MOSAIC : a Multiple–strategy Oriented Sequential ATPG for Integrated Circuits

A. Dargelas♦♣ , C. Gauthron ♣ and Y. Bertrand ♦

dargel_a@compass.fr, chrisg@compass.fr, bertrand@lirmm.fr

♦ LIRMM (Laboratoire d'Informatique, Robotique et Micro-électronique de Montpellier), UMR 9928, Univ. Montpellier II / CNRS, 161, rue Ada, 34392 Montpellier Cedex 5 FRANCE
♣ COMPASS Design Automation 505 Route des Lucioles 06560 Sophia–Antipolis FRANCE

Abstract

The paper proposes a novel approach in an attempt to solve the test problem for sequential circuits. Up until now, most of the classical test pattern techniques use a number of algorithms in several passes to detect faults. Our so–called Multiple Strategy Approach takes into account the existing techniques and algorithms, (improvements are proposed for some of them) and at each step selects the strategy that is best adapted to catch the targeted faults. This work has been done with a focus on designing a real industrial ATPG, able to handle real circuits consisting of several hundreds of thousands of gates.

I / Introduction

During the past decade much academic work has been done in an attempt to solve the problem of Automatic Test Pattern Generation (ATPG) for sequential circuits [Mar86, Che88a, Che88b, MaD88, Gou91, Lee91, Nie91, Ono91, Kel93]. More recently several industrial tools (HITEC, GENTEST, ...) have been developed for inclusion in CAD suites. Two main techniques are classically used to generate test vectors for circuits, namely, the deterministic approach and the simulation–based approach. The simulation–based approach may use either random or genetic [Saa94, Pri94, Rud95] generation. In some cases both techniques are found to be combined in the same tool, in others they are separated.

It seems that no single technique gives the best results for all the test cases. Taking this fact into account, we decided to develop a new Sequential–ATPG, the so–called MOSAIC tool, which aims at being able to cope with industrial circuits. The designs targeted are real designs, which may or may not be provided with partial scan. This means that some sequential elements are not included in scan chains. In addition to this, depending on the circuit under consideration, the sequential elements (FFs) may or may not be provided with reset facilities. Normally, an initialization sequence is given by the designer to set the circuit into its reset state, but there are some exceptions that we have to deal with. As a consequence we do not assume a reset state for these sequential elements.

The fundamental concepts on which our approach is based originates mainly from the basic works of Fujiwara and Shimono [Fuj83] on the FAN algorithm and Gouder and Kaibel [Gou91] on the CONSEQUENT model. Different strategies we have used are derived from various papers : [Che88a, Sch88, Sch89, Nie91, Lee91, Ono91, Kel93]. Section II describes the techniques commonly used in test generation and explains the way we have modified these techniques in MOSAIC. Section III exposes our implementation of the Multiple Strategy technique derived from [Min89], and Section IV describes our approach for sequential circuits. Section V discusses Strategy choices, then, Section VI presents the MOSAIC results obtained on ISCAS89 benchmark circuits. Lastly, Section VII gives conclusions and proposes some future extensions.

II / Basic Techniques

The various techniques presented in this section are not limited to combinational circuits. They are used with the iterative array representation of sequ. circuits [Abr90].

II.1 / Value system

Since [Rot66] and the famous 5–Valued D–algorithm a large number of value systems have been introduced in an attempt to design a complete algorithm, in particular the 9–Valued algorithm [Mut76] and the Split Model [Che88b]. More recently Gouders and Kaibel [Gou91] have introduced the so–called "bit–oriented coding for the CONSEQUENT circuit model" that allows decision inversion in sequential circuits without violating the

A. Dargelas is supported by ANRT grant n° 625/94

ED&TC '97 on CD-ROM

completeness of the search process, as HITEC [Nie91] does.

The *256–Valued system* used in MOSAIC is derived from that proposed by Gouders and Kaibel. The basic 16–Valued alphabet is as follow: 0 , 1 , Z, x(01) ... x(01ZU) , U. In this alphabet, 0 , 1 and Z represent the 3–state logic, x(...) represents the unspecified value (which can be more or less specified) and U represents the unknown and non assignable value. For the bit–oriented coding we take 0 , 1 , Z , U as bit values. Value x(...) can take every 2 by 2, 3 by 3 or 4 by 4 combination inside the set of bit values. For example, x(01) can be interpreted as follow: the value can take the values 0 or 1 by specification. Table 1 shows the coding of the value system:

values/coding	Ubit	Zbit	1bit	0bit
0	0	0	0	1
1	0	0	1	0
Z	0	1	0	0
U	1	0	0	0
x(01)	0	0	1	1
...	...	...	...	...
x(01ZU)	1	1	1	1
x(Φ)	0	0	0	0

Table 1: Value system encoding.

The *256–Valued system* is a Split–Model like system [Che88b], having two machine representations within a single byte, but without the relation component of the original model. The combination of the two 16–Valued systems (good circuit and faulty circuit value systems) gives the final system. The advantages of this system are multiple. First, the precision of the unspecified values is increased by the fact that x(...) is never totally unspecified. The value of x(...) is specified during the generation and reaches a completely specified value at the end. This makes it possible to detect conflicts earlier, and guides heuristics such as Multiple Backtrace [Fuj83] more powerfully. This value system makes it possible to deal with the non resettable sequential circuit problem owing to the U value which represents the value present on the output of a flip–flop after power–up, when no reset is available and no value can be justified for this flip–flop. This value cannot be replaced by any other, thus preventing the prospect of forbidden search–space branches.

II.2 / Multiple Backtrace

The FAN algorithm of Fujiwara and Shimono [Fuj83] includes the Multiple–Backtrace concept. This procedure allows the simultaneous satisfaction of a set of objectives instead of a single one, as Single Backtrace does. The Multiple–Backtrace process is used in MOSAIC with modifications that allow it to fit our value system. Every primitive has a Multiple–Backtrace inference method to fill the 6 counters n0g, n1g, nZg, n0b, n1b, and nZb (g for good machine, b for bad). This defines a new objective by the 7–uplet (objective_gate, n0g, n1g, nZg, n0b, n1b, nZb). Our Multiple Backtrace does not stop on a head line as in the original version, since this concept has only a fringing effect for huge looping circuits. Throughout this paper it should be taken into consideration that the Multiple Backtrace is made through all the allocated time frames from objectives towards (i) primary inputs (PIs) in all time frames and (ii) flip–flops (FFs) in the lowest allocated time frame. These PIs and FFs constitute the set of decision nodes. Our simplified implementation of Multiple Backtrace preserves the advantage of the original Multiple Backtrace against Single Backtrace, that is to say the concurrent search of the best decision to take. In figure 1, the efficiency of the 256–Valued system is illustrated. With a 9–Valued system for example, all the x(...) values should be replaced with a X value and no differences between x(0Z) and x(1Z) should be seen. In this case an incorrect choice for the decision to take can occur. In the same case, with the 256–Valued system, the correct decisions will be taken without any backtrack. In the figure, the circuit values are given at the initial state, the decision values are not implied.

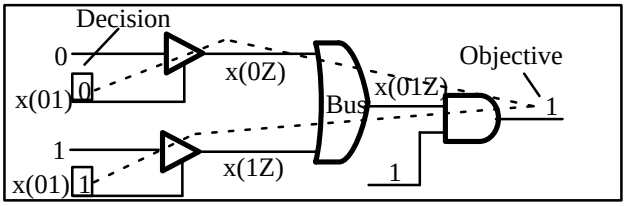

Figure 1: 256–Valued system and Multiple Backtrace.

II.3 / Basic Propagation technique

The propagation is based on a new method which takes the maximum advantage of our 256–Valued system. Each time the procedure wants to propagate a fault, it checks where the fault has been propagated in the preceding step (the fault site is the seed of this propagation). Then, from amongst the candidates, i.e., the set of gates with a fault effect on their inputs and an *unspecified* output, the procedure selects the gate which is the most easily observable with compatible x(...) values on other inputs (enabling propagation). Following this a propagation decision is taken and all the pending implications are calculated. The alternatives are pushed into the stack in

case of backtracking. Figure 2 gives an example of the basic technique:

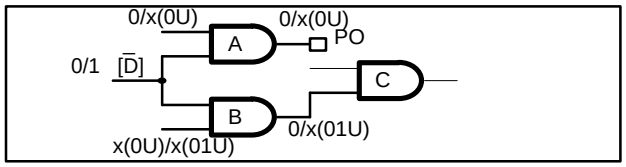

Figure 2: 256–Valued system and fault propagation.

The fault effect has reached A and B, and A is easier to observe than B. With a classical value system, every x value represents a ”don’t care” value and both paths are interpreted as being equivalent by X–path check [Abr90]. Therefore, A is chosen to be the next gate to propagate the fault. With our value system, we can see that A cannot propagate the fault, so the only choice is B. The various strategies for fault propagation are discussed later. This technique can be seen as a kind of targeted D propagation.

II.4 / Basic Justification technique

a) Backtracking techniques

As previously presented, the Multiple Backtrace is the main heuristic used in the decision tree process. The justification is processed as long as there remains an objective in the J–Frontier [Abr90]. The mechanism used is similar to that described in [Fuj83].

In the context of the bit–oriented coding of the value system, the backtracking process (the testing of alternative decisions), can be of two kinds: (i) inverting the previous decision, (ii) unseting the corresponding failing bit [Gou91]. The first technique consists of changing a 0 (respectively 1) to a 1 (0) when 0 (1) has failed to justify objectives. The other technique in its original version [Gou91] consists of unseting the bit 0 (1) at the decision node (Pseudo–Primary Inputs) when previous implication failed and implicating the changes. The first backtracking technique (inverting the decision made) suffers from the commonly known *over specification* problem [Gou91]. In this paper we will call it the *Binary Inversion Technique*. The second technique (unseting failing bits) solves this problem because it does not force values that are not needed to justify objectives. It just bounds the search space by specifying the unspecified values and acts as a marker for the backtrace. We will call it the *Bit Inversion Technique*. One drawback of the method is the fact that under some conditions the process does not rapidly converge. For example, if the current conflicting state is [0,0] and the only non conflicting state is [1,1], we have for both techniques the worst cases for the backtracking processes :

Notations:

. identification number of the decision pushed onto the decision tree : i

. Backtrace + Implication Operator : $=D_i>$

. Conflict + Backtrack Operator : $\#D_i>$

1rst technique: [0,x(01U)] $=D_1>$ [0,0] $\#D_1>$ [0,1] $\#D_0>$ [1,x(01U)] $=D_1>$ [1,0] $\#D_1>$ [1,1]

2nd technique: [0,x(01U)] $=D_1>$ [0,0] $\#D_1>$ [0,x(1U)] $=D_2>$ [0,1] $\#D_2>$ [0,x(0U)] $\#D_2>$ $\#D_1>$ $\#D_0>$ [x(1U),x(01U)] $=D_1>$ [x(1U),0] $=D_2>$ [1,0] $\#D_2>$ [x(0U),0] $\#D_2>$ $\#D_1>$ [x(1U),x(1U)] $=D_2>$ [1,x(1U)] $=D_3>$ [1,1].

In the worst case, the first technique needs 3 backtracks and 1 decision to be pushed onto the decision tree to achieve the result. The second one needs 8 backtracks and 3 decisions.

b) Decision Ordering techniques

Our investigations detected another problem occurring in Sequential State justification: this is the relative weight that is given to the Decisions done on the Primary Inputs and the State Decisions done on the Pseudo Primary Inputs. This problem occurs because we use the Multiple Backtrace instead of the Single Backtrace which gives a single decision to take. The Multiple Backtrace gives us a set of decisions, with some weights that do not take into account the nature of the decision node: PI or PPI. The testability measures are not sufficiently efficient to deal with this problem. As a result, we made a preliminary study on the ISCAS89 benchmarks and found out that the choice of whether to push the PI Decisions or the PPI Decisions first can drastically improve or degrade the performances. Two efficient ordering–and–backtracking techniques have been drawn from this study: *PI first, short popping* and *PPI first, long popping*. Figure 3 illustrates these two techniques used in reverse processing for state justification.

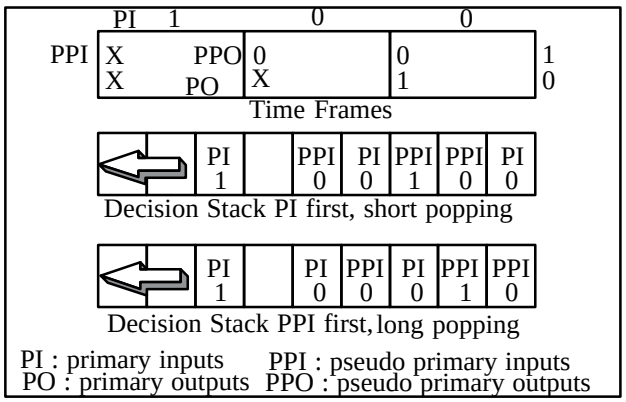

Figure 3 : Ordering and backtracking techniques

For *PI first, short popping*, the Multiple Backtrace proposes a set of Decisions to take, with some weight, so we take the PI decision with the highest weight as the current decision. Following this, when no more PI decisions are to be taken, we push the PPI decisions. When the current time frame is completely justified, a check is made on the prevention of state looping by state cover comparison [Nie91] before stepping to a previous time frame. If a backtrack occurs, the decisions are popped in the reverse order they were pushed.

For *PPI first, long popping*, we take the PPI decision with the highest weight as the current decision. Then, when no more PPI decisions are to be taken, a check on the prevention of state looping is done before we push the PI decisions. When the current time frame is completely justified, a step into the previous time frame can be made. If a backtrack occurs, the PI decisions are popped without considering their alternatives as long as we met a PPI decision that will be considered for backtracking.

c) Illegal State Learning

Illegal state learning [Nie91] is done for each fault during reverse time processing. It is available only during the current fault test generation. Under the condition that no backtrack has occurred due to the faulty state machine, the illegal states learned are kept for all the faults.

These two *Backtracking Techniques* and *Decision Ordering Techniques* constitute the first set of Multiple Strategies. Next paragraph discuses more general Strategies.

III / Multiple Strategy

Up to the present time the evidence suggestes that no particular test generation strategy has been recognized to be universally the best for all the faults in any circuit. One of the challenges for present–day sequential ATPGs is to have the ability to switch easily from one strategy to another, depending on the testability degree of the targeted fault in a given circuit. In particular, it has been pointed out by Min [Min89] that during the backtrace process it may be useful (in terms of number of backtracks) to use a combination of various search strategies rather than a single one. The advantage of this concept is that a hard–to–detect fault can be detected by a particular strategy well suited for this particular fault. A badly suited strategy requires a large number of backtracks to converge to a solution, a better strategy will result in more easy convergence.

MOSAIC allows the use of various alternative strategies and a counter of aborted faults provides information about the historical efficiency of each strategy. From this information, the best strategy at any time (i.e., the one with the smallest counter) is used in the attempt to catch the current fault. This approach is similar to that used by [Min89] and [Wai90], with certain differences. In the Multiple Strategy approach of Min, Single Backtrace was performed, so that strategies were oriented by a classification based on objective satisfaction ordering. In MOSAIC we perform Multiple Backtrace where no priority is given to objectives. Therefore we made a classification of orthogonal strategies based on the criterion of generation phase ordering, i.e. the ordering of the propagation and justification phases. The choice of the best strategy to apply at any time is an auto–adaptive process. In practice, after an adaptative phase (during which the strategy choice is arbitrarily done) the best suited strategy is first applied to the current fault. For a small set of faults, several strategies are tried before classifying them as either redundant or aborted.

The set–up is defined in this paper as the sensitization of a given stuck–at value. The main strategies are listed here: *a) Set–up / propagation first / justification last*, *b) Set–up / interlaced propagation and justification*.

For example, circuit c6288 which is a multiplier, has a huge number of paths. Attempting to sensitize an entire path from the fault site to a PO and then justify it in one shot, such as *Set–up / propagation first / justification last* strategy will probably fail. Instead, by applying a small propagation then immediately justifying it, the next propagation step is constrained to be done correctly. This is the reason why the *Set–up / interlaced propagation and justification* is efficient in that case.

The *Set–up / propagation first / justification last* strategy can be interpreted in terms of human reasoning as a

deductive process where a solution is first searched for based on much hypothesis or assumption, and then when found, a second phase of reasoning deals with the verification of hypothesis. This was the high–level mechanism of the D–Algorithm [Roth66].

The *Set–up / interlaced propagation and justification* can be interpreted as a small step–by–step deductive search, where a part of the solution is found and corresponding hypothesis immediately verified.

We now examine these two strategies in detail. Figure 4 illustrates the Multiple–Strategy Switching in a single time frame.

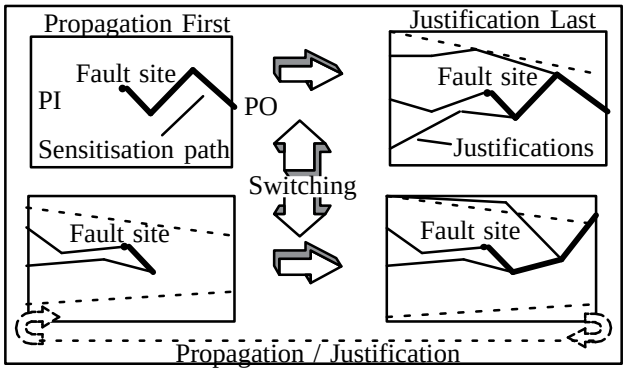

Figure 4 : Switching Strategies

III.1 / Set–up / propagation first / justification last strategy

– The Set–up is done by implying the stuck–at value. This fills the J–frontier.

– A fault cone flag is set, in time frame 0, from fault site to primary outputs and sequential elements inputs. The *propagation engine* is used until fault effect reaches either a primary output or a sequential element input. In the second case, an additional positive time frame is allocated, fault cone is updated and propagation is launched again in this new time frame. At the end of this phase we obtain the set of all the time frames allocated by propagation, i.e. from t_0 (the set–up time frame) to t_n (the first time frame for which the fault appears on a PO). During the propagation, the breaking of state looping is ensured by state cover comparison whenever a new time frame is allocated.

– The *justification engine* tries to justify all objectives present in all time frames allocated by propagation, by assigning the PIs of all these time frames (t_0...t_n) and the PPIs of the first time frame (t_0). When the J–frontier contains only sequential elements in the first time frame, with identical values on good and bad machines (no fault effect on PPIs at time frame 0), then a good machine state justification is tried. A negative time frame is allocated and the good machine values of the sequential elements are implied in the negative time frame. The basic justification technique and negative time frame allocation are applied until no more gates remain in J–frontier. At this step we have n positive time frames (t_0...t_n) for fault propagation and m negative time frames (t_{-m}...t_{-1}) for good machine state justification. For negative time frames we copy the good PI values on the bad PI values and simulate. From this, two conclusions are possible:

. Good and Bad values at PPOs in time frame –1 are identical. A valid justification sequence has been found.

. Fault effect has reached PPOs in time frame –1 [Nie91]. We did not find a justification sequence for the corresponding propagation sequence but a self initialization sequence. As a result, we re–propagate with this new knowledge (we keep the previous propagation sequence in case of backtracking).Illegal state learning [Nie91] is done in forward reverse time generation, for each fault. It is available only during the current fault test generation.

III.2 / Set–up / interlaced propagation and justification

– This Strategy applies the same techniques as the previous one, but the basic propagation technique is performed one time in the first time frame. After this, the justification technique is performed and if needed, the good state justification is also performed in reverse time processing.

– Then the basic propagation technique is launched again toward Primary outputs or State elements inputs. In the second case a new time frame is allocated for propagation.

– This mechanism loops until the fault reaches output and all the time frames are justified.

IV / Sequential Technique

Since the sequential test generation problem was introduced, three *time strategies* (based on the iterative array model) have been proposed in this paper to make the combinational technique applicable for sequential circuits: *Forward Only time processing, Backward Only time processing, Forward Reverse time processing.* Each strategy has advantages and drawbacks, and are separately discussed in [Kel93]. In MOSAIC, we choose to switch

between two of them: *Forward Reverse time processing* and *Forward Only time processing*. The first one is necessary in order to have a complete algorithm and starts from an unknown state. The second uses the fault simulation knowledge and starts from a known state (both fault free and faulty state are used). It helps to detect faults which should be aborted by the Forward Reverse time processing by starting at an unknown state.

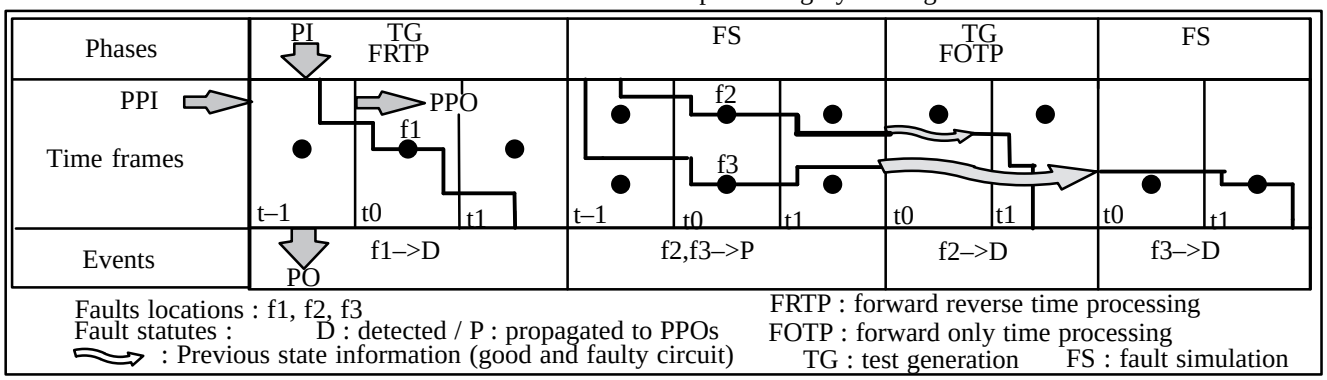

Figure 5: Sequential test generation overview.

The rule to choose between the two time strategies is simple, if there is a previous state information available for a fault [Ono91] [Kel93], choose Forward Only time processing for this fault, choose Forward Reverse time processing elsewhere. A general overview of the test generation process is given in figure 5. The previous state information problem is solved by keeping in memory (after every fault simulation) the list of the FFs reached by each fault effect, with the associated bad value and the fault–free circuit state. In figure 5, we can see that fault f1 is detected using 3 time frames, time frame t_0 is the set–up time frame (where fault is activated), time frame t_1 is the time frame where the observation of the fault effect is allowed, and time frame t_{-1} is the time frame where all the states are justified (values on PPI in time frame t_{-1} are x(...) or U). Then during the fault simulation phase applied on f1 test sequence, some previous state information is learned for faults f2 and f3. Fault f2 is chosen to be tried by Forward Only time strategy. For fault f3, previous state information is kept and used for the next fault simulation phase. In this example, an explicit initialization sequence is needed for fault f1. On the other hand, the initialization sequences for faults f2 and f3 are implicitly contained in previous initialization and propagation sub–sequences.

In practice a large number of faults will have a *previous state information* after fault simulation, so we choose to try the faults which have the greatest activity (the greatest number of FFs reached) first. This is done until one is detected. We can limit the maximum number of tries before switching again in Forward Reverse Time Processing. This heuristic reduces the global number of vectors generated.

V / Strategy Choice

Backtracking Technique	Decision Ordering Technique	Search Strategy	Time Strategy
Binary Inversion	PPI first, long pop	Prop. First, Just Last	Forward–Only
Bit Invertion	PI first, short pop	Interlaced Prop–Just	Forward–Reverse

Table 2: Summary of the different strategies

Table 2 summaries the various strategies available in MOSAIC. In practice, the two Search Strategies and Time Strategies are always used for all the faults with the switching mechanisms explained in previous sections. We started a study on the influence of the Backtracking Technique and the Decision Ordering Technique. The first results show that the couples (Binary Inversion ; PPI first,long popping) and (Bit Inversion ; PI first, short popping) are the most useful on the ISCAS89 benchmarks.

Back. Tech	Deci. Ord.	Search Str.	Time Strat.	# Detected	# Vector	CPU time
Bin Inv	PPI first	Both	Both	367	1842	9.7min
Bin Inv	PPI first	Both	Forw–Rev	367	2474	11.2min
Bit Inv	PI first	Both	Both	285	809	17.6min
Bin Inv	PPI first	Prop. First	Both	364	1693	4.7min
Bin Inv	PPI first	Interlaced	Both	347	1312	8.2min

Table 3: s400 experiments with different strategies

For example, s382, s400, s420, s444, s820, s834, s1488, s1494 are treated powerfully with the first one, while s208, s298, s344, s526, s1196, s1238, s13207, s1423, s15850 are treated powerfully by the second one. Some further investigations need to be done. Table 3 gives an experiment conduced on s400 using an UltraSparc 175 Mhz to illustrate the effectiveness of the different strategies. The conditions are identical for each experiment: 1000 backtracks max. per fault, 500 time frames max. for forward or reverse processing, 1 pass on the fault list. The best configuration of strategy is the first one for this particular circuit. The second one generates more vectors because it does not benefit from the previous state knowledge and the Forward–Only strategy. The other combinations achieve less fault coverage in that particular case so they are not interesting.

The efficiency of MOSAIC has been experimented by generating tests for several classical benchmark circuits. Test generation results are given in table 4. For comparison, a compilation of the results for HITEC [Nie91, Rud95] are also reported on these tables. MOSAIC achieves a better fault coverage with less vectors than HITEC does. The Bold rows mark the benchmarks for which we achieve a better fault coverage than HITEC, or at equal fault coverage we generate less vectors.

Circuits s382, s400, s444, s526, s1423, s5378, s35932 are the illustration of the ability of MOSAIC to reach a higher fault coverage than HITEC with less CPU effort. Many circuits illustrate the fact that MOSAIC generates more compact test sequences than HITEC.

VI / Results

		MOSAIC (SPARC20–70MHz)				HITEC [Rud95] (SPARC 20)			
Circuit	Faults	Time	Untest.	Vectors	Detect.	Detect.	Vectors	Untest.	Time
s208	**215**	**15s**	**53**	**133**	**137**	137	184	78	29s
s298	308	48.7s	40	343	265	265	281	26	32.3m
s344	**342**	**9.4s**	**5**	**138**	**327**	324	139	11	17.6m
s349	**350**	**3.1m**	**9**	**97**	**334**	334	111	13	11.5m
s382	**399**	**39m**	**11**	**3462**	**358**	301	1665	10	3.05h
s386	384	6.05m	44	308	314	314	275	70	11.2s
s400	**424**	**15m**	**16**	**1842**	**367**	342	1669	17	2.31h
s420	**430**	**9.2m**	**212**	**147**	**179**	179	218	251	45.3m
s444	**474**	**17.1m**	**25**	**1165**	**400**	378	2060	25	2.84h
s526n	**553**	**11.4m**	**19**	**679**	**379**	–	–	–	–
s526	**555**	**10.1m**	**18**	**861**	**407**	346	680	22	10.7h
s641	467	49s	36	225	404	404	184	63	6.44s
s713	581	71s	76	225	476	476	190	105	9.95s
s820	850	6.8m	30	1029	814	814	1113	36	1.01m
s832	**870**	**19.1m**	**33**	**1077**	**817**	817	1181	53	8.72m
s838	**857**	**29.7m**	**460**	**177**	**244**	–	–	–	–
s953	**1079**	**36.2s**	**976**	**24**	**89**	89	41	990	15.6m
s1196	**1242**	**11.4s**	**3**	**323**	**1239**	1239	460	3	6.34s
s1238	**1355**	**17.6s**	**72**	**343**	**1283**	1283	469	72	9.97s
s1423	**1515**	**18.2m**	**9**	**301**	**1049**	776	177	14	27.5h
s1488	1486	33.1m	42	979	1429	1444	1138	41	31.0m
s1494	1506	33.8m	42	1087	1452	1453	1178	52	18.3m
s5378	**4603**	**1.2h**	**156**	**628**	**3337**	3238	941	225	36.3h
s9234	**6927**	**0.1s**	**6909**	**4**	**18**	18	24	3916	2.08m
s13207	**9815**	**1.3h**	**8960**	**218**	**626**	–	–	–	–
s15850	**11727**	**4.1m**	**11631**	**9**	**86**	86	32	11403	28.4m
s35932	**39094**	**5.28h**	**3984**	**632**	**35002**	34898	439	3984	8.07h

HITEC results on circuits s208, s420, s510, s953, s9234, s15850 are obtained on a SPARC 2.

Table 4: Iscas 89 benchmark results

VII / Conclusions

In this paper we describe a new test generator (MOSAIC) that was developed to cope with the test of real industrial circuits. The circuits targeted may (i) be very large, (ii) contain more or less sequentiality (iii) use 3–state components and (iv) use unresettable flip–flops. A multiple strategy approach has been chosen to profit from the various existing techniques by switching from one to another according to the targeted fault in a given context. Several strategies may be employed to deal with sequential circuits in the framework of time array model. Results are

presented on the complete set of ISCAS89 benchmarks. Without using any learning techniques [Sch89], which are known to be inefficient on huge circuits (due to memory need), we achieve a high fault coverage in a compact test length. These good results are due to the following concepts used in MOSAIC. First, we introduce a new value system, the 256–Valued model, that gives increased accuracy for defining unspecified values. Second, we choose to use two different backtracking mechanisms and two decision orderings. Finally, we give MOSAIC the ability to switch between various Search Strategies and Time Strategies, which is the best way to detect the greatest number of different fault types. Multiple Backtrace technique have been extended to the sequential domain largely thanks to the different decision orderings. Impressive results are obtained compared to HITEC in terms of fault coverage increase, test length and CPU time reduction. Future extensions of this work will concern the improvement of choice between the large set of strategies we have and preprocessing for untestable faults identification.

References:

[Abr90] M. Abramovici, M.A. Breuer, and A.D. Friedman, ”Digital Systems testing and Testable Design”, Computer Science Press, 1990.

[Che88a] W.T. Cheng, ”The BACK algorithm for sequential test generation,” Proc. Int. Conf. Computer Design , pp. 66–69, 1988.

[Che88b] W.T. Cheng, ”Split circuit model for test generation,” Proc. 25–th Design Automation Conf., pp. 96–101, 1988.

[Fuj83] H. Fujiwara and T. Shimono, ”On the acceleration of test generation algorithms”, IEEE Trans. on Computers, Vol. C–32, n° 12, pp. 1137–1144, December 1983.

[Gou91] N. Gouders and R. Kaibel ”Test generation techniques for sequential circuits” Proc. IEEE VLSI Test Symposium, pp. 221–226, 1991.

[Kel93] T.P. Kelsey, K.K. Saluja, and S.Y. Lee, ”An efficient algorithm for sequential circuit test generation” IEEE Trans. on Computers, Vol. 42, n° 11, pp. 1361–1371, November 1993.

[Lee91] D.H. Lee and S.M. Reddy, ”A New Test Generation Method for Sequential Circuits” Proc. Int. Conf. on Computer–Aided Design, pp. 446–449, 1991.

[**MaD88]** H.–K.T. Ma, S. Devadas, A.R. Newton, and A. Sangiovanni–Vincentelli, ”Test generation for sequential circuits”, IEEE Trans. Computer–Aided Design, Vol. 7, N°10, pp. 1081–1093, October 1988.

[**Mar86]** R. Marlett, ”An effective test generation system for sequential circuits”, ” Proc. 23–th Design Automation Conf., pp. 250–256, 1986.

[Min89] H. B. Min and W. A. Rogers, ”Search Strategy Switching : An Alternative to Increased Backtracking”, Proc. Int. Test Conf., pp. 803–811, 1989.

[**Mut76]** P. Muth, ”A nine–valued circuits model for test generation”, IEEE Trans. on Computers, Vol. C–25, n° 6, pp. 630–636, June 1976.

[Nie91] T. Niermann and J.H. Patel, ”HITEC: A test generation package for sequential circuits”, Proc. European Conf. on Design Automation, Amsterdam, The Netherlands, pp. 214–218, February 1991.

[Ono91] T. Ono and M. Yoshida, ”A Test Generation Method For Sequential Circuits Based on Maximum Utilisation of Internal States” Proc. Int. Test Conf., pp. 75–82, 1991.

[Pri94] P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, ”An automatic test pattern generator for large sequential circuits based on genetic algorithms ” Proc. Int. Test Conf., pp. 240–249, 1994.

[Rot66] J.P. Roth, ”Diagnosis of automata failures: A calculus and a method”, IBM Journal Research and Development, Vol. 10, July 1966.

[Rud95] E.M. Rudnick and J.H. Patel, ”Combining deterministic and Genetic Approaches for sequential circuit test generation”, Proc. 32–th Design Automation Conf., pp. 183–188, 1995.

[Saa94] D.G. Saab, Y.G. Saab and J.A. Abraham, ”Iterative [simulation–based genetics+deterministic techniques] complete ATPG”, Proc. Int. Conf. on Computer–Aided Design, pp. 40–43, 1994.

[Sch88] M.H. Schulz, E. Trischler, and T.M. Sarfet, ”SOCRATES: A Highly Efficient Automatic Test pattern Generation System”, IEEE Transactions on Computer–Aided Design, Vol.7, n°1, January 1988.

[Sch89] M.H. Schulz and E. Auth, ”ESSENTIAL: an efficient self–learning test pattern generation algorithm for sequential circuits”, Int. Test Conf. , pp. 28–37, 1989.

[Wai90] J.A. Waicukauski, P. A. Shupe, D.J. Giramma, A. Matin, ”ATPG for Ultra–Large Structured Designs”, Proc. Int. Test Conf., pp. 44–51, 1990.