An Efficient Dynamic Parallel Approach to Automatic Test Pattern Generation

H.-Ch. Dahmen, U. Gläser, H. T. Vierhaus

The German National Research Center for Information Technology (GMD)
System Design Technology Institute
Schloß Birlinghoven
D53757 St. Augustin
email: [dahmen, glaeser, vierhaus]@gmd.de
Tel: (+49)2241-14-2875, FAX: (+49)2241-14-2242

Abstract

Automatic test pattern generation yielding high fault coverage for CMOS circuits has received a wide attention in industry and academic for a long time. Since ATPG is an NP complete problem with complexity exponential to the number of circuit elements, the parallelization of ATPG is an attractive of research.

In this paper we describe a parallel sequential ATPG approach which is either running on a standard network of UNIX workstations and also, without any changing of the source code, on one of the most powerful high performance parallel computers, the IBM SP2.

The test pattern generation is performed in three phases, two for easy-to-detect faults, using fault parallelism with an adapting limit for the number of backtracks and a third phase for hard-to-detect faults, using search tree parallelism.

The main advantage over existing approaches is a dynamic solution for partitioning the fault list and the search tree resulting in a very small overhead for communication without the need of any broadcasts and an optimal load balancing without idle times for the test pattern generators.

Experimental results are shown in comparison with existing approaches and are promising with respect to small overhead and utilization of resources.

1. Introduction

Automatic test pattern generation for combinational logic based on the FAN algorithm [FuSh83] has reached a high level of maturity. FAN has also been modified for test generation in synchronous sequential circuits presented in [ChBA88], [SuAu89], [ChCh89], [GoKa91], [NiPa91], [GlVi95], [GlVi96]. Although good experimental results have been shown for all those approaches, they are still no solution to the sequential ATPG problem, since they do not provide good results for complex real-life circuits.

The main objective of our present work is to develop a distributed algorithm for test pattern generation with the following features:

- reducing the run times for test generation for large circuits from days to hours
- executable on any network of workstations (including heterogeneous networks) and on the high performance parallel computer IBM SP2
- minimal overhead for communication
- optimal load balancing without idle times for the test pattern generators

To meet these features, we designed a communication scheme with "active servers" for test pattern generation and a "passive client" to organize the fault list and one server to handle fault simulation. In contrast to standard client server models, our "active servers" for test pattern generation send messages to the fault list handler client, requesting a fault to be treated. In this way there is no need for any broadcasting to synchronize the test pattern generators, synchronization is done by the fault list handler, distributing fault by fault after every request from one "active server".

This communication scheme with a dynamic dividing of the fault list and of the search tree is in contrast to the existing approaches with mostly a static dividing of the fault list and of the decision tree. Since the communication overhead need in our approach is quit low, we can use any standard network of UNIX workstations for test pattern generation even with slow connections between, the workstations. For even faster test pattern generation we can use the IBM SP2

1066-1395/97 $10.00 © 1997 IEEE

high performance parallel system (or any other parallel computer with UNIX nodes).

The test pattern generation in each node is based on the FOGBUSTER-algorithm [GlVi95] and [GlVi96]. In contrast to the BACK-algorithm [ChBA88] FOGBUSTER uses a forward propagation and backward justification technique, which is in general more efficient than the exclusive reverse time processing that BACK uses.

2. Related Work

Parallel ATPG approaches have been proposed recently. Similar approaches for combinational ATPG are described in [AgTo93], [KlKa93] and [KlKa93]. A similar algorithm with full partitioning of the fault list over the test generation processes is presented in [AgAg93]. Another approach distributing the search space is shown in [RaBa92] and [Krau94]. In [BuAg95] an approach with an adaptive time limit is shown, also using fault list and search tree parallelism.

The aim of this paper is to compare our parallel approach with the existing approaches in order to show the advantages of our dynamic communication scheme over the existing approaches.

Our approach is dynamic since during the first two phases, the fault list is dynamically divided by a fault list handler process, and during a third phase the decision tree is dynamically distributed between the TPG-nodes. This is in contrast to most of the existing approaches, which divide the fault list or the decision tree statically, resulting in an increasing amount of communication at the end of the ATPG, when the process of dividing must be re-done several times to ensure a good load balance.

The rest of the paper is organized as follows: In section 3 we provide details of the parallel sequential ATPG approach. In section 4 some implementation details are presented. Experimental results and comparison with the existing approaches are shown in section 5, followed by conclusions.

3. The Parallel ATPG Approach

In the following, our PARSET algorithm (Parallel Test Generation for Synchronous Sequential Circuits) is described in detail.

3.1. The algorithm overview

PARSET aims at parallel test generation with each test generator handling a different fault at the same time (parallelization via the fault list) during the first two phases, during the third phase each test generator handles the same fault but a different part of the decision tree (parallelization via the search tree). Whenever a test generator finds a test sequence for a fault or a fault is proved to be redundant, this is reported to the fault list handler. In case a test sequence is found, this sequence is simulated by the fault simulator. In case of redundancy the fault list is updated (during the third phase only if all the test pattern generators report redundancy), and the fault simulation updates also its own fault list for redundant and covered faults.

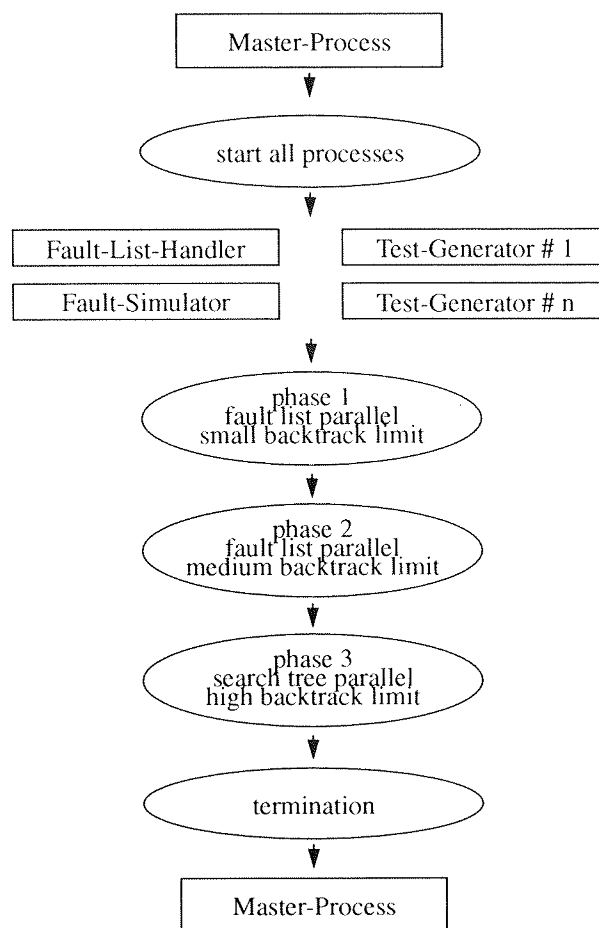

*FIGURE 1:*PARSET Overview

We found out that in general it does not make sense to parallelize the fault simulator because this parallelization will not increase the efficiency of the approach significantly. The reason is the different complexity of test pattern generation (> NP complete) and fault simulation ($\in$ P).

3.2. The processes

PARSET consists of four different processes:

- The master-process (MS): A single top level process handling subprocess incarnation and subprocess starvation.
- The fault-list-handler (FLH): A single process handling the fault list, a "passive client".
- The fault-simulator (FS): A single process handling fault simulation, a "normal server".
- The test-pattern-generators (TPG): A number of processes handling test generation for different faults and different parts of the decision-tree, "active servers".

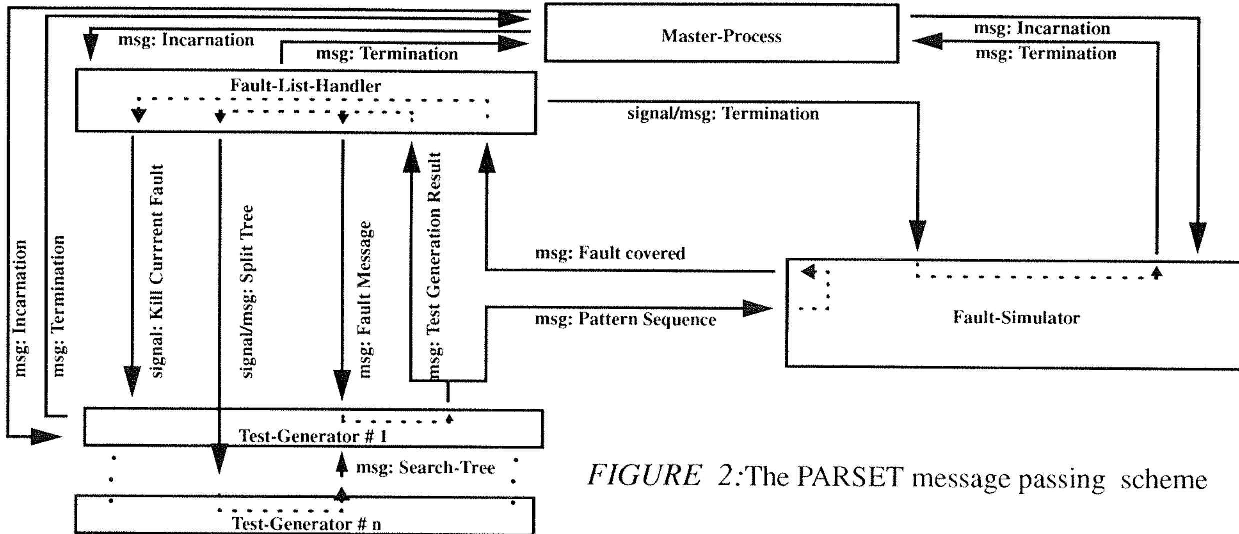

*FIGURE 2:*The PARSET message passing scheme

The master process starts all the other processes and watches for termination. In big intervals it checks all the processes for being alive, thus allowing to handle abnormal termination of the TPG. In case of abnormal termination of one TPG it sends a message to the FLH to reset the fault the terminated TPG was working on.

The FLH, the FS and the TPG are based on the same executable program. The MS sets them up to do their right work.

The FLH organize the fault list and gives the faults to the TPG and receives the results from the TPG and from the FS to update the fault list.

The FS receives test patterns from the TPG, simulates these patterns and sends the covered faults to the FLH.

The TPG work on one fault per time, during the third phase only on a part of the search tree. Results are reported to the FLH and in case of success to the FS.

3.3. The message types

The message passing scheme of PARSET is shown in figure 1 and is nearly the same during the three phases of test generation. There are eight different messages sent between the processes:

- Incarnation messages sent by the master to any subprocess and termination messages received from any subprocess by the master.
- A "fault message" is sent from the fault list handler to any test generator after a request from the test generator.
- The "pattern sequence message" is sent from a test generator to the fault simulator in case of success.
- The "test generation result message" is sent from any test generator to the fault list handler after finishing with one fault, thus also asking for the next fault.
- The "fault covered message" sent from the fault simulator to the fault list handler.
- A kill signal is sent from the fault list handler to any test generator if the fault which the test generator is actually dealing with is found to be covered by the pattern currently investigated by the fault simulator.

The kill signal is used to avoid redundant work in the test pattern generation process on one hand and to reduce the number of test patterns on the other hand. Although a test sequence for a covered fault is sometimes useful to cover some other faults, we decided to stop test generation in this case for reduced test generation time.

The following messages occur only during the phase for handling the search tree parallelism:

- The "split search tree signal" is send to a test generator from the fault list handler to start a splitting of the decision tree
- A "part of the search tree" is send from one test generator to another test generator after receiving the "split search tree signal"

3.4. Process algorithms

In the following, the main process algorithms are described with pseudo code.

```
fault_list_handler() {
do {
  if fault-covered-message received { (1)
    mark fault covered;
    remove fault from fault-list;
    if a test-generator is working on this fault
        send kill signal to test generator;
    }
  else if test-generation-message received { (2)
    analyze result of test generation;
    mark fault with result
    take detected fault from fault list;
    send next fault to test generator;
    if phase = 3
```

```
      send split signal to selected test generator
    }
  }while fault list is not empty;
}
```

The fault list handler receives two different types of messages. The first message type (1) is received from the fault-simulator whenever a fault is found to be covered. The fault list is updated after this message. If a test generator is currently working on this fault, a kill signal is sent to the test generator.

The second type of message is the test generation result for a fault received from a test generator (2). Due to the result of the test generation, the fault is mark tested, abandoned or redundant, respectively. Then a new fault of the fault list, which is not marked yet, is sent to the test generator. During the third phase another test generator is chosen to split his search tree and therefore the split signal is send to this test generator.

```
fault_simulator() {
while test sequence is received {
    fault simulate test sequence;
    for all faults covered in the fault simulation
        send fault-covered-message to fault-list-handler;
    }
}
```

The fault simulator waits for a test sequence from a test generator. When a test sequence message is received, the test sequence is fault simulated, and all the faults, which are covered by this pattern sequence, are sent to the fault list handler by a fault covered message in order to update the status of the fault list handler.

```
test_generator() {
send begin message to fault list handler;
  while a fault to test is received {
    if fault list parallelism
      receive decision tree
    test generation for fault with signal handling;
    send test-generation-message;
    if test-generation was successful
        send pattern sequence;
    }
}
```

After sending a begin message to the fault list handler a test generator receives a fault and the test generation function is called to generate a test sequence for the fault or to proof the fault untestable. The result of the test generation is returned to the fault list handler, thus asking for the next fault. If the test generation was successful, the test sequence is sent to the fault simulator. When a kill signal is received by a test generator, test generation is immediately stopped.

During the third phase the test generator waits for a part of the search tree after receiving a new fault from the fault list handler. During the test generation procedure a signal handling is performed to send a part of his own search tree to another test generator. This is done by scanning the decision tree from the root to the first optional decision, marking this decision as non-optional and sending this part of the decision tree to another test generator.

3.5. Adaptive backtrack limit

For limitation of the run time for one fault we use a limit for the number of allowed backtracks. This limit is very important for the result of the test pattern generation (used time and received fault coverage).

After a certain amount of non-successful ATPG-attempts, the backtrack limit is increased by factor.

The limits for multiplying the number of backtracks are based on experimental results.

3.6. Communication and memory management

The communication overhead is very low in our approach. Each process works on its own data structure copy. The message sizes for the communication are fixed to some few bytes with two exceptions: the pattern sequence and the parts of the search tree. For the pattern sequence, the message size depends on the length of the sequence, for the decision tree the message size depends on the position of the first optional decision, which is normally located near the root of the tree. We make use of the message packing in PVM [GeBe94] to keep the message size low. Thus the size of a test sequence or part of the decision tree package is usually not larger than a few hundred bytes.

One limitation of this approach is that every process has to store the data structure of the circuit and thus there is redundancy in the main memory management. This redundancy cannot be avoided when working on a network of workstations with low communication overhead.

A typical communication diagram is shown in figure 3. Each black line between the processes indicates a communication message between them. The starting point and the end point of the lines also indicate the times when the message is sent and received. It is visible that the fault list handler (FLH) and the fault simulator (FS) have the highest communication effort, while the communication effort of the test generators (TPG) is relatively low.

3.7. Idle times

The master (MS) has only little work to do and is thus idle most of the time. For this reason we normally start this process on the same physically machine as the fault list handler.

Due to the dynamic handling of the fault list and the splitting of the search tree, the idle times of the test generators are very low. The amount of idle time depends on the time the process is waiting for a fault from the fault list handler and, in phase three, on the time the process is waiting for a part of the search tree. If the fault list handler is idle at the time, a test generator finished its work, a new fault is immediately returned to the test-generator and there

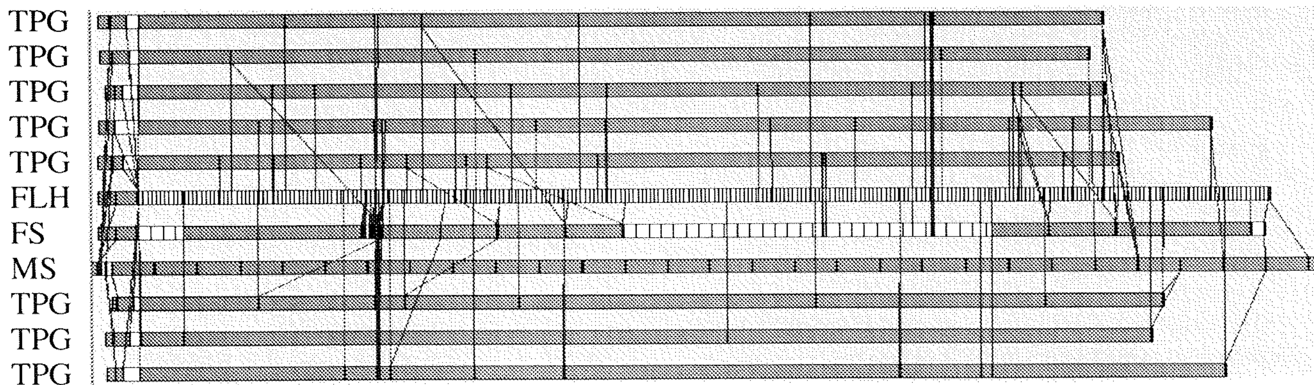

FIGURE 3: An XPVM output displaying communication of processes and idle times in s208 circuit with 8 TPGs

is no idle time. In some cases, when the fault list handler is communicating with the fault simulator, there is some small amount of idle time for the test generator, since the communication of the fault list handler with the fault simulator has higher priority than the communication with a test generator. This priority is given to the communication with the FS to avoid sending one fault to a TPG and then receiving a covered fault message from the FS for this fault resulting in a kill message to the TPG. The time to wait for a part of the search tree is also very short, because the signal handling is performed immediately by the test generator that should send the part of the decision tree. We found in experiments that the idle times of the test generators are usually less than 1% of the overall time of a test generator.

4. Implementation issues

PARSET is implemented in C++ language with about 20.000 lines of code. For parallelization we use PVM software (Parallel Virtual Machine) [GeBe94] and UNIX signals [Brow94]. The PVM library is available as public domain software for most of the existing UNIX Hardware and also from IBM with the same functional interface for the SP2.

5. Experimental results

We tested the software on the ISCAS '89 benchmark [BrBr89] circuits using the stuck-at fault model with the reset option. For the test generation processes we used SUN IPC and SUN Sparc 1 workstations plugged to the GMD Ethernet network and the IBM SP2 at GMD with 34 nodes, using standard Ethernet communication between the nodes.

Due to user restrictions in using the SP2 we can not show at this moments results for all circuits on the SP2. We present results for most of the circuits up to the s526 and also one result for the s1423. The wall clock time for test pattern generation for the s1423 was externally limited to 25 hours, resulting in the missing of the time for FS, the killed time and the communication time.

In our final paper we will present more results on the SP2, including results for different numbers of processors on the same circuit to show the received speed up. From our current experiments we can report normally a linear speed up and sometimes a super-linear speed up like reported in other approaches.

In table 1 we show complete results for a number of circuits received on the SP2.

The 'Time for TPG' is the maximum of the times used in every TPG for test pattern generation. The 'Killed Time' is the sum of all times in test pattern generation which are aborted by a kill signal. 'PVM Time for Commun.' is the maximum of time used in the TPG for sending / receiving messages.

The small amount of communication time shows the efficiency of our communication scheme.

Table 1: ISCAS '89 BENCHMARK RESULTS FOR PARSET ON THE SP2

Circuit Name	Fault Coverage	Effi-ciency	Redun-dant Faults	Aborted Faults	Max Time for TPG in s	No of CPU's	Sum of Killed Time in s	PVM Time for Commun. in s
s298	89.73	95.27	16	14	18	4	<1	<1
s344	97.00	100.00	10	0	7	4	<1	<1
s349	97.08	99.42	8	2	4	4	<1	<1
s382	96.16	97.95	7	8	481	20	138	2
s386	81.77	99.74	69	1	8	4	<1	<1
s400	94.47	98.08	15	8	330	10	42	<1
s420	60.69	97.44	158	11	18	8	20	<1
s444	86.70	98.50	55	7	227	20	2,745	2
s526	80.04	84.10	22	86	12,967	8	5,184	12
s1423	43.23	44.72	25	862	90,000	32	17,312	415

In table 2 we show a comparison with other approaches. Note the following when comparing the run times:

- different hardware as listed in the table
- sometimes a higher efficiency by longer run times
- by some approaches can be found a big amount of idle and / or communication time not included in the TPG time.

The results for GENTEST par where taken from [AgAg93], for GENTEST adapt from [SiAg95], for ESSENTIAL / SCOAP from [Krau94] and for ProperTEST / SCOAP from [RaBa92.]

Table 2: ISCAS '89 BENCHMARK RESULTS FOR PARSET IN COMPARISON WITH OTHER APPROACHES

Circuit Name	PARSET SUN/Ethernet SUN SPARC IPX / 1+			PARSET IBM SP2 IBM SP2			GENTEST parallel SUN SPARC 2			ESSENTIAL / SCOAP HP 9000/720			ProperTEST / SCOAP SUN SPARC ?			GENTEST adaptive SUN SPARC 10		
	Fault Coverage Efficiency	Time for TPG (s)	No CPU	Fault Coverage Efficiency	Time for TPG (s)	No CPU	Fault Coverage Efficiency	Time for TPG (s)	No CPU	Fault Coverage Efficiency	Time for TPG (s)	No CPU	Fault Coverage	Time for TPG (s)	No CPU	Fault Coverage	Time for TPG (s)	No CPU
s298	95.21	83	8	95.27	18	4	93.3	193	4									
s344				100	7	4	95.0	105	4									
s349	99.42	80	8	99.42	4	4	95.7	44	4									
s382				97.95	481	20	100	47	12									
s386	98.96	62	8	99.74	8	4				89.58	135	40	81.8	28	8			
s400	92.31	4406	8	98.08	330	10	93.6	648	12									
s420	97.21	113	8	97.44	18	8	99.5	63	12									
s444	91.20	11,117	8	98.50	227	20	97.5	1,320	12							91.98	190	16
s526	86.88	31,444	8	84.10	12,967	8	81.2	3,641	12	53.51	3,983	40						
s713	100	3	8										98.8	6	8			
s820	96.12	2666	8				85.9	1,763	12									
s832	96.09	4793	8				79.9	2,309	12									
s953	82.85	1057	8				99.9	3	4									
s1196	100	25	8				100	7	4				100	0.9	8			
s1238	100	13	8										100	2	8			
s1423				44.72	90,000	32												

6. Conclusion and further work

This paper presents a new dynamic approach to parallel sequential test generation. The new PARSET algorithm allows for efficient test generation for synchronous sequential circuits. Idle times of the test generation processes are very low, due to a dynamic fault list handling and search space partitioning. Communication overhead is very low. The software is implemented using the PVM package and thus can work on an existing standard network of workstations and also on high performance parallel computers like IBM SP2. Experimental results for the ISCAS '89 benchmark circuits are encouraging. In further work we will mainly concentrate on two extensions of this method. On one hand we will introduce and test a first phase with a small backtrack limit for all faults including the faults covered through fault simulation to get probably some useful test patterns. On the other hand we will spend some research on the ordering of the fault list, because its seems to be a very important factor for efficient test generation.

7. References

[AgAg93] P. Agrawal, V. D. Agrawal, J. Villoldo: *Sequential Circuit Test Generation on a Distributed System*, Proc. DAC '93, pp. 107-111

[AgTo93] M. J. Aguado, E. de la Torre, et al.: *Distributed Implementation of an ATPG System using Dynamic Fault Allocation*, Proc. ITC '93, pp. 409-418

[BrBr89] F. Brglez, F. Bryant, D. Kozminski: *Combinational Profiles of Sequential Benchmark Circuits*, Proc. 1989 Int. Symp. Circ. and S ystems

[Brow94] C. Brown: *UNIX Distributed Programming*, Prentice Hall International (UK) Ltd.

[ChAb90] C. H. Chen, J. A. Abraham: *Mixed-Level Sequential Test Generation Using a Nine-Valued Relaxation Algorithm*, Proc. ICCAD '90, pp. 230-233

[ChBA88] W. Cheng: *The BACK-Algorithm for Sequential Test Generation*, Proc. ICCD '88, pp. 66-69

[ChCh89] W. Cheng, T. J. Chakraborty: *GENTEST An Automatic Test-Generation System for Sequential Circuits*, IEEE Computer '89, pp. 43-49

[ChMa93] K. T. Cheng, T. Ma: *On the over-speci- fication problem in sequential ATPG algorithms*, IEEE Trans. Computer aided design, october 1993, pp. 1599-1604

[ChSP88] W. T. Cheng: *Split Circuit Model for Test Generation*, Proc. 25. DAC 1988, pp. 96-101

[FuSh83] H. Fujiwara, T. Shimono: *On the Accelaration of Test Generation Algorithms*, IEEE Trans. on Computers (C-32), pp. 1137-1144, 1983

[GeBe94] A. Geist, A. Beguelin, et al.: *PVM: Parallel Virtual Machine - A Users Guide and Tutorial for Networked Parallel Computing*, 1994, avaiable via ftp from netlib2.cs.utk.edu

[GlVi95] U. Glaeser, H. T. Vierhaus: *FOGBUSTER: An Efficient Approach to Sequential Test Generation*, Proc. EURODAC '95

[GlVi96] U. Glaeser, H. T. Vierhaus: *Mixed Level Test Generation for Synchronous Sequential Circuits using the FOGBUSTER-Algorithm*, Accepted for Transactions on CAD

[GlVi92] U. Glaeser, H. T. Vierhaus: *MILEF: An efficient approach to mixed level automatic test pattern generation*, Proc. EURODAC '92, pp. 318-321

[GoKa91] N. Gouders, R. Kaibel: *Advanced Techniques for Sequential Test Generation*, Proc. 2.nd European Test Conference, Munich, 1991

[KlKa93] R. H. Klenke, L. Kaufman, et al.: *Workstation Based Parallel Test Generation*, Proc. ITC '93, pp. 419-428

[Krau94] Peter A. Krauss, *Applying a New Search Space Partitioning Method to Parallel Test Generation for Sequential Circuits*, SFB-Bericht 342/09/94 A (TUM-I9415), Institut für Informatik, Technische Universität München, 1994

[MaNi88] W. Maly, P. Nigh: *Built-in Current Testing-a Feasibility Study*, Proc. IEEE ICCAD'88, pp. 340-343, 1988

[NiPa91] T. Niermann, J. H. Patel: *HITEC: A test generation package for sequential circuits*, Proc. EDAC '92, pp. 214-218

[PaBa94] S. Parkes, P. Banerjee, J. Patel: *ProperHITEC: A Portable, Parallel, Object-Oriented Approach to Sequential Test Generation*, Proc. DAC '94, pp. 717-721

[RaBa92] B. Ramkumar, P. Banerjee: *Portable Parallel Test Generation for Sequential Circuits*, Proc. ICCAD '92, pp. 220-223

[SiAg95] J. Sienicki, M. Bushnell, P. Agrawal, V. Agrawal: *An Adaptive Distributed Algorithm for Sequential Circuit Test Generation*, Proc. IEEE EURO-DAC '95, pp. 236-241

[StMa91] T. Storey, W. Maly, J. Andrews, M. Miske: *Comparing Stuck Fault and Current Testing via CMOS Chip Test*, Proc. 2nd European Test Conf., Munich, 1991, pp. 149-156

[SuAu89] M. H. Schulz, E. Auth: *Essential: An Efficient Self-Learning Test Pattern Generation Algorithm for Sequential Circuits*, Proc. IEEE Int. Test Conf. 1989, pp. 28-37