New Static Compaction Techniques of Test Sequences for Sequential Circuits

F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda

Politecnico di Torino
Dipartimento di Automatica e Informatica
Torino, Italy

Abstract[*]

This paper describes an algorithm for compacting the Test Sequences generated by an ATPG tool without reducing the number of faults they detect. The algorithm is based on re-ordering the sequences so that some of them can be shortened and some others eliminated. The problem is NP-complete, and we adopt Genetic Algorithms to obtain optimal solutions with acceptable computational requirements. As it requires just one preliminary Fault Simulation experiment, the approach is much more efficient than others proposed before; experimental results gathered with Test Sets generated by different ATPG tools show that the method is able to reduce the size of the Test Set by a factor varying between 50% and 62%.

1.Introduction

Traditionally, Automatic Test Pattern Generators (ATPGs) for sequential circuits are evaluated according to three parameters: the attained Fault Coverage, the required CPU time, and the number of generated Test Vectors.

When th**e** last parameter is considered, several techniques can be exploited to improve the performance of an ATPG tool: some of them require modifying the tool itself to make it able to generate shorter test sequences, others simply perform a post-processing of the sequences, which can thus be compacted while still guaranteeing the same detection capabilities. The techniques belonging to the former group are known as dynamic compaction techniques [PoRe96b], those belonging to the latter one as static compaction techniques [RNPA88][PoRe96a].

The main advantage of static compaction techniques is that they are independent of the adopted ATPG tool and can thus be implemented as a separate package. On the other side, as the performance of ATPGs often depends on the length of the generated sequences, including a dynamic compaction technique into the ATPG algorithm can sometimes result in a significant reduction in the CPU time required by the whole test process.

In this paper, we show how to reduce the length of available test patterns using a static compaction approach under the assumption that they can be partitioned into independent sequences, each starting either from a state which can be forced by the outside or from the all-Xs state. This situation occurs, for example, when the circuit has an external reset signal, so that any flip flop can be easily forced in the reset state when required. The circuits we will use for experimentally evaluating our method satisfy this condition. Moreover, we assume that the set of test vectors generated by the ATPG only includes 0 and 1 values.

We propose a compaction algorithm based on re-ordering the sequences so that some of them are shortened, and some others eliminated. This requires the solution of an NP-complete optimization problem, and we adopt Genetic Algorithms to obtain a sub-optimum solution with acceptable computational requirements.

Our method is somehow similar to the Vector Selection procedure described in [PoRe96a]; however, it is much more efficient, as it only requires a preliminary fault simulation phase, in which all the sequences are simulated once and for all. Experimental results on the ISCAS'89 circuits are reported, showing

[*] This work has been partially supported by MURST through the 40% project *Affidabilità e Diagnostica in Elettronica*. Contact address: Paolo Prinetto, Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, I-10129 Torino (Italy), e-mail Paolo.Prinetto @polito.it

ED&TC '97 on CD-ROM

the approach efficiency even when sequences generated by different ATPG tools are considered.

The paper is organized as follows: Section 2 introduces some definitions and notations used in the rest of the paper. Section 3 formally defines the problem. Section 4 describes the proposed algorithm, and Section 5 reports some experimental results. Some conclusions are eventually drawn in Section 6.

2.Definitions and Notation

For the purposes of this paper, we will use the notation described in this section, which is partly derived from the one adopted in [PoRe96a].

A *Test Sequence* (or simply a *Sequence*) S_i is an ordered set of L *Test Vectors* (or simply *Vectors*)

$$S_i=(v_{i,1}v_{i,2}...v_{i,L})$$

where $v_{i,j}$ is the Test Vector applied at time unit t_j. When the first Test Vector of every sequence is applied, the circuit is supposed to be in the reset state.

We assume that a set of Test Sequences (i.e., a *Test Set* T) is available, which is able to detect a subset F_T of the single stuck-at faults belonging to the full set F. An order is also assigned to the sequences belonging to T, and each sequence is responsible for detecting some faults not covered by the previous sequences in T.

We will denote by $F_{\leq i}$ the subset of faults detected by the Test Sequence S_i ($F_{\leq i} \subseteq F_T$). We will also denote by F_i the set of faults detected by S_i and not detected by any sequence in T before S_i. Obviously, $F_i \subseteq F_{\leq i}$. Given a sequence S_i, the time unit where a fault $f_j \in F_i$ is detected for the first time is denoted by $tdet_i(f_j)$.

To represent in a compact way the information defined above for a given Test Set, an *n* x *m* integer matrix can be used, where *n* is the number of sequences, and *m* the number of faults. The element of the matrix with coordinates i,j holds the value 0, if the sequence S_i does not detect the fault f_j, or the value $tdet_i(f_j)$ otherwise. We will denote this matrix as *Detection Matrix*.

The *effective test length* $L_{eff}(S_i)$ of a sequence S_i is the minimum length of a subsequence of S_i that starts at time t=1 and extends to the detection time of every fault detected by S_i and not detected by any sequence in T before S_i, or

$$L_{eff}(S_i) = \max \{ tdet_i(f): f \in F_i \}$$

The total length of the Test Set T generated by the ATPG is thus supposed to be the sum of the effective test lengths of the sequences belonging to T when applied in the order specified by the ATPG.

3.Problem Definition

Our compaction algorithm is based on re-ordering the Test Sequences generated by the ATPG. The rationale behind our approach is that a proper re-ordering of the sequences can modify the order in which they detect faults, thus reducing the effective test length of some sequences. In some case, the re-ordering can even cause some sequence to become useless and thus be discarded.

<u>Example</u>

Let us consider a sample case, in which the ATPG tool generated three sequences: S_1 is composed of 3 vectors, and detects fault f_1 at time unit t=1, and fault f_3 at time unit t=3. S_2 is composed of 5 vectors, and detects fault f_1 at time unit t=2, and fault f_2 at time unit t=5. Finally, S_3 is composed of 4 vectors, and detects fault f_2 at time unit t=3, fault f_3 at time unit t=1, and fault f_4 at time unit t=4. This is graphically shown in Fig. 1.a, where a faults appears in bold when it is detected for the first time.

The Detection Matrix for this case is the following

	f_1	f_2	f_3	f_4
S_1	1	0	3	0
S_2	2	5	0	0
S_3	0	3	1	4

The total length of the original Test Set composed of sequences S_1, S_2, and S_3 (in that order) is 12.

If the new order S_3, S_2, S_1 of the sequences is considered, one can see that:

- sequence S_1 is not useful any more, as all the faults it detects are already detected by S_3 and S_2
- the effective length of sequence S_2 is reduced to 2, as f_2 is already detected by S_3.

As a result, the length of the same Test Set in the new order (S_3, S_2, S_1) is 6. This is shown in Fig. 1.b, where useless vectors are filled in gray.

The problem we are addressing can thus be formulated in the following way.

Given

- a set of faults F
- a Test Set T composed of a set of Test Sequences S_1, S_2, …, S_n
- the Detection Matrix for T

find an ordering of the sequences in T, such that the test length (i.e., the sum of the effective test lengths of all the sequences, taken in that order) is minimum.

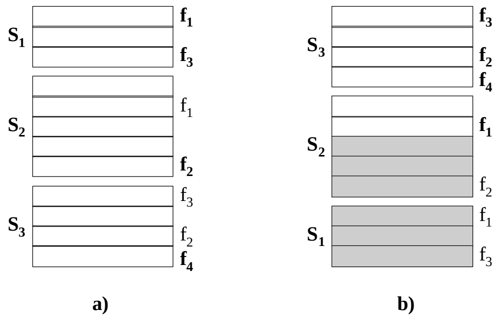

Fig. 1. Faults detected by each vector before and after re-ordering.

It is interesting to note that if the simplifying assumption is made, that in each sequence all the faults detected by the sequence are detected by the last vector, then the problem defined above more simply corresponds to finding the minimum set of sequences detecting all the faults. This correspond to finding a minimum cover for the fault graph, which is a well-known NP-complete problem in the graph theory. Therefore, the test sequence compaction problem as defined in this paper also belongs to the same complexity class.

4.Compaction Algorithm

The input for our algorithm is the Detection Matrix for the available Test Set; this can be easily computed through a proper fault simulation experiment with Fault Dropping enabled just during the simulation of each sequence.

In order to solve the search problem defined in the previous Section, we adopted Genetic Algorithms [Gold89], which are known to be best suited for the optimal solution of large problems.

The Genetic Algorithm we devised can be summarized as shown in the pseudo-code of Fig. 2: a *Population* is a set of *Individuals*, each corresponding to a possible solution of the problem, encoded as a string of integers (known as *genes*). The initial population is randomly generated. A *Fitness Function* is associated to each individual, which is a measure of its goodness as a solution of the search problem. The current population is repeatedly modified through generations by generating new individuals (*sons*) from the existing ones using operators belonging to three categories: cross-over (two *parent* individuals are merged), mutation (one individual is randomly modified), and heuristic operators (the best individual is transformed according to a local optimization procedure). Two alternative *stopping conditions* can be adopted: either the system reached stability (i.e., no more improvements in the population fitness are recorded for a given number of generations), or a maximum predefined number of generations MAX_GENERATION_NUM have been stepped through. For the purpose of the experiments described in Section 5 we always adopted the latter solution.

The characteristics of the algorithm are described in more details in the following Subsections.

4.1.Encoding

We adopted a straightforward encoding: each chromosome is a string composed of *n* genes (*n* being the number of sequences), and directly corresponds to an ordering.

As an example, let us consider again the Test Set composed of three Sequences introduced before. Possible individuals are (1 2 3) and (2 3 1).

4.2.Fitness

For every individual we compute an evaluation function corresponding to the number of vectors eliminated with respect to the original Test Set provided by the ATPG.

We assign to each individual a fitness value which represents the position of the individual in the ranking based on ordering the individuals according to their increasing evaluation function. This mechanism reduces the risk for an individual with evaluation function much higher than the others to prevail in every selection procedure.

4.3.Operators

The *uniform cross-over* is adopted: one half of the genes of the son (randomly chosen) are kept from one parent, the others are kept form the other parents, in the same order in which they appear in it.

```
$P_0$ = create_initial_population();
compute_fitness($P_0$);
i=0;
while (stopping_condition()≠TRUE)
{ $A$ = $P_i$;
  /* generations cycles */
  for j=0 to NEW_INDIVIDUALS
  { /* new element generation */
    $s_1$ = select_an_individual();
    $s_2$ = select_an_individual();
    $s_j$ = cross_over_operator($s_1$, $s_2$);
    if(rand()≤$p_m$)
      $s_j$ = mutation_operator($s_j$);
    $A = A \cup s_j$;
    j=j+1;
  }
  compute_fitness($A$);
  $P_{i+1}$={the POP_SIZE best individuals $\in$
            $A$ };
  if(rand()≤$p_h$)
  { $s_{best}$ = best_individual($P_{i+1}$);
    $s_{best}$ = heuristic_operator($s_{best}$);
  }
  i=i+1;
}
```

Fig. 2. Pseudo-Code of the algorithm.

Example

Let us consider the following two individuals, coming from a sample problem in which 6 sequences have to be ordered:

```
5 3 1 4 2 6
3 6 5 2 4 1
```

One half of the genes of the new individual is taken from the first parent:

```
- 3 1 - 2 -
```

The other genes are taken from the second parent, using the values which have not been already inserted, in the order in which they appear in the second parent:

```
6 3 1 5 2 4
```

Individuals are selected for being parents on the basis of their fitness function: individuals with higher evaluation function have a higher probability of being selected. *Roulette-wheel* has been adopted for this purpose.

The *mutation* operator randomly selects two genes in an individual and swaps them. It is activated with probability p_m on each newly generated individual.

The *heuristic* operator implements a simple local optimization procedure on the best individual.

It considers every couple of adjacent genes in the individual and checks whether swapping them improves the fitness: only in the positive case the individual is changed accordingly. Scanning the couples for possible swaps is repeated until no more improvements are found.

It is activated at each generation with probability p_h on the best individual of the current population.

5.Experimental Results

To implement the algorithm described above we wrote two tools:

- *DMMaker* aims at computing the Detection Matrix: this tool is based on an efficient Fault Simulator developed at our institution and based on the PROOFS algorithm [NCPa92]
- *GACOMP* implements the Genetic Algorithm: its input is simply the Detection Matrix (it does not perform any Fault Simulation) and its output is the optimal sequence ordering it has found (possibly with some sequence missing).

The two tools amount to about 3,000 and 1,000 C code lines, respectively, and for the purpose of the experiments described here have been run on a SUN SPARCstation 5/110.

To evaluate the effectiveness of our approach, we considered the test sequences generated by three ATPG tools:

- GATTO, a state-of-the-art tool developed at our institution and based on also Genetic Algorithms [CPRS96a]
- HITEC, the state-of-the-art tool when deterministic ATPG algorithm are considered [NiPa91]
- Symbat [CCPS93], a prototypical tool we developed, which implements an ATPG algorithm exploiting BDDs and symbolic traversal techniques [CHSo93].

The ISCAS'89 benchmark circuits [BBKo89] and those belonging to the *Addendum* benchmark set [Adde93] have been adopted: as HITEC does not deal with resettable Flip Flops, it has been run on a modified version of the benchmark circuits, in which a primary input has been added, and a multiplexer has been inserted on the data input of each flip flop; by driving the new input, it is possible to force all the flip flops to hold the value 0.

The results of our experiments for the three ATPGs are reported in Table 1, 2, and 3, respectively. Columns 2 and 3 give the number of sequences and vectors composing the Test Sets generated by the three ATPG tools, respectively. DMMaker was run on them, and built the Detection Matrix. GACOMP has been finally run: the characteristics of the compacted test sets it produced (in terms of number of sequences and vectors), together with the CPU time it required, are reported in columns 4, and 5. In column 6 we show the achieved percent reduction in terms of number of vectors. The CPU times required by DMMaker and GACOMP have been reported in column 7. When running GACOMP the parameters appearing in Fig. 2 have been set to the following values: POP_SIZE=50, NEW_IN-DIVIDUALS=30, MAX_GENERATION_NUM=100, p_m=0.2, p_h=0.1.

Several observations can be made concerning the data in Table 1, 2, and 3:

- Symbat is not able to generate a Test Set for all the considered circuits, due to the limitation in the underlying algorithm; we reported the compaction results for all the circuits it can deal with;
- for some circuits (e.g., s9234 for GATTO, or s499 for HITEC) the ATPG tools generated a single sequence: in this case we were obviously not able to apply our approach.

The experiments demonstrate that our algorithm is able to significantly compact Test Sets no matter the ATPG they have been generated by and with acceptable CPU time requirements. In particular, it is worth noting that:

- the average reduction in the Test Sets we obtained is about 50% for GATTO, 57% for HITEC, and 62% for Symbat. This can be explained by observing that both HITEC and GATTO exploit some sort of fault ordering strategy, and this obviously affects the order and size of the generated sequences, too. In the case of the former tool the ordering is based on a preliminary testability analysis on faults, in that of the latter tool this ordering operation is automatically done by phase 1 when selecting target faults for phase 2 (see [CPRS96a] for more details on the GATTO algorithm).
- the CPU time requirements are always much lower that the ones required by the three ATPGs to generate the Test Sets; detailed information about the parameters we used to run them and about their requirements can be found in [CPRS96b]
- comparison with other compaction tools can hardly be done, as they normally do not assume the availability of a reset signal.

6.Conclusions

This paper introduces a new approach to the static compaction of Test Sets generated by ATPG tools. By re-ordering the Test Sequences it is possible to eliminate some and to reduce the length of many of them without any reduction in the Fault Coverage they attain. As finding the optimum reordering is an NP-complete problem, we adopted Genetic Algorithms and devised a solution able to produce optimal results with acceptable computational requirements. According to our experiments, our algorithm is able to reduce the total number of vectors by a factor varying from 50% to 62%, depending on the ATPG tool the starting Test Sets were generated by.

We are now currently investigating other static compaction techniques which can successfully complement the one presented in this paper and thus allow an even larger reduction in the number of Test Vectors.

References

[Adde93] These benchmark circuits are downloadable at the address `http://www.cbl.ncsu.edu/www/CBL_Docs/Bench.html`

[BBKo89] F. Brglez, D. Bryant, K. Kozminski, “Combinational profiles of sequential benchmark circuits,” Proc. *Int. Symp. on Circuits And Systems*, 1989, pp. 1929-1934

[CCPS93] G. Cabodi, F. Corno, P. Prinetto, M. Sonza Reorda, “Symbat’s User Guide,” Politecnico di Torino, Internal Report No. IR-DAI/CAD/ATSEC#3/93, Sept. 93

[CHSo93] H. Cho, G.D. Hatchel, F. Somenzi, “Redundancy Identification/Removal and Test Generation for Sequential Circuits Using Implicit State Enumeration,” *IEEE Trans. on CAD/ICAS*, Vol. CAD-12, No. 7, pp. 935-945, July 1993

[CPRS96a] F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, “A Genetic Algorithm for Automatic Test Pattern Generation for Large Synchronous Sequential Circuits,” *IEEE Trans. on CAD/ICAS*, Vol. CAD-15, No. 8, August 1996

[CPRS96b] F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda: “Comparing topological, symbolic and GA-based ATPGs: an experimental approach,” Proc. *IEEE Int. Test Conf.*, October 1996

[Gold89] D.E. Goldberg, “Genetic Algorithms in Search, Optimization, and Machine Learning,” Addison-Wesley, 1989

[NCPa92] T.M. Niermann, W.-T. Cheng, J.H. Patel, “PROOFS: A Fast, Memory-Efficient Sequential Circuit Fault Simulator,” *IEEE Trans. on CAD/ICAS*, Vol. 11, No. 2, pp. 198-207, February 1992

[NiPa91] T. Niermann, J.H. Patel, “HITEC: A Test Generator Package for Sequential Circuits,” Proc. *European Design Automation Conf.*, 1991, pp. 214-218

[PoRe96a] I. Pomerantz, S.M. Reddy, “On Static Compaction of Test Sequences for Synchronous Sequential Circuits,” Proc. *ACM Design Automation Conf.*, 1996

[PoRe96b] I. Pomerantz, S.M. Reddy, “Dynamic Test Compaction for Synchronous Sequential Circuits using Static Compaction Techniques,” Proc. *IEEE Fault Tolerant Computing Symp.*, 1996, pp. 53-61

[RNPA88] R.K. Roy, T.M. Niermann, J.H. Patel, J.A. Abraham, R.A. Saleh, “Compaction of ATPG-Generated Test Sequences for Sequential Circuits,” Proc. *IEEE Conf. on Comp. Aided Design*, 1988, pp. 382-385

Circuit	Original Test Set		Compacted Test Set			CPU time
	#seq.	#vectors	#seq.	#vectors	% red.	[s]
s208	36	1,096	6	749	31.66	1/8
s298	24	302	11	161	46.69	1/8
s344	19	141	10	75	46.81	1/7
s349	19	144	11	60	58.33	1/7
s382	17	840	7	355	57.74	3/8
s386	38	418	15	197	52.87	2/17
s400	16	916	7	414	54.80	3/9
s420	33	797	8	464	41.78	4/21
s444	22	1,434	9	646	54.95	5/12
s499	29	465	9	273	41.29	2/19
s510	37	989	7	752	23.96	3/18
s526	18	1,050	9	281	73.24	5/14
s526n	16	862	6	339	60.67	4/12
s641	48	395	24	174	55.95	3/25
s713	55	557	23	307	44.88	4/10
s820	38	669	14	322	51.87	5/14
s832	33	425	10	229	46.12	4/12
s838	37	1,323	11	850	35.75	13/16
s938	37	1,323	11	850	35.75	13/16
s953	75	1,099	32	560	49.04	12/20
s967	72	1,223	31	554	54.70	13/19
s991	20	448	9	83	81.47	3/5
s1196	133	1,805	73	674	62.66	25/43
s1238	123	1,554	72	547	64.80	25/50
s1269	52	450	29	205	54.44	12/17
s1423	107	2,691	28	1,407	47.71	68/46
s1488	65	1,824	19	878	51.86	21/24
s1494	62	1,244	19	592	52.41	18/25
s1512	52	772	14	483	37.44	15/27
s3271	132	2,529	50	997	60.58	262/111
s3330	108	2,028	44	961	52.61	89/110
s3384	58	888	22	478	46.17	70/60
s4863	112	1,533	42	787	48.66	163/154
s5378	71	919	41	426	53.65	78/139
s6669	64	592	36	289	51.18	170/114
s13207	34	544	9	357	34.38	287/207
s15850	10	153	3	62	59.48	111/96
s35932	59	903	8	595	34.11	3,239/706
s38417	95	1,617	31	920	43.10	5,919/1,601
s38584	271	8,065	108	4,716	41.53	35,025/4,918

Tab. 1: results for the GATTO Test Set.

Circuit	Original Test Set		Compacted Test Set			CPU time
	#seq.	#vectors	#seq.	#vectors	% red.	[s]
s208	44	741	10	450	39.27	1/3
s298	19	217	7	99	54.38	1/2
s344	10	61	6	15	75.41	1/1
s349	15	84	9	20	76.19	1/2
s382	15	359	2	203	43.45	2/3
s386	58	258	31	96	62.79	2/8
s400	15	357	2	201	43.70	2/4
s420	51	788	10	513	34.90	4/10
s444	17	308	2	103	66.56	2/4
s510	37	847	27	223	73.67	3/6
s526	17	260	2	87	66.54	2/5
s526n	16	256	2	89	65.23	2/4
s641	78	306	36	136	55.56	3/12
s713	74	270	34	99	63.33	4/14
s820	120	1,170	64	498	57.44	13/33
s832	111	1,058	60	439	58.51	12/33
s838	52	675	12	365	45.93	9/24
s938	52	675	12	365	45.93	9/24
s953	111	825	38	421	48.97	14/35
s967	120	831	38	424	48.98	15/39
s991	50	83	25	37	55.42	4/14
s1196	189	509	109	170	66.60	18/69
s1238	191	513	108	181	64.72	20/83
s1269	66	255	25	99	61.18	9/28
s1423	49	283	16	95	66.43	12/33
s1488	24	69	16	13	81.16	3/18
s1494	60	523	43	99	81.07	13/33
s1512	59	283	14	165	41.70	9/35
s3271	61	984	22	495	49.70	122/57
s3330	132	764	85	215	71.86	59/150
s3384	17	212	8	47	77.83	18/22
s4863	105	376	57	119	68.35	74/169
s5378	95	250	49	98	60.80	52/194
s6669	68	466	22	207	55.58	135/117
s9234	6	19	1	9	52.63	13/50
s13207	14	97	6	39	59.79	71/96
s15850	14	39	4	24	38.46	96/126
s35932	376	1,712	13	1,468	14.25	7,530/4,259
s38417	280	806	14	674	16.38	3,581/4,828
s38584	48	509	30	74	85.46	2,375/1,000

Tab. 2: results for the HITEC Test Set.

Circuit	Original Test Set		Compacted Test Set			CPU time
	#seq.	#vectors	#seq.	#vectors	% red.	[s]
s208	64	2,049	34	695	66.08	4/4
s298	34	344	17	151	56.10	1/3
s344	47	187	23	76	59.36	1/4
s349	46	184	24	71	61.41	2/5
s382	59	2,580	25	1,262	51.09	8/8
s386	76	340	43	136	60.00	2/11
s400	59	2,538	26	1,186	53.27	9/8
s420	49	9,377	25	629	93.29	62/12
s444	39	2,034	24	772	62.05	8/6
s499	33	418	23	120	71.29	2/7
s510	59	1,066	41	256	75.98	4/9
s526	74	3,607	31	1,928	46.55	17/14
s526n	73	3,573	31	1,894	46.99	17/14
s635						
s641	160	516	97	176	65.89	7/27
s713	164	538	100	182	66.17	9/36
s820	202	1,425	109	645	54.74	19/54
s832	195	1,370	107	602	56.06	18/55
s838						
s938						
s953	155	1,261	85	500	60.35	21/46
s967	162	1,322	88	527	60.14	22/47
s991						
s1196	297	613	200	237	61.34	26/109
s1238	300	619	206	234	62.20	27/132
s1269						
s1423						
s1488	157	1,709	99	599	64.95	36/64
s1494	160	1,787	100	647	63.79	38/68
s1512						
s3271						
s3330						
s3384						
s4863						
s5378						
s6669						
s9234						
s13207						
s15850						
s35932						
s38417						
s38584						

Tab. 3: results for the Symbat Test Set.