@umich.edu

National Taiwan University
EE Department
Taipei, Taiwan 106

Valeria Bertacco
University of Michigan
EECS Department
Ann Arbor, MI 48109
valeria@umich.edu

ABSTRACT

Post-silicon validation has recently become a major bottleneck in IC design. Several high profile IC designs have been taped-out with latent bugs, and forced the manufacturers to resort to additional design revisions. Such changes can be applied through metal fix; however, this is impractical without carefully pre-placed spare cells. In this work we perform the first comprehensive analysis of the issues related to spare-cell insertion, including the types of spare cells that should be used as well as their placement. In addition, we propose a new technique to measure the heterogeneity among signals and use it to determine spare-cell density. Finally, we integrate our findings into a novel multi-faceted approach that calculates regional demand for spare cells, identifies the most appropriate cell types, and places such cells into the layout. Our approach enables the use of metal fix at a much smaller delay cost, with a reduction of up to 37% compared to previous solutions.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-Aided Design

General Terms

Algorithms, Design

1. INTRODUCTION

Due to the recent increase in design complexity, more and more bugs escape pre-silicon validation and are found post-silicon, forcing designers to realize the fixes through additional tape-out and manufacturing revisions (steppings) [15, 18]. As of today, post-silicon debugging already contributes 35% of a chip's design cycle [1]. The delayed market entry due to extended post-silicon debugging often results in a reduced market window and huge revenue loss [8]. In addition, respins are becoming more and more expensive, with mask costs approaching $10M per set [24]. These challenges necessitate better post-silicon debugging methodologies as well as new techniques that reduce the cost of respins.

A mask set consists of 20-40 individual masks, whose costs vary depending on their minimal feature sizes. Since transistor masks exhibit the smallest feature size, they are the most expensive, while metal layers have larger features and are consequently cheaper. Therefore, respin costs can be reduced by reusing transistor masks and only changing the metal layer masks. A fix that only changes metal layers is often called *metal fix* [10]. Instead of a respin, metal fix can also be implemented by modifying individual silicon dies through *Focused Ion Beam (FIB)*. FIB enables the evaluation of layout changes during post-silicon debugging without producing new masks, but it is too slow for mass production. Whether metal fix is implemented via a respin or FIB, *spare cells* are often pre-placed throughout the layout before the die is manufactured to support future logic changes. In the 1990s, circuits were tightly packed to reduce area, and very few spare cells could be used. However, with whitespace taking 30-70% of the area in modern circuits, a greater number of spare cells can be inserted after placement and buffer insertion without increasing the circuit area. For example, modern CPUs can have as many as 1% spare cells.

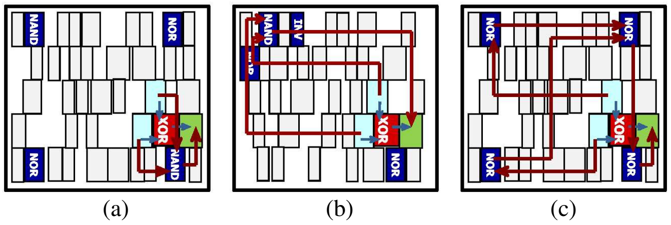

Figure 1: A design where an XOR gate must be replaced by a NAND using spare cells. (a) A high-quality fix with little perturbation of the layout. (b) A low-quality fix that requiring long wires due to poor spare-cell placement. (c) Another low-quality fix using several cells due to a poor selection of cell types.

Traditionally, post-silicon error repair is performed manually. However, with an increasing rate of post-silicon bug escapes, there is a pressing need for automation. Recently, techniques that automate this repair process have been proposed [6, 7]. All these techniques assume that spare cells have been properly inserted in advance to allow metal fix. In practice, however, it is difficult to find an ideal distribution of spare cells. Therefore, engineers often rely on bug analysis from previous silicon dies to determine how spare cells should be inserted. As Figure 1(a) shows, a good spare-cell selection and placement facilitate metal fix with minimal perturbation of the silicon die. On the other hand, Figures 1(b) and 1(c) show that poorly placed spare cells can only be reached through long wires, leading to large increments in the propagation delay of the circuit; and that a poor selection of cell types requires the use of more cells to fix the same error.

Existing techniques for spare-cell insertion either provide better spare cells that are more powerful in generating new logic functions [3, 5, 9, 11, 12, 13, 16, 19, 17, 20] or strive to find better placement for the spare cells, so that they are located in proximity of a potential metal fix demand [3, 4, 5, 9, 11, 13, 17, 20]. Since these techniques are only described in patents, no empirical eval-

Author, year	Spare cell type	Placement and routing methods	Drawbacks/limitations
Yee[20], 1997	Most commonly used cell in the design; **one of the earliest works on spare-cell insertion**	Spare cells scattered after placement	Designed for 2 metal layers only
Lee[11], 1997	**NAND/NOR gates with many inputs, BUF, INV, DFF (new spare-cell selection)**	**Placed close to potentially buggy region**	High-input gates may waste space; other cell types may be more useful
Payne[13], 1999	**Gate array (new structure)**	Spare cells scattered after placement	No new placement technique claimed
Wong[19], 2001	**Configurable logic and INV (new structure)**	N/A	No placement technique claimed
Schadt[16], 2002	**Programmable cells (new structure); elevated lower-level wires improve FIB access**	Spare-cell islands scattered before placement	Uses 2 metal layers only; inputs/outputs of spare cells must be elevated
Chaisemartin [5], 2003	**NAND, DFF, trigate (new structure)**	Placed in a zigzag pattern; **stand-by tracks for routing**	Stand-by tracks may create routing congestion
Bingert[3], 2003	Gate-array islands	**Floorplaned with the design, then scattered uniformly**	Spare-cell islands may occupy too much space
Giles[9], 2003	**New spare-cell selection within cell islands including INV, DFF, MUX, AND, NAND, NOR and BUF**	**Placed according to design hierarchy**	Each module is allowed only one additional I/O; only fixed blocks supported
Or-Bach [12], 2004	**New FPGA-like structure**	N/A	Uses 3 metal layers only; no placement technique claimed
Vergnes[17], 2004	**New structure with functional input bus and an equation input bus**	**Placed with potentially buggy modules by hardwiring inputs of spare cells to signals in those modules**	Occupied routing tracks may create congestion
Brazell[4], 2006	N/A	**Whitespace allocated during Floorplanning; cells inserted after placement**	Spare cells occupy all remaining whitespace — impractical for modern layouts

Table 1: A summary of existing spare-cell insertion techniques described in US patents. Major contributions are marked in boldface.

uation has been reported, particularly in the context of metal fix. As a result, their utility in post-silicon debug remains unclear. In addition, each of the above techniques uses the same combinations of spare-cell types throughout the design. However, different logic blocks may require different types of spare cells. Another problem is that most existing techniques also assume that spare-cell insertion has negligible impact on important circuit parameters, and do not discuss them. However, we found from our analysis that the impact may be significant. To support today's need for repairing errors in complex designs, more flexible and robust spare-cell insertion methodologies are a critical requirement.

Our work offers the first evaluation of strategies for spare-cell selection and placement in the context of post-silicon debugging. It answers the following important questions.

- What types of spare cells are most useful for metal fix?
- Do different types of designs or bugs need different combinations of spare cells? How to select such combinations?
- What is the impact of different spare-cell placement methods on important circuit parameters before and after metal fix?
- Should spare-cell density be different in different regions of a circuit? How to determine the best density automatically?

Our key contributions are:

- A new technique to evaluate which type of cell is most useful to repair a given circuit, called *SimSynth*. SimSynth also measures the heterogeneity among signals and is the first technique that addresses the cell density problem.
- A novel spare-cell insertion methodology that covers both spare-cell selection and placement.

As the first empirical study of strategies for spare-cell insertion, our work contributes new insights into this important problem. Our methodology, developed based on these insights, improves both the selection and the placement of spare cells to facilitate post-silicon metal fix.

The rest of the paper is organized as follows. In section 2 we describe several post-silicon metal fix techniques and review existing solutions for spare-cell insertion. We analyze the utility of spare-cell types in Section 3 and study the placement of spare cells in Section 4. We propose a new spare-cell insertion methodology based on our analysis in Section 5. The results of our work are summarized in Section 6, and Section 7 concludes the paper.

2. BACKGROUND AND PREVIOUS WORK

Post-silicon metal fix techniques: errors observed after tape-out (post-silicon) can be classified into functional, electrical and manufacturing/yield problems. These problems often need to be resolved via metal fix. Traditionally, post-silicon metal fix has been a manual process, but techniques that automate this error-repair process were proposed recently. For example, Chang *et al.* [6] fix functional errors using exhaustive search of resynthesized netlists followed by ECO routing. They also use symmetry-based rewiring and local resynthesis to fix electrical errors. Chen [7] fixes timing-related errors using spare cells to simulate gate sizing, buffer insertion and technology remapping.

After the layout of a circuit has been modified, the change must be implemented on the silicon die via metal fix. Metal fix can be carried out by changing the masks for the metal layers followed by a respin. This approach can implement any change in the layout and has maximum flexibility. However, even though transistor masks can be reused in metal fix, respin still takes several weeks and is expensive. To quickly evaluate candidate fixes without a respin, Focused Ion Beam (FIB) can be used. Nonetheless, the change that can be made by FIB is often limited. For example, it is difficult to access lower-level metal or generate long wires with FIB.

Existing spare-cell insertion methods: spare-cell insertion is a design-dependent challenge whose solutions often rely on bug analysis from previous chips. Due to the confidentiality of relevant data, the results are only revealed in patents. The lack of empirical studies leaves the state of the art unclear, and hampers improvement upon existing techniques. In Table 1 we summarize existing solutions that address the spare-cell insertion problem. Note that some techniques emphasize elevating lower-level wires for easier FIB access, which can also reduce respin cost because only masks for upper-level metal need to be updated. However, the elevated vias and metal segments may cause routing congestion and worsen circuit delay, hurting the overall circuit's performance in the end.

3. CELL TYPE ANALYSIS

As suggested by Table 1, many existing techniques seek better selections of spare-cell types so as to generate more complex logic functions for metal fix [5, 9, 11, 17, 20]. Here, one tries to avoid low-utility cells that waste valuable whitespace. A careful analysis of references suggests that none of the existing techniques vary the spare-cell selection throughout the design. However, intuitively it seems that different circuit blocks in the design may benefit best from different types of spare cells, and in general, some types may be more useful than others. In this work we developed a novel algorithm, called SimSynth, that evaluates quantitatively the usefulness of a specific cell type, and we deployed on a range of designs to determine if the type of spare cells has a relevant impact on the quality of metal fix. Note that currently, we only consider combinational cells in SimSynth. The utility of sequential cells will depend on the sequential error repair technique being applied, which is a more sophisticated problem.

3.1 The SimSynth Technique

Before we developed the SimSynth technique, we tried two other cell-utility measurement experiments. In the first experiment, we resynthesized 30 small subcircuits extracted from our benchmarks using the ABC synthesis tool [22], and then we collected cell types used in the resynthesized netlists. We found that the results were biased toward AND/NAND/INV gates because ABC's internal data structure is the And-Inverter Graph (AIG). To overcome this problem, our second experiment exhaustively searched for valid resynthesized netlists with minimal number of gates. We found that the results were biased toward MUX2 (3-input multiplexer) because having three inputs allows it to generate many more netlists than what is possible with 2-input gates.

Based on these observations, when designing SimSynth we make sure that the underlying resynthesis algorithm is not biased towards one or another type of gates. The SimSynth algorithm relies on a pool of input vectors for the circuit that can either be provided by a high-level simulator or acquired through a random selection. We then compute a signature for each internal circuit wire.[1] These signatures can be thought of as partial truth tables that exclude all controllability don't-cares and they are the input to the SimSynth algorithm as indicated in the pseudocode of Figure 2. The algorithm's output is the success rate to generate a signature that already exists in the design region. To collect the signatures, we select a random wire, search for gates within 40μm from the driver of the wire, and then retrieve the signatures from the outputs of those gates to form a signature pool. *SimSynth* is then called using the signature pool as its input. Note that the 40μm constraint is based on the observation that cells too far away will not be useful in metal fix because the wires that connect to them will be too long and will exhibit significant delay. In addition, FIB cannot generate long wires efficiently. We also exclude resynthesis options that end up with exact same gate type and inputs because the circuit remains unchanged.

Since we are measuring how easily an existing signal can be re-generated, the cell utility is useful for electrical error repair, which generates resynthesized netlists without modifying the circuit's logic functions. However, this technique can also measure the cell utility for functional error repair. The reason is that we are comparing signatures (partial truth-tables) of the signals. If two signals share the same signature, they must be functionally similar,

[1]For example, consider an AND gate with inputs A, B and output O. Given two input vectors (A, B) = (0, 0) and (0, 1), the simulated values of O are 0 and 1. The signatures of A, B and O are then 00,01 and 01, respectively. Note that bit-parallel simulation favors signatures with 32 or 64 bits for efficiency reasons.

```
function SimSynth(candiSigs)
  foreach cell ∈ spareCellTypes
    foreach inputSigs ∈ combinations of signatures from candiSigs
      sig ← cell.compute(inputSigs);
      if (sig ∈ candiSigs)
        success[cell]++;
      count[cell]++;
  return success/count;
```

Figure 2: The SimSynth algorithm.

but can differ on input vectors that have not been used to generate the signatures. This is similar to fixing functional errors: typically, a new signal that fixes a functional error is only slightly different from an existing one because most of the circuit's functions are already correct in post-silicon debugging [6]. In general, more input vectors will bias the utility of spare-cell types toward fixing electrical errors because the generated signals will be closer to existing ones, while the selection will be biased toward functional errors when fewer vectors are used. In our experiments we start with 256 input vectors per circuit, and observe that the numerical results for the utility of cell types stabilize when 2048 or more vectors are used. To make SimSynth more relevant to studying functional errors, we can also consider signatures that are only slightly different from an existing one: generating a signature that is 1-bit different from an existing one can also be counted as a success. In practice, it is also possible that a fix requires a significant change to the circuit's functions. Implementing such a dramatic change, however, typically requires more complex resynthesized netlists involving large numbers of spare cells, which can make metal fix difficult or even impossible. In this work we do not discuss the utility of spare cells for fixing such extensive errors.

Further analysis shows that SimSynth can also be used to determine spare-cell density. The reason is that what SimSynth really measures is the heterogeneity among signals in the circuit. If the success rate is high, then the logic functions of the signals are similar, and generating a new signal that is close to any existing one should be easy. If the rate is low, then the functions of signals are quite different from each other, and generating a new signal using those signals would require more gates. This analysis is confirmed by our experimental results shown in Section 6.2.

EXAMPLE 1. *Figure 3 shows a SimSynth execution example using a full adder, where gate g1 should be XOR instead of OR. Two input vectors are used, producing a 2-bit signature for each wire. Suppose we want to measure the utility of cell types for the region indicated by the dashed line that contains two distinct signatures. SimSynth tries different cell types with different combinations of inputs (only 1 combination in this example) and measures the success rate to replicate an existing signature. The results on the right of the figure show that AND and XOR are more useful than NAND in this case. Note that the correct cell type to fix the bug can be successfully identified because signatures are only partial truth tables, which allow the identification of spare cells that can generate different signals. In general, additional input vectors will bias the cell-type selection towards the one that allows less function change.*

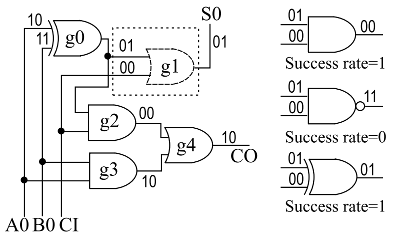

Figure 3: SimSynth example using a full adder.

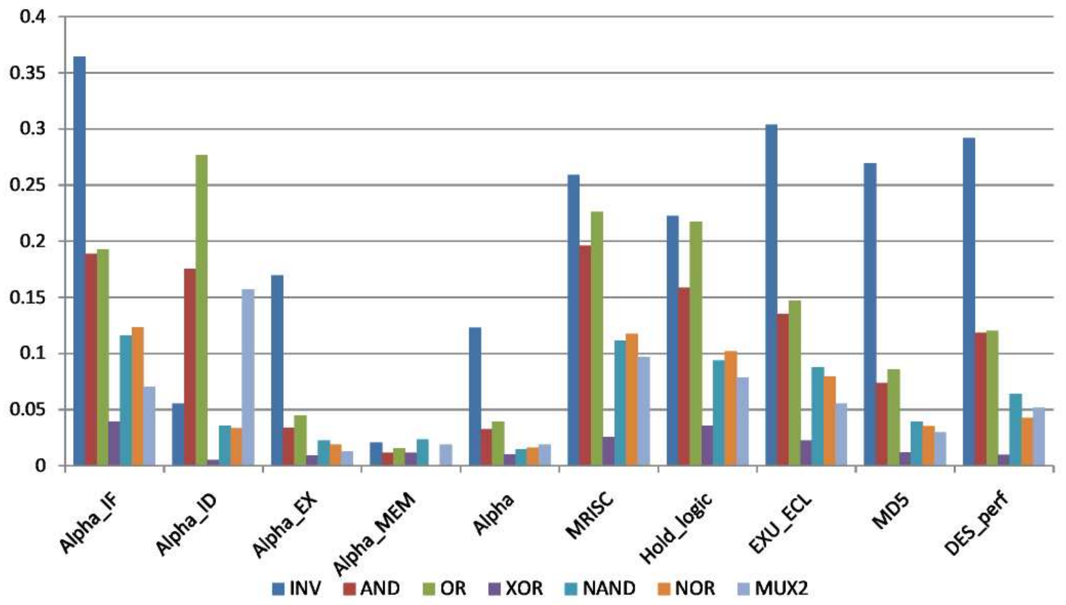

Figure 4: Using single gates of different types to generate desired signals. The success rates are shown in percent.

3.2 Experimental Setup

Our implementation platform is based on the OAGear package [26] from Cadence Labs that uses the OpenAccess database and is integrated with the Capo placer [2]. We use benchmarks provided by the Bug UnderGround project [23] (Alpha), which includes a number of actual bugs found in fully functional microprocessor designs. Other benchmarks are from OpenCores [25] (MRISC, MD5 and DES_perf), picoJava (Hold_logic), and OpenSparc (EXU_ECL) [27]. The characteristics of these benchmarks are summarized in Table 2, where the first four are individual modules of the Alpha processor, followed by the full fledged Alpha design. Next, we show another processor (MRISC), followed by two CPU control blocks (Hold_logic and EXU_ECL) and two cryptographic cores (MD5 and DES_perf). To generate the layout information for these designs, we first synthesize the designs with Cadence RTL Compiler 4.10 based on a 0.18 μm library, and then we instruct Capo to place the design with uniform whitespace. By using uniform whitespace we produce lower bounds for the trends we observe, and the actual trends should be stronger with more realistic placement techniques that distribute design cells to aggressively optimize interconnect. We use Cadence NanoRoute 4.10 to route the final design and calculate the routed wirelength and circuit delay. The cell types considered in our analysis are INV (inverter), AND, OR, XOR, NAND, NOR, and MUX2. All cells except INV and MUX2 have two inputs. To evaluate a region with 200 signals using SimSynth, approximately 6 seconds are required on an AMD 2.4GHz Opteron workstation.

Benchmark	Description	Cell count	Delay (ns)
Alpha_IF	Instruction fetch unit of Alpha	1205	1.15
Alpha_ID	Instruction decode unit of Alpha	11806	1.91
Alpha_EX	Instruction execution unit of Alpha	20903	3.89
Alpha_MEM	Memory stage unit of Alpha	363	0.44
Alpha	Alpha CPU full chip	30212	6.93
MRISC	MiniRISC CPU	4359	2.66
Hold_logic	Part of PicoJava IU control	67	0.61
EXU_ECL	Part of OpenSparc EXU control	2083	0.99
MD5	MD5 encryption/decryption core	9181	6.92
DES_perf	DES encryption/decryption core	100776	3.37

Table 2: Characteristics of benchmarks

3.3 Empirical Results

The experimental results are summarized in Figure 4, which shows two interesting trends. First, the distribution of cell-type utility varies widely among modules of the Alpha processor: signatures can often be re-generated easily using one gate in the IF and ID blocks, but not in the EX and MEM blocks. The reason is that IF and ID contain mostly control logic. Since control logic is mainly generated from “if-then” constructs, most signals are generated by ANDing, ORing or multiplexing the same group of signals. As a result, the logic functions between two signals are often very similar, making it easier to generate identical signatures using one gate. On the other hand, EX is dominated by datapaths. Since signals in such modules usually compute more distant functions, a single gate is less likely to re-generate an existing signature. For example, the first bit and the last bit in an adder compute very different functions. This result shows that to fix errors in arithmetic cores, more spare cells may be needed than fixing similar errors in control logic. Second, we observe that MUX2 is more useful in control logic (Alpha_IF, Alpha_ID, Hold_logic and EXU_ECL) than in arithmetic cores. The reason is that control logic is typically composed of many "if-then" constructs that can be efficiently implemented and modified using multiplexers.

3.4 Discussion

Our empirical results suggest that AND, NAND, OR, NAND and INV are the most useful in general, while XOR is the least useful. But CMOS standard cells that implement INV, NAND and NOR are smaller than those for AND and OR gates, making INV, NAND and NOR preferable as spare cells due to their functional completeness. The utility of MUX2, however, is unclear: it is useful in only some of the benchmarks. Since MUX2 has three inputs, it should be useful in fixing functional errors because it can generate many different functions. In addition, the “if-then” construct commonly used in control logic can be modeled easily using MUX2. Since MUX2 is not a good candidate to fix electrical errors (MUX2 implemented using active transistors is large and slow), it should be implemented using pass transistors to fix functional errors.

In summary, our results suggest that: (1) different types of designs or errors need different combinations of spare-cell types; and

(2) the most useful types are simple ones such as INV, NAND and NOR, while more complex gates such as XOR and MUX2 are less useful. Since there is no clear trend to predict the types of spare cells that will be more useful in a design, performing empirical analysis beforehand for each block in the design should help select the most adequate spare-cell types and distributions.

4. PLACEMENT ANALYSIS

Placement of spare cells is another major factor that affects the quality of metal fix. When errors occur too far from pre-placed spare cells, the required wire connections may be too long to be practical. Even if such wires can be implemented by FIB or respin, the wire delay may also be large. Existing solutions either place the spare cells before design placement [5, 16], with design placement [9, 11, 17], or after design placement [3, 13, 20, 4]. To make sure that spare cells are available where necessary, uniform distribution of spare cells has been used by many existing solutions [3, 5, 16], while several other solutions focus on identifying potentially buggy regions and place spare cells close to them [11, 9, 17]. The spare cells are often grouped into spare-cell islands and then placed on a uniform grid; however, it is also possible to uniformly distribute individual cells instead of grouped cell islands. Since there is little research that evaluates different placement methods, the relative advantages of known techniques remain unclear.

A high-quality spare-cell placement should have minimal impact on important circuit parameters before metal fix to avoid increasing circuit delay or wirelength inadvertently and hurting design quality. It should also facilitate metal fix with the smallest impact on circuit parameters to provide high-quality repair. We observe that most existing techniques either scatter spare cells after design placement or place spare-cell islands uniformly before design placement. We call the former method PostSpare placement and the latter ClusterSpare. PostSpare covers the placement methods described in patents proposed by Yee [20] and Payne [13], while ClusterSpare covers those proposed by Schadt [16], Chaisemartin [5] and Bingert [3]. In ClusterSpare-based techniques, a cell island typically contains one cell for each selected type. Therefore, the number of cells in each island is usually large. An illustration of these placement methods is given in Figure 5.

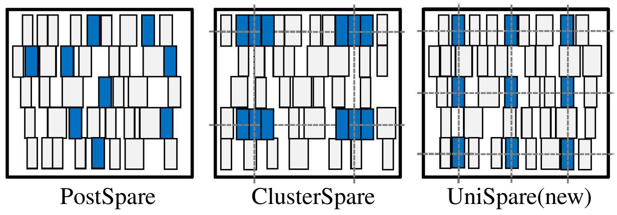

Figure 5: Illustration of different placement methods. Dark cells are spare cells. PostSpare inserts spare cells after design placement. Since design cells may be clustered in some regions, spare-cell distribution is typically non-uniform. ClusterSpare inserts spare-cell islands on a uniform grid before design placement, while UniSpare inserts single spare cells.

PostSpare placement should have minimal impact on important circuit parameters because spare cells are inserted after design placement. However, the error-repair quality of this method may be poor when design cells form high-utilization areas, forcing spare cells into sparser regions. When this happens, long wires may be needed to reach those cells. ClusterSpare placement may have larger impact on circuit parameters because the cell islands will act like macros and reduce the optimization that can be performed by the placer. However, it should provide better error-repair quality because their uniform distribution makes their access easier. Therefore, shorter wires can be used to reach the cell islands. In addition, connections among cells within the same island only require local wires and will be easy to implement. Note that, however, even the relatively short wires necessary to reach the spare-cell islands of ClusterSpare may trigger unacceptable wire delay increase in current silicon technology nodes, which are extremely delay-sensitive.

In this work we propose UniSpare, a solution that pre-places individual spare cells uniformly on a grid, as illustrated schematically in Figure 5(c). In this way, the average distance from a design cell to the closest spare cell is reduced. For example, when the size of the clusters reduces from 16 to 1 while maintaining the total number of spare cells, the average distance to reach a spare cell is reduced by 4 times. In a resynthesized netlist involving many gates, these individual cells can also act like buffers to increase signal strength, thus further reducing wire delay. Our experimental results in Section 6 indicate that the UniSpare placement technique is superior to previous methods.

5. OUR METHODOLOGY

Based on our analysis, we propose a new spare-cell insertion methodology, illustrated in Figure 6. Below we explain how it performs the selection and placement of spare cells.

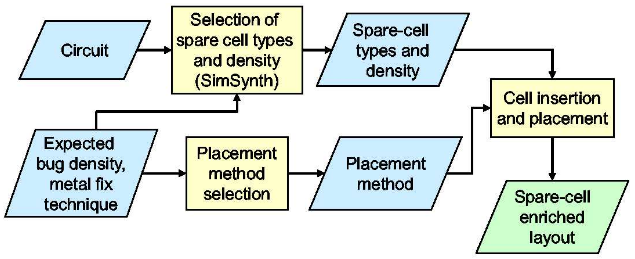
Circuit
Selection of spare cell types and density (SimSynth)
Spare-cell types and density
Cell insertion and placement
Expected bug density, metal fix technique
Placement method selection
Placement method
Spare-cell enriched layout

Figure 6: Our spare-cell insertion flow.

Our analysis suggests that different types of circuits require different distributions of spare-cell types. To select appropriate types, we apply our SimSynth technique (described in Section 3.1) in each design module and use the resulting cell-type distribution to determine the types of spare cells that should be inserted to each module. Since AND and OR gates require greater area than NAND and NOR, in our methodology we always use INV, NAND and NOR. In addition, for control blocks we insert multiplexers implemented using pass transistors to fix functional errors. Cell structures that provide greater flexibility, such as programmable logic or gate array [3, 12, 13, 16, 19], can also be used. However, they often require additional long wires to support programming.

The density of spare cells can be determined by the expected bug rate. If a circuit module is potentially buggy, then more spare cells should be placed in that module. For example, a perfectly working/verified circuit that is being scaled down to a new technology may encounter new electrical errors, but functional errors should not be prominent. In arithmetic cores, functional errors are relatively unlikely because these cores are usually heavily verified and are reused among designs. If bugs do occur, however, they may be difficult to repair using metal fix alone because all 32 or 64 bits may be affected. Wagner *et al.* [18] showed that most errors found in high-profile processors are in control logic. Therefore, more spare cells should be placed there.

If the expected bug rate is unknown, the results from SimSynth could be used. If the success rate measured by SimSynth in a block is lower than other blocks, then the heterogeneity among signals in the block is high and more spare cells should be placed. Suppose that there are n blocks in a circuit, the average success rate for block B_i is S_i, and the average success rate for all the blocks is S_{avg}. Also assume that the target overall spare-cell density is $D_{all}\%$. Formula 1 shows how to determine the spare-cell density D_i for block B_i. In the formula, P is a parameter that determines the impact of S_i on D_i and should be determined empirically. For example, based on our evaluation, P should be 20% for the blocks in the Alpha processor.

$$D_i = [\frac{(S_i - S_{avg}) \times P}{S_{avg}} + 1] \times \frac{D_{all}}{100\%} \tag{1}$$

The placement of spare cells depends on the expected bug rate and the metal-fix technique being used. If the expected bug rate is low, spare cells can be scattered uniformly after design placement. This helps ensure that spare cells do not affect circuit performance. If the expected bug rate is higher or unknown, then spare cells should be pre-placed uniformly before design placement so that wherever a fix must be applied, there are spare cells close to the repair site. To reduce the impact of the fix on important circuit parameters, spare cells should be placed individually or as small islands throughout the design using our proposed UniSpare method. Note that spare cells not connected during metal fix can also be used as buffers to improve circuit timing, as [7] suggests.

6. EXPERIMENTAL RESULTS

In this section we empirically evaluate our techniques and compare them with existing solutions.

6.1 Cell-Type Selection

Experiment design: in this experiment we compare our results with two cell-selection methods: Giles [9] and Yee [20]. According to Figure 4, we use INV, NAND and NOR for most benchmarks, while Alpha_ID also includes MUX2. Giles uses INV, DFF, MUX, AND, NAND, NOR and BUF as spare cells. Since Yee selects the "most-commonly used cell types" without indicating the number of types that should be used, we synthesized the benchmarks again using the seven types from which spare cells are drawn, and then selected the most-used two types for each benchmark, which were consistently NAND and INV. We use the UniSpare placement method for all three spare-cell selections to make sure the results are not affected by placement. To perform the experiment, we first select a subcircuit composed of 1-6 cells that are connected to each other. Next, we mimic a "fix" by resynthesizing the subcircuit and then map the resynthesized netlist to spare cells close to the subcircuit. Finally, we measure the delay and wirelength of the circuit after routing the modified netlist using NanoRoute's ECO mode. Better spare-cell selections should allow metal fix to be performed with smaller impact on circuit delay and wirelength. We ran each experiment 50 times to collect 50 data points for statistical analysis.
Results: the results are summarized in Figure 7. The graph shows that our spare-cell selection produces 23% and 4% smaller delay increase compared to Yee and Giles at a comparable wirelength increase. This result shows that our spare-cell selection can find more useful cells for each design and provides better error-repair quality after metal fix.

6.2 Spare-Cell Placement

Three different types of placement methods are used in our experiments, and an illustration is given in Figure 5. PostSpare inserts individual spare cells after design placement; UniSpare inserts individual cells on a uniform grid before design placement; and ClusterSpare inserts spare-cell islands on a uniform grid before design placement, where each island is composed of 9 cells. We use INV, NAND and NOR gates as spare cells in our experiments, and

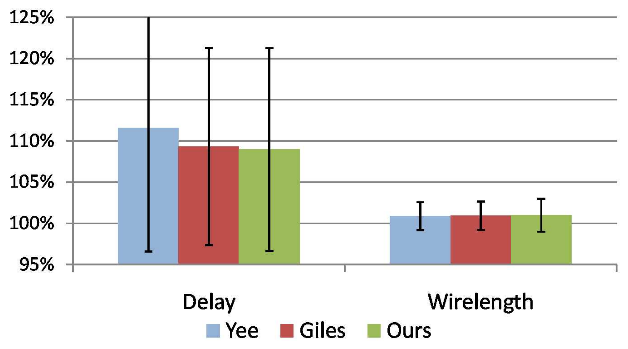

Figure 7: Delay and wirelength increase *after* metal fix when using three different sets of spare-cell selections. Ours has 23% and 4% smaller delay increase compared to Yee and Giles, while the wirelength increase is approximately the same.

each benchmark contains approximately 4% spare cells. The placer and router used in these experiments are Capo and NanoRoute. We ran each experiment 50 times to collect 50 distinct data points for statistical analysis.
Circuit parameter analysis before metal fix: in this experiment we first insert spare cells using the three methods described earlier. Next, we place and route the design using Capo and NanoRoute. Finally, we measure the impact of different placement methods on important circuit parameters, including delay and wirelength.

Figure 8(a) shows the average results of the benchmarks, and the error bars represent the range of one standard deviation. The figure shows that PostSpare placement does not affect circuit delay or wirelength. This is expected because the spare cells are placed after design placement; therefore, delay and wirelength should not be affected by spare-cell insertion. ClusterSpare placement shows a very interesting trend where the delay is increased while wirelength decreases. The reason is that large cell islands act like macros and force Capo to place design cells closer together, thus reducing total wirelength. At the same time, longer wires must be used to connect cells around the spare-cell islands, resulting in larger delay. For more aggressive placers, however, this trend may not be observed. The results also show that wirelength increased by 0.9% in UniSpare placement. This is because pre-placed spare cells will occupy certain placement sites, reducing the number of sites that can be used by the placer. Therefore, the optimization that can be performed by the placer will also be limited, resulting in larger wirelength. The delay, however, is only slightly affected by the inserted spare cells because connecting cells around a single cell only needs slightly longer wires, resulting in 24% smaller delay increase than ClusterSpare placement. We also note that the standard deviations are large in ClusterSpare and UniSpare placement methods, suggesting that spare-cell insertion may destabilize existing placement and routing tools.
Repair quality analysis after metal fix: after errors in a circuit have been repaired by metal fix, the circuit's major physical parameters may change, including interconnect length and maximum delay. Typically, repairs with higher quality can minimize the perturbation of those parameters. Since the quality of metal fix is affected by the placement of spare cells, we reused the experiment described in Section 6.1 to measure the impact of placement methods on error-repair quality. Since fixes that do not affect a critical path have no impact on circuit delay, we only selected data points whose delay has been changed to measure the true impact of placement methods on delay.

The average changes of physical parameters after metal fix are shown in Figure 8: Figure 8(b) shows the impact of placement on circuit delay and wirelength, while Figure 8(c) shows the impact

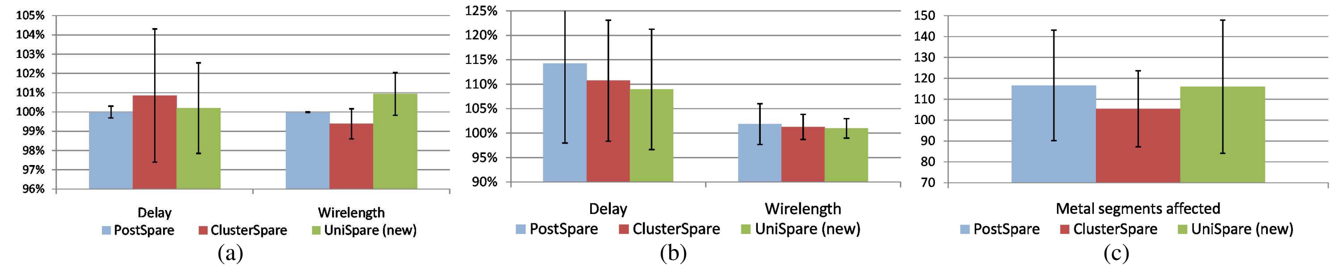

Figure 8: Impact of spare-cell placement methods on circuit parameters: (a) before metal fix; (b)(c) after metal fix. Ours has 24% smaller delay inrease before metal fix compared with ClusterSpare. The delay increase after metal fix is 37% and 17% better than the PostSpare and ClusterSpare methods, respectively.

on the number of affected metal segments. The error bars represent one standard deviation. The results show that PostSpare placement produces poor repair quality because it triggers a larger increase in delay and wirelength. In addition, it also affects more metal segments, making FIB more difficult. These trends should be stronger with non-uniform distribution of whitespace. From Figure 8(b), we observe that UniSpare placement has smaller delay and similar wirelength increase compared to ClusterSpare. The reason is that the cell islands placed by ClusterSpare are farther away from each other than the spare cells placed by UniSpare. As a result, longer wires are needed to connect to those cell islands, resulting in larger delay. On the other hand, Figure 8(c) shows that smaller numbers of metal segments are affected in circuits produced by ClusterSpare. This is because once those long wires reach the cell islands, connections among the cells in the same island only require local wires and will not perturb other wires. On average, UniSpare placement results in 37% and 17% smaller delay increase compared with PostSpare and ClusterSpare respectively, suggesting that it is the best placement method.

To further study the impact of different placement methods on important physical parameters, we took benchmark Alpha and plotted Figure 9 to show the relationship between the increase in wirelength and the number of spare cells used in metal fix. In this figure, we show the data points from three placement types and their linear regression lines. The results show that wirelength increases when more cells are used in metal fix because more cells need more wires to connect. The regression lines suggest that when more cells are used in metal fix, wirelength of PostSpare will increase faster than the other two types. This is because the irregularity of spare-cell distribution produced by PostSpare may make the required spare cells difficult to reach, resulting in very long wires. Figure 9 also shows that the linear regression lines of ClusterSpare and UniSpare are close to each other. This is not surprising because spare cells are placed uniformly in both techniques. Therefore, on average the lengths of the wires to connect those cells will not differ too much. However, the delay may be different, as Figure 8(b) suggests.

In Figure 10 we plot the number of metal segments affected by the performed fixes against the number of spare cells used in the fixes. The results show that when a fix requires more spare cells, more metal segments will be affected in a circuit produced by PostSpare than a circuit produced by UniSpare, while ClusterSpare has the smallest number. As explained earlier, PostSpare creates many long wires and will affect more metal segments, while ClusterSpare placement can utilize local connections, thus reducing the number of segments affected.

Density of Spare Cells: another interesting placement-related issue is the density of spare cells. Several existing techniques suggest that spare cells should be inserted close to potentially-buggy circuit modules [11, 17]. This approach is certainly useful if such information is available. However, it cannot be used if the bug distribution of a chip is unknown. As discussed in Section 3.1, SimSynth can address this problem. To evaluate its effectiveness, we counted the average number of spare cells used in the fixes produced by our previous experiment, and we contrast the results with Figure 4.

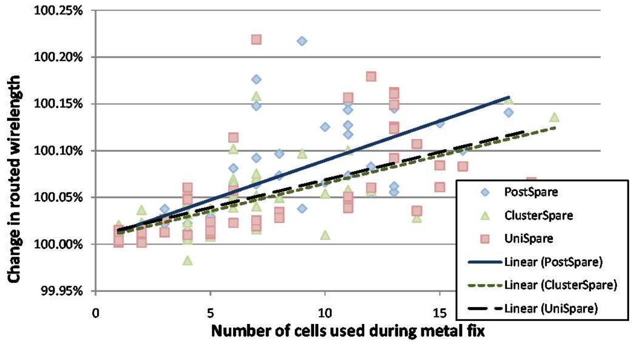

Figure 9: Relationship between wirelength increase and the number of spare cells used in metal fix.

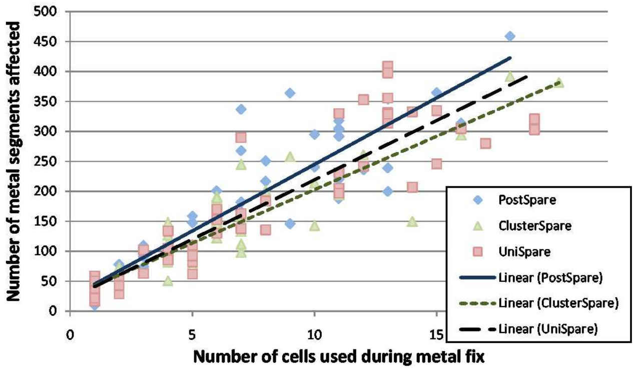

Figure 10: Relationship between the number of metal segments affected and the number of spare cells used in metal fix.

The results of this experiment are shown in Figure 11. This figure shows that to generate the same signal, the Alpha processor needs more spare cells than its EX block, followed by its ID and IF blocks. If we contrast this result with Figure 4, we can see that the IF block has the highest success rate in generating an existing signature using one spare cell, followed by ID, EX and the Alpha processor. These two observations are correlated because if it is easier to generate an existing signal using one gate, the number of cells needed to replicate a signal should also be smaller, at least on average. This phenomenon can also be observed on MD5 and DES_perf: MD5 requires more cells in each fix, and the success rate to generate an existing signal using one gate is also smaller. This result suggests that measuring the success rate of our SimSynth experiment can help determine the density of spare cells that should be placed on a silicon die.

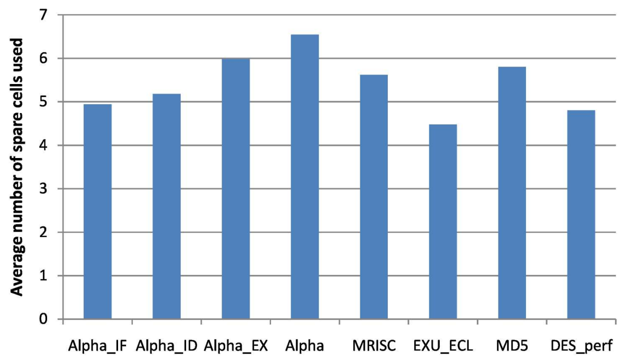

Figure 11: Average numbers of cells used when fixing bugs in the benchmarks. By contrasting with Figure 4 we show that SimSynth can help determine spare-cell density. For example, Alpha has smaller success rate in Figure 4 than its EX block, followed by its ID and IF blocks. This figure shows that the Alpha design requires more cells than its EX, ID and IF blocks.

7. CONCLUSIONS

In this work we performed a comprehensive analysis of spare-cell insertion to study the nature of this problem. Based on what we learned from this analysis, we proposed a new methodology that is more flexible than existing solutions and covers both spare-cell selection and placement. Furthermore, we described a SimSynth technique that can measure the heterogeneity among signals in a particular region of a placed netlist. It can help determine the spare-cell density automatically — a problem that has not been previously addressed in the EDA literature.

Our work evaluates, for the first time, several rules of thumb commonly used in spare-cell insertion. First, several existing solutions suggest to use the "most-commonly used" cell type in the design as the spare-cell type. According to our results, the most popular cell type is indeed very useful, but (1) other types can be equally useful, and (2) using a blend of several spare-cell types provides better error-repair quality than using only one or two types. Second, most existing solutions use large spare-cell islands. Our analysis shows that this approach hurts circuit's wirelength and timing, and we believe that the difference will grow with each technology node due to poor scaling of interconnect delay. To reduce this impact, smaller islands should be used so as to reduce the average distance from a design cell to the closest spare cell. This will shorten the wires that connect to spare cells and improve circuit delay after metal fix. Third, most existing solutions neglect the impact of spare-cell insertion on circuit parameters. However, we showed that this impact may be significant. Without careful planning, spare-cell insertion can worsen circuit timing and wirelength.

The success of post-silicon metal fix is contingent upon a good spare-cell insertion methodology. However, the EDA literature offers practically no accounts of research on this topic. To this end, our flexible spare-cell insertion methodology not only provides better selections of spare cells but also generates placements that minimize the impact on circuit parameters. As shown by empirical evaluation, our placement and spare-cell selection techniques provide 17-37% and 4-23% smaller delay increase compared with existing solutions, demonstrating the ability of our methodology to improve the quality of post-silicon metal fix. Due to interconnect scaling, we expect stronger trends for more advanced cell libraries than the 180nm technology we used. More aggressive placement tools with non-uniform utilization should also strengthen the trends (our experiments only give a lower bound).

Acknowledgments: The authors thank Prof. Todd Austin for motivating this research. This work was supported by grants from SRC, GSRC and NSF.

8. REFERENCES

[1] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi and D. Miller, "A Reconfigurable Design-for-Debug Infrastructure for SoCs", *DAC'06*, pp. 7-12

[2] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa and I. L. Markov, "Unification of Partitioning, Floorplanning and Placement", *ICCAD'04*, pp. 550-557

[3] C. Bingert, C. D. Gorsuch, O. G. Mercado, A. K. Myers, J. A. Schadt and B. W. Yeager, "Integrated Circuit and Associated Design Method Using Spare Gate Islands", US Patent 6600341 B2, Jul. 2003

[4] M. Brazell and A. Essbaum, "Method for Allocating Spare Cells in Auto-Place-Route Blocks", US Patent 6993738 B2, Jan. 2006

[5] P. Chaisemartin, "Structure and Method of Repair of Integrated Circuits", US Patent 6586961 B2, Jul. 2003

[6] K.-H. Chang, I. L. Markov and V. Bertacco, "Automating Post-Silicon Debugging and Repair", *ICCAD'07*, pp. 91-98

[7] Y.-P. Chen, J.-W. Fang and Y.-W. Chang, "ECO Timing Optimization Using Spare Cells", *ICCAD'07*, pp. 530-535

[8] C. Chiang and J. Kawa, "Design for Manufacturability and Yield for Nano-Scale CMOS", Springer, 2007

[9] C. M. Giles, "Modular Collection of Spare Gates for Use in Hierarchical Integrated Circuit Design Process", US Patent 6650139 B1, Nov. 2003

[10] D. Josephson, "The Good, the Bad, and the Ugly of Silicon Debug", *DAC'06*, pp. 3-6.

[11] D. Lee, "Method and Apparatus for Quick and Reliable Design Modification on Silicon", US Patent 5696943, Dec. 1997

[12] Z. Or-Bach, "Customizable and Programmable Cell Array", US Patent 6756811 B2, Jun. 2004

[13] R. L. Payne, "Cell-Based Integrated Circuit Design Repair Using Gate Array Repair Cells", US Patent 5959905, Sep. 1999

[14] R. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization", *IEEE TCAD*, pp. 727-750, Sep. 1987

[15] S. Saranqi, S. Narayanasamy, B. Carneal, A. Tiwarj, B. Calder and J. Torrellas, "Patching Processor Design Errors with Programmable Hardware", IEEE Micro, Vol. 27(1), 2007, pp. 12-25

[16] J. A. Schadt, "Integrated Circuit with Standard Cell Logic and Spare Gates", US Patent 6404226 B1, Jun. 2002

[17] A. Vergnes, "Spare Cell Architecture for Fixing Design Errors in Manufactured Integrated Circuits", US Patent 6791355 B2, Sep. 2004

[18] I. Wagner, V. Bertacco and T. Austin, "Shielding Against Design Flaws with Field Repairable Control Logic", *DAC'06*, pp. 344-347

[19] J. Wong, D. Chiang and J. Tolentino, "Efficient Use of Spare Gates for Post-Silicon Debug and Enhancements", US Patent 6255845 B1, Jul. 2001

[20] C. L. Yee, S. Aji and S. Rusu, "Method and Apparatus to Distribute Spare Cells within a Standard Cell Region of an Integrated Circuit", US Patent 5623420, Apr. 1997

[21] H. Xiang, L.-D. Huang, K.-Y. Chao, and M. D. F. Wong, "An ECO Algorithm for Resolving OPC and Coupling Capacitance Violations", *ASICON'05*, pp. 784-787

[22] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 51205. `http://www-cad.eecs.berkeley.edu/~alanmi/abc/`

[23] Bug UnderGround, `http://bug.eecs.umich.edu/`

[24] International Technology Roadmap for Semiconductors 2005 Edition, `http://www.itrs.net`

[25] `http://www.opencores.org/`

[26] `http://openedatools.si2.org/oagear/`

[27] `http://www.sun.com/`