IMPROVED PATH RECOVERY IN PSEUDO FUNCTIONAL PATH DELAY TEST USING EXTENDED VALUE ALGEBRA

A Thesis

by

PRASENJIT BISWAS

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2016

Major Subject: Computer Engineering

ABSTRACT

Scan-based delay test achieves high fault coverage due to its improved controllability and observability. This is particularly important for our K Longest Paths Per Gate (KLPG) test approach, which has additional necessary assignments on the paths. At the same time, some percentage of the flip-flops in the circuit will not scan, increasing the difficulty in test generation. In particular, there is no direct control on the outputs of those non-scan cells. All the non-scan cells that cannot be initialized are considered “uncontrollable” in the test generation process. They behave like “black boxes” and, thus, may block a potential path propagation, resulting in path delay test coverage loss. It is common for the timing critical paths in a circuit to pass through nodes influenced by the non-scan cells. In our work, we have extended the traditional Boolean algebra by including the “uncontrolled” state as a legal logic state, so that we can improve path coverage. Many path pruning decisions can be taken much earlier and many of the lost paths due to uncontrollable non-scan cells can be recovered, increasing path coverage and potentially reducing average CPU time per path. We have extended the existing traditional algebra to an 11-value algebra: Zero(stable), One(stable), Unknown, Uncontrollable, Rise, Fall, Zero/Uncontrollable, One/Uncontrollable, Unknown/Uncontrollable, Rise/Uncontrollable, and Fall/Uncontrollable. The logic descriptions for the NOT, AND, NAND, OR, NOR, XOR, XNOR, PI, Buff, Mux, TSL, TSH, TSLI, TSHI, TIE1 and TIE0 cells in the ISCAS89 benchmark circuits have been extended to the 11-value truth table. With 10% non-scan flip-flops, improved path delay fault coverage has been observed in comparison to that with the traditional algebra. The greater the number of long paths we want to test; the greater the path recovery advantage we achieve using our algebra. Along with improved path recovery, we have been able to test a greater number of transition fault sites. In most cases, the average CPU time per path is also lower while using the 11-value algebra. The number of tested paths increased by an average of 1.9x for robust tests, and 2.2x for non-robust tests, for K=5 (five longest rising and five longest falling transition paths through each line in the circuit), using the eleven-value algebra in contrast to the traditional algebra. The transition fault coverage increased by an average of 70%. The improvement increased with higher K values. The CPU time using the extended algebra increased by an average of 20%. So the CPU time per path decreased by an average of 40%. In future work, the extended algebra can achieve better test coverage for memory intensive circuits, circuits with logic black boxes, third party IPs, and analog units.

DEDICATION

To the women in my life

Tania, Ratna, Rita, Nandita

&

Evans Library

ACKNOWLEDGEMENTS

I would like to express my sincerest gratitude and respect for my M.S. committee chair Dr. Duncan M. (Hank) Walker. I would like to thank him for his technical guidance, encouragement. He has been a source of immense support and patience throughout the course of the work. I am thankful to him for having the faith in me and providing me with all the support and suggestions for all my work.

I am grateful to my M.S. committee members Dr. Aakash Tyagi and Dr. Jiang Hu. I would like to extend my gratefulness for their encouragement and support.

I would also like to thank all the faculty and staff members in both the departments of “Computer Science and Engineering” and “Electrical and Computer Engineering” for making my academic experience with Texas A&M University so special, unique and full of joy.

I want to thank all my friends, colleagues and lab mates for making this journey so wonderful and enjoyable.

Finally, I would like to express my respect and gratitude to my parents for their unconditional love and support. In addition, a very special thanks to my wonderful wife for all her support and constant encouragement throughout the journey.

NOMENCLATURE

DFT	Design For Test
ATPG	Automatic Test Pattern Generation
CPU	Central Processing Unit
PI	Primary Input
PPI	Pseudo Primary Input
PO	Primary Output
PPO	Pseudo Primary Output
SFF	Scan Flip-Flop
NS	Non-Scan
KLPG	K Longest Paths Per Gate
PKLPG	Pseudo Functional K Longest Paths Per Gate
CUT	Circuit Under Test
CNF	Conjunctive Normal Form
SAT	Satisfiability

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION AND LITERATURE REVIEW

1.1 Path Delay Test

Verification of the timing specification is one of the most important requirements before we can declare a digital chip as “Ready to Market” [1]. Failure to meet certain timing specifications may lead to incorrect circuit performance, set-up and hold time violations, or limited voltage or temperature operating range. A better design always has minimized timing margin in order to maximize the performance while reducing power consumption. To ensure that the circuit under test has met the required timing margin, delay testing is essential. For any kind of structural testing we need a fault model. Here the fault revolves around the timing specification. Hence, we need a delay model. Different delay models have been discussed in [2][3][4]. One of the most popular delay models is the *path delay fault model*. In a circuit we have local delay faults and distributed delay faults. The performance of the path delay fault model is very impressive in detecting both. The path delay in a circuit is a combination of on-path gate input-to-output delay and the interconnect delay. The delay can be further split into gate transport delay and interconnect propagation delay to facilitate the unique computation of the delay to every fan-out of every gate. Another delay, namely inertial delay, is required to analyze the glitch behavior of the circuit. In a path delay model, the path is generated using a two pattern test. The generation of the path starts from the Primary Inputs (PI) or Pseudo Primary Inputs (PPI) and then the path travels through the desired fan-outs, forms the path and finally finishes at some primary output (PO) or Pseudo Primary Output (PPO). The PI or

PPIs are used to launch the transitions into the circuit, and the responses are captured at the POs or PPOs. PPIs and PPOs are used in a scan-based test, since the PIs must remain fixed and POs ignored in a low-cost production tester [5]. The response time should match the desired specification. In order to detect the delay faults the PPO values must match the expected results. The two-pattern test is composed of an *initialization vector* and a *test vector*. The first vector sets the circuit to the desired initial state. The second vector is responsible for launching the transitions.

1.1.1 Longest Path Delay

A generic synchronous digital circuit is a combination of combinational logic and several flip-flops synchronized by a common clock. In a scan based test, a scan chain is formed by serially connecting the sequential elements of the circuit. The flip-flops that are part of the chain are called scan flip-flops (SFFs). We can directly control the input nodes of the scan flip-flops, namely PPIs or Pseudo Primary Inputs, and we can directly observe the responses from the outputs of the scan flip-flops, namely Pseudo Primary Outputs or PPOs. The transitions flow from PPIs to PPOs via the combinational logic. All the gates that participate in the propagation of the transition are called *on-path gates* and the input of the gate that receives the transition is termed an *on-path input* [6]. The remaining input nodes of the gates are considered *off-path inputs* [6]. The path is a victim of a delay fault if the response time of the path is more than some pre-specified value. The delay of the path is the cumulative delay over all the gates and the interconnects on the path. Much prior work has been done on delay testing. Extensive studies on delay faults can be found in [7][8][9][10][11]. Based on several experiments and basic intuition, it has been concluded that the longest path in a circuit has always been subject to the maximum delay fault. If we can test the longest path in a circuit to ensure that it is free from delay fault, many shorter length paths are also being tested automatically. Hence, generating a longest path in a circuit has always been very important. A circuit is actually a graph and detecting all the possible sensitizable paths in a graph is an NP-hard problem. A reasonable approximation or tractable approach is to generate a limited set of longest paths and test them to ensure timing margin correctness. In [12], an approach has been suggested to generate K Longest Paths Per Gate (KLPG). The biggest advantage of the approach is the ability to detect if more than one path of the gate performs slower than expected. The local delay defects are manifestations of slow gate performance and global process variations [13]. The KLPG approach is able to detect both.

1.1.2 The Problem of Delay Test

Figure 1 shows a schematic of a generic combinational digital circuit to explain the basic concept of the path delay fault model. These combinational blocks are the "Can of Worms" for the suspected delay faults. Figure 2 shows a combinational circuit that has three inputs *x1*, *x2*, *x3* that are being fed by two vectors *v1* and *v2*. *(v1,v2)* is the required vector pair as discussed above. *v1* is the initialization vector and *v2* is the test vector. The figure also shows the individual delays of each gate. *y* is the output of the circuit and based on the vector pair the circuit expects a rising transition at the output node *y*. The cumulative time delay for the rising transition is expected to be *7 time units*. If any of the gates on the path incurs any extra delay, the expected rising transition would appear at *y* node after the desired *7 time units*. Figure 3 depicts the input/output transition where the shaded region, namely transient region, spans over the desired propagation delay. The extra delay in any gate due to defects, process variation, or noise would shift the output transition out of the shaded region.

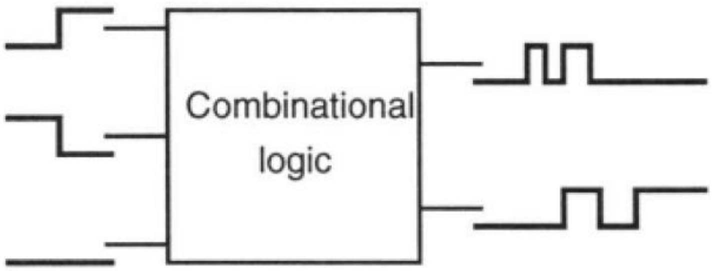


Figure 1. The combinational logic residing between the sequential logic (“Can of Worms” for delay faults)

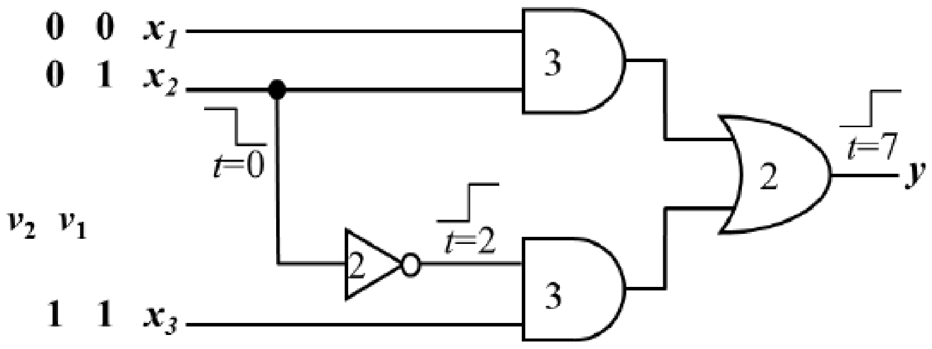


Figure 2. Delay Fault Problem in circuit

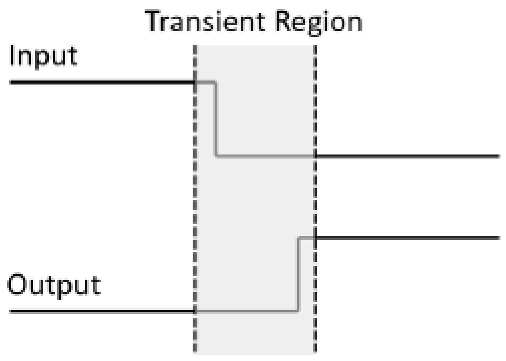


Figure 3. I/O transition and Transient region

As shown in Figure 2, if the timing margin is 10 time units and the OR gate incurs an extra delay of *4* (total *6 units*), then the output transition at *y* would happen at *11 time units*, which naturally leads to a path delay fault.

The total number of paths in a circuit is an exponential function of the number of gates in it and the fan-out number of each gate. If we consider all possible input vector pairs *(v1,v2)*, *"the longest delay combinational path"* of the circuit under test is termed the *critical path*. The delay of the critical path of any circuit dictates the shortest possible clock period, i.e. the highest possible clock frequency of the circuit with correct functionality. A circuit can be declared free from delay faults if output transitions for any possible vector pair never exceed the clock period.

1.2 The steps and the variants

A path in a circuit represented at the logic gate level should be sensitized first using the preferred ATPG-based approach. The approach is a three-step process.

1.2.1 Sensitization, Propagation and Justification

1. ***Fault Sensitization:*** In this step, the signal driving the fault site is forced to an opposite value from the fault value. This is done to distinguish between the correct circuit and the incorrect circuit.
2. ***Fault Propagation:*** This step involves the propagation of the fault effect. It may use one or more paths from the PI/PPI to PO/PPO.
3. ***Line Justification:*** In this step we check the consistency of off-path signal value assignments with respect to required values to be fed to the PI/PPIs. All the intermediate signal values get justified by assigning proper values to the PI/PPIs. If consistency is not found, the path is a false path.

"A path is said to be testable if a rising/falling transition can propagate from the primary input to the primary output associated with the path, under certain sensitization criteria" [14][15][16][17][18][19]. An inconsistent or non-sensitizable path cannot be tested. For a fault or transition propagation to occur successfully, all the off-path inputs must have non-controlling values [5]. Figure 4 is a small example of the path sensitization requirement for the path *a*-*c*-*d*. If a transition is launched at node *a*, then side input *b* of the OR gate on the path must have the signal value 0. At the same time, the AND gate on the path requires *b* to be 1. Since we cannot simultaneously set non-controlling values on the AND and OR gate side inputs, path *a*-*c*-*d* is false.

1.2.2 The variants of Path Delay Test

While defining a delay test, we need to mention what kind of delay we are interested in. Delay test of a path can be broadly classified into *robust* delay testing and *non-robust* delay testing. A circuit can have several delay faults. If the fault detection is

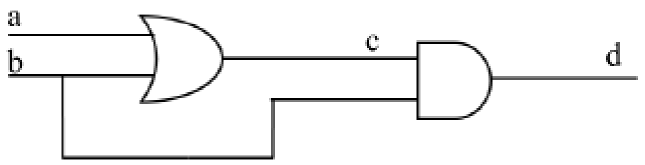


Figure 4. Path that cannot be tested

independent of the presence or absence of other remaining delay faults in the circuit, then we can consider that as a robust path delay fault test. On the contrary, if the fault can only be detected when no other fault is present, this is a non-robust delay test. Figure 5 explains these two variants of path delay test.

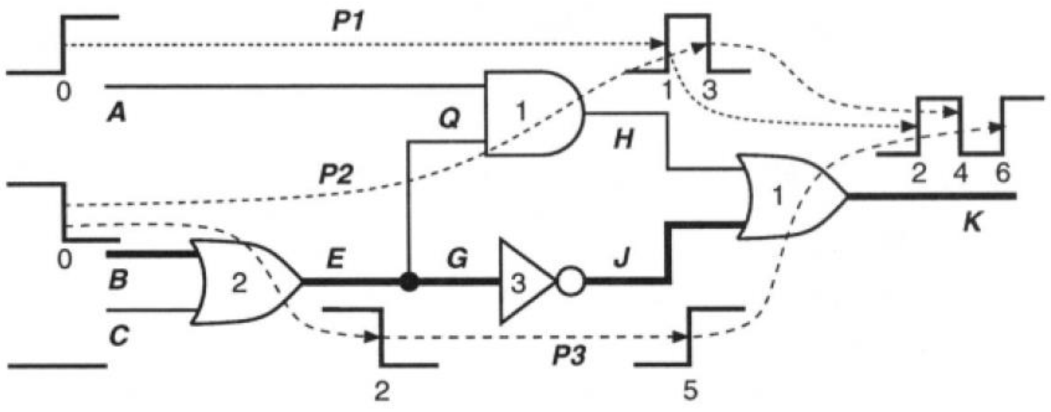


Figure 5. Transitions getting propagated (Robust and Non-Robust Path) [6]

In this example, suppose the clock cycle is *7 time units*. To operate correctly, all possible paths must have a delay of less than *7*. If any path delay exceeds *7 units*, that is a potential delay fault. The numbers shown on each gate in the figure are the propagation delay through them. We assume no interconnect delay. We can see very easily that among all the paths the longest is *p3* and hence, it is the critical path. If paths *p1*, *p2*, *p3* all incur extra delay and propagate the output transition after *7 time units*, the output waveform would be right-shifted and fault detection would be performed. Now, say, only *p2* and *p3* are faulty and *p1* is not, then the rising transition from path 1 would be detected at *7 time units* and would give a false impression of the circuit being correct. Hence, in spite of the *p3* path being delayed, the extra delay from path *p2* would never let the opposite value appear at the output capture time. Hence, the *p3* path delay fault cannot be tested if there is a delay fault in *p2*. Clearly *p3* is a non-robust path. For both kinds of tests, static sensitization criteria should be satisfied along with the launch of the transition at the start of the path using the test vector pair [5]. Robust test can be achieved if we can assure only one output transition, whereas for non-robust tests more than one output transition is permissible. It should be understood that all robust paths are also non-robust paths while the reverse is not true. Hence, non-robust path count can be termed as the total number of paths that can be tested (where we can identify the path under test). For robust paths, we need to ensure two things about the transition. First, the transition should be a real event, i.e an event that can exist without the help of other events. Second, it should be a controlling event, i.e it should not let other events happen prior to it. While using the two vector pattern (*v1*,*v2*), if the on-path transition is from a controlling (in *v1*) to non-controlling (in *v2*) value, it would block other transitions. Hence, the off-path input can be anything in *v1* and non-controlling only in *v2*. If the on-path transition is from non-controlling (in *v1*) to controlling (in *v2*), then the off-path inputs are expected to be non-controlling in both vectors *v1* and *v2*, in order to allow only one output transition. If these criteria are not met, the test is not robust.

1.3 Scan based Delay Test

In DFT (Design for Test) the most popular and effective approach is Scan-based Testing. Most individual sequential elements in a circuit are connected serially and thus provide full controllability of the outputs of those sequential units (PPOs) and full observability of the inputs of the sequential elements (PPIs). To facilitate scan operation, the flip-flops can be combined with multiplexer logic and form Scan-Flip-Flops (SFF). Figure 6 shows the internal structure of a SFF.

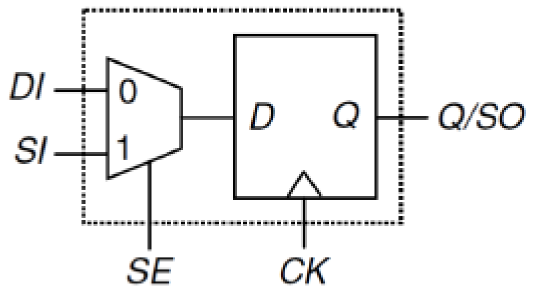


Figure 6. Muxed D Scan cell (Combination of D Filp-Flop and 2:1 Mux)

A scan FF with an input multiplexer is also called a Muxed D Scan Cell. For scan operation, the cell ports are scan input (SI), scan output (SO) and the scan enable (SE). To extend the serial chain, the SO port of each scan cell is connected to the SI port of the neighboring cell.

1.3.1 The operating modes

Scan based designs have three operating modes.

1. **Shift:** Assertion of the scan enable signal during clocking shifts the initialization vector (1st of the vector pair) into the SFFs. The clock frequency during shift is much lower than during functional operation, in order to simplify scan chain routing.
2. **Normal:** Normal mode starts with the de-assertion of the SE signal. This is just the functional mode of the circuit operating at at-speed frequency. The second vector of the vector pair, i.e the transition or test vector, is applied in this mode.
3. **Capture:** In this mode the circuit response is captured. SE is asserted and the captured result is shifted out of the chip. Normally Shift and Capture are overlapped.

Figure 7 presents the high-level block diagram of a typical scan chain. The SO ports from individual SFFs are connected to the SI ports of the neighboring SFFs.

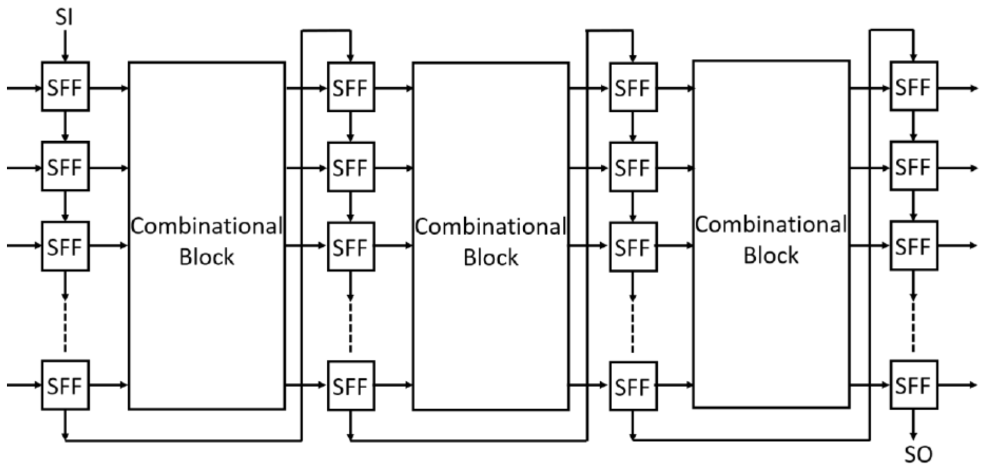


Figure 7. Scan Chain Block Diagram

1.3.2 Two wide-spread approaches

The scan-based designs are broadly classified into the following two approaches.

1.3.2.1 Muxed D Scan based approach

In Figure 8, we can see that the output ports of each SFF are connected to the Combinational Logic as Pseudo Functional Inputs (PPIs). Similarly, all the outputs of the combinational logic work as Pseudo Functional Outputs (PPOs) and are fed to the DI input ports of the SFFs. The Primary inputs (PIs) are *X1*, *X2*, *X3* and the Primary Outputs (POs) are *Y1* and *Y2*. These are the functional signals of the circuit under test. The PIs are driven by user inputs or the upstream logic. The POs can be observed directly and for the PPOs capture needs to be done at the SFF outputs of the scan chain. If a single scan chain becomes too long, we can break it into multiple chains with multiple SI and SO signals.

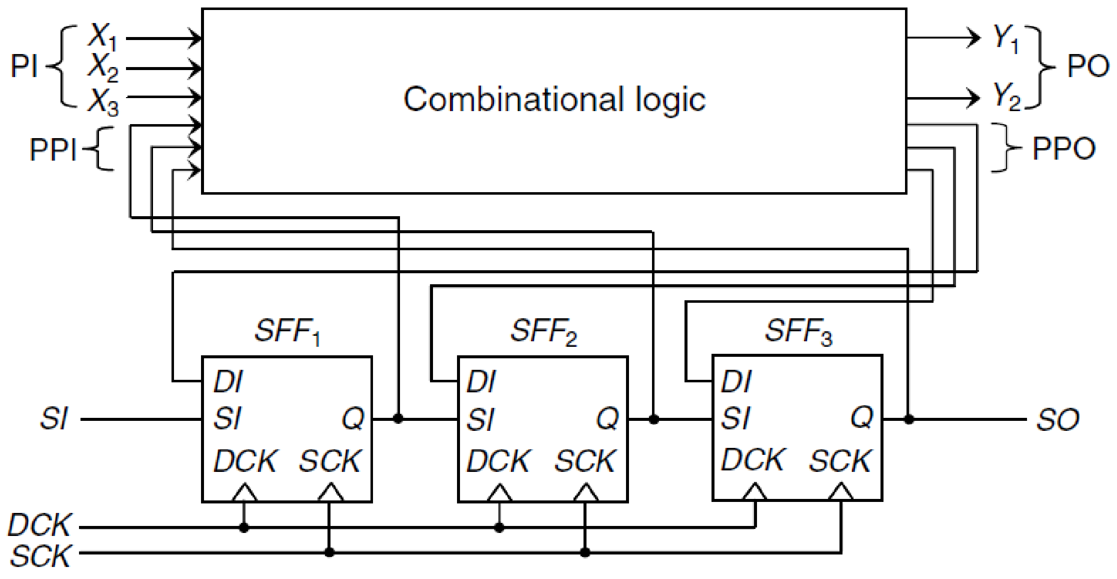


Figure 8. Muxed D Scan Based Design [5]

1.3.2.2 Enhanced Scan based approach

A pair of vectors is used to launch transitions and finally capture the response at the SFF SO ports at functional speed. In delay testing, we use pair of vectors. Now these two independent vectors can jeopardize the initialization of the circuit. The problem seems solvable if we can somehow insert hold latches into each scan flip-flop. Enhanced Scan Based design (Figure 9) is a suitable DFT architecture to achieve that. Here two bits of data can be applied simultaneously to the combinational logic. Vector *v1* is stored in shifted in and then transferred to the D-latches with an *UPDATE* signal. The test vector *v2*, is then shifted into the scan chain while the *UPDATE* signal is de-asserted. Assertion of the *UPDATE* gain happens after *v2* has been completely shifted in. This changes the values at the combinational logic inputs from *v1* to *v2* and thus the transition is launched into the combinational logic. The response is captured in the SFFs and scanned out.

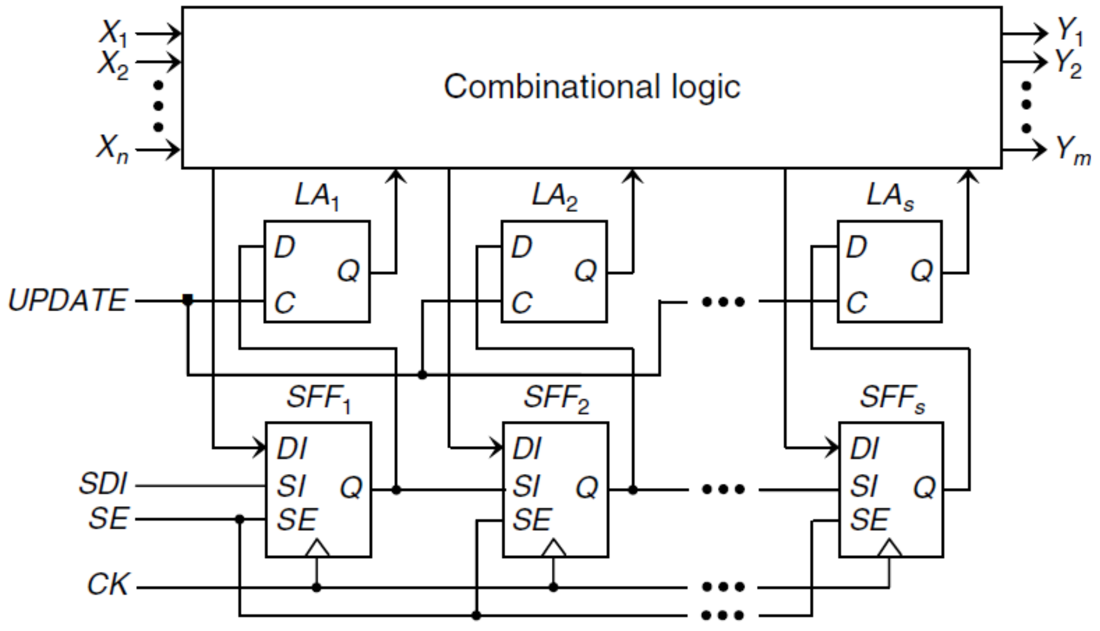


Figure 9. Enhanced Scan Based Design [5]

Using this Enhanced Scan approach, we achieve better delay coverage at the cost of area and delay overhead caused by the extra latches. False paths may also be activated, and thus cause over-testing. Several innovative clocking schemes have been proposed in [5][20][21] to get rid of the disadvantages.

1.3.3 Clocking Schemes

If the circuit does not have enhanced scan, then the test vector must be generated from the initialization vector. Two clocking schemes are widely used to generate the test vector.

1.3.3.1 Launch on Shift (LOS)

This clocking scheme is also called *skewed load*. In this scheme, the last shift pulse, launching the transition, is immediately followed by a capture clock pulse to capture the output test response. The second capture clock pulse is operated at functional speed. The SE signal must switch at functional speed between the last-shift launch clock pulse and the capture clock pulse. This essentially requires the SE signal to be implemented as a clock network. Figure 10 shows the Launch on Shift clocking scheme.

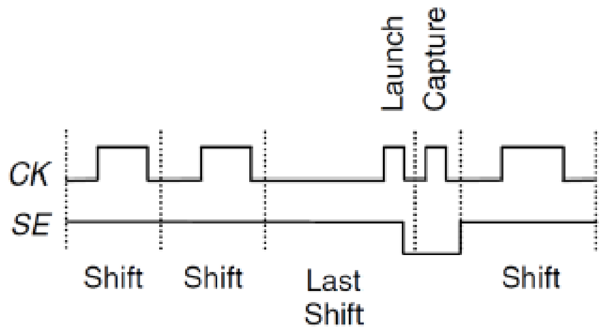


Figure 10. Clocking Scheme for Scan (Launch on Shift) [5]

1.3.3.2 Launch on Capture (LOC)

The other name of this scheme is *broadside* or *double capture*. Figure 11 shows the clocking scheme used in this strategy. Two consecutive functional cycles are employed to launch the transition and capture the response. The advantage is not having any speed related constraint on the scan enable (SE) signal. SE is de-asserted after loading the test vectors. After waiting for SE to stabilize during dead cycles, the launch and capture cycles are applied to the CUT. This kind of clocking scheme requires more test vectors and results in lower fault coverage in comparison to Launch on Shift. But the relaxed timing requirement on the SE signal has made this scheme very popular in high speed circuits. In addition, using the circuit response to generate the test vector eliminates many non-functional transitions, reducing over-testing and yield loss.

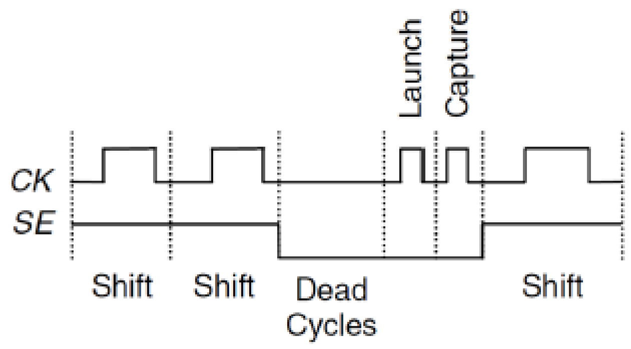


Figure 11. Clocking Scheme for Scan (Launch on Capture) [5]

In addition to the above two clocking schemes, clock domain grouping based schemes are also used. One-hot clocking and staggered clocking are two alternative schemes in this arena leading to reduction in test time and power in scan mode.

1.4 K Longest Paths Per Gate (KLPG) algorithm

To increase fault coverage, we may need to test multiple paths through each fault site. The KLPG algorithm [22][23] tests K longest sensitizable paths through each gate. This algorithm is able to detect both slow-to-rise (STR) and slow-to-fall (STF) faults on gate inputs and outputs, as well as distributed delay faults. Figure 12 is the overall flow of the algorithm. The PIs or PPIs serve as launch points and the POs or PPOs are the capture points. (In a low-cost production tester, only PPIs launch transitions, PPOs capture outputs, the PIs are fixed, and the POs are ignored). The first step is preprocessing, when static timing analysis deduces the maximum possible delay from each gate to the capture points. This delay does not consider logic constraints, and is termed the PERT delay. A path that has started at the launch point but has not reached the capture points is termed a *partial path*. The partial paths are initialized from the launch points. The upper bound of the delay of any partial path is measured in terms a metric called *Esperance*. Esperance is the delay of the partial path plus the PERT delay from the last node on the partial path to a capture point. Esperance is the upper bound of the delay of a partial path when it reaches a capture point [5].

1.4.1 The steps

The entire flow of the KLPG algorithm can be broken into three main steps:

1. Path Initialization
2. Partial Path Growth
3. Path Justification

Before path generation, controllability and observability (SCOAP) measures are computed for each gate [6]. The gates are rank ordered, with each gate being assigned the upper bound of the rank of the fan-in gates. This helps to determine the PERT delay [6] and SCOAP measures. The SCOAP measures also need the calculations of the fan-in and fan-out cones of each gate. Controllability is associated with the PIs/PPIs and dictates how easily these lines can be set to the desired logic values. Observability measures how easily a line value can be propagated to a PO/PPO. Controllability is computed with a forward traversal of the circuit, and then observability is computed in a backward traversal, using the controllability values.

During the second main step of KLPG, i.e. path generation, a gate is added in each iteration to the current partial path. When a partial path encounters a fan-out, it is split into different branches, generating additional partial paths. These intermediate partial paths are stored in a temporary pool, sorted in decreasing order of Esperance. In each iteration, the partial path with maximum Esperance is selected for extension by adding one gate to it. This generates a new partial path that is stored in the pool, and the iteration repeats. Each time a gate is added to an existing partial path for extension, constraints are added to the off-path inputs of that gate, based on the robust/non-robust sensitization criteria. For a successful propagation through that gate, the off-path inputs need be assigned non-controlling values. The sensitization constraints are propagated throughout the circuit using direct implications. If the constraints conflict with existing constraints, then the gate cannot be added to the partial path, and this path is rejected as false. If the partial path has not reached a capture point, false path elimination techniques are used in order to drop false paths. For example, if the minimum Esperance of a gate in a path is higher than the maximum Esperance of another gate in the same path, we conclude that path is false. When a partial path reaches a capture points, it becomes a *complete path*. We then perform final justification, to generate a test pattern that meets the path constraints. The entire procedure explained so far is iterated again and again until sufficient number (*K*) of complete paths through the target line have been generated.

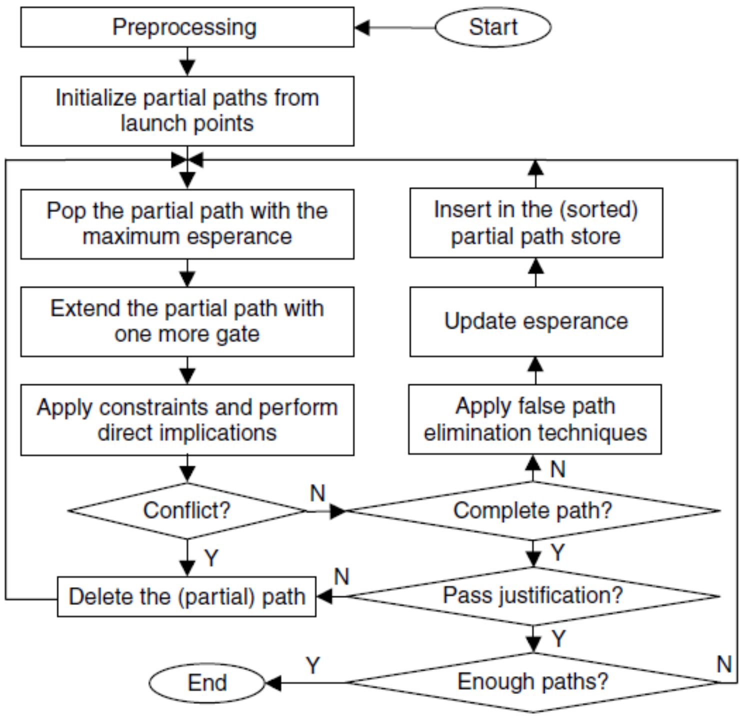

Preprocessing
Start
Initialize partial paths from launch points
Pop the partial path with the maximum esperance
Extend the partial path with one more gate
Apply constraints and perform direct implications
Conflict?
N
Y
Delete the (partial) path
Insert in the (sorted) partial path store
Update esperance
Apply false path elimination techniques
N
Complete path?
Y
N
Pass justification?
Y
End
Y
Enough paths?
N

Figure 12. The flow of the KLPG algorithm [5]

To reduce the number of test patterns, compaction and compression is done on the test patterns. Both static and dynamic compaction are available in the system. Static compaction is done after test patterns have been generated, while dynamic compaction is performed by the justification engine during test generation. Dynamic compaction produces many fewer patterns, but at the cost of more justification runs, and maintaining a pool of partially-filled test patterns. For this work, the entire KLPG algorithm has been realized in the tool *Codgen*.

1.4.2 Pseudo Functional Testing

There are many paths in the circuit that are functionally infeasible. Test pattern generation may produce initialization or test vectors that do not occur in functional operation, activating functionally infeasible paths. This leads to over-testing and unnecessary test time and power. Pseudo functional test reduces this problem. In pseudo functional scan testing, the initialization and test vectors are the same or similar to functionally reachable states. The main challenge is identifying reachable states. Several techniques [24][25][26] have been developed, but have not been deployed due to their high cost.

1.4.3 Pseudo Functional KLPG

In Launch on Capture, when the Scan Enable (SE) signal switches (between scan mode and functional mode), dead cycles are inserted to allow enough time for the SE to settle. During this time, off-chip currents in the power grid reach a quiescent state. The activation of launch and capture cycles at functional speed causes a sudden increase in off-chip current demand. The off-chip inductance limits the rate of increase in off-chip current. In the meantime, charge is consumed from on-chip capacitors, leading to a supply voltage droop. This is referred to as *dI/dt* noise. The voltage droop causes the circuit to operate slower than normal, which can lead to good chips being rejected as bad, termed test overkill. Figure 13 shows the voltage droop event on the power grid of an Intel Itanium processor [27].

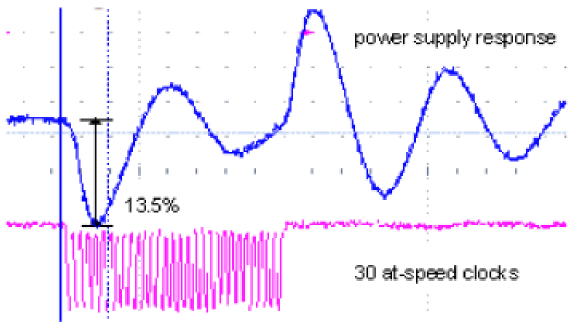


Figure 13. Voltage Droop in power supply due to delay test inductance [28]

As a solution, we need to give enough time for the voltage to stabilize before applying the at-speed test. Our solution is to apply a number of medium-speed functional cycles, termed *preamble cycles*, before the at-speed test. These extra cycles produce on-chip activity that ramps the off-chip current to functional levels. At the same time, these cycles filter out many non-functional states, so that the launch state is closer to a functional state. Hence we term this approach pseudo functional KLPG (PKLPG) test [23][28]. Figure 14 shows the clocking scheme used for PKLPG. The assertion of SE signal happens only during scan-in and scan-out operations.

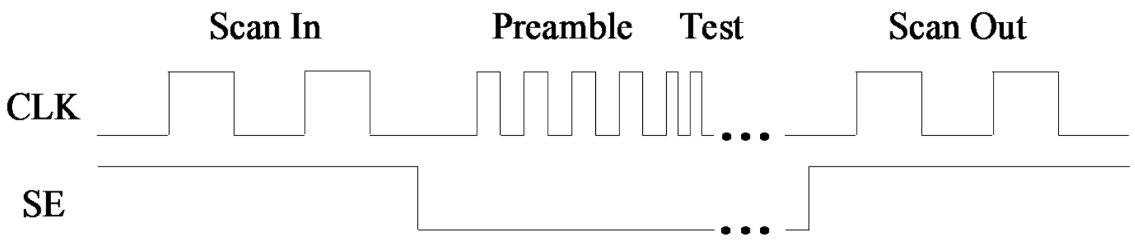


Figure 14. Clocking scheme (Preamble cycles) used in PKLPG test [28]

1.5 Boolean Satisfiability

1.5.1 SAT and CNF

Circuit verification and testing are the fields where Boolean satisfiability is used extensively. A circuit is first represented in Conjunctive Normal Form (CNF) [29]. SAT solvers based on techniques like Boolean Constant Propagation (BCP) [30] and backtracking with conflict analysis learning are used to determine whether the CNF formula is satisfiable, and supply the test pattern that satisfies it.

A CNF is a logical AND of several clauses which are the logical OR of several literals. The literals can have values either 0 or 1. The goal is get a proper combination of values (0 or 1) for every literal so that the entire expression finally is satisfied, i.e. has a logical true value. As an example, we can use the example of an AND gate. The logical AND operation is $C = A \cdot B$. The CNF representation of C is $(\sim C + A)(\sim C + B)(\sim A + \sim B + C)$. The goal is to find suitable values for A, B, C so that the entire expression produces the value 1. This is possible only when A, B, C all have the value 1. This is exactly the behavior of an AND gate. The clause with three variables is called the 3-CNF form. Finding a satisfiable input set for any k-CNF with k being more than 2 is an NP-hard problem. The heuristics in modern SAT solvers can find solutions to most SAT problems in reasonable time.

SAT can be applied to generate vectors in ATPG. The obvious difficulty involves the incorporation of real delay values. The approach in [31] uses a mixed approach using structural and functional test. A structural approach is used to generate the paths and SAT is used for path justification. Techniques to speed up SAT solvers have been extensively studied. In [31], dynamic SAT solving (DSS) uses the structural information of a circuit to reduce the solution search space. During CNF creation, much circuit structural information is lost. Heuristics such as Direction of Gates and Circuit Observability Don't Cares (Cir-ODC) help in reducing solution time [31].

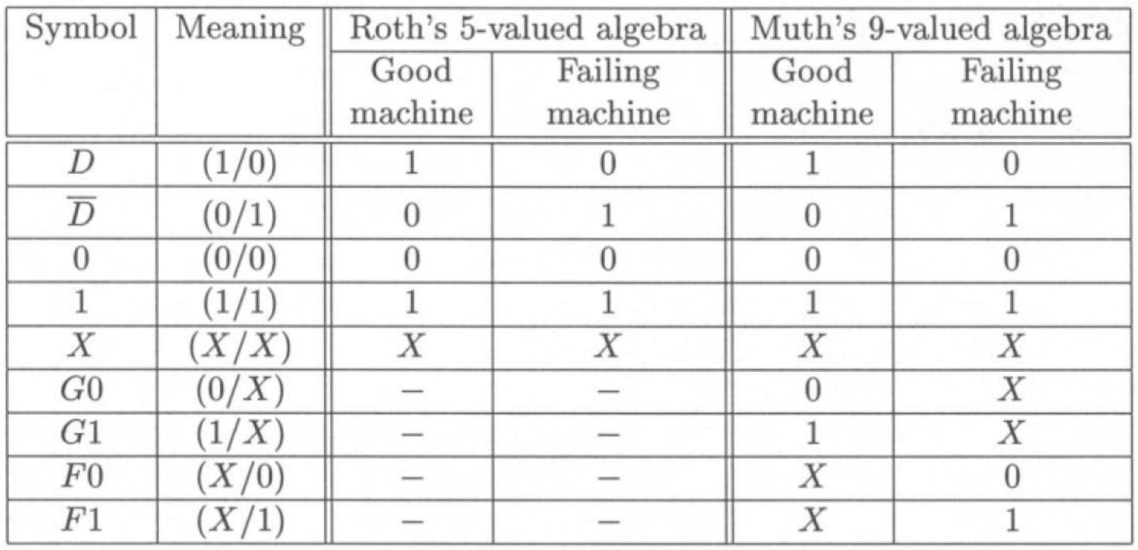
1.5.2 MiniSAT: The SAT in *CodGen*

MiniSAT [32] is an open-source SAT solver that is in use in *CodGen*. Final justification and dynamic compaction are the two stages where SAT is used extensively. In the KLPG algorithm, once a partial path becomes a complete path, all the values are assigned to the gates on that path and justification is performed. Since, a pair of vectors is used in the LOC scheme for launching the transition; there are two variables that are used for SAT solving in two time frames. If PKLPG is used, the circuit is unrolled in time and the SAT engine solves the problem in terms of the scan-in pattern.

1.6 The need for an extended Algebra

With the emergence of scan test, we have achieved the much-required controllability of all the flip-flop output nodes deeply hidden in the circuit. However, it is not feasible to make all flip-flops in a circuit into scan flip-flops. Commonly a few percent of flip-flops are non-scan flip-flops. There is no control on the output nodes of these non-scan flops. These are uncontrolled signals from the perspective of testing. Hence, many possible signal propagation paths going through those nodes must be discarded, resulting in a delay test with reduced path delay fault coverage. It is common for the timing critical paths in a circuit to pass through non-scan flops and so have poor test coverage.

If the circuit is not a full-scan design, then the non-scan flip-flops may or may not be initialized after the scan-in of the first vector. Hence, all these non-scan flip-flops that cannot be initialized are considered “uncontrollable” in the test generation process. They behave like black-boxes and thus may block a potential path propagation, resulting in fewer paths being tested.



Symbol	Meaning	Roth’s 5-valued algebra		Muth’s 9-valued algebra	
		Good machine	Failing machine	Good machine	Failing machine
D	$(1/0)$	1	0	1	0
$\overline{D}$	$(0/1)$	0	1	0	1
0	$(0/0)$	0	0	0	0
1	$(1/1)$	1	1	1	1
X	(X/X)	X	X	X	X
$G0$	$(0/X)$	–	–	0	X
$G1$	$(1/X)$	–	–	1	X
$F0$	$(X/0)$	–	–	X	0
$F1$	$(X/1)$	–	–	X	1

Figure 15. Multiple Valued Algebras [34][35]

We are already familiar with Roth's five valued algebra [34][35] used in combinational circuit stuck-at fault testing and Muth's nine-valued algebra [35] used to minimize unknown signal values during time-frame expansion based stuck-at testing of sequential circuits. Figure 15 shows these algebras. In this work, we extend the Muth algebra by including the uncontrolled state as a legal logic state, with the goal of reducing pessimism in test generation. A more advanced algebra permits many path pruning decisions to be taken earlier, reducing test time, and many of the lost paths due to uncontrollable non-scan flops can be recovered, increasing path delay fault coverage.

1.7 Structure of the Thesis

In this thesis, we present an extended algebra based approach that results in improved path coverage. There are 11 logic values being considered here for the formation of truth tables of basic logic entities. Two vector (initialization and test vectors) testing led to the decision of having 11 values: Zero (Stable), One (Stable), Unknown, Uncontrollable, Rise, Fall, Zero/Uncontrollable, One/Uncontrollable, Unknown/Uncontrollable, Rise/Uncontrollable, and Fall/Uncontrollable. Results have been obtained from runs on benchmark circuits for different values of *K* (i.e. number of long paths) to demonstrate the benefit of this algebra.

The organization of the thesis is as follows. In Chapter 1, we introduce the generic descriptions of design for test (DFT), delay testing, different scan based testing approaches and KLPG algorithms, along with the constructs used to implement it. Chapter 2 describes the motivation behind the work. The implementation details of *CodGen* are presented briefly in Chapter 3. In Chapter 4, we show the 11-value extended algebra and different truth tables used to represent logical entities. Chapter 5 summarizes the benchmark results showing improvements in path recovery, fault coverage and average CPU time per path.

In Chapter 6, we conclude our work with remarks on the future work that can be done based on this work.

2. MOTIVATION

2.1 Controlling all the sequential elements?

Billions of transistors on a single chip have made testing immensely complex [36]. We have already seen that how scan based approaches provide a cost-effective way to test the circuit logic behavior. Each sequential element in the chip is replaced by a SFF and those SFFs are connected serially to form a scan chain. In a modern chip with hundreds of thousands of sequential elements, multiple scan chains are used to limit chain length. Even so, shifting test patterns in and results out of the scan chains consumes the bulk of the digital logic test time.

In practice it is not possible to connect all flip-flops into scan chains, due to area, power or delay constraints. In addition, circuits contain elements such as embedded memories and register files that cannot be converted into scan chains. For those sequential elements, we cannot observe the values going into them or control the values coming out. These blocks act as black boxes producing uncontrollable values, blocking many functional paths. Hence, there is always a need for an approach that would lead to better coverage of paths for chips with unavoidable non-scan sequential elements.

2.2 Path Generation in *CodGen*

The pseudo functional KLPG algorithm has been implemented in a tool *CodGen*. Each partial path has an Esperance value associated with it. When a partial path has multiple fan-outs, Esperance is used to select the branch that leads to the potentially longest complete. In the existing implementation, *CodGen* assumes that all the sequential elements in the circuit are SFFs. If a sequential element is not a SFF, the path being produced by KLPG would be a untestable, since either a transition could not be launched on the path, or the result could not be shifted out. Moreover, we do not have any controllability on the output nodes of those non-scan flops. For those uncontrollable nodes, many paths would be discarded due to direct implication and path justification failures and conflicts.

2.3 Clocking Scheme (Coda Cycles for recovery)

There is one way that can be implemented in the clocking scheme in order to move a captured FF value to a SFF. Additional functional cycles after the capture cycle, termed *coda cycles*, can be added before asserting the SE signal. The coda cycles are slow enough that there are no timing considerations. The clocking scheme for such an implementation is shown in Figure 16. Figure 17 shows the flowchart of the PKLPG algorithm.

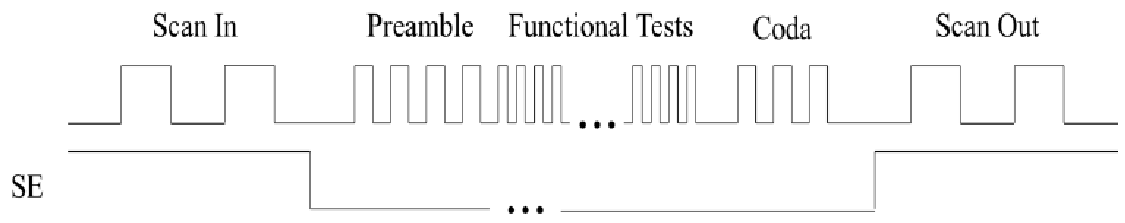


Figure 16. Use of Coda Cycles (at-speed delay testing) [28]

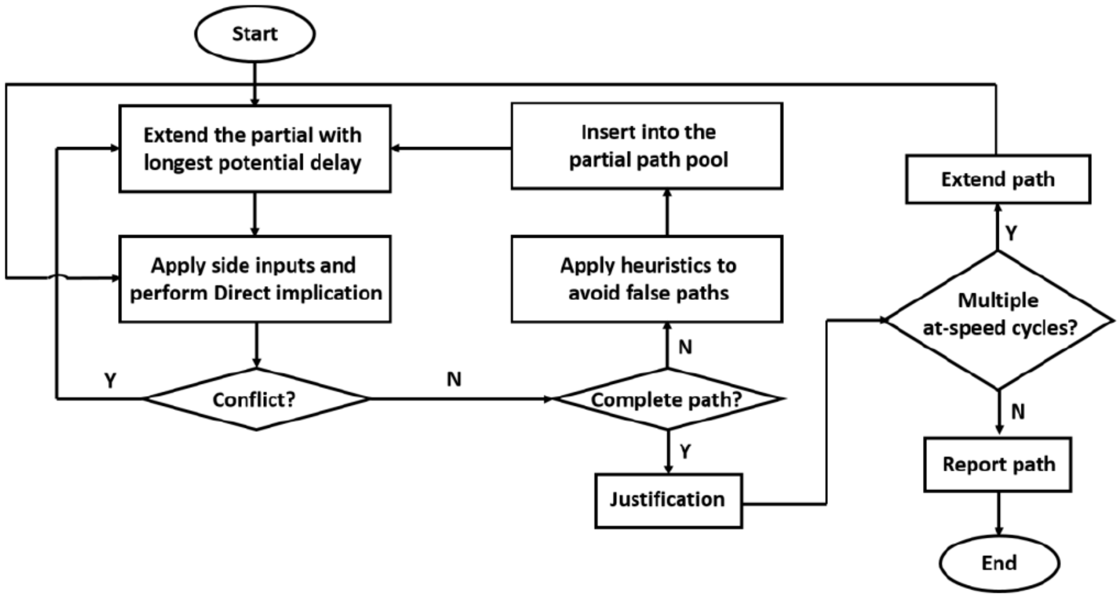

Start
Extend the partial with longest potential delay
Insert into the partial path pool
Extend path
Apply side inputs and perform Direct implication
Apply heuristics to avoid false paths
Y
Multiple at-speed cycles?
N
Y
Conflict?
N
Complete path?
Y
N
Report path
Justification
End

Figure 17. Pseudo Functional KLPG Algorithm (PKLPG) [28]

2.4 Related previous work

In [12] the KLPG algorithm is described in detail. It explains how the KLPG algorithm generates *K* longest paths per Gate. The KLPG algorithm is applicable for both sequential and combinational logic circuits. In [33] strategies are shown that extend the KLPG algorithm over multiple cycles. The longest path generation process is realized over all clock cycles. The metric being used for the determination of the longest path is called Esperance [12]. Justification is an important step of the KLPG algorithm. For justification test patterns are generated using a SAT engine. The details of the approach can be found in [37]. The entire work presented in this thesis is built on top of this existing software base. The collective name of these tools is *CodGen*. The goal of *CodGen* is to detect combined local/global defects. The work in [22] describes the problem of having non-scan sequential elements such as an embedded memory in a circuit. The work in [22] justifies the need for an approach that brings more optimism in terms of path coverage out of those uncontrollable nodes. Other examples of the extension of traditional algebra for improvements in testing can found in Roth’s 5-value algebra [34] and Muth’s 9-value algebra [35].

3. IMPLEMENTATION

3.1 KLPG Strategy for Path Generation

In the existing *CodGen* implementation, preamble cycles can be inserted before the at-speed cycles. Usually all the sequential elements are assumed to be SFFs. Hence, the presence of non-scan flops leads to path coverage loss. As a remedy, coda cycles and observability-based backtracking can be used to propagate captured values from a non-scan cell to a SFF. During the process of KLPG, the behaviors of the gates have been defined in truth tables. The direct implication step uses those values to deduce the paths. Extensions of those truth tables have been done to include the uncontrollable and auxiliary logic states. Using the extended tables, many paths are recovered in spite of the presence of non-scan flip-flops. The following section briefly explains the steps involved during a full cycle execution of the KLPG algorithm.

3.2 Different *CodGen* constructs for gate processing

The first step of *CodGen* is targeted towards gate processing. It is composed of several sub-steps. Each sub-step contributes the advancements as reported below.

3.2.1 Parsing the HDL description of the circuits

The current version of *CodGen* only understands Verilog HDL input. It parses the Verilog structural description of the and forms an intermediate gate network. The software has a class namely “Gate”. This class hosts the definitions of all the logical entities, including AND, NAND, OR, NOR, XOR, XNOR, NOT, MUX, and BUFF. The behaviors of each of these entities have been captured in a header file TruthTable.h. Each entity has an individual truth table. The description of the sequential elements is present in a different Verilog file. It defines the serial connection of the flip-flops forming the scan chain (here we assume one scan chain). There is a provision for a separate file that contains scan attributes. The goal is to designate the flip-flops as scan or non-scan. The parser uses the scan attribute file along with the Verilog files.

Each gate is assigned a unique GateID. The terminal nodes, i.e. PIs, PPIs, POs, PPOs are also assigned GateIDs to maintain consistency in the flow. The position of a gate (in terms of rank) from the input or output is used to define its level. Levelization [6] is performed for each gate of the circuit. SCOAP measurements of controllability and observability are then computed [6].

3.2.2 Assignment of delay to each gate

There is a delay file that *CodeGen* uses to assign delay to each gate in the circuit. The delays will typically come from a cell library and perhaps back-annotation. If delays are not available, unit delays are assumed. If delay variation is provided, worst-case delays are used. The delay assignment is accomplished for both Rise and Fall transitions.

3.2.3 Search space – The Fan-In/Fan-out cone

The intermediate gate network (in-memory circuit structure) requires the knowledge of fan-in and fan-out cones of each gate. How fan-in and fan-out cone define the search space the KLPG algorithm is shown in Figure 18. In Figure 18, the search space has been shown for gate *g* in terms of its fan-in and fan-out cone. The signal lines that we see within the cones are termed as on-path and the ones out of the cones contribute to the off-path or side-input set [6]. The values of the off-path inputs of gates should always be non-controlling values. The non-controlling value assignments to the side-inputs are necessary for proper propagation of any transition from PPIs to the fault sites.

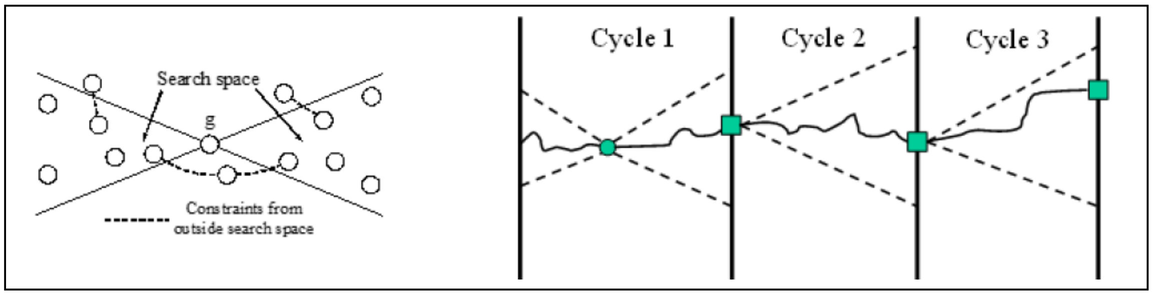


Figure 18. Search space of KLPG flow [5] [33]

3.2.4 Circuit initialization

This step is responsible for deriving the controllability and observability of the gates based on the levelization value already calculated. CC0 and CC1 are the controllability metrics that define the ease with which a combinational logic node can be set to 0 or 1 respectively. All the nodes with level 0 i.e. PIs and PPIs, have CC0 and CC1 values set to 1. Then the controllabilities of the first logic level are computed, then the next level, and so on. Observability is computed starting from POs or PPOs. The POs and PPOs are assigned an observability (CO) value of 0. The analysis then proceeds by level backwards in the circuit. Figure 19. shows a snapshot of a circuit along with the controllability and observability of each gate.

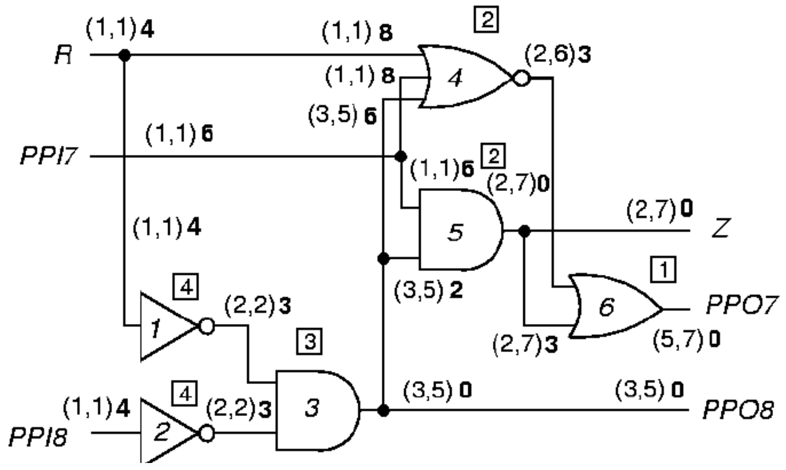


Figure 19. Calculation of Controllability and Observability (SCOAP values)

The numbers on each gate in Figure 19 denote the level of that particular gate. The combinational controllability (CC) and combinational observability (CO) values for each gate has been represented as (CC0,CC1) CO.

3.3 Path generation (Path ending at PPO or input of SFF)

In the existing version of KLPG, path generation starts from a SFF. There is a function "AimingPathGen" that creates a pool for the partial paths. This pool of partial paths iteratively keeps track of the paths that keep expanding by one gate in each iteration. The growth of a path is triggered by both Rise and Fall transitions. For each transition from a SFF, (*K* multiplied by *i*) defines the total numbers of iterations required. *K* is the parameter that defines how many longest paths per gate are to be generated and *i* denotes the fan-out of a SFF.

Each iteration encounters the pool of partial paths sorted based on their Esperance value. The partial path with highest Esperance value is chosen for extension. The growth of the partial path stops when it reaches a PO/PPO. Once a PO/PPO is reached, the partial path becomes a complete path and is excluded from the partial path pool. The next partial path is taken if the total number of total paths through that gate has not reached *K*. Justification is performed on the complete path using the SAT engine, and the corresponding test pattern generated, if it exists. Figure 20. shows the typical Esperance based path growth in KLPG.

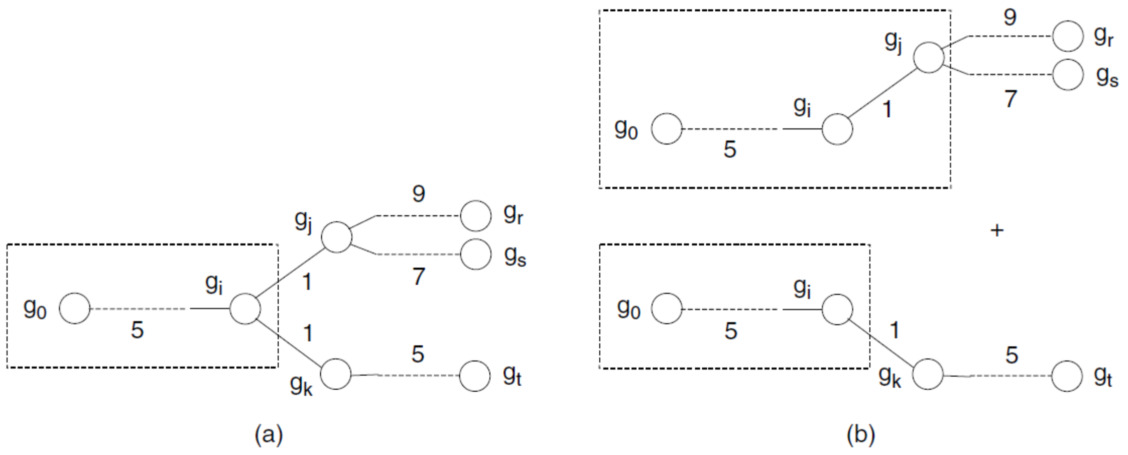


Figure 20. Path growth (Esperance based path growth to obtain longest path)

3.4 Time Frame Expansion

The automatic test pattern generation logic in *CodGen* uses the time-frame expansion approach. The combinational logic of the circuit is replicated or expanded *n* times, where *n* is the number of time frames considered. For a pair of time frames, two copies of the combinational logic are glued together, and thus creating a circuit model of double size. The circuit instance with time frame tag frame-1 receives its input from the copy of the circuit with time frame tag frame-0. The delay test vector pair is applied to the combined circuit. How time expansion is executed using the "Launch-on-Capture" (LOC) scheme is shown in Figure 21.

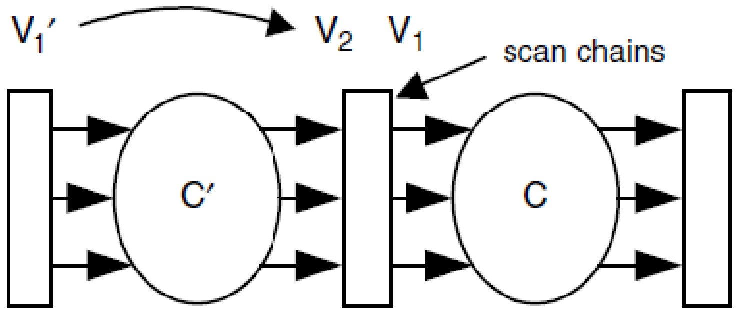


Figure 21. Time Frame Expansion methodology (*V1, V2* delay test vector pair)

The primary aim of this step is to assign required logic values to a gate in the path so that the desired transition can be propagated successfully. If we want to propagate a rising transition, the frame-0 vector needs to be '1' and frame-1 vector should be '0'.

Splitting into branches happens if the last gate on a partial path is followed by multiple fan-outs. For successful propagation of the transition through the path, sensitization constraints are applied on the side inputs of the path. Side inputs are expected to have non-controlling values for transition propagation to the fault site or PPOs or until the end of the path. Recursive direct implication is executed on the search space of each gate [5]. The logical state change of any output node needs to also be propagated back to the downstream logic.

Failure in direct implication occurs when a feasible input combination cannot be applied to a gate due to logical inconsistencies or environment constraints (e.g. preset signals for a certain operating mode). If the same line is the side input of two different gates with different non-controlling values, only one assignment is possible. Thus, the other gate becomes incapable of propagating the transition, so the path is false. Figure 22 gives an example of a conflict that prevents propagation of the transition on g_i through gate g_j. The search process must then consider propagation through g_k. This process, termed search-space trimming. [5], reduces the path search space by avoiding infeasible regions. In this example, all paths through both g_i and g_j are false, and so will not be extended.

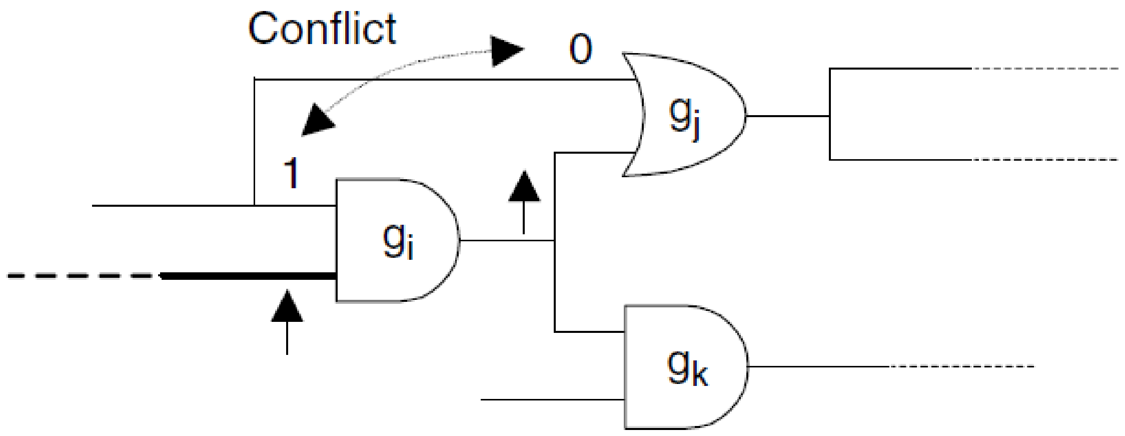


Figure 22. Assignment conflict during Path sensitization

3.5 Final Justification

Once a partial path becomes a complete path, it goes through a step called final justification. Until it is justified successfully, the path cannot be reported as a true path. In the situations where the path extends over multiple at-speed cycles, or coda, cycles, justification is done after the path has extended to the end of each cycle, to avoid searching over many cycles for a path that turns out to be false.

4. 11-VALUE ALGEBRA

4.1 Beasts beyond control !!!!

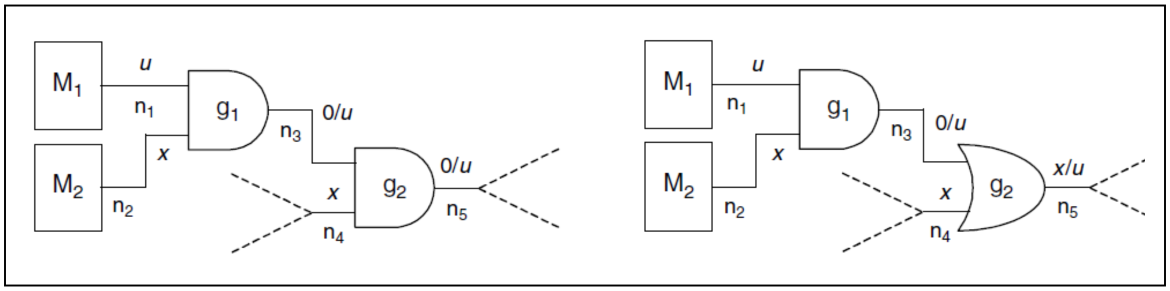
Real designs do not have 100% scan flip-flops. In real designs, some sequential elements cannot be initialized to a constant 0 or 1, even after simulating the test sequence through the scan procedure. There are many time critical paths where the extra delay caused by the MUX in the SFF cannot be allowed, Moreover, the embedded memories and register files provide limited access to the dense sequential logic inside them. There are many logic “black-boxes”, third party IPs, and analog units in every design whose input/output nodes are hard to observe or control. For all the above entities, it becomes extremely difficult to determine the output signal values. Those hard to predict signal values are assumed to be “uncontrollable”. The aim of SFF is to enhance controllability, but the presence of uncontrollable elements makes it difficult to apply necessary constraints to side-inputs for successful propagation of transitions. Because of less controllability, many partial paths cannot be extended, which otherwise would have been complete paths. This reduces path delay fault coverage.

4.2 The Eleven Values

Our approach to dealing with uncontrollable values is to add more analysis to the logic value propagation, to reduce pessimism. Instead of outright rejection due to an uncontrollable side-input, is it possible for us to extend the logic truth tables that distinguishes between controllable and uncontrollable unknown values? The answer is yes and is being done using the extended 11-value algebra. As our test set uses a two vector testing pattern, every final logic value of *v2* (i.e. 0 and 1) has two variants depending upon the value of *v1*. If *v1* is 0 and *v2* is 1, then the value of the pattern is Rise. If *v1* is 1 and *v2* is also 1, then the value of the pattern is Stable 1. Note that, in both cases *v2* is 1. Just the value of *v1* has created two situations of Rise and Stable 1. The same analysis is applicable when *v2* is 0. Overall, Rise, Fall, Stable 1, and Stable 0 are different events and in our extended algebra they are treated as separate logic states. Along with these, we have the unknown logic value (i.e. *X*). Uncontrollable itself would be a legal logic state. Depending upon the logic behind it, each logic entity would lead to either a controllable value or some value that is beyond control. Hence, there are cases when a node either can have one among all the certain logic states (Stable 1/0, Rise/Fall, Unknown) or uncontrollable values. Stable 1/0, Rise/Fall, and Unknown logic levels can be merged with the uncontrollable logic state to create additional combined logic states. Here the 11 logic states are Stable 1, Stable 0, Rise, Fall, Unknown, Uncontrollable, Zero/Uncontrollable, One/Uncontrollable, Unknown/Uncontrollable, Rise/Uncontrollable, and Fall/Uncontrollable. The uncontrollable logic state is symbolized as *u*. When the test generation starts all the uncontrollable non-scan flip-flops, embedded memory elements, black-box outputs are assumed to be *u*. All the other lines are *x* i.e. unknown, but a known value can be assigned to them during test generation. For example, if we have an AND gate with one input as *x* and the second input as *u* (i.e. output of some uncontrollable logic entity), the possible outputs of the AND gate are either 0 (if *x* is 0) or *u* (if *x* is 1). We refer to this output value as *0/u*. So, the AND gate output being uncontrollable depends on the choice of the first input but not solely on the presence of an uncontrollable value on the second input.

4.3 Use of Extended Algebra at Circuit Level

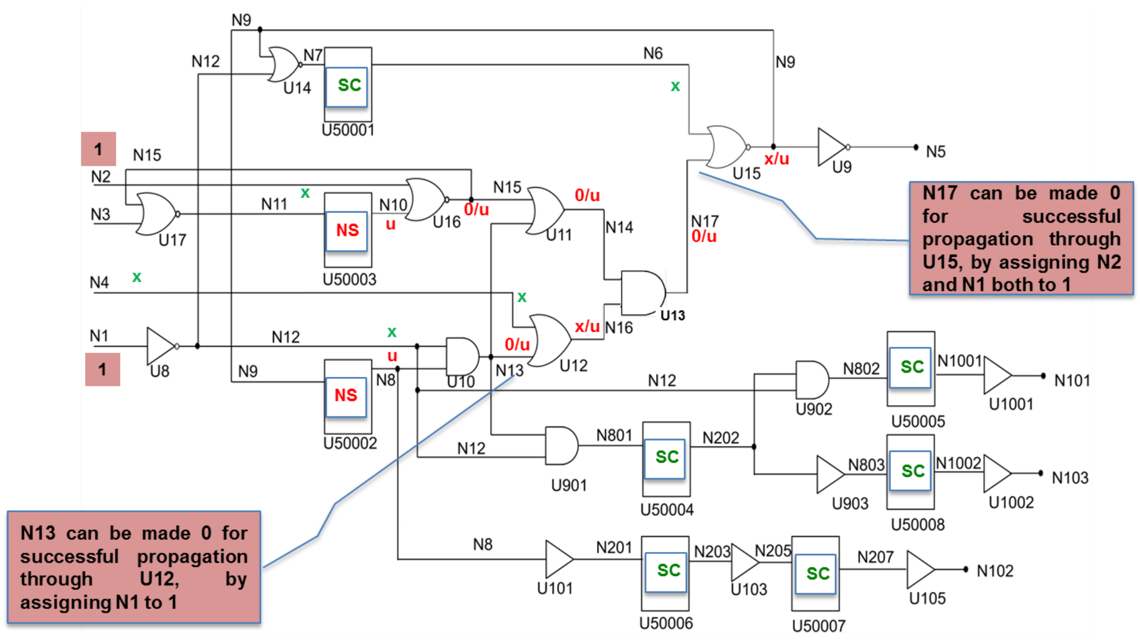
In Figure 23 *M1* is an uncontrollable entity, say a non-scan memory cell, while *M2* is a SFF. Using conventional three value (0,1,x) algebra, all the lines would be *x*.



(a) Search space trimming **(b) Scope of additional search**

Figure 23. Use of Extended Algebra

n3 is the output of AND gate *g1* whose input *n1* is *u* and input *n2* is *x*. The *g1* output is *0/u* i.e. *g1* can never be 1. In Figure 23(a) for paths through *n4*, the propagation through gate *g2* requires side input *n3* to be 1, which is not possible. Therefore all paths through *g2* are false. This early determination reduces CPU time. Using the traditional algebra, this search space trimming would not have been done and finally would end up exploring many untestable paths. Here we also see that *n5* can never be logic 1, so the transition delay faults can never be tested at *n5*. In Figure 23(b) we see the scope of additional path search leading to more coverage. Say, the partial path via *n4* has grown to the level of gate *g2* (i.e. *g2* is the last gate on the partial path). Now for successful propagation through gate *g2*, we require the side input *n3* to be set at logic 0. Using the extended truth table, the *n3* node should have the logic value *0/u*. If we perform direct implication to determine if can be set to 0, we easily find a solution by setting the controllable input *n2* to 0. Hence, irrespective of *n1* being assigned *u* an uncontrollable unit, *g1* still produces the required off-path non-controlling value 0 for successful propagation through *g2*. Further direct implication is to be done on *M2* now for ensuring *n2* to be 0. If the traditional algebra had been used, then the direct implication would have stopped at *g1*. Here, the use of extended algebra has paved the way for additional path search.



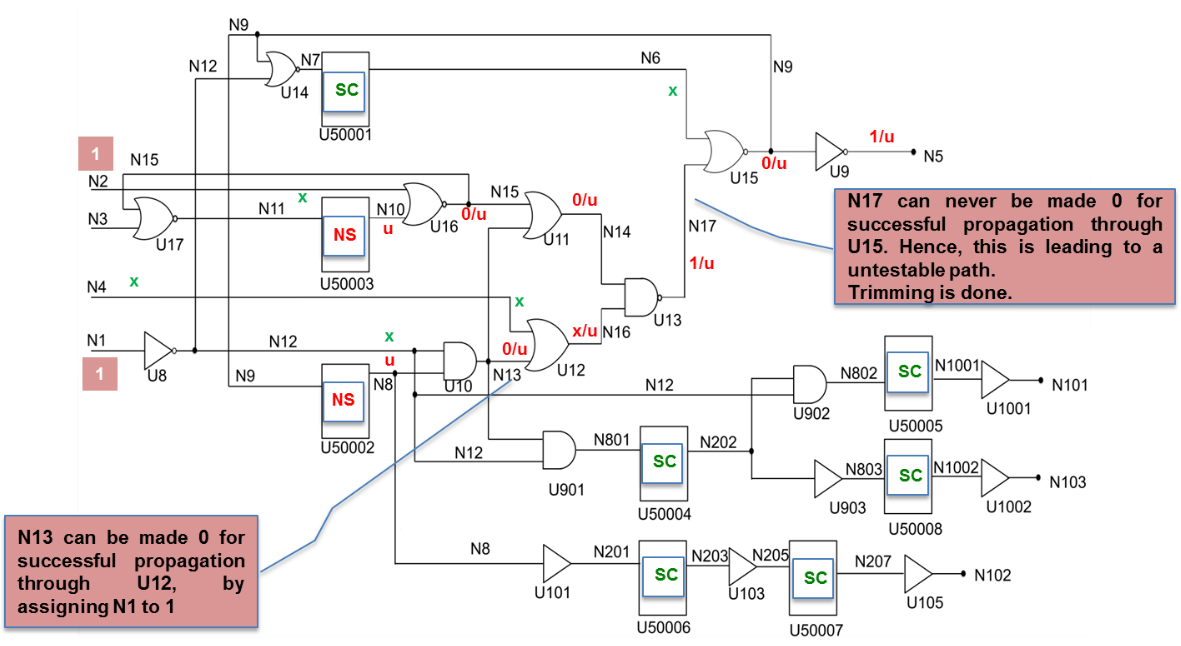
N17 can be made 0 for successful propagation through U15, by assigning N2 and N1 both to 1

N13 can be made 0 for successful propagation through U12, by assigning N1 to 1

Figure 24. Use of Extended Algebra for an ISCA89 S27 modified Circuit

Figure 24. shows the use of extended algebra for extra path coverage using an example of an ISCAS benchmark circuit. In the modified s27 circuit, the flip-flops U50002 and U50003 are non-scan. The remainder of the flip-flops are SFFs. The outputs of U50002 and U50003 have been assigned the logic state *u*. For the path growth of the partial path through *N4*, we need to ensure that successful propagation happens via the OR gate *U12*. To achieve that, *N13* should have the value 0. *N13* is the output of an AND gate whose one of the inputs is *u* (the output of U50002). Therefore, we cannot do direct implication for *N8*. However, the other input *N12* can be made 0 if the primary input *N1* is assigned 1. Hence, by assigning 1 to primary input *N1,* we can extend the partial path through *N4*, in spite of the presence of an uncontrollable unit in the off path. Similarly, the growth of the partial path through *N6* is possible only when we can ensure a logic state 0 for the line *N17*. Using the extended algebra, the value that can *N17* would hold is *0/u*. (logical AND of 0/u and x/u in extended truth table). We do not have any control on the value *x/u*. So the direct implication should be extended backward using *U11* that produces either 0 or *u* (i.e. *0/u*). The source of *u* in *N14* (the output of *U11*) is the non-scan flop U50003. So we do not have any control over U50003. But the other input *N2* can be made 1, and thus assign 0 at *N17*. Hence, by assigning 1 to both *N1* and *N2* primary inputs, successful propagation of the transition through *U15* can be ensured. Partial paths through *N6* can be extended.

If we make a slight modification to the circuit by converting the *U13* AND gate into a NAND gate, we get a *1/u* value at the *N17* net. *U15* output becomes *0/u* and net *N5* would be *1/u*. It makes it impossible to produce 1 at *U15* and *0* at net *N5*. We cannot test the fault sites *N5* and the output node of gate U15. All paths through *N6* also go through *U15*. Hence, we cannot test faults at *N6*. Thus due to the inability to create/propagate transitions, we would not be able to test the paths passing through *U15*. Hence, we can trim off the associated fan-out logic. Figure 25 shows the trimming example that we have discussed so far.



N17 can never be made 0 for successful propagation through U15. Hence, this is leading to a untestable path. Trimming is done.

N13 can be made 0 for successful propagation through U12, by assigning N1 to 1

Figure 25. Logic trimming in ISCA89 S27 modified Circuit

4.4 Realization of extended algebra in *CodGen*

We have focused on extending the work of [22] leading towards higher path delay fault coverage in the presence of non-scan cells, embedded memories and black boxes. The steps of realizing the extended algebra in *CodGen* are given below.

1. Extend the existing traditional algebra to an 11-value algebra. The 11 values being considered are zero (stable), pne (stable), unknown, uncontrollable, rise, fall, zero/uncontrollable, one/uncontrollable, unknown/uncontrollable, rise/uncontrollable, and fall/uncontrollable.
2. Generate an extended truth table for each of the following logic cells: NOT, AND, NAND, OR, NOR, XOR, XNOR, PI, Buff, Mux, TSL, TSH, TSLI, TSHI, TIE1, and TIE0. Figure 19 shows the extended truth tables used in the work.
3. We have assumed "Fall" and "Rise" to be controlling values for respective gates. (along with 0 and 1). This is applicable for both the original and extended algebras. Previously there was no consideration of Rise and Fall as controlling values. The reasons for pruning a path is that either the side input is a controlling value or XXX (i,e. *u*) or XX (i,e. output when the one of the inputs is *u*). In the 11-valued algebra, an optimistic approach is taken by allowing XXX and XX (i.e. 1_XXX, 0_XXX, X_XXX, Rise_XXX, Fall_XXX).
4. Using a scan attribute construct, the percentages of non-scan and scan flip-flops have been tweaked to realistic values.
5. There are up to 5 - 10% non-scan flip-flops in some circuits. Results have been generated for all ISCA89 circuits assuming approximately 10% non-scan flops. In the presence of non-scan flip-flops better path delay fault coverages have been observed in comparison to the results available using the traditional algebra.

6. Results have been obtained from runs for different values of *K* (i.e. number of long paths) to show the sustained improvement. Results have been generated for both robust paths and non-robust paths.

4.5 The Extended Truth Tables

In this section, we show the individual truth tables used for different log entities. In Table 1 (a to k), we see tables for NOT, BUFF, AND, NAND, OR, NOR, XOR, XNOR, TSL, TSH, TSLI, TSHI. In these table, *v1* having a non-u value and *v2* having a *u* value has been assumed to be *u*, with the rationale that any transition to *u* is also uncontrollable *(u)*.

Table 1. Truth Tables of basic gates using 11-value algebra

One input gates:

(a) Truth tables for NOT, BUFF gates

BUFF	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u

NOT	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
	1	0	x	u	1/u	0/u	x/u	Fall	Rise	Fall/u	Rise/u

Two input gates:

(b) Truth table for AND gate

AND	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
x	0	x	x	0/u	0/u	x/u	x/u	x	0	x/u	0/u
u	0	u	0/u	u	0/u	u	0/u	u	0	u	0/u
0/u	0	0/u	0/u	0/u	0/u	0/u	0/u	0/u	0	0/u	0/u
1/u	0	1/u	x/u	u	0/u	1/u	x/u	Rise/u	0	Rise/u	Fall/u
x/u	0	x/u	x/u	0/u	0/u	x/u	x/u	x/u	0	x/u	x/u
Rise	0	Rise	x	u	0/u	Rise/u	x/u	Rise	0	Rise/u	0/u
Fall	0	Fall	0	0	0	0	0	0	Fall	0	0
Rise/u	0	Rise/u	x/u	u	0/u	Rise/u	x/u	Rise/u	0	Rise/u	0/u
Fall/u	0	Fall/u	0/u	0/u	0/u	Fall/u	0/u	0/u	0	0/u	0/u

(c) Truth table for NAND gate

NAND	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
0	1	1	1	1	1	1	1	1	1	1	1
1	1	0	x	u	1/u	0/u	x/u	Fall	Rise	Fall/u	Rise/u
x	1	x	x	1/u	1/u	x/u	x/u	x	1	x/u	1/u
u	1	u	1/u	u	1/u	u	1/u	u	1	u	1/u
0/u	1	1/u	1/u	1/u	1/u	1/u	1/u	1/u	1	1/u	1/u
1/u	1	0/u	x/u	u	1/u	0/u	x/u	Fall/u	1	Fall/u	Rise/u
x/u	1	x/u	x/u	1/u	1/u	x/u	x/u	x/u	1	x/u	x/u
Rise	1	Fall	x	u	1/u	Fall/u	x/u	Fall	1	Fall/u	1/u
Fall	1	Rise	1	1	1	0	1	0	Rise	1	1
Rise/u	1	Fall/u	x/u	u	1/u	Fall/u	x/u	Fall/u	1	Fall/u	1/u
Fall/u	1	Rise/u	1/u	1/u	1/u	Rise/u	1/u	1/u	1	1/u	1/u

(d) Truth table for OR gate

OR	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
0	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
1	1	1	1	1	1	1	1	1	1	1	1
x	x	1	x	1/u	x/u	1/u	x/u	1	x	1/u	x/u
u	u	1	1/u	u	u	1/u	1/u	1	u	1/u	u
0/u	0/u	1	x/u	u	0/u	1/u	x/u	1	0/u	1/u	Fall/u
1/u	1/u	1	1/u	1/u	1/u	1/u	1/u	1	1/u	1/u	1/u
x/u	x/u	1	x/u	1/u	x/u	1/u	x/u	1	x/u	1/u	x/u
Rise	Rise	1	1	1	1	1	1	Rise	1	1	1
Fall	Fall	1	x	u	0/u	1/u	x/u	1	Fall	1/u	Fall/u
Rise/u	Rise/u	1	1/u	1/u	1/u	1/u	1/u	1	1/u	1/u	1/u
Fall/u	Fall/u	1	x/u	u	Fall/u	1/u	x/u	1	Fall/u	1/u	Fall/u

(e) Truth table for NOR gate

NOR	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
0	1	0	x	u	1/u	0/u	x/u	Rise	Fall	Fall/u	Rise/u
1	0	0	0	0	0	0	0	0	0	0	0
x	x	0	x	0/u	x/u	0/u	x/u	0	x	0/u	x/u
u	u	0	0/u	u	u	0/u	0/u	0	u	0/u	u
0/u	1/u	0	x/u	u	1/u	0/u	x/u	0	1/u	0/u	Rise/u
1/u	0/u	0	0/u	0/u	0/u	0/u	0/u	0	0/u	0/u	0/u
x/u	x/u	0	x/u	0/u	x/u	0/u	x/u	0	x/u	0/u	x/u
Rise	Fall	0	0	0	0	0	0	Fall	0	0	0
Fall	Rise	0	x	u	1/u	0/u	x/u	0	Rise	0/u	Rise/u
Rise/u	Fall/u	0	0/u	0/u	0/u	0/u	0/u	0	0/u	0/u	0/u
Fall/u	Rise/u	0	x/u	u	Rise/u	0/u	x/u	0	Rise/u	0/u	Rise/u

(f) Truth table for XOR gate

XOR	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
0	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
1	1	0	x	u	1/u	0/u	x/u	Fall	Rise	Fall/u	Rise/u
x	x	x	x	u	x/u	x/u	x/u	x	x	x/u	x/u
u	u	u	u	u	u	u	u	u	u	u	u
0/u	0/u	1/u	x/u	u	0/u	1/u	x/u	Rise/u	Fall/u	Rise/u	Fall/u
1/u	1/u	0/u	x/u	u	1/u	0/u	x/u	Fall/u	Rise/u	Fall/u	Rise/u
x/u	x/u	x/u	x/u	u	x/u	x/u	x/u	x/u	x/u	x/u	x/u
Rise	Rise	Fall	x	u	Rise/u	0/u	x/u	0	1	0/u	1/u
Fall	Fall	Rise	x	u	Fall/u	1/u	x/u	1	0	1/u	0/u
Rise/u	Rise/u	Fall/u	x/u	u	Rise/u	Fall/u	x/u	0/u	1/u	0/u	1/u
Fall/u	Fall/u	Rise/u	x/u	u	Fall/u	Rise/u	x/u	1/u	0/u	1/u	0/u

(g) Truth table for XNOR gate

XNOR	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
0	1	0	x	u	1/u	0/u	x/u	Fall	Rise	Fall/u	Rise/u
1	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
x	x	x	x	u	x/u	x/u	x/u	x	x	x/u	x/u
u	u	u	u	u	u	u	u	u	u	u	u
0/u	1/u	0/u	x/u	u	1/u	0/u	x/u	Fall/u	Rise/u	Fall/u	Rise/u
1/u	0/u	1/u	x/u	u	0/u	1/u	x/u	Rise/u	Fall/u	Rise/u	Fall/u
x/u	x/u	x/u	x/u	u	x/u	x/u	x/u	x/u	x/u	x/u	x/u
Rise	Fall	Rise	x	u	Fall/u	1/u	x/u	1	0	1/u	0/u
Fall	Rise	Fall	x	u	Rise/u	0/u	x/u	0	1	0/u	1/u
Rise/u	Fall/u	Rise/u	x/u	u	Fall/u	Rise/u	x/u	1/u	0/u	1/u	0/u
Fall/u	Rise/u	Fall/u	x/u	u	Rise/u	Fall/u	x/u	0/u	1/u	0/u	1/u

(h) Truth table for TSL gate

TSL	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
0	0	u	0/u	u	0/u	u	0/u	u	0	u	0/u
1	1	u	1/u	u	1/u	u	1/u	u	1	u	1/u
x	x	u	x/u	u	x/u	u	x/u	u	x	u	x/u
u	u	u	u	u	u	u	u	u	u	u	u
0/u	0/u	u	0/u	u	0/u	u	0/u	u	0/u	u	0/u
1/u	1/u	u	1/u	u	1/u	u	1/u	u	1/u	u	1/u
x/u	x/u	u	x/u	u	x/u	u	x/u	u	x/u	u	x/u
Rise	Rise	u	Rise/u	u	Rise/u	u	Rise/u	u	1	u	1/u
Fall	Fall	u	Fall/u	u	Fall/u	u	Fall/u	u	0	u	0/u
Rise/u	Rise/u	u	Rise/u	u	Rise/u	u	Rise/u	u	1/u	u	1/u
Fall/u	Fall/u	u	Fall/u	u	Fall/u	u	Fall/u	u	0/u	u	0/u

(i) Truth table for TSH gate

TSH	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
0	u	0	0/u	u	u	0/u	0/u	0	u	0/u	u
1	u	1	1/u	u	u	1/u	1/u	1	u	1/u	u
x	u	x	x/u	u	u	x/u	x/u	x	u	x/u	u
u	u	u	u	u	u	u	u	u	u	u	u
0/u	u	0/u	0/u	u	u	0/u	0/u	0/u	u	0/u	u
1/u	u	1/u	1/u	u	u	1/u	1/u	1/u	u	1/u	u
x/u	u	x/u	x/u	u	u	x/u	x/u	x/u	u	x/u	u
Rise	u	Rise	Rise/u	u	u	Rise/u	Rise/u	1	u	1/u	u
Fall	u	Fall	Fall/u	u	u	Fall/u	Fall/u	0	u	0/u	u
Rise/u	u	Rise/u	Rise/u	u	u	Rise/u	Rise/u	1/u	u	1/u	u
Fall/u	u	Fall/u	Fall/u	u	u	Fall/u	Fall/u	0/u	u	0/u	u

(j) Truth table for TSLI gate

TSLI	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
0	1	u	1/u	u	1/u	u	1/u	u	1	u	1/u
1	0	u	0/u	u	0/u	u	0/u	u	0	u	0/u
x	x	u	x/u	u	x/u	u	x/u	u	x	u	x/u
u	u	u	u	u	u	u	u	u	u	u	u
0/u	1/u	u	1/u	u	1/u	u	1/u	u	1/u	u	1/u
1/u	0/u	u	0/u	u	0/u	u	0/u	u	0/u	u	0/u
x/u	x/u	u	x/u	u	x/u	u	x/u	u	x/u	u	x/u
Rise	Fall	u	Fall/u	u	Fall/u	u	Fall/u	u	0	u	0/u
Fall	Rise	u	Rise/u	u	Rise/u	u	Rise/u	u	1	u	1/u
Rise/u	Fall/u	u	Fall/u	u	Fall/u	u	Fall/u	u	0/u	u	0/u
Fall/u	Rise/u	u	Rise/u	u	Rise/u	u	Rise/u	u	1/u	u	1/u

(k) Truth table for TSHI gate

TSHI	0	1	x	u	0/u	1/u	x/u	Rise	Fall	Rise/u	Fall/u
0	u	1	1/u	u	u	1/u	1/u	1	u	1/u	u
1	u	0	0/u	u	u	0/u	0/u	0	u	0/u	u
x	u	x	x/u	u	u	x/u	x/u	x	u	x/u	u
u	u	u	u	u	u	u	u	u	u	u	u
0/u	u	1/u	1/u	u	u	1/u	1/u	1/u	u	1/u	u
1/u	u	0/u	0/u	u	u	0/u	0/u	0/u	u	0/u	u
x/u	u	x/u	x/u	u	u	x/u	x/u	x/u	u	x/u	u
Rise	u	Fall	Fall/u	u	u	Fall/u	Fall/u	0	u	0/u	u
Fall	u	Rise	Rise/u	u	u	Rise/u	Rise/u	1	u	1/u	u
Rise/u	u	Fall/u	Fall/u	u	u	Fall/u	Fall/u	0/u	u	0/u	u
Fall/u	u	Rise/u	Rise/u	u	u	Rise/u	Rise/u	1/u	u	1/u	u

5. EXPERIMENTAL RESULTS

5.1 Robust path tests

Experimental results have been obtained for both Robust and Non-Robust path delay tests.

5.1.1 Paths discovered with different K values and two algebras

We have validated the expected path recovery advantage by running experiments on *CodGen* to discover paths from the ISCA89 benchmark circuits, using launch-on-capture test patterns with no preamble cycles. Table 2 shows the number of robust paths discovered for different ISCA89 circuits using both the traditional algebra and the 11-value algebra. The path count has been reported for different *K* values. Figure 26 shows the additional extra robust paths recovered for different ISCA89 benchmark circuits. The

Table 2. Robust paths discovered with different K values using both algebras

Benchmark circuits	Gates	Total Flip-flops	Non-scan ones (around 10%)	Robust Paths being discovered									
				Using Traditional Algebra					Using 11 value algebra				
				K=1	K=2	K=3	K=4	K=5	K=1	K=2	K=3	K=4	K=5
s5378	2993	179	20	93	130	168	201	270	510	928	1279	1605	1865
s9234	5844	211	20	46	78	78	78	78	48	82	84	86	88
s13207	8651	639	60	277	326	370	413	430	372	424	472	515	543
s15850	10833	534	50	71	74	74	74	74	78	81	81	81	81
s38417	22142	2426	240	721	743	747	751	755	725	751	757	762	766
s38584	23843	2636	260	312	314	314	314	314	313	316	316	316	316

type of circuit controls the path distribution and the number of paths being blocked due to uncontrollable non-scan (NS) flip-flops. The common trend that is evident here is, "The longer paths we want to cover, the greater the robust path recovery advantage we achieve using our algebra."

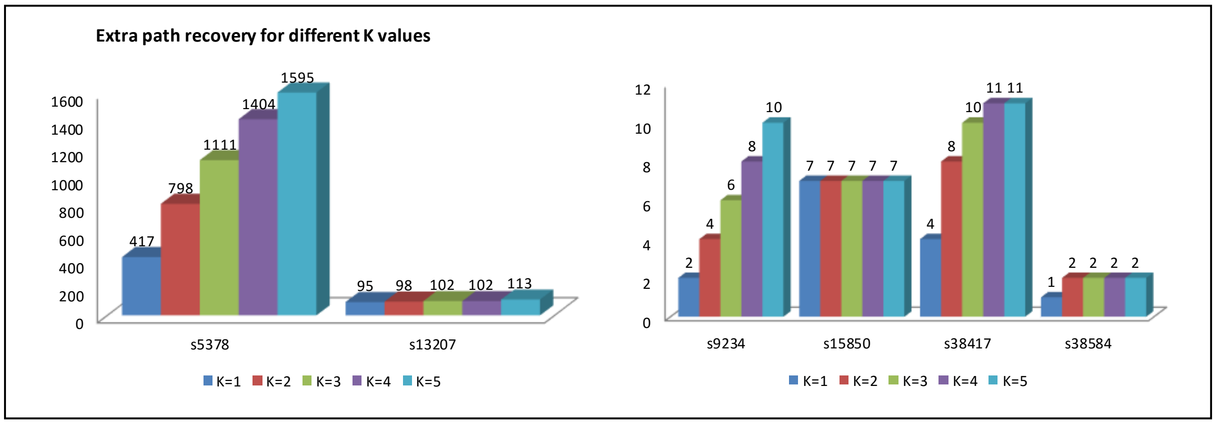


Figure 26. Improvements in Robust path recovery for different K values

5.1.2 Faults detected with different K values and two algebras

Table 3 shows the number of transition faults detected using robust tests in ISCA89 benchmark circuits using both the traditional algebra and the 11-value algebra. As we explore more number of paths, the more number of transition faults get exposed to our test environment. For all the circuits we have observed an increase in the number of transition fault detection metric. This sub-section reports the advantages achieved in terms of transition fault count. The numbers of faults detected have been reported for different *K* values. Figure 27 shows the number of extra faults recovered for different ISCA89 benchmark circuits. Here we can see that the more paths recovered, the more

Table 3. Faults detected in Robust path test with different K values using both types of algebra

Benchmark circuits	Faults Detected									
	Using Traditional Algebra					Using 11 value algebra				
	K=1	K=2	K=3	K=4	K=5	K=1	K=2	K=3	K=4	K=5
s5378	339	407	458	476	502	3922	4238	4607	4741	4968
s9234	492	652	652	652	652	513	682	692	702	712
s13207	1482	1723	1947	2165	2253	1964	2228	2466	2684	2825
s15850	458	497	497	497	497	511	550	550	550	550
s38417	2808	2971	2999	3027	3055	2845	3034	3074	3108	3136
s38584	632	636	636	636	636	657	662	662	662	662

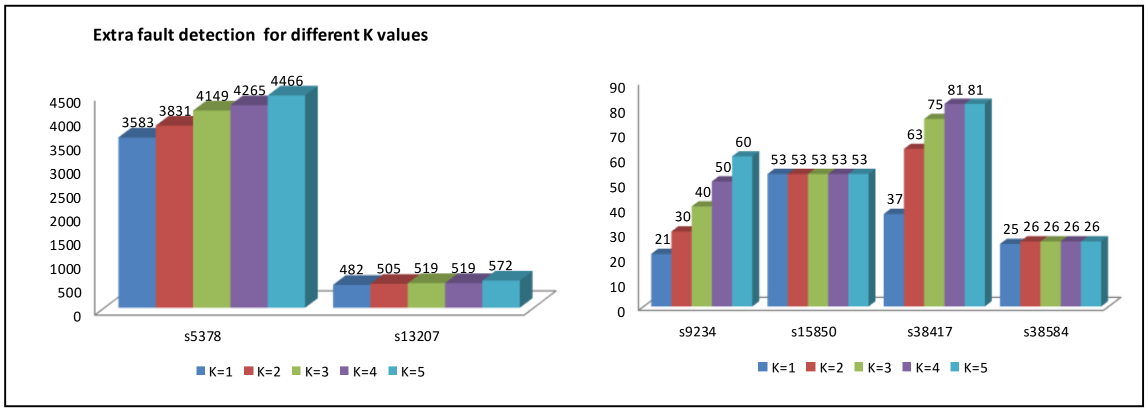


Figure 27. Improvements in Transition Fault recovery (Robust Test)

fault sites that are tested. Thus, we see that the 11-value algebra results in enhancements in fault coverage. Whichever algebra is used, increasing the value of *K* always results in more testable faults than with lower values of *K*. The increase stops when all testable fault sites have been detected. For s15850, we see that saturation has been reached for all values of *K* while using both the algebras. For s38584, fault detection reaches saturation at *K*=2. With lower *K*, many times back-tracking limits and max iteration counts prevents exploring more testable fault sites. Otherwise, the fault coverage should be the same for all *K* values. With increased *K*, more faults are detected fortuitously, increasing the fault coverage.

5.1.3 CPU runtime comparison with different K values and two algebras

Table 4 reports the CPU time taken by each set of experiments. From the numbers we can easily see that in most of the cases the 11-value algebra approach has taken almost the same amount of time as the traditional algebra but has produced per path.

Figure 28(a) to Figure 28(f) show the per-path CPU time for different circuits. From the figures, we can see that in most cases CPU time per path is reduced using the 11-value algebra. We can infer from these results that in comparison to the traditional algebra, 11-value algebra discovers an average path faster. s38584 is the only circuit for which the traditional algebra consistently reports lower average CPU time, but only slightly lower.

Table 4. CPU time taken for Robust path test with different K values using both types of algebra

Benchmark circuits	CPU Time (Sec)									
	Using Traditional Algebra					Using 11 value algebra				
	K=1	K=2	K=3	K=4	K=5	K=1	K=2	K=3	K=4	K=5
s5378	8	17	23	36	51	40	68	93	112	130
s9234	9	11	12	12	12	9	11	12	12	12
s13207	49	57	61	70	71	64	73	82	88	92
s15850	22	23	21	23	23	24	24	24	24	25
s38417	278	285	288	290	290	281	287	284	294	294
s38584	113	113	106	115	115	120	121	119	118	121

s5378	K=1	K=2	K=3	K=4	K=5
Traditional	0.086022	0.130769	0.136905	0.179104	0.188889
11-Value	0.078431	0.073276	0.072713	0.069782	0.069705

Normalized CPU time (Sec) per robust Path

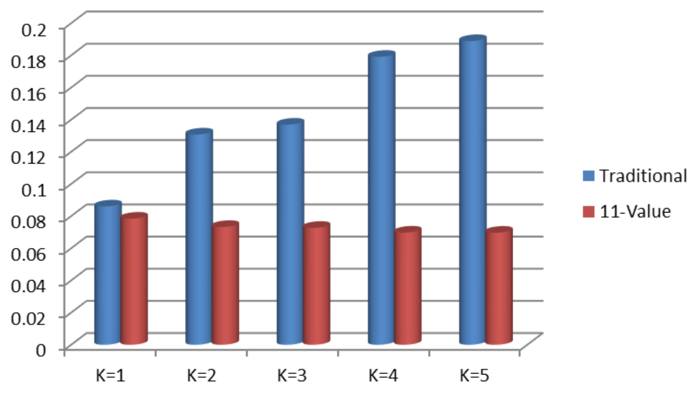


Figure 28. (a) s5378 : Relative comparison of CPU time normalized over Robust path count

s9234	K=1	K=2	K=3	K=4	K=5
Traditional	0.195652	0.141026	0.153846	0.153846	0.153846
11-Value	0.1875	0.134146	0.142857	0.139535	0.136364
Normalized CPU time (Sec) per robust Path					

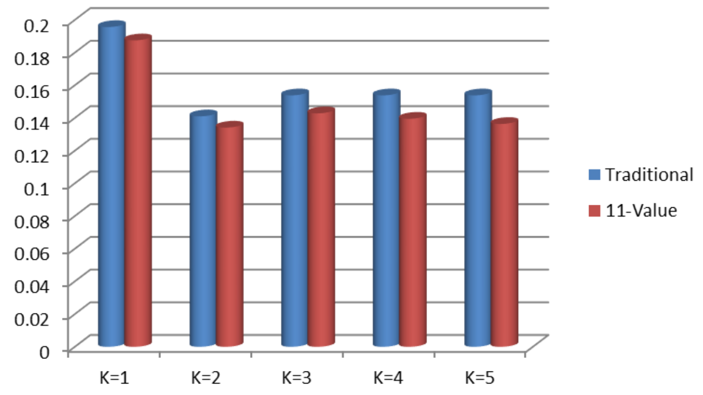


Figure 28. (b) s9234 : Relative comparison of CPU time normalized over Robust path count

s13207	K=1	K=2	K=3	K=4	K=5
Traditional	0.176895	0.174847	0.164865	0.169492	0.165116
11-Value	0.172043	0.17217	0.173729	0.170874	0.169429
Normalized CPU time (Sec) per robust Path					

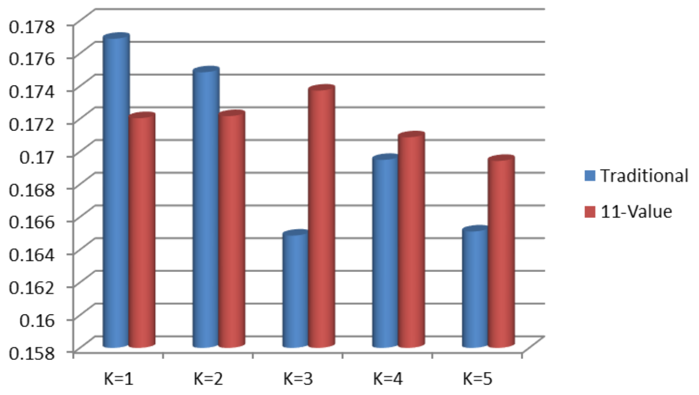


Figure 28. (c) s13207:Relative comparison of CPU time normalized over Robust path count

s15850	K=1	K=2	K=3	K=4	K=5
Traditional	0.309859	0.310811	0.283784	0.310811	0.310811
11-Value	0.307692	0.296296	0.296296	0.296296	0.308642

Normalized CPU time (Sec) per robust Path

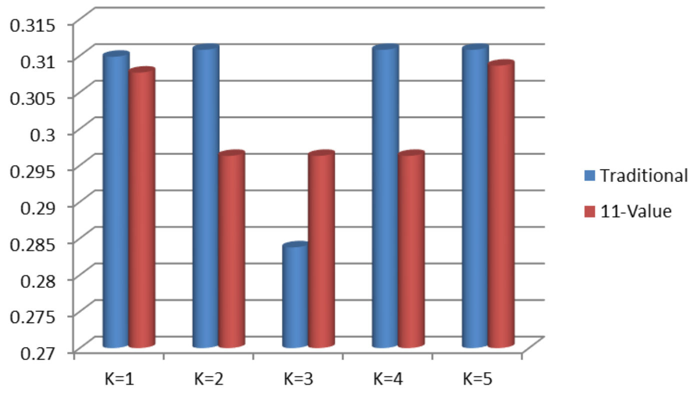


Figure 28. (d) s15850:Relative comparison of CPU time normalized over Robust path count

s38417	K=1	K=2	K=3	K=4	K=5
Traditional	0.385576	0.38358	0.385542	0.386152	0.384106
11-Value	0.387586	0.382157	0.375165	0.385827	0.383812

Normalized CPU time (Sec) per robust Path

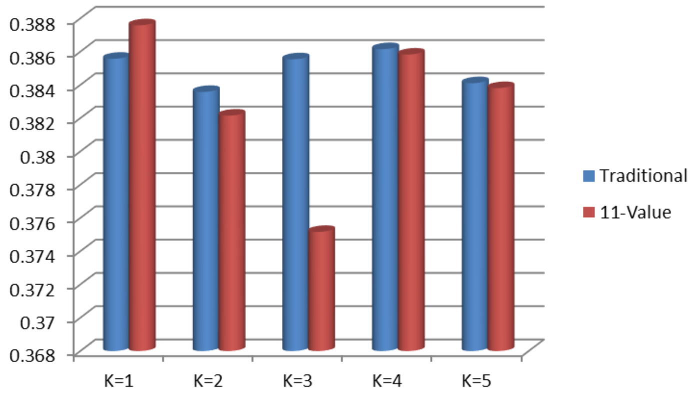


Figure 28. (e) s38417:Relative comparison of CPU time normalized over Robust path count

s38584	K=1	K=2	K=3	K=4	K=5
Traditional	0.362179	0.359873	0.33758	0.366242	0.366242
11-Value	0.383387	0.382911	0.376582	0.373418	0.382911

Normalized CPU time (Sec) per robust Path

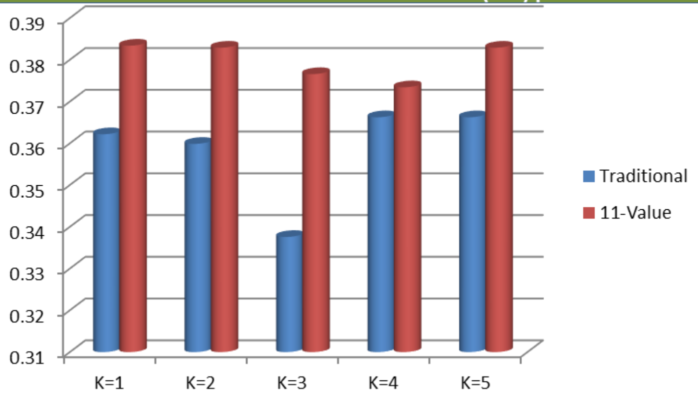


Figure 28. (f) s38584:Relative comparison of CPU time normalized over Robust path count

Any algebra with more logic values has inherently more computational complexity than an algebra with fewer values. Here, for s38584 the higher value complexity dominates over the early conflict resolution (untestable path detection).

5.2 Non-Robust path tests

Almost the similar trends have been observed for Non-Robust tests also. The number we have obtained are presented in the following sections.

5.2.1 Paths discovered with different K values and two algebras

Table 5 shows the non-robust path counts of the ISCA89 circuits for different *K* values. The results have trends similar to the robust test. For the same circuit, the more long non-robust paths we are interested in, the more paths are discovered. This increase is common for both the algebras. However, the 11-value algebra with larger values of *K*

Table 5. Non-Robust paths discovered with different K values (2 types of algebra)

Benchmark circuits	Gates	Total Flip-flops	Non-scan ones (around 10%)	Non-Robust Paths being discovered									
				Using Traditional Algebra					Using 11 value algebra				
				K=1	K=2	K=3	K=4	K=5	K=1	K=2	K=3	K=4	K=5
s5378	2993	179	20	93	138	177	210	285	542	980	1474	1879	2329
s9234	5844	211	20	46	78	78	78	78	52	86	88	90	92
s13207	8651	639	60	281	332	378	421	438	414	486	538	583	621
s15850	10833	534	50	80	80	80	80	80	90	90	90	90	90
s38417	22142	2426	240	744	752	756	760	764	751	761	767	772	777
s38584	23843	2636	260	312	314	314	314	314	316	316	316	316	316

has a larger advantage. Figure 29 shows the extra paths recovered from the ISCAS89 circuits while using the 11-value algebra in *CodGen*. For s5378 and s13207 we observe the significant advantages of path recovery. For other circuits it is less. For s38584 and s15850, the path recovery advantage is the same for all *K* values. These variations are due to differences in the nature of the circuits. The circuits with more recovery are the ones most impacted negatively by the presence of non-scan flip-flops.

5.2.2 Faults detected with different K values and two algebras

Just like the path counts, Table 6 shows the number of transition faults detected

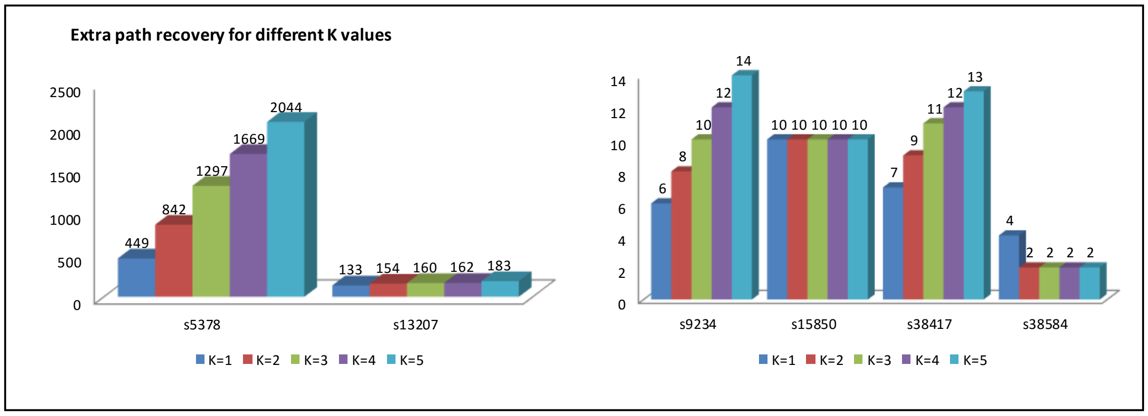


Figure 29. Improvements in Non-Robust path recovery

for different values of *K* using both the algebras. As expected, we see the fault coverage getting better when the 11-vaue algebra is used. Increasing values of *K* finally result in

Table 6. Faults detected in Non-Robust path test with different K values using both types of algebra

Benchmark circuits	Faults Detected									
	Using Traditional Algebra					Using 11 value algebra				
	K=1	K=2	K=3	K=4	K=5	K=1	K=2	K=3	K=4	K=5
s5378	339	360	372	383	396	3052	3230	3491	3565	3679
s9234	492	652	652	652	652	513	681	683	685	687
s13207	1511	1757	1986	2199	2282	1947	2227	2471	2658	2830
s15850	563	563	563	563	563	619	619	619	619	619
s38417	2991	3042	3070	3098	3126	3035	3086	3114	3142	3170
s38584	632	636	636	636	636	662	662	662	662	662

increasing path count and higher fault coverage. We have deduced the extra faults detectable each time using the 11-value algebra in comparison to that of the traditional algebra (Figure 30). The advantage is greatest for s5378 and s13207. For the rest of the circuits there is a more modest advantage.

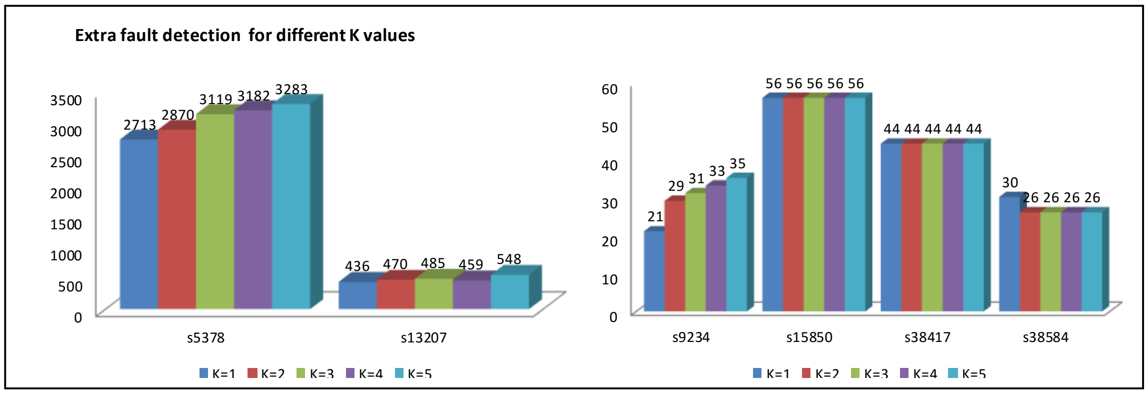


Figure 30. Improvements in Transition Fault recovery in Non-Robust Path test

5.2.3 CPU runtime comparison with different K values and two algebras

Table 7 shows the CPU time taken by all the non-robust path recovery experiments. The total CPU time is almost the same for both algebras for each circuit. However, since the 11-value algebra recovers more paths, its per-path CPU time is lower in many cases, as shown in Figure 31(a) to Figure 31(f). Only s38417 and s38584 have lower CPU time using the traditional algebra.

Table 7. CPU time taken for Non-Robust path test with different K values using both types of algebra

Benchmark circuits	CPU Time (Sec)									
	Using Traditional Algebra					Using 11 value algebra				
	K=1	K=2	K=3	K=4	K=5	K=1	K=2	K=3	K=4	K=5
s5378	9	18	25	38	52	39	65	98	123	151
s9234	9	11	12	11	12	9	12	13	14	13
s13207	50	57	66	73	75	71	82	91	98	106
s15850	23	23	23	25	24	27	26	26	26	26
s38417	287	287	291	293	300	293	293	294	303	310
s38584	114	114	116	114	112	119	118	119	125	123

s5378	K=1	K=2	K=3	K=4	K=5
Traditional	0.096774	0.130435	0.141243	0.180952	0.182456
11-Value	0.071956	0.066327	0.066486	0.06546	0.064835

Normalized CPU time (Sec) per Non-robust Path

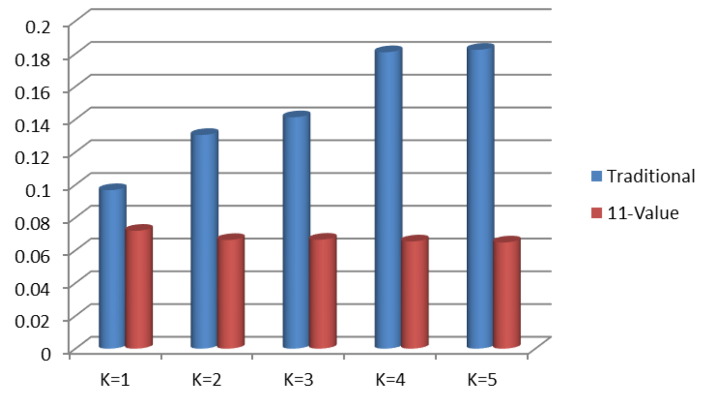


Figure 31. (a) s5378

Relative comparison of CPU time normalized over Non-Robust path count

s9234	K=1	K=2	K=3	K=4	K=5
Traditional	0.195652	0.141026	0.153846	0.141026	0.153846
11-Value	0.173077	0.139535	0.147727	0.155556	0.141304

Normalized CPU time (Sec) per Non-robust Path

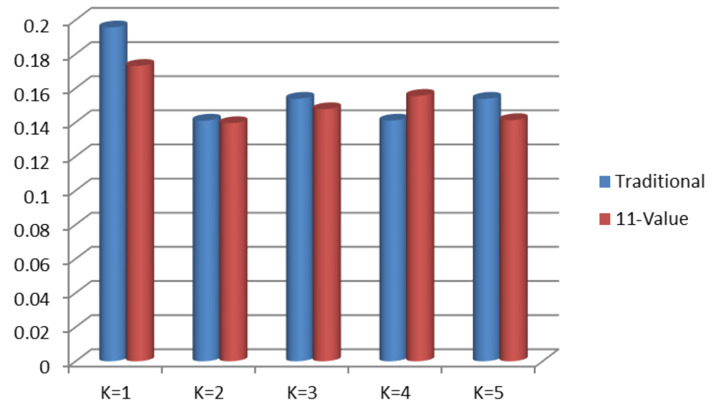


Figure 31. (b) s9234

Relative comparison of CPU time normalized over Non-Robust path count

s13207	K=1	K=2	K=3	K=4	K=5
Traditional	0.177936	0.171687	0.174603	0.173397	0.171233
11-Value	0.171498	0.168724	0.169145	0.168096	0.170692

Normalized CPU time (Sec) per Non-robust Path

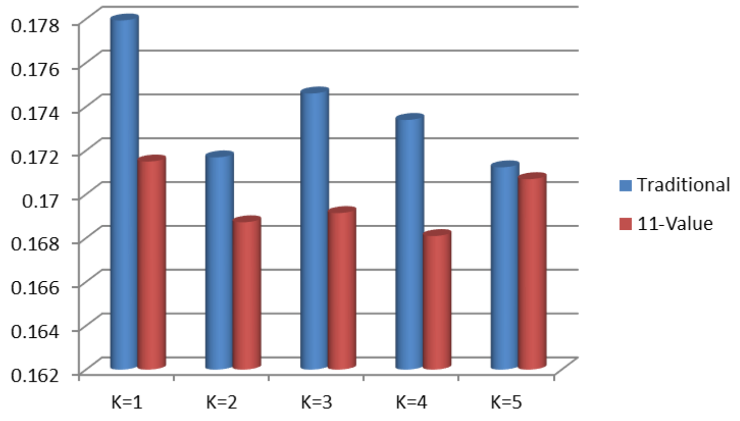


Figure 31. (c) s13207

Relative comparison of CPU time normalized over Non-Robust path count

s15850	K=1	K=2	K=3	K=4	K=5
Traditional	0.2875	0.2875	0.2875	0.3125	0.3
11-Value	0.3	0.288889	0.288889	0.288889	0.288889
Normalized CPU time (Sec) per Non-robust Path					

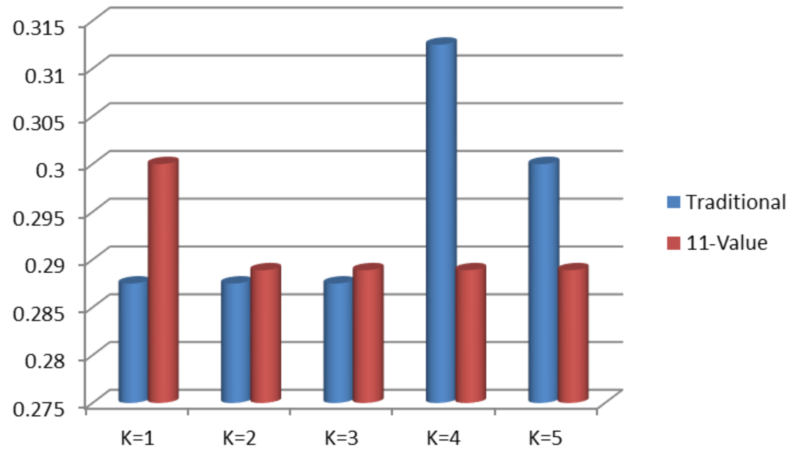


Figure 31. (d) s15850

Relative comparison of CPU time normalized over Non-Robust path count

s38417	K=1	K=2	K=3	K=4	K=5
Traditional	0.385753	0.381649	0.384921	0.385526	0.39267
11-Value	0.390146	0.38502	0.383312	0.392487	0.39897
Normalized CPU time (Sec) per Non-robust Path					

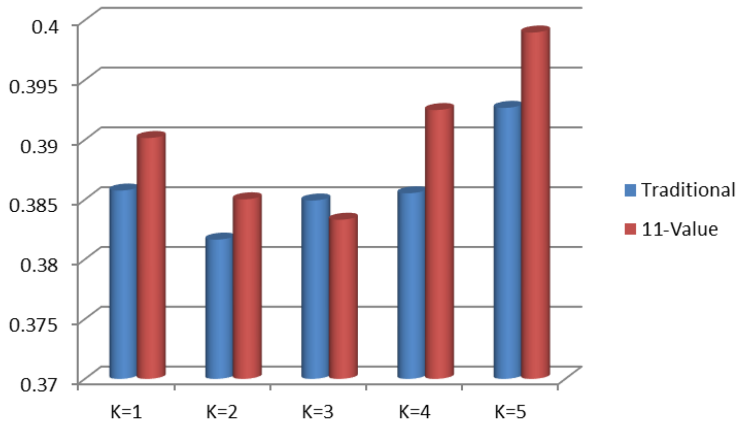


Figure 31. (e) s38417

Relative comparison of CPU time normalized over Non-Robust path count

s38584	K=1	K=2	K=3	K=4	K=5
Traditional	0.362179	0.359873	0.33758	0.366242	0.366242
11-Value	0.379747	0.382911	0.376582	0.373418	0.382911

Normalized CPU time (Sec) per Non-robust Path

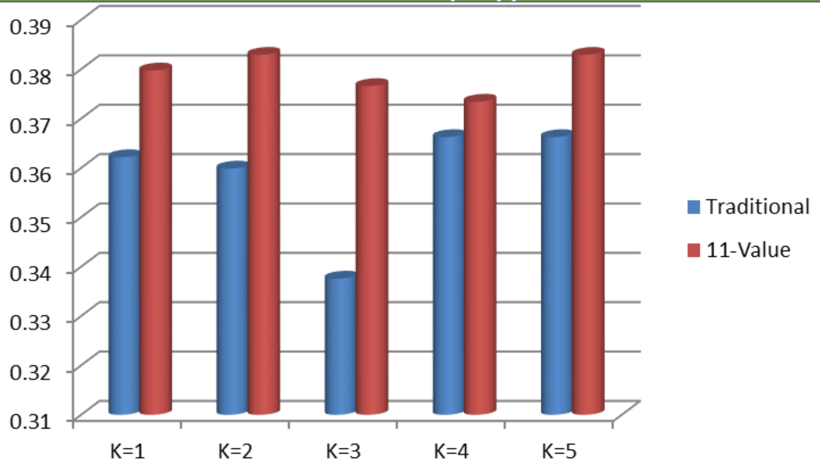


Figure 31. (f) s38584

Relative comparison of CPU time normalized over Non-Robust path count

Table 8. Longest Testable Path Lengths using both algebras

Benchmark circuits	Longest Testable Robust Path Length	
	Using Traditional Algebra	Using 11 value algebra
s5378	4	19
s9234	13	13
s13207	15	24
s15850	13	13
s38417	9	9
s38584	5	10

Benchmark circuits	Longest Testable Non-Robust Path Length	
	Using Traditional Algebra	Using 11 value algebra
s5378	4	23
s9234	13	13
s13207	15	24
s15850	13	13
s38417	9	9
s38584	5	10

5.3 Comparison of longest testable path using both the algebras

Table 8. shows the longest testable path lengths achieved for different circuits using both algebras. As we can see, in many cases, the 11-value algebra finds considerably longer paths in some circuits. This is critical for timing accuracy.

6. CONCLUSIONS AND FUTURE WORK

In this work, improved path and fault coverage has been achieved using the extended 11-value algebra. In many cases, improvements have been observed in terms of per-path CPU time. We expect the gain to increase with a higher percentage of non-scan or uncontrollable flip-flops.

As future work, the impact of the extended value algebra will be evaluated on industrial circuits. In our experiments, the flip-flops have been converted to non-scan arbitrarily. The actual impact on path recovery will be more visible with realistic selection of non-scan flip-flops. Circuits with embedded memories are also ideal candidates to validate the advantage of 11-value algebra.

The current implementation does not take advantage of the pseudo functional test option. In this case, the non-scan flip-flops may be set during the preamble cycles. Treating these flip-flops as uncontrollable for every cycle is pessimistic, since they are only uncontrollable for the initial scan-in phase, and then enter functional (transparent) mode. We will evaluate cases where the cells are only uncontrollable in the initial input. This requires extending the SAT engine to support uncontrollable input values.

REFERENCES

[1] R. Rajsuman. 2000. “*System-On-A-Chip: Design and Test* “, 1st ed. Artech House, Inc., Norwood, MA, USA.

[2] M. Tehranipoor, K. Peng, and K. Chakrabarty. 2014. “*Test and Diagnosis for Small-Delay Defects”*, Springer Publishing Company, Incorporated.

[3] I. Pomeranz, S. M. Reddy, *"Transition Path Delay Faults: A New Path Delay Fault Model for Small and Large Delay Defects"* in Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.16, no.1, pp.98-107, Jan. 2008

[4] G. L. Smith, *“Model for Delay Faults Based Upon Paths”* in IEEE International Test Conference, Oct. 1985, pp. 342-349.

[5] Laung-Terng Wang, Charles Stroud, Nur Touba *"System-on-Chip Test Architectures, 1st Edition Nanometer Design for Testability"*, Elsevier, 2007

[6] M. Bushnell and V. D. Agrawal. 2013. “*Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits”*, Springer Publishing Company, Incorporated.

[7] W. N. Li, S. M. Reddy, S. K. Sahni, *“On Path Selection in Combinational Logic Circuits”* in IEEE Trans. On Computer-Aided Design, vol. 8, no. 1, Jan. 1989, pp.56-63.

[8] A. K. Majhi, V. D. Agrawal, J. Jacob, L. M. Patnaik, *“Line Coverage of Path Delay Faults”* in IEEE Trans. on VLSI Systems, vol. 8, no. 5, Oct. 2000, pp. 610-613.

[9] A. Murakami , S. Kajihara, T. Sasao, I. Pomeranz, S.M. Reddy, *"Selection of Potentially Testable Path Delay Faults for Test Generation"* in IEEE International Test Conference, 2000, pp. 376-384.

[10] Y. Shao, S.M. Reddy, I. Pomeranz, S. Kajihara, *"On Selecting Testable Paths in Scan Designs"* in IEEE European Test Workshop, 2002, pp. 53-58. 52

[11] K. Fuchs, F. Fink, M. H. Schulz, *"DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults"* in IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 1991, vol.10, no.10, pp.1323-1335.

[11] M. Sharma, J. H. Patel, *"Finding a Small Set of Longest Testable Paths that Cover Every Gate" in* IEEE International Test Conference, 2002, pp. 974-982.

[12] W. Qiu, D. M. H. Walker, *"An Efficient Algorithm for Finding the K Longest Testable Paths Through Each Gate in a Combinational Circuit"* in IEEE International Test Conference, Sept. 2003, pp. 592-601.

[13] N. Drego, A. Chandrakasan; D. Boning, *"An All-Digital, Highly Scalable Architecture for Measurement of Spatial Variation in Digital Circuits"* in Solid-State Circuits Conference, 2008. A-SSCC '08. IEEE Asian, pp.393-396, 3-5 Nov. 2008

[14] J. Liou, L.-C. Wang, K.-T. Cheng, *"On Theoretical and Practical Considerations of Path Selection for Delay Fault Testing"* in Proc. IEEE/ACM International Conference on Computer Aided Design, 2002, pp. 94-100.

[15] C. Lin, S. Reddy, *"On Delay Fault Testing in Logic Circuits"* in IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol.6, no.5, Sept. 1987, pp. 694-703.

[16] P. McGeer, R. Brayton, *"Efficient Algorithms for Computing the Longest Viable Path in a Combinational Network"* in Proc. ACM/IEEE Design Automation Conference, June 1989, pp. 561-567. 53

[17] J. Benkoski, E. Meersch, L. Claesen, H. Man, *"Timing Verification using Statically Sensitizable Paths"* in IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol.9, no.10, Oct. 1990, pp.1073-1084.

[18] H. Chang, J. Abraham, *"VIPER: An Efficient Vigorously Sensitizable Path Extractor"* in Proc. ACM/IEEE Design Automation Conference, June 1993, pp.112-117.

[19] J. Liou, A. Krstic, L.-C. Wang, K.-T. Cheng, *"False-path-aware Statistical Timing Analysis and Efficient Path Selection for Delay Testing and Timing Validation"* in Proc. ACM/IEEE Design Automation Conference, 2002, pp. 566-569.

[20] H. Furukawa; X. Wen, K. Miyase, Y. Yamato, S. Kajihara, P. Girard, L. T. Wang, M. Tehranipoor, *"CTX: A Clock-Gating-Based Test Relaxation and X-Filling Scheme for Reducing Yield Loss Risk in At-Speed Scan Testing"* in Asian Test Symposium, 2008. ATS '08. 17th vol., no., pp.397-402, 24-27 Nov. 2008

[21] Y. Bonhomme, P. Girard, L. Guiller, C. Landrault, S. Pravossoudovitch, *"A Gated Clock Scheme for Low Power Scan Testing of Logic ICs or Embedded Cores"* in Test Symposium, 2001. Proceedings.10th Asian, pp.253-258,2001.

[22] W. Qiu and D. Walker, *“An Efficient Algorithm for Finding the K Longest Testable Paths Through Each Gate in a Combinational Circuit*”, in Proc. IEEE Int. Test Conf., pp. 592–601, September 2003.

[23] W. Qiu, J. Wang, D. Walker, D. Reddy, X. Lu, Z. Li, W. Shi, and H. Balachandran, “*K Longest Paths Per Gate (KLPG) Test Generation for Scan-based Sequential Circuits”*, in Proc. IEEE Int. Test Conf., pp. 223–231, October 2004.

[24] J. Zhao, E. Rudnick, and J. Patel, “*Static Logic Implication with Application to Redundancy Identification”*, in *Proc. IEEE VLSI Test Symp.*, pp. 288–293, April 1997.

[25] L.-T. Wang, C.-W. Wu, and X. Wen, editors, “*VLSI Test Principles and Architectures: Design for Testability”*, Morgan Kaufmann, San Francisco, 2006.

[26] X. Liu and M. Hsiao, *“Constrained ATPG for broadside transition testing”*, in Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems, pp. 175–182, November 2003.

[27] P. Pant, J. Zelman, *“Understanding Power Supply Droop During At-Speed Scan Testing”* in IEEE VLSI Test Symposium, May 2009, pp.227-232.

[28] W. Qiu, J. Wang, D. M. H. Walker, D. Reddy, X. Lu, Z. Li, W. Shi and H. Balachandran, *“K Longest Paths Per Gate (KLPG) Test Generation for Scan- Based Sequential Circuits”* in IEEE International Test Conference, Oct. 2004, pp. 223-231.

[29] Y. C. Lin, F. Lu, K. Yang, K. T. Cheng, *"Constraint Extraction for Pseudo-functional Scan-based Delay Testing"* in Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific, vol.1, pp.166-171 Vol. 1, 18-21 Jan. 2005

[30] F. Zhaohui, Y. Yinlei, S. Malik, *"Considering Circuit Observability Don't Care-s in CNF Satisfiability"* in Design, Automation and Test in Europe, 2005. Proceedings, vol., no., pp.1108-1113 Vol. 2, 7-11 March 2005

[31] K. Bian, D. M. H. Walker, S. Khatri, S. Lahiri, *“Mixed Structural-Functional Path Delay Test Generation and Compaction”* IEEE International Symposium Defect and Fault Tolerance in VLSI and Nanotechnology Systems, Oct. 2013, pp. 7-12.

[32] N. Eén, N. Sörensson, *“The MiniSat Page, Introduction”.* Retrieved from minisat.se on March 2015.

[33] S. Chakraborty, D. M. H. Walker, *"At-Speed Path Delay Test"* in Test Workshop (NATW), 2015 IEEE 24th North Atlantic, vol., no., pp.39-42, 11-13 May 2015. 54

[34] L.-T. Wang and Y.-W. Chang, “*Electronic Design Automation: Synthesis, Verification, and Test (Systems on Silicon)”*. Morgan Kaufmann, Burlington, MA, 2009.

[35] M. Bushnell and V. D. Agrawal. 2013. “*Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits”*, Springer Publishing Company, Incorporated.

[36] P. T. Gonciari, B. M. Al-Hashimi, N. Nicolici, *"Improving Compression Ratio, Area Overhead, and Test Application Time for System-on-Chip Test Data Compression/Decompression"*, in Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings Year: 2002.

[37] T. Larrabee, *“Test Pattern Generation Using Boolean Satisfiability”*, in *IEEE Trans. Computer*-Aided Design, vol. 11, no. 1, pp. 4-15, Jan. 1992.