PROR compaction scheme for larger circuits and longer vectors with deterministic ATPG

Suresh k Devanathan

Michael L Bushnell

***Abstract*—*Reverse order restoration ROR techniques have found great use in sequential automatic test pattern generation ATPG, esp. spectral and perturbation-based ATPG. This paper deals with improving ROR for that purpose. We introduce parallel-fault multi-pass 2-level polynomial reverse order restoration PROR algorithms with constant complexity of the form $H(n)G(n) + c$ where $H(n)$ is the number of vectors to be released this iteration and $G(n)$ is the attenuation factor. In PROR $H(n) = n^k$ and G(n) here is $1 - H(n)/no\_vectors$ with c about 1 to 64, where n is the number of iterations, the fault has been hanging around in the compactor and k is the polynomial complexity of the algorithm, in each iteration. k is variant and can take on any real value. We also divide the vector length such that it has a maximum of 10000 units at best, so that if the vector length is greater than 10000, it is still rounded to 10000 by considering a bigger chunk size. We also added PODEM of backtrace limit 30 on the last vector to get a faster and better quality test set. On contrast to algorithms which do not have results reported on large circuits and on longer vector lengths, we showcase our algorithm. For example, on b22, we achieved a 94.36% FC in 1.31 days with VL 146476.*

I. Introduction

To understand the importance of compaction, we discuss problems with sequential ATPG schemes. Sequential ATPG has been traditionally seen as unreliable, slow, providing poor quality test set, etc. This is mostly because sequential ATPG is a hard problem. From days of deterministic test generators such as HITEC [5] and SEST, and till GA-based STRATEGATE, sequential ATPG has generally been seen as bad. It is with the introduction of compaction-based ATPG schemes, such as , till the more recent wavelet-based compaction ATPG [2] [3], it is showing some promise. Although, not all problems, have been alleviated, at least some of the problems have been fixed and a good compactor is crucial to these schemes.

We introduce PROR, a new variant of ROR-type compactors. They are noticeably very fast, and do have a lower quality trade off with vector length. In general, in this paper, we also present tricks that we used to tweak our tool. In the end, we present results.

II. ROR

Out of the compaction algorithms, the most useful ones are RORs because unlike omission based sequential compactors, they take less time and unlike some schemes, they do not use a lot of memory space. The algorithm for a typical ROR is as follows.

- Drop all vectors
- Pick fault with highest detection time
- Restore vectors from that time instant
- If fault is detected, stop, else keep restoring vectors
- Move on to the fault of the next highest detection time
- Repeat the process until necessary

Different algorithms play on this scheme. An LROR, for example, restores only one vector at a time and a *radix reverse order restoration* RROR scheme restores an exponential number of vectors each instant. For example, with radix-2, the scheme would restore 1,2,4,8 vectors, etc. RRORs although fast, are over ambitious, whereas LROR are too slow. The invention of PROR exists to solve these problems.

Unlike fault simulators, which are exact, or have to be consistent with each other, each compactor has its own flavor. Compaction algorithms are inexact. The only exact thing about them is that their results have to be verifiable by a fault simulator and that they, in general, have to produce smaller test sequences, than their initial test sequence.

III. Prior Work

These schemes are also once again either too slow or over-ambitious. LRORs are generally better at providing higher compaction ratios, however, on a larger benchmark circuit, with longer vector counts, they may never complete in time. This has been our observations, esp. with developing compaction for sequential ATPG. RROR and other radix based algorithms do provide a faster compaction rate, however, their compaction ratios are lower and with sequential ATPG, the vector lengths start exploding, after a few iterations of running the algorithms. So, they are not completely practical.

A. Variants

Several variants of RORs have been proposed. Some of them are multi-fault algorithms that take advantage of 32/64-bit machine parallelisms. Most of the speed up from this scheme only provides a constant order of magnitude difference. Our PROR is rather an algorithmic modification. Other algorithms such as Pomeranz et al. [4], do restoration, but also change the order of fault detection between the compacted and initial set. Although for STRATEGATE vector compaction, they may save time, this becomes a problem, for ATPG, as the ATPG, will spend a lot of its time rearranging vectors. So, preserving detection times, may actually save time for compactor schemes used in ATPG.

IV. Theory of PROR

PROR is a constant complexity ROR scheme with polynomial vector length restoration for each iteration. PROR restores vectors based on two expectancies. One, faults that have not been detected yet, but have been, on the compaction cycle need more vectors restored for detection. Two, as restored vector lengths approach the total vector length, lesser vectors have to be restored. They are represented by $H(n)$ and $G(n)$ functions respectively. And c is the default number of vectors to be restored in case $H(n)G(n)$ is 0. In our case, c varies between 1 and 64, depending on the number of maximum faults the compactor is targeting in the current cycle.
Observe that $H(n)/H(n-1) \rightarrow 1$ as $n \rightarrow \infty$. Unlike RRORs, where $H(n)/H(n-1)$ is some number > 1, the percentage of vectors released on each cycle, by PROR, starts staying the same, for every cycle, as $n \rightarrow \infty$. This is similar to the LROR, except that $H(n)/no\_vectors$ for the LROR is 0 for large n, and whereas, it is a non-zero number for PROR.
This has many advantages. One, the algorithm always restores a constant percentage of vectors, for larger n. Secondly, this percentage is significant macroscopically, unlike an LROR scheme. The PROR accounts for the majority of the time-savings by this algorithm.

A. $H(n)$ and $G(n)$

$H(n)$ is given by n^k. Practically, it is implemented as $n^k + 1$. This stops the number from going to 0. $G(n)$ is $1 - H(n)/no\_vectors$. The number of vectors restored for the current fault is $H(n)G(n) + 1$. A multi-fault compactor, like ours, will invoke this calculator for all targets faults in the machines. So, in actuality, for the current iteration, the number of vectors restored is the sum of vectors restored individually.

B. Calculation of k

k, the complexity of our PROR, is dependent on the vector length. Longer vector lengths will have a larger k. This brings down compaction time. The theory is that when a compactor has to perform for a longer vector set, faults detected by that set, take longer restoration sequences before being detected. That is to say, there is a correlation between no. of vectors restored and the initial no. of vectors.

$$k = 1.1\frac{\log(no\_vectors + 1)}{\log 61518} \tag{1}$$

The constants 61518 and 1.1 are empirical values that gave us generally good results for large circuits. Basically, what it means is that for vector lengths about 61518, k is about 1.1 .
We make a modification to the above formula

$$k = 1.1\frac{\min(\log(no\_vectors + 1), 10000)}{\log 61518} \tag{2}$$

such that at worst the algorithm behaves as if only 10000 vectors are present. This is the called the resolution of the algorithm. This is related to the percentage-based complexity of constant time 10000.

V. Speed-up Tips and Tricks

In the next few sections, we will discuss several speed-up tips and tricks. Some of them are modifications of schemes already in use. However we found this configuration to work, better and faster.

VI. Initialization Sequence Restoration

Pomeranz *et. al* [4] have already proposed restoring initialization sequence restoration. Basically, in this process, the first few vectors are restored. This has been known to reduce compaction time. We will discuss specific numbers that have worked for us. From our experience, we found that restoring first 5% of the vectors, on average, does a good job. Secondly, all faults restored by the initialization sequence may be dropped simply because it will always be detected, as we are not throwing away vectors that detect them.
This has another implication. Do not restore a fault (further) when its detection time falls, within the contiguous block of vectors, from the initial vectors that have been restored. This only saves time on a minority of faults. Other modifications, discussed in this paper, will take over the majority of the savings.

VII. Multi-fault Compaction Algorithm

Pomeranz *et. al* [4] have discussed several multi-fault algorithms. Their latest RROR algorithms, handle multi-faults. These algorithms start from a fault with the highest detection time and proceed to include faults, which are part of vectors that are needed for detection of faults in the current cycle. Pomeranz *et. al* [4] also discuss multi-fault LROR schemes, which tries to restore 64 targets faults at a time.
Our implementation is similar yet, different. Similar to Pomeranz *et. al* [4] method 2, we restore a batch of 64 faults, in parallel. However, unlike their method, we keep the list always populated. In our scheme, there are always 64 faults being restored in parallel. The attempt is always made. When a fault is detected, it is replaced by an undetected fault of a lower detection time. It must be noted that just because there are 64 faults in parallel, will not mean, that the PROR speed up is 64 times higher than a PROR without it, because different faults need different restoration vector lengths. Instead, however, it definitely means that the 32-bit parallelism of the machine will always be exploited.
Method 1 scheme also rearrange the order of vectors. As we discussed earlier, this is not good for ATPG, as this process will be repeated more than often. In case of ATPG, what happens is that vectors for faults with lower detections are constantly being dropped. ATPG also benefits from our implementations, as it produces schemes with better spectral properties, by compacting them.
To improve quality of multi-fault compaction, when a target fault is introduced, do not restore any vectors, for the first cycle. Normally, a weaker fault will be a detected by vectors for a harder-to-test fault with a higher detection time. This

saves vector length.

VIII. 2-level Compaction

One more scheme that have not been previously discussed in the literature, by others, is use of 2-level compaction schemes. Basically, the idea is as follows:

- Restore vectors
- Once all faults have been restored, fault simulate for *all* faults
- For undetected faults, do restoration. Note that restoration from the previous step does not guarantee all faults to be detected. This is because the algorithm restores a select number of faults at a time. The result will not carry over to all faults.
- Once all faults have been restored, fault simulate for *only* the undetected set from the previous pass.
- For undetected faults, do restoration.
- Worry *only* about the undetected fault list.
- Once undetected faults is empty, do fault simulation for *all* faults
- Repeat until all faults are detected.

This is 2-level compaction scheme. Notice the difference. It only tries to concentrate on the undetected fault list. This scheme saves a lot of time, by reducing the number of calls to the fault simulator with the fully-populated fault list. A populated fault list can consume a lot of time, in the compactor, if we keep making repeated calls to it.

IX. Flip flop signature based restoration

One of our observations, with running compactors, is that compactors spend a lot of their time, simulating the good machine. In fact, unlike fault simulators FS, good machine simulation is on par with, if not higher than bad-machine simulation, on compaction. So, we tried to devise a method that would minimize the number of good machine event counts.
Here's the mini algorithm:

- S_0 = initial FF state
- For each good machine simulation
- S_n = current FF state
- Let $FFchanges_n$ = count ($S_{n,j}$ ==$S_{n-1,j}$)) where j is the flip flop index
- Save $FFchanges_n$
- end loop
- Use $FFchanges_n$ for vector restoration, which is explained in further in future sections. This has been known to contribute to speed.

X. Multi-Pass Algorithm

Our algorithm, like Pomeranz *et al.* [4] does multiple passes on the compacted set to reduce it further. However, unlike theirs, it stops after 4 iterations. This is mostly because the compacted set is not of the highest quality since we are making trade-offs with time and the algorithm can most likely find a way to improve the quality further. Stopping multiple passes over a limit, keeps the initial goal of devising an algorithm for speed.

XI. Structure of the Program

For each pass, until no more improvement or terminating condition, do the multi-level algorithm:

- In each level, do PROR:
  - Target multiple faults
  - Pick faults with high detection time
  - calculate K = max(1, no.vectors/10000)
  - For each target, calculate $P(n) = (H(n)G(n) + 1) * K$ where n is number of iterations the fault has stayed in the compactor
  - For undetected faults, restore $P(n)$ vectors.
  - Replenish target fault list as more faults are detected
  - Repeat process until target fault list is empty

A. Restoration Procedure Details

Unlike Pomeranz *et al.* [4] methods, in our implementation, $(H(n)G(n) + 1) * K$ is the number of newer vectors to restored from the higher detection time. That would mean if there are vectors, in between, those vectors are not included in the count, and the algorithm will start scouting towards the beginning to restore more vectors to meet the requirement. Let's call this absolute location $L(n)$.In general, the method is efficient and offers a lower trade-off with quality.

B. Improving Quality

We added another trick to improve the quality, in vector restoration procedure. To improve quality, the algorithm considers 10 part restoration space by restoring vectors at P(n)/10 vectors at location L(n), at 9L(n)/10,8L(n)/10,etc. This improves quality significantly. We call this multi-space restoration.

C. Deterministic ATPG

PODEM is used on the last vector of restoration for a fault to promise early detection times with less vectors. The algorithm changes the last vector, if the fault can be detected within a backtrack limit of 30. It marks removed[i]=2 meaning the vector at i has been changed. Algorithms must be careful while restoring vectors since it may conflict with other fault requirement and would have to restore the vector to its original in case that fault is not detectable.

XII. Flaky Compaction for ATPG

We discuss flaky compaction technique that provides definitive speed for compaction-based ATPG tools. Flaky compaction works by breaking the 2-level compaction algorithm. Basically, when the undetected list becomes empty, it does not check to see, if all faults are indeed detected. Instead, it terminates and goes into another pass. This increases speed of ATPG. There is little to worry about loss of detection of those faults, as they will be detected in subsequent iteration by even more stable sequences.

XIII. Complexity Analysis

PROR have a worst case complexity of $O(Nc)$. That is to say, the amortized cost, of PROR is $O(1)$. This may seem counter-intuitive. We will provide an informal proof as to why this is the case.

Suppose we would like to calculate the number of iterations, the PROR will go through, in the worst case for fault. First thing to notice is that suppose N is maximum no. of vectors, number of vectors restored cannot be greater than N.

$$\Sigma_{i=1}^{m} i^k (1 - \frac{i^k}{N}) \leq N \tag{3}$$

Because $\Sigma_{i=1}^{m} i^k$ is $m^{k+1}/(k+1)$ approximately, the expression simplifies to,

$$\frac{m^{k+1}}{k+1} - \frac{1}{N}\frac{m^{2k+1}}{2k+1} \leq N \tag{4}$$

Now, we would to know m, when the equality is met, because that's the maximum, it can reach. This equation cannot be solved exactly. So, we assume the following: $m \approx cN^{1/k}$, where c is some constant and check to see if it does solve the equation, we find, the that the $m \approx cN^{1/k}$ is one of the solution to the above equation. We leave it to the reader to verify this. Notice that

$$N^{1/k} = \exp \log N^{1/k} \tag{5}$$

$$= \exp 1/k \log N \tag{6}$$

$$\tag{7}$$

Using that $k \approx 1.1\frac{\log(N)}{\log 61518}$. We will get $N^{1/k} = 61518^{1/1.1}$. So, the amortized cost is $O(1)$. The complexity of the compactor is $O(Nc)$ or $O(N)$ is the complexity of fault simulation, in terms of vector length.

TABLE I
Wavelet ATPG [2] Results for PROR without fault sampling

Ckt.	Det. Flts.	Vec. Length
s38584.1	29281	113849
b01	128	81
b02	65	77
b03	391	305
b04	1431	315
b08	621	419
b10	448	1417
b11	1261	598
b20	32450	46755
b21	33482	52094
b22	48743	161403

XIV. Experimental Results

We ran our experiments on a PC. Our PROR compactor used the HOPE fault simulator and was written in C. We used wavelet ATPG [2] for test pattern generation to test the dexterity of our compaction schemes. The ATPG was modified to include a combinational ATPG initial sequence generator using a PODEM like technique. We also added some of the heuristics used in wavelet BIST [3], such as correlation. We report results on larger circuits and circuits where no one has previously reported like bXX series of ITC test benches with reset. Results indeed show that PROR may be suitable for large scale sequential ATPG. It is really fast and we ran the ATPG with and without using fault sampling option. Results are shown in tables I and II. This is good news since the ATPG ran for no more than 6 iterations on the larger circuits. Results may be better if it ran for more iterations. Sometimes PROR takes sequences $>$ 1 million in size and compacts them to 10% of their original size, as seen in table III.

TABLE II
Wavelet ATPG [2] Results for PROR with fault sampling

Ckt.	Det. Flts.	Vec. Length	Time
b01	128	86	20.1(sec)
b02	65	64	16.2 (sec)
b03	391	329	21.3 (sec)
b04	1431	519	40.7 (sec)
b08	419	522	30.0 (sec)
b10	448	1155	97.2 (sec)
b11	1261	1257	65.8 (sec)
b12	1622	6389	5.93 (min)
b13	256	97	25.1(sec)
b14	14479	5445	20.8 (min)
b15	15140	9605	47.5 (min)
b20	32422	50389	4.71(hr)
b21	33232	67205	7.73(hr)
b22	49118	146476	1.31 (day)

TABLE III
Compaction Results for PROR in Wavelet ATPG [2] iteration

Ckt.	I/F D F	TDF	IVL	FVL	Time
b22	1122/1140	49118	1593477	148837	10.8 (hr)

I/F DF: Initial/Final Detected Faults
TDF: Total Detected Faults
IVL: Initial Vector Length
FVL:Final Vector Length

XV. Conclusion

In general, our PROR compaction scheme did pretty good and may be suitable for industrial level benchmarks. This PROR set of compactors may be the first step in producing commercial grade sequential non-deterministic ATPG. Our results indicate that spectral wavelet ATPG combined with PROR may be the best solution in terms of speed and efficiency. In the future, we plan to make the algorithm a little bit deterministic and use better techniques for analysis. PODEM part can be improved. And so can plenty of other things, such as quality of the test sequences. Quality of test sequences will be a better topic in future installments of this series of papers. In conclusion, we built a good compactor and had better ATPG results.

Acknowledgment

The authors would like to thank...

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] Suresh Kumar Devanathan and Michael L. Bushnell, *Sequential Spectral ATPG Using the Wavelet Transform and Compaction*, VLSI Design, 2006, pp. 407-412

[3] Suresh Kumar Devanathan and Michael L. Bushnell, *Test Pattern Generation Using Modulation by Haar Wavelets and Correlation for Sequential BIST*, VLSI Design, 2007 pp. 485-491

[4] lrith Pomeranz, *Procedures for Static Compaction of Test Sequences for Synchronous Sequential Circuits Based on Vector Restoration*, DATE, 1998

[5] T M Niermann, *HITEC: A test generation package for sequential circuits*, EDA, 1991

[6] H. Fujiwara , *Fan: a fanout-oriented test pattern generation algorithm* ISCAS, 1985

[7] Hyung Ki Lee, Dong Sam Ha , *HOPE: an efficient parallel fault simulator* DAC, 1992, pp. 336-340

[8] H. K. Lee, D. S. Ha , *Atalanta: An Efficient ATPG for Combinational Circuits* ,1993