

ELEC 7250
SEQUENTIAL PARALLEL FAULT SIMULATOR
Madhurima Maddela

Abstract

Fault simulators are used to determine which faults are detected by a test sequence. Designing fault simulators for sequential circuits is much more complicated than for combinational circuits because of the feedback and timing constraints. This report presents an overview of the sequential fault simulators and an at length analysis of two different asynchronous sequential parallel fault simulator models-PROOFS and PARIS which have been further modified to get better performance in the form of HOPE and VISION respectively. The comparison of performances on benchmark circuits shows that with a few exceptions, HOPE proves to be faster than the rest. This paper concentrates on synchronous sequential fault simulators alone.

I. INTRODUCTION

Logic circuits can be broadly classified as either combinational or sequential logic circuits. In **combinational logic**, the outputs depend only on the present inputs. But in **sequential logic**, the outputs are a function of not only the present inputs but of the past inputs as well. In other words, sequential logic has storage (memory) whereas combinational logic does not. Figure 1 shows the block diagram of a sequential circuit.

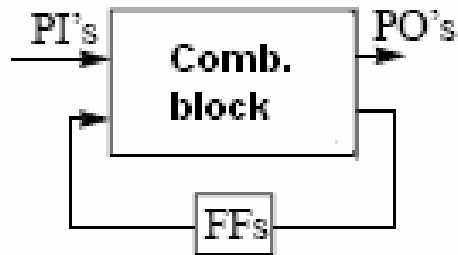


Fig. 1: Block diagram of a sequential circuit

Again, sequential circuits can either be synchronous or asynchronous. A clocked sequential logic is termed as **synchronous sequential logic**. It has a clock signal and all internal memory change at the clock edge. The basic storage element in synchronous sequential logic is a flip-flop. Every operation in the circuit must be completed inside a fixed interval of time between two clock pulses, called a 'clock cycle'. It is simple but has some disadvantages. The clock signal must be distributed to every flip-flop in the circuit, so power is dissipated; even the flip-flops that are doing nothing consume a small amount of power. Also, the maximum possible clock rate is determined by the slowest logic path in the circuit, otherwise known as the critical path. Hence the circuit is only as fast as the critical path. **Asynchronous sequential logic** is the most general kind of sequential logic, because of its flexibility, but it is also the most difficult kind to design. The basic storage element in asynchronous logic is a latch. Latches can change state at any time, depending on the transitions of other signals which may themselves be produced by other latches. The complexity of asynchronous circuits tends to rise very rapidly as the number of logic gates increases.

This paper takes us through different types of sequential fault simulators and discusses four of them in depth---PROOFS, PARIS, VISION and HOPE, in that order.

II. FAULT SIMULATION TECHNIQUES

With the development of VLSI technologies, test sequences with very high fault coverage have become increasingly important in maintaining acceptable field reject rates. Fault simulators are used to determine which faults are detected by a test sequence. This information not only grades the quality of the test sequence but also speeds up the test generation process. After a test sequence is generated for one target fault by a time-consuming test generator, a fault simulator is usually used for finding other faults that are

also detected. In this manner, the number of faults which need to be targeted by a test generator can be dramatically reduced. Fault simulators are also used to find test vectors by guiding search methods. In addition, fault simulators are used for generating fault dictionaries for diagnosis and for computing aliases in signature analysis; in both cases all faults must be simulated for the entire test sequence. These two applications require a very fast fault simulator with a very efficient memory.

Three simulation methods, concurrent fault simulation, parallel pattern single fault simulation and parallel fault simulation combined with single fault propagation are notable due to their efficiency, flexibility and /or versatility.

One of the oldest methods for sequential circuit fault simulation is **concurrent fault simulation**. In this technique, a fault-free circuit and all the faulty circuits are simulated simultaneously. The biggest advantage of concurrent fault simulation lies in its flexibility and versatility. It can easily accommodate various delay models and functional modules. But it is quite slow and requires a lot of memory.

Single fault propagation was originally developed for combinational circuits. In this technique, faults are simulated one at a time, under the application of a test pattern, and only the differences from the good circuit are simulated. This technique requires significantly less memory than concurrent fault simulation. This method was substantially improved later by simulating multiple patterns simultaneously and termed **parallel pattern single fault simulation** (PPSFP).

Mojtahedi and Geisselhardt proposed a new fault simulator, COMBINED which combines the original single pattern single fault propagation method and the PPSFP technique [8]. In COMBINED, faults are initially simulated by applying one pattern at a time. After a certain number of faults are detected, any remaining faults are simulated using the PPSFP technique. This is an example of **parallel fault simulation combined with single fault propagation** [6].

III. PROOFS

Cheng and Yu proposed a **differential parallel fault simulator (DSIM)** which is based on single fault propagation [9]. Like single fault propagation, DSIM simulates one fault at a time; but the difference is that DSIM simulates each fault, except the first based on the previous fault simulation result, while single fault propagation processes all faults based on the fault-free circuit simulation. The use of the previous fault simulation result enables DSIM to avoid restoring fault-free values before simulating each fault. Cheng *et al.* incorporated parallel fault simulation into DSIM. This new fault simulator, **PROOFS** simulates a packet of 32 faults in parallel [3].

The PROOFS fault simulation algorithm can be thought of as a hybrid of the concurrent, differential and parallel fault simulation algorithms. It retains the advantage of fault dropping that concurrent fault simulation allows, while exploiting the word level parallelism of the computer and retaining the low memory requirement of differential fault simulation. It uses a dynamic fault grouping strategy to fully utilize all the bit spaces in the computer word to simulate several faulty machines at once. This dynamic strategy avoids wasting space in the machine word for faults already detected. In addition the dynamic strategy removes faults from the parallel word in the time frames in which they

are inactive. A new technique to inject faults is used to avoid a computation overhead for each evaluation. Instead of using bit masks, the circuit is changed to reflect the faults that are active. The overall algorithm of PROOFS is shown in Figure 2.

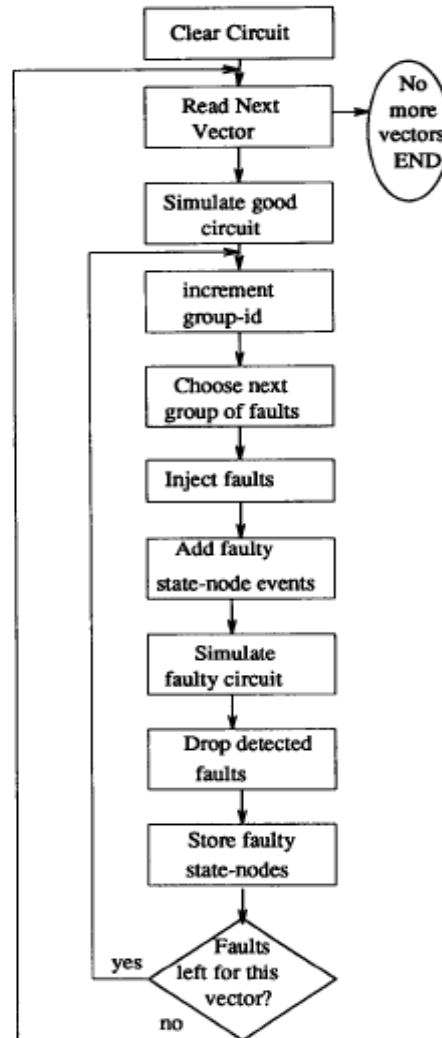


Fig. 2: PROOFS algorithm

It consists of a main loop which reads in the next input vector, does the good machine evaluation, and then for each fault grouping does the faulty evaluation. To evaluate a fault group, the group id is first incremented so that the faulty values of other machines are not used in the evaluation of this machine. Next, the 32 faulty machines are selected for the fault group. The faults are then injected into the circuit and the node values for the state nodes from the previous input vector are inserted into the faulty machines. The faulty machines in this fault group are evaluated, and the state-node values are stored for the next vector. A one-dimensional array is used to store the good circuit value of every line in the circuit. Another array stores the value of every line in the faulty circuit. Accompanying each faulty circuit line is a **group id** to indicate whether the line values of the faulty machines in the current fault group are different from the good machine value. Each value consists of two 32-bit words, V_0 and V_1 , where each bit is used to store a

different faulty machine's value. A four-valued logic (0, 1, X, and Z) is used. Two bits are used to code the four values, one in V_0 and one in V_1 ; 0 is coded as (1, 0), 1 as (0, 1), X as (0, 0), and Z as (1, 1). The use of a 32-bit word to evaluate 32 faulty machines in parallel is up to six times faster than a single-bit evaluation.

Definitions:

A synchronous sequential circuit can be decomposed into flip-flops and a combinational block. The outputs of flip-flops which are the inputs of the combinational block are called **pseudoprimary inputs (PPIs)** and the inputs of the flip-flops which are the outputs of the combinational block are called **pseudoprimary outputs (PPOs)**.

If the effect of a fault, f does not propagate to a flip-flop, i.e., a PPO at a time frame, the fault-free and the faulty values of all the PPI's are identical at a time frame, then it is called a **single event fault** for the time frame. On the other hand, if there exists at least one PPI whose fault-free and the faulty values are different, then it is called a **multiple event fault**. The propagation of the single and multiple event faults are shown in figure 3.

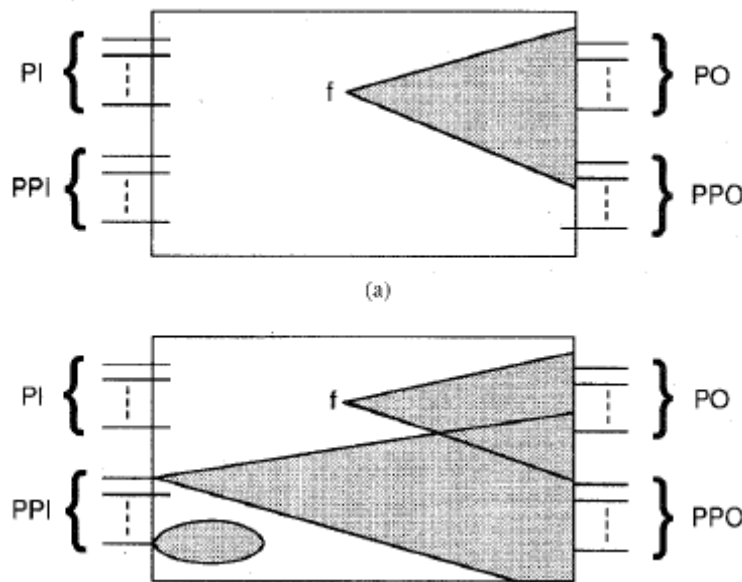


Fig. 3: Propagation of (a) single and (b) multiple event faults

Also, if a fault, f is not activated; it is called an **inactive fault**. A **Fan-out free region (FFR)** is a region obtained when the fan-out stems of a combinational circuit are removed.

The PROOFS algorithm described here is a 0-delay simulator. The use of a 32-bit word to evaluate 32 faulty machines in parallel is up to six times faster than a single-bit evaluation. PROOFS is not easily extensible to a variable delay simulation, since every node in the circuit is potentially a state node at one time or another. Also, PROOFS is not very suitable for switch-level fault simulator since many nodes at the switch level are potential state nodes because of charge storage.

IV. PARIS

Gouders and Keibel applied the PPSFP technique to synchronous sequential circuits and came up with a simulator called PARIS (**PAR**allel **I**terative **S**imulator). PARIS is the first and a unique parallel pattern fault simulator for synchronous sequential circuits. In PARIS, a packet of 32 test patterns are simulated in parallel under the injection of a fault. In order to accommodate the unknown state X, PARIS employs the 3-valued logic system: 0, 1 and X. A status word which comprises of two 32-bit words is used to represent the 32 logic values of a signal line. The least significant bit (LSB) of the status word represents the logic value corresponding to the first pattern and the most significant bit (MSB) represents the logic value corresponding to the last pattern. Circular partitioning is employed.

A. Logic and Fault Simulation:

For logic simulation, a packet of 32 patterns is applied to primary inputs. All the flip-flops are assumed to be at an unknown state initially. Every bit of each PPI word is set to the unknown state X. The unknown state is accurate only for the first bit. An event-driven logic simulation is performed to complete the first iteration of the logic simulation. For the second iteration, each PPO word obtained during the first iteration is shifted left by one bit, and it is placed on the corresponding PPI. The LSB of the new PPI word is again set to X.

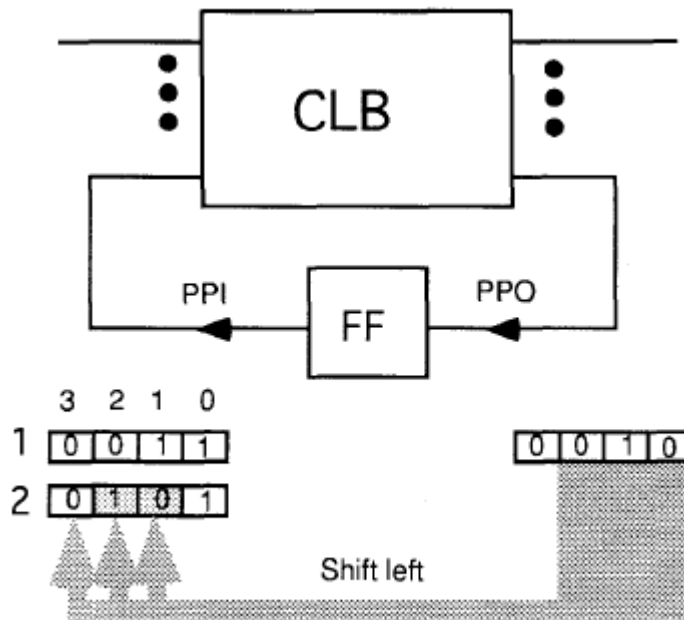


Fig 4: Multiple bit correction in PARIS

If the new PPI word is different from the existing PPI word, it means some initial state(s) of the flip-flops is incorrect, and it requires another iteration. Hence, in the worst case it may require 32 iterations to correct all the initial states of the flip-flops. After the states

of all the flip-flops are corrected, the MSB of each PPO word becomes the LSB of the PPI word for the next packet of test patterns.

The fault simulation is essentially the same as the logic simulation except the faulty circuit under the injection of a fault is simulated. Each iteration of the simulation corrects at least one initial state of the flip-flops under consideration. Consider an example shown in Figure 4. After iteration 1, bit 1 of the PPI word which is equivalent to the initial state of the flip-flops for the second test pattern is corrected to value 0. The new PPI word to be used for iteration 2 has two new bit values (shaded bits) compared with that of the previous one. Hence, there is a chance that bit 3 is also corrected in along with bit 2 during iteration 2. Due to the chances of correcting multiple bits, we can say that PARIS employs a **multiple bit correction** scheme.

B. Fault Dropping and Filler Bits:

One of the disadvantages of PARIS is that it drops faults only after simulating all of a packet of 32 faults. The delayed fault dropping results in unnecessary gate evaluations for faults which have been already detected by earlier patterns of the packet. This is particularly true for circuits in which a large portion of faults are dropped by a few patterns. Also, the number of test patterns is often not an integral multiple of the packet size (i.e. 32). In PARIS, this is dealt with by filling the remaining bit positions with unknown state X into the last packet. So the filler bits create unnecessary events for each iteration of the simulation.

In spite of these problems, PARIS performs better than the original version of PROOFS described before. This comparison is shown in Table 1. The only problem PARIS faces is with a huge circuit like s35932.

TABLE 1: Comparison of Fault Simulation Times of PARIS and PROOFS [6]

Name	CPU time (sec)		Speedup ratio PROOFS_O/PARIS_O
	PROOFS_O	PARIS_O	
s298	1.1	1.0	1.1
s344	0.8	0.8	1.00
s382	17.2	7.9	2.18
s444	20.8	15.5	1.34
s526	11.0	11.5	0.96
s641	1.4	0.6	2.33
s713	1.4	0.8	1.75
s820	5.1	3.5	1.46
s832	4.8	3.4	1.41
s953	0.5	0.4	1.25
s1238	4.9	0.6	8.17
s1423	3.8	2.1	1.81
s1488	12.7	10.1	1.26
s1494	10.7	12.8	0.84
s5378	51.5	10.3	5.00
s35932	173.1	1260.0	0.14

V. VISION

VISION was developed from PARIS to address its shortcomings through the addition of the following four new heuristics:

- A. Immediate fault dropping
- B. Filler bit injection into the first packet
- C. New method for look-ahead of initial states
- D. Single bit correction of flip-flop states

A. Immediate Fault Dropping:

In PARIS, each iteration of the logic of fault simulation corrects the logic value of each signal by at least one bit. Hence, after n iterations, it is guaranteed that the first n bits of any primary output are correct. If the first n bits of each primary output value are examined, it is possible to decide whether the fault under consideration has been detected or not.

VISION improves the speed of the algorithm by implementing immediate fault dropping, i.e., this method drops detected faults after the simulation of each pattern (each iteration) where as in PARIS, faults are dropped only after a packet of 32 patterns is simulated. The improvement in CPU time due to this is shown in Table 2.

TABLE 2: Performance benefit due to Immediate fault dropping [5]

Name	No. of gates	No. of tests	Fault cov. (%)	No. of gate evaluations (x 1000)		CPU time (sec)		Speed up ratio
				PARIS	VISION I	PARIS	VISION I	
s298	119	162	85.71	140	100	0.9	0.9	1.00
s344	160	91	96.20	117	76	0.7	0.5	1.40
s382	158	2463	90.98	1455	1038	9.0	8.2	1.10
s444	181	1881	89.45	2822	1973	16.0	14.1	1.13
s526	193	754	75.32	1827	1297	11.5	9.9	1.16
s641	379	133	86.30	97	59	0.8	0.7	1.14
s713	393	107	80.90	114	84	1.0	0.8	1.25
s820	289	411	81.88	547	320	4.1	2.7	1.52
s832	287	377	81.38	647	364	4.8	3.4	1.41
s953	395	16	7.78	48	47	0.6	0.6	1.00
s1238	508	349	94.69	53	53	0.8	0.8	1.00
s1423	657	36	24.42	416	375	3.3	3.5	0.94
s1488	653	590	92.60	2421	1183	17.9	9.7	1.85
s1494	647	469	91.10	2287	1298	16.6	10.2	1.63
s5378	2779	408	74.02	1301	1107	10.9	9.9	1.10
s35932	16065	86	87.99	140674	29435	1180.1	236.4	4.99

From this table, we can see that as expected the number of evaluations for VISION are lesser than PARIS as we have eliminated some unnecessary events. The average speed up

ratio is 1.48. The performance improvement is particularly significant for the largest circuit s35932 as compared to the case in Table 1.

B. Filler bits:

Instead of inserting the filler bits into the last part of the last packet as done in PARIS, in VISION, the filler bits are injected into the first part of the first packet. Initially, all the signal lines are at unknown state X. Hence, the filler bits do not cause any extra events and help avoid unnecessary simulation. The salient point of the proposed method is that it does not incur any additional processing overhead and it eliminates the need for masking the filler bit positions while checking for fault detection. Table 3 shows the results of implementing this heuristic. It is to be noted that at this point, the heuristics are being applied one at a time. So any change in performance can be solely attributed to the current heuristic being applied.

TABLE 3: Performance benefit due to Filler bit positioning [5]

Name	nf	No. of gate evaluations (x 1000)		CPU time (sec)		Speed up ratio
		PARIS	VISION_F	PARIS	VISION_F	
s298	30	140	146	0.9	1.0	0.95
s344	5	117	122	0.7	0.8	0.93
s382	1	1455	1447	9.0	9.1	0.99
s444	7	2822	2808	16.0	15.9	1.00
s526	14	1827	1824	11.5	11.4	1.01
s641	27	97	85	0.8	0.7	1.11
s713	21	114	95	1.0	0.8	1.16
s820	5	547	533	4.1	4.0	1.04
s832	7	647	725	4.8	5.3	0.91
s953	16	48	42	0.6	0.5	1.10
s1238	3	53	53	0.8	0.8	1.00
s1423	20	416	316	3.3	2.4	1.38
s1488	18	2421	1769	17.9	13.6	1.32
s1494	11	2287	2282	16.6	16.6	1.00
s5378	8	1301	1322	10.9	11.2	0.98
s35932	10	140674	77217	1180.1	616.9	1.91

From Table 3, we can see that again the biggest reduction ratio is achieved for the largest circuit s355932. The proposed bit filler scheme reduces the number of gate evaluations for 11 out of 16 circuits. Filler bits being in the last packet in PARIS and in the first packet in the current heuristic result in different packing of test patterns into packets. It can be seen that due to the above reason, this heuristic is not always advantageous but the performance improves for large circuits.

C. Look-ahead of initial states:

In general, a fault at a distance from a flip-flop tends not to propagate to the flip-flop. In contrast, a fault proximity to a flip-flop is likely to propagate to the flip-flop. For

sequential circuits, a sequence of test patterns is necessary to detect a fault. The sequence of test patterns may have a similar fault propagation characteristic.

In PARIS, the initial states of the flip-flops (i.e., the upper 31 bits of PPI words) for the logic simulation are the residual values of the previous logic simulations. For fault simulation, the initial states of the flip-flops are set heuristically. If the number of differences between the good circuit states and the faulty circuit states of a flip-flop after the simulation of the previous packet is smaller than 5, the initial states of the flip-flop are set to the current good circuit states. Otherwise, they are the residual values for the previous fault simulations. This is based on an assumption that the behavior of a faulty circuit is similar for two subsequent packets of test patterns.

So the new look-ahead heuristic provides a new method for setting the initial states of the flip-flops. Here the initial states of a PPI word are either the same as the value of the LSB or good values.

Table 4 shows the comparison between PARIS and the new heuristic. It can be observed that the number of evaluations is reduced; the speedup ratio ranges from 1.1 to 1.4. But the largest circuit is not helped by this heuristic.

TABLE 4: Performance of new Look-ahead of initial states heuristic [5]

Name	No. of gate evaluations (x 1000)			CPU time (sec)		Speed up ratio
	PARIS	VISION _L	Red. ratio	PARIS	VISION _L	
s298	140	123	1.14	0.9	0.8	1.12
s344	117	106	1.10	0.7	0.7	1.02
s382	1455	1204	1.21	9.0	7.9	1.13
s444	2822	2156	1.31	16.0	12.8	1.24
s526	1827	1399	1.31	11.5	9.0	1.28
s641	97	75	1.29	0.8	0.7	1.26
s713	114	95	1.20	1.0	0.9	1.12
s820	547	364	1.50	4.1	2.9	1.41
s832	647	492	1.32	4.8	3.8	1.26
s953	48	48	1.00	0.6	0.5	1.03
s1238	53	53	1.00	0.8	0.7	1.05
s1423	416	405	1.03	3.3	3.1	1.07
s1488	2421	1810	1.34	17.9	13.5	1.32
s1494	2287	1735	1.32	16.6	12.8	1.29
s5378	1301	1324	0.98	10.9	10.8	1.01
s35932	140674	145500	0.97	1180.1	1224.1	0.96

D. Single bit correction:

For a flip-flop which does not form a feedback path, the next state of the flip-flop is independent of its current state. Therefore, the next state can be set correctly in advance.

The multiple bit correction scheme is efficient for such flip-flops. However, the scheme may waste its effort for a flip-flop with a feedback loop. For example, suppose that the

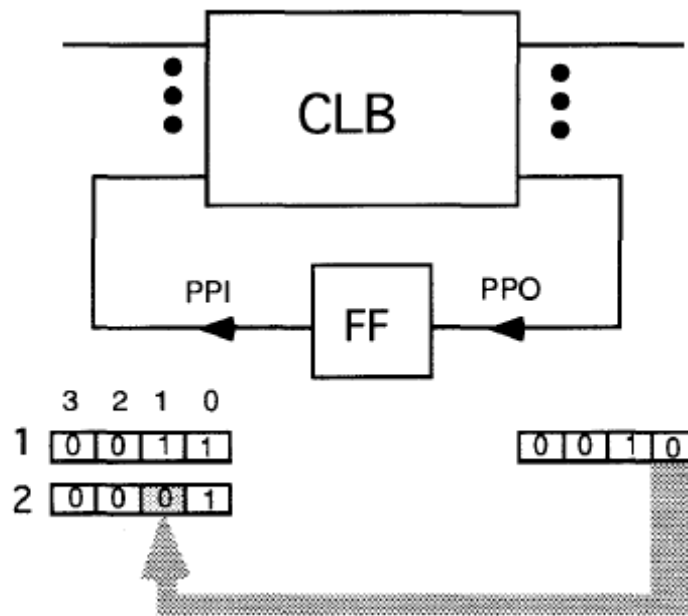


Fig 5: Single bit correction

TABLE 5: Performance of the Single bit correction heuristic [5]

Name	S	No. of gate evaluations (x 1000)		CPU time (sec)		Speed up ratio
		PARIS	VISION_S	PARIS	VISION_S	
s298	6	140	136	0.9	0.9	1.02
s344	4	117	107	0.7	0.7	1.02
s382	4	1455	1295	9.0	8.4	1.07
s444	2	2822	2374	16.0	13.9	1.15
s526	4	1827	1585	11.5	9.9	1.17
s641	7	97	93	0.8	0.8	1.09
s713	8	114	110	1.0	0.9	1.08
s820	4	547	526	4.1	3.9	1.06
s832	6	647	644	4.8	4.7	1.02
s953	5	48	48	0.6	0.5	1.10
s1238	4	53	53	0.8	0.8	1.00
s1423	7	416	415	3.3	3.2	1.03
s1488	7	2421	2294	17.9	16.8	1.06
s1494	7	2287	2212	16.6	16.1	1.03
s5378	∞	1301	1301	10.9	10.9	1.00
s35932	∞	140674	140674	1180.1	1180.1	1.00

correct value of bit 2 for the PPI word in figure 4 is 0. During the second iteration, the attempt to correct it to 1 creates unnecessary events to slow down the simulation. As the next states of the flip-flops depend on the present states as well as the current test pattern,

it is more advantageous to correct one state at a time. The purpose of the single bit correction scheme, as demonstrated in figure 5 is to correct only the bit whose present state is known. In other words, it corrects only the n^{th} bit for n^{th} iteration. All other bits remain unchanged during the iteration to minimize the number of events. Figure 4 illustrates the strategy. In the figure, only bit 2 is corrected during the second iteration. Since one bit is corrected, the number of events is less compared with that for the multiple bit scheme.

The results after applying this heuristic individually are shown in Table 5. The most significant improvement is achieved in s526 with a speedup ratio of 1.17. Otherwise, the comparison of the two speedup ratios shows that the new heuristic is only marginally better. The multiple bit correction scheme employed in PARIS is effective for the early stage of the simulation of each packet, and the proposed single bit correction scheme is effective for the late stage.

The overall performance of VISION as compared to PARIS when all the four heuristics are applied is shown in Table 6. The table indicates that VISION performs better than PARIS for all circuits except s5378. The average speedup of VISION over PARIS is 1.46. It is remarkable that VISION is faster than PARIS by more than 3 times for the largest circuit, s35932 due to the cumulative effect of all four heuristics.

TABLE 6: Overall Performance of VISION compared to PARIS [5]

Name	CPU Time (sec)		Speedup ratio PARIS/VISION
	PARIS	VISION	
s298	0.9	0.7	1.29
s344	0.7	0.6	1.17
s382	9.0	7.1	1.27
s444	16.0	10.0	1.60
s526	11.5	7.2	1.60
s641	0.8	0.6	1.33
s713	1.0	0.7	1.43
s820	4.1	2.5	1.64
s832	4.8	3.1	1.55
s953	0.6	0.5	1.20
s1238	0.8	0.8	1.00
s1423	3.3	2.3	1.44
s1488	17.9	7.3	2.45
s1494	16.6	10.5	1.58
s5378	10.9	11.5	0.95
s35932	1180.1	353.9	3.34

VI. HOPE

HOPE algorithm is based on PROOFS; it employs several heuristics to efficiently drop faults and avoid simulation of many inactive faults. HOPE essentially incorporates the following three heuristics into PROOFS:

A. Reduction of single event faults before being simulated in parallel:

The strategy here is to reduce the number of single event non-stem faults simulated in parallel in two phases. In the first phase, all single event non-stem faults inside fan out free regions are mapped to the corresponding stem faults by local fault simulation of the non-stem faults. In the second phase, mapped stem faults that are sensitive are examined further for possible elimination from parallel simulation. This method does not apply to multiple event faults.

TABLE 7: Reduction of faults performance of HOPE compared to PROOFS [6]

name	no. of gates	no. of flip-flops	no. of tests	no. of faults	occurrence of single and multiple event faults (%)		no. of faults simulated in parallel		
					single event faults	multiple event faults	PROOFS	HOPE _R	reduction ratio
s298	119	14	162	308	83.53	16.47	6131	2436	2.52
s344	160	15	91	342	79.83	20.17	2864	1351	2.12
s382	158	21	2463	399	64.17	35.83	83556	59318	1.41
s444	181	21	1881	474	60.35	39.65	106587	76531	1.39
s526	193	21	754	555	80.32	19.68	91694	39617	2.31
s641	379	19	133	467	93.01	6.99	7032	1530	4.60
s713	393	19	107	581	91.63	8.37	7986	1926	4.15
s820	289	5	411	850	98.46	1.54	51512	3404	15.13
s832	287	5	377	870	98.49	1.51	48851	3111	15.70
s953	395	29	16	1079	80.98	19.02	12132	5019	2.42
s1238	508	18	349	1355	97.64	2.36	53633	6713	7.99
s1423	657	74	36	1515	67.05	32.95	30091	18208	1.65
s1488	653	6	590	1486	98.16	1.84	59701	4128	14.46
s1494	647	6	469	1506	97.94	2.06	56227	4208	13.36
s5378	2779	179	408	4603	90.54	9.46	307550	86116	3.57
s35932	16065	1728	86	39094	85.14	14.86	553870	202627	2.73

The above table shows that this heuristic helps only when the occurrence of single event faults which was targeted in this heuristic is high.

B. Functional fault injection:

For example, consider a circuit with a two-input exclusive-OR gate whose function is $f(x, y) = x'y + xy'$. If input, y is injected with a s-a-1 fault, then the fault causes a new gate, $f(x, y) = x'$. Once all the faults are injected, all the gates are coded following the codes shown in figure 6. Values 1 to 9 are assigned for fault-free gates. Faulty gates get 20 +

faulty bit position as their index code. Now the lowest level gate is retrieved from the vent queue and the gate function is examined using switch and case statements which define the functionality for AND, OR and the other gates. This method does not incur an overhead in CPU time as the fault-free gates are examined in the switch-case statements. No extra gates are needed and no added events occur. But one shortcoming is that it requires a separate evaluation procedure for the faulty gates which is more complex than that for fault-free gates.

Table 10 shows the results of this experiment. The speedup ratio averages around 1.23.

function index	gate type
1	AND
2	NAND
3	OR
4	NOR
5	XOR
6	XNOR
7	inverter
8	buffer
9	D flip-flop
20 & above	reserved for faulty gates

Fig. 6: Coding index

TABLE 10: Functional fault injection for HOPE compared to PROOFS [6]

name	CPU time (secs)		speedup ratio
	PROOFS	HOPE _f	
s298	0.6	0.5	1.20
s344	0.4	0.3	1.33
s382	11.0	8.8	1.25
s444	12.2	10.3	1.18
s526	9.0	8.6	1.05
s641	0.7	0.7	1.00
s713	0.8	0.7	1.14
s820	5.0	3.9	1.28
s832	4.7	3.7	1.27
s953	1.1	0.9	1.22
s1238	3.2	2.5	1.28
s1423	2.3	2.0	1.15
s1488	12.3	10.8	1.14
s1494	11.4	10.0	1.14
s5378	32.5	27.9	1.16
s35932	110.6	78.9	1.40
average			1.20

C. Static and dynamic fault ordering methods:

Static fault ordering:

Instead of going from in a regular order from PIs to Pos, the static fault ordering process suggests that non-stem faults in each FFR be grouped first and the rest be grouped depth-wise starting from POs. The fault ordering is performed only once during preprocessing.

Dynamic Fault Ordering:

The activity of some faults is higher than the others. The main reason for that are the potentially detected faults. If a flip-flop has a potentially detected fault, then the value, X will be propagated to a PO. As X propagates to the PO, it is likely that it will also propagate to a PPO. This implies that a potentially detected fault is likely to remain potentially detected in the following time frames, hence making them highly active faults.

TABLE 10: Number of events for HOPE with static and dynamic fault ordering compared to PROOFS [6]

name	average no. of events per pattern				reduction ratio		
	PROOFS	HOPE _S	HOPE _D	HOPE _{SD}	HOPE _S	HOPE _D	HOPE _{SD}
s298	196	191	219	217	1.03	0.89	0.90
s344	257	249	268	264	1.03	0.96	0.97
s382	249	251	232	232	0.99	1.07	1.07
s444	406	363	345	344	1.12	1.18	1.18
s526	776	699	618	593	1.11	1.26	1.31
s641	312	310	311	321	1.01	1.00	0.97
s713	428	403	438	430	1.06	0.98	1.00
s820	778	896	593	605	0.87	1.31	1.29
s832	830	920	604	616	0.90	1.37	1.35
s953	4296	4521	4274	4513	0.95	1.01	0.95
s1238	396	398	397	398	0.99	1.00	0.99
s1423	3656	3276	3653	3366	1.12	1.00	1.09
s1488	1597	1579	1010	1016	1.01	1.58	1.57
s1494	1856	1858	1123	1131	1.00	1.65	1.64
s5378	5615	5129	4475	4260	1.09	1.25	1.32
s35932	74643	69911	59869	54748	1.07	1.25	1.36
average	6018	5685	4902	4566	1.02	1.17	1.19

TABLE 10: CPU times for HOPE with static and dynamic fault ordering compared to PROOFS [6]

name	CPU time (secs)				speedup ratio		
	PROOFS	HOPE _S	HOPE _D	HOPE _{SD}	HOPE _S	HOPE _D	HOPE _{SD}
s298	0.6	0.6	0.6	0.6	1.00	1.00	1.00
s344	0.4	0.4	0.4	0.4	1.00	1.00	1.00
s382	11.0	11.0	9.9	9.9	1.00	1.11	1.11
s444	12.2	11.2	11.0	10.8	1.09	1.11	1.13
s526	9.0	8.0	7.7	7.4	1.13	1.17	1.22
s641	0.7	0.7	0.7	0.7	1.00	1.00	1.00
s713	0.8	0.8	0.8	0.8	1.00	1.00	1.00
s820	5.0	5.2	4.2	4.3	0.96	1.19	1.16
s832	4.7	4.8	4.0	4.1	0.98	1.18	1.15
s953	1.1	1.1	1.0	1.1	1.00	1.10	1.00
s1238	3.2	3.2	3.1	3.1	1.00	1.03	1.03
s1423	2.3	2.2	2.2	2.1	1.05	1.05	1.10
s1488	12.3	12.2	9.1	9.2	1.01	1.35	1.34
s1494	11.4	11.4	8.2	8.2	1.00	1.39	1.39
s5378	32.5	28.7	27.5	26.4	1.13	1.18	1.23
s35932	110.6	105.1	98.7	94.1	1.05	1.12	1.18
average					1.03	1.12	1.13

The technique being attempted here is that if the high activity faults are grouped together, it is more likely that the propagation paths of the faults overlap with each other, thus reducing the overall gate evaluation time.

So, we make two groups—Group A contains faults that have not been potentially detected so far and Group B contains faults which have been potentially detected at least once. If a fault in Group A becomes potentially detected, it is moved to Group B. But this occurs at most once in the entire process, so the overhead is not much.

Tables 9 and 10 show the performances of static, dynamic and a combination of both heuristics in terms of no. of events per pattern and CPU time consumed. The combination proves to be marginally better.

The overall performance comparison of PROOFS and HOPE is shown below in Table 11. HOPE proves to be better than PROOFS in all but one case where it is just as good.

TABLE 11: Overall Performance of HOPE compared to PROOFS [6]

Name	CPU Time (sec)		Speedup ratio PROOFS/HOPE
	PROOFS	HOPE	
s298	0.6	0.5	1.20
s344	0.4	0.4	1.00
s382	11.0	8.4	1.31
s444	12.2	8.8	1.39
s526	9.0	5.6	1.61
s641	0.7	0.6	1.17
s713	0.8	0.6	1.33
s820	5.0	2.2	2.27
s832	4.7	2.1	2.24
s953	1.1	0.7	1.57
s1238	3.2	1.9	1.69
s1423	2.3	1.7	1.35
s1488	12.3	6.2	1.98
s1494	11.4	5.2	2.19
s5378	32.5	21.3	1.53
s35932	110.6	64.1	1.73

The table below is a result of combining overall performance comparisons in [5] and [6]. It is to be noted that PROOFS_O corresponds to PROOFS algorithm in [3] and PROOFS corresponds to PROOFS algorithm in [7]. Similarly, the difference in PARIS and PARIS_O is that PARIS_O has been demonstrated in [4] and it varies from PARIS because it incorporates circular partitioning while the latter does not. The table below is a result of combining overall performance comparisons in [5] and [6]. We can see that HOPE gives better performance in all but 4 cases when compared with VISION.

TABLE 12: Performance of HOPE w.r.t. PROOFS, PARIS and VISION [5-6]

Name	CPU time (sec)						Speedup ratio
	PROOFS_O	PROOFS	PARIS_O	PARIS	VISION	HOPE	VISION/HOPE
s298	1.1	0.6	1.0	0.9	0.7	0.5	1.4
s344	0.8	0.4	0.8	0.7	0.6	0.4	1.5
s382	17.2	11.0	7.9	9.0	7.1	8.4	0.85
s444	20.8	12.2	15.5	16.0	10.0	8.8	1.14
s526	11.0	9.0	11.5	11.5	7.2	5.6	1.29
s641	1.4	0.7	0.6	0.8	0.6	0.6	1.00
s713	1.4	0.8	0.8	1.0	0.7	0.6	1.17
s820	5.1	5.0	3.5	4.1	2.5	2.2	1.14
s832	4.8	4.7	3.4	4.8	3.1	2.1	1.48
s953	0.5	1.1	0.4	0.6	0.5	0.7	0.71
s1238	4.9	3.2	0.6	0.8	0.8	1.9	0.42
s1423	3.8	2.3	2.1	3.3	2.3	1.7	1.35
s1488	12.7	12.3	10.1	17.9	7.3	6.2	1.18
s1494	10.7	11.4	12.8	16.6	10.5	5.2	2.02
s5378	51.5	32.5	10.3	10.9	11.5	21.3	0.54
s35932	173.1	110.6	1260.0	1180.1	353.9	64.1	5.52

VII. CONCLUSIONS

Different sequential parallel fault simulation algorithms have been studied. Their performance variations with the application of different heuristics have been observed. Over all, HOPE algorithm has managed to achieve a good performance benefit with its new heuristics although some anomalies need to be further investigated.

VIII. REFERENCES

- [1] Digital Logic Testing and Simulation”, Alexander Miczo, *Harper & Row Publishers*, 1st edn. , 1986
- [2] A. K. Varshney, B. Vinnekota, E. Skuldt, B. Keller, “High Performance Parallel Fault Simulation”, *Intl. Conf. on Computer Design*, pp. 308-313, sep. 2001
- [3] W. Cheng, J. H. Patel, “PROOFS: A super-fast fault simulator for sequential circuits”, *Proc. Euro. Design Automation. Conf.*, pp 475-479, Mar1990
- [4] N. Gouders, R. Kaibel, “PARIS: A parallel pattern fault simulator for synchronous sequential circuits”, *Proc. Intl. Conf. Computer-Aided Design*, pp. 542-545, Oct 1991
- [5] R. Nair, D. S. Ha, “VISION: An efficient parallel pattern fault simulator for synchronous sequential circuits”, *Proc. IEEE VLSI Test Symp.* , pp. 221-226, Apr 1995

- [6] H. K. Lee, D. S. Ha, "HOPE: An efficient parallel fault simulator for synchronous sequential circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, Sep 1996
- [7] T. M. Niermann, W. T. Cheng; J. H. Patel, "PROOFS: a fast, memory-efficient sequential circuit fault simulator", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 2, pp. 198 – 207, Feb 1992
- [8] M. Mojtahedi, W. Geisselhardt, "PARIS: New methods for parallel pattern fast fault simulation for synchronous sequential circuits", *Proc. Intl. conf. Computer-aided Design*, pp. 546-549, Nov 1993
- [9] W. T. Cheng, M. L. Yu, "Differential fault simulation-A fast method using minimal memory", *Proc. Design Automation conf.*, pp. 424-428, Jun1989