Fault Tolerant Placement and Defect Reconfiguration for nano-FPGAs

Amit Agarwal, Jason Cong and Brian Tagiku
Department of Computer Science
University of California at Los Angeles
Los Angeles, USA
{agarwal, cong, btagiku}@cs.ucla.edu

Abstract—**When manufacturing nano-devices, defects are a certainty and reliability becomes a critical issue. Until now, the most pervasive methods used to address reliability, involve injecting spare resources. However, these methods use predetermined spare placement that is not optimized for each netlist. This is the first work (to the best of our knowledge) that addresses the problem of fault tolerance for nano-FPGAs at the placement stage; fault tolerant placements are generated that are amenable to fast defect reconfiguration through replacement of defective logic elements with spares. We propose a simulated-annealing based placement algorithm that produces placements with the objective of maximizing the chances of successful recovery from faults in logic elements within the circuit's timing constraints. In addition, our study of the fault reconfiguration problem shows it is NP-Complete, and we propose a fast scheme for achieving a good reconfiguration solution for a random or clustered fault map. Experimental results show that these techniques can increase the probability of successful fault reconfiguration by 55% (compared to a uniform spare distribution scheme), without significantly degrading the circuit performance.**

I. Introduction

The decrease in size of lithographic transistors cannot continue indefinitely. Nano-devices, such as nano-wires or carbon nano-tubes, seem to be the most promising successor to lithographic based ICs. However, at such miniscule dimensions, nano-devices are likely to have a regular structure generated by a stochastic self assembly process and, consequently, are highly prone to defects and transient faults. The exact level of defect densities is unknown, but it is assumed that 1-15% of the resources on a chip (wires, switches FETs, etc.) may be defective [7, 8, 9]. This suggests the requirement of a reconfigurable architecture, similar to FPGAs, to enable the devices to function in spite of defects.

The inherent redundancy and reconfigurable nature of FPGA architectures enables fault tolerance through the use of spare unused logic elements. Reconfiguration of circuit placement, to map around faulty elements, alters the critical path delay of the circuit, and it is necessary that reconfiguration algorithms meet the target delay or guarantee low timing degradation. Additionally, tolerating transient run-time faults requires the reconfiguration process to be fast. Redoing circuit placement and routing, avoiding the faulty elements, takes a prohibitively long time and is not an attractive solution, particularly for handling frequently occurring transient faults in large nano-electronic circuits.

A commonly employed fault tolerance approach is to reserve redundant resources, which are used to replace faulty resources by re-routing their connections [2, 3, 5 10]. The explicit redundancy introduced has an area overhead, and these methods are limited in the number of defects that can be tolerated. The predetermined generic placement of spare resources is not optimal for each netlist and may lead to increase in the circuit delay upon reconfiguration.

Another approach is to attempt mapping the system function to a set of fault-free resources. Methods using this approach [11, 12] utilize the inherent redundant resources of the FPGA and do not require additional redundancy. These techniques provide a tradeoff between reconfiguration time and timing degradation. An elaborate re-route results in low timing degradation at the cost of high run-time and vice-versa.

Our work first focuses on addressing the problem of defect tolerance starting at the placement stage. We present a simulated annealing based placement method for generating reconfiguration friendly placements. The unused logic elements (spares) are distributed in an effective manner across the FPGA to maximize the probability of successful reconfiguration of random or clustered transient faults in logic elements, within the circuit's timing constraint.

We then study the problem of fault reconfiguration by direct replacement of faulty logic elements with spare resources, without affecting non-faulty elements, and show that it is NP-complete. We develop a fast branch & bound algorithm for defect reconfiguration; it utilizes inherent redundant resources of FPGAs effectively distributed at the placement stage, and causes minimal increase in circuit's critical path delay. Faulty logic elements are directly swapped

This work is partially supported by the NSF Grant CCF-0530261 for the project "International Center on Design for Nanotechnologies".

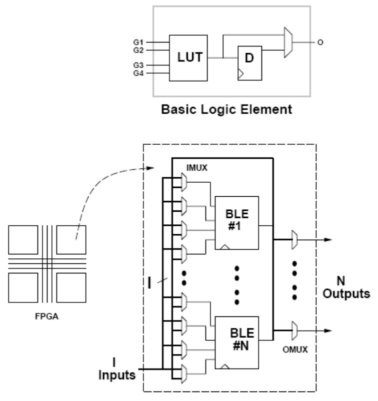


Figure 1. Basic Logic Element (BLE) and Cluster Logic Block (CLB)

with spares, requiring re-routing of the fault node connections only, to achieve fast reconfiguration with low timing degradation. We also present a reconfiguration algorithm inspired by the ripple move placement legalization algorithm in Mongrel [18]. Here the faulty elements are reconfigured through a sequence of moves from the fault node to a neighboring spare resource, effectively rippling the faults to spare logic elements. This algorithm yields higher reconfiguration success rate and lower timing degradation compared to the first method, at the cost of an increase in the number of re-routes of the circuit connections.

II. Preliminaries And Definitions

Modern FPGA architectures typically have a 2-level hierarchy consisting of an array of identical logic blocks, each of which is a cluster of a fixed number of basic logic elements (BLE). Altera Stratix-II and Xilinx Virtex-4/5 are examples of this form of architecture. Figure 1 shows the basic structure of a BLE and a clustered logic block (CLB) of size N—i.e. containing N BLEs. The logic block inputs and outputs are fully connected to the inputs of each BLE. The interconnect delay between BLEs within the same CLB, is usually much smaller than the delay between BLEs in different CLBs.

A. Timing Model

In this section we review the timing model used by VPR [15, 16] and SCPlace [13, 14] placement tools, on top of which the placement and reconfiguration algorithms presented in this paper are built.

The circuit is represented as a directed acyclic graph. Nodes in the graph represent input and output pins of circuit elements such as LUTs, registers, and I/O pads. Connections between nodes are modeled with edges in the graph which are annotated with a delay corresponding to the physical delay between the nodes. To determine the delay of the circuit, a breadth-first traversal is performed on the graph starting at sources (input pads, and register outputs). *Arrival time*, $T_{arrival}$, is computed at all nodes in the circuit with the following equation:

$$T_{\text{arrival}}(i) = max_{j \in fanin(i)}\{T_{arrival}(j) + d(j, i)\} \quad (1)$$

where node *i* is the node currently being computed, and d(j, i) is the delay value of the edge joining node j to node i. The delay of the circuit is then the maximum arrival time, D_{max}, of all nodes in the circuit.

The amount of delay that may be added to a connection before it becomes critical is called the *slack* of that connection. To compute the slack of a connection, the *required arrival time*, $T_{required}$, is computed at every node in the circuit. $T_{required}$, at all sinks (output pads and register inputs), is set to the target delay D_{TARGET} and then propagated backwards starting from the sinks with the following equation:

$$T_{required}(i) = \min_{j \in fanout(i)}\{T_{required}(j) - d(i, j)\} \quad (2)$$

Finally, the slack of a connection *(i,j)* driving node *j*, is defined as:

$$Slack(i, j) = T_{required}(j) - T_{arrival}(i) - d(i, j) \quad (3)$$

For modeling interconnect delay, a “delay profile” of the FPGA is created that is used to rapidly evaluate the delay of each connection in the circuit given the placement of its terminals. In a “tile-based” FPGA, the FPGA structure is homogeneous, i.e. every x, y location in the FPGA is constructed from identical tiles. Exploiting the uniformity of such architectures, a *delay lookup matrix* is computed where the delay of a connection between two blocks is computed as a function only of the distance (Δx, Δy) between them. VPR’s timing-driven router is used to perform routing between a source block placed at a location (x_{source}, y_{source}) in the FPGA, and a sink block placed at (x_{source}+Δx, y_{source}+Δy), and the delay is recorded in the (Δx, Δy) entry of the lookup matrix.

B. Fault Model

In this work we only consider faults in BLEs; faults in routing blocks are not considered. We use a BLE level fault model, which assumes that any fault in a BLE renders it unusable, and its functionality needs to be mapped to a different spare BLE. Two different models for distribution of faults are used in our experiments:

- Independent Fault Model: This model assumes that faults in a BLE are independent—the probability of a BLE being faulty is independent of the state of the neighboring BLEs.
- Clustered Fault Model: This model assumes that faults occur in clusters. Thus, if a BLE is faulty then its neighboring BLEs too have a higher probability of being faulty. This model is simulated by assuming that a BLE has a fixed probability of being the center of a fault cluster of radius R. The BLEs within distance R from the center are marked faulty with an exponentially decreasing probability distribution $f(X, \lambda) = \lambda e^{-\lambda X}$.

C. *Fault Detection*

In this work the methods presented are targeted towards tolerating the faults; it is assumed that faults have been previously detected and diagnosed to obtain the fault map. [18, 19, 20] discuss techniques for LUT based FPGA fault detection and diagnosis.

III. Problem Formulation

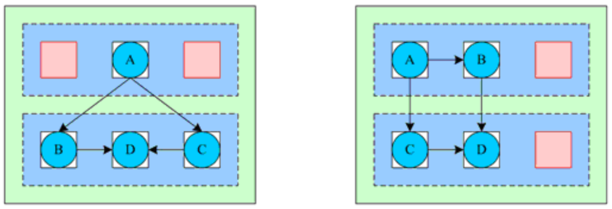
We represent a BLE netlist as a directed acyclic graph $G = (V, E)$, where the vertex set V represents the BLEs and the edge set E represents (directed) connections between BLEs. To avoid confusion between the BLEs of a netlist and the physical BLEs on an FPGA, we will refer to the netlist BLEs as nodes. A delay function $d\colon A \times A \rightarrow R^+$ gives the interconnect delay times between BLEs on the FPGA. Typically, the delay over an intra-cluster interconnect is small compared to the delay over an inter-cluster interconnect.

A mapping of the netlist into the FPGA is given by the injective function $f\colon V \rightarrow A$. Here, a node v is implemented by a BLE a if $f(v) = a$. Note that this function is injective since no two nodes can be mapped to the same BLE. It is not necessary that f be surjective. The set of remaining BLEs that are not used, which we denote as $S = A - f(V)$, is the set of spare BLEs to be used for fault-tolerant reconfiguration.

We will represent a circuit as a 5-tuple $\hat{C} = (G, A, C, d, f)$ comprising a netlist G, an FPGA C with BLE set A, a delay function d and a mapping f. Delay through the circuit, denoted $\Delta(\hat{C})$, is computed as the maximum arrival time in the circuit which will always occur at a primary output (PO). The problem of tolerating logic faults in FPGAs is formulated as the following sub-problems:

1. Effective distribution of inherent spare BLEs over the FPGA, during placement, to maximize chances of successful recovery from logic faults within the circuit's timing constraints.
2. Fault Reconfiguration to replace defective BLEs with spare BLEs, by rerouting a faulty BLE's connections to a spare BLE. A valid reconfiguration must meet the target circuit delay.

Typically, placement algorithms aim to minimize the estimated wire-length and timing delays. Figure 2 shows two possible placements for an example netlist. Assume that the architecture used contains three BLEs per CLB, each BLE having a delay of 1 unit, and the intra- and inter-cluster connection delays being 0 and 3 units respectively. We will refer to the placement on the left as first placement and the placement on the right as second placement. Both these placements have a critical path delay of 6 units. It can be easily show that the second placement has better fault tolerance properties than the first placement.



Circuit Delay	Probability
6	0.89
9	0.09
---	0.02

Circuit Delay	Probability
6	0.97
9	0.01
---	0.02

Figure 2. Impact of Placement on Fault Tolerance –the placement on the right can tolerate a BLE fault within a timing constraint of 6 units, but the placement on the left cannot.

In the event of failure of node D, the first placement does not have any spares in the containing CLB to cover the fault, and the node needs to be mapped to one of two spare BLEs in the other CLB. This causes the critical path delay to increase to 9 units because of two inter-cluster connections on the critical path. However, in the second placement, the faulty node can be replaced with a spare BLE in the same CLB without causing any increase in the critical path delay. Figure 2 shows the probabilities of various circuit delay values after reconfiguration, assuming a fault rate of 10%. Assuming a target circuit delay of 6 units, the second placement has a much higher probability of meeting the delay constraint upon fault reconfiguration. This clearly illustrates the need for fault tolerance awareness at the placement stage. Reconfiguration of faulty BLEs within timing constraints, by direct replacement with spare BLEs (without disturbing non-faulty BLEs), requires spare elements to be appropriately located in the original placement.

Problem 1: Fault Tolerant Placement. Given a netlist G, FPGA C with BLE set A and a delay function d, find a mapping f of G into C which maximizes the probability of reconfiguring a set of random faults within the target circuit delay D.

The fault reconfiguration problem consists of assigning defective BLEs to spare BLEs by re-routing its I/O connections. Suppose we are given a BLE netlist already mapped onto the FPGA and a set of defective BLEs in the FPGA. Our objective is to determine an assignment of spares to defective BLEs for fault coverage without causing significant degradation in the performance of the circuit; we need to find an assignment such that the circuit meets some timing requirement.

Problem 2: Failure Assignment (Decision Version). Given a circuit $\hat{C} = (G, A, C, d, f)$, a set of failed BLEs $F \subseteq A$ and a target delay $D \in R^+$, find a modified, injective mapping f' with properties

1. $f'(v) = f(v)$ *whenever* $f(v) \notin F$

2. $f'(v) \notin F$ *whenever* $f(v) \in F$

such that the delay through circuit $\hat{C}' = (G, A, C, d, f')$ satisfies $\Delta(\hat{C}') \leq D$.

IV. Fault Reconfiguration

A. Complexity Results

It can be shown that the NP-complete problem Exact Cover by 3-Sets [17] can be reduced to the failure assignment problem using a cluster size of 4. We also found that the Failure Assignment problem is not only difficult to solve optimally, but also hard to solve approximately within any constant factor of optimal. This can be shown by a simple modification of the reduction from the Exact Cover by 3-Sets problem.

Theorem 1: The Failure Assignment problem is NP-complete (in the strong sense) when the cluster capacity is a fixed constant $c \geq 4$.

Theorem 2: The Failure Assignment problem is NP-hard to approximate within a factor of 2 of the optimal solution when the cluster capacity is a fixed constant $c \geq 4$.

The proofs have been omitted for brevity.

B. Branch & Bound Reconfiguration Algorithm

Given the hardness of computation of the reconfiguration problem, here we present a simple branch & bound algorithm for reconfiguration of defective BLEs within the circuit's target delay constraint. By carefully ordering the selection of defective BLEs, for replacement with spare BLEs, the algorithm is able to quickly find a feasible reconfiguration.

Initially, the maximum available slack (under the delay constraint) at each node is computed by performing static timing analysis on the circuit. Then for each faulty node, the list of potential replacement spares is computed. A spare is a candidate replacement for a faulty node if rerouting the defective BLE's connections to the spare BLE, does not result in violation of the timing constraint.

Now the faulty nodes are iteratively selected for reconfiguration in ascending order of the number of available replacement spares. For each faulty node, we greedily pick the spare which results in least degradation in the node's slack. After reassigning the fault node to this spare, resulting timing changes are incrementally propagated by performing a breadth-first traversal of the circuit netlist DAG. If timing on any unassigned faulty nodes is affected, the list of candidate spares for each of these defective nodes is recomputed. If an unassigned faulty node is found to no longer have a potential spare replacement, the current node's spare assignment is rejected and a new replacement spare is attempted. When no candidate spare assignments exist for the current fault node, the algorithm backtracks by undoing the previous fault node's spare assignment and explores a different branch by attempting a different spare assignment for the previous faulty node.

The algorithm successfully terminates when a valid reconfiguration has been found by assigning all defective nodes to spare BLEs within the timing constraint. The algorithm terminates indicating reconfiguration failure if:

1. It backtracks to the topmost level and there are no new spare assignments to be tried for the first faulty BLE.
2. Times-out, failing to find a feasible reconfiguration after attempting certain (fixed) number of spare BLE assignments to fault nodes.

C. Ripple Move Reconfiguration Algorithm

This algorithm is inspired by the ripple move placement legalization procedure in Mongrel [18]. Here each fault is reconfigured by performing a sequence of node moves originating at the CLB with a faulty BLE and terminating at a CLB with a spare BLE, effectively rippling fault nodes to spare BLEs. For ease of exposition, we will henceforth refer to a CLB with one or more faulty BLEs as a faulty CLB.

Fault nodes are reconfigured in order of timing criticality. For each fault node, a directed acyclic graph (DAG) is constructed with the faulty CLB as source and the *K* nearest CLBs with spare BLEs as destinations. *K* is an algorithm parameter that determines reconfiguration quality — a higher value of *K* evaluates more destination spares for each fault's reconfiguration, improving likelihood of finding a valid reconfiguration, at the cost of increased runtime. Edges in the DAG correspond to moves of nodes from one CLB to another. To ensure low timing degradation, the node selected for a move is always the one which will have the highest slack (among all nodes in the source CLB) on being reassigned to the adjacent CLB. Further, only moves to adjacent CLBs in the direction of the destination CLB containing the spare BLE are allowed preventing cycles in the graph. Edges are weighted by the resultant decrease in the node's slack upon moving it from its existing location to the adjacent CLB — lower timing degradation corresponds to smaller weights. Finally, the reconfiguration path for the fault node is determined by finding the shortest (least cost) path between the source CLB and any of the *K* destination CLBs with a spare BLE. The fault is reconfigured by moving nodes to adjacent CLBs along this shortest path. This process is repeated for each fault node, till all faults are successfully reconfigured or no path is found for some fault node resulting in reconfiguration failure.

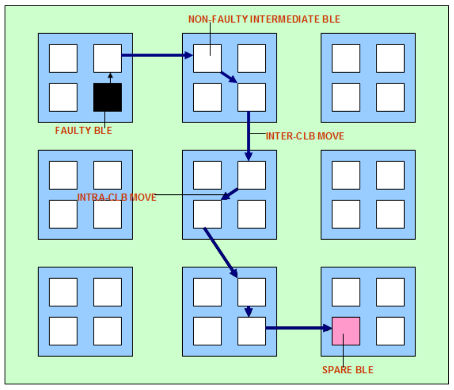


Figure 3. Ripple Move Reconfiguration

The branch & bound reconfiguration procedure only involves direct moves from faulty CLBs to CLBs with spare BLEs. In contrast, the ripple move reconfiguration method also involves moving non-faulty nodes in other CLBs to improve the chances of finding a reconfiguration within the target delay constraint with low timing degradation. Consequently, this algorithm involves relatively higher number moves for reconfiguration and hence more re-routing as part of the reconfiguration procedure.

D. Reconfiguration Runtime Complexity

Let n be the number of spare BLEs and k be the number of BLE faults. For the branch & bound algorithm the search space consists of all possible fault to spare assignments and is exponential in size — ${}^{n}P_{k}$. Since only spares within the 'slack neighborhood' of a faulty node are candidate replacements, in practice the size of the solution space is much smaller than this exponential upper bound. We bound the exponential worst-case run-time by limiting the number of assignments attempted to a fixed constant. Further, selection of faulty nodes in order of criticality leads the algorithm to feasible solutions quickly.

The ripple move reconfiguration procedure requires finding the shortest path to a spare CLB for each fault node. The worst case time complexity of finding a shortest path is N^2, where N is the total number of CLBs in the FPGA. The total time complexity of this algorithm is the time required for reconfiguring k faults — kN^2.

V. Fault Tolerant Placement

A. SCPlace

Simulated annealing (SA) is one of the most popular approaches for FPGA placement, yielding high-quality placements with low estimated wire-length and circuit delay. VPR [15, 16] is a state-of-the-art simulated-annealing based FPGA placement framework with tools for clustering and placement. In the clustering step, netlist nodes are packed into CLBs, minimizing the number of CLBs and inter-cluster connections on the critical path. During placement, CLB locations are determined using a simulated-annealing algorithm, with the objective of minimizing estimated wire-length and circuit delay.

SCPlace [13, 14] is a simulated annealing placement tool built on top of the VPR framework for simultaneous clustering and placement. Its main contribution is cluster optimization by allowing BLEs to move across CLBs during the placement step. It utilizes detailed physical placement information to improve the initial assignment of nodes to CLBs done at the clustering stage.

We developed and evaluated a couple of different placement techniques on top of SCPlace, which take fault tolerance into consideration as one of the placement objectives.

B. Even Distribution of Spares

We observed that the basic (non-fault tolerance aware) SCPlace placement algorithm pushes spare BLEs to the edge of the FPGA. As a result, these placements are not amenable to fault reconfiguration through replacement with spare elements. Our first technique adopts a simple approach to address this problem by evenly distributing available spare BLEs across the FPGA and locking them prior to start of the simulated-annealing placement procedure. These pre-placed spare BLEs are not displaced during placement — with spare BLEs evenly spread across the FPGA, upon occurrence of faults, spares close to the faulty BLEs are available for replacement without severely degrading circuit performance.

Critical path nodes have lower available slack and hence require higher proximity to spare nodes to avoid violation of target delay constraint upon reconfiguration. The even spare distribution method fails to account for timing criticality of nodes while determining spare BLE locations. The example in Figure 4 illustrates this problem. Let's assume the delay through a BLE to be 1, intra-cluster delays to be 0, and inter-cluster delay to be 3 units. Nodes A, B and C are on the critical path, and the circuit has a critical path delay of 6 units. Suppose that the target delay of the circuit is 6 units. Now if all of A, B and C become faulty, then for the first placement (evenly distributed spares) in Figure 4, there does not exist any possible reconfiguration which meets the target delay. On the other hand, with the second placement, these faults can be reconfigured within the timing constraint of 6 units, because of the spares being more appropriately located (in proximity of critical nodes).

C. Spare Demand Supply Based Approach

To address the problem illustrated above, besides the regular objectives of minimizing estimated wire-length and circuit timing, the placement algorithm should also maximize fault tolerance of the generated placement. Fault tolerance of a placement is defined as the probability of successful reconfiguration in event of logic faults. Placements with spare BLEs appropriately located, have better chances of recovering from faults within the target delay constraints, by replacing faults with nearby spares.

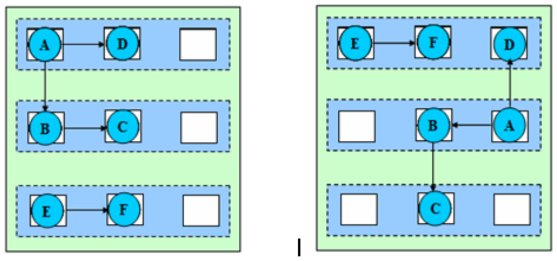


Figure 4. Drawback of even spare distribution –placement in the left figure cannot tolerate faults in all of nodes A, B and C without increasing circuit delay, while the placement in the right figure can.

We incorporate a new "*spare coverage cost*" component in the SCPlace simulated annealing objective cost function, in addition to the bounding box wire-length and timing delay cost components. This new cost component introduces fault tolerance awareness in the placement algorithm.

$$\textit{Total Cost} = \alpha C_W + \beta . C_T + (1 - \alpha - \beta) . C_S \qquad (4)$$

where C_W, C_T and C_S are wire-length, timing and spare costs respectively. α and β are constants that specify the relative weights of the timing and wire-length minimization placement objectives. SCPlace uses values of 0.5 for both α and β, there being no spare cost component. We use values of 0.45 and 0.5 for α and β, which leaves a weight of 0.05 for the fault tolerance objective. These values were empirically found to yield best results, without causing much effect on the wire-length and timing objectives.

If a node can relocate to a neighboring CLB within the available slack, the CLB is said to be in the *"slack neighborhood"* of the node. For each node, a constant spare "*demand*" is evenly distributed among all CLBs in its slack neighborhood. Nodes on the critical path have fewer CLBs in the slack neighborhood and hence place high demands on each neighbor. The total spare demand on a CLB is the sum of spare demands placed by all nodes that have that CLB in their slack neighborhood. The exact level of defect densities in nano-FPGAs is currently unknown, but it is assumed that 1-15% of the resources on a chip (wires, switches FETs, etc.) will be defective [7, 8, 9]. We assumed a maximum fault rate of 10% for our experiments, and thus the maximum number of expected faults is computed as the least of: i) Number of spare BLEs available ii) 10% of total node count in the netlist. Spare demands are normalized such that the total demand over all CLBs is equal to the maximum number of expected faults.

$$D(v_i, B_j) = 1/\text{No. CLBs covering node } v_i \qquad (5)$$

where $D(v_i, B_j)$ represents node v_i's demand on CLB B_j.

Spare BLE "*supply*" is modeled by distributing a supply of *1* for each spare BLE to the containing and neighboring CLBs. The supply received by a CLB is inversely proportional to its distance from the CLB containing the spare BLE. The *"coverage spread"* of a spare BLE is the farthest a CLB can be from the spare site to receive a portion of the supply and is determined by ratio of CLBs to spare BLEs in the circuit. The total spare coverage cost is sum of the gap between spare demand and supply over all CLBs.

$$C_S = \sum\nolimits_{CLBs} \max(0, D - S) \qquad (6)$$

where C_S is spare cost, D is the spare demand on a CLB and S is spare supply of the CLB.

Change in a node's assignment to a BLE during placement, results in a change in its slack and the list of CLBs in its slack neighborhood—which in turn changes the spare demands on the CLBs. We observe that the reassignment of nodes to BLEs results in very little change in the spare demand map, because of the following reasons:

1. During initial iterations of SA, slacks are high and nodes have large numbers of neighboring CLBs. Hence the demand is almost evenly distributed across the FPGA blocks.
2. At lower temperatures (final iterations of SA), most of the moves (changes in node to BLE assignments) are local and hence cause little change in the demand map.

Thus, inaccuracy due to not keeping spare demands up to date is low and tolerable. Spare demands are periodically re-computed to fix the inaccuracies in demand map.

However, spare "*supply*" values are sensitive to changes in spare locations because the spare count is usually much lower than the circuit's node count and a spare's supply is limited to fewer CLBs compared to the demands placed by nodes. Hence, spare supply values are accurately maintained by updating them upon any change in the location of a spare BLE during placement. The spare cost component of the SA objective cost function only changes upon relocation of a spare BLE and encourages placement of the spare BLEs in regions where the demand is highest. Other moves are evaluated solely based on their effect on the wire-length and timing delay cost components of the objective cost function.

D. Runtime Complexity

Let n be the total number of CLBs in the FPGA, N be the number of BLEs in each CLB and α be the fraction of fragment level (BLE) moves. The time complexity of SCPlace at each temperature of the simulated annealing process is $O((\alpha nN)^{4/3})$.

Spare demands and supply are re-computed a fixed number of times at each temperature. In worst case, each node in the netlist can place a demand on every CLB in the FPGA and the time complexity of demand computation is $O((\mathrm{k}(nN)n)^{4/3})$, k being the number of times spare cost is recomputed at each temperature. Spare supply computation takes constant time per CLB and is performed for each CLB in the FPGA, making it an $O(\mathrm{k}n)^{4/3}$ operation. Hence the overall time complexity of spare cost computation is $O((\mathrm{k}(nN)n)^{4/3}) + O((\mathrm{k}n)^{4/3}) = O((\mathrm{k}(nN)n)^{4/3})$. We used a value of *10* for k, which was empirically determined to yield good results.

VI. Experimental Results

Here we present results of our fault tolerant placement and fault reconfiguration schemes, from tests performed on the 20 largest MCNC benchmarks. Results of the circuits *bigkey*, *dsip* and *des* have been excluded, since with them having more spare BLEs than nodes in the netlist, the fault reconfiguration problem for these circuits is trivial.

Following is description of the experiment procedure:

1. Perform placement using each of the two different placement schemes, viz. even spares distribution and demand-supply-based approach.
2. Inject desired number of faults randomly or as per the clustered fault model across the placed circuit *20* times, each time with a different random seed, for generating *20* different fault maps.
3. Perform reconfiguration using the proposed branch & bound and ripple move algorithms and measure success percentages by counting number of fault configurations for which the algorithms succeed in finding a feasible reconfiguration within target delay.

Fault reconfiguration is tested for different fault rates between 50-100% of the maximum number of expected faults (least of spare count and 10% of node count). The target circuit delay constraint was set to 1% higher than the maximum critical path delay in placements obtained from the basic SCPlace, even spare distribution and demand-supply-based placement methods. The choice of this strict timing constraint was been made to test the effectiveness of the proposed reconfiguration techniques under stringent circuit performance requirements.

Table I shows the critical delay and wire length costs of placements obtained from the two placement methods. The spare demand-supply-based method, on an average, does not have any degradation in critical path delay compared to basic SCPlace placements and has only 7.4% higher wire-length cost. This illustrates that reconfiguration-friendly placements can be generated without significantly compromising the placement quality.

Table II. Placement Results

	Even Spares distribution		Spare Demand Supply	
Circuit	% Circuit Delay increase	% Wire-length cost	% Circuit Delay increase	% Wire-length cost increase
alu4	0.65	4.73	-2.32	10.55
apex2	2.96	1.98	3.37	6.89
apex4	1.54	1.49	3.58	6.61
clma	-2.27	3.92	-1.20	11.42
diffeq	1.08	4.59	-2.99	12.97
elliptic	0.82	5.94	2.00	13.77
ex5p	1.00	6.82	3.24	10.08
ex1010	1.19	0.07	0.54	1.58
…..	…..	…..	…..	…..
AVERAGE	0.12	2.59	0.01	7.41

Placement quality results for the even spare distribution and spare-demand-supply based placement methods.

Table II shows comparative results of the 2 reconfiguration algorithms for the even spare distribution and demand-supply-based placement methods. Reconfiguration success rate is the fraction of times all faults were successfully reconfigured within the target delay constraint, from among 20 different fault maps. Timing degradation is the percentage increase in the circuit critical delay upon reconfiguration compared to the critical delay of the original non fault tolerant SCPlace placement.

For the branch & bound reconfiguration method spare-demand-supply based placements have an average of 36% and 44% higher reconfiguration success rates compared to even spare distribution placements, for random and clustered faults respectively. Also, the timing degradation upon reconfiguration is up to 56% lower for demand-supply based placements. This demonstrates that fault tolerant placements with spare BLEs appropriately located are more amenable to fault reconfiguration within the target circuit delay and incur lower timing degradation.

The ripple move reconfiguration algorithm yields far better (up to 47%) reconfiguration success rates and lower (up to 30%) timing degradation, compared to the branch & bound reconfiguration method. This clearly illustrates the effectiveness of allowing greater flexibility in choosing moves during the reconfiguration procedure. Also, here the

Table I. Branch & Bound vs Ripple Move Reconfiguration Methods

	Reconfiguration Success Rate (%)			Timing Degradation (%)		
Test Configuration	Branch & Bound	Ripple Move	%Increase	Branch & Bound	Ripple Move	%Increase
Even Spare Distribution with independent fault distribution	54.71	80.88	47.83%	0.97	0.69	-28.87%
Demand-supply placement with independent fault distribution	74.41	84.71	13.84%	0.43	0.49	13.95%
Even Spare Distribution with clustered fault distribution	33.53	62.06	85.09%	0.55	0.81	46.61%
Demand-supply placement with clustered fault distribution	48.24	62.65	29.88%	0.70	0.73	4.70%

Random and clustered fault reconfiguration results for Branch & Bound and Ripple Move reconfiguration methods.

demand-supply based placements have an average of 4% higher reconfiguration success rate and 30% lower timing degradation compared to even spare distribution placements. This strengthens the claim that considering fault re-configurability at the placement stage improves fault tolerance of the design, even for elaborate reconfiguration methods.

Finally, the reconfiguration success rates for clustered faults are much lower compared to random fault distribution, because of a large number of faults being localized within a region. Fault tolerant placements perform much better than even spare distribution placements for the clustered fault model, because reconfigurability under this model is much more sensitive to spares being appropriately located in proximity of critical nodes.

VII. Related Work And Conclusions

Prior work on FPGA fault tolerance methods can be broadly categorized into two groups, based on the level of abstraction at which faults are tolerated [1]. *"Device-Level" (DL)* methods attempt to construct a fault-free array from a larger array containing faulty resources. Techniques in this category [2, 3, 5 10] typically introduce redundant resources and reconfigure faults by replacement with the spare resources. These methods require re-routing of only the faulty elements' connections and hence result in fast reconfiguration. However, additional redundancy requirements result in increased area requirements and these methods are limited in the number of defects that can be resolved. Further, the placement of spare resources is pre-determined and is not optimized for each netlist. Consequently, reconfigured circuits may have significant degradation in circuit performance.

Configuration-level (CL) methods attempt to map the system function to fault free resources through reconfiguration. While these techniques [4, 6, 11, 12] do not require additional redundancy, their primary drawback is the time consuming reconfiguration step to place and route the circuit according to the new configuration. Similar to our ripple move reconfiguration method, the min-cost flow fault reconfiguration method proposed in [11], allows relocation of intermediate non-faulty nodes. However, this work does not consider optimization of spare BLE locations at the placement stage for improving defect reconfigurability.

We propose that the placement step be made fault tolerance aware, to optimize locations of inherent spare resources for maximizing probability of reconfiguration within target delay constraints. Our fault tolerant placement algorithm avoids additional redundancy by strategically placing the inherent unused BLEs in the FPGA. Reconfiguration of faults in the branch & bound method is done by re-routing only the faulty BLE connections, requiring minimal changes to the original layout of the FPGA and little degradation in the circuit's performance. Experimental results demonstrate the effectiveness of this approach—large fault counts are tolerated with high success rates and very low degradation in circuit performance. The ripple move reconfiguration algorithm utilizes the flexibility of relocating non-faulty nodes to achieve much higher reconfiguration success rates at the expense of more re-routes.

VIII. References

[1] J. A. Cheatham, J. M. Emmert, and S. R. Baumgart, "A Survey of Fault Tolerant Methodologies for FPGAs," *ACM Transactions on Design Automation of Computer Systems*, April, 2006.

[2] F. Hatori, T. Sakurai, K. Sawada, M. Takahashi, M. Ichida, M. Uchida, I. Yoshii, Y. Kawahara, T. Hibi, Y. Saeki, H. Muroga, A. Tanaka, K. Kanzaki, "Introducing redundancy in field programmable gate arrays", *Proceedings of the IEEE Custom Integrated Circuits Conference*, Volume 7, Number 1, pp. 1-4.

[3] F. Hanchek, S. Dutt, "Design methodologies for tolerating cell and interconnect faults in FPGAs", *International Conference on Computer Design*, pp. 326–331, 1996a.

[4] F. Hanchek, S. Dutt, "Node-covering based defect and fault tolerance methods for increased yield in FPGAs", *Proceedings of the 9th International Conference on VLSI Design*. Pp. 225–229, 1996b

[5] N. J. Howard, A. M. Tyrell, N.M. Allison, "The yield enhancement of field programmable gate arrays", *IEEE Trans. VLSI Syst. 2*, Volume 1, pp. 115–123, Mar 1994.

[6] V. Lakamraju, R. Tessier, "Tolerating operational faults in cluster-based FPGAs", *Proceedings of the ACM International Workshop on Field Programmable Gate Arrays*, pp. 187-194, 2000.

[7] Y. Chen et al., "Nanoscale Molecular-Switch Crossbar Circuits,", *Nanotechnology*, vol.14, no. 4, pp. 462-468, 2003.

[8] Y. Huang et al., "Directed Assembly of One-Dimensional Nanostructures Into Functional Networks,", *Science,* vol. 291, no. 5504, pp. 630–633, 2001.

[9] M. Mishra, S.C. Goldstein, "Defect Tolerance at the End of the Roadmap," *Int'l Test Conference Proceedings*, 2003.

[10] S. Durand, C. Piguet, "FPGA with selfrepair capabilities", *Proceedings of the ACM International Workshop on FPGAs,* 1994

[11] A. Mathur, C. L. Liu, "Timing-Driven Placement Reconfiguration for Fault Tolerance and Yield Enhancementin FPGAs, *Proceedings of the European conference on Design and Test*, pp. 165, 1996

[12] J. Emmert, D. Bhatia, "Incremental routing in FPGAs", *Proceedings of the 11th Annual IEEE International ASIC Conference*, 1998

[13] G. Chen and J. Cong, "Simultaneous Timing-Driven Clustering and Placement for FPGAs", *Proc. International Conference on Field Programmable Logic and its Applications*, August 2004

[14] G. Chen and J. Cong, "Simultaneous Timing-Driven Placement and Duplication", *Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, February 2005.

[15] A. Marquardt, V. Betz and J. Rose, "Timing-Driven Placement for FPGAs", *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 203 – 213, February 2000

[16] A. Marquardt, V. Betz and J. Rose, "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density", *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 37-46, 1999

[17] Michael R. Garey, David S. Johnson, *"Computers and Intractability: A Guide to the Theory of NP-Completeness"*, W.H. Freeman and Company, 1979.

[18] Shyue-Kung Lu, Fu-Min Yesh, Jen-Shing Shih, *"Fault Detection and Fault Diagnosis Techniques for LookupTable FPGAs"*, VLSI Design, 2002

[19] Chi-Feng Wu and Cheng-Wen Wu, *"Fault Detection and Location of Dynamic Reconfigurable FPGAs"*, International Symposium on VLSI Technology, Systems, and Applications, 1999.

[20] E. Bareisa, V.Jusas, K.Motiejunas, R.Seinauskas, *"Testing of FPGA Logic Cells",* Elektronika IR Elektrotechnica, 2004