International Journal of Research
Available at https://edupediapublications.org/journals

e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 04 Issue-17
December 2017

Low Power Test Pattern Generator using LFSR for Speed up the ATP Process

D.Rekha & B.Siva Kumar

1. D.Rekha, M.Tech Student, Dept of Ece, Ellenki Institute of Engineering & Technology, Patelguda, Patancheru, Sangareddy (Dist),TS, India.
2. B.Siva Kumar, Assistant Professor, Dept of Ece, Ellenki Institute of Engineering & Technology, Patelguda, Patancheru, Sangareddy (Dist),TS, India.

Abstract: *A new test generation methodology is proposed that takes advantage of shared memory multi-core systems. Appropriate parallelization of the main steps of ATPG allocates resources in order to minimize workload duplication and multithreading race contention, often encountered in parallel implementations. The proposed approach ensures that the obtained acceleration grows linearly with the number of processing cores and, at the same time, keeps the test set size close to that obtained by serial ATPG. The experimental results demonstrate that the proposed methodology achieves higher degree of speed-up than comparable state-of-the-art multi-core based tools, while maintains similar test set sizes.*

I.INTRODUCTION

Technology shrinking in the integrated circuit manufacturing process allowed the implementation of multiple processing units (cores) on a single chip as well as large amounts of on chip memory. These developments offer extensive processing power that can be used in various computationally intensive problems including popular electronic design automation processes. However, the distributed fashion of this processing power guides towards the development of parallel methodologies that scale well as the number of cores per chip are expected to increase beyond two dozens to hundreds. Automatic Test Pattern Generation (ATPG), a well-known NP-hard problem, becomes more demanding as devices

under test are becoming larger and more complicated and as emerging defects require new fault models of higher complexity. While previously proposed procedures are very effective, see the recent works in[1]-[3], among many others, they are inherently non-parallel and thus, cannot rely on automatic parallelization using sophisticated compilers. Proper problem decomposition, workload distribution and final test set recomposition are essential to guarantee the quality of the results while maintaining fault coverage. Since, typically, each core does not consider the entire search space, parallel approaches tend to choose local optimal solutions resulting in test set increase[4], known as the test inflation problem. Parallel ATPG has been studied before the on-chip multicore era, by either applying bit level parallelism or distributing ATPG components among multiple processing units, not

physically on the same chip [4,5]. These approaches were designed to avoid/minimize communication overhead and were constrained by the machine’s word size. In current on chip multi-core architectures with shared memory, on-chip communication is much faster, significantly reducing the cost of inter-core communication. Furthermore, high level of memory coherency is guaranteed and the number of available cores keeps increasing. These new developments and trends motivate towards the investigation of parallel

ATPG approaches capable of achieving speed-up scalability as the number of on-chip cores increases, while overcoming new challenges such as shared memory contention, as well as efficient workload distribution parallel threads. Recent works on ATPG parallelization for on-chip multicore environments exploit a variety and, often mixture, of parallelism dimensions such as fault parallelism, structural (circuit) parallelism, and algorithmic (including search-space) parallelism. Moreover, the goal of utilizing parallelism often varies. For example, [6] exploits algorithmic parallelism via SAT solver parallelism for maximizing fault coverage with limited speed-up with respect to the corresponding serial process. Similarly, [7] applies bit-level parallelism to generate multiple test patterns concurrently that meet different quality metrics to achieve higher physical-aware n-detect coverage. Static fault parallelism is explicitly considered in [8] using a master-slave architecture to reduce inter-process communication which achieves sub-linear speedup up to 8 cores but suffers from increased test set sizes (test inflation).

Parallelization speed-up rates and test set inflation are investigated in the recent work of [9] which also considers a shared memory architecture model. Shared memory is utilized as an extremely low latency communication mean with high capacity to leverage synchronization and communication of the process. The work in [10] proposes a low communication circular pipeline parallel ATPG procedure which emulates the deterministic execution of a serial ATPG in order to be able to reproduce the same test set every time the parallel algorithm is executed. This leads to limitations in speedup scalability. The series of works in [11]-[13] target both parallelization speedup and test inflation minimization strategies, incorporated in state-of the art commercial tools. In particular, [11] achieves high speed-up by applying dynamic fault partitioning and depth-first-search based compaction in a shared memory architecture. [12] extends [11] to be used in distributed multicore hybrid architectures, while [13] incorporates a copy-on write technique for private data protection in order to reduce memory locking when the same part of the memory is used currently by more than one cores. Similarly to the above approaches, the work proposed here is targeted towards achieving high degree of speed-up, as the number of available cores increases, and at the same time limiting test set inflation. Some parallel approaches have also been proposed targeting Graphic Processing Units (GPUs) based architectures. In contrast to the fault simulation problem where the GPU model can be very effective due to its concurrent nature which can directly adopt the single instruction multiple data (SIMD) approach of GPUs [14,15], GPU-based ATPG has received limited attention [16,17]. This is mainly attributed to the architecture's memory limitation which leads to unacceptable test set size increase. In this work we propose a parallel ATPG methodology for shared-memory systems geared towards high speed-up and test inflation containment. The methodology takes advantage of fast and low cost shared memory communication inherent in the underlying architecture in order to coordinate the main steps of the ATPG to avoid redundant work and dynamically allocate the workload while minimizing memory contention caused by multiple cores (threads) when accessing shared data. A test generation flow is proposed in which hard-to-detect

faults are targeted first, followed by a parallel fault simulation based merging process to maximize fault coverage. This process employs a series of newly proposed parallelization heuristics to explicitly avoid simultaneous consideration of the same faults by two or more cores, in order to minimize extra work and thread idle time. Any remaining undetected faults are targeted during a following phase, in a similar manner. The obtained experimental results demonstrate the effectiveness of the proposed approach in speeding-up the ATPG process

and provide comparisons with relevant recent work.

The rest of the paper is organized as follows. Section II presents a high level description of the proposed parallel ATPG while Section III focuses on particular parallel optimizations used to reduce the test inflation problem and favor speedup. Section IV presents and discusses the experimental results and Section V concludes the paper.

II. PROPOSED HIGH-LEVEL ATPG

A common parallelization procedure consists of three steps: (i) decomposition (domain or functional), (ii) parallel execution, and (iii) final result assembly. Step (ii) can result in significant compromise of the quality of the obtained results and, at the same time, not offer the expected speed-up. An efficient parallel algorithm should effectively overcome challenges such as memory contention and imbalanced workload distribution. The proposed ATPG method appropriately designs all three steps to ensure that these challenges are treated efficiently. Specifically, two conceptual approaches are adopted: (i) problem partitioning to avoid executing the same work concurrently in different cores and (ii) fine-grained granularity of each step to provide dynamic distribution of work. Various parallel optimization heuristics based on these concepts are discussed in Section III; this section presents the test generation flow of the proposed methodology which is based on the rationale of these two concepts. The proposed

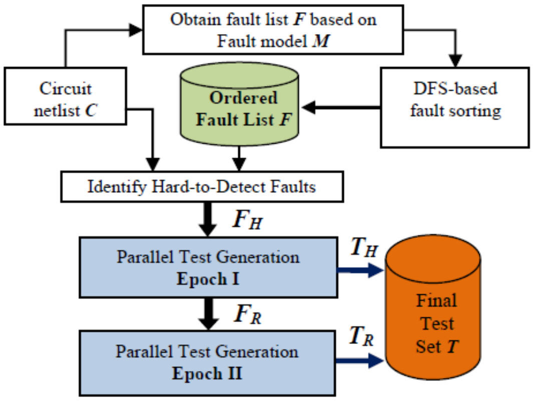

Obtain fault list F based on Fault model M
Circuit netlist C
Ordered Fault List F
DFS-based fault sorting
Identify Hard-to-Detect Faults
F_H
Parallel Test Generation Epoch I
T_H
F_R
Parallel Test Generation Epoch II
T_R
Final Test Set T

Fig: High level flow of the main Test Generation (TG) processes

single fault ATPG process, provides the desired granularity that allows mutually exclusive is attribution of work in the different cores. Such distribution benefits the exploration of different parallelization directions, including dynamic partitioning and adaptive decision making for test merging. However,

an approach with high granularity may perform large amount of unnecessary work when not taking advantage of fault dropping. Fault dropping plays a critical role in test generation anyway, as it can significantly affect test set size. In parallel test generation, inefficient dropping of faults can also restrict speed-up, regardless from the fact that the main process for identifying faults to be dropped (fault simulation) can be implemented very efficiently in parallel environments [14,15]. A fair trade-off between high granularity and fault dropping consideration is to develop a methodology based on distinct test epochs, one targeting hard-to-detect faults and a following one targeting the remaining undetected faults. Fig.1 presents the high level description of the proposed methodology. Firstly, the circuit netlist is analyzed to obtain a collapsed fault list ***F*** for the underlying fault model ***M***. Consequently, the fault list is sorted in a depth-first-search (DFS) order (based on their location in the netlist) in an attempt to implicitly group faults with structural similarities in ***F***. This fault locality property of the input fault list benefits fault dropping after ***F*** is partitioned to the available cores. The next step identifies hard-to-detect faults to be targeted by the first test epoch (Epoch I) of the methodology. We use random test pattern generation, which is a simple, quick and acceptable way to classify faults; however, other more sophisticated methods can be incorporated. Hard-to-detect faults are identified using a multiple detection approach where 10% (set by experimental exploration) of the faults in ***F*** with the fewer detections are considered as hard and used as the input fault list of test Epoch I (***FH***). Epoch I performs explicit test generation for each fault in ***FH*** while also considering faults in ***F−FH*** during fault simulation to identify faults detected coincidentally (***FC***). A merging process to reduce the number of tests obtained follows, and the final output is a set of test patterns ***TH*** detecting all faults in ***FH*** ∪ ***FC***. A second test epoch(Epoch II), similar to the first one, is invoked to target the remaining faults, i.e. ***FR*** = ***F* – (*FH*** ∪ ***FC*)** producing a set of tests ***TR*** such that ***T*** = ***TH*** ∪ ***TR*** detects all faults in ***F***. Detailed description of the individual steps taken during a test epoch is

provided in Section III.A.

III.PARALLELIZATION METHODOLOGY AND OPTIMIZATIONS

This section describes in further detail how the test generation process is partitioned and discusses the decisions taken to address the main parallelization challenges presented in Section II. Section III.A describes the major steps undertaken during a test epoch, discussing dynamic fault partitioning and core synchronization, while Section III.B describes a number of optimizations proposed to overcome parallelization issues.

A. Test-Epoch Parallelization :Fig.2 presents a flowchart illustrating the basic steps of the parallel Test Generation (TG) methodology followed during a test epoch, namely *seed-based TG* and *dynamic test merging and restricted TG*. An epoch explicitly targets on a fault-by fault basis, only a small subset of the fault list ***F*** (***FH*** for Epoch I and ***FR*** for Epoch II). Note that ***FC*** = ***F* – (*FH*** ∪ ***FR*)** typically constitutes the overwhelming majority of the faults which are easily detectable in an implicit manner (i.e., via fault simulation). The faults in a fault list are sorted based on structural similarities of fault locations (netlist), in order to increase the probability of proximate faults to be detected by the same test. During the first step (seed-based TG in Fig. 2), each available core performs test seed generation (TG with maximal don't care bits) for the next undetected fault ***fi*** in the list using a PODEM-based process optimized to identify tests with a large number of unspecified bits. The order of the selection of the next fault(s) is not important here, as the partitioning is designed to work in an independent manner and produce standalone results. The system shared memory holds the updated fault list indicating faults not yet targeted) and, therefore, duplication of work is avoided as each core works on a distinct fault. For each test seed ***ti***generated during this step parallel fault simulation is performed and all faults detected (including those in ***F -FH***) are stored in a list ***di***. Faults in ***di*** are not immediately dropped as this information is used during the following step. Also, the input *necessary assignments* (NA) of ***ti***are stored, along with ***di***, to be exploited in the next step. This first step terminates when all faults in ***FH*** have been targeted, constituting a synchronization barrier in the process. ***TPF*** contains the test seeds and ***DPF*** contains the corresponding fault simulation results which are both kept in the shared memory. Upon completion of the first step, the next step is invoked (dynamic test merging & restricted TG in Fig. 2) in order to merge compatible tests and reduce the size of ***TPF***. Each core selects its primary test target ***ti***from ***TPF*** to be the test seed with the larger detection list *di* (test with the highest number of coincidental detections) and immediately marks it so that other cores cannot select it. Tests are selected in an iterative manner until no further merging is possible. A detailed description of this selection is given in Section III.B under *detection-based test selection*. This merging step is dynamic *due to the efficient communication of the merged tests through the shared memory.* Thus, in each iteration, the number of candidate tests for merging is reduced at a fast rate. Fig.3 shows the merging process undertaken by each core while the shared memory accommodates information about faults detected and tests discarded. The faults detected by the primary test ***ti***(kept in ***di***) are immediate dropped from further consideration. Then, pair wise compatibilities of the primary test ***ti***with each remaining test ***tj***of ***TPF*** are calculated and ranked based on increasing Hamming distance. The test pair (***ti***, ***tj***) with the smallest Hamming distance is thereafter selected to be merged. Upon merging, ***ti***is updated and ***tj***is discarded from ***TPF.*** Also, all faults in the corresponding list ***dj***are dropped from the shared fault list. When no further merging is possible, restricted TG for ***ti***is performed based on the necessary assignments (NAs) on primary inputs collected during the first step. NAs are used as hard constraints for test generation, yet only for faults corresponding to tests not already marked and having identical NAs as ***ti.*** This iterative step terminates when no more tests with identical NAs exist that could lead to further test discarding.

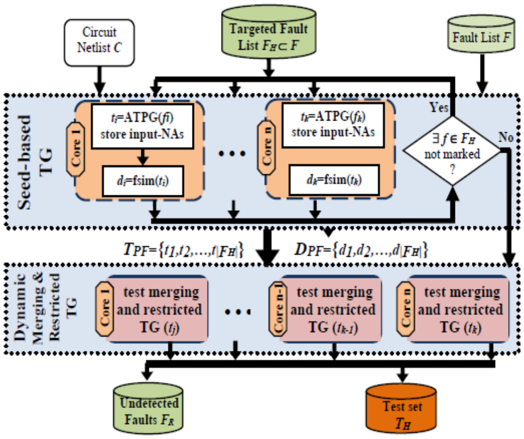

Circuit Netlist C
Targeted Fault List $F_H \subset F$
Fault List F
Seed-based TG
Core 1
t_i=ATPG(f_i) store input-NAs
d_i=fsim(t_i)
Core n
t_k=ATPG(f_k) store input-NAs
d_k=fsim(t_k)
Yes
No
$\exists f \in F_H$ not marked ?
$T_{PF}=\{t_1,t_2,\ldots,t_{|F_H|}\}$
$D_{PF}=\{d_1,d_2,\ldots,d_{|F_H|}\}$
Dynamic Merging & Restricted TG
Core 1
test merging and restricted TG (t_j)
Core n-1
test merging and restricted TG (t_{k-1})
Core n
test merging and restricted TG (t_k)
Undetected Faults F_R
Test set T_H

Fig: A test epoch targeting hard-to-detect faults (Epoch I). Same steps are repeated in Epoch II, with input fault list *FR* and resulting test set *TR.*

As a final step, the remaining unspecified bits of ***ti***are assigned specified values and the test is fault simulated to identify any further coincidental detection of faults. All the tests obtained by the process of Fig.3 are appended at the output of the corresponding test epoch, i.e., ***TH*** for Epoch I and ***TR*** for Epoch II.

B. Parallelization Optimizations: **Detection-Based Primary Test Selection.** In the merging step of Fig.3, test selection is very important for the efficient evolution of merging since it sets the constraints and outcomes of consequent merging iterations, restricted TG, and fault simulation. Practice in ATPG suggests that early fault dropping plays a more important role than having fewer constraints (more unspecified bits) in the test seed. For this reason, the *primary test* ***ti***during dynamic merging (merging seed) is selected based on its number of detected faults in ***di***. Recall that the fault simulation process performed at the end of the first step of the test epoch (Fig.2) does not drop faults;

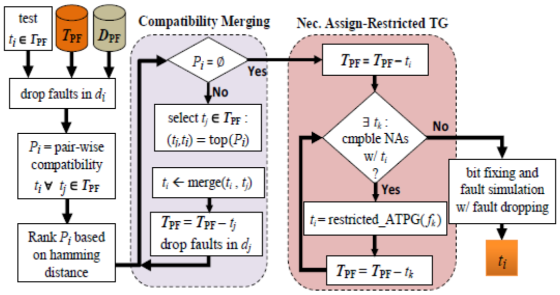

test
$t_i \in T_{PF}$
T_{PF}
D_{PF}
drop faults in d_i
P_i = pair-wise compatibility $t_i \forall\ t_j \in T_{PF}$
Rank P_i based on hamming distance
Compatibility Merging
$P_i = \emptyset$
Yes
No
select $t_j \in T_{PF}$: $(t_j,t_i) = \text{top}(P_i)$
$t_i \leftarrow \text{merge}(t_i , t_j)$
$T_{PF} = T_{PF} - t_j$ drop faults in d_j
Nec. Assign-Restricted TG
$T_{PF} = T_{PF} - t_i$
$\exists\ t_k$: cmpble NAs w/ t_i ?
No
Yes
t_i= restricted_ATPG(f_k)
$T_{PF} = T_{PF} - t_k$
bit fixing and fault simulation w/ fault dropping
t_i

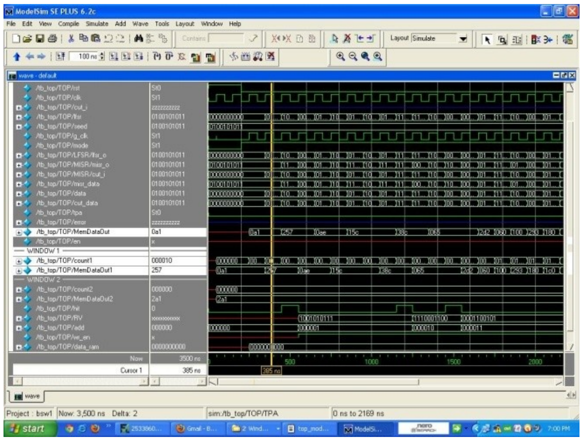
Fig: Dynamic Test merging and Restricted TG processes per core

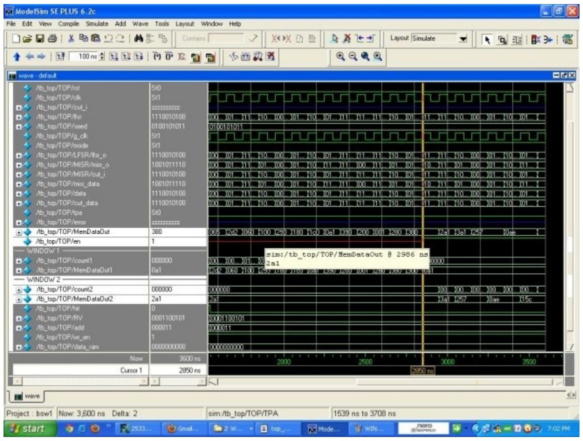
instead, it is used for providing a more precise metric for this selection during the second step. Tests to be merged (with the primary test) are then selected based on their Hamming distance to the primary test. The Hamming distance based merging produces merged tests with a smaller number of specified bits and, hence, fewer constraints in the following

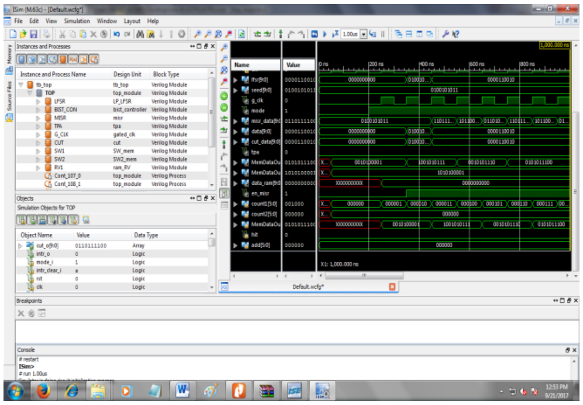
iterations of the merging process. In the (often common) case where more than one tests have the same Hamming distance to the primary test, their fault detection metric is used to decide which test will be merged. This optimization greatly assists in *dynamic workload balancing* and *minimization of unnecessary work* since high amount of early fault dropping reduces the faults for which explicit test generation is needed. Test Set Private Consideration. The search for the best candidate tests to be merged (either the primary or the ones to follow) involves high interaction of each core with the shared memory. Specifically, selecting the primary test, as well as computing the pair-wise compatibilities with the remaining tests in ***TPF,*** inherently involves memory contention since all cores are searching ***TPF.*** This issue is addressed by dynamically partitioning ***TPF*** in ***n*** private subsets (***n*** being the number of available cores), one for each core. Each core can only select tests from its own private subset of ***TPF*** (and the corresponding ***DPF***) which can be safely moved to its own private cache. This *implicitly minimizes concurrent memory access requests from different cores* that can result in inefficient memory utilization due to memory contention. Moreover, it implicitly *minimizes duplication of work* as each core considers a distinct part in ***TPF***. When a core finishes with the merging process within its private part of ***TPF***, it is allowed to work on the entire set in order to *ensure workload balancing by avoiding idle periods in cores.* At this point, concurrent memory accesses can occur, however, their impact is minimal as the bulk of the merging process has already occurred during the private consideration, and, hence, the size of ***TPF*** is by this point significantly reduced. Test Provisional Marking. During compatibility merging, the list ***Pi*** which holds pair-wise compatibilities between tests, requires updating after each merging. This updating is highly demanding in processing resources as it is of cubic complexity in the worst case. To avoid this issue the proposed methodology calculates and ranks compatibilities only once for each test ***ti***. If a test ***tj***is selected to be merged with ***ti***, it is provisionally marked in ***TPF*** so that it is not merged in another core, *explicitly avoiding imposing unnecessary constraints in another thread that performs merging*. If compatibility between ***ti***and a test ***tj***in ***Pi*** is invalidated by a previous merging, merging between ***ti***and ***tj***is not completed and the provisional marking is cleared. Otherwise, provisional marking indicates permanent discarding of ***tj***from ***TPF***. **Balanced Workload Distribution.** Distribution of workload to the available cores can significantly impact the speed-up of a parallel methodology. Test generation and fault simulation processes have unpredictable execution times due to the nature of the problems and fault dropping. Core idle time is minimized by dynamically selecting: (i) the next fault to be targeted in seed-based test generation in each epoch

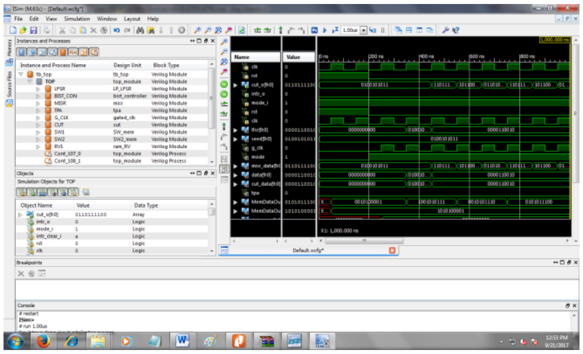
(Fig.2), (ii) the next test to be used as primary in test merging (Fig.2), (iii) the tests to be merged with the primary test seed (Fig.3), and (iv) the next fault to be targeted in restricted TG based on necessary assignments (Fig.3). Since, data is stored in shared memory (fault list and test seeds), and thus, is easily accessible by all cores, provides a punctual way of determining how the workload will be selected at each step and by each optimization mechanism of the approach.

IV.RESULTS









V.CONCLUSION

We propose a parallel test pattern methodology for shared memory multi-core environments. A number of newly proposed heuristics attempt to avoid assigning the same workload to multiple cores, while the distribution of work in the available resources to minimize core idle time. Experimental results demonstrate high speed-up rates that keep increasing as the number of the available cores increases. Test set size increase is limited and comparable to other state-of-the-art parallel approaches.

VI.FUTURE SCOPE

For very large designs and when testing at-speed fault models removes ATPG from the critical path for taping out a Design.

Multicore processorsare now widely available, but require additional tool support to maximize the compute power of these platforms.

This support extends the runtime benefits of distributed processing to multicore platforms, while significantly reducing the memory consumed compared to distributed processing.

The memory consumed by multicoreprocessingis less than half of the total memory required compared to an equivalent distributed processing run.

REFERENCES

[1] K. Scheibler, D. Erb and B. Becker, "Improving test pattern generation in presence of unknown values beyond restricted symbolic logic," in Proc. of *ETS*, pp. 1-6, 2015

[2] S. Eggersglub, K. Schmitz, R. Krenz-Baath and R. Drechsler, "Optimization-based multiple target test generation for highly compacted test sets," in Proc. of *ETS*, pp. 1-6, 2014. [3] I. Pomeranz, "Generation of compact multi-cycle diagnostic test sets," in Proc. of *ETS*, pp. 1-1, 2013. [4] S. Patil and P. Banerjee, “Fault partitioning issues in an integrated parallel test generation/fault simulation environment,” in Proc. of *ITC*, pp. 718–726, 1989.

[5] J. Wolf, L. Kaufman, R. Klenke, J. H. Aylor, and R. Waxman, “An analysis of fault partitioned parallel test generation,” *IEEE Trans. on CAD*, vol. 15, no. 5, pp. 517–534, 1996.

[6] A. Czutro, I. Polian, M. Lewis, P. Engelke, S. M. Reddy and B. Becker, "Thread-parallel integrated test pattern generator utilizing satisfiability analysis," *International Journal of Parallel Programming*, vol. 38, pp. 185-202, 2010.

[7] K-Y. Liao, C-Y. Chang and JC-M Li "A parallel test pattern generation algorithm to meet multiple quality objectives," *IEEE Trans. on CAD*, vol.30, no. 11, pp. 1767-1772, 2011.

[8] K.-W. Yeh, M.-F. Wu and J.-L. Huang, “A low communication overhead and load balanced parallel ATPG with improved static fault partition method,” in Proc. of *Intl. Conf. on Algorithms and Architectures for Parallel Processing*, pp. 362-371, 2009.

[9] JC-Y. Ku, RH-M. Huang, LY-Z. Lin and CH-P. Wen,"Suppressing test inflation in shared-memory parallel Automatic Test Pattern Generation," in Proc. of *ASP-DAC*, pp. 664-669, 2014.

[10] K-W. Yeh, J-L. Huang, H-J. Chao and L-T. Wang "A circular pipeline processing based deterministic parallel test pattern generator," in Proc. of *ITC,* pp. 1-8, 2013.

Author Profile’s:

Ms.D.Rekha has completed her B.Tech in ECE Department from DVR College of Engineering &Technology, JNTU Hyderabad. Presently she is pursuing her Masters in VLSI System Design in, Ellenki Institute of Engineering and Technology, Patelguda(v), near BHEL, Hyderabad, India.

MrB.Siva Kumar has completed BTech (ECE) from Hi-Tech College of Engineering& Technology from JNTUH University and M.Tech (VLSI) from Sree Venkateshwara Perumal College of Engineering &Technology from JNTUA University. He is having 6 years of experience in Academic, Currently working as Associate Prof at Ellenki Institute of Engineering and Technology, Patelguda(v), near BHEL, Hyderabad, India.