Efficient Reconfigurable Techniques for VLSI Arrays With 6-Port Switches

Wu Jigang, Thambipillai Srikanthan, and Heiko Schröder

***Abstract*—This paper proposes an efficient techniques to reconfigure a two-dimensional degradable very large scale integration/wafer scale integration (VLSI/WSI) array under the row and column routing constraints, which has been shown to be NP-complete. The proposed VLSI/WSI array consists of identical processing elements such as processors or memory cells embedded in a 6-port switch lattice in the form of a rectangular grid. It has been shown that the proposed VLSI structure with 6-port switches eliminates the need to incorporate internal bypass within processing elements and leads to notable increase in the harvest when compared with the one using 4-port switches. A new greedy rerouting algorithm and compensation approaches are also proposed to maximize harvest through reconfiguration. Experimental results show that the proposed VLSI array with 6-port switches consistently outperforms the most efficient alternative proposed in literature, toward maximizing the harvest in the presence of fault processing elements.**

***Index Terms*—Degradable very large scale integration/wafer scale integration (VLSI/WSI) array, fault-tolerance, greedy algorithm, reconfiguration, VLSI routing.**

I. Introduction

The mesh-connected processor array has a regular and modular structure and allows fast implementation of many signal and image processing algorithms. With the advancement in very large scale integration (VLSI) and wafer scale integration (WSI) technologies, integrated systems can now be built on a single chip or wafer by interconnecting a large number of processing elements (PEs), such as processors or memory cells [1]. As the density of VLSI/WSI arrays increase, the probability of the occurrence of defects in the arrays during fabrication also increases. Thus, fault-tolerant techniques must be employed to enhance the yield and reliability of the arrays.

There are generally two methods for reconfiguration, namely, the redundancy approach [2]–[4] and the degradation approach [5]–[8]. The techniques in this paper belong to the latter, which treats all PEs of the system in a uniform way and uses as many fault-free PEs as possible to construct the target system. Different algorithms have been reported in [6]–[8], which are based on the array connected by 4-port switches, and each PE has two internal bypass links. Such an architecture is unable to support rerouting two neighboring PEs lying in same physical row into same logical column.

In this paper, we focus on the design and analysis of efficient greedy algorithms since the problem is NP-complete [5]. In an attempt to increase the harvest, we form a 6-port switch consisting of a 4-port switch and one bypass link. Unlike the existing architecture, in which the bypass is within the PE, we incorporate the bypass capability within the switch. This provides for the ability to reroute two neighboring PEs lying in same physical row into same logical column to obtain higher harvest. At the same time, we eliminate need to incorporating a bypass circuitry within a PE.

Manuscript received June 12, 2003; revised April 25, 2004. This paper was presented in part at the 8th International Conference on Computing and Combinatorics, 2002.

W. Jigang and T. Srikanthan are with the Centre for High Performance Embedded Systems, Nanyang Technological University, 639798 Singapore (e-mail: asjgwu@ntu.edu.sg; astsrikan@ntu.edu.sg).

H. Schröder is with the School of Computer Science and Information Technology, RMIT University, Melbourne, Vic. 3001, Australia (e-mail: heiko@cs.rmit.edu.au).

Digital Object Identifier 10.1109/TVLSI.2005.853603

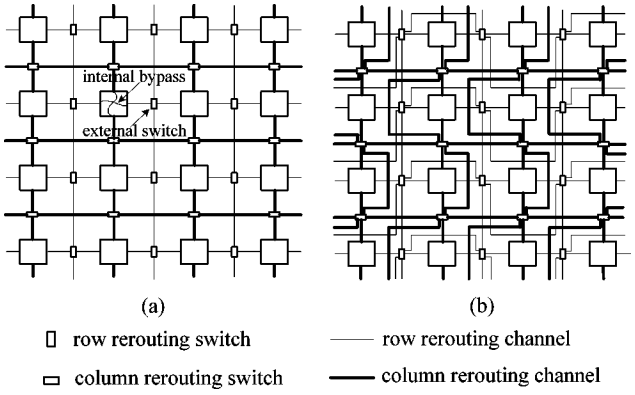

Fig. 1. 4×4 arrays linked by 4-port and 6-port switches, respectively.

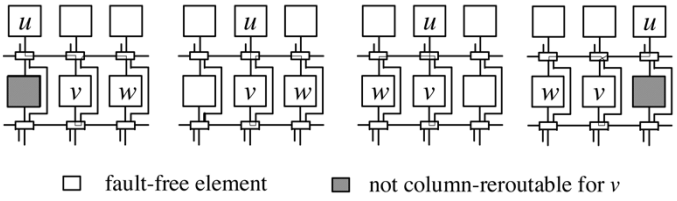

Fig. 2. Rerouting two PEs v and w into the same logical column based on 6-port switches. u is the predecessor of v and $\mathrm{col}(u) \neq \mathrm{col}(w)$.

II. Preliminaries

The original array after manufacturing is called a physical array or host array which may contain faulty PEs. A degradable subarray of the physical array, which contains no faulty PE, is called a logical array or target array. The rows (columns) in the physical array are called physical rows (columns). The rows (columns) in logical array are called logical rows (columns). In this paper, $\mathrm{row}(e)$ $(\mathrm{col}(e))$ denotes the physical row (column) index of the PE e. H (S) denotes the physical (logical) array. R_i denotes the ith logical row. $e(i,j)$ $(e'(i,j))$ denotes the PE located in the ith row and in the jth column of physical (logical) array. Fig. 1(a) shows the architecture in [5]–[8], where each PE has two internal bypass links, and thus can be converted into a connecting one if necessary. In a host array, if the PE $e(i,j)$ can communicate directly to $e(i',j)$ with external switches, where $|i'-i| \leq d$, we call d the row compensation distance. Keeping the same assumption as in [7], d is set to 1 in this paper.

Typical routing schemes include the row (column) bypass scheme and the row (column) rerouting scheme. In the row bypass scheme, a PE u in row i can be directly connected to another PE v in the same row. In the process, all PEs in row i that lie between u and v are bypassed. In the row rerouting scheme, a fault-free PE u in physical row i can be directly connected to another fault-free PE v in physical row i' if $|i'-i| \leq d$. Thus, in the row rerouting scheme, each row has both bypass and rerouting capabilities. The column bypass scheme and the column rerouting scheme can be similarly defined. The reconfiguration problem in this paper is formulated as follows.

Given an $m \times n$ mesh-connected host array, integers r and c, find an $m' \times n'$ fault-free subarray under the row and column rerouting scheme such that $m' \geq r$ and $n' \geq c$.

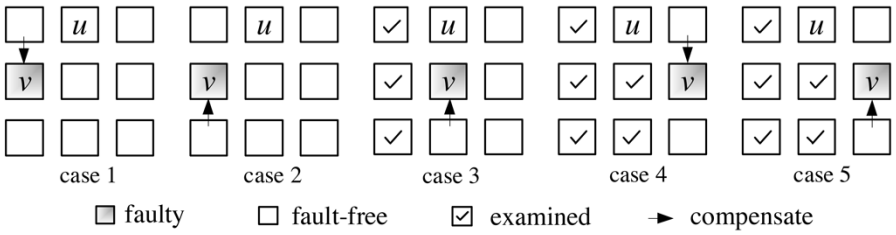

Fig. 3. Five cases occurred in local compensation.

Based on the architecture shown in Fig. 1(a), the most efficient algorithm under the row and column rerouting is described in [7], denoted as *RCRT* in this paper. It consists of two procedures called *GCR* and *LRE*. *GCR* is used for finding a target array that contains a set of selected logical rows. It reroutes the fault-free PEs to form logical columns, the successor of the fault free PE u in R_i is selected from the set of the adjacent of u, $\mathrm{Adj}(u)$, in a left-to-right manner, where $\mathrm{Adj}(u) = \{v : v \in R_{i+1}, v \text{ is fault-free}, |\mathrm{col}(u) - \mathrm{col}(v)| \leq 1\}$. *LRE* selects one row, say R_γ, to be excluded from a set of logical rows that was previously selected from host array and uses it to compensate for faulty PEs in its neighboring rows. *GCR* and *LRE* are executed iteratively until the target array is found. As the same assumptions as in [7], faults are considered to be associated only with PEs. Interconnects, reconfiguration controllers, and switches are assumed to be fault free.

III. Proposed Architecture

The 4-port switch occupies less area than 6-port switch and supports a more realistic assumption of fault-free switches and interconnects. This ignores the additional bypass circuitry within each PE. Moreover, two neighboring PEs lying in same physical row cannot be rerouted into same logical column. The proposed 6-port switch consists of pass gates to establish all possible connection pair among the six input rails. In other words, any pair combination of the six ports can be used to establish a pair wise connection. The only restriction is that no port can be connected to more than 1 port. Fig. 1(b) shows an array linked by the new switch model. In the new architecture two neighboring PEs can communicate directly via the external switches. Internal bypass links through PEs are not involved as a PE can be bypassed through tracks that run externally. Moreover, the proposed architecture is capable of allocating two neighboring fault-free PEs on the same physical row into same logical column. Fig. 2 shows the feasible routing manners.

IV. Algorithms

A. New Column Rerouting

Assume v is faulty. Its *upper (lower) neighbor* is defined as the fault-free PE $e'(i,j)$, where $i = \mathrm{row}(v) - 1$ $(\mathrm{row}(v) + 1)$, and $j = \mathrm{col}(v)$. The compensation for v with its upper or lower neighbor during column rerouting is called local compensation for v. Fig. 3 shows the five cases that might occur during local compensation. The switches and the links are omitted. Since the local compensation considers the faulty PEs, we extend the definition of $\mathrm{Adj}(u)$ to $\mathrm{Adj}a(u) = \{v : v \in R_{i+1} \text{ and } |\mathrm{col}(u) - \mathrm{col}(v)| \leq 1\}$, where the PEs in $\mathrm{Adj}a(u)$ are ordered in increasing column numbers for each $u \in R_i$. It is clear that $|\mathrm{Adj}a(u)| \leq 3$.

Our greedy rerouting algorithm, denoted as *New_GCR*, attempts to connect the PE u to the leftmost PE v of $\mathrm{Adj}a(u)$ that has not been previously examined. In *GCR*, if this step fails in doing so, a logical column containing the current PE u cannot be formed and backtracking occurs. But in *New_GCR*, local compensation is employed, *i.e.*, the upper or lower neighbor of v is examined to compensate v whenever possible. *New_GCR* backtracks to the previous PE p, connected to u, only if the local compensation fails. It then attempts to connect p to the leftmost PE of $\mathrm{Adj}a(p) - \{u\}$ that has not been previously examined. The selection of the leftmost PE is a greedy choice and the local compensation potentiates the greedy choice. To simplify the column rerouting, rerouting u_{i+1} to the successor of u_i is not allowed.

Fig. 4 outlines *New_GCR*, in which, to shorten the paper, *Local_Comp(v)* is assumed to be the procedure to finish the local compensation for the faulty v. It returns the compensated v (*fault-free* for successful compensation or *fault* for reverse) by examining the cases shown in Fig. 3, from case 1 to case 5. In Fig. 4, logical columns are constructed in left-to-right manner (see step 3). Each logical column is produced in a top-to-down manner. The first PE of each logical column is chosen from R_0 or R_1 by step 3.2, in which local compensation is employed if the current PE in R_0 is faulty. Step 3.4 continually constructs the remaining part of the current logical column. In *New_GCR*, for each $u \in R_i$, at most eight interconnects (see Fig. 3) are examined at each step and each valid interconnect is examined at most twice. Thus, following the analysis for *GCR*, it can be deduced that *New_GCR* is in linear time $O(N)$ for a host array of N fault-free PEs.

B. New Compensation Strategy

Assume the previously selected rows are $R_0, R_1, \ldots, R_k$; R_γ is to be excluded; $e'(\gamma, j) \in R_\gamma$ and it is fault-free, where $0 < \gamma < k$ and $1 \leq j \leq n$. In RCRT, compensations only occur in $R_{\gamma-1}$ and $R_{\gamma+1}$. When $e'(\gamma - 1, j)$ and $e'(\gamma + 1, j)$ are both fault-free, $e'(\gamma, j)$ will not be utilized in the successive reconfigurations though it is fault-free as R_γ will be excluded. In our algorithm, however, the *nearest upper fault* of $e'(\gamma, j)$, say $e'(up, j)$, will be found. Then $e'(up, j)$ will be compensated with $e'(up + 1, j)$ and $e'(up + 1, j)$ will be replaced by $e'(up + 2, j)$ and so on. This process will continue until $e'(\gamma - 1, j)$ is replaced by $e'(\gamma, j)$. If the PEs considered for compensation do not satisfy the limitation of compensation distance $(d \leq 1)$, the algorithm will turn to find the *nearest lower fault*, and then a similar compensation process will be done in the PEs below $e'(\gamma, j)$. The aim of doing so is to utilize as many fault-free PEs as possible in R_γ.

Let *Up_Comp* (k, γ, j) be the function to search for the nearest upper fault of $e'(\gamma, j)$ in $R_0, R_1, \ldots, R_k$, then do compensations and replacements from $e'(up+1, j)$ to $e'(\gamma, j)$, one after another. To shorten the paper, we omit its formal description. The function will return 1 for successful compensation, or 0 for the reverse. There is no need for compensation in the case of all upper PEs of $e'(\gamma, j)$ are fault-free. In this case, *Up_Comp* terminates and returns 0, and $e'(\gamma, j)$ is saved to compensate its nearest lower fault. The time complexity of *Up_Comp* is bounded by $O(N_j)$ for N_j fault-free PEs in the jth column of the current array. Symmetrically, let *Down_Comp* (k, γ, j) with the same time complexity as that of *Up_Comp* (k, γ, j) to search for the nearest lower fault and implement its compensation. The compensations for the whole logical array with all fault-free PEs in R_γ are described in *Overall_Comp* shown in Fig. 5 (upper part). It runs before R_γ is excluded. Its time complexity is $O\left(\sum_{j=0}^{n} N_j\right)$, *i.e.*, $O(N)$ for the host array of N fault-free PEs.

```
Input: the host arrary H and the logical rows R0, R1, ..., Rk-1
Output: the target array with n columns.
Procedure New_GCR(H, R0, R1, ..., Rk-1, n )
begin
1  n:=0;
2  for cach u in R0, R1, ..., Rk-1 do      /* Initialize data */
       begin  unmark u;  pred(u):=nil;  end;
3  while there are unmarked PEs in R0 do
   begin /* Create logical column one by one from left to right */
   3.1 cur0 := the leftmost unmarked PE in R0 ,
   3.2 while cur0 is faulty do /* look for the first place/PE--*/
       begin                  /* -- in R0 for the current logical column */
            Local_Comp(cur0);
            if (cur0 is fault-free ) then cur:= cur0 /* fixed */
            else begin /* continue the search for the first PE */
                 mark cur0 ;
                 cur0 :=the leftmost unmarked PE in R0
                 end; /* of if */
       end; /* of while */
   3.3 mark cur;
   3.4 repeat /* continue the current routing in the remain rows */
           if there are unmarked PEs in Adja(cur)
           then begin /* greedy strategy is used to form the column */
                v:= leftmost unmarked PE in Adja(cur);
                Local_Comp( v);
                if ( v is fault-free ) and ( row(cur)<row(v)
                  or (row(cur)=row(v) and col(pred(cur)) ≠ col(v)))
                then begin  pred(v):=cur;  cur:=v;  mark v end
                else  Restore the changed PEs in local
                      compensation into their original state;
                end /* of if */
           else if cur ∉ R1 then  cur:=pred(cur); /* backtrack */
       until (cur∈ Rk-1 ) or (cur∈ R0);
   3.5 if (cur∈ Rk-1 ) then n:=n+1; /* Get a new logical column */
   end; /* of while*/
end.
```

Fig. 4. Formal description of the algorithm *New_GCR*.

```
Input: the logical array {Ri}, 0≤ i ≤ k with Rγ to be excluded.
Output: the compensated logical row;
Procedure Overall_Comp( k, γ)
/*Compensate with Rγ in whole m×n array*/
begin
1  for j:= 0  to  n  do
   1.1 if  (k > 2 ) /* for more than 2 logical rows*/
       then  if e'(γ, j) is fault-free then
                 begin
                  flag := Up_Comp( k, γ, j );
                  if flag = 0 then Down_Comp( k, γ, j );
                 end /* of if */ ;
   end /* of for */;
end.

Input: the host array H, the given constants r, c.
Output: a target array.
Procedure New_Row_First(H, m, n, r, c)
/* find a row-based target array of maximal size */
begin
1  Let S={ R0, R1, ..., Rm };
   New_GCR (H, S, n); /* Construct a target array */
2  m' := m;
3  while ( m' ≥ r) and ( n < c ) do
    begin
     3.1  Call approach[7] to select the row Rγ from S;
     3.2  Overall_Comp(m', γ) ;
     3.3  Delete the row Rγ from S;
     3.4  New_ GCR (H, S, n);
     3.5  m':=m'-1;
    end;  /* of while */ ;
4  if (m' ≥ r) and ( n > c)
   then the target array is obtained
   else "algorithm failed";
end.
```

Fig. 5. Formal description of *Overall_Comp* and *New_Row_First*.

C. Main Algorithm

Assume *New_Row_First* (*New_Column_First*) is to find a row (column) based target array of maximum size. We outline the main algorithm as follows.

Algorithm New_RCRT
1) Call *New_Row_First* to find a target array of maximum size (say $m_1 \times n_1$) based on the row.
2) Call *New_Column_First* to find a target array of maximum size (say $m_2 \times n_2$) based on the column.
3) The resultant target array is $\max\{m_1 \times n_1, m_2 \times n_2\}$.

New_Row_First is shown in Fig. 5 (lower part). Initially, all rows in the host array are selected for inclusion into the target array (see step 1). Thus, each logical row is also a physical row. *New_GCR* is employed to construct the first feasible subarray. Step 3.1 selects one row in linear time [7] in each iteration. Compensation approach is employed in step 3.2, and then *New_GCR* produces a new feasible subarray in step 3.4 after excluding the selected row. Both step 3.2 and 3.4 run in $O(N)$, and thus each iteration of *New_Row_First* runs in $O(N)$. Following the analysis for *RCRT*, given an $m \times n$ host array H with N fault-free PEs and integers r and c, the time required by *New_RCRT* to find an $m' \times n'$ target arrays such that $m' \geq r$ and $n' \geq c$ is $O(\max\{(m-r)N, (n-c)N\})$ which is the same as that of *RCRT*.

V. Experimental Results

We compare *New_RCRT* with *RCRT*. The two algorithms are implemented in C on an Intel Pentium-III 500-MHz computer. In order to make a fair comparison, we maintain the assumptions made in [7], i.e., the faults in random host arrays were generated by a uniform random generator; The fault size in host array is from 0.1% to 10%. Both algorithms are tested with the same random input instances. The size of each target array obtained by *New_RCRT* is compared with 1) an upper bound on the size of target array and 2) the size of the target array obtained by *RCRT*. The upper bound of the target array size is calculated with the same method as described in [7]. Tables I and II summarize the experimental results.

Table I shows the target-array comparisons, both in maximal target array (MaxTA) and in maximal square target array (MaxSTA). For example, for the host array of size 256 × 256 with 10% faulty elements, the theoretical maximal target array is 234 × 252. The result derived from *New_RCRT* is 240 × 236, which is closer to the theoretical maximum of 234 × 252. Also, the target array size is notably compared to that produced by RCRT (i.e., 256 × 196). In addition, *New_RCRT* obtains a larger square target array of size 237 × 237 than that produced by *RCRT* (i.e., 208 × 207).

Table II shows the performance comparison, 20 random instances averaged, for 512 × 512 arrays. *harvest* is the ratio of target array size to the number of the good cells in host array. Degradation $= 1 - \lambda$, where λ is the ratio of target array size to the host array size. It can be see that the average harvest of *New_RCRT* is greater than 94% and the average degradation is less than 14% for each type of random instances. Especially for MaxSTA, the harvest can be improved by up to 20% for

TABLE I
COMPARISON OF TARGET ARRAY WITH THEORETICAL MAXIMUM

			MaxTA		MaxSTA	
Host Array	Fault (%)	Theoret. Maximum	RCRT	New_RCRT	RCRT	New_RCRT
64×64	0.1	64×63	64×63	64×63	63×63	64×63
64×64	1.0	63×64	64×61	63×63	62×62	63×63
64×64	10.0	61×60	63×49	60×59	54×53	58×59
128×128	0.1	127×128	128×127	127×128	127×127	127×127
128×128	1.0	127×127	126×124	127×126	124×124	125×125
128×128	10.0	117×126	125×99	121×117	106×106	118×118
256×256	0.1	255×256	255×254	255×255	254×254	255×255
256×256	1.0	255×254	255×248	253×254	249×249	253×254
256×256	10.0	234×252	256×196	240×236	208×207	237×237
512×512	0.1	511×512	512×508	512×510	509×509	511×510
512×512	1.0	509×509	511×498	509×507	498×498	508×507
512×512	10.0	469×503	512×388	505×452	413×413	474×473

TABLE II
AVERAGE PERFORMANCE COMPARISON FOR 512×512 ARRAYS

		$harvest(\%)$		$degradation(\%)$	
Targets	Faults(%)	RCRT	New_RCRT	RCRT	New_RCRT
MaxTA	0.1	99.41	99.63	0.69	0.47
	1.0	97.92	99.28	3.06	1.71
	10.0	84.89	96.05	23.60	13.55
MaxSTA	0.1	98.94	99.36	1.15	0.74
	1.0	95.37	99.16	5.58	1.83
	10.0	71.95	94.71	35.25	14.76

10% fault size. Hence, the new algorithm is more efficient than the old one.

The proposed architecture overcomes the drawbacks of previous approaches, be it at the expense of a small increase in hardware. Assuming that the chip area of a 6-port switch can be as much as 50% more than the 4-port switch, and the typical gate count for a PE is 50 000 gates. The increment in the ratio of the switching circuits to the mesh array is only 3.59% for a 256×256 array by simple calculation based on the analysis in hardware overhead in [9].

VI. CONCLUSION

We have proposed a 6-port switch based structure and associated algorithms for the reconfiguration of two-dimensional degradable VLSI arrays. The proposed architecture notably improves the harvest through improved connectivity. The bypass links internal to the processing elements are no longer required, thereby compensating for the increase in the complexity of a 6-port switch. Evaluations based on the new reconfiguration algorithms show that, the average harvest of *New_RCRT* is greater than 94% and the average degradation is less than 14% for a number of random instances, when compared with the latest techniques based on 4-port switches.

REFERENCES

[1] T. E. Mangir and A. Avizienis, "Fault-tolerant design for VLSI: Effect of interconnection requirements on yield improvement of VLSI design," *IEEE Trans. Comput.*, vol. C–31, no. 7, pp. 609–615, Jul. 1982.

[2] J. W. Greene and A. E. Gamal, "Configuration of VLSI array in the presence of defects," *J. ACM*, vol. 31, no. 4, pp. 694–717, Oct. 1984.

[3] Y. Y. Chen, S. J. Upadhyaya, and C. H. Cheng, "A comprehensive reconfiguration scheme for fault-tolerant VLSI/WSI array processors," *IEEE Trans. Comput.*, vol. 46, no. 12, pp. 1363–1371, Dec. 1997.

[4] R. Negrini, M. G. Sami, and R. Stefanelli, *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays*. Cambridge, MA: MIT Press, 1989, pp. 151–171.

[5] S. Y. Kuo and I. Y. Chen, "Efficient reconfiguration algorithms for degradable VLSI/WSI arrays," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Sys.*, vol. 11, no. 10, pp. 1289–1300, Oct. 1992.

[6] C. P. Low and H. W. Leong, "On the reconfiguration of degradable VLSI/WSI arrays," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Sys.*, vol. 16, no. 10, pp. 1213–1221, Oct. 1997.

[7] C. P. Low, "An efficient reconfiguration algorithm for degradable VLSI/WSI arrays," *IEEE Trans. Comput.*, vol. 49, no. 6, pp. 553–559, Jun. 2000.

[8] W. Jigang and T. Srikanthan, "Partital rerouting algorithm for reconfigurable VLSI arrays," in *Proc. IEEE Int. Symp. Circuits and Systems*, 2003, pp. 641–644.

[9] M. Fukushi and S. Horiguchi, "Self-reconfigurable mesh array system on FPGA," in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2000, pp. 240–248.

Design and Analysis of Compact Dictionaries for Diagnosis in Scan-BIST

Chunsheng Liu and Krishnendu Chakrabarty

***Abstract*—We present a new technique for generating compact dictionaries for cause–effect diagnosis in scan-BIST. This approach relies on the use of three compact dictionaries and target both modeled and unmodeled faults. We present analytical results that provide useful guidelines for the design of these compact dictionaries. We also present experimental results for the larger ISCAS-89 benchmark circuits for the diagnosis of various types of unmodeled faults.**

***Index Terms*—Compaction, diagnostic resolution, fault diagnosis, interval-based dictionary, linear feedback shift register (LFSR).**

I. INTRODUCTION

An advantage of cause–effect fault diagnosis based on fault dictionaries is that it alleviates the need for repeated fault simulation [5], [13]. In contrast, effect–cause diagnosis requires repeated fault simulation runs [12], [14], [17], [18], [20]. However, as designs grow in complexity, dictionary-based diagnosis becomes infeasible due to prohibitively large dictionary sizes. Dictionaries for realistic circuits tend to be too large to fit in the limited memory of testers.

A number of techniques have recently been proposed for reducing dictionary size through compaction [1], [3]–[5], [7], [10], [13], [15]. These techniques attempt to identify and eliminate redundant information in the dictionary. A problem for all these methods is that reduced dictionary size can lead to a larger candidate set, i.e., diagnostic resolution is adversely affected.

Manuscript received November 14, 2004; revised April 12, 2005. The work of C. Liu was supported in part by a faculty research fellowship and a Layman award. The work of K. Chakrabarty was supported in part by the National Science Foundation under Grants CCR-9875324 and CCR-0204077. A preliminary version of this paper was published in the *Proceedings of the 2004 International Symposium on VLSI*.

C. Liu is with the Department of Computer and Electronics Engineering, University of Nebraska-Lincoln, Omaha, NE 68182-0572 USA (e-mail: chunshengliu@ unlnotes.unl.edu).

K. Chakrabarty is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA.

Digital Object Identifier 10.1109/TVLSI.2005.853624