Regular Paper

Scan Tree Design: Test Compression with Test Vector Modification

Kohei Miyase† and Seiji Kajihara†

This paper presents a method to reduce test data volume and test application time for a full-scan circuit. The proposed method constructs a scan tree in which scan flip-flops are placed and routed in a tree structure. Although one scan input to the scan tree drives several scan chains with varying length, it is guaranteed that every test vector of a test set can be loaded into the scan tree. Since the height of the scan tree decides test data volume of the test set, the method modifies the test set so as to minimize the height. The procedure of test vector modification consists of don't care identification for the test set and a solution to a vertex coloring problem for an incompatibility graph constructed from the test set including don't cares. Experimental results for ISCAS-89 benchmark circuits show that the proposed method could reduce, on the average, test data volume and test application time by 70%.

1. Introduction

Reduction of test data volume and test application time is important for saving the test cost of SoC designs. Many methods to reduce test data volume and/or test application time have been proposed in Refs. 1)–12), 15)–17). As a typical approach, research on compact test generation has been done for a long time and significant results have been obtained [1),2)]. However, the number of test vectors increases with the size of circuits. Besides, even if the number of test vectors could be reduced, test application time for a circuit including many scan flip-flops would be large. Thus a lot of research on test compression, which incorporates DFT techniques, has been done [3)~10)].

Multiple scan chain design is known as a practical method to reduce test application time. However, test data volume is unchanged, on the contrary, additional scan inputs and outputs are required. In order to reduce test data volume for multiple scan designs, methods to encode test data have been proposed in Refs. 3)–7). The encoding methods aim at reducing the number of input pins required to transfer test data from a tester to the chip. The encoded test data is changed to the original test data applied to the circuit-under-test through an on-chip decompressor. For some of these methods [6),7)], the hardware overhead of the decompressor circuit would increase with increased compression. Other test compression methods, which devise how to place and route scan flip-flops, have been proposed in Refs. 8)–11). Since one scan input drives more than one scan chain, decompressors required in Refs. 3)–7) are not necessary. However, allowed values in scan flip-flops in different scan chains are correlated because the scan flip-flops on different scan chains receive values from the same scan input. This may preclude detection of some detectable faults. To alleviate this problem, Illinois scan [8)] has single scan chain mode in addition to the multiple scan chain mode. A further enhancement used in Ref. 15) is reconfiguring scan chains for a different parallel mode.

In this paper, we propose a method of test compression for a full-scan circuit. Based on a test set generated for stuck-at faults, the proposed method constructs a scan tree in which scan flip-flops are placed and routed on scan chains with a tree structure. The root of the tree is a scan input and the leaves of the tree are the scan outputs. Although one scan input to the scan tree drives several scan chains with varying length, it is guaranteed that every test vector of a test set can be loaded into the scan tree. Since the height of the scan tree decides test data volume of the test set and test application time, the method modifies the test set so as to minimize the height. The procedure of test vector modification consists of don't care identification for the test set [13)] and a solution to a vertex coloring problem for an incompatibility graph constructed from the test set including don't cares identified ☆. Note that, in this work, we focus on reducing scan-in data, but not scan-out data. Scan-out data can be compressed using MISRs. Experimental results for ISCAS-89 benchmark circuits show that the proposed method could reduce test data vol-

† Kyushu Institute of Technology

ume and test application time on average by 70%. While a scan tree requires a single scan input, it requires more than one scan output. It requires long shift registers in a space compactor or a MISR to compress the output response. In this paper we also propose how to construct a scan tree where the number of outputs is limited. Experimental results with the limitation of the scan outputs show that the method could reduce test data volume and test application time on average by approximately 68%. At another aspect of the scan tree, we assume that scan flip-flops can be placed and routed as we want. It is known that requiring specific ordering of scan chains to realize scan tree may be costly and may impact the functional timing of the circuit [14),15)]. For such situations we investigated a modified method that imposes some constraints on the manner the scan tree is designed. Experimental results show that the constrained scan tree could still achieve effective reduction of test data volume and test application time.

This paper is organized as follows. In Section 2, techniques used in the proposed method are presented. In Section 3, the proposed method is explained. In Section 4, we propose how to limit the number of scan outputs. In Section 5, we give some experimental results for benchmark circuits, and give conclusions in Section 6.

2. Preliminaries

2.1 Scan Tree

Figure 1 illustrates a single scan chain in which scan flip-flops are connected serially. The amount of test data for the scan chain is computed as the number of scan flip-flops times the number of test vectors. Hence in the case of Fig. 1 with the four test shown, the test data volume is 36 $(= 9 * 4)$ bits. We can observe that there exists scan flip-flops that receive the same logic values as some other scan flip-flops for every test vector. For example, scan flip-flops ff_3, ff_4 and ff_6 receive the same logic values for every test vector. Such scan flip-flops are called "compatible" [16)]. There are five groups of compatible scan flip-flops in Fig. 1, $\{ff_3, ff_4, ff_6\}$, $\{ff_2, ff_7\}$, $\{ff_1, ff_9\}$, $\{ff_8\}$, and $\{ff_5\}$. As the compatible scan flip-flops can be driven by one scan input, the scan chain can be reconstructed into a scan tree given in **Fig. 2**. The root of the tree is the scan input and the leaves of the tree are the scan outputs. The concept of scan tree was introduced in Refs. 9), 10), 12) as a method of test compression for a full-scan circuit. The scan tree in Fig. 2 reduces the length of scan chain from 9 to 5, then the test data volume is reduced to 20 $(= 5 * 4)$ bits. Similarly, the test application time for each test vector is reduced from 9 clock cycles to 5 clock cycles.

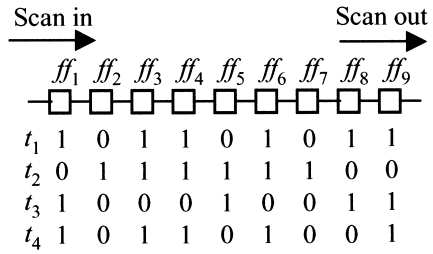


Fig. 1 Single scan chain.

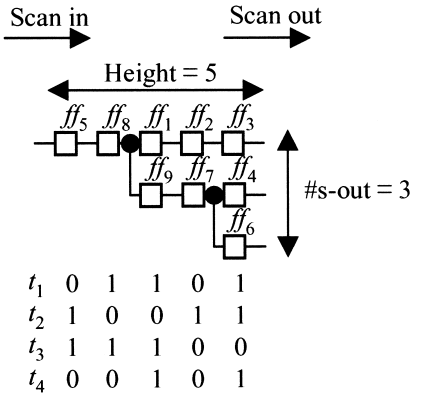


Fig. 2 Scan tree.

2.2 Don't-Care Identification

Test vectors of a test set generated do not have unspecified values generally because they are specified by random fill or static/dynamic test compaction [17)] for the detection of faults other than the target fault. However, some primary input values may be changed to opposite logic values without losing fault coverage. One can regard such input values as don't cares (Xs). A method for identifying Xs of a generated test set without losing fault coverage has been proposed in Ref. 13). As the method utilizes fault simulation and procedures similar to implication and justification of ATPG algorithms, the computational time is practical enough.

Figure 3 shows the basic procedure in

☆ This part has been published as a technical note in Miyase, K., Kajihara, S. and Reddy, S. M.: A Method of Static Test Compaction Based on Don't Care Identification, IPSJ Journal, Vol.43, No.5, pp.1290–1293 (2002).

```
Basic procedure X-search(C,T)
    Circuit C; Test set T;
{
  for each test pattern t_i in T {
      F=collect_essential_fault(t_i);        /* Step 1 */
      t_i'=find_value(F);                    /* Step 2 */
      fault_simulation(t_i');                /* Step 3 */
  }
  for each test pattern t_i in T {
      G=collect_undetected_fault(t_i);       /* Step 4 */
      t_i' +=find_value(G);                  /* Step 5 */
      fault_simulation(t_i');                /* Step 6 */
  }
  return T' composed of t_i';
}
```

Fig. 3 Basic procedure for identifying X inputs.

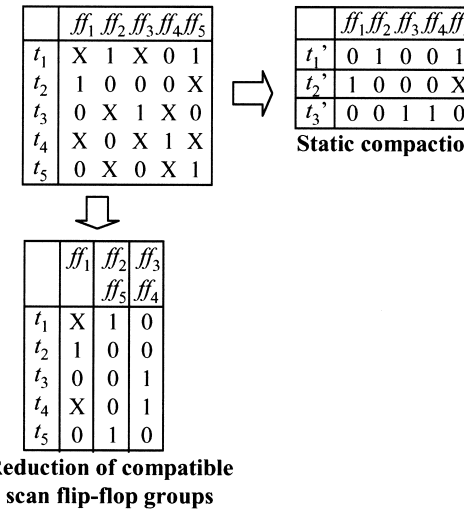


Fig. 4 Test compaction and test compression.

Ref. 13). The method has two phases in terms of which faults to be targeted. At the first phase, it collects essential faults detected by each test vector. Note that an essential fault of a test vector is the fault detected by the test vector but not detected by any other test vectors in the test set. Then it finds logic values to guarantee that each test vector could detect the essential faults. For the logic values found, fault simulation is performed to drop faults. Then the method additionally specifies logic values to detect faults undetected. Finally unspecified values can be treated as *Xs*. In Ref. 13) it was reported that the approximately 50% of the input values are *Xs* even in compacted test sets for ISCAS benchmark circuits [2]. After a test set including *Xs* is obtained, arbitrary logic values can be assigned to the *Xs*.

In this paper, we employ the procedure of don't care identification in Ref. 13). To the *Xs* identified, we will assign logic values so as to minimize the height of a scan tree, to reduce test data volume and test application time.

3. Optimal Construction of A Scan Tree

In this section we present the proposed method to determine an optimal scan tree for a given design with a single scan chain and a given test set. The method has three steps. In the first step maximal number of specified values in the test set are changed to *Xs* without reducing fault coverage. For this step we use the procedure in Ref. 13). Next we find a group of compatible flip-flops with minimum size, for the test set obtained in the first step, by solving an appropriate vertex coloring problem. Next we try to reduce the number of the groups of compatible scan flip-flops by converting some of the specified values in the modified test set into the opposite values. This will help further reduce the number of groups of compatible scan flip-flops and hence the height of the scan tree. Finally we also iterate the last step to further reduce the height of the scan tree.

3.1 Overview of The Proposed Method

When a test set is generated, the effects of test compression with a scan tree depend on the number of groups of compatible scan flip-flops. The smaller the number of groups of compatible scan flip-flops is, the lower the height of the scan tree is and the test data volume and test application time are reduced proportionately. Compatible scan flip-flops are determined from a given test set. If all values in the test set have been specified to either 0 or 1, there may not be many compatible scan flip-flops. In the proposed method, we first identify *Xs* in the given test set, and then we reassign appropriate logic values to the *Xs* so as to increase the number of compatible scan flip-flop groups.

The procedure of reassigning logic values to *Xs* is similar to static compaction that is known as one of the test compaction techniques [17]. As shown in **Fig. 4**, static compaction aims at decreasing the number of test vectors by merging compatible tests. On the other hand, the procedure considered here aims at decreasing the number of groups of compatible scan flip-flops. We employ this procedure with some modifications. In this procedure, reassignment of values to *Xs* is done so that all scan flip-flops in

Table 1 Test set with *Xs*.

	ff_1	ff_2	ff_3	ff_4	ff_5
t_1	1	1	X	0	1
t_2	1	0	0	0	X
t_3	0	X	1	X	0
t_4	1	0	X	1	X
t_5	0	X	0	X	1

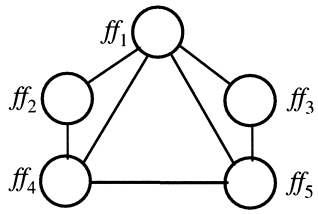


Fig. 5 An incompatibility graph.

a group of compatible scan flip-flops have the same value. For this purpose an incompatibility graph is constructed and the problem of optimal reassignment of *Xs* is reduced to a vertex coloring problem for the graph. In the following subsections, we describe how to construct the incompatibility graph and discuss the vertex coloring problem.

3.2 Incompatibility Graph

An incompatibility graph is a graph that represents relations of values applied to scan flip-flops for every test vector. A vertex of the graph represents a scan flip-flop. An edge between two vertices exists if and only if two scan flip-flops corresponding to the vertices are incompatible. **Table 1** shows some partially specified test vectors and **Fig. 5** shows the incompatibility graph for those scan flip-flops. Scan flip-flop ff_1 and ff_2 are incompatible because the second bits are different. Hence there is an edge between the vertices corresponding to ff_1 and ff_2. On the other hand, scan flip-flops ff_2 and ff_3 are compatible. Hence there is no edge between vertices corresponding to ff_2 and ff_3.

3.3 Vertex Coloring Problem

For the incompatibility graph, we solve a vertex coloring problem, that is, we assign a color to each vertex of the graph such that any pair of adjacent vertices have different colors. As a color corresponds to a group of compatible scan flip-flops, the number of colors to be used must be minimized. From a solution of the coloring problem, we can find that scan flip-flop ff_i represented by vertex v_i can be compatible to other scan flip-flops whose vertex color is the same as of v_i. For example in **Fig. 6**, color c_1 is assigned to ff_1, color c_2 is assigned ff_2 and ff_5, and color c_3 is assigned ff_3 and ff_4. There are some assignments of colors in general. While the vertex coloring problem is to find an assignment of the minimum number of colors, it is very time-consuming to find the optimum solution of the problem. In the proposed method, we solve the problem with a greedy algorithm, i.e., we assign colors to each vertex in order of edges incident on it.

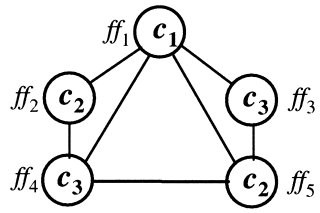
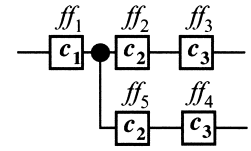


Fig. 6 A colored incompatibility graph.



t_1	1	1	0
t_2	1	0	0
t_3	0	0	1
t_4	1	0	1
t_5	0	1	0

Fig. 7 An obtained scan tree.

The colored incompatibility graph implies a structure of a scan tree. The vertices with the same color give groups of compatible scan flip-flops. The compatible scan flip-flops are placed at the same level in the scan tree. Hence the number of colors used, which is the number of groups of compatible scan flip-flops, denotes the height of the scan tree. For example, since the graph in Fig. 6 has 3 colors, the number of compatible scan flip-flop groups is 3, $\{ff_1\}$, $\{ff_2, ff_5\}$, and $\{ff_3, ff_4\}$. Thus the corresponding scan tree, shown in **Fig. 7**, has a height of 3.

3.4 Iteration of the Proposed Method and Don't-Care Identification on Specific Bits

After we obtain a modified test set by specifying the *Xs* so that all bits corresponding to a group of compatible scan flip-flops are the same, we can again apply the procedure of test modification by regarding the modified test set as a given test set. Because the combination of faults detected by each test vector would be

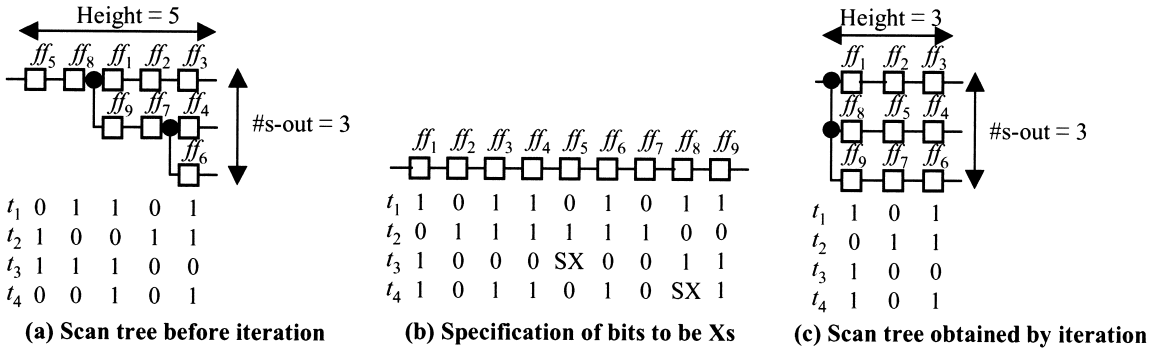


Fig. 8 Iteration of the proposed method.

changed after modifying a test set, *Xs* are identified on different bit positions before and after modifying the test set. Thus the repetition may allow us to compress the test set furthermore.

In the process of iterating the procedure, we try to decrease the number of groups of compatible scan flip-flops. If a scan flip-flop belongs to a large group of compatible scan flip-flops, values of such flip-flops should be left specified. On the other hand, if a scan flip-flop is not compatible with any other scan flip-flop, it is meaningful to change its values to *Xs* such that it will be compatible with some other flip-flop(s). Thus we need to selectively convert specified values to *Xs*. Such a procedure was given in Ref. 18), and we employ it in the process of iterating the procedure to reduce the height of the scan tree by reducing the number of groups of compatible scan flip-flops. Note that fault coverage of the original test set is still guaranteed in the iteration of the procedure because the procedures of Refs. 13) and 18) can identify *Xs* without losing fault coverage.

We show an example using **Fig. 8**. Scan flip-flop ff_5 in Fig. 8 (a) is not compatible with others. However, compared with logic values of ff_2 or ff_7, the difference of logic values of ff_5 is only one bit, namely the third bit. Therefore, if the third bit of ff_5 can be changed to X without loss of fault coverage, the number of groups of compatible scan flip-flops is decreased. Similarly, the fourth bit of ff_8 is required to be an X, to join the compatible scan flip-flop group of to ff_9 and ff_1. Thus, we can specify bits as shown in Fig. 8 (b) where "*SX*" shows a bit that we should attempt to change to X. If both bits discussed above are changed to *Xs*, the height of the scan tree is reduced to 3 as shown in Fig. 8 (c).

4. Limitation of the number of scan outputs

The scan tree often increases the number of scan outputs required. If the number of scan outputs is unlimited, the height of the scan tree is minimized, that is, test data volume and test application time can be minimized. However, the number of scan outputs is generally limited by the number of I/O pins, otherwise it requires long shift registers in a space compactor or a MISR to compress the output response. In order to avoid this problem, we limit the number of scan outputs for each scan tree to a predetermined number. The overhead of the space compactor or the MISR is reduced with a trade-off to the increase of test data volume and test application time.

Because the number of scan outputs of a scan tree is determined with the number of compatible flip-flops in the maximum group, we need to limit the number of flip-flops of each group. As a result, the height of scan tree is increased, that is, test data volume and test application time increase. When the number of scan flip-flops in a group is more than the limited number of scan outputs, we remove some scan flip-flops in the group and build a new group for the removed flip-flops. An example is given in **Fig. 9**. Assuming that the limited number of scan outputs is 2, scan flip-flops ff_9 and ff_{10} are moved to the leaves of the original scan tree.

5. Experimental Results

We implemented the proposed method in C language on a PC (Dual Athlon MP 2000+, 512 MB), and applied it to ISCAS'89 benchmark circuits. The test sets used in this experiment were generated by a test generator that includes test compaction techniques [2]. **Table 2**

Table 2 Experimental results for compacted test sets.

	#tv	#PI	#FF	height	#bits		ratio(%)	Total #bits	#s-out	#it	time(sec)
					single scan	scan tree					
s5378	100	35	179	94	17900	9400	47.5	12900	4	3	3.8
s9234	111	19	228	115	25308	12765	49.6	14874	7	3	9.32
s13207	235	31	669	101	157215	23735	84.9	31020	49	2	19.82
s15850	97	14	597	196	57909	19012	67.2	20370	17	5	30.13
s35932	12	35	1728	21	20736	252	98.8	672	262	4	32.56
s38417	87	28	1636	546	142332	47502	66.6	49938	52	5	100.4
s38584	114	12	1452	359	165528	40926	75.3	42294	13	5	113.28
Average							70.0				

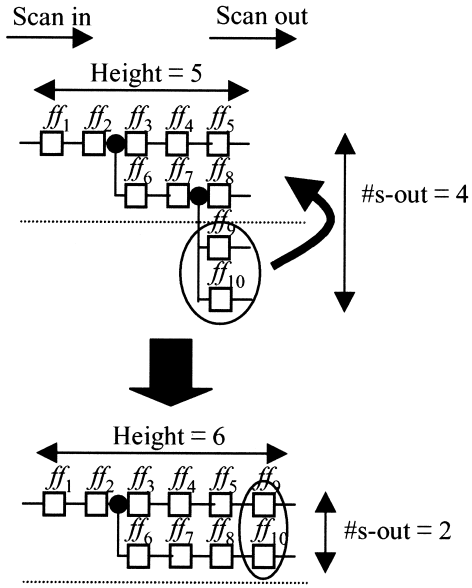


Fig. 9 Limitation of # scan outputs.

shows the results. The first four columns give the circuit name, the number of test vectors, the number of primary inputs and the number of scan flip-flops (pseudo primary inputs). The next column "height" shows the height of scan trees obtained by the proposed method. The next column "#bits" gives the test data volume in number of bits for the case of a single scan chain under "single scan" and for a single scan tree under "scan tree". We can compute the number of bits required for the scan tree environment as the number of test vectors times the height of the scan tree. The next column "ratio(%)" gives the compression percentage of test data volume calculated by the following expression:

$$Ratio(\%) = (|T_{org}| - |T_{comp}|)/|T_{org}| * 100$$

where $|T_{org}|$ is the total number of bits in the original test data under single scan environment and $|T_{comp}|$ is the total number of bits in the compressed test data under scan tree environment. In this paper, although we focus on the compression for scan-in data volume, the column "Total #bits" gives test data volume for both primary inputs and scan inputs (pseudo primary inputs), that is, the test data volume stored in a tester. The test data volume calculated by the following expression:

$$Total\#bits = (\#PI + height) * \#tv$$

The column "#s-out" gives the number of scan outputs of the scan tree. The columns "#it" shows the number of iterations when we iterate the proposed method as described in Section 3.4. The last column shows CPU time in seconds.

As shown in Table 2, the proposed method could reduce approximately 70% of test data volume and test application time on average. For circuit s35932, the proposed method could achieve 98% of reduction. It can be seen that the more reduction in test data volume the proposed method achieves, the more number of scan outputs the scan tree requires. In **Table 3**, we show experimental results for un-compacted test sets. It can be seen that the percentage reduction of test data volume for the un-compacted test sets is higher than that for the compacted test sets. However the test data volume for the un-compacted test sets is higher than that for the compacted test sets for every circuit in terms of the number of bits. In order to measure the effectiveness of iteration of the procedure, we show results without iteration of the proposed method in **Table 4**. The columns "wo-it" and "w-it" show the results without iteration and with iteration, respectively. The proposed method with iteration could achieve higher compression than the method without iteration.

In **Table 5**, we show the comparison of the proposed method with some previous works in terms of the number of bits required. For fair

Table 3 Experimental results for un-compacted test sets.

	#tv	#PI	#FF	height	#bits		ratio(%)	Total #bits	#s-out	#it	time(sec)
					single scan	scan tree					
s5378	333	35	179	64	59607	21312	64.2	32967	11	4	15.03
s9234	480	19	228	76	109440	36480	66.7	45600	12	2	22.65
s13207	586	31	669	64	392034	37504	90.4	55670	46	2	43.5
s15850	500	14	597	107	298500	53500	82.1	60500	29	8	175.59
s35932	76	35	1728	148	131328	11248	91.4	13908	44	12	184.31
s38417	1243	28	1636	186	2033548	231198	88.6	266002	61	3	477.32
s38584	854	12	1452	281	1240008	239974	80.6	250222	18	4	472.12
Average							80.6				

Table 4 Experimental results for compacted test sets with iteration.

	#FF	height		ratio(%)		#it	time(sec)	
		wo-it	w-it	wo-it	w-it		wo-it	w-it
s5378	179	100	94	44.1	47.5	3	1.33	3.80
s9234	228	116	115	49.1	49.6	3	3.29	9.32
s13207	669	101	101	84.9	84.9	2	10.30	19.82
s15850	597	226	196	62.1	67.2	5	6.40	30.13
s35932	1728	23	21	98.7	98.8	4	8.12	32.56
s38417	1636	601	546	63.3	66.6	5	19.59	100.40
s38584	1452	386	359	73.4	75.3	5	23.68	113.28
Average				68.0	70.0			

Table 5 Comparison with previous works.

	FTCS'99 8)	DAC'01 6)	VTS'02 7)	Proposed
s13207	28808	**25344**	56635	31020
s15850	29328	22784	23474	**20370**
s35932	2288	7218	10788	**672**
s38417	54336	89856	65163	**49938**
s38584	**33136**	38796	63612	42294

Table 6 Experimental results with limited numbers of scan outputs.

	#tv	#PI	#FF	height	#bits		ratio(%)	Total #bits	#s-out	#it	time(sec)
					single scan	scan tree					
s5378	100	35	179	94	17900	9400	47.5	12900	8	3	3.82
				94	17900	9400	47.5	12900	32	3	3.86
s9234	111	19	228	115	25308	12765	49.6	14874	8	3	9.27
				115	25308	12765	49.6	14874	32	3	9.31
s13207	235	31	669	140	157215	32900	79.1	40185	8	2	19.89
				104	157215	24440	84.5	31725	32	2	19.91
s15850	97	14	597	207	57909	20079	65.3	21437	8	5	30.05
				196	57909	19012	67.2	20370	32	5	30.17
s35932	12	35	1728	222	20736	2664	87.2	3084	8	4	32.68
				64	20736	768	96.3	1188	32	4	32.55
s38417	87	28	1636	595	142332	51765	63.6	54201	8	5	100.53
				548	142332	47676	66.5	50112	32	5	100.31
s38584	114	12	1452	379	165528	43206	73.9	44574	8	5	113.24
				359	165528	40926	75.3	42294	32	5	113.35
Average							68.1				

comparison we set the number of scan chains to 16 in the methods of Ref. 7) according to the method of Ref. 8). The values in Table 5 show the number of bits for primary inputs and scan inputs (pseudo primary inputs). The best result for each circuit is indicated in bold let-

Table 7 Experimental results with the constraint for compacted test sets.

	#FF	height	#bits of s-tree	ratio(%)	#s-out	#it	time(sec)
s5378	179	151	15100	15.6	4	4	4.67
s9234	228	203	22533	11.0	3	4	11.57
s13207	669	472	110920	29.4	16	9	77.18
s15850	597	394	38218	34.0	17	9	46.40
s35932	1728	181	2172	89.5	32	3	12.41
s38417	1636	1071	93177	34.5	19	12	155.00
s38584	1452	1296	147744	10.7	8	6	99.47
Average				32.1			

Table 8 Experimental results with the constraint and 8 scan outputs.

	#FF	height	#bits of s-tree	ratio(%)	#s-out	#it	time(sec)
s5378	179	151	15100	15.6	8	4	4.64
s9234	228	203	22533	11.0	8	4	11.59
s13207	669	477	112095	28.7	8	8	68.65
s15850	597	403	39091	32.5	8	7	36.26
s35932	1728	301	3612	82.6	8	3	12.39
s38417	1636	1087	94569	33.6	8	10	129.56
s38584	1452	1296	147744	10.7	8	6	99.45
Average				30.7			

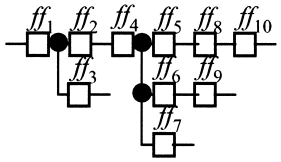


Fig. 10 Constrained scan tree.

ters. For three circuits, the proposed method obtained the best results. While the methods of Refs. 6) and 7) need decompressors circuit, the proposed method doesn't need a decompressor.

In **Table 6**, we show the experimental results when we limit the number of scan outputs to 8, 32. The height of scan tree was increased from the unlimited case but we could achieve effective compression when the number of scan outputs is 32.

The proposed method has some undesirable features in design as follows:

(1) Routing overhead due to reordering scan flip-flops would increase.

(2) Circuit delay would increase.

For the case for which arbitrary scan trees may not feasible due to layout constraints, we made additional experiments. Though the place and route information depends on the circuit, we assume that the order of scan flip-flops in the scan chain is fixed and a scan flip-flop is allowed to be compatible with adjacent flip-flops only. An example of a scan tree with this constraint is shown in **Fig. 10**. **Table 7** shows results of the scan trees with this constraint. The percentage reduction in test data volume given in Table 7 is smaller than the one in Table 2 for every circuit. However, the constrained scan trees still give effective results. For circuit s35932, the constrained scan tree could achieve 89% compression compared to 98% for the unconstrained case. The number of scan outputs required for the scan tree is at most 32 in Table 7. In **Table 8**, we limit the number of scan output as we keep the layout constraints. We set the number to 8. The scan tree still achieves 30% compression.

6. Conclusions

In this paper we proposed a method of test compression for full-scan circuits. The proposed method constructed a scan tree based on a given test set. Since the height of the scan tree decides test data volume of the test set, the method modified the test set so as to minimize the height of the scan tree. In the procedure of test vector modification, an incompatibility graph was constructed and the problem of value assignment was reduced to a vertex coloring problem. To improve the effectiveness of test compression, the proposed method was iterated. Experimental results for ISCAS-89 benchmark circuits show that the proposed method could reduce test data volume and test application time by 70% on average. Even in cases that we limited the number of scan outputs and that we added a hypothetical lay-

out constraint, the proposed method could still achieve effective compression.

References

1) Hamzaoglu, I. and Patel, J.H.: Test Set Compaction Algorithms for Combinational Circuits, *ITC*, pp.283–289 (1998).
2) Kajihara, S., Pomeranz, I., Kinoshita, K. and Reddy, S.M.: Cost-Effective Generation of Minimal Test Sets for Stuck-at Faults in Combinational Logic Circuits, *IEEE Trans. CAD*, Vol.14, No.12, pp.1496–1504 (1995).
3) Koenemann, B., et. al.: A Smart BIST Variant Guaranteed Encoding, *10th Asian Test Symposium*, pp.325–330 (2001).
4) Barnhart, C., Brunkhorst, V., Distler, F., Fransworth, O., Keller, B. and Koenemann, B.: OPMISR: The Foundation for Compressed ATPG Vectors, *ITC*, pp.784–757 (2001).
5) Rajski, J., Tyszer, J., Kassab, M., Mukherjee, N., Thompson, R., Tsai, H., Hertwig, A., Tamarapalli, N., Murgalski, G., Eide, G. and Qian, J.: Emebedded deterministic test for low cost manufacturing test, *ITC*, pp.301–310 (2002).
6) Bayraktaroglu, I. and Orailoglu, A.: Test Volume and Application Time Reduction through Scan Chain Concealment, *DAC*, pp.151–155 (2001).
7) Reddy, S.M., Miyase, K., Kajihara, S. and Pomeranz, I.: On Test Data Volume Reduction for Multiple Scan Chain Designs, *20th IEEE VLSI Test Symposium*, pp.103–108 (2002).
8) Hamzaoglu, I. and Patel, J.H.: Reducing Test Application Time for Full Scan Embedded Cores, *Int'l Symposium on Fault-Tolerant Computing*, pp.260–267 (1999).
9) Chang, S.-C., Lee, K.-J., Wu, Z.-Z. and Jone, W.-B.: Reducing test application time by scan flip-flops sharing, *IEE Proc. Comput. Digit. Tech*, Vol.147, No.1 (2000).
10) Sybille, S., Liang, H.-G. and Wunderlich, H.-J.: A Mixed Mode Bist Scheme Based on Reseeding of Folding Counters, *ITC*, pp.778–784 (2000).
11) Lee, K.-J., Chen, J.-J. and Huang, C.-H.: Using a single input to support multiple scan chains, *ICCAD*, pp.74–78 (1998).
12) Rau, J.C., Jone, W.B., Chang, S.C. and Wu, Y.L.: Tree-structured LFSR synthesis scheme for pseudo-exhaustive testing of VLSI circuits, *IEE Proc. Comput. Digit. Tech*, Vol.147, No.5 (2000).
13) Kajihara, S. and Miyase, K.: On Identifying Don't Care Inputs of Test Patterns for Combinational Circuits, *ICCAD*, pp.364–369 (2001).
14) Makar, S.: A layout-based approach for ordering scan chain flip-flops, *ITC*, pp.341–347 (2000).
15) Pandey, A.R. and Patel, J.H.: Reconfiguration Technique for Reducing Test Time and Test Data Volume in Illinois Scan Architecture Based Designs, *20th IEEE VLSI Test Symposium*, pp.9–15 (2002).
16) Chen, C.-A. and Gupta, S.K.: Efficient BIST TPG Design and Test Compaction via Input Reduction, *IEEE Trans. CAD*, Vol.17, No.8, pp.692–705 (1998).
17) Goel, P. and Rosales, B.C.: Test Generation and Dynamic Compaction of Tests, *Digest of Papers 1979 Test Conf.*, pp.189–192 (1979).
18) Miyase, K., Kajihara, S., Pomeranz, I. and Reddy, S.M.: Don't care identification on specific bits of test patterns, *International Conference on Computer Design*, pp.194–199 (2002).

(Received October 17, 2003)
(Accepted March 5, 2004)

Kohei Miyase received the B.E. and M.E. degrees from Kyushu Institute of Technology, in 2000 and 2002, respectively. Since 2002, he has been a Ph.D. student of Graduate School of Computer Science and Systems Engineering, Kyushu Institute of Technology. His research interests include reduction of SoC testing cost and Design for the Testability. He is a student member of the IEEE and the IEICE.

Seiji Kajihara received the B.S. and M.S. degrees from Hiroshima University, Japan, and the Ph.D. degree from Osaka University, Japan, in 1987, 1989, and 1992, respectively. From 1992 to 1995, he was an Assistant Professor at Osaka University. In 1996, he joined the Department of Computer Science and Electronics of Kyushu Institute of Technology, Japan, where he is a Professor currently. His research interest includes test generation, delay testing, and design for testability. He received the Young Engineer Award from IEICE in 1997 and the Yamashita SIG Research Award from IPSJ in 2002. Dr. Kajihara is a member of the IEEE and the IEICE.